A Graph-Based Synthesis Algorithm for AND/XOR Networks [1]

Yibin Ye Kaushik Roy

School of Electrical and Computer Engineering

Purdue University, West Lafayette, IN 47907-1285, USA

Abstract

In this paper, we introduce a *Shared Multiple Rooted* XOR-based Decomposition Diagram (XORDD) to represent functions with multiple outputs. Based on the XORDD representation, we develop a synthesis algorithm for general Exclusive Sum-of-Product forms (ESOP). 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. We have applied our synthesis techniques to a large set of benchmark circuits in both PLA and combinational formats. Results of the minimized ESOP forms obtained from our synthesis algorithm are also compared to the SOP forms generated by ESPRESSO. Among the 74 circuits we have experimented with, the minimized ESOP's have fewer product terms than those of SOP's in 39 circuits.

1 INTRODUCTION

In many applications, the AND/XOR realizations of digital circuits are very efficient. While powerful optimization tools for AND/OR based synthesis have been developed in the last few years, (such as ESPRESSO [11], MIS [1] and BOLD [6]), 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. Recently, Sasao [12] 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 graph-based minimization method for general ESOP's. We start from a decision diagram (e.g. BDD [2]) 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 and removing edges and by changing edge values rather than computing new sub-functions. Although our synthesis method is implemented for optimizing two-level AND/XOR networks, it is possible to be applied to multi-level AND/XOR networks with some modifications. This is because a *Shared Multiple Rooted* XORDD is multi-level representation of functions.

The rest of the paper is organized as follows. Section 2 introduces the XOR-Based Decomposition Diagrams. Section 3 presents a synthesis algorithm based on XORDD representation for minimizing ESOP's. Results of our synthesis algorithm on benchmark circuits are detailed in Sect. 4. In Sect. 5, we give the conclusions and discuss the further expansion of our algorithm to multilevel AND/XOR networks.

2 XOR-BASED DECOMPOSITION DIAGRAM

2.1 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

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

Decision Diagram, the Ordered Functional Decision Diagrams (OFDD), have been introduced [7, 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:

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

While the OBDD representation of a Boolean function is based on Eq. (1), the OFDD is based on Eq. (2) and (3). A more general decision diagram, the Ordered Kronecker Functional Decision Diagram (OKFDD), has been introduced recently in [4]. All three decomposition types are used in OKFDDs, and hence OKFDD is a combination of OBDD and OFDD. The size of OKFDDs is, in general, smaller than that of OBDDs and OFDDs.

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*,* 1 *or* 2*, which give the edge value* $\overline{x}_i$ *,* x_i*, or* 1*, 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. What distinguishes GDD from an arbitrary multi-level logic representation is the restriction 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.*

Note that XORDD is a decomposition diagram rather than a decision diagram. A non-terminal vertex in an XORDD can have one, two, or more children vertices. Outgoing edges from a vertex can have the same label (and hence, the same value). Fig. 1 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.

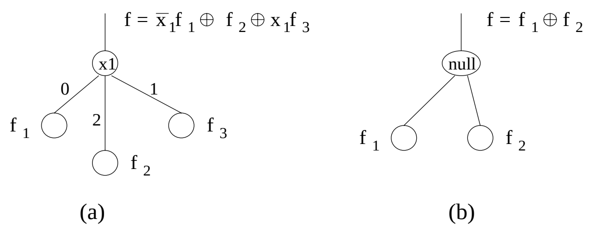

Figure 1: The XORDD representation of functions

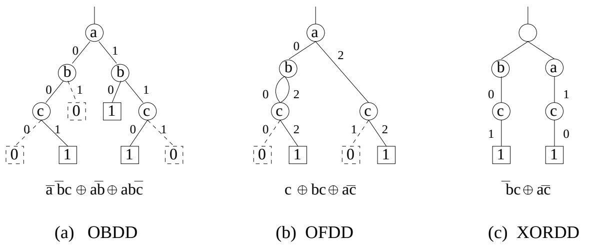

Figure 2: The same function represented by OBDD, OFDD and XORDD, respectively

OBDD, OFDD and OKFDD are special cases of XORDD based on our definition of XORDD. 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 1 *Let us consider the function* $f = a\overline{b} + a\overline{c} + \overline{b}c$*. Fig. 2 shows all three types of graph representations. Note that in the OFDD, both decomposition types of Eq. (2) and (3) have been applied, where label* 0 *denotes* f_i^0*,* 1 *denotes* 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 1 and Fig. 2 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 a sub-class of ESOP expressions, the canonical Reed-Muller form [10]. The XORDD representation, however, gives general ESOP form. Conversely, an ESOP expression can always be represented by an XORDD, and only Fixed Polarity Reed-Muller [3, 10] expressions can always 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

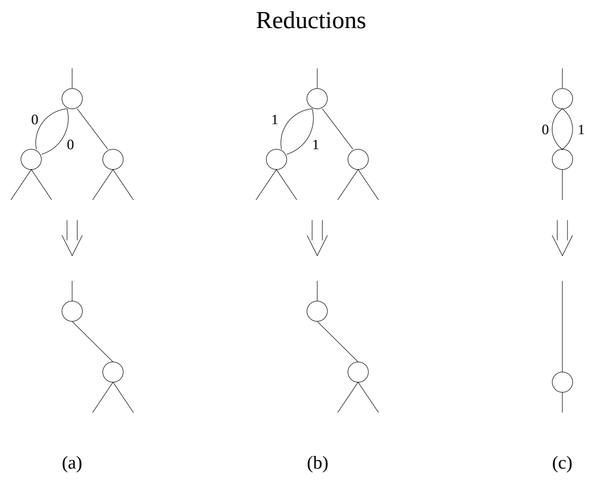

Figure 3: Three basic reduction rules of XORDD

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.*

2.2 Reductions on XORDD

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. 3(a) and 3(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. 3(c)).
4. Delete non-terminal vertices that do not have successors.
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 a Decision Diagram (OBDD, OFDD or OKFDD) which has two terminal vertices labeled with 0 and 1 respectively. Reduction rule (2) and (3) are illustrated in Fig. 3, which are based on the following facts:

$$x_i g_j \oplus x_i g_j = \overline{x}_i g_j \oplus \overline{x}_i g_j = 0 \tag{4}$$
$$x_i g_j \oplus \overline{x}_i g_j = g_j(x_i \oplus \overline{x}_i) = g_j \tag{5}$$

2.3 Transformations on XORDD

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 Eq.s (1) – (3) can be rewritten as follows:

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

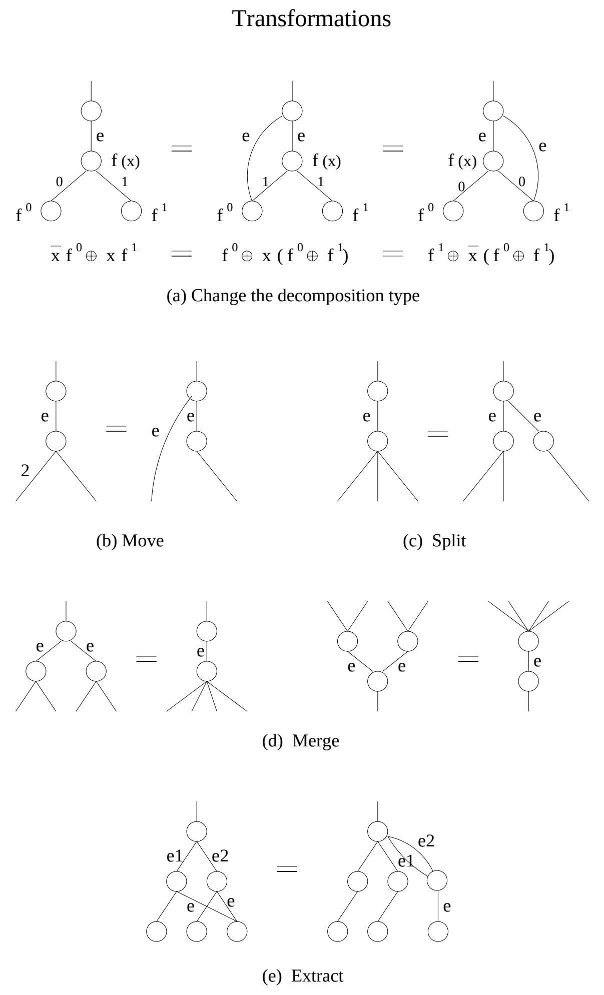
Transformations
e
f (x)
0
1
f^0
f^1
e
e
f (x)
1
1
f^0
f^1
e
e
f (x)
0
0
f^0
f^1
$\overline{x} f^0 \oplus x f^1 = f^0 \oplus x (f^0 \oplus f^1) = f^1 \oplus \overline{x} (f^0 \oplus f^1)$
(a) Change the decomposition type
e
2
e
e
(b) Move
e
e
e
(c) Split
e
e
e
e
e
e
(d) Merge
e1
e2
e
e
e2
e1
e
(e) Extract

Figure 4: Basic transformation steps of XORDD

where $f_i^0 \oplus f_i^1 = f_i^2$, which was defined in Sect. 2. 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. 4(a), where the label $e \in \{0, 1, 2\}$.

Move: An edge with label 2 (or value 1) can move to connect to parent nodes, as shown in Fig. 4(b)

Split: A vertex with multiple incoming or outgoing edges can be split into two or more vertices, as illustrated in Fig. 4(c).

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. 4(d).

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. 4(e). *Extract* is essentially a compound operation consisting of *split* and *merge*.

3 SYNTHESIS ALGORITHM FOR ESOP

Our synthesis strategy of minimizing ESOP's is as follows: Starting from an initial XORDD representation of a multiple output function (can be OBDD, OFDD or OKFDD), we iteratively apply transformations and reductions to finally obtain a *shared multiple rooted* XORDD which gives a minimized ESOP.

3.1 Algorithms for 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. One exception is that a common 1-path shared by multiple roots should be counted only once.

By examining the reductions and transformations introduced in Sect. 2.3, one can observe that reductions always reduce the objective functions (# of product terms and literals). However, transformations do not directly result in a more compact graph. While *merge*, *split*, and *extract* do not affect the objective functions, *change of decomposition* actually increases the value of objective functions. The purpose of performing transformations on an XORDD is to create opportunities for more reductions. 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. 5. In *loop type 1* (Fig. 5(a)), we can first *merge* two nodes and then apply reduction rule (3). In *loop type 2* (Fig. 5(b)), an edge label is changed first. We then *merge* two nodes followed by a reduction. In *loop type 3*(Fig. 5(c)), after changing an edge label, a reduction can be applied.

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

Step 1: Create a *Multiple Rooted Shared* OBDD (or OFDD, OKFDD) as the initial XORDD from the original two-level or multi-level circuit descriptions.
Step 2.1: Start from the lowest level vertices. Examine each node to determine if reductions can be directly applied.
Step 2.2: Detect three reducible loops discussed in Sect. 2.3 (also shown in Fig. 5). Change the label of some incoming edges (i.e., change decompositions), then apply reductions on parent node(s).
Step 3: For every vertex in the next lowest level, repeat the same transformations and reductions given in steps 2.1 and 2.2.
Step 4: Level by level from the bottom up to the root vertices, repeat the same operations given in steps 2.1 and 2.2.
Step 5: For each node, examine the incoming edges, do *merge* and *extract* as shown in Fig. 4 to change the topology of the graph. Then repeat step 2 – 4.

3.2 Global transformations

Note that only small loops are detected in step 2.2. Similar opportunities for reduction exist in larger loops. However, it is usually difficult to detect larger loops and further identify

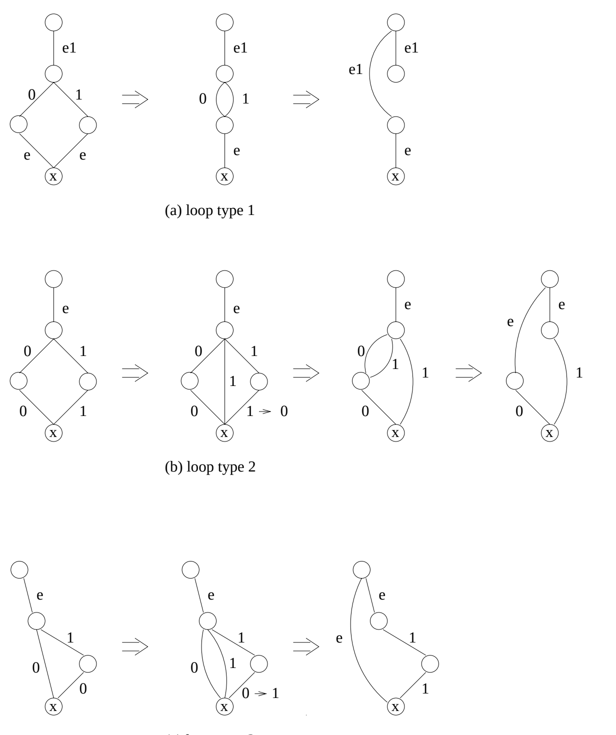

Figure 5: Three types of loops are identified, on which reductions can be applied after performing certain transformations.

which particular transformations should be applied. Potential reductions will be missed if only these small loops shown in Fig. 5 are considered. As an example, let us consider the loop shown in Fig. 6. The original loop in Fig. 6(a) will not be detected in the algorithm shown above. If we exchange the order of variable b and c, as shown in Fig. 6(b), we can merge two nodes of variable b. Now the loop shrinks and will be detected for reduction, as shown in Fig. 6(c). Therefore, if we allow variables move from one level to another, there are chances that large reducible loops will shrink, and become detectable. We refer these transformations that allow variables move as to "global transformations".

Step 6 – Global transformations
Step 6.1: Randomly select a variable, move all the nodes of this variable to the bottom.
Step 6.2: Repeat step 6.1 n times, where n depends on the number of primary inputs (in our implementation, n = *# of primary inputs*). Then apply local transformations (step 3–5) again.
Step 6.3: Repeat global transformations step 6.2 until very few reductions can be found.

Let us consider a function $f = ab\overline{c} + a\overline{b}c + dbc + d\overline{b}\overline{c}$ as an example. The minimal SOP representation 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 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 $1 \to 0$, 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,

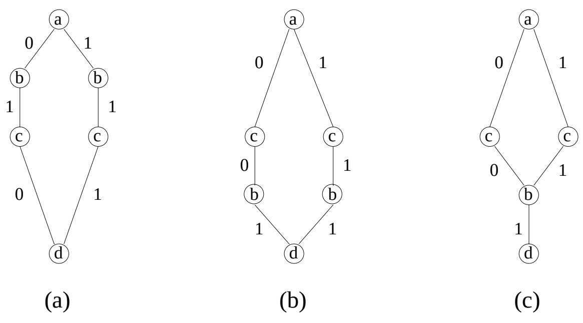

Figure 6: An example of global transformation. More reducible loops can be found by performing global transformations.

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.

4 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 table 1. 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 many previous work [9, 10], 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. Among the 74 circuits we have experimented with, the minimized ESOP's have fewer product terms than those of SOP's in 39 circuits. For some particular types of functions, one representation is better than the other. On average, the XORDD-based synthesis algorithm consumes 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 50% 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 1. As long as the initial decision diagram are successfully created, the XORDD-based method can always generate results. Table 2 listed several circuits for which

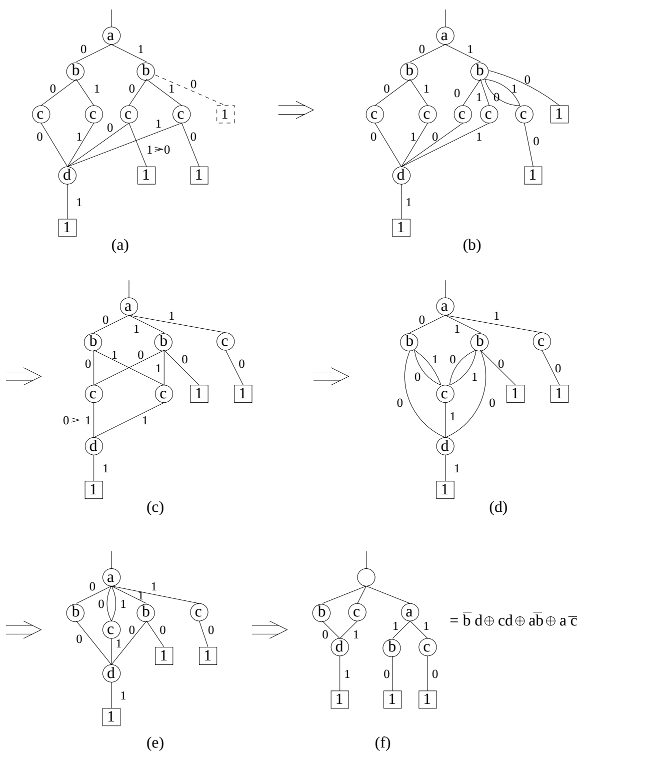

Figure 7: An example showing how the algorithm is performed on a function—-from initial BDD to the final compact XORDD, which gives a minimized ESOP form

ESPRESSO did not produce any result in 30 minutes on a SUN SPARC workstation. The last column *node* in Table 2 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 created XORDDs, however, the number of product terms and literals are excessively large. We feel that these circuits are inherently not suitable for two-level realizations.

5 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. The method successfully minimized large functions with multiple outputs.

In this paper, only two-level AND/XOR minimization has 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. This can be accomplished by using the number of literals as the objective function(not to be confused with the literal counts in two level ESOP's) and by relaxing the restriction that each variable can only be encountered once in a path from a root vertex to the terminal vertex in XORDD.

Table 1: Minimized ESOP's for benchmark circuits (in both PLA and combinational formats) 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	39	141	0.7
9sym*	9	1	86	602	0.6	60	423	0.9
apex1*	45	45	206	2842	5.7	496	4386	8.4
apex3*	54	50	281	3303	7.4	436	3295	17.5
apex5*	117	88	1088	7281	106.3	401	4002	7.3
b12*	15	9	43	207	0.6	29	143	0.6
cps*	24	109	163	2836	10.7	342	4128	11.0
e64*	65	65	66	2210	1.3	65	2210	6.3
ex4*	128	28	279	1928	13.3	345	3232	5.0
ex5*	8	63	74	1122	65.5	122	767	13.1
rd53*	5	3	31	175	0.1	15	48	0.2
rd73*	7	3	127	903	0.4	44	191	0.9
rd84*	8	4	255	2070	1.6	61	309	4.6
spla*	16	46	252	3194	64.3	423	5048	52.8
t481*	16	1	481	5233	2.8	13	41	1.2
xor5*	5	1	16	96	0.0	5	6	0.0
apex6	135	99	656	4456	154.3	416	3088	6.4
apex7	49	37	434	3305	25.4	209	1808	2.3
ALU2	10	6	143	1029	1.8	89	528	2.2
ALU4	14	8	608	5444	21.7	534	4421	10.4
b1	3	4	4	18	0.0	6	14	0.0
b9	41	21	106	512	1.9	80	507	1.0
c8	28	18	79	333	0.7	51	172	0.7
cmb	16	4	17	74	0.3	4	39	0.2
count	35	16	169	793	1.5	50	268	0.8
dalu	75	16	2076	24972	74.6	1742	18816	73.4
frg1	28	3	119	914	0.9	124	1392	2.7
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	143	493	3.2
i8	133	81	951	7966	41.9	1198	9974	22.4
i9	88	63	1279	8523	50.0	1287	8189	18.0
parity	16	1	**	**	**	16	17	0.1
ttt2	24	21	124	700	1.6	65	429	0.7
vda	17	39	93	1442	4.5	191	1123	6.7
x2	10	7	17	84	0.2	20	78	0.6
x3	135	99	656	4458	188.6	412	4062	12.0
x4	94	71	520	3062	21.6	307	1875	3.1

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

Table 2: Some combinational circuits with large number of product terms and literals

Circuit			XORDD based synthesis			
	in	out	terms	literals	cpu	nodes
des	256	245	9266	59782	49.5	3562
i3	132	6	202234	4076549	2.5	130
i4	192	6	212642	7191492	7.8	192
my_adder	33	17	36258	612348	8.1	65
C432	36	7	1.2e6	3.0e7	116.5	1280

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] R.Drechsler, A.Sarabi, M.Theobald, B.Becker, M.A.Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Ordered Kronecker Functional Decision Diagrams," *DAC, Proceedings of the Design Automation Conference*, 1994, pp. 415-419.

[5] 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.

[6] 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.

[7] U.Kebschull, E.Schubert, and W.Rosenstial, "Multilevel Logic Based on Functional Decision Diagrams," *Proc. European Design Automation Conf.*, 1992, pp. 43-47.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.