Efficient Synthesis of AND/XOR Networks

Yibin Ye Kaushik Roy

School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285, USA.
ye@ecn.purdue.edu kaushik@ecn.purdue.edu

Abstract— **A new graph-based synthesis method for general Exclusive Sum-of-Product forms (ESOP) is presented in this paper. Previous research has largely concentrated on a class of ESOP's, the Canonical Restricted Fixed/Mixed Polarity Reed-Muller form, also known as Generalized Reed-Muller (GRM) form. However, for many functions, the minimum GRM can be much worse than the ESOP.**

We have defined a *Shared Multiple Rooted* XOR-based Decomposition Diagram (XORDD) to represent functions with multiple outputs. By iteratively applying transformations and reductions, we obtain a compact XORDD which gives a minimized ESOP. Our method can synthesize larger circuits than previously possible. The compact ESOP representation provides a form that is easier to synthesize for XOR heavy multi-level circuit, such as arithmetic functions. The method successfully minimized large functions with multiple outputs. Results are also compared to the minimized SOP's obtained from ESPRESSO. Experimental results show that for many circuits ESOP's have considerably more compact form than SOP's.

I. Introduction

In many applications, the AND/XOR realizations of digital circuits are very efficient. It has long been the experience of circuit designers that AND/XOR forms are more economical in large classes of circuits such as those used in arithmetic, telecommunication, and linear systems.

While the slow speed of the XOR gate has been the main obstacle, recent advances in technologies make the use of this logic more practical. Circuits in AND/XOR forms can be easily mapped to Field Programmable Gate Arrays (FPGAs). Moreover, in table-lookup FPGAs such as the Xilinx 3000 family, XOR gates do not cause any extra cost in terms of chip area and speed.

While powerful optimization tools for AND/OR based synthesis have been developed in the last few years, (such as ESPRESSO [9], MIS [1] and BOLD [5]), there are few minimization and synthesis tools available which include AND/XOR realizations.

There are several classes of the AND/XOR forms, of which the Exclusive Sum-of-Product (ESOP) expression is the most general 2-level form. The problem of finding the minimal SOP's of a Boolean function is a classical problem in switching theory which has been studied for many years. The minimization of ESOP's is much more difficult than that of SOP's. So far no efficient method is known to obtain a minimum ESOP of a function except for those with very small number of variables. Many previously published algorithms consider the canonical-restricted forms [7], [8], [3], such as the Canonical Restricted Fixed/Mixed Polarity Reed-Muller form, also known as Generalized Reed-Muller (GRM) form. However, for most functions, the minimum GRM is worse than the ESOP.

The research was supported in part by NSF (9633516-MIP) and by ARPA (F33615-95-C-1625), and IBM Corporation.

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

Recently, Sasao [11] developed a heuristic approach which applied methods similar to ESPRESSO by iteratively finding smaller covers of products and by looking for parity-like patterns. While ESPRESSO succeeds in minimizing the SOP's of most practical functions, a similar methodology is less powerful when applied to minimize the ESOP's. This is due to the fact that only prime implicants need to be searched in the minimization of SOP's, while the search space is drastically larger in minimizing the ESOP's.

In this paper, we present a new graph-based minimization method for general ESOP's. We start from the OBDD [2] representation of a function (multiple primary outputs in general). By iteratively applying transformations and reductions, we finally obtain a compact *Shared Multiple Rooted* XOR-based Decomposition Diagram (XORDD) which gives a minimized ESOP. Our method promises to be efficient since transformations are performed by adding/removing edges and by changing edge values rather than computing new sub-functions. Note that building the initial BDD is not an additional cost because the BDD provides a *disjoint Sum-of-Product* form (as the initial ESOP) before applying minimization procedures. In fact, if the original circuit descriptions are in multi-level combinational format, creating an OBDD is usually far more efficient than flattening the multi-level circuit and then transforming the flattened circuits into *disjoint SOP* form. Results on benchmark circuits are compared with the SOP forms obtained from ESPRESSO to show the effectiveness and efficiency of our algorithm.

II. Preliminaries and Definitions

A. Exclusive Sum-of-Products Representations of Boolean Functions

A Boolean function of n variables $f: \mathbf{B}^n \rightarrow \mathbf{B}$, can be written in the *Reed-Muller Canonical* (GMC) form as:

$$f(x) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_1 x_2 \oplus a_4 x_3 \oplus \cdots \oplus a_{2^n-1} x_1 x_2 \cdots x_n \quad (1)$$

where the RM (Reed-Muller) coefficients a_i can take the values of 0 or 1.

If x_i in each product term is allowed to take either *positive polarity* x_i or *negative polarity* $\overline{x}_i$, then we have a *Generalized Reed-Muller* (GRM) form, or *Mixed Polarity Reed-Muller Canonical* (MPRM) form representation. When each variable is restricted to retain the same polarity, either positive or negative, then the expression is that of *Fixed Polarity Reed-Muller Canonical* (FPRM) form. While the GMC representation of a Boolean function is unique, there are 2^n possible FPRM representations and $2^{n2^{n-1}}$ possible MPRM representations for a function of n variables [7].

A general and most compact AND/XOR form is that of the ESOP representation which is not necessarily canonical. In the following example, we compare the four AND/XOR forms we have discussed above.

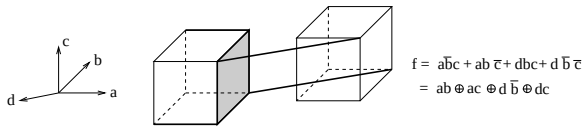


Fig. 1. An example function in both SOP and ESOP forms.

Example 1: Consider the function $f = x_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3$. The various AND/XOR forms are given by:

$$\begin{aligned} f &= 1 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 && \text{(GMC)} \\ &= x_1 \oplus x_1\overline{x}_2 \oplus x_1\overline{x}_3 \oplus \overline{x}_2\overline{x}_3 && \text{(FPRM)} \\ &= x_1x_2 \oplus x_1\overline{x}_3 \oplus \overline{x}_2\overline{x}_3 && \text{(MPRM)} \\ &= x_1x_2x_3 \oplus \overline{x}_1\overline{x}_2\overline{x}_3 && \text{(ESOP)} \end{aligned}$$

Now let us consider an example to compare the SOP and ESOP representations for a simple function shown in Fig. 1.

Example 2: Consider the function shown in Fig. 1. The minimal SOP representation is $f = ab\overline{c} + a\overline{b}c + dbc + d\overline{b}\overline{c}$, which has 4 product terms and 12 literals, and the minimal ESOP representation is $f = ab \oplus ac \oplus d\overline{b} \oplus dc$, which has 4 product terms and 8 literals. Note that the minimal SOP form is unique. The minimal ESOP representation, however, has a number of equivalent expressions.

B. Graph Representations of Boolean Functions

Bryant presented the Ordered Binary Decision Diagram (OBDD) [2] as a canonical form for Boolean functions. It is currently one of the most popular data structures for the representation of Boolean functions. OBDDs have been widely used in logic synthesis, verification, and testing. While OBDDs are MUX-based representations, another type of Decision Diagram, the Ordered Functional Decision Diagrams (OFDD), have been introduced [3], which are based on AND/XOR representations. In the following paragraph, we first examine the decomposition equations on which the OBDDs and OFDDs are based, then we define our new graph representations for Boolean functions.

Let $f: \mathbf{B^n} \rightarrow \mathbf{B}$ be a Boolean function over the variable set $\mathbf{X_n}$. Let f_i^0 denote the *cofactor* of f with respect to $x_i = 0$ given by $f_i^0(x) = f(x_1, \cdots x_{i-1}, 0, x_{i+1}, \cdots, x_n)$. Similarly, f_i^1 denotes the *cofactor* of f with respect to $x_i = 1$ given by $f_i^1(x) = f(x_1, \cdots x_{i-1}, 1, x_{i+1}, \cdots, x_n)$. We also define f_i^2 as $f_i^2 = f_i^0 \oplus f_i^1$. Note that all f_i^0, f_i^1 and f_i^2 are independent of x_i. Using the above notations, we have the following decompositions:

$$\begin{aligned} f &= \overline{x}_i f_i^0 + x_i f_i^1 = \overline{x}_i f_i^0 \oplus x_i f_i^1 && (2) \\ &= f_i^0 \oplus x_i f_i^2 && (3) \\ &= f_i^1 \oplus \overline{x}_i f_i^2 && (4) \end{aligned}$$

While the OBDD representation of a Boolean function is based on equation (2), the OFDD is based on equation (3) and (4). Many functions that have large cube covers have compact OBDD representations. Practical functions with compact cube covers seldom have very large OBDDs [6] (although one can design some functions that are of simple SOP forms to have exponential OBDD size). It is known that for certain classes of functions, the size of OFDDs are significantly smaller than that of OBDD representations, and vice versa.

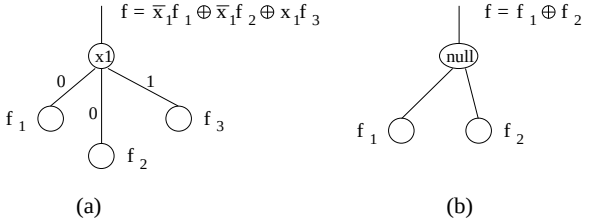


Fig. 2. The XORDD representation of functions

We now define *General Decomposition Diagrams* (GDD) and *XOR-based Decomposition Diagrams* (XORDD) which are used for representation and minimization of ESOP's.

Definition 1: A General Decomposition Diagram (GDD) over a set of variables $\mathbf{X} := \{null, x_1, x_2, \cdots, x_n\}$ is a rooted directed acyclic graph (DAG) G=(V,E) with two types of vertices, *non-terminal* and *terminal*. A non-terminal vertex v is associated with an operation (OR, XOR, etc.) as well as a variable $\in \mathbf{X}$, and has one or more children vertices $\in V$. Edges from a non-terminal vertex of variable x_i to its children vertices are labeled with either a 0 or 1, which give the edge value $\overline{x}_i$ or x_i, respectively. Edges from a non-terminal vertex associated with *null* variable are not labeled and always have the value of 1. A terminal vertex is labeled with either a 0 or a 1 and has no children vertices. Each variable is encountered at most once when traversing from root to a terminal vertex.

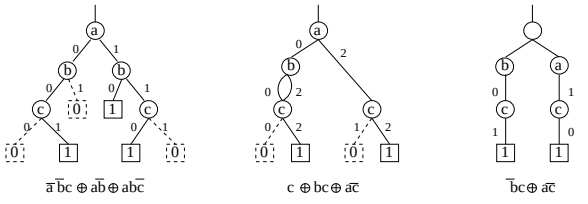
The GDD is simply a generic graph representation of Boolean functions. The only restriction that distinguishes GDD from an arbitrary multi-level logic representation is the condition that each variable is encountered at most once when traversing from root to a terminal vertex in a GDD. This restriction is due to the type of decompositions to be imposed on the graph.

Definition 2: An XOR Decomposition Diagram (XORDD) over a set of variables $\mathbf{X} := \{null, x_1, x_2, \cdots, x_n\}$ is a special case of GDD, in which only the XOR operation is associated with each non-terminal vertex.

The XORDD is defined in such a way that any ESOP can be represented by an XORDD (Shown later in this section). Unlike an OBDD or OFDD, an XORDD is not a decision diagram. It is a decomposition diagram as defined earlier. A non-terminal vertex in an XORDD can have one, two, or more children vertices. Edges going from a vertex can have the same label (and hence, the same value). Fig. 2 illustrates how an XORDD relates to the function it represents. Assigning a value to each edge is an important feature in XORDD representations, because changing the type of decompositions can be realized by changing the edge connections and edge values rather than computing new sub-functions.

BDD is a special case of XORDD based on our definition of XORDD. OFDD, however, does not belong to the class of XORDD because of the different rules used for assigning the edge values. We assume the readers are familiar with the OBDD and OFDD representations of a Boolean function. For the purposes of comparison, the following example gives the three graph representations given by OBDD, OFDD and XORDD, respectively.

Example 3: Let us consider the function $f = a\overline{b} + a\overline{c} + \overline{b}c$. Fig. 3 shows all three types of graph representations. Note that in the OFDD, both decomposition types of equation (3) and (4) have been applied, where label 0 denotes f_i^0, 1 de-



(a) OBDD (b) OFDD (c) XORDD

Fig. 3. The same function represented by OBDD, OFDD and XORDD, respectively

notes f_i^1 and 2 denotes f_i^2.

While XORDD is a multi-level representation, the two-level AND/XOR form can be easily constructed from the graph as follows: A *1-path* is a path from root to a 1-labeled terminal vertex. Each 1-path defines a product term which is the product of all the edge values in the path. The function can then be expressed in form of XOR sum of product terms of all 1-paths. Therefore, the number of 1-paths is equal to the number of terms in the ESOP's. One can observe that the 0-labeled terminal vertex is not useful in the XORDD representations. Unless otherwise mentioned, the XORDD's have only one type of terminal vertex with label 1.

Let us consider Example 3 and Fig. 3 again. The two-level expressions given by OBDD, OFDD and XORDD are $\overline{a}\,\overline{b} \oplus a\overline{b} \oplus ab\overline{c}$, $c \oplus bc \oplus a\overline{c}$ and $\overline{b}c \oplus a\overline{c}$, respectively.

It is important to note that an OFDD representation of a function always gives the GRM form, i.e., canonical form, while XORDD representation gives a more general ESOP form. Conversely, an ESOP expression can always be represented by an XORDD, and only Fixed Polarity Reed-Muller expressions can be represented by OFDDs.

So far, only single output functions (hence single rooted XORDD's) have been discussed. For functions with multiple outputs, we construct *Shared Multiple Rooted* XORDD representations, in which common internal vertices are shared by several component functions. While this extension is straightforward, we need to note that common 1-path should be counted only once in determining the number of product terms of ESOP's. Since in most cases we deal with multiple-output circuits, unless otherwise mentioned, the XORDD is a *shared multiple rooted graph*.

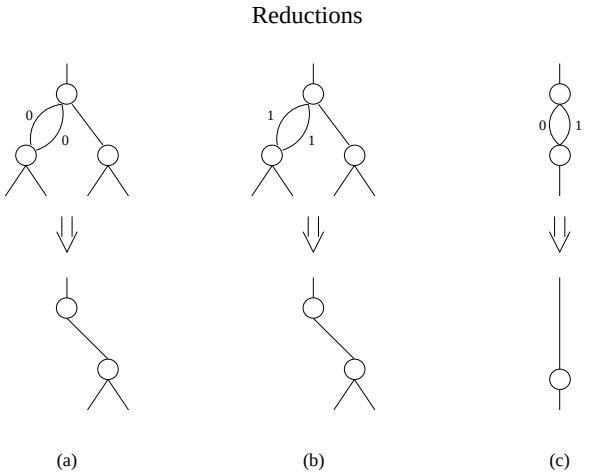
We now state our synthesis strategy of minimizing ESOP's. Starting from a shared OBDD representation of a multiple output function, we iteratively apply transformations and reductions to finally obtain a *shared multiple rooted* XORDD which gives a minimized ESOP.

III. Algorithms

A. Transformations and Reductions on XORDDs

For a given XORDD, the following reduction rules can be applied to reduce the size of XORDD.

1. Delete terminal vertices labeled with 0.
2. If two edges between two vertices have the same value, then they are removed (Fig. 4(a) and 4(b)).
3. If two edges from vertex v to v' are labeled with 0 and 1 respectively, we remove them and redirect edges pointing to v to point to v' (Fig. 4(c)).
4. Delete non-terminal vertices that do not have successors.



(a) (b) (c)

Fig. 4. Three basic reduction rules of XORDD

5. Delete non-terminal vertices (except root vertices) that do not have predecessors.

Reduction rule (1) is necessary because the initial XORDD we start with is an OBDD which has two terminal vertices labeled with 0 and 1 respectively. Reduction rule (2) and (3) are illustrated in Fig. 4, which are based on the following facts:

$$x_i g_j \oplus x_i g_j = \overline{x}_i g_j \oplus \overline{x}_i g_j = 0 \quad (5)$$

$$x_i g_j \oplus \overline{x}_i g_j = g_j(x_i \oplus \overline{x}_i) = g_j \quad (6)$$

After applying reduction rules (2) and (3), there may exist vertices that do not have successors or predecessors. Then reduction rules (4) and (5) can be applied. Besides the reductions given above, the following transformations on an XORDD will be applied on an XORDD.

Change of decomposition: The three types of decomposition described in equations (2) – (4) can be rewritten as follows:

$$\overline{x}_i f_i^0 \oplus x_i f_i^1 = f_i^0 \oplus x_i(f_i^0 \oplus f_i^1) = f_i^1 \oplus \overline{x}_i(f_i^0 \oplus f_i^1),$$

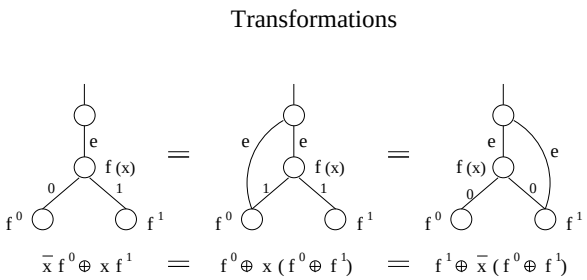
where $f_i^0 \oplus f_i^1 = f_i^2$, which was defined in section II-B. The change from one type of decomposition to another can be realized by changing the value of an edge and adding another edge. This is illustrated in Fig. 5(a), where the label $e \in \{0, 1\}$.

Split: A vertex with multiple incoming or outgoing edges can be split into two or more vertices, as illustrated in Fig. 5(b).

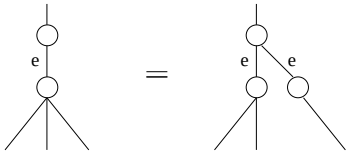
Merge: If two or more outgoing (incoming) edges of a vertex have the same value and go to (come from) vertices of the same variable, then they can be merged as shown in Fig. 5(c).

Extract: If two or more incoming (outgoing) edges of a vertex have the same value, they may be extracted in a way as illustrated in Fig. 5(d). *Extract* is essentially a compound operation consisting of *split* and *merge*.

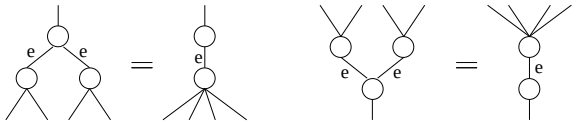
As one can see, applying reduction rules always reduce the size of XORDD (reduce edges and/or nodes). However, transformations do not directly result in a more compact graph. The purpose of performing transformations on an XORDD is to create opportunities for more reductions.



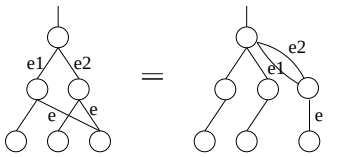
(a) Change the decomposition type



(b) Split



(c) Merge



(d) Extract

Fig. 5. Basic transformation steps of XORDD

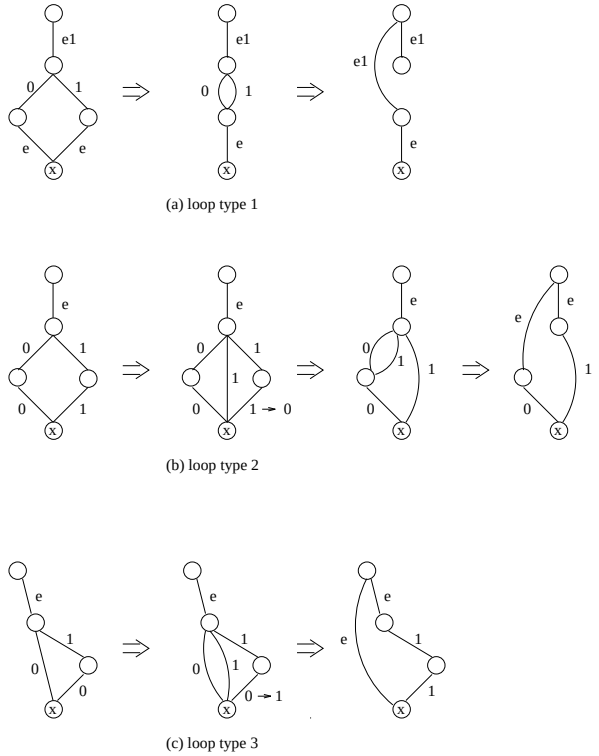


Fig. 6. Three types of loops are identified, on which reductions can be applied after performing certain transformations.

Thus, we need to identify which types of transformations should be performed on a particular node or edge. In our synthesis techniques, *local loop detection* is used in determining transformations and reductions.

Local loop detection: Detection of three basic loops are performed in our synthesis technique. These are illustrated in Fig. 6. In *loop type 1* (Fig. 6(a)), we can first *merge* two nodes and then apply reduction rule (3). In *loop type 2* (Fig. 6(b)), an edge label is changed first. We then *merge* two nodes followed by a reduction. In *loop type 3*(Fig. 6(c)), after changing an edge label, a reduction can be applied.

B. Algorithms of Minimization

Our goal is to obtain a compact XORDD from the initial large XORDD by iteratively applying transformations and reductions. The number of product terms and literals should be simultaneously minimized. We concluded in the previous section that the number of product terms is equal to the number of 1-paths in an XORDD and that each 1-path defines a product term which is the product of all edge values in the path. Thus, we can recursively calculate the number of terms and literals from lowest level vertices to roots in a *shared multiple rooted* XORDD by the following two equations:

$$p(v) = \sum_{v' \in successors\ of\ v} p(v') \tag{7}$$

$$l(v) = \sum_{v' \in successors\ of\ v} (l(v') + p(v')e(v)) \tag{8}$$

where $e(v)$ is defined by

$$e(v) = \begin{cases} 1, & \text{if node } v \text{ is associated with a variable} \\ 0, & \text{if node } v \text{ is associated with the } null \text{ variable.} \end{cases}$$

One exception is that a common 1-path shared by multiple roots should be counted only once. For two-level ESOP minimization, the two functions defined in equation (7) and (8) are the objective functions which need to be minimized. By examining the reductions and transformations introduced in section III-A, one can observe that reductions always reduce the objective functions. While *merge*, *split*, and *extract* do not affect the objective functions, *change of decomposition* actually increases the value of objective functions. However, more reductions are possible only after certain transformations as we have discussed in section III-A.

An algorithm to minimize the objective functions of the XORDD is given below:

Step 1: Create a *Multiple Rooted Shared* OBDD (as the initial XORDD) from the original two-level or multi-level circuit descriptions.
Step 2: Recursively compute the two objective functions (# of terms and literals) for the initial XORDD. Compare the number of 1-paths and 0-paths. If there are considerably more 1-paths than 0-paths, change the 0-terminal into 1-terminal and vice versa. Add a terminal

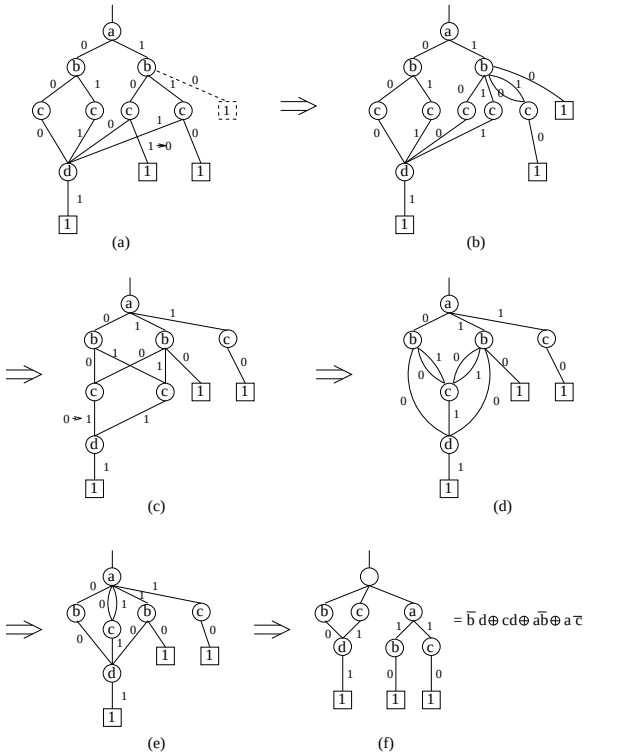


Fig. 7. An example showing how the algorithm is performed on a function—-from initial BDD to the final compact XORDD, which gives a minimized ESOP form

edge for every root vertex. This is equivalent to inverting the function twice. (Exchange of the 0-terminal and the 1-terminal inverts the function once. Putting a 1-terminal edge on the root vertices inverts the function again $(1 \oplus f = \overline{f})$). Delete the 0-terminal and all its incoming edges.

Step 3.1: Start from the lowest level vertices. Examine each node to determine if reductions can be directly applied.

Step 3.2: Detect three reducible loops discussed in section III-A (also shown in Fig. 6). Change the label of some incoming edges (i.e., change decompositions), then apply reductions on parent node(s).

Step 4: For every vertex in the next lowest level, repeat the same transformations and reductions given in steps 3.1 and 3.2.

Step 5: Level by level from the bottom up to the root vertices, repeat the same operations given in steps 3.1 and 3.2.

Step 6: For each node, examine the incoming edges, do *merge* and *extract* as shown in Fig. 5 to change the topology of the graph. Then repeat step $3-5$.

Let us consider the function shown in Fig. 1 as an example. The initial OBDD after removing the 0-terminal is shown in Fig. 7(a). First, consider edges of the 1-terminal. *Merge* or *extract* can not be directly applied. After changing an edge label from $1 \rightarrow 0$, the two nodes of variable c can be merged and reduction can be applied. Thus we obtain the graph of Fig. 7(b). We now consider the next lowest level variable d. By examining the four incoming edges of node d, the four nodes of variable c can be merged into two nodes, as shown in Fig. 7(c). From node d, two *type 2* reducible loops (both are $d \rightarrow c \rightarrow b \rightarrow c \rightarrow d$) can be detected. By changing the label (from 0 to 1) of the edge between node d and c, we can merge the two nodes of variable c to obtain Fig. 7(d). We now consider variable c. A number of reductions are available on node c. After reductions, we have the graph in Fig. 7(e). There is a new reduction available between node c and node a. We can also merge two nodes of b. After a final reduction step, we obtain the XORDD of Fig. 7(f), which gives the minimum ESOP expression.

IV. Experimental Results

The XORDD-based synthesis method is applied to a large set of benchmark circuits in both PLA and combinational formats. Results are summarized in tables I. In the first column, circuits with an asterisk are in PLA format. *in* and *out* represent the number of inputs and outputs of the circuits. *cpu* denotes the cpu time in seconds on SPARC 5 workstation. Unlike most previous works [7], [8], we synthesized multiple outputs simultaneously to explore the common product terms shared by multiple outputs.

Minimized SOP's obtained from ESPRESSO are also included in the tables. For many functions, e.g. *5xp1, apex5, rd53, rd73, rd84, t481, xor5* etc., the ESOP's have considerably fewer product terms than the SOP's. On the other hand, for functions such as *apex1, apex3, cps, spla,* etc., the SOP's are significantly more compact. On average, the compactness of these two forms are comparable. For some particular types of functions, one representation is better than the other. On average, the XORDD-based synthesis algorithm consumes a comparable *cpu* time with that of ESPRESSO. However, for many larger circuits we examined, the XORDD-based algorithm consumes significantly less *cpu* time. Note that most of the *cpu* time consumed in XORDD-based synthesis is for the initial BDD creation. The synthesis algorithm usually consumes less than 20% of the total *cpu* time reported in the table.

For some combinational circuits, no compact two-level forms in either ESOP or SOP could be obtained. We experimented with circuits such as *i3, i4, rot, pair, C432,* and *C1908*. The number of product terms are more than tens of thousands for such circuits. We have not listed these circuits in Table I. As long as BDDs are successfully created, the XORDD-based method can always generate results. Table II listed several circuits for which ESPRESSO did not produce any result in 30 minutes on a SUN SPARC 5 workstation. The last column *node* in Table II represents the number of nodes in an XORDD. It should be noted that the size of XORDD's is still small even though the circuits have large number of product terms and literals. The shared multi-level structure of XORDD's gives a compact representation of these functions. Moreover, the *cpu* time depends on the size of the graph. For some even larger circuits, such as *C2670, C3540,* and *C7550*, we are unable to have compact XORDDs because of the excessive size of the initial BDDs. However, we feel that these circuits are inherently not suited for two-level realizations.

V. Conclusions and Future Work

We have defined a new *Shared Multiple Rooted* XOR-based Decomposition Diagram to represent multiple output functions. Based on this type of graph, we presented an efficient algorithm to minimize the ESOP's.

In this paper, only two-level AND/XOR minimization has

TABLE I

Minimized ESOP's for benchmark circuits and their comparison with SOP's from ESPRESSO

Circuit			ESPRESSO			XORDD-based Synthesis		
	in	out	terms	literals	cpu	terms	literals	cpu
5xp1*	7	10	65	347	0.3	42	146	0.5
9sym*	9	1	86	602	0.6	84	548	0.6
apex1*	45	45	206	2842	5.7	518	4692	5.7
apex3*	54	50	281	3303	7.4	449	3432	13.1
apex5*	117	88	1088	7281	106.3	428	4056	5.4
b12*	15	9	43	207	0.6	30	145	0.4
cps*	24	109	163	2836	10.7	342	4340	8.2
e64*	65	65	66	2210	1.3	65	2210	4.9
ex4*	128	28	279	1928	13.3	354	3276	3.7
ex5*	8	63	74	1122	65.5	128	784	9.2
rd53*	5	3	31	175	0.1	20	48	0.2
rd73*	7	3	127	903	0.4	54	191	0.7
rd84*	8	4	255	2070	1.6	90	399	2.8
spla*	16	46	252	3194	64.3	446	5291	29.4
t481*	16	1	481	5233	2.8	13	41	1.1
xor5*	5	1	16	96	0.0	5	6	0.0
apex6	135	99	656	4456	154.3	432	4012	3.7
apex7	49	37	434	3305	25.4	218	1874	1.6
ALU2	10	6	143	1029	1.8	111	641	1.8
ALU4	14	8	608	5444	21.7	565	4625	8.3
b1	3	4	4	18	0.0	6	14	0.0
b9	41	21	106	512	1.9	81	509	0.8
c8	28	18	79	333	0.7	55	177	0.5
cmb	16	4	17	74	0.3	4	39	0.2
count	35	16	169	793	1.5	54	292	0.6
dalu	75	16	2076	24972	74.6	1754	19108	49.7
frg1	28	3	119	914	0.9	127	1405	1.8
frg2	143	139	**	**	**	1180	11997	43.6
i1	25	16	28	129	0.2	22	81	0.3
i6	138	67	202	919	3.6	145	498	2.1
i8	133	81	951	7966	41.9	1224	9997	12.4
i9	88	63	1279	8523	50.0	1287	8189	14.6
parity	16	1	**	**	**	16	17	0.1
ttt2	24	21	124	700	1.6	65	429	0.5
vda	17	39	93	1442	4.5	197	1183	4.9
x2	10	7	17	84	0.2	20	78	0.5
x3	135	99	656	4458	188.6	419	4131	10.2
x4	94	71	520	3062	21.6	321	1913	2.5

* circuits in PLA format ** not available after 15 minutes of *cpu* time

TABLE II

Some combinational circuits with large number of product terms and literals

Circuit			XORDD-based synthesis			
	in	out	terms	literals	cpu	nodes
des	256	245	9828	60103	46.4	3883
i3	132	6	212643	4194303	2.1	138
i4	192	6	212642	7191492	7.8	192
my_adder	33	17	262158	4128803	4.0	579
C432	36	7	1.2e6	3.0e7	116.5	1280

been implemented. With some modifications, our algorithm can be extended to optimize multi-level AND/XOR networks. This is because a *Shared Multiple Rooted* XORDD is a multi-level representation of functions. An algorithm is currently being developed to minimize multi-level AND/XOR networks.

References

[1] R.Brayton, R.Rudell, A.Sangiovanni-Vincentelli, and A.Wang, "MIS: A Multi-level Logic Optimization System," *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, No. 6, November 1987, pp. 1062-1081.

[2] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, Vol. C-35, No. 8, August 1986, pp. 677-691.

[3] R.Drechsler, M.Theobald, and B.Becker, "Fast OFDD based Minimization of Fixed Polarity Reed-Muller Expressions," *Proc. European Design Automation Conf.*, 1994, pp. 2-7.

[4] H.Fleisher, M.Tavel, and J.Yeager, "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms," *IEEE Trans. on Computers*, Vol. C-36, No.2, 1987, pp. 247-250.

[5] G. Hachtel, M.Lightner, R.Jacoby, C.Morrison, P.Moceyunas, and D. Bostik, "Bold: The Bould optimal Logic Design System," Hawaii Int. Symp. on Systems Sciences, 1988.

[6] S. Malik, A.R.Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams," *Proc. Intl. Conference on Computer-Aided Design.*, 1988, pp. 6-9.

[7] A.Sababi, and M.A.Perkowski, " Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 30-35.

[8] M.A.Perkowski, L.Csanky, A. Sarabi, and I.Schafer, "Fast Minimization of Mixed-Polarity AND/XOR Canonical Networks," *Proc. Intl. Conference on Computer Design*, 1992, pp. 33-36.

[9] R.Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued Minimization for PLA Optimization," *IEEE transactions on CAD/ICAS*, Vol. CAD-6, No. 5, September 1987, pp727-750.

[10] T.Sasao and P.Besslich, "On the Complexity of Mod-2 Sum PLA's," *IEEE Trans. on Computers*, Vol. 39, No. 2, February 1990, pp. 262-266.

[11] T.Sasao, "EXMIN2: A Simplification Algorithm for Exclusive-OR-SUM-of Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions," *IEEE Trans. on Computer-Aided Design*, Vol. 12, No. 5, May 1993, pp. 621-632.