A Heuristic Algorithm to Design 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—An AND-OR-EXOR network, where the output EXOR gate has only two inputs, is one of the simplest three-level architecture. This network realizes an EXOR of two sum-of-products expressions (EX-SOP). In this paper, we show an algorithm to simplify EX-SOPs for multiple-output functions. Our objective is to minimize the number of distinct products in the sum-of-products expressions of EX-SOPs. The algorithm uses a divide-and-conquer strategy. It recursively applies the Shannon decomposition on a function with more than five variables. The algorithm obtains EX-SOPs for the five-variable functions by using an exact minimization program, then combines those EX-SOPs to generate EX-SOPs for the functions with more variables. We present experimental results for a set of benchmark functions, and show that EX-SOPs require many fewer products and literals than sum-of-products expressions. This is evidence that AND-OR-EXOR is a powerful architecture to realize many practical logic functions.

Index Terms—Three-level network, AND-EXOR, logic minimization, decomposition, programmable logic device.

I. Introduction

Many logic design systems use AND and OR gates as their basic elements. Such systems are suitable for control circuits and often results in optimal designs. However, they produce poorly optimized networks for arithmetic and telecommunication circuits [20, 21, 27, 29]. For these circuits, two-level AND-EXOR networks require fewer gates and interconnections than two-level AND-OR networks [19–21, 23, 27]. This suggests that arithmetic and telecommunication circuits are well-suited for EXOR (exclusive-OR) based design. However, two-level AND-EXOR networks require EXOR gates with unlimited fan-in. In most technologies, EXOR gates with many inputs are slow and expensive, since they are often implemented as a cascade or tree of two-input EXOR gates [32].

In this paper, we consider an AND-OR-EXOR three-level network, where a single two-input EXOR gate is used at each output. Each output of the network realizes an EXOR of two sum-of-products expressions (EX-SOP). Our objective is to reduce the total number of distinct products in the sum-of-products expressions (SOPs) of EX-SOPs. Simplification of an EX-SOP for a function $f(X)$ is equivalent to finding a decomposition that has the form $f(X) = g(X) \oplus h(X)$, such that the number of distinct products in the SOPs for $g(X)$ and $h(X)$ are minimal. Figure 1 shows an example of an AND-OR-EXOR three-level network. Note that the AND-OR-EXOR network in this paper is not an ordinary three-level network. It is a special type of three-level network, where the output EXOR gate has only two inputs. In a *general* three-level network, all the gates have unlimited fan-in. For example, gates in AND-OR-EXOR and OR-AND-OR three-level networks in [21] and [24], respectively, have no fan-in constraints.

An AND-OR-EXOR three-level network is suitable for implementing arithmetic functions. For example, Texas Instruments' SN181 arithmetic circuit and SN283 four-bit adder have two-input EXOR gates in the outputs [28]; Monolithic Memories' ZHAL20X8A eight-bit counter realizes EX-SOPs [14]. Programmable logic arrays (PLAs) with two-input EXOR gates at the outputs efficiently realize high-speed adders [31]. An 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]. An AND-OR-EXOR three-level network is also suitable for efficient implementation of many random functions. For example, simplified EX-SOPs for six-variable pseudo-random functions require 25 percent fewer products and 40 percent fewer literals than simplified SOPs [6]. For an arbitrary function of six variables, minimum SOPs require up to 32 products [11], while minimum EX-SOPs require at most 15 products [6].

It has been shown in [24] that three-level networks are sufficient to realize most of the logic functions with optimal number of gates. However, no significant reduction in the number of gates can be obtained by increasing the number of levels into four or more [24]. Therefore, three-level networks are especially desirable, because they require considerably fewer gates than two-level AND-OR networks for many functions. EX-SOPs produce not only a network with fewer gates, but also a decomposed network. A decomposed network is suitable for implementation in PLDs. In many cases, we cannot implement a

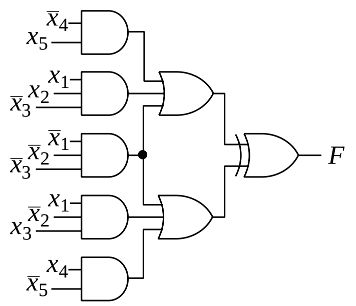

Fig. 1. Example of a minimum AND-OR-EXOR three-level network.

network in PLDs without decomposition [16]. Since the networks have only three levels, they are fast and their propagation delays are easy to estimate.

Design methods for AND-OR-EXOR three-level networks were considered in the past [10, 26]. A cut-and-try method was reported in [16] and a table look-up based heuristic algorithm to simplify EX-SOPs was shown in [4]. Several heuristic algorithms to simplify EX-SOPs were presented in [22], which also showed the benchmark results for EX-SOPs for the first time. Another heuristic algorithm to simplify EX-SOPs was reported in [9]. To the best of our knowledge, except [9] and [22], no other published benchmark results for EX-SOPs are available. Exact minimization algorithms for EX-SOPs with up to five variables were shown in [5, 7]. Recently, we developed algorithms to obtain optimal EX-SOPs for single-output six-variable functions [6]. Upper bounds on the number of products in minimum EX-SOPs were reported in [5–8]. Decompositions that have the form $f(X_1,X_2) = g(X_1) \oplus h(X_2)$ have been studied in [13, 25]. AND-OR-AND three-level networks, where the output AND gate have only two inputs were considered in [12]. Architecture of the PLDs with two-input logic elements at the outputs were illustrated in [12, 16, 22]. Design methods for EXOR-AND-OR networks, where EXOR gates are used at the input parts to reduce the complexity of the AND-OR networks, were presented in [30].

In this paper, we present an algorithm to simplify EX-SOPs. The algorithm uses a divide-and-conquer strategy: An EX-SOP for f is derived from the EX-SOPs for $f|_{x=0}$ and $f|_{x=1}$, where x is a variable on which f depends. The fewer the products in the EX-SOPs for $f|_{x=0}$ and $f|_{x=1}$, the fewer the products we can expect in the EX-SOP for f. The algorithm recursively applies the Shannon decomposition on a function with more than five variables. It generates minimum EX-SOPs for the five-variable functions by using a table look-up approach [5]. We then obtain simplified EX-SOPs for the six-variable functions by combining EX-SOPs for the five-variable functions. In the similar way, we obtain EX-SOPs for the n-variable functions by combining EX-SOPs for the $(n-1)$-variable functions. By using these techniques, we simplified EX-SOPs for a set of benchmark functions. We found that EX-SOPs require many fewer products and literals than SOPs for the majority of these functions.

The remainder of the paper is organized as follows: Section II introduces terminology. Section III presents the key concept to simplify EX-SOPs. Section IV develops an algorithm to simplify EX-SOPs for multiple-output functions. Section V reports the experimental results. Section VI presents conclusions.

II. Definitions and Terminologies

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 function.

Definition 2.1 *A sum-of-products expression (SOP) is the OR of product terms. An exclusive-OR (EXOR) sum-of-products expression (ESOP) is the EXOR of product terms. An EX-SOP is the EXOR of two SOPs.*

In the rest of the paper, unless otherwise specified, an EX-SOP represents an EX-SOP with product sharing.

Example 2.1 *Consider the logic function* $f(x_1,x_2,x_3,x_4,x_5) = \sum(5,6,9,10,13,14,17,18,20,22,23,24,25,27,29,30)$. *The network in Figure 1 realizes an EX-SOP for f:* $F = (\bar{x}_4x_5 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3) \oplus (\bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_4\bar{x}_5)$, *where $\bar{x}_1\bar{x}_2\bar{x}_3$ is the shared product between two SOPs of F.*

Definition 2.2 *An expression of a function is said to be minimum if it has the least product terms.*

Definition 2.3 *Let $\tau(F)$ be the number of distinct products in an expression F and $\tau(EX\text{-}SOP : f)$ be the number of distinct products in a minimum EX-SOP for f.*

Example 2.2 *In Example 2.1, $\tau(F) = 5$. F is a minimum EX-SOP, thus $\tau(EX\text{-}SOP : f) = 5$.*

Remark 2.1 *For the function f shown in Example 2.1, the minimum SOP requires 10 products and 38 literals, while the minimum EX-SOP requires only 5 products and 13 literals.*

Definition 2.4 *Two functions that are identical under the permutation of the variables and/or the negation of one or more variables are NP-equivalent [11, 15]. NP-equivalent functions form an NP-equivalence class of functions.*

III. Basic Properties

When product sharing is not permitted, an EX-SOP for an n-variable function can be derived from a pair of EX-SOPs for $(n-1)$-variable functions, without increasing the number of products [5, 8]. In this section, we prove that it is also true when the sharing of products in an EX-SOP is permitted.

Lemma 3.1 ([7]) *If* $f \cdot g = 0$, *then* $f \cdot (h_{11} \oplus h_{12}) \vee g \cdot (h_{21} \oplus h_{22}) = (fh_{11} \vee gh_{21}) \oplus (fh_{12} \vee gh_{22})$.

Theorem 3.1 *Let f be an arbitrary function and x be a variable on which f depends. Let $g = f|_{x=0}$ and $h = f|_{x=1}$. Let G and H be the EX-SOPs for g and h, respectively. Then, there is an EX-SOP F for f, such that $\tau(F) \leq \tau(G) + \tau(H)$.*

Proof: By using the Shannon decomposition with respect to x, we have

$$f = \bar{x}g \vee xh. \tag{3.1}$$

Let $G = G_{as} \oplus G_{bs}$, such that $G_{as} = G_a \vee G_s$ and $G_{bs} = G_b \vee G_s$, and let $H = H_{as} \oplus H_{bs}$, such that $H_{as} = H_a \vee H_s$ and $H_{bs} = H_b \vee H_s$. Here G_s and H_s represent shared products in the EX-SOPs G and H, respectively. Thus, $\tau(G) = \tau(G_a) + \tau(G_b) + \tau(G_s)$ and $\tau(H) = \tau(H_a) + \tau(H_b) + \tau(H_s)$. By putting the EX-SOPs for g and h into (3.1), we have an expression F_a for f:

$$F_a = \bar{x}(G_{as} \oplus G_{bs}) \vee x(H_{as} \oplus H_{bs}). \tag{3.2}$$

By using Lemma 3.1 to (3.2), we have two expressions F_b and F_c for f:

$$F_b = (\bar{x}G_{as} \vee xH_{as}) \oplus (\bar{x}G_{bs} \vee xH_{bs}), \tag{3.3}$$

$$F_c = (\bar{x}G_{as} \vee xH_{bs}) \oplus (\bar{x}G_{bs} \vee xH_{as}). \tag{3.4}$$

By putting the expressions for G_{as}, H_{as}, G_{bs}, and H_{bs} into (3.3), we have an EX-SOP F_d for f:

$$\begin{aligned} F_d &= (\bar{x}(G_a \vee G_s) \vee x(H_a \vee H_s)) \\ &\quad \oplus (\bar{x}(G_b \vee G_s) \vee x(H_b \vee H_s)). \\ \Rightarrow F_e &= ((\bar{x}G_a \vee xH_a) \vee (\bar{x}G_s \vee xH_s)) \\ &\quad \oplus ((\bar{x}G_b \vee xH_b) \vee (\bar{x}G_s \vee xH_s)). \end{aligned} \tag{3.5}$$

In (3.5), $\bar{x}G_s \vee xH_s$ represents shared products in the EX-SOP F_e for f. We can represent $\bar{x}G_a \vee xH_a$, $\bar{x}G_b \vee xH_b$, and $\bar{x}G_s \vee xH_s$ by the SOPs F_{aa}, F_{bb}, and F_{ss}, respectively, such that $\tau(F_{aa}) \leq \tau(G_a) + \tau(H_a)$, $\tau(F_{bb}) \leq \tau(G_b) + \tau(H_b)$, and $\tau(F_{ss}) \leq \tau(G_s) + \tau(H_s)$. Thus, we have an EX-SOP F for f:

$$F = (F_{aa} \vee F_{ss}) \oplus (F_{bb} \vee F_{ss}).$$

Note that,

$$\begin{aligned} \tau(F) &\leq \tau(G_a) + \tau(H_a) + \tau(G_b) + \tau(H_b) + \tau(G_s) + \tau(H_s), \\ &= \tau(G) + \tau(H). \end{aligned}$$

Hence, we have the theorem. ❑

Note that G and H in Theorem 3.1 need not to be minimized or simplified. From Theorem 3.1, we have the following:

Corollary 3.1 *Let f be an arbitrary function and x be a variable on which f depends. Then*

$$\tau(EX\text{-}SOP : f) \leq \tau(EX\text{-}SOP : f_0) + \tau(EX\text{-}SOP : f_1),$$

where $f_0 = f|_{x=0}$ and $f_1 = f|_{x=1}$.

IV. Simplification Techniques

Theorem 3.1 shows that an EX-SOP for an n-variable function can be derived from a pair of EX-SOPs for $(n-1)$-variable functions, without increasing the number of products. Our simplification technique is based on this concept.

In the proof of Theorem 3.1, we used (3.3) to derive an EX-SOP for f. In a similar way, we can derive another EX-SOP for f by using (3.4). Thus, we can produce a pair of EX-SOPs for f if EX-SOPs for g and h are available, and choose the EX-SOP with the fewer products as the final solution. However, this method often fails to obtain good quality solutions, because it searches only a limited part of the entire solution space. To improve the quality of the solutions, we consider many simplified EX-SOPs for g and h. Let we have k simplified EX-SOPs for each of g and h. Then by using the Shannon decomposition with respect to a particular variable, we can generate $2k^2$ simplified EX-SOPs for f. For an n-variable function, we can perform the Shannon decomposition in n different ways. Thus, we can produce $2nk^2$ simplified EX-SOPs for f.

Based on the discussions of this section, we have developed a recursive procedure, called SIMPLIFY_EX-SOP(f,k). The procedure returns a set of simplified or minimized EX-SOPs for the n-variable ($n \geq 5$) single-output function f, such that the number of EX-SOPs in the set is at most k. Figure 2 shows the pseudocode of the procedure. In the pseudocode, we use a set of two SOPs to represent an EX-SOP. For example, $\{F_1,F_2\}$ at line 27 represents an EX-SOP $F_1 \oplus F_2$, where F_1 and F_2 are SOPs. $P(f)$ holds a set of EX-SOPs for f. If $\{F_1,F_2\} \in P(f)$, then $f = F_1 \oplus F_2$. We use the following definition at lines 22 and 26:

Definition 4.1 *Let $\tau(G_1,G_2)$ be the total number of distinct products in the SOPs G_1 and G_2.*

When f is a function of five variables, the procedure generates a set of minimum EX-SOPs for f at line 14, such that the number of EX-SOPs in the set is at most k. At line 13, $|X|$ denotes the number of elements in the set X. Note that many functions have only one minimum EX-SOP. We have a minimization algorithm that produce a minimum EX-SOP with no product sharing for the five-variable functions [5]. We modified this algorithm to obtain a set of minimum EX-SOPs with no product sharing.

When f is a function of six or more variables, we use lines 16–33 of the pseudocode to obtain a set of simplified EX-SOPs for f. We use recursive calls to SIMPLIFY_EX-SOP(g,k) and SIMPLIFY_EX-SOP(h,k) at lines 18 and 19, respectively, where $g = f|_{x=0}$ and $h = f|_{x=1}$. The pseudocode shows that, when f is a function of five variables, no recursive calls are made. At lines 27 and 29, SIMPLIFY_SOP(f_a,f_b) returns a pair of simplified SOPs for f_a and f_b, such that the two SOPs can share products. There are efficient algorithms [1,18] to obtain simplified SOPs for multiple-output functions, such that the SOPs can share products. At line 34, if two or more EX-SOPs in $P(f)$ are the same, we eliminate all but one. We choose the best k EX-SOPs at line 34 by specifying fewer products as the primary criterion and fewer literals as the secondary criterion.

In order to improve the efficiency of our algorithm, we save all the intermediate functions and their EX-SOPs in S at line 36 of the pseudocode. It avoids many redundant computations. If S contains an element corresponding to f, then without any computation, we can immediately return $P(f)$ from S at line 11. Reuse of intermediate results is also employed in other areas of computer-aided design, such as to build binary decision diagrams [2]. By finding an element corresponding to f in S, we can avoid many recursive calls necessary to obtain EX-SOPs for f. The following observation illustrates the effectiveness of this strategy.

```
/* This procedure returns at most $k$ simplified or minimized EX-SOPs */
/* for the single-output $n$-variable ($n \geq 5$) function $f(x_1,x_2,\ldots,x_n)$. */
1 global var /* define global variables */
2     $S$ : set of ($f_{any}$,$P(f_{any})$); /* $P(f_{any})$ is defined at line 7 */
3 procedure SIMPLIFY_EX-SOP($f$,$k$) {
4     local var /* define local variables */
5         $x$ : variable; $X$ : set of variables; $g, h$ : logic functions;
6         $F$'s, $G$'s, $H$'s : SOPs; $t$ : number of products;
7         $P(f_{any})$ : set of EX-SOPs for $f_{any}$; $W$ : set of ($x$,$P(g)$,$P(h)$);
8     if this is the first call to this procedure then
9         $S \leftarrow \emptyset$;
10    if $S$ has an element corresponds to $f$ then
11        return $P(f)$;
12    $X \leftarrow$ {the variables of $f$};
13    if $|X| < 6$ then
14        $P(f) \leftarrow$ {at most $k$ minimum EX-SOPs for $f$};
15    else {
16        $W \leftarrow \emptyset$; $t \leftarrow \infty$; $P(f) \leftarrow \emptyset$;
17        for each $x \in X$ do {
18            $P(g) \leftarrow$ SIMPLIFY_EX-SOP($(g|g = f|_{x=0})$,$k$);
19            $P(h) \leftarrow$ SIMPLIFY_EX-SOP($(h|h = f|_{x=1})$,$k$);
20            $W \leftarrow W \cup \{(x,P(g),P(h))\}$;
21            for each $\{G_1,G_2\} \in P(g)$ and $\{H_1,H_2\} \in P(h)$ do
22                $t \leftarrow \mathbf{min}(t,\tau(G_1,G_2) + \tau(H_1,H_2))$;
23        }
24        for each $(x,P(g),P(h)) \in W$ do {
25            for each $\{G_1,G_2\} \in P(g)$ and $\{H_1,H_2\} \in P(h)$ do {
26                if $\tau(G_1,G_2) + \tau(H_1,H_2) = t$ then {
27                    $\{F_1,F_2\} \leftarrow$ SIMPLIFY_SOP($\bar{x}G_1 \vee xH_1$,$\bar{x}G_2 \vee xH_2$);
28                    $P(f) \leftarrow P(f) \cup \{\{F_1,F_2\}\}$;
29                    $\{F_1,F_2\} \leftarrow$ SIMPLIFY_SOP($\bar{x}G_1 \vee xH_2$,$\bar{x}G_2 \vee xH_1$);
30                    $P(f) \leftarrow P(f) \cup \{\{F_1,F_2\}\}$;
31                }
32            }
33        }
34        $P(f) \leftarrow$ {the best $k$ distinct EX-SOPs from $P(f)$};
35    }
36    $S \leftarrow S \cup \{(f,P(f))\}$;
37    return $P(f)$;
38 }
```

Fig. 2. Pseudocode of the procedure SIMPLIFY_EX-SOP.

Algorithm 4.1 *(Simplification of EX-SOPs for m-output ($m \geq 1$) functions)*

1. *For single-output function f, obtain a set of EX-SOPs by using* SIMPLIFY_EX-SOP(f,k) *and choose the EX-SOP with the fewest products as the final solution.*
2. *For m-output ($m \geq 2$) function, generate m single-output functions $f_1,f_2,\ldots,f_m$ and do the following steps.*
3. *For each f_i ($1 \leq i \leq m$), generate a set of EX-SOPs by using* SIMPLIFY_EX-SOP(f_i,k).
4. *From each set of EX-SOPs, retain the EX-SOPs with only the fewest products and delete other EX-SOPs.*
5. *Randomly choose one EX-SOP from each set of EX-SOPs. Let an EX-SOP for f_i ($1 \leq i \leq m$) is represented by two single-output functions f_{ia},f_{ib}, such that $f_i = f_{ia} \oplus f_{ib}$. Form the functions f_{1a},f_{1b}, $f_{2a},f_{2b},\ldots,f_{ma},f_{mb}$.*
6. *Simplify SOPs for $f_{1a},f_{1b},f_{2a},f_{2b},\ldots,$ f_{ma},f_{mb}, such that they can share products. There are efficient algorithms to do this [1, 18].*
7. *Continue steps 5 and 6 while reduction in the number of distinct products in the simplified SOPs for $f_{1a},f_{1b},f_{2a},f_{2b},\ldots,f_{ma},f_{mb}$ is possible.*
8. *The simplified SOPs for $f_{1a},f_{1b},f_{2a},f_{2b}$, $\ldots,f_{ma},f_{mb}$ with the fewest number of distinct products, obtained at step 7, is the final solution for the m-output function.*

Fig. 3. Algorithm 4.1.

Observation 4.1 *Let us consider a four-bit adder, namely adr4. To obtain a simplified EX-SOP for adr4 without using intermediate results, we must generate EX-SOPs for 12,140 five-variable functions. However, if we save and reuse intermediate results, then we have to generate EX-SOPs for only 509 five-variable functions.*

We can further improve the efficiency of our algorithm by considering NP-equivalence of logic functions. If two functions f_a and f_b are NP-equivalent, then we can generate EX-SOPs for f_a from the EX-SOPs for f_b without doing any logic minimization. To generate EX-SOPs for f_a from the EX-SOPs for f_b, we find a Boolean match [3] between f_a and f_b. There are good algorithms [3] to find a Boolean match between two functions. Thus, if S contains an element corresponding to f_b, then without performing any logic minimization, we can return $P(f_a)$ from S at line 11.

Observation 4.2 *If we save intermediate results in S and reuse EX-SOPs for the NP-equivalent functions from S, then we have to generate EX-SOPs for only 55 five-variable functions to obtain a simplified EX-SOP for adr4. Observation 4.1 shows that EX-SOPs for 12,140 five-variable functions are necessary when intermediate results are not saved. Note that, 55 is the 0.45 percent of 12,140. This is a significant reduction in the number of five-variable functions. We found similar tendencies for many other logic functions.*

Based on the procedure SIMPLIFY_EX-SOP(f,k), we developed Algorithm 4.1 to simplify EX-SOPs (Figure 3).

V. Experimental Results

We implemented Algorithm 4.1 in C. The program requires about 40 megabytes of memory space. The quality of the solution and the computation time of Algorithm 4.1 for function f depend on the parameter k in the procedure SIMPLIFY_EX-SOP(f,k). For functions with

TABLE I
NUMBER OF PRODUCTS AND LITERALS TO REALIZE BENCHMARK FUNCTIONS

			Number of products				Number of literals			
Data	In	Out	SOP	ESOP	EX-SOP	$\frac{\text{EX-SOP}}{\text{SOP}}$	SOP	ESOP	EX-SOP	$\frac{\text{EX-SOP}}{\text{SOP}}$
5xp1	7	10	63	32	34	0.54	278	120	108	0.39
9sym	9	1	84	51	65	0.77	504	374	392	0.77
addm4	9	8	189	91	100	0.53	1225	521	535	0.44
adr3	6	4	31	15	13	0.42	116	44	28	0.24
adr4	8	5	75	31	26	0.35	340	112	78	0.23
clip	9	5	117	67	72	0.62	631	402	354	0.56
cm82a	5	3	23	13	9	0.39	80	33	17	0.21
f51m	8	8	76	32	35	0.46	326	112	109	0.33
inc8	8	9	37	15	16	0.43	100	43	46	0.46
life	9	1	84	49	62	0.74	672	311	443	0.66
log8	8	8	123	104	100	0.81	730	550	514	0.70
mlp4	8	8	121	62	75	0.62	736	305	354	0.48
nrm4	8	5	120	69	80	0.67	716	391	428	0.60
rd53	5	3	31	14	17	0.55	140	39	58	0.41
rd73	7	3	127	35	54	0.43	756	134	254	0.34
rd84	8	4	255	59	99	0.39	1774	267	547	0.31
rdm8	8	8	76	32	35	0.46	325	112	110	0.34
rot8	8	5	57	36	42	0.74	305	197	204	0.67
sqr8	8	16	180	112	134	0.74	1068	546	606	0.57
squar5	5	8	25	20	20	0.80	95	57	49	0.43
z4	7	4	59	29	22	0.37	252	111	107	0.42

6, 7, 8, and 9 variables, we use $k = 10$, 8, 5, and 3, respectively. We found that a higher value of k drastically increases the computation time without any significant improvement in the solution.

We obtained three types of expressions for a set of benchmark functions. Table I compares the number of products and literals in these expressions. In this experiment, SOPs were minimized by using Quine-McCluskey algorithm for multiple-output functions [15]; ESOPs were simplified by using EXMIN2 [20]; and EX-SOPs were simplified by using Algorithm 4.1. This table shows that, for the set of benchmark functions, EX-SOPs require many fewer products and literals than SOPs. For many functions, the numbers of products and literals in the ESOPs and EX-SOPs are nearly the same.

Table II compares experimental results for Algorithm 4.1 and AOXMIN [9], another heuristic simplification program for EX-SOPs. For all the benchmark functions shown in this table, Algorithm 4.1 outperforms AOXMIN. The improvement is up to 48 percent for rd84. It should be noted that AOXMIN can simplify EX-SOPs for functions with more inputs and the computation time of AOXMIN is smaller than that of Algorithm 4.1.

TABLE II
COMPARISON WITH AOXMIN [9]

			Number of products in EX-SOPs		
Data	In	Out	AOXMIN	Proposed method	$\frac{\text{Proposed method}}{\text{AOXMIN}}$
5xp1	7	10	42	34	0.81
9sym	9	1	73	65	0.89
clip	9	5	95	72	0.76
rd53	5	3	19	17	0.89
rd73	7	3	83	54	0.65
rd84	8	4	192	99	0.52

TABLE III
COMPARISON WITH ANOTHER METHOD [22]

			Number of products in EX-SOPs		
Data	In	Out	Ref.[22]	Proposed method	$\frac{\text{Proposed method}}{\text{Ref.[22]}}$
adr4	8	5	37	26	0.70
inc8	8	9	15	16	1.07
log8	8	8	116	100	0.86
mlp4	8	8	109	75	0.69
nrm4	8	5	93	80	0.86
rd84	8	4	135	99	0.73
rdm8	8	8	54	35	0.65
rot8	8	5	49	42	0.86
sqr8	8	16	176	134	0.76

Table III compares the number of products generated by Algorithm 4.1 with those generated by another heuristic simplification program for EX-SOPs [22]. For the nine benchmark functions shown in this table, on the average, Algorithm 4.1 produced solutions which require about 23 percent fewer products than the methods presented in [22]. The greatest improvement, 35 percent is for rdm8. It should be noted that the techniques presented in [22] can simplify EX-SOPs with large number of inputs, but Algorithm 4.1 can simplify EX-SOPs with only small number of inputs. Also, Algorithm 4.1 requires more computation time and memory than the methods presented in [22].

VI. CONCLUDING REMARKS

In this paper, we developed a heuristic algorithm to design AND-OR-EXOR three-level network for multiple-output functions, where EXOR gates at the outputs of the network have only two inputs. The network realizes EX-SOPs. Our present data structure for the algorithm can handle functions with up to nine variables. We are modifying the data structure. The modified implementation will be able to handle functions with more variables. An AND-OR-EXOR network can be efficiently implemented in many commercial PLDs, or AND-OR part of the AND-OR-EXOR network can be used as an initial network for multi-level design by using fan-in limited gates. For the majority of the functions shown in Table I, EX-SOPs require many fewer products than SOPs. However, there are functions for which EX-SOPs require as many products as SOPs; for those functions we should use AND-OR two-level networks. To implement an EX-SOP, we need a two-input EXOR gate in addition to AND and OR gates. A two-input EXOR gate is several times more expensive than a two-input OR gate. Let us consider the function squar5 shown in Table I. Its AND-OR-EXOR network requires eight more EXOR gates and five fewer AND gates than AND-OR network. Thus, AND-OR-EXOR network is unattractive for squar5. On the other hand, AND-OR-EXOR based design is attractive for adr4. An AND-OR-EXOR network for adr4 requires five more EXOR gates and 49 fewer AND gates than AND-OR network. For AND-OR-EXOR and AND-OR networks of adr4, the ratio of the fan-in of the AND gates is 0.23. Our experimental results in Table I show that AND-OR-EXOR network is very attractive for the efficient realization of many practical logic functions.

Acknowledgement

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. We would like to thank Prof. Jon. T. Butler for carefully reviewing the manuscript.

References

[1] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[2] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[3] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[4] 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.

[5] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level networks," *Proc. Asia and South Pacific Design Automation Conference*, pp. 545–550, Jan. 1997.

[6] D. Debnath and T. Sasao, "Exclusive-OR of two sum-of-products expressions: Simplification and an upper bound on the number of products," *Proc. 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Oxford, U.K., pp. 45–60, Sept. 1997.

[7] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," *IEICE Trans. Information and Systems*, Vol. E80-D, No. 10, pp. 1001–1008, Oct. 1997.

[8] 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, Mar. 1995.

[9] E. V. Dubrova, D. M. Miller, and J. C. Muzio, "AOXMIN: A three-level heuristic AND-OR-XOR minimizer for Boolean functions," *Proc. 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Oxford, U.K., pp. 209–218, Sept. 1997.

[10] 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. International Conference on Computer-Aided Design*, pp. 203–205, Nov. 1985.

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

[12] A. A. Malik, D. Harrison, and R. K. Brayton, "Three-level decomposition with application to PLDs," *Proc. International Conference on Computer Design*, pp. 628–633, Oct. 1991.

[13] Y. Matsunaga, "An attempt to factor logic functions using exclusive-OR decomposition," *Proc. The Sixth Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI'96)*, Fukuoka, Japan, pp. 78–83, Nov. 1996.

[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 (in Japanese).

[18] T. Sasao, "Input variable assignment and output phase optimization of PLA's", *IEEE Trans. Comput.*, Vol. C-33, No. 10, pp. 879–894, Oct. 1984.

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

[20] 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. CAD-12, No. 5, pp. 621–632, May 1993.

[21] T. Sasao, "Logic synthesis with EXOR gates," in T. Sasao, ed., *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[22] T. Sasao, "A design method for AND-OR-EXOR three-level networks," *Proc. International Workshop on Logic Synthesis*, Lake Tahoe, California, pp. 8:11–8:20, May 1995.

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

[24] T. Sasao, "OR-AND-OR three-level networks," in T. Sasao and M. Fujita, eds., *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[25] T. Sasao and J. T. Butler, "On bi-decompositions of logic functions," *Proc. International Workshop on Logic Synthesis*, Lake Tahoe, California, May 1997.

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

[27] N. Song and M. A. Perkowski, "Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, No. 4, pp. 385–395, April 1996.

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

[29] C. Tsai and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions," *Proc. 33rd Design Automation Conference*, pp. 242–247, June 1996.

[30] D. Varma and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-8, No. 8, pp. 901–916, Aug. 1989.

[31] A. Weinberger, "High-speed programmable logic array adders," *IBM Journal of Research and Development*, Vol. 23, No. 2, pp. 163–178, Mar. 1979.

[32] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1993.