Journal of Computer Sciences 1 (4): 521-529, 2005
ISSN 1549-3636

BDD Path Length Minimization Based on Initial Variable Ordering

[1]P.W.C. Prasad, [1]M. Raseen, [2]A. Assi and [3]S.M.N.A. Senanayake
[1]College of Information Technology, United Arab Emirates University, Al Ain, UAE
[2]American University College of Technology, Department of Computer Engineering, Lebanon
[3]School of Engineering, Monash University, Malaysia Campus, Malaysia

Abstract: A large variety of problems in digital system design, combinational optimization and verification can be formulated in terms of operations performed on Boolean functions. The time complexity of Binary Decision Diagram (BDD) representing a Boolean function is directly related to the path length of that BDD. In this paper we present a method to generate a BDD with minimum path length. The Average Path Length (APL) and Longest Path Length (LPL) of the BDD are evaluated and discussed. The proposed method analyses the essentiality of a given variable order based on the complexity of sub functions derived from variable substitution. The variable that produces minimal cumulative complexity for the sub-functions is given priority over other variables. The experimental results and comparisons using benchmark circuits show that the proposed method is an encouraging approach towards minimizing the evaluation time of Boolean functions, consequently minimizing the time complexity of BDDs.

Key words: Binary decision diagram, boolean function, average path length, longest path length

INTRODUCTION

For the last two decades, BDD have gained great popularity in representing discrete functions. BDD in general is a direct acyclic graph representation of Boolean functions proposed by Bryant and Akers[1,2]. The success of this technique has attracted many researchers in the area of synthesis and verification of digital VLSI circuits. Because of its efficiency in representing a variety of practical functions[3,4], BDDs became very popular data structures. The efficiency of BDDs depends mainly on its size, which is the size of their graph representations. This size depends dramatically on the variable ordering adopted to build the BDD[5-7]. Finding an optimal variable order is often worth spending considerable computational efforts because this implies savings for further operations on the constructed BDD[8]. Some functions such as adders and depending on the variable ordering adopted, may lead to an exponential BDD sizes in terms of the number of input variables. Finding an optimal variable ordering is an NP-hard problem[9]. Another critical parameter during the construction of BDDs is the maximum memory requirement, which is directly proportional to the number of nodes. A good variable ordering can lead to a smaller BDD and faster runtime, whereas a bad ordering can lead to an exponential growth in the size of the BDD and hence can exceed the available memory[10]. Consequently, much attention has been devoted to techniques dedicated to finding good variable ordering. In general, variable ordering techniques fall into two categories: Static Variable Ordering (SVO) algorithms[11,12] and Dynamic Variable Ordering (DVO) algorithms[6,13].

The evaluation time is also another important parameter, which uses BDDs to evaluate logic functions. The evaluation time is proportional to the path length in the BDD. Therefore, minimization of the path length can improve the performance of the circuit implementing a Boolean function, which will eventually enhance the performance of the final implementation. In general the minimization of the path length in Decision Diagrams (DD) is important in database structures, pattern recognition, logic simulation and software synthesis[14]. The methods proposed for the minimization of APL[14-16] reduces the average evaluation time of logic functions. Most of these methods are based on either heuristic variable ordering or dynamic variable ordering techniques. The minimization of APLs leads to circuits with smaller depth of paths from the Root to the Terminal nodes of the BDD. The resulting circuit will be optimized for speed on one hand and on the other hand the number of very long paths in the BDD will be reduced[17]. The minimization of APLs is of great importance in real time operating system applications[18-20]. The minimization of the LPL of a BDD can reduce the longest evaluation time, which is very important for Pass Transistor Logic (PTL)[21-23]. One of the main problems with pass transistor networks is the presence of long paths: the delay of a chain of n pass transistors is proportional to n^2. Inserting buffers can reduce the

Corresponding Author: P.W.C. Prasad, College of Information Technology, United Arab Emirates University, P.O. Box 17555, Al Ain, U.A.E. Tel:971-3- 7133124, Fax: 971-3-7626309

path length, but this increases the silicon area. So the minimization of the longest evaluation time will improve the performance of the circuit[21,23]. In this study we propose an algorithm that minimizes the path length of BDDs. The resulting initial variable ordering will produce BDDs with the minimum possible APL and LPL, consequently reducing the number of nodes to an acceptable size. Hereafter, we introduce next the necessary terminologies and definitions, followed by the proposed method that computes the minimum APL and LPL of the BDD based on static variable ordering, experimental results and the conclusion of our study with an outline of our future work.

PRELIMINARIES

Basic definitions for BDDs are given in[1,2,24,25]. In the following we review some of these definitions.

Definition 1: A *BDD* is a directed acyclic graph (DAG). The graph has two sink nodes labeled 0 and 1 representing the Boolean functions 0 and 1. Each non-sink node is labeled with a Boolean variable v and has two out-edges labeled 1 (or *then*) and 0 (or *else*). Each non-sink node represents the Boolean function corresponding to its edge "1" if v = 1, or the Boolean function corresponding to its edge "0" if v = 0.

Definition 2: An *Ordered BDD* (OBDD) is a BDD in which each variable is encountered no more than once in any path and always in the same order along each path.

Definition 3: A *Reduced OBDD* (ROBDD) is an OBDD with only two reduction rules: *deletion rule* and *merging rule*. The Reduction rules remove redundancies within the OBDD.

Variable Ordering: The size of a BDD is largely affected and its variation can be linear or exponential depending on the choice of the variable ordering in building the BDD. Figure 1 illustrates the effect of the variable ordering[1] on the size of BDDs for the Boolean function (1):

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

Definition 4: In a BDD, a sequence of edge and nodes leading from the root node to a terminal node is called *Path*. The number of non-terminal nodes on the path is called the *Path Length*.

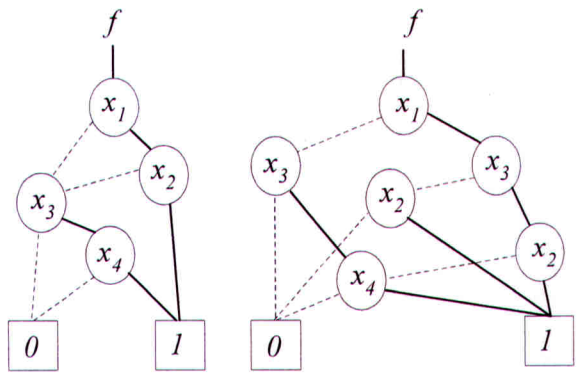

Fig. 1: Effect of the variable ordering on the size of BDD

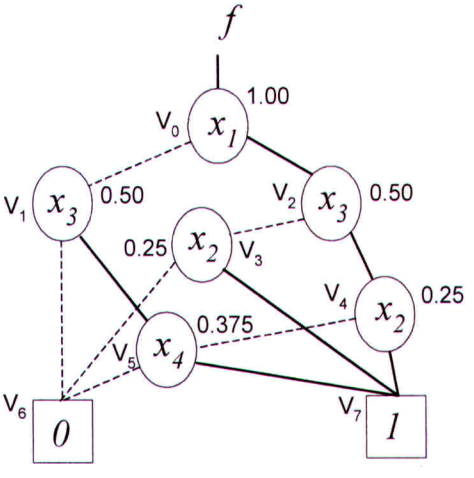

Fig. 2: Node Traversing Probability in a BDD

Definition 5: The *APL* is equal to the sum of the node traversing probabilities of the non-terminal nodes[14,18]. Node traversing probability denoted by $P(v_i)$ is the fraction of all 2^n assignments of values to the variables whose path includes node v_i. The APL can be expressed by the following equation (2):

$$APL = \sum_{i=0}^{N-1} P(v_i) \qquad (2)$$

Where, N is the number of non-terminal nodes.

Definition 6: The edge traversing probability, denoted by $P(e_{i0})$ (or $P(e_{i1})$), is the fraction of all 2^n assignments of values to variables whose path includes e_{i0} (or e_{i1}), where e_{i0} (or e_{i1}) denotes edge "0" (or the edge "1") directed from away node V_i [14]. Since all paths include the root node, this node is traversed with

probability 1.00. Since all assignments to values of variables are equally likely, we can use the following equation to calculate the $P(V_i)$ for the rest of the nodes:

$$\frac{P(vi)}{2} = P(e_{i0}) = P(e_{i1}) \quad (3)$$

Definition 7: *The Longest Path Length (LPL)* of a BDD denoted by LPL (BDD), is the *Length of the Longest Path.*

Example: Consider the BDD graph shown in Fig. 2. In this example we will compute the APL and the LPL:

The root node $P(V_0)$ is always equal to 1.00.

$P(V_1) = P(e_{0_0}) = 0.50$

$P(V_2) = P(e_{1_0}) = 0.50$.

$P(V_3) = P(e_{2_0}) = 0.25$

$P(V_4) = P(e_{2_1}) = 0.25$

$P(V_5) = P(e_{4_0}) + P(e_{1_1}) = 0.125 + 0.25 = 0.375$

Finally

$APL = \sum_{I=0}^{5} P(V_i) = 2.875$

$LPL = LongestPath = x_1 \rightarrow x_3 \rightarrow x_2 \rightarrow x_4 = 4$

Definition 8: In the DD of a logic function f, the *memory size of the DD,* denoted by *Mem (DD),* is the number of words needed to represent the DD in memory[18].

In a memory, each non-terminal node requires an index and pointers to the succeeding nodes. Since each non-terminal node in a BDD has two pointers, the memory size needed to represent a BDD is given by:

$$Mem(BDD) = (2+1) \times nodes(BDD) \quad (4)$$

PROPOSED METHOD

The proposed method is a static variable ordering technique[26,27], which uses the input Boolean expression to find the variable ordering used to minimize the APL and LPL. This is based on a single level Boolean function; hence, if a Boolean function is multilevel, then it is converted to a single level function prior to applying the proposed method. MVSIS (Multi Valued Logic Synthesis Tool) Version 1.0 is used for the conversion of Boolean functions from multilevel to single level. The proposed method consists of selecting the variable based on the complexity of sub-functions derived by assigning logic 1 and logic 0 to that variable. The variable that produces sub-functions with minimum complexity is given priority over other variables. The complexity of the sub-functions is evaluated based on the number of variables (*NV*), number of product terms (*NPT*) and the number of variable occurrences (*NVO*). Using this method we can produce the BDD graph with the shortest possible paths among all nodes including the terminal nodes, which will eventually reduce the path length. The complete steps of the proposed method are explained in the following algorithm:

Step 1: Note the number of inputs (*N*) of the Boolean function with input variables $x_1, x_2, x_3, \cdots, x_n$

Step 2: Set the variable counter (*M*) to 1

Step 3: Substitute logic 0 for variable X_M in the input function

Step 4: Simplify the resulting function using McCluskey's simplification method

Step 5: Note the number of variables (*NV*), the number of product terms (*NPT*) and the number of variable occurrences (*NVO*)

Step 6: Substitute logic 1 for variable X_M in the input function and repeat steps 4 and 5

Step 7: Note the total of *NV*, *NPT* and *NVO* obtained in steps 5 and 6

Step 8: Repeat steps 3 to 7 for $M = 2$ to n

Step 9: The variable that produces the least *NV* is selected as next variable in the order

Step 10: If two or more variables have the same *NV*, then selection is based on the least *NPT*

Step 11: If two or more variables have the same *NPT*, then selection is based on the least *NVO*

Step 12: If there are still two or more variables that have equal *NV*, *NPT* and *NVO*, then the first of these variables is selected

Step 13: The selected variable is then substituted for 0 and 1 in the input function and two sub-expressions are obtained

Step 14: Steps 1 to 12 are repeated for the two new sub-expressions till we get the second variable of the order

Step 15: The above steps are repeated for 2, 4, 8 ….., sub-expressions until we find all the variables of the order

For each of the iterations described above, we note that the number of expressions increases but the complexity of the expressions decreases. This decrease in complexity reduces the procedure to find the variables that come next in the variable order. The obtained variable order is used to build the BDD and compute the APL and LPL using Colorado University Decision Diagram (CUDD) package[28]. The following example illustrates the proposed algorithm.

Example: Consider the Boolean function $F = x_1 \cdot x_2 \cdot \overline{x_4} + x_2 \cdot \overline{x_3} \cdot x_4 + x_1 \cdot x_3$, with 4

Table 1: The parameters for *NV*, *NPT* and *NVO*

Variables	NV			NPT			NVO		
	Logic 0	Logic 1	Total	Logic 0	Logic 1	Total	Logic 0	Logic 1	Total
X_1	3	2	5	1	2	3	3	2	5
X_2	2	3	5	1	3	4	2	6	8
X_3	3	1	4	2	1	3	4	1	5
X_4	3	3	6	2	2	4	4	4	8

Table 2: The *NV*, *NPT* and *NVO* for the twelve sub functions

Variables		NV			NPT			NVO		
		Logic 0	Logic 1	Total	Logic 0	Logic 1	Total	Logic 0	Logic 1	Total
X_3=0		2	1		1	1		2	1	
	X_1			3			2			3
X_3=1		0	0		0	0		0	0	
X_3=0		0	2		0	2		0	2	
	X_2			4			4			4
X_3=1		1	1		1	1		1	1	
X_3=0		2	1		1	1		2	1	
	X_4			5			4			5
X_3=1		1	1		1	1		1	1	

variables x_1, x_2, x_3 and x_4. Substituting logic 0 and logic 1 for the four variables x_1, x_2, x_3 and x_4 we get the following sub-functions:

$$x_1 = 0 \Rightarrow x_2 \cdot \overline{x_3} \cdot x_4 \quad (5)$$

$$x_1 = 1 \Rightarrow x_2 \cdot \overline{x_4} + x_2 \cdot \overline{x_3} \cdot x_4 + x_3 \Rightarrow x_2 + x_3 \quad (6)$$

$$x_2 = 0 \Rightarrow x_1 \cdot x_3 \quad (7)$$

$$x_2 = 1 \Rightarrow x_1 \cdot \overline{x_4} + \overline{x_3} \cdot x_4 + x_1 \cdot x_3 \quad (8)$$

$$x_3 = 0 \Rightarrow x_1 \cdot x_2 \cdot \overline{x_4} + x_2 \cdot x_4 \Rightarrow x_1 \cdot x_2 + x_2 \cdot x_4 \quad (9)$$

$$x_3 = 1 \Rightarrow x_1 \cdot x_2 \cdot \overline{x_4} + x_1 \Rightarrow x_1 \quad (10)$$

$$x_4 = 0 \Rightarrow x_1 \cdot x_2 + x_1 \cdot x_3 \quad (11)$$

$$x_4 = 1 \Rightarrow x_2 \cdot \overline{x_3} + x_1 \cdot x_3 \quad (12)$$

The parameters *NV*, *NPT* and *NVO* for the above six sub-functions are shown in Table 1.

From Table 1, variable X_3 is selected as first variable in the order since it has the least total of *NV* compared to other variables. Two new sub-expressions (9) and (10) are obtained from substituting $x_3 = 0$ and $x_3 = 1$ respectively. From expressions (9) and (10) we re-start the procedure to find the second variable of the order. Substituting logic 0 and logic 1 for the three variables x_1, x_2 and x_4 in expressions (9) and (10) we obtain the following sub-functions:

$$x_3 = 0,\ x_1 = 0 \Rightarrow x_2 x_4 \qquad x_3 = 0,\ x_1 = 1 \Rightarrow x_2$$

$$x_3 = 1,\ x_1 = 0 \Rightarrow 0 \qquad x_3 = 1,\ x_1 = 1 \Rightarrow 1$$

$$x_3 = 0,\ x_2 = 0 \Rightarrow 0 \qquad x_3 = 0,\ x_2 = 1 \Rightarrow x_1 + x_4$$

$$x_3 = 1,\ x_2 = 0 \Rightarrow x_1 \qquad x_3 = 1,\ x_2 = 1 \Rightarrow x_1$$

$$x_3 = 0,\ x_4 = 0 \Rightarrow \cdot x_1 \cdot x_2 \qquad x_3 = 0,\ x_4 = 1 \Rightarrow x_2$$

$$x_3 = 1,\ x_4 = 0 \Rightarrow x_1 \qquad x_3 = 1,\ x_4 = 1 \Rightarrow x_1$$

Table 2 indicates the values of *NV*, *NPT* and *NVO* for the twelve sub functions. From Table 2 the variable x_1 is selected as next variable since it has the least number of total *NV*. It should be noted here that, so far in this example there was no need to check the parameters *NPT* and *NVO*, since *NV* was always the least. The first two variables are selected, hence, four sub-expressions are obtained from the substitution set (x_3, x_1) = (0,0), (0,1), (1,0) and (1,1). The 4 sub-functions can be used to find the third variable of the order. Performing the same procedure described above one can find all the variables in the order. In our example, after going through all required steps we obtain the variables order x_3, x_1, x_2, x_4. This variable order is used to build the BDD and find the APL and LPL.

RESULTS

Here we present two sets of results using the Colorado University Decision Diagram (CUDD) package[28], on a Pentium 4 machine with 512 MB RAMs. The first set represents the results derived from three of the best CUDD variable ordering techniques (Symmetric Sifting, Swapping and Window Permutation) and the proposed method. The second set of results is a comparison between the proposed method and two available APL methods. All experiments were performed using selected ISCAS benchmark circuits[29,30]. In Table 3, 5 and 6, a comparison with CUDD methods was performed for the parameters APL, Mem (BDD) and LPL respectively. Table 4 illustrates the results of the comparison with the latest available methods for the minimization of APL. In each of the Table 3-6 the first column shows a list of selected ISCAS benchmark circuits we have used to demonstrate the performance of the proposed method.

Average path length: In Table 3, columns 2, 4 and 6 illustrate the results obtained for three CUDD variable reordering methods, namely the swapping, symmetric sifting and window permutation in terms of number of nodes. Columns 3, 5 and 7 show the results in terms of APL. The results shown in columns 8 and 9 are from the implementation of our proposed method in terms of number of nodes and APL respectively. The minimum number of Nodes, the minimum APL and the number of nodes for the minimum APL are illustrated in columns 10, 11 and 12 respectively. The gain factors of each method against the minimum APL are given in columns 13, 14, 15 and 16.

The results obtained indicate the efficiency of the proposed method compared to other CUDD methods in term of APL and number of nodes. In general, the results obtained in column 16 indicate the effectiveness of the proposed method, where a minimum APL gain is obtained for 97% of the circuits, compared to 26, 18 and 24% achieved by the Random Swapping method, the Symmetric Sift method and the Window Permutation method respectively.

Even though all the circuits equally benefit from the proposed algorithm, in some cases like i6, i7, X4, X2, alu2, pm1, 5xp1, sao2, mux, cm150a, cm162a, cm163a and cm151a, the reduction is such as to make a dramatic difference in the processing of the circuits. The results indicate the potential of the proposed method, where it managed to minimize both the APL and the number of nodes for 52% of ISCAS benchmarks, i.e. i1, X2, apex4, b9, cm42a, decod, misex1, misex2, pm1, 9sym, z4ml, clip, majority, cm138a, mux, cm150a and cm163a.

Table 3: APL Results for selected benchmark circuits

Benchmark Circuits	CUDD Methods						Proposed Method		Minimum Number of nodes	Minimum APL	Number of Nodes for Minimum APL	APL Gain Factor against Minimum APL			
	Swapping		Symmetric Sift		Window Permutation										
	Nodes	APL	Nodes	APL	Nodes	APL	Nodes	APL				Swapping	Symmetric Sift	Window Permutation	Proposed Method
i1	72	23.37	71	22.37	77	22.37	70	21.27	70	21.27	70	0.91	0.95	0.95	1.00
i6	433	206.25	408	205.38	440	207.88	478	188.38	408	188.38	418	0.91	0.92	0.91	1.00
i7	535	233.13	510	228.13	666	265.25	535	199.25	510	199.25	535	0.85	0.87	0.75	1.00
X4	895	229.78	690	186.23	1199	233.10	720	173.60	690	173.60	720	0.76	0.93	0.74	1.00
X2	60	18.59	53	15.61	68	20.83	51	14.86	51	14.86	51	0.80	0.95	0.71	1.00
Apex4	1452	119.57	1410	119.65	1583	118.13	1384	112.02	1384	112.02	1384	0.94	0.94	0.95	1.00
Alu2	189	29.17	179	28.36	249	29.96	182	24.23	179	24.23	182	0.83	0.85	0.81	1.00
B9	245	69.51	196	65.63	275	69.10	190	61.42	190	61.42	190	0.88	0.94	0.89	1.00
cc	108	39.53	95	36.28	108	42.91	104	35.88	95	35.88	104	0.91	0.99	0.84	1.00
Cm42a	50	18.75	50	18.75	50	18.75	50	18.75	50	18.75	50	1.00	1.00	1.00	1.00
decod	96	31.00	96	31.00	96	31.00	96	31.00	96	31.00	96	1.00	1.00	1.00	1.00
Misex2	176	41.15	174	39.62	178	41.34	174	36.49	174	36.49	174	0.89	0.92	0.88	1.00
Pm1	78	29.23	77	27.10	81	26.29	74	22.18	74	22.18	74	0.76	0.82	0.84	1.00
Tcon	48	24.00	48	24.00	48	24.00	48	24.00	48	24.00	48	1.00	1.00	1.00	1.00
5xp1	88	36.81	79	34.44	89	37.44	100	31.23	79	31.23	100	0.85	0.91	0.83	1.00
9sym	25	7.34	25	7.34	25	7.34	25	7.34	25	7.34	25	1.00	1.00	1.00	1.00
Con1	18	6.44	16	6.31	18	6.31	21	6.04	16	6.04	21	0.94	0.96	0.96	1.00
Misex1	68	23.69	66	23.84	68	23.53	66	22.03	66	22.03	66	0.93	0.92	0.94	1.00
F51m	66	28.58	61	28.11	68	28.58	79	27.18	61	27.18	79	0.95	0.97	0.95	1.00
Z4ml	36	17.13	32	17.88	45	18.50	32	16.33	32	16.33	32	0.95	0.91	0.88	1.00
Sao2	113	11.47	117	14.20	171	22.31	121	10.59	113	10.59	121	0.92	0.75	0.47	1.00
Cm85a	41	9.70	41	8.22	41	7.72	41	7.72	41	7.72	41	0.80	0.94	1.00	1.00
Squar5	54	22.69	52	21.81	57	21.56	53	20.44	52	20.44	53	0.90	0.94	0.95	1.00
Rd84	54	24.15	54	24.15	54	24.15	54	24.15	54	24.15	54	1.00	1.00	1.00	1.00
clip	152	28.81	116	30.09	202	31.13	112	27.88	112	27.88	112	0.97	0.93	0.90	1.00
C17	10	5.50	11	5.00	13	5.50	11	5.50	10	5.00	11	0.91	1.00	0.91	0.91
Mm4a	962	64.21	550	60.47	2973	75.25	632	58.69	550	58.69	632	0.91	0.97	0.78	1.00
majority	8	2.56	8	3.81	8	3.81	8	2.56	8	2.56	8	1.00	0.67	0.67	1.00
B1	12	6.50	12	6.50	12	6.50	12	6.50	12	6.50	12	1.00	1.00	1.00	1.00
Cm151a	41	9.00	34	7.00	558	20.50	36	6.42	34	6.42	36	0.71	0.92	0.31	1.00
Cm138a	56	15.75	56	15.75	56	15.75	56	15.75	56	15.75	56	1.00	1.00	1.00	1.00
Rd53	24	13.00	24	13.00	24	13.00	24	13.00	24	13.00	24	1.00	1.00	1.00	1.00
B12	82	21.91	77	23.32	97	25.18	83	21.91	77	21.91	83	1.00	0.94	0.87	1.00
mux	36	6.00	33	5.50	2327	14.90	33	3.50	33	3.50	33	0.58	0.64	0.23	1.00
cm150a	78	7.50	33	5.50	332	18.16	32	3.50	32	3.50	32	0.47	0.64	0.19	1.00
inc	223	58.92	220	59.64	237	57.82	247	57.32	220	57.32	247	0.97	0.96	0.99	1.00
cm162a	49	14.64	48	12.82	62	16.20	54	11.70	48	11.70	54	0.80	0.91	0.72	1.00
cm163a	46	15.01	42	12.70	55	17.70	42	11.70	42	11.70	42	0.78	0.92	0.66	1.00
Gain Factor against Minimum APL in %												0.26	0.18	0.24	0.97

Table 4: Results Comparison with available method

Benchmark Circuits			Currently Available Methods				Proposed Method		APL Gain Factor over Available Methods	
			Paper 8		Paper 11					
Name	Input	Output	Nodes	APL	Nodes	APL	Nodes	APL	Paper 8	Paper 11
5xp1	7	10	91	31.31	79	31.28	90	31.23	1.003	1.002
Con1	7	2	16	6.06	16	5.94	18	6.04	1.004	0.984
Misex1	8	7	68	22.16	64	21.97	66	22.03	1.006	0.997
F51m	8	8	76	27.45	64	27.45	79	27.18	1.010	1.010
Z4ml	7	4	32	17.13	28	16.38	32	16.35	1.048	1.002
Sao2	10	4	128	10.71	121	10.59	121	10.59	1.011	1.000
Cm85a	11	3	47	8.28	38	7.72	41	7.72	1.073	1.000
Cm151a	12	2	36	6.50	32	6.00	36	6.42	1.012	0.935
B12	15	9	81	22.22	71	21.88	83	21.91	1.014	0.999
cm162a	14	5	59	11.7	48	11.71	54	11.7	1.000	1.001
cm163a	16	5	42	11.7	36	11.7	42	11.7	1.000	1.000
mux	21	1	33	3.5	32	3.5	33	3.5	1.000	1.000
cm150a	21	1	33	3.5	32	3.5	32	3.5	1.000	1.000

Table 5: Memory size requirement for selected benchmark circuits

Benchmark Circuits	Memory Size of BDD					Gain Facotr against minimum Mem(BDD)			
	Swapping	Symmetric Sift	Window Permutation	Proposed Method	Minimum Memory	Swapping	Symmetric Sift	Window Permutation	Proposed Method
i1	216	213	231	210	210	0.97	0.99	0.91	1.00
i6	1299	1224	1320	1254	1224	0.94	1.00	0.93	0.98
i7	1605	1530	1998	1605	1530	0.95	1.00	0.77	0.95
X4	2685	2070	3597	2160	2070	0.77	1.00	0.58	0.96
X2	180	159	204	153	153	0.85	0.96	0.75	1.00
Apex4	4356	4230	4749	4152	4152	0.95	0.98	0.87	1.00
Alu2	567	537	747	546	537	0.95	1.00	0.72	0.98
B9	735	588	825	570	570	0.78	0.97	0.69	1.00
cc	324	285	324	312	285	0.88	1.00	0.88	0.91
Cm42a	150	150	150	150	150	1.00	1.00	1.00	1.00
decod	288	288	288	288	288	1.00	1.00	1.00	1.00
Misex2	528	522	534	522	522	0.99	1.00	0.98	1.00
Pm1	234	231	243	222	222	0.95	0.96	0.91	1.00
Tcon	144	144	144	144	144	1.00	1.00	1.00	1.00
5xp1	264	237	267	300	237	0.90	1.00	0.89	0.79
9sym	75	75	75	75	75	1.00	1.00	1.00	1.00
Con1	54	48	54	63	48	0.89	1.00	0.89	0.76
Misex1	204	198	204	198	198	0.97	1.00	0.97	1.00
F51m	198	183	204	237	183	0.92	1.00	0.90	0.77
Z4ml	108	96	135	96	96	0.89	1.00	0.71	1.00
Sao2	339	351	513	363	339	1.00	0.97	0.66	0.93
Cm85a	123	123	123	123	123	1.00	1.00	1.00	1.00
Squar5	162	156	171	159	156	0.96	1.00	0.91	0.98
Rd84	162	162	162	162	162	1.00	1.00	1.00	1.00
clip	456	348	606	336	336	0.74	0.97	0.55	1.00
C17	30	33	39	33	30	1.00	0.91	0.77	0.91
Mm4a	2886	1650	8919	1896	1650	0.57	1.00	0.18	0.87
majority	24	24	24	24	24	1.00	1.00	1.00	1.00
B1	36	36	36	36	36	1.00	1.00	1.00	1.00
Cm151a	123	102	1674	108	102	0.83	1.00	0.06	0.94
Cm138a	168	168	168	168	168	1.00	1.00	1.00	1.00
Rd53	72	72	72	72	72	1.00	1.00	1.00	1.00
B12	246	231	291	249	231	0.94	1.00	0.79	0.93
mux	108	99	6981	99	99	0.92	1.00	0.01	1.00
cm150a	234	99	996	96	96	0.41	0.97	0.10	1.00
inc	669	660	711	741	660	0.99	1.00	0.93	0.89
cm162a	147	144	186	162	144	0.98	1.00	0.77	0.89
cm163a	138	126	165	126	126	0.91	1.00	0.76	1.00
Gain in %						0.32	0.76	0.26	0.58

Table 6: LPL results for selected benchmark circuits

Benchmark Circuits	CUDD Methods						Proposed Method		Minimum Number of Nodes	Minimum LPL	Number of Nodes for Minimum LPL	LPL Gain Factor against the Minimum LPL			
	Swapping		Symmetric Sift		Window Permutation							Swapping	Symmetric Sift	Window Permutation	Proposed Method
	Nodes	LPL	Nodes	LPL	Nodes	LPL	Nodes	LPL							
i1	72	57	71	56	77	57	70	56	70	56	70	0.98	1.00	0.98	1.00
i6	433	259	408	240	440	240	478	240	408	240	478	0.93	1.00	1.00	1.00
i7	535	281	510	271	666	303	535	239	510	239	535	0.85	0.88	0.79	1.00
X4	895	541	690	532	1199	554	720	532	690	532	720	0.98	1.00	0.96	1.00
X2	60	34	53	33	68	39	51	31	51	31	51	0.91	0.94	0.79	1.00
Apex4	1452	171	1410	171	1583	173	1384	168	1384	168	1384	0.98	0.98	0.97	1.00
Alu2	189	36	179	36	249	37	182	36	179	36	179	1.00	1.00	0.97	1.00
B9	245	150	196	150	275	153	190	147	190	147	190	0.98	0.98	0.96	1.00
cc	108	74	95	69	108	71	104	69	95	69	95	0.93	1.00	0.97	1.00
decod	96	80	96	80	96	80	96	80	96	80	96	1.00	1.00	1.00	1.00
Misex2	176	134	174	132	178	136	174	132	174	132	174	0.99	1.00	0.97	1.00
Pm1	78	63	77	63	81	63	74	63	74	63	74	1.00	1.00	1.00	1.00
5xp1	88	49	79	49	89	49	100	49	79	49	79	1.00	1.00	1.00	1.00
Con1	18	10	16	9	18	10	21	10	16	9	16	0.90	1.00	0.90	0.90
Misex1	68	34	66	36	68	34	66	34	66	34	66	1.00	0.94	1.00	1.00
F51m	66	36	61	34	68	36	79	36	61	34	61	0.94	1.00	0.94	0.94
Z4ml	36	22	33	22	45	22	32	21	32	21	32	0.95	0.95	0.95	1.00
Sao2	113	40	117	40	171	40	121	40	113	40	113	1.00	1.00	1.00	1.00
Cm85a	41	29	41	29	41	29	41	29	41	29	41	1.00	1.00	1.00	1.00
Squar5	54	34	52	32	57	34	53	32	52	32	52	0.94	1.00	0.94	1.00
clip	152	45	116	43	202	45	112	42	112	42	112	0.93	0.98	0.93	1.00
Cm151a	41	14	34	10	558	24	36	10	34	10	34	0.71	1.00	0.42	1.00
B12	82	54	77	53	97	54	83	51	77	51	83	0.94	0.96	0.94	1.00
mux	36	7	33	6	2327	18	33	5	33	5	33	0.71	0.83	0.28	1.00
cm150a	78	9	33	6	332	20	32	5	32	5	32	0.56	0.83	0.25	1.00
inc	223	106	220	106	237	108	247	107	220	106	220	1.00	1.00	0.98	0.99
cm162a	49	37	48	37	62	39	54	37	48	37	48	1.00	1.00	0.95	1.00
cm163a	46	31	42	31	55	34	42	31	42	31	42	1.00	1.00	0.91	1.00
cm42a	50	40	50	40	50	40	50	40	50	40	50	1.00	1.00	1.00	1.00
mm4a	962	138	550	136	2973	141	550	136	550	136	550	0.99	1.00	0.96	1.00
Gain Factor in %												0.36	0.66	0.26	0.90

Table 4 illustrates the benchmark results comparison with the previous work done in[14,16]. It can be inferred that the proposed method improves the APL in 100% of the benchmarks compared to [16], which uses three different algorithms, mainly bottom-up, top-down and middle-way, with the initial heuristic variable ordering. The proposed method was able to achieve improvement in the APL for 70% of the benchmarks compared to[14], which uses dynamic variable ordering technique. Benchmarks 5xp1, f51m, Z4ml, cm85a, cm162a, cm163a, mux and cm150a proved to be gaining the maximum from the proposed method. In general the proposed method provides far better results than the method in[16] and gives more competitive results than the method based on dynamic ordering in[14].

Memory size of BDD: For each BDD the memory size is computed using the equation (4) and tabulated in Table 5. In this Table columns 2, 3 and 4 illustrate the memory size obtained for the same three CUDD reordering methods that were used before. The memory size needed for the BDD construction using the proposed method and the minimum memory size resulting from the use of all four methods are given in columns 5 and 6. The gain factors of each method against the minimum memory size are given in columns 7, 8, 9 and 10.

It can be inferred that the proposed method managed to achieve the minimum memory size for 58% of the circuits against other reordering methods, which were mainly designed for the optimization of BDD size.

Longest path length: In Table 6, columns 2, 4 and 6 illustrate the results obtained for three CUDD variable reordering methods, namely the swapping, symmetric sifting and window permutation in terms of number of nodes and columns 3, 5 and 7 show the results for the same in terms of LPL. The results shown in columns 8 and 9 are from the implementation of our proposed method in terms of the number of nodes and the LPL respectively. The minimum LPL size out of all the methods is given in column 10 and the gain factors of the proposed method against the three CUDD methods are given in columns 10 to 12.

The obtained results indicate the efficiency of the proposed method compared to other CUDD methods in terms of minimum LPL. In general, the obtained results indicate that the proposed method managed to minimize

the LPL for more than 90% of the benchmarks compared to the efficiency of 37, 27 and 63% of Swapping, Window Permutation and Symmetric Sift reordering methods respectively. On the other hand, when the limitations of BDD nodes are set, the proposed method can achieve a maximum reduction in both the number of nodes and the LPL of 53% of the BDDs compared to 10, 13 and 56% for Swapping, Window Permutation and Symmetric Sift respectively. Benchmark circuits i1, i6, i7, X2, Apex4, B9, Z4ml, clip, B12, mux and cm150a lead to a maximum gain with the proposed algorithm compared to all three CUDD methods. Benchmark circuits X4, cc, squar5, misex2 and cm151a achieve better gain than two of the CUDD methods and circuits alu2, decod, 5xp1, sao2, cm85a, cm162a and cm163a lead to equally good results compared to CUDD methods.

In general, the proposed method gives a higher probability of achieving the minimum path length for most of the medium scale IASTED benchmark circuits. The minimization of the APL and the LPL leads to circuits with a smaller depth in the paths from Root to Terminal nodes. On the one hand this will lead to optimize the circuit for speed and on the other hand reduce the number of very long paths. Since the path length is directly related to the evaluation time of logic design, the above results prove that the proposed method minimizes the evaluation time and the space complexity of the circuit, which will eventually minimize the cost of the design.

CONCLUSION

A new algorithm for minimizing the evaluation time in BDD has been developed. The algorithm has been implemented using ISCAS benchmark circuits and the results have been compared with the three CUDD reordering methods and two of the available methods for path length minimization. Experimental results indicate that this algorithm is promising, yielding better results than more mature reordering techniques for most of the benchmark circuits. Even though most of the minimization methods use dynamic variable ordering technique, the above results proved that the static ordering techniques, too, could lead to some competitive results for path length minimization. It is also quite clear that the minimization of the evaluation time of BDDs can improve the performance of the circuit and have a strong influence on the quality of the final implementation. Our future work and developments will concentrate on investigating the APL and LPL minimizations for larger scale benchmark circuits.

ACKNOWLEDGEMENT

The authors would like to extend their thanks to Professor Nazar M. Zaki from the College of Information Technology at the United Arab Emirates University and to the English Department at the American University College of Technology for their valuable comments to improve the quality of this paper.

REFERENCES

1. Bryant, R.E., 1986. Graph-based algorithm for boolean function manipulation. IEEE Trans. Comp., 35: 677-691.
2. Akers, S. B., 1978. Binary decision diagram. IEEE Trans. Comp., 27: 509-516.
3. Priyank, K., 1997. 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.
4. Ingo, W., 1987. Complexity of Boolean Function. John Wiley & Sons Ltd and B. G. Teubner, Stuttgart.
5. Prasad P.W.C. and A.K. Singh, 2003. An efficient method for minimization of binary decision diagrams. Proceedings of 3rd Int. Conf. Advances in Strategic Technologies, pp: 683-688.
6. Rudell, R., 1993. Dynamic variable ordering for ordered binary decision diagrams. Proc. Intl. Conf. Computer Aided Design (ICCAD), pp: 42-47.
7. Ebendt, R., 2003. Reducing the number of variable movements in exact BDD minimization. Proc. 2003 Int. Symp. Circuits and Systems, pp: 605-608.
8. Aloul, F.A., I.L. Markov and K.A. Sakallah, 2000. Improving the efficiency of circuit-to-BDD conversion by gate and input ordering. 20th Intl. Conf. Comp. Design, pp: 64-69.
9. Harlow, J.E. and F. Brglez, 2001. Design of experiments and evaluation on of BDD ordering heuristics. Intl. J. Software Tools for Technol. Transfer., 3: 193-206.
10. Aloul, F., I. Markov and K. Sakallah, 2005. MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. To appear in Journal of Universal Computer Science (JUCS).
11. Fujita, M., H. Fujisawa and N. Kawato, 1988. Evaluation and improvements of boolean comparison method based on binary decision diagrams. Proc. Intl. Conf. Computer Aided Design (ICCAD), pp: 2-5.
12. Malik, S., A. Wang, R. Brayton and A. Sangiovanni-Vincentelli, 1988. Logic verification using binary decision diagrams in a logic synthesis environment. Proc. Intl. Conf. Comp. Aided Design (ICCAD), pp: 6-9.

13. Somenzi, F., 2001. Efficient manipulation of decision diagrams. Intl. J. Software Tools for Technol. Transfer, (STTT), 3: 171-181.
14. Nagayama, S., A. Mishchenko, T. Sasao and J.T. Butler, 2003. Minimization of average path length in BDDs by variable reordering. Intl. Workshop on Logic and Synthesis, pp: 207-213.
15. Ebendt, R., S. Hoehne, W. Guenther and R. Drechsler, 2004. Minimization of the expected path length in BDDs based on local changes. Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC'2004), pp: 866-871.
16. Liu, Y., K.H. Wang, T.T. Hwang and C.L. Liu, 2001. Binary decision diagrams with minimum expected path length. Proc. DATE 01, pp: 708-712.
17. Fey, G., J. Shi and R. Drechsler, 2004. BDD circuit optimization for path delay fault-testability. Proc. EUROMICRO Symp. Digital System Design, pp: 168-172.
18. Nagayama, S. and T. Sasao, 2004. On the minimization of longest path length for decision diagrams. Intl. Workshop on Logic and Synthesis (IWLS-2004), pp: 28-35.
19. Balarin, F., M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E.M. Sentovich and K. Suzuki, 1999. Synthesis of software programs for embedded control applications. IEEE Trans. CAD., 18: 834-849.
20. Lindgren, M., H. Hansson and H. Thane, 2000. Using measurements to derive the worst-case execution time. 7th Intl. Conf. Real-Time Systems and Appl. (RTCSA'00), pp: 15-22.
21. Nagayama, S. and T. Sasao, 2004. On the minimization of longest path length for decision diagrams. Intl. Workshop on Logic and Synthesis (IWLS-2004), pp: 28-35.
22. Shelar, R.S. and S.S. Sapatnekar, 2001. Recursive bipartitioning of BDD's for performance driven synthesis of pass transistor logic. Proc. IEEE/ACM ICCAD, pp: 449-452.
23. Bertacco, V., S. Minato, P. Verplaetse, L. Benini and G.D. Micheli, 1997. Decision diagrams and pass transistor logic synthesis. Stanford University CSL Technical Report, No. CSL-TR-97-748.
24. Drechsler, R. and B. Becker, 1998. Binary Decision Diagrams Theory and Implementation. Kluwer Academic Publishers.
25. Drechsler, R. and D. Sieling, 2001. Binary Decision Diagrams in Theory and Practice. Springer-Verlag Trans., pp: 112-136.
26. Prasad, P.W.C., A. Assi, M. Raseen and A. Harb, 2005. BDD minimization based on minimal cumulative sub-functions complexity. Proc. Intl. Conf. Research Trends in Science and Technology, Lebanon.
27. Prasad, P.W.C., M. Raseen and S. Sasikumaran, 2005. Delay minimization in pass transistor logic use of binary decision diagram. Accepted for Presentation in 2nd Intl. Conf. Inform. Technol., (ICIT 2005), Jordan.
28. Somenzi, F., 2003. CUDD: CU Decision Diagram Package. ftp://vlsi.colorado.edu/ pub/.
29. Yang, S., 1991. Logic Synthesis and Optimization Benchmarks User Guide Version 3.0. Technical report. Microelectronic Centre of North Carolina, Research Triangle Park, NC.
30. Hansen, M., H. Yalcin and J.P. Hayes, 1999. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. IEEE Intl. J. Design and Test, 16: 72-80.