World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:2, 2008

Binary Decision Diagrams: An Improved Variable Ordering using Graph Representation of Boolean Functions

P.W. C. Prasad, A. Assi, A. Harb and V.C. Prasad

***Abstract*—** This paper presents an improved variable ordering method to obtain the minimum number of nodes in Reduced Ordered Binary Decision Diagrams (ROBDD). The proposed method uses the graph topology to find the best variable ordering. Therefore the input Boolean function is converted to a unidirectional graph. Three levels of graph parameters are used to increase the probability of having a good variable ordering. The initial level uses the total number of nodes (*NN*) in all the paths, the total number of paths (*NP*) and the maximum number of nodes among all paths (*MNNAP*). The second and third levels use two extra parameters: The shortest path among two variables (SP) and the sum of shortest path from one variable to all the other variables (*SSP*). A permutation of the graph parameters is performed at each level for each variable order and the number of nodes is recorded. Experimental results are promising; the proposed method is found to be more effective in finding the variable ordering for the majority of benchmark circuits.

***Keywords*—** Binary Decision Diagrams, Graph Representation, Boolean Functions Representation, Variable Ordering

I. Introduction

A large variety of problems in digital system design, combinational optimization and verification can be formulated in terms of operations on Boolean functions [1]. Algorithms that efficiently manipulate Boolean functions have become increasingly popular and important in Very Large Scale Integration (VLSI) and Computer Aided Design (CAD) applications. Many methods have been developed, such as truth tables, disjunctive normal form and Boolean formulas. Due to the ever increasing demand for better performance, most of these methods have been unable to produce efficient solutions for combinational problems in digital systems. This demand has encouraged the researchers to look for a better method to manipulate Boolean functions.

During the last two decades, Binary Decision Diagrams (BDDs) have achieved great popularity as data structures for representing Boolean functions in solving most of the combinational problems which arise in synthesis and verification of digital systems [1]-[3]. Despite the fact that the BDD is a relatively old technique, its advantages for canonical representation were recognized and emphasized by Bryant [2]. Significant breakthroughs in the optimization of digital circuits have been achieved using the two remarkable and important properties of BDDs. First, BDDs are canonical; this property is useful when verifying the equivalence between two circuits [3]. In fact, two circuits are equivalent if and only if their BDDs are identical for a specific variable ordering. Second, BDDs are effective structures for the representation of large combinatorial sets.

BDDs can be represented by algorithms that travel through all the nodes and edges of the directed graph in some order and therefore take polynomial time in the current size of the graph. However, when new BDDs are created, it might significantly increase the number of nodes in the BDD, depending on the node placement in the graph, which can lead to exponential memory and run time requirements [4]. The choice of BDD variable order has a direct impact on the size of the BDD, and determining an optimal variable ordering is an NP-hard problem [5].

The ROBDD representation is defined by imposing restrictions on the BDD specification of Akers [6] such that the resulting form is canonical. ROBDD gained widespread use in logic design verification, test generation, fault simulation and logic synthesis. Since the size of an ROBDD heavily depends on the variable order used, the researchers in this area are actively involved in finding a variable order that minimizes the number of nodes in the ROBDD. While the ROBDD is canonical for a given variable ordering, the number of nodes that form the ROBDD may vary dramatically from one order to the other. Accordingly, much attention has been devoted to techniques used to find optimal variable orderings. All these variable ordering techniques fall into two categories: mainly Static [7], [8] or Dynamic [9]-[11] variable ordering techniques.

Manuscript received February 12, 2006.

P. W. C. Prasad, is with the College of Information Technology, United Arab Emirates University, P.O. Box 17555, Al Ain, UAE (telephone: 971-3-7133051, e-mail: prasadc@uaeu.ac.ae,).

A. Assi is with the College of Engineering, Electrical Department, United Arab Emirates University, P.O. Box 17555, Al Ain, UAE (telephone: 971-3-7133609, e-mail: ali.assi@uaeu.ac.ae).

A. Harb is with the College of Engineering, Electrical Department, United Arab Emirates University, P.O. Box 17555, Al Ain, UAE (telephone: 971-3-7133606, e-mail: adnan.harb@uaeu.ac.ae).

V. C. Prasad is with Faculty of Engineering Technology, Multimedia University, Jalan Ayer Keroh Lama, 75450 Melaka, Malaysia (e-mail: vishnu.prasad@mmu.edu.my

Some applications have ROBDDs with different variable ordering, whereas further manipulations of these ROBDDs require identical variable ordering. For truth tables, this is not the case, since truth tables are explicit representations of each and every point of the function [12]. In a BDD, each path (from the root node to the terminal node) corresponds to a cube of the Boolean function and the order of variables defines the min-terms which will be simplified as cubes. Reduction of the number of paths would imply reduction in the number of cubes. This also can be done by finding an optimal order of the BDD variables [13].

In general, it is hard to predict the effect of variable ordering on the BDD size, this requiring the trial of all possible ordering schemes. It is also hard to find the best order for a given Boolean function [12], [14], [15]. However, there are some observations that help in finding a good variable ordering.

There are two main concerns in building the BDDs of Boolean functions: time and space requirements. For a given Boolean function, the primary requirement to minimize time and space complexity is to represent its BDD with minimum a number of nodes [14]-[16]. Usually, the number of nodes in a BDD is directly related to the variable ordering of the BDD [17]-[23]. There are a variety of methods to find the optimal variable ordering for BDDs but none can fulfill both the time and the space requirements.

The rest of this paper is organized as follows: In the second section, background information pertaining to the construction and implementation of BDDs is discussed. In section three, the new variable ordering technique based on three level permutations of graph parameters (3-LPGP) is introduced and the experimental results are given in section four. Finally in section five we conclude our paper with an outline of our future work.

II. Preliminaries

Basic definitions for binary decision diagrams are detailed in standard sources [2], [3], [6], [24]. However, we summarize the following definitions.

Definition 2.1: A BDD is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v has as label a variable $index(v) \in \{x_1, x_2, ..., x_n\}$ and two children $low(v), high(v) \in V$. A terminal vertex v is labeled with a value and has no outgoing edge: $value(v) \in \{0,1\}$

A BDD can be used to compute a Boolean function $f(x_1, x_2, ..., x_n)$ in the following way. Each input $a = (a_1, a_2, ..., a_n) \in \{0,1\}^n$ defines a computation path through the BDD that starts at the root. If the path reaches a non-terminal node v that is labeled by x_i, it follows the path $low(v)$ if $a_i = 0$, and it follows the path $high(v)$ if $a_i = 1$. The label of the terminal vertex determines the return value of the BDD on input a. In a more formal way, we can define recursively a Boolean function that corresponds to a BDD.

Definition 2.2: A BDD having root vertex v denotes a Boolean function f_v defined as follows:

1) If v is a terminal vertex and $value(v) = 1 (value(v) = 0)$, then $f_v = 1 (f_v = 0)$.

2) If v is a non-terminal vertex and $index(v) = x_i$, then f_v is given by

$$f_v(x_1, x_2, ...x_n) = .$$
$$\overline{x_i} \cdot f_{low(v)}(x_1, x_2, ...x_n) + x_i \cdot f_{high(v)}(x_1, x_2, ...x_n)$$

The variable x_i is called the *decision variable* for v.

Definition 2.3*: An ordered binary decision diagram (OBDD)* is a BDD with the constraint that the input variables are ordered in every source to sink path in the OBDD visits the input variables in ascending order.

Definition 2.4*: A reduced ordered binary decision diagram (ROBDD)* is an OBDD where each node represents a distinct logic function. It has the following two properties:

(i) There are no redundant nodes in which both of the two edges leaving the node point to the same next node are present within the graph. If such a node exists, it is removed and the incoming edges redirected to the following node.

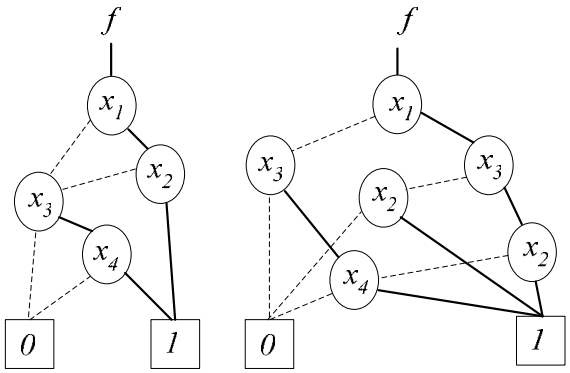
(ii) If two nodes point to two identical sub-graphs (i.e. isomorphic sub-graphs), then one sub-graph will be removed and the remaining one will be shared by the two nodes.

2.1 *Variable Ordering*

The size of a BDD is largely affected by the choice of the variable ordering. This is illustrated by the following example:

Example: Let $f = x_1 \cdot x_2 + ... + x_{2n-1} \cdot x_{2n}$. If the variable ordering is given by $(x_1, x_2, ..., x_n)$, i.e. $\pi(i) = x_i \forall_i$, the size of the resulting BDD is $2n$. On the other hand, if the variable ordering is chosen as $(x_1, x_3, ..., x_{2n-1}, x_2, x_{4,} ....x_{2n})$, the size of the BDD is $\theta(2^n)$. Thus, the number of nodes in the graph varies from linear to exponential, depending on the variable ordering chosen. Fig. 1 shows the effect of the variable ordering on the size of BDDs[2] for the following function (1):

$$f = x_1 \cdot x_2 + x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4 + \overline{x_1} \cdot x_3 \cdot x_4 \quad (1)$$



(a) $x_1x_2x_3x_4$ (b) $x_1x_3x_2x_4$

Fig.1 Effect of the variable ordering on the size of BDDs

III. Proposed Variable Ordering Algorithms

The proposed variable ordering algorithms use the graph topology to find a good variable order. In these algorithms the input Boolean function is converted into a unidirectional graph [25]. All Boolean operations in the Boolean function are converted into combinations of AND and NOT operations [26], which are represented as nodes in the graph. Each input and output of the Boolean function is also represented as a node in the graph.

3.1 *Level-I Algorithm (Node-Path Level)*

In this Level, the variable order is found based on the Number of Paths (*NP*), the Number of Nodes (*NN*) and the Maximum Number of Nodes among All Paths (*MNNAP*) per variable in the graph.

Since the variable with the highest number of nodes has the greatest effect on the circuit, it is placed first in the variable ordering. The proposed algorithm is explained in the following:

***Step 1*:** For every variable, we compute the *NP* from the corresponding input node to all output nodes.

***Step 2*:** For every variable, we compute the *NN* in every path and we note the *MNNAP*.

***Step 3*:** For every variable, we add the *NN* in all paths to get the total number of nodes *(TNN)*.

***Step 4*:** The variables are then sorted in descending order according to their *TNN*, then according to their *NP*, and *MNNAP* respectively. A heuristic selection is made if two or more variables have equal *TNN*, *NP* and *MNNAP*.

Example:

Consider the following Boolean function (2):

$$f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_4} + \overline{x_1} \cdot x_4 + x_2 \cdot \overline{x_4} + x_4 \cdot x_2 \cdot x_3 \quad (2)$$

This Boolean function that contains AND, OR and NOT operations, is first converted into an equivalent Boolean function (3) with only AND and NOT operations:

$$f = \overline{\overline{\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_4}}.\overline{\overline{x_1} \cdot x_4}.\overline{x_2 \cdot \overline{x_4}}.\overline{x_4 \cdot x_2 \cdot x_3}} \quad (3)$$

Fig. 2 shows the graph of the function *f* where each operation is represented as a node. Nodes x_1, x_2, x_3 and x_4 are the input nodes and node 13 is the output node. Nodes 1, 3, 4, 6, 7, 9, 11 and 13 represent NOT operations and nodes 2, 5, 8, 10, and 12 represent AND operations. In this graph, the *NP*, *TNN,* and *MNNAP* are given in Table 1.

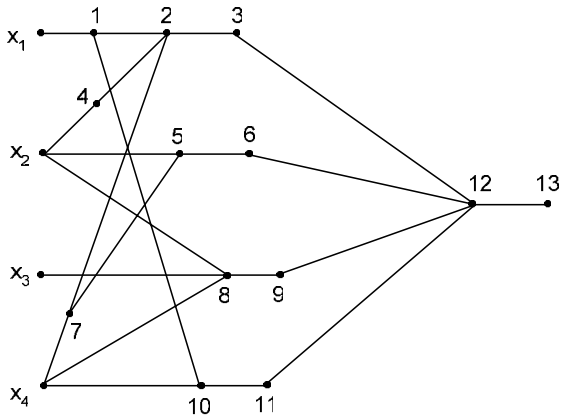


Fig. 2 Graph representation for function (3)

According to this table, the algorithm selects (x_4, x_2, x_1, x_3) as BDD variable ordering, which is the descending order of *TNN*. The algorithm will not compare *NN* or *MNNAP* since there is no equal *TNN*.

TABLE I
GRAPHICAL PARAMETERS FOR EQUATION (3)

Graphical Parameter	X_1	X_2	X_3	X_4
Number of different paths (*NP*)	2	3	1	4
Total number of nodes among all paths (*TNN*)	10	13	4	18
Max number of nodes among all paths (*MNNAP*)	5	5	4	5

3.2 *Level II Algorithm*

This level is an improved version of the Node-Path level algorithm explained in Section 3.1. It uses an additional parameter per variable that is the Sum of the Shortest Paths (*SSP*). The *SSP* of a variable is the sum of the shortest paths from this variable to all other variables. In this level, the variable order is found based on Primary (*NP*, *TNN and SSP*) and Secondary (*MNNAP*) parameters. In the following, we explain the proposed algorithm. Steps 1 to 3 are the same as algorithm-I and they are repeated here for convenience.

***Step 1*:** For every variable, we compute the *NP* from the corresponding input node to all output nodes.

***Step 2*:** For every variable, we compute the *NN* in every path and we note the *MNNAP*.

***Step 3*:** For every variable, we add the *NN* in all paths to get the *TNN*.
***Step 4*:** For every variable, we compute the *SSP*.
***Step 5*:** *NP*, *TNN*, and *SSP* are considered as primary parameters 1, 2, and 3. *MNNAP* is considered as secondary parameter 4.
***Step 6*:** The variables are then sorted in descending order according to parameter 1, then according to parameters 2, 3 and 4. A heuristic selection is made if two or more variables have equal parameters 1, 2, 3, and 4.
***Step 7*:** For the obtained variable ordering we note the number of nodes of the BDD.
***Step 8*:** We then repeat step 6 for the remaining permutations of parameters 1, 2 and 3. The variable ordering and the number of nodes are recorded for every permutation.
***Step 9*:** Among all permutations, the variable ordering with the minimum number of nodes is considered as the best variable ordering.

Example:

Consider the following Boolean function (4):

$$f = x_1 \cdot x_2 + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_3 \quad (4)$$

This Boolean function (4) is converted into an equivalent Boolean function, with only AND and NOT operations (5) as follows:

$$f = \overline{\overline{(x_1 \cdot x_2)}.\overline{(\overline{x_1} \cdot \overline{x_2} \cdot x_3)}.\overline{(\overline{x_1} \cdot x_3)}} \quad (5)$$

Fig. 3 shows the graph of the new function *f*. Nodes x_1, x_2 and x_3 are the input nodes and 10 is the output node.

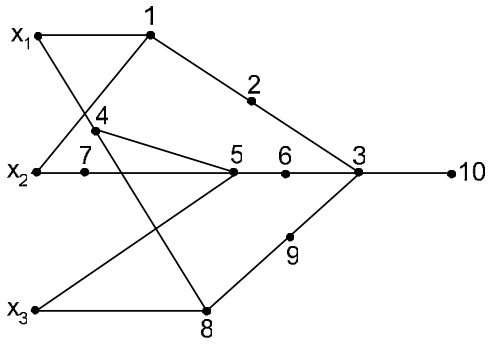


Fig. 3 Graph representation for function (4)

Nodes 1, 2, 3, 4, 5, 6, 7, 8, and 9 are intermediate nodes. From Fig. 3 the *NP*, *TNN*, *SSP* and *MNNAP* are calculated and summarized in Table 2. Table 3 presents the variable ordering obtained using algorithm II for various permutations of primary parameters.

TABLE II
GRAPHICAL PARAMETERS FOR EQUATION (5)

Parameters		X_1	X_2	X_3
Primary	*NP*	3	2	2
	TNN	14	9	8
	SSP	5	5	6
Secondary	*MNNAP*	5	5	4

TABLE III
VARIABLE ORDERINGS BY PERMUTATION

Parameter Sequence	Variables Ordering	Number of Nodes
NP, TNN, SSP, MNNAP	x_1, x_2, x_3	3
TNN, SSP, NP, MNNAP	x_1, x_2, x_3	3
SSP, NP,T NN, MNNAP	x_3, x_1, x_2	4
NP, SSP, TNN, MNNAP	x_1, x_3, x_2	3
SSP, NN, NP, MNNAP	x_3, x_1, x_2	4
NN, NP, SSP, MNNAP	x_1, x_2, x_3	3

The algorithm then selects the best variable ordering, i.e. x_1, x_2, x_3 or x_1, x_3, x_2 as the final variable order.

3.3 *Level III Algorithm*

This variable ordering algorithm is an improved version of the Parameter Permutation algorithm explained in Section 3.2. In addition to the four parameters *TNN*, *NP*, *SSP* and *MNNAP*, the Matrix algorithm uses one more parameter, which is the shortest path (*SP*) between every pair of variables. In the following we explain the proposed algorithm:
***Step 1*:** For every variable, we compute the *NP* from the corresponding input node to all output nodes.
***Step 2*:** For every variable, we compute the *NN* in every path and we note the *MNNAP*.
***Step 3*:** For every variable, we add the *NN* in all paths to get the *TNN*.
***Step 4*:** For every variable, we find the *SP* to all other input variables and the *SSP*.
***Step 5*:** *SP* is considered as primary parameter 1, and *NP*, *TNN*, *SSP* and *MNNAP* are considered as secondary parameters 2, 3, 4 and 5 respectively.
***Step 6*:** The variables are then sorted in descending order according to parameter 2, then according parameter 3, 4 and 5. A heuristic selection is made if two or more variables have equal parameters 2, 3, 4, and 5.
Step 7: The first variable in the sorted list is considered the first variable in the order.
***Step 8*:** We repeat steps 9 to 13 for $i = 2$ to n (where i is the variable sequence in the variables ordering and n is number of variables)
***Step 9*:** The variable with which the $(i\text{-}1)^{th}$ variable has the minimum *SP* value is ranked next in the variable ordering.

***Step 10*:** If, in step 9, two or more variables have an equal minimum value of parameter 1 (*SP*), then preference is given to the variable with the higher value of parameter 2, 3, 4, and 5 in order.

***Step 11*:** For the obtained variable ordering we note the number of nodes of the BDD.

***Step 12*:** We then repeat steps 6 to 11 for the remaining permutations of parameters 2, 3, 4 and 5 respectively. The variables ordering and the number of nodes are recorded for all permutations.

***Step 13*:** From all permutations, the variables ordering with the minimum number of nodes is considered as the best variable ordering.

Example:

Consider the following Boolean function (6):

$$f = x_1 \cdot x_2 \cdot \overline{x_4} + \overline{x_1} \cdot x_3 \cdot \overline{x_4} + \overline{x_1} \cdot x_4 + x_2 \cdot \overline{x_4} \quad (6)$$

This Boolean function (6) is converted into an equivalent Boolean function (7) with only AND and NOT operations. Fig. 4 shows the graph of the converted function.

$$f = \overline{(x_1 \cdot x_2 \cdot \overline{x_4}).\overline{(\overline{x_1} \cdot x_3 \cdot \overline{x_4}).\overline{(\overline{x_1} \cdot x_4).\overline{(x_2 \cdot \overline{x_4})}}}} \quad (7)$$

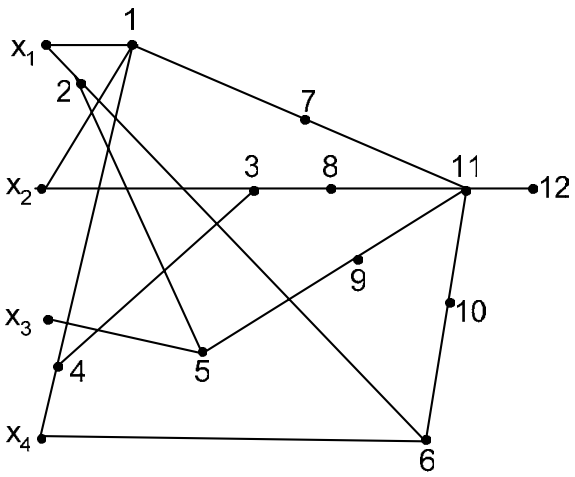


Fig. 4 Graph representation for function (6)

Table 4 summarizes the *SP* of this graph. *NP*, *NN*, *SSP*, and *MNNAP* are summarized in Table 5.

The number of nodes for all possible parameter permutations is recorded and x_4, x_2, x_3, x_1 is considered as the final variable ordering for level III method.

TABLE IV
SHORTEST PATH TABLE

	X_1	X_2	X_2	X_4
X_1	0	2	3	3
X_2	2	0	6	3
X_2	3	6	0	3
X_4	3	3	3	0

TABLE V
GRAPHICAL PARAMETERS FOR EQUATION (6)

Parameters	X_1	X_2	X_3	X_4
TNN	14	8	4	19
NP	3	2	1	4
MNNAP	5	4	4	5
SSP	8	11	12	9

The Permutation of the results from all three levels will produce the best possible variable ordering sequence for a given Boolean function.

IV. Experimental Results

In this section we present the experimental results obtained by applying the proposed three-level permutation method to selected ISCAS benchmark circuits using the Colorado University Decision Diagram (CUDD) Package [27]. A large collection of ISCAS benchmark circuits [28]-[30] has been selected to demonstrate the performance of the proposed method. Tables 6 and 7 summarize a comparison of our results to the best results obtained by three different CUDD variable reordering methods (Random Swapping, Symmetric Sifting and Window Permutation) for benchmark circuits with 20-139 inputs and 22-137 outputs.

In Table 6 column 1 shows the ISCAS benchmark name, and columns 2 and 3 show the size of the benchmark in term of inputs and outputs number. Columns 4 to 6 show the number of nodes required for the construction of the ROBDD using those three CUDD variable ordering methods. The results of the 3-LPGP method are shown in column 7, and the gain factors of the proposed method are given in columns 8, 9, and 10.

Table 7 shows a comparison with the result of the method that leads to the minimum number of nodes. Therefore it takes first seven columns of Table 6, along with four additional columns. Column 8 shows the minimum number of nodes among all four methods (i.e. the minimum number of nodes shown in columns 4, 5, 6, and 7 for a given benchmark circuit). The gain factors of each method against the method that gives the minimum numbers of nodes are given in columns 9, 10, 11 and 12.

TABLE VI
COMPARISON OF THE THREE-LEVEL PERMUTATION METHOD AGAINST THREE CUDD METHODS FOR ISCAS BENCHMARK CIRCUITS

Benchmark Circuits			Number of Nodes				Gain Factor of 3-LP GP		
			CUDD Methods			Proposed			
Name	Input	Output	Random swapping	Symmetric Sift	Window Permutation	Algorithms (3-LP GP)	Random swapping	Symmetric Sift	Window Permutation
5xp1	7	10	98	94	89	90	1.089	1.044	0.989
Alu2	11	6	247	199	251	198	1.247	1.005	1.268
Apex4	9	19	1410	1452	1583	1312	1.075	1.107	1.207
Cm42a	5	10	50	52	50	50	1.000	1.040	1.000
Cm85a	12	3	12	10	10	10	1.200	1.000	1.000
Con1	8	2	17	18	19	17	1.000	1.059	1.118
Cu	14	22	69	75	64	66	1.045	1.136	0.970
Misex1	8	7	81	87	78	73	1.110	1.192	1.068
Sao2	10	4	117	113	171	104	1.125	1.087	1.644
Sqrt8	8	4	33	30	34	33	1.000	0.909	1.030
Squar5	5	8	55	56	64	63	0.873	0.889	1.016
F51m	14	8	61	66	68	57	1.070	1.158	1.193
X2	10	7	53	60	68	53	1.000	1.132	1.283
Z4ml	7	4	32	36	45	32	1.000	1.125	1.406
B12	16	9	105	94	111	103	1.019	0.913	1.078
Apex6	143	139	1413	1056	2252	1028	1.37	1.03	2.19
Apex7	48	37	910	596	1672	678	1.34	0.88	2.47
Apex1	45	45	2850	2502	7232	2788	1.02	0.90	2.59
B9	42	21	261	225	331	244	1.07	0.92	1.36
Cc	21	20	126	124	123	100	1.26	1.24	1.23
Cht	47	36	255	217	271	190	1.34	1.14	1.43
Comp	32	3	393	294	8981	144	2.73	2.04	62.37
C8	28	18	136	139	136	136	1.00	1.02	1.00
C880	60	26	33817	15857	464256	9333	3.62	1.70	49.74
C432	36	7	3705	2470	13360	1933	1.92	1.28	6.91
C499	41	32	125352	60434	144556	63823	1.96	0.95	2.26
C1355	41	32	154789	64075	169382	101455	1.53	0.63	1.67
C1908	33	25	27560	18062	84541	23762	1.16	0.76	3.56
C3540	50	22	62500	44402	3129412	45903	1.36	0.97	68.17
Misex2	25	18	184	177	178	119	1.55	1.49	1.50
My_adder	33	17	856	681	66155	474	1.81	1.44	139.57
Mux	21	1	34	64	58	53	0.64	1.21	1.09
I1	26	13	82	74	83	71	1.15	1.04	1.17
I6	138	67	408	433	440	410	1.00	1.06	1.07
I7	199	67	510	535	666	512	1.00	1.04	1.30

The results shown in Table 6 indicate that the proposed 3-LPGP algorithm decreases the number of nodes in 94% of the circuits compared to Window Permutation, 93% compared to Random Swapping and almost 69% compared to Symmetric Sift.

TABLE VII
RESULTS COMPARISON AGAINST THE MINIMUM OF ALL FOUR METHODS FOR ISCAS BENCHMARK CIRCUITS

Benchmark Circuits			Number of Nodes								
			CUDD Methods			Proposed	Minimum	Gain Factor against Miminun			
Name	Input	Output	Random swappin	Symmetric Sift	Window Permutation	Algorithms (3-LP GP)	number of Nodes	Random swapping	Symmetric Sift	Window Permutatio	Proposed Algorithm
5xp1	7	10	98	94	89	90	89	0.91	0.95	1.00	0.99
Alu2	11	6	247	199	251	198	198	0.80	0.99	0.79	1.00
Apex4	9	19	1410	1452	1583	1312	1312	0.93	0.90	0.83	1.00
Con1	8	2	17	18	19	17	17	1.00	0.94	0.89	1.00
Cu	14	22	69	75	64	66	64	0.93	0.85	1.00	0.97
Misex1	8	7	81	87	78	73	73	0.90	0.84	0.94	1.00
Sao2	10	4	117	113	171	104	104	0.89	0.92	0.61	1.00
Sqrt8	8	4	33	30	34	33	30	0.91	1.00	0.88	0.91
Squar5	5	8	55	56	64	63	55	1.00	0.98	0.86	0.87
F51m	14	8	61	66	68	57	57	0.93	0.86	0.84	1.00
X2	10	7	53	60	68	53	53	1.00	0.88	0.78	1.00
Z4ml	7	4	32	36	45	32	32	1.00	0.89	0.71	1.00
B12	16	9	105	94	111	103	94	0.90	1.00	0.85	0.91
Apex6	143	139	1413	1056	2252	1028	1028	0.73	0.97	0.46	1.00
Apex7	48	37	910	596	1672	678	596	0.65	1.00	0.36	0.88
Apex1	45	45	2850	2502	7232	2788	2502	0.88	1.00	0.35	0.90
B9	42	21	261	225	331	244	225	0.86	1.00	0.68	0.92
Cc	21	20	126	124	123	100	100	0.79	0.81	0.81	1.00
Cht	47	36	255	217	271	190	190	0.75	0.88	0.70	1.00
Comp	32	3	393	294	8981	144	144	0.37	0.49	0.02	1.00
C8	28	18	136	139	136	136	136	1.00	0.98	1.00	1.00
C880	60	26	33817	15857	464256	9333	9333	0.28	0.59	0.02	1.00
C432	36	7	3705	2470	13360	1933	1933	0.52	0.78	0.14	1.00
C499	41	32	125352	60434	144556	63823	60434	0.48	1.00	0.42	0.95
C1355	41	32	154789	64075	169382	101455	64075	0.41	1.00	0.38	0.63
C1908	33	25	27560	18062	84541	23762	18062	0.66	1.00	0.21	0.76
C3540	50	22	62500	44402	3129412	45903	44402	0.71	1.00	0.01	0.97
Misex2	25	18	184	177	178	119	119	0.65	0.67	0.67	1.00
My_adder	33	17	856	681	66155	474	474	0.55	0.70	0.01	1.00
Mux	21	1	34	64	58	53	34	1.00	0.53	0.59	0.64
I1	26	13	82	74	83	71	71	0.87	0.96	0.86	1.00
I6	138	67	408	433	440	410	408	1.00	0.94	0.93	1.00
I7	199	67	510	535	666	512	510	1.00	0.95	0.77	1.00
Gain factor in %								24.23	27.28	9.09	60.61

The 3-LPGP method reaches its maximum gain for circuits Alu2, Apex4, apex6, Misex1, Sao2, F51m, cc, cht, comp, C880 and C432. Table 7 shows the superiority of the 3-LPGP method since it is able to produce the minimum number of nodes in almost 61% of the circuits (Alu2, Apex4, Con1, Misex1, Sao2, F51m, X2, Z4ml, Apex6, cc, cht, comp, c8, C880, C432, Misex2, My_adder, i1, i6 and i7) in comparison with 24%, 27% and 9% for Random Swapping, Symmetric Sift and Window Permutation respectively.

In general, it can be inferred that using the 3-LPGP method gives a higher probability of achieving the minimum number of nodes for medium and large scale benchmark circuits. The number of nodes in BDDs is directly related to the space complexity (i.e. area) of the circuit design.

So the above results prove that the proposed method minimizes the space complexity of the circuit, which will eventually minimize the cost of the design.

V. Conclusion

A three-level permutation of a graph parameter algorithm for minimizing the number of nodes in BDDs has been presented. The proposed 3-LPGP algorithm is capable of handling multiple output benchmark circuits. This algorithm has been implemented and tested using ISCAS benchmark circuits and the results have been compared with three selected CUDD reordering methods. The algorithm is deterministic in the sense that there is no heuristic involved in any of the primary parameters of the algorithm. Experimental results indicate that this algorithm is a promising alternative to existing reordering methods to reduce the number of nodes in BDD. Our on-going work will address. We will also analyze the cases when this method fails to produce near-best variable ordering. Our next step in this research work will be the use of SYNOPSYS tool to validate our results; especially the silicon area requirements of the benchmarks used to support and justify the proposed 3-LPGP algorithm.

References

[1] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. On Computers,* vol. 40, pp. 203–213, 1991.

[2] R. E. Bryant, "Graph−Based Algorithm for Boolean Function Manipulation," *IEEE Trans. Computers,* vol. 35, pp. 677-691, 1986.

[3] K. Priyank, "VLSI Logic Test, Validation and Verification, Properties & Applications of Binary Decision Diagrams," *Lecture Notes,* Department of Electrical and Computer Engineering University of Utah, Salt Lake City, UT 84112, 1997.

[4] F. Aloul, I. Markov, K. Sakallah, "MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation," *to appear in Journal of Universal Computer Science* (JUCS), 2005.

[5] Justin E. Harlow and Franc Brglez, "Design of Experiments and evaluation of BDD ordering Heuristics," *Inter. Journal on Software tools for Technology Transfer,* vol. 3, pp.193-206, 2001.

[6] S. B. Akers, "Binary Decision Diagram," *IEEE Trans. Computers,* vol. 27 pp. 509-516, 1978.

[7] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams," *in Proceedings of the International Conference on Computer Aided Design (ICCAD)*, 1988, pp. 2-5.

[8] Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," *in Proceedings of the International Conference on Computer Aided Design (ICCAD)*, 1988, pp. 6-9.
[9] S. Panda and F. Somenzi, "Who Are the Variables in Your Neighborhood," *in Proceedings of the International Conference on Computer Aided Design (ICCAD)*, 1995, pp. 74-77.
[10] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *in Proceedings of the International Conference on Computer Aided Design (ICCAD)*, 1993, pp. 42-47.
[11] F. Somenzi, "Efficient Manipulation of Decision Diagrams," *in International Journal on Software Tools for Technology Transfer (STTT),* vol. 3, pp.171-181, 2001.
[12] S. J. Friedman and K.J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *IEEE Trans. On Computers,* vol. 39, pp. 710–713, 1990.
[13] F. Görschwin, R. Drechsler, "Minimizing the Number of paths in BDDs," *in Proceedings of 15th symposium on Integrated Circuits and Systems Design*, 2002, pp. 359-364.
[14] R. Ebendt, "Reducing the number of variable movements in exact BDD minimization," *in Proceedings of 2003 Int. Symp. on Circuits and Systems,* 2003, pp. 605-608.
[15] R. Ebendt, W. Gűnther and R. Drechsler, "Combination of lower bounds in exact BDD minimization," *in Proceedings of Design Automation and Test in Europe Conf. and Exhibition,* 2003, pp. 758-763.
[16] R. Drechsler, N. Drechsler and.W.Günther, "Fast Exact Minimization of BDD's," *IEEE Trans. on CAD of IC and Systems*, vol.19, pp. 384–389, 2000.
[17] P. W. C. Prasad and A. K. Singh, "An Efficient Method for Minimization of Binary Decision Diagrams," *in Proceedings of 3rd Int. Conf. on Advances in Strategic Technologies,* 2003, pp. 683-688.
[18] K.S. Brace, R.L. Rudell and R.E. Bryant, "Efficient implementation of a BDD package," *in Proceedings of Design and Automation conf.*, 1990, pp. 40-45.
[19] Tani, K. Hamaguchi, and S. Yajima, "The Complexity of the Optimal Variable Ordering Problems of A Shared Binary Decision Diagram," *in Proceedings of 4th International Symposium on Algorithms and Computation* (ISAAC'93), LNCS 762, 1993.
[20] R. Rudell, "Dynamic Variable ordering for ordered binary decision diagrams," *in Proceedings of IEEE Inter. Conf. on CAD*, 1993, pp. 42-47.
[21] P. Chung, I. N. Haji and J. H. Patel, "Efficient Variable Ordering Heuristics for Shared ROBDDs," *in Proceedings of Int. Sym. on Circuits and Systems*, 1993, pp. 40-45.
[22] Y. Lu, J. Jain, E. Clarke and M. Fujita, "Efficient Variable Ordering using a BDD Based Sampling," *in Proceedings of 37th Design Automation Conf.,* 2000, pp. 687-692.
[23] M. G. Karpovsky, R. S. Stankovic, and J. T. Astola, "Reduction of Sizes of Decision Diagrams by Autocorrelation Functions," *IEEE Transaction on computer*, vol. 52, pp. 592-606, 2003.
[24] R. Drechsler and D. Sieling, "Binary Decision Diagrams in Theory and Practice," *Springer-Verlag Trans.*, pp.112-136, 2001.
[25] P. W. C. Prasad, M. Raseen and A. Assi, "Improved Variables Ordering for Binary Decision diagram," *in Proceedings of Int. Research Conf. on Innovation in Information Technology*, 2004, pp. 329-333.
[26] A. Kuehlmann, F. Krohm, "Equivalence checking using cuts and heaps," *in Proceedings of 34th Design Automation conference (DAC'97)*, 1997, pp.263-268.
[27] F. Somenzi, "CUDD: CU Decision Diagram Package," *ftp://vlsi.colorado.edu/ pub/.*, 2003.
[28] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," *Technical report*, Microelectronic Centre of North Caroline, Research Triangle Park, NC, January 1991.
[29] M. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Trans. On Design and Test*, vol. 16, pp. 72-80, 1999.
[30] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in Fortran," *in Proceedings of Int. Symposium on Circuit and Systems, Special Sess. On ATPG and Fault Simulation*, 1985, pp. 663-6985.