International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

Selective min-terms based tabular method for BDD Manipulations

P.W. C. Prasad[1], A.Assi[2], M. Raseen[1] and A. Harb[3]
[1]United Arab Emirates University, College of Information Technology, U.A.E
[2]American University of Technology, Department of Computer Engineering, Lebanon
[3]United Arab Emirates University, Department of Electrical Engineering, U.A.E

Abstract**—** The goal of this work is to describe a new algorithm for finding the optimal variable order, number of nodes for any order and other ROBDD parameters, based on a tabular method. The tabular method makes use of a pre-built backend database table that stores the ROBDD size for selected combinations of min-terms. The user uses the backend table and the proposed algorithm to find the necessary ROBDD parameters, such as best variable order, number of nodes etc. Experimental results on benchmarks are given for this technique.

Keywords**—**Tabular Method, Binary Decision Diagram, BDD Manipulation, Boolean Function.

I. Introduction

The efficient approach for Boolean function manipulation is the key factor in many areas, i.e.: synthesis, design and testing of VLSI CAD circuits [1]. BDD are one of the most commonly used synthesis tool for logic optimization of digital systems [2]. Varieties of BDD are used in several commercial applications. However the size and complexity of functions some times make BDD beyond any acceptable limit in terms of space and memory requirement. Hence most of the applications are heavily relying on good variable ordering [3], [4]. Many heuristics have been suggested to minimize the space and memory complexity by providing a more effective initial variable ordering for BDD [5], [6], [7], [8], [9]. But the search for alternative solution still active in order to minimize the problems caused by the variable ordering to the BDD size.

A new algorithm is discussed to address the variable ordering problem. The proposed method is based on a table that includes a number of variables, a number of min-terms for BDD, and the corresponding BDD size. This paper is structured as follow: In section II we recall definitions pertaining to BDD. Proposed method with the algorithm is explained in section III followed by the experimental results for selective benchmark circuits in section IV. The section V summarizes the advantages of this approach and we conclude with section VI.

II. Preliminaries

Basic definitions for binary decision diagrams are detailed in [1], [2], [10], [11]. The following is a recall of 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 variables *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 1 edge if *v*=1, or the Boolean function corresponding to its 0 edge if v=0.

Definition 2: An 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 ordered binary decision diagram (ROBDD) is an OBDD where each node represents a distinct logic functions. 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 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.

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. Fig. 1 shows the effect of the variable ordering on the size of BDDs.

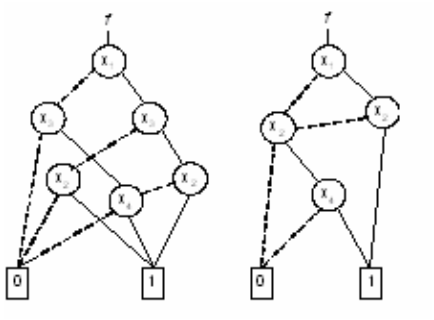

Figure 1: Effect of the variable ordering on the size of BDDs

III. Proposed Method

The proposed method is based on a table that acts as a backend database. The user application uses the details in the backend table to perform the necessary computations.

A. Backend BDD size table

The table is auto-generated by a program that utilizes the Colorado University Decision Diagram (CUDD) package in order to build the ROBDD. The program is automated and records the ROBDD size for selected groups of min-terms. The min-term groups are selected to cover all possible combinations of Boolean functions. For each of the selected min-term group, the ROBDD size is recorded for a fixed variable ordering of $x_1, x_2, x_3, \ldots, x_n$

For example with 3 variables and 6 min-terms the programs uses Boolean rules to generate only 28 ($_8C_6$) min-term groups to cover all possible 262144 (8^6) min-term combinations. The 28 min-term group will also cover the non min-term SOPs i.e. the non min-term SOP $x_1 \cdot x_2 + x_2 \cdot x_3$ will be covered by the min-term group $x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x_3} + \overline{x_1} \cdot x_2 \cdot x_3$.

The number of nodes for 3 variables with 6, 7 and 8 min-term groups is illustrated in table 1. Consider the first record in the table. Min-term group 0 1 2 3 4 5 is expressed in decimal, these decimal representations correspond to the actual min-terms $\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$, $\overline{x_1} \cdot \overline{x_2} \cdot x_3$, $\overline{x_1} \cdot x_2 \cdot \overline{x_3}$, $\overline{x_1} \cdot x_2 \cdot x_3$, $x_1 \cdot \overline{x_2} \cdot \overline{x_3}$ and $x_1 \cdot \overline{x_2} \cdot x_3$.

B. ROBDD size for the variable order $x_1, x_2, x_3, .., x_n$

Given a Boolean function that is a sum of min-terms the number of nodes for the order $x_1, x_2, x_3, .., x_n$ can be found from the backend table directly. For a Boolean function that is a SOP of non min-terms or combinations of min-terms and non min-terms, the function can be expanded and simplified to the min-terms form. Then the order can be found from the table. For circuits that are multi level, synthesis tools can be used to obtain a single level SOP. For example, a function with three variables and 6 min-terms (4 3 7 2 0 5), the number of nodes for variable order x_1, x_2, x_3 is obtained by arranging the min-terms in ascending order (0 2 3 4 5 7) and then comparing with the backend table. Consider the function $x_1 \cdot x_2 + x_2 \cdot x_3$, expanding the function will give min-term SOP as $x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x_3} + \overline{x_1} \cdot x_2 \cdot x_3$. This min-term group can be sorted in ascending order and the number of nodes can be found by checking in the backend table.

Table 1
BDD Size table for 3 variables with 6,7 and 8 min-terms.

Number of Minterms	Minterm Group	Number of Nodes	Number of Minterms	Minterm Group	Number of Nodes
6	012345	3	6	024567	3
6	012346	3	6	034567	4
6	012347	4	6	123456	5
6	012356	4	6	123457	5
6	012357	3	6	123467	5
6	012367	3	6	123567	3
6	012456	3	6	124567	4
6	012457	5	6	134567	3
6	012467	5	6	234567	3
6	012567	5	7	0123456	4
6	013456	5	7	0123457	4
6	013457	3	7	0123467	4
6	013467	5	7	0123567	4
6	013567	5	7	0124567	4
6	014567	3	7	0134567	4
6	023456	5	7	0234567	4
6	023457	5	7	1234567	4
6	023467	3	8	01234567	1
6	023567	5			

C. ROBDD Size for any variable order

For any Boolean function the min-term SOP form can be obtained. Using the min-terms of the resulting function, the ROBDD size for any order can be found as explained in the algorithm below:

Step 1: Express the function as SOP of min-terms;
Step 2: Convert all the min-terms into binary form;
Step 3: Create a new group of min-terms with the bits mapped from the old group according to the new variable ordering;
Step 4: Convert the new group to decimal and sort them in ascending order;
Step 5: Use the backend table to find the number of node for the new group of min-terms.

The number of nodes obtained for the new group will be the number of nodes for the original function but for the new variable ordering. The following example explains the procedure in details.

Example: Consider the function of three variables with min-terms groups 6 3 0 1 4 5. To find the number of nodes for the variable order of x_1, x_3, x_2, the min-terms are first written in binary form (110, 011, 000, 001, 100, 101). A new group of min-term is then generated from the existing group according to the variable order.

Since the variable order is x_1, x_3, x_2, the new group is generated as follow:

-- The first bit of the new group min-terms will be the same as first bit of old group min-terms, since the new variable order has x_1 in first position;
-- The second bit of the new group min-terms will be the third bits of old group min-terms, since the new variable order has x_3 in second position;
-- The third bit of the new group min-terms will be the second bit of old group min-terms, since the new variable order has x_2 in third positions.

The new min-term group will be 101, 011, 000, 010, 100, 110 (5, 3, 0, 2, 4, 6). Sorting the new group in ascending order we get 0 2 3 4 5 6. From the table, the number of nodes for the new sorted group is 5. This gives the number of nodes for the original function (6 3 0 1 4 5) but in the new variable order of x_1, x_3, x_2. This reflects the advantage of the proposed method.

D. Method to find the best variable ordering

Given a Boolean function, the best variable order can be found using the Backend min-term table that has been previously generated. The method to find the best variable order is explained in the following algorithm:

Step 1: The given Boolean function is converted into SOP of min-terms.
Step 2: The number of variables and the number of product terms for the resulting min-terms SOP is noted.
Step 3: In the backend table the group that has the same number of variables and same number of SOP terms is marked.
Step 4: In the marked group, the SOP terms that have the least number of nodes are selected.
Step 5: The bits in all the min-terms of the given function are shuffled to match one of the selected min-term groups. If the matching happens, then the shuffling order is the best variable order for the given function. If not the min-term group with next least number of nodes is selected and matched.
Step 6: Step 5 is repeated continuously for min-term SOP, with next least number of nodes, until a shuffle match is found.

The shuffle order for the marching group is the best variable ordering for the given function. The following example explains the algorithm in detail.

Example: Consider a function with 3 variables and 2 min-terms 3 and 6. Table 2 shows the number of nodes for the default order x_1, x_2, x_3 for 2 min-term groups. From table 2 the number of nodes for the given function in the variable order x_1, x_2, x_3 is 5. To find the best variable order, the table is scanned for 2 min-terms SOPs with least number of nodes. There are 11 SOPs (0 1, 0 2, 0 4, 1 3, 1 5, 2 3, 2 6, 3 7, 4 6, 5 7, 6 7) that have 2 as number of nodes. The bits of min-terms 3 6 are shuffled in the same order and compared with each of the 11 SOPs. It is realized that none of the 11 SOPs matches the after shuffling SOP; hence the group of SOPs with next least number of nodes is selected. There are 9 (0 3, 0 6, 1 2, 1 7, 2 4, 3 5, 4 5, 4 7, 5 6) that have 4 as number of nodes. The shuffling of bits is performed again. If we rearrange the bits of original function min-terms 3 6 (011 110) in the order of x_1, x_3, x_2 we get the min-terms 3 5 (011 101). The new min-terms 3 5 is one of 9 newly selected SOPs. Since the shuffling matches, the best variable ordering for the given function with min-terms 3 6 is x_1, x_3, x_2.

TABLE 2
BDD SIZE TABLE FOR 3 VARIABLES WITH 2 MIN-TERMS.

Number of Minterms	Minterm Group	Number of Nodes	Number of Minterms	Minterm Group	Number of Nodes	Number of Minterms	Minterm Group	Number of Nodes
2	0 1	2	2	1 5	2	2	3 6	5
	0 2	2		1 6	5		3 7	2
	0 3	4		1 7	4		4 5	4
	0 4	2		2 3	2		4 6	2
	0 5	5		2 4	4		4 7	4
	0 6	4		2 5	5		5 6	4
	0 7	5		2 6	2		5 7	2
	1 2	4		2 7	5		6 7	2
	1 3	2		3 4	5		-	-
	1 4	5		3 5	4		-	-

E. Functions with equal ROBDD complexity

One of the major applications of the backend table is the detection of equal complexities for different functions, with any number of variables. From table 1 and table 2 we can infer that a function with 3 variables and 6 min-terms 0 1 3 5 6 7 (5 nodes), has the same complexity as the function with 3 variables and 2 min-terms 1 4 (5 nodes too). From the tables and from the methods explained in section III-D we can find functions that have equal complexities for any variable order. Referring to last example, a function of three variables with min-terms 3 6 has 4 nodes for the variable order x_1, x_3, x_2, which has equal complexity with the function with min-terms 0 6 but with the different order x_1, x_2, x_3.

IV. EXPERIMENTAL RESULTS

The results presented in this section were observed using the Colorado University Decision Diagram (CUDD) package, on a Pentium IV machine with 512 MB RAM and measured by the number of nodes. Runtime entries refer to the time taken for reordering by the CUDD and finding the best order using our method. The results have been obtained as average of 50 executions of each benchmark circuits for three CUDD methods and the proposed method.

In Table 3, the first column provides to the names of the ISCAS benchmark circuits. Columns 2-9 illustrate the results obtained by three of the CUDD variable reordering methods (i.e. Reorder Random, Symmetric Sift and window coverage 2) and the proposed method. Columns 3, 5, 7, and 9 show the number of nodes required to construct the ROBDD for the benchmarks using the above reordering methods. Columns 2, 4, 6, and 8 show the CPU time in seconds..

The results indicate the efficiency of the proposed method compared to the selected CUDD ones in term of number of nodes, especially for circuits *C17*, alu2, cm42a, 5xp1, pm1. For all the benchmark circuits the run time for

proposed method decreases drastically compared with the three CUDD methods.

TABLE 3
RESULTS FOR SELECTED BENCHMARK CIRCUITS.

Benchmark Circuits	CUDD Methods						Proposed Method	
	Reorder Random		Symmetric Sift		Window Converge 2			
	CPU Time (sec.)	Number of Nodes	CPU Time (sec.)	Number of Nodes	CPU Time (sec.)	Number of Nodes	CPU Time (sec.)	Number of Nodes
squar5	0.0111	48	0.0124	48	0.0110	48	0.0084	48
rd84	0.0114	36	0.0113	36	0.0124	36	0.0092	36
rd53	0.0111	18	0.0112	18	0.0112	18	0.0091	18
majority	0.0110	6	0.0112	6	0.0123	6	0.0103	6
t481	0.0512	16	0.0117	16	0.0112	16	0.0110	16
z4ml	0.0120	11	0.0120	11	0.0354	11	0.0060	11
mm4a	0.0124	4	0.0363	4	0.0364	4	0.0148	4
sao2	0.0114	44	0.0352	44	0.0112	44	0.0094	44
c17	0.0111	11	0.0111	12	0.0111	11	0.0102	11
alu2	0.0195	247	0.0428	199	0.0176	251	0.0084	183
cm138a	0.0112	56	0.0113	56	0.0112	56	0.0079	56
cm42a	0.0112	50	0.0350	52	0.0113	50	0.0102	46
5xp1	0.0129	98	0.0130	94	0.0137	85	0.0082	76
pm1	0.0120	80	0.0124	79	0.0116	81	0.0111	79
clip	0.0113	50	0.0114	50	0.0124	50	0.0071	50
square5	0.0111	48	0.0348	48	0.0111	48	0.0101	48

V. ADVANTAGES

The proposed algorithms for ROBDD manipulation have the following advantages:

-- The backend table needs to be built only once and it can be used for any Boolean function and any algorithm.
-- The algorithm will be more effective compared to the traditional methods since the backend table is already built and ready. The time to build the backend table will not be a factor since it is not accounted for the algorithm execution.
-- Knowing the number of nodes for the default order, the proposed algorithm will enable to calculate the number of nodes for any other heuristic order, without building the ROBDD.
-- The best variable ordering for a given function can be found without checking all possible combinations of the variable order.

VI. CONCLUSION

In this work, we address the problem of variable ordering based on tabular method which consists of a table including the number of variables, the number of min-terms, and the size of the BDD. The algorithm will be more effective compared to the traditional methods since it uses a backend table that includes the results of all calculations. The experimental results prove the superiority of the new method over 3 selected CUDD methods in terms of number of nodes and calculation time. Our future development will be to apply the new method on more complex benchmark circuits.

ACKNOWLEDGMENTS

The authors would like to thank the financial support of the American University of Technology (AUT).

REFERENCES

[1] K. Priyank, “VLSI Logic Test, Validation and Verification, Properties & Applications of Binary Decision Diagrams,” Department of Electrical and Computer Engineering University of Utah, Salt Lake City, UT 84112.
[2] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. Computers*, Vol. 40, pp. 203–213, 1991.
[3] R. E. Bryant, "Graph–Based Algorithm for Boolean Function Manipulation," *IEEE Trans. Computers*, Vol. 35, pp. 677-691, 1986.
[4] S. Malik, A. Wang, R. Brayton, A. Sangiovanni,, “Logic Verification using Binary Decision Diagrams in Logic Synthesis Environment”. International Conference on Computer Aided Design, 1988.
[5] K.M. Butler, D.E. Ross, R. Kapur. and M. R. Mercer, “Heuristics to Compute Variable Ordering for Efficient Manipulations of Ordered Binary Decision Diagrams", DAC-90, pp. 52-57, 1990.
[6] P.W.C. Prasad and A. K. Singh, "Representation of Boolean Function using Partial Binary Decision Diagram," contribution talk, *5th International Congress on Industrial and Applied Mathematics*, Australia, 2003.
[7] P.W.C. Prasad, A. Assi, and M. Raseen, ”BDD Minimization Using Graph Parameter Permutation”, *The 2004 International Conference on VLSI*, 2004, pp. 491-494.
[8] P.W.C. Prasad, and A. K. Singh, "An Efficient Method for Minimization of Binary Decision Diagrams," *3rd International Conference on Advances in Strategic Technologies (ICAST)*, pp. 683-688, 2003..
[9] C. Yang and, M. Ciesielski, “BDS: A BDD–Based Logic Optimization System,” *IEEE Trans. On CAD of IC and Systems*, Vol.21, pp. 866–876, 2002.
[10] S. B. Akers, "Binary Decision Diagram," *IEEE Trans. Computers*, Vol. 27, pp. 509-516, 1978.
[11] K. Brace, “Efficient implementation of a BDD package,” in *Proceedings of Design and Automation conference*, pp 40-45, 1993.