Implementation of an Improved Initial Order in Various Dynamic Variable Ordering Techniques for BDDs

A dissertation submitted in partial fulfillment of the requirement for the award of degree of

Master of Technology

in

VLSI Design

Submitted by

Md. Balal Siddiqui

Roll No. 601161007

Under the supervision of

Ms. Manu Bansal

Assistant Professor, ECED

Thapar University, Patiala

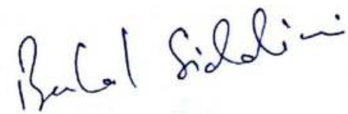
Department of Electronics & Communication Engineering

Thapar University, Patiala – 147004

DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled, **"Implementation of an Improved Initial Order in Various Dynamic Variable Ordering Techniques for BDDs"** in partial fulfilment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Manu Bansal, Assistant Professor, ECED and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University/Institute for the award of degree.



Date: 10-07-2013

Md Balal Siddiqui

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

(Ms. Manu Bansal) 10/7/13

Assistant Professor

ECED, Thapar University

Countersigned by:

Head,

ECED, Thapar University,

Patiala-147004

Dean of Academic Affairs

Thapar University,

Patiala- 147004

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to **Ms. Manu Bansal, Assistant Professor**, Electronics and Communication Engineering Department, Thapar University, Patiala for his patient guidance and support throughout this report. I am truly very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am also thankful to our head of the department, **Dr. Rajesh Khanna, Professor** as well as PG Coordinator, **Dr. Kulbir Singh, Assistant Professor**, Electronics and Communication Engineering Department. I would like to thank entire faculty and staff of Electronics and Communication Engineering Department and then friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.
Lastly, I would like to thank my parents for their years of unyielding love and encourage. They have always wanted the best for me and I admire their determination and sacrifice.

Md. Balal Siddiqui

ABSTRACT

The binary decision diagram is one of the important data structures having a very wide application including the area of logic synthesis, Boolean logic manipulation, verification and testing. Every Boolean function can be represented by the Binary decision diagrams. The number of nodes used in Binary Decision Diagrams gives the size, if node count is less, then it results in compact size, hence consumes less chip area. There are many rules for optimizing Binary Decision Diagrams including ordering of variables. These rules, when applied, results in Reduced Order Binary Decision Diagrams, in general this Reduced Order Binary Decision Diagrams are called BDD. The ordering of variables in BDDs has a great effect on the size of BDDs. There are many dynamic and static algorithms available to find the good variable ordering in BDDs. The dynamic algorithms start with a random initial ordering and then iteratively generate new ordering based on the approach. In this work an improved initial ordering is proposed and observed that the initial ordering has a big effect on the final size of BDDs in many dynamic approaches. This work is an implementation of an improved initial ordering which is applied on two dynamic ordering techniques available in BuDDy BDD package: sift and win2ite. The BDD size is calculated with the proposed initial ordering by using the two ordering methods for the different benchmark circuits from LGSynth93 benchmark suite. Also we have calculated the BDD size of different adder circuits with the proposed initial ordering. The result obtained with the proposed initial ordering is compared with the BDD with use of a random initial ordering in the different reordering techniques. The result obtained showed that there is a good improvement in the size of BDDs in many circuits with the proposed initial ordering. For some big circuits the improvement in BDD size is found to be more than 25 percents.

TABLE OF CONTENTS

Declaration			i
Acknowledgements			ii
Abstract			iii
Table Of Contents			iv
List Of Figures			vi
List Of Tables			viii
Chapter 1	**Introduction**		**1-3**
1.1	Motivation		1
1.2	Studies Organization		3
Chapter 2	**Binary Decision Diagrams**		**4-20**
2.1	Binary Decision Tree		4
2.2	Binary Decision Diagrams		5
	2.2.1	RBDD using Shannon expansion	10
	2.2.2	Ordering of variables in BDD	13
	2.2.3	Ordered BDD	14
2.3	Sift Algorithm for Ordering of Variables		15
2.4	BDD using MUX		17
2.5	BDD using Pass Transistor Logic		18
2.6	Application of BDD		19
2.8	BDD Packages		20
Chapter 3	**Literature Review on Algorithms for BDD Ordering**		**21-33**

Chapter 4	**BDD Using Improved Initial Order**	**34-28**
Chapter 5	**Results**	**39-44**
Chapter 6	**Conclusions & Future Scope**	**45**
	List of Publications	**46**
	References	**47**

LIST OF FIGURES

Figure 2.1	AND-OR circuit implementation for function f (a,b,c,d) = ab + cd.	4
Figure 2.2	Binary decision tree for function f (a,b,c,d) = ab + cd.	5
Figure 2.3	Removal of Duplicate Terminals	7
Figure 2.4	Removal of Duplicate non terminal	7
Figure 2.5	A pair of duplicate sub-BDDs in the intermediate of Figure 2.3	8
Figure 2.6	Intermediate BDD after applying rule R2 to the duplicate sub-BDDs	8
Figure 2.7	Nodes where both the outgoing edges point to the same node	9
Figure 2.8	Intermediate BDD when rule R3 is applied to node 'b'	9
Figure 2.9	Removal of Redundant Test	10
Figure 2.10	Primitive BDDs	11
Figure 2.11	BDD of a function of variable 'a'	11
Figure 2.12	Initial diagram of the function f(a) = (ac + bc + ab)	12
Figure 2.13	Diagram of the function f(a) = (ac + bc + ab) with redundant nodes	13
Figure 2.14	Final diagram of the function f(a) = (ac + bc + ab)	13
Figure 2.15	Reduction rules for Reduced Ordered BDDs	14
Figure 2.16	BDDs for function f = ab + a'c + bc'd with different ordering	15
Figure 2.17	Idea of the Sift Algorithm	16
Figure 2.18	Implementation of BDD using MUX	18

Figure 2.19	Implementing a BDD node in PTL	18
Figure 2.20	implementation of the function f = a' + bc' in PTL	19
Figure 3.1	Sliding window strategy	22
Figure 3.2	Variable Ordering Crossover Procedure	23
Figure 3.3	Framework of genetic tabu hybrid algorithm	28
Figure 3.4	Graph Representation for a function	30
Figure 3.5	The flow of the Genetic Programming algorithm	31
Figure 3.6	Branching scheme for B&B algorithm	32
Figure 4.1	Swapping Scheme for the proposed method.	37
Figure 4.2	Flow chart of the proposed method	38
Figure 5.1	Comparison of BDD size using both initial ordering in sift algorithm	41
Figure 5.2	Comparison of BDD size using both initial ordering in win2ite method	41
Figure 5.3	Comparison of BDD size of adder circuits using win2ite method	42
Figure 5.4	Comparison of average area of benchmark circuits using both the ordering method.	44

LIST OF TABLES

Table 2.1 Sift of variable x2 for different positions 16

Table 2.1 Truth Table of an example function 17

Table 4.1 Cost Matrix 35

Table 4.2 Minimum Cost Matrix 36

Table 5.1 BDD Size Comparison Using Sift Algorithm 39

Table 5.2 BDD Size Comparison Using win2ite Method 40

Table 5.3 BDD Size Comparison of Adder Circuits Using win2ite Method 40

Table 5.4 Area of a 2×1 MUX using Synopsys 43

Table 5.4 Area calculation of different benchmark circuits

ABBREVIATIONS

BDD	Binary Decision Diagram
OBDD	Ordered Binary Decision Diagram
RBDD	Reduced Binary Decision Diagram
ROBDD	Reduced Ordered Binary Decision Diagram
VLSI	Very Large Scale Integration
IC	Integrated Circuits
MUX	Multiplexer
BDT	Binary Decision Tree
PTL	Pass Transistor Logic
SOP	Sum Of Product
POS	Product Of Sum
B&B	Branch and Bound
CMOS	Complementary Metal Oxide Semiconductor

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION:

Today, in VLSI system design, to increase the overall efficiency of the device, it requires a designer to explicitly incorporate area and power reducing features within the system design. Different approaches exist for lowering the area and power. One is to reduce the operating voltage of the circuit, or to reduce the voltage supply and others includes minimization of switching functions represented by the system. These parameters are guided by a global cost function, which evaluates the current configuration with respect to user-specified objectives on area, delay, and power consumption. The current trend in VLSI design is mainly driven by two forces the growing demand for long-life autonomous portable equipment, and the technological limitations of high-performance VLSI systems. For the first category of products, area efficiency is one of the major goals which also results in an improvement in speed and power consumption. High speed and high integration density are the objectives for the second application category.

The output of every digital system is a switching function. There are many techniques used to efficiently minimize the switching functions and to make them area as well as more memory efficient. One of the techniques used in area minimization is representing a switching function by Binary Decision Diagrams (BDD) and then effectively reduce the size of the BDD which in turns reduce the area utilization of the corresponding switching functions. The BDDs can be directly mapped to a circuit based on multiplexors. So if realized with pass transistor logic, multiplexor cells can be used for synthesis at low cost. When a Boolean function represented by a BDD is implemented in pass transistor logic each internal node of the BDD is implemented by a MUX cell. Obviously, the size of the BDD has direct influence on the chip area of the derived BDD circuit. For this reason, it is important to find a BDD representation as small as possible to minimize chip area.

BDDs are first proposed by Lee [1] to represent the Boolean functions. In the last decades BDD has emerged as the most effective style of representation and manipulation of Boolean functions [3]. For Boolean functions having very large numbers of input variables, BDD uses very efficient memory when compare with other style of representation [2]. BDDs have found application in the field of synthesis, verification, test generation and fault simulation of digital systems [3, 4].

The BDD is a rooted, directed, acyclic graph having one root node, two terminal nodes and many internal nodes [5]. The number of internal nodes in BDD is known as size of the BDD. For PLA based design of the digital functions the size of the BDD represents the chip area. So to reduce area utilization of the function we have to reduce the size of the BDDs [6]. The size of a BDD strongly depends on the ordering of input variables. For a BDD have compact size with a good variable order may have an exponential increase in size when a bad order is chosen [7]. For example a BDD, which represent the carry-out of an adder, having size n for a good variable ordering may have size $2^{(n/2)}$ when some other ordering is chosen [8]. Many variable ordering methods have been proposed in the last decades. These methods include static and dynamic techniques. Static techniques are applied before constructing the BDD to generate an order depends upon the implemented function. Like the static technique used by Fujita which do a depth-first traversal through a circuit from the output to the inputs [9]. Even though static techniques are good over dynamic techniques, it requires the prior knowledge about function's behavioural, like effect of each input variables on function, which is not known in many cases. Dynamic techniques are focused on generating new variable orders to improve the size of the already constructed BDD. One of such technique is proposed by Ruddel is sifting algorithm [10]. It is the swapping of two variables. It first selects one variable, move it to each position and find the number of nodes. Then the variable is fixed at the position which gives the minimum number of nodes. Now the next variable is picked and moved to each position except the fixed positions by previous variables and then its position is fixed for minimum number of nodes. In this way all variables are shifted from its old position to its new position. The final ordering by fixing all variables is the improved ordering to find by sifting algorithm. This sift algorithm decreases the number of permutations of ordering from $n!$ to n^2 but in many cases the size of BDD is far from optimal. Many other methods use genetic algorithm [11-14] and different heuristics [15-16] to generate new ordering. The genetic

algorithm based approach used genetic operations like crossover and mutation to generate a new variable ordering based on some fitness criteria. Genetic algorithm bases approaches have good run time but when compare to other techniques, the result has not quite improved. The simulated annealing based approach is proposed in [17] which gives better result but on the cost of long run time. Variable ordering by scatter search is proposed in [15] which give better result with better run time in many cases but with a limited number of input variables. For circuits having large number of input variable scatter search based method is not satisfactory. Prasad had proposed a new graph based approach for ordering which use different graphical approaches and shows improvement on many previous works [18]. Although many methods are present to minimize the BDD size they still shows exponential variation in BDD size for many functions. In this paper a new method has proposed which use with other standard method gives improved result.

This dissertation is the implementation of an improved initial ordering of variables in the two dynamic ordering algorithms which finds out the improved order of variables for the minimum number of nodes. For that we have used the two different ordering methods available in the BDD suits (sift and win2ite) and given them a new improved initial order instead a simple initial order which they use. We have implemented our method on five different benchmark circuits and three different adder circuits. To perform the BDD operations we have used BuDDy library package designed for BDD manipulations.

1.2 ORGANIZATION OF THE DISSERTATION

This dissertation report is organized into four chapters.

Chapter 1 introduces the motivation for the proposed work and outlines the problem addressed.

Chapter 2 discusses the basics of BDD, ordering issues in BDDs and different implementation of BDDs.

Chapter 3 focuses on various algorithms for ordering of variables in BDDs available in the literature.

Chapter 4 is detail of the proposed method.

Chapter 5 discusses the result for proposed method.

Chapter 6 summarizes the dissertation and suggests future scope of this works.

CHAPTER 2

BINARY DECISION DIAGRAMS

This chapter introduces the basic principles of formation of a Binary Decision Diagram (BDD) for a Boolean function.

2.1 BINARY DECISION TREE

Binary decision tree (BDT) for a Boolean function is a tree such that

1) The internal nodes of a tree are labelled by variables of the function.
2) The leaves of a tree are labelled by 0 and 1.
3) Every internal node in a tree has exactly two children, the two arcs from the node to the children are labelled by 0 (dashed line) and by 1 (solid line). The dashed line indicates that the value assigned to the variable is 0 and the solid line indicates that the value assigned to the variable is 1.

For example, consider the following Boolean function f on four variables a, b, c and d.

f(a,b,c,d) = ab + cd. The AND-OR circuit implementation of this function is given in Figure 2.1.

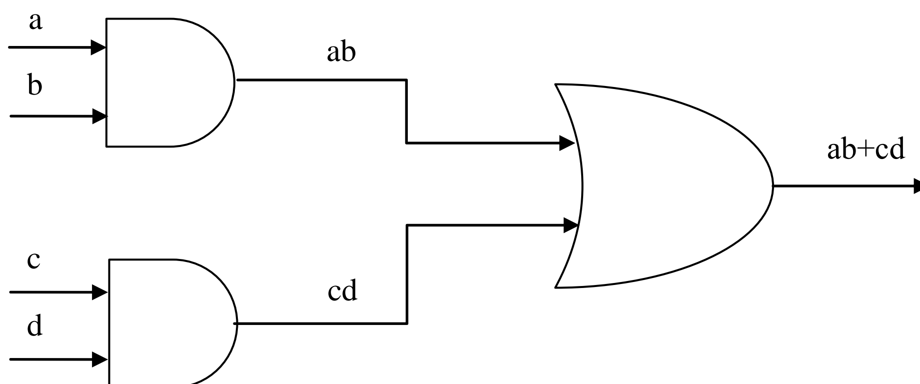


Figure 2.1 AND-OR circuit implementation for function f (a,b,c,d) = ab + cd

The above Boolean function contains 4 input variables a, b, c, d. If we represent this Boolean function by BDT then we will get 16 terminals (i.e., 2^4) (leaves) and 15 non terminals (internal nodes). The BDT for this Boolean function is shown in Figure 2.2.

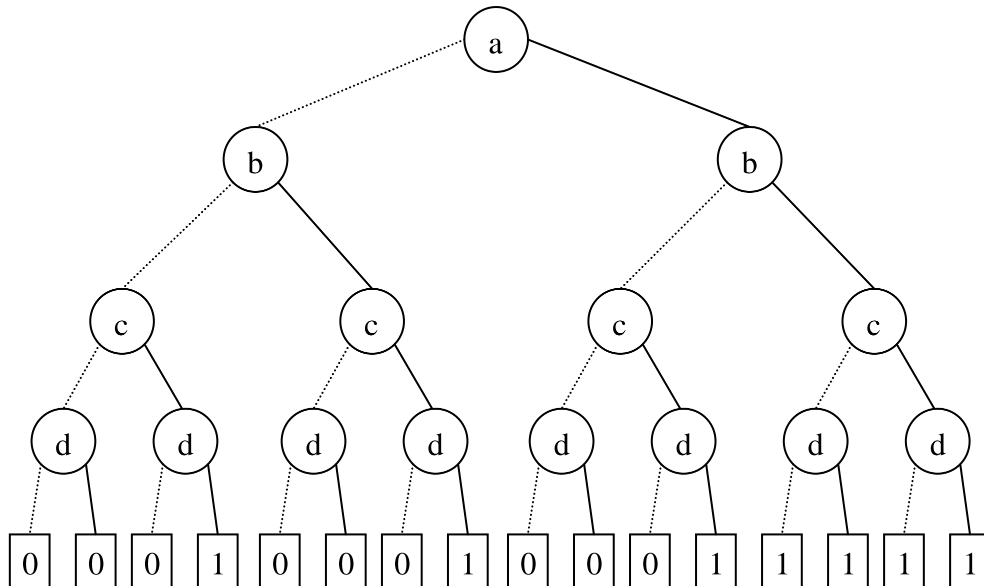


Figure 2.2 Binary decision tree for the function $f(a, b, c, d) = ab + cd$

From the function it may be noted that if a=1, b=1, c=d=0, then the function evaluates to '0'; this is represented by the path start from 'a', left edge to 'b', left edge to 'c', left edge to 'd' and left edge to the leaf node '0'. Also, if then the function evaluates to '1'; this is represented by the path start from 'a' , right edge to 'b' , right edge to 'c' , left edge to 'd' and left edge to leaf node '1'.

2.2 BINARY DECISION DIAGRAMS

A binary decision diagram is a directed acyclic graph in which every node represents some Boolean function. There are exactly two outgoing edges for every non-terminal node. There are only two terminal nodes representing the constants '1' and '0' [1].

From the example illustrated above, it may be noted that BDT cannot be generated within practical time lines for a reasonably complex circuit. The main reason is the exponential number of nodes in a BDT with respect to the number of inputs. However, it may be noted that there are many redundant nodes in the BDT; simply, in the leaf level

we can have one node with '0' and the other with '1' and redirect paths from the nodes of leaf level -1 accordingly. Similarly, this reduction is carried out at all layers till the root.

Just like BDT, BDD is an acyclic graph representation of a Boolean expression but unlike BDT, it does not comprise any redundant node either in the leaf or the non-leaf level. So BDD is called Reduced BDD.

The reduction from BDT to Reduced BDD is done using three rules

1) R1: Removing of duplicate terminals (leaf nodes): If a BDD contains more than one terminal 0-node, then we redirect all edges which point to such 0-node to just one of them and other 0-node can be removed. Similarly we do with 1-node also.
2) R2: Removal of duplicate non terminals (internal nodes): If two distinct nodes n and m in the RBDD are the roots of structurally identical sub-BDDs, then we eliminate one of the nodes, and redirect all its incoming edges to the other one.
3) R3: Removal of Redundant test: If both outgoing edges of a node n point to the same node m, then we eliminate the node n, and point all its incoming edges to m.

After application of rules R1, R2 and R3 repeatedly on BDT we get the reduced BDD. A BDD is said to be reduced if none of the above rules can be applied further i.e., no more reduction is possible.

Figure 2.3 illustrates the intermediate BDD when rule R1 is applied to the BDT of Figure 2.2. It may be noted that there is only one leaf node with '0' and another one with '1'. From Figure 2.2 it may be observed that the path from root to leaf node corresponding a=b=c=d=0 reaches a leaf node with '0'. Also the path from corresponding to a=b=c=0; d=1reaches a leaf node with '0'. Now as there is a single node at leaf level with '0' in the Reduced BDD as shown in Figure 2.3 both the paths corresponding to a=b=c=d=0 and a=b=c=0; d=1 has been redirected the single '0' leaf node. Similarly, redirection of all paths in Figure 2.3 can be explained.

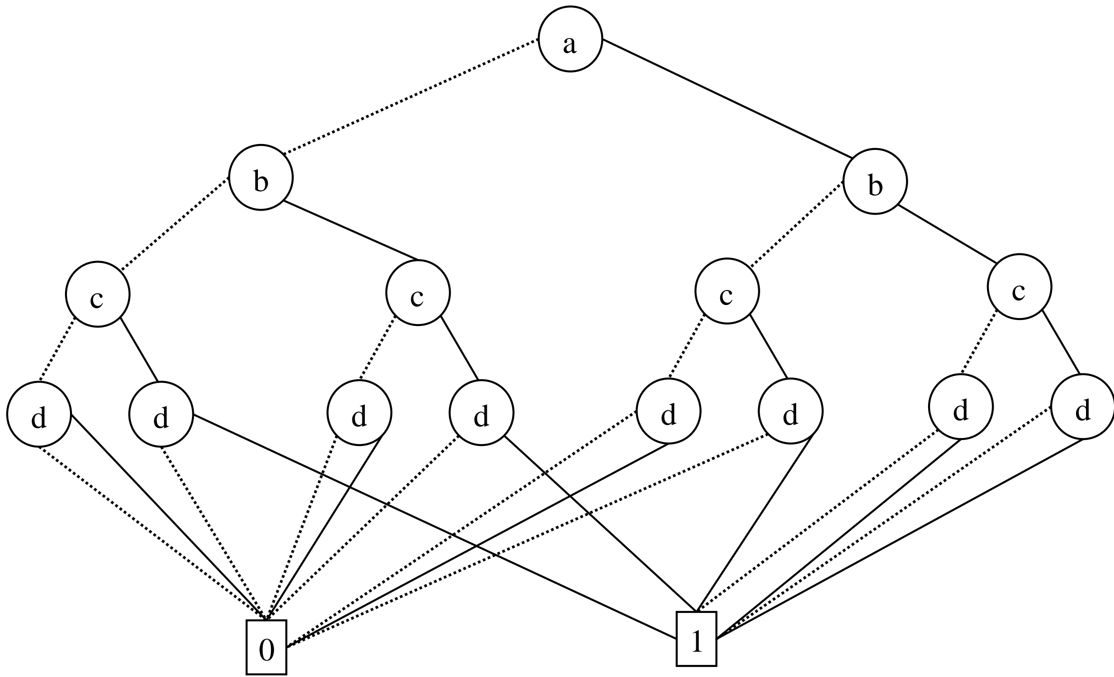


Figure 2.3 Removals of Duplicate Terminals (Leaf Nodes)

Figure 2.4 illustrates the intermediate BDD when rule R2 is applied to the BDD of Figure 2.3. From Figure 2.5 it may be noted that the sub-BDDs enclosed by polygons are structurally identical. So we retain only one of them and redirect edges incoming to the eliminated sub-BDD to the one being retained; resulting intermediate BDD is shown in Figure 2.6. When rule R2 is applied to all structurally identical sub-BDDs of Figure 2.3 we obtain the BDD of Figure 2.4.

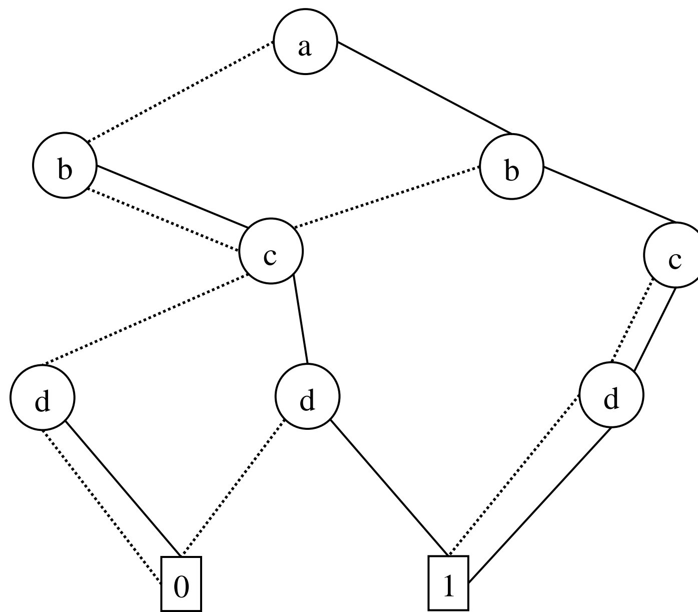


Figure 2.4 Removal of Duplicate non terminals

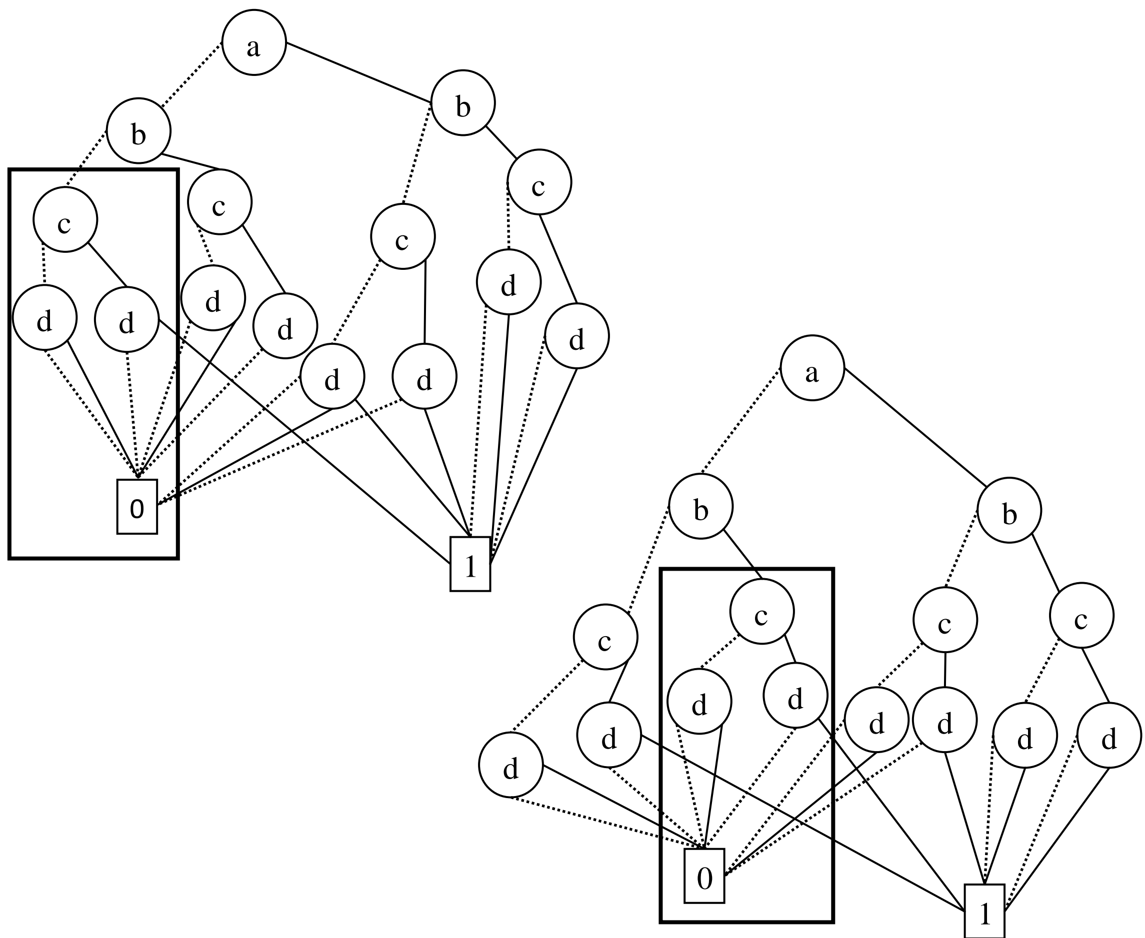


Figure 2.5 A pair of duplicate sub-BDDs (polygons) in the intermediate of Figure 2.3

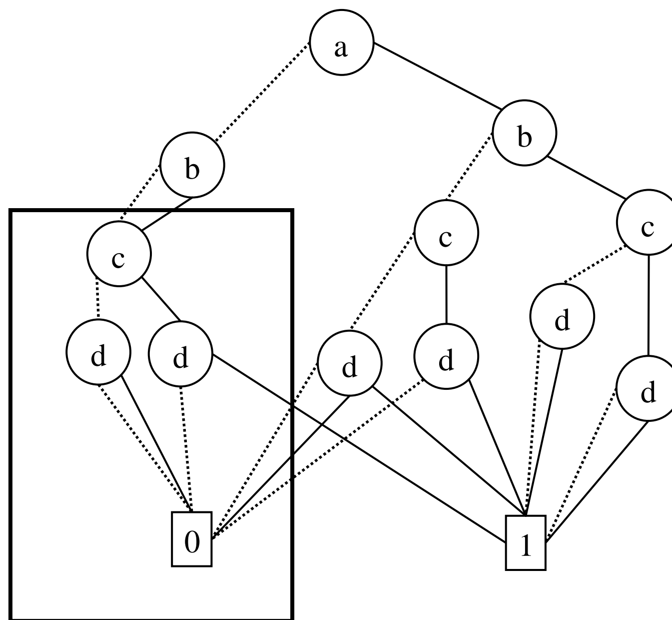


Figure 2.6 Intermediate BDD after applying rule R2 to the duplicate sub-BDDs

From Figure 2.4 it may be noted that there nodes n say, where both the outgoing edges point to the same node m say ; reduction rule R3 eliminates such a node n and points all its incoming edges to m . Nodes of such type in BDD of Figure 2.4 are marked and shown in Figure 2.7.

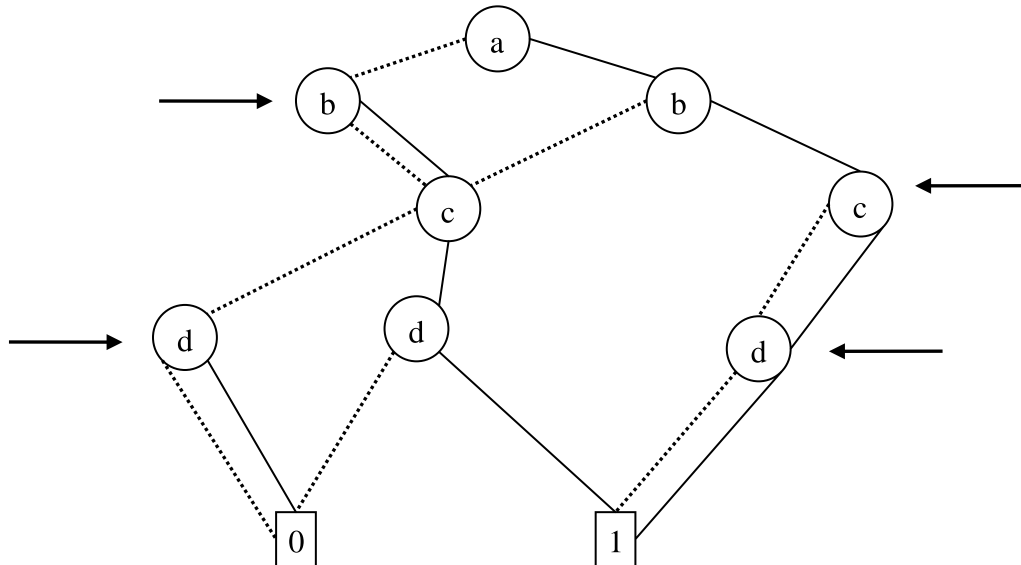


Figure 2.7 Nodes (marked) where both the outgoing edges point to the same node

Figure 2.8 illustrates the intermediate BDD when rule R3 is applied to node 'b', which is marked by an arrow in Figure 2.7. In that both the outgoing edges point to the same node 'c'. When rule R3 is applied to all such nodes, the BDD obtained is shown in Figure 2.9.

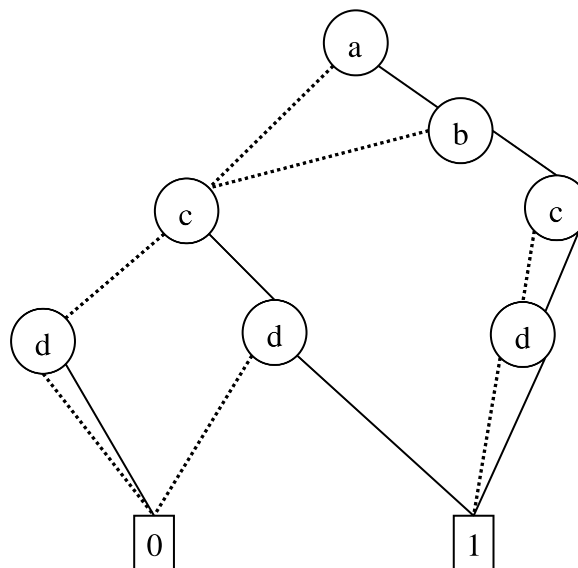


Figure 2.8 Intermediate BDD when rule R3 is applied to node 'b' (Figure 2.7)

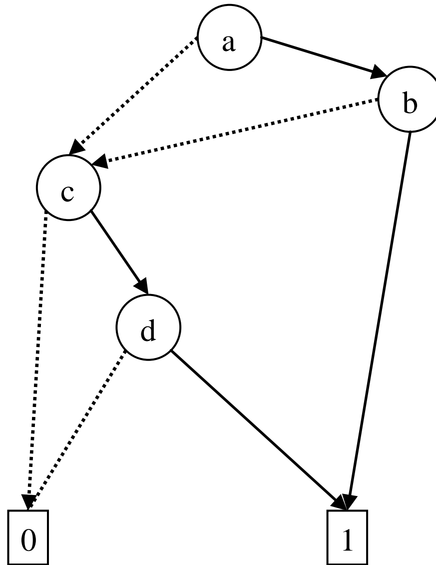


Figure 2.9 Removal of Redundant Test

2.2.1 RBDD USING SHANNON EXPANSION

RBDD is based on Shannon Expansion, according to which Boolean function can be represented by the sum of two sub-functions of the original [2]. In particular, a function f(x) can be written as shown in equation (2.1):

$$f(x) = x.f[1/x] + x'.f[0/x], \tag{2.1}$$

where x' means complement of x. F [1/x] and f [0/x] mean function obtained by replacing the variable x with '1' and '0' respectively.

For example,

If $$f(a) = ac + bc + ab, \tag{2.2}$$

then $$f[1/a] = 1.c + bc + 1.b = b + c + bc \tag{2.3}$$

$$f[0/a] = 0 + bc + 0, \tag{2.4}$$

So using equations (2.3) and (2.4) in (2.2), f(a) can be written as:

$$f(a) = a.f[1/a] + a'f[0/a]. \tag{2.5}$$

The basic or primitive BDDs to represent Boolean function are given below.

B0: represents the Boolean constant 0.

B1: represents the Boolean constant 1.

Bx: represents the Boolean variable x.

These primitive BDDs are shown in Figure 2.10.

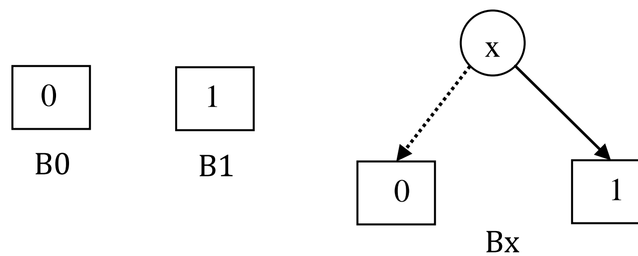


Figure 2.10 Primitive BDDs

Diagrammatically, a BDD of a function of variable a is rooted at a node representing variable 'a', and its children represent the two sub-functions whose sum results in the original expression. The child of the root with a dotted line from parent represents f[0/a] and the one with a solid line from parent represents f[1/a] . BDD of a function of variable 'a' is shown in Figure 2.11.

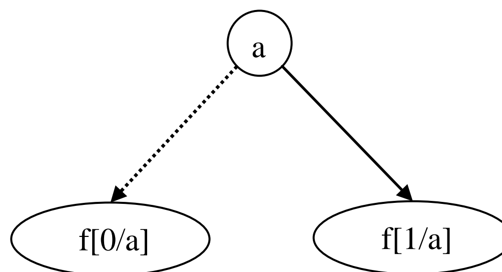


Figure 2.11 BDD of a function of variable 'a'

For the above considered example of an equation (2.2) the initial diagram would be of the form shown in Figure 2.12.

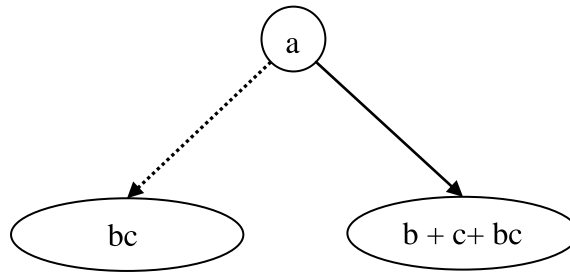


Figure 2.12 Initial diagram of the function f(a) = ac + bc + ab

Further expansion of the sub-functions would lead to the following expressions, which are to be found out till we find the leaf node of the diagram:

Let $f[1/a] = b + c + bc = g(b)$ (2.5)

and $f[0/a] = bc = h(b),$ (2.6)

Using equation (2.5)

$$g[1/b] = 1 + c + 1.c = 1 \quad (2.7)$$

$$g[0/b] = c = i(c), \quad (2.8)$$

Similarly from equation (2.6)

$$h[1/b] = c = i(c) \quad (2.9)$$

$$h[0/b] = 0 \quad (2.10)$$

Similarly from equation (2.9)

$$i[1/c] = 1 \quad (2.11)$$

$$i[0/c] = 0 \quad (2.12)$$

Now the diagram for the expression of equation (2.2) is shown in Figure 2.13. The diagram obtained using Shannon Expansion can further be reduced using the reduction rules explained earlier. The third rule 'Removal of duplicate terminals' is applied and the

final RBDD for the expression is shown in Figure 2.14. In the above case, no more reduction is possible.

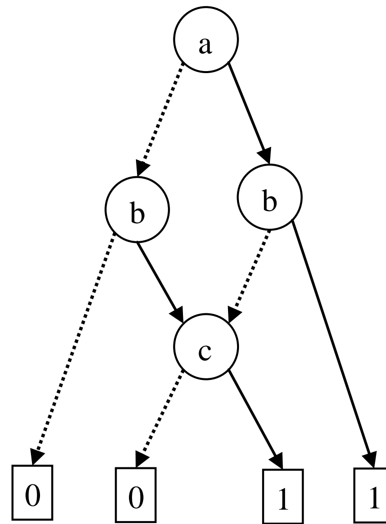


Figure 2.13 Diagram of the function f(a) = ac + bc + ab with redundant nodes

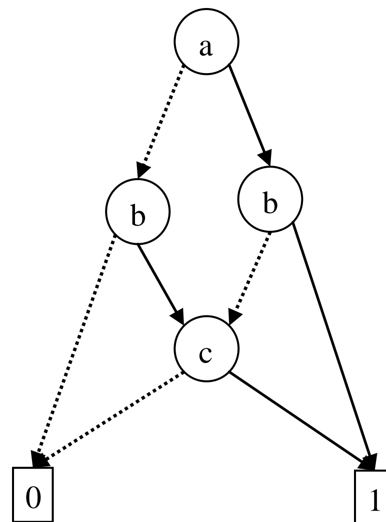


Figure 2.14 Final diagram (RBDD) of the function f(a) = ac + bc + ab

2.2.2 ORDERING OF VARIABLES IN BDD

Ordering of variables in the context of RBDD is important because number of nodes for a given Boolean function depends on the ordering of variables and also there may be inconsistent paths in the BDD if there is no ordering. Many algorithms are available to find the ordering of variable to reduce the number of nodes in a BDD [6-10]. Ideally there should be n! number of orders for the n variable BDD but these algorithms have reduced the number of searches for ordering [6-10].

2.2.3 ORDERED BDD

Binary Decision Diagrams are single rooted direct acyclic graph used to describe a switching function. It was first proposed by Lee [1] and was based on Shannon decomposition in which each node of the decision diagram represents some sub function. The BDD is later ordered and reduced based on some rules proposed by Bryant. Ordered means BDD in which the ordering of variables is consistent on all paths of the graph [9]. Each non terminal node in BDD is labelled by an input variable and have two outgoing branches known as 0-branch and 1-branch. When both the branches of a node point to same node then the node is eliminated. Also when two nodes have identical branches then one node is merged to another node. These reduction rules are shown in Figure 2.15. In Figures the dotted line shows 0-branch and other line shows 1-branch.

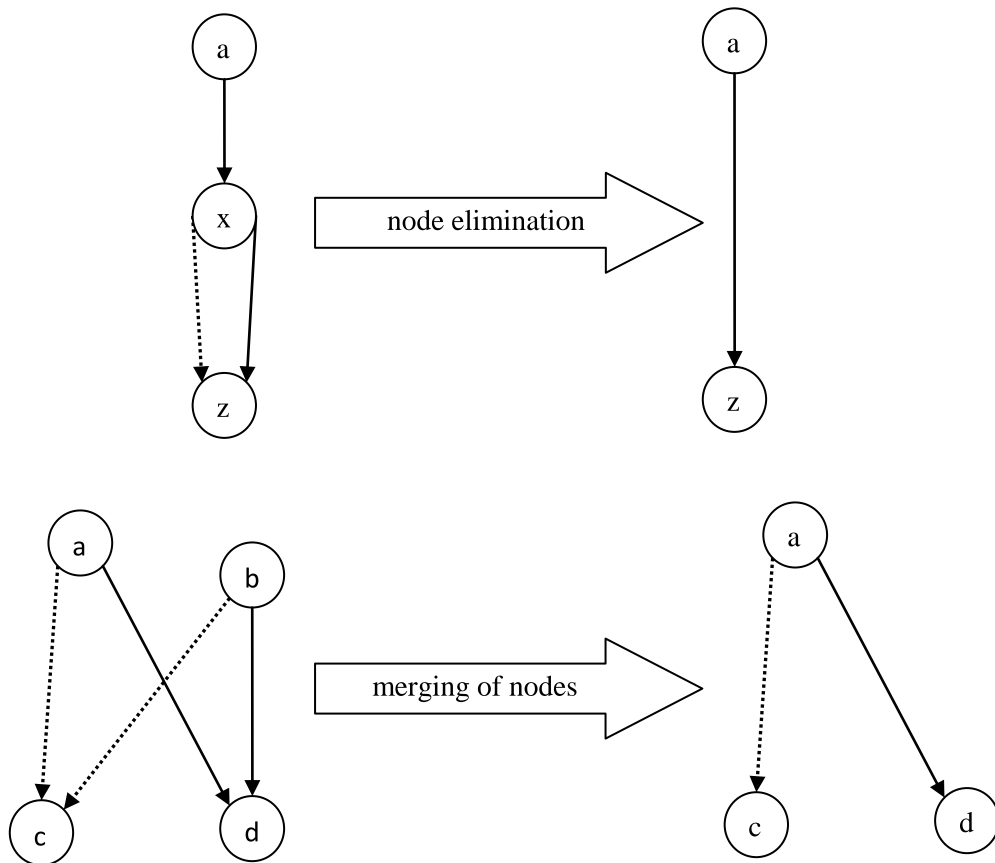


Figure 2.15 Reduction rules for Reduced-Ordered BDDs

The Reduced-ordered Binary Decision Diagrams are often called as BDD. It gives a canonical representation of a Boolean function [2]. The order of the input variables has a strong effect on the size on BDD. The size may vary exponentially if a bad order is

chosen. In Figure 2.16 two BDD for function f = ab + a'c + bc'd with different variable order has shown.

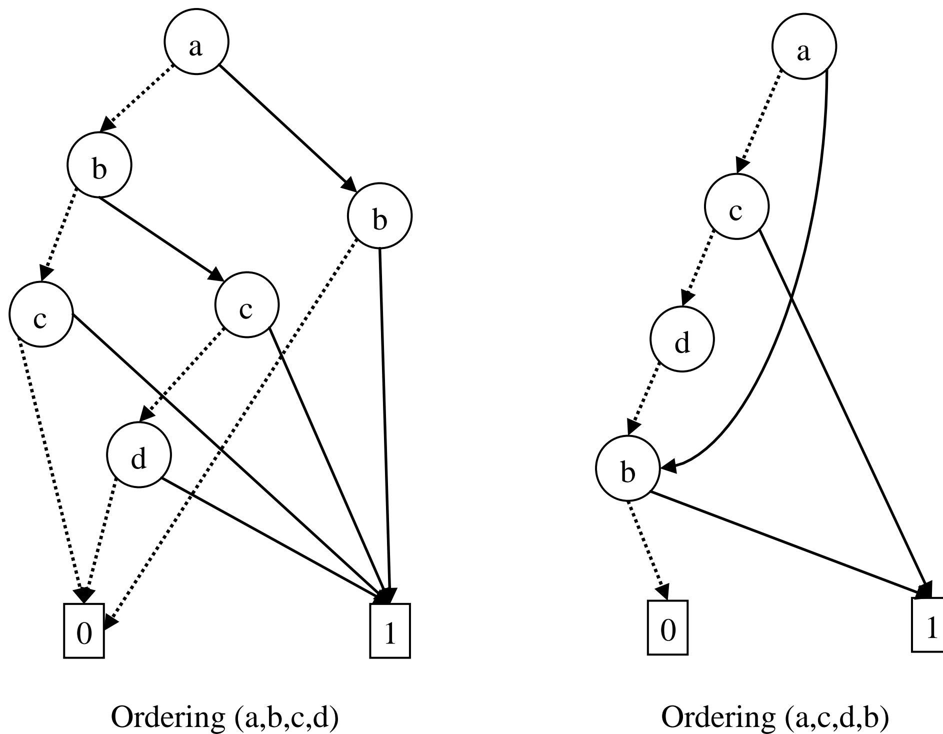


Figure 2.16 BDDs for function f = ab+a'c+bc'd with different ordering

In Figure 2.16, BDD with order a,b,c,d has a size of 6 nodes whereas when order a,c,d,b is used the size is reduced to only 4 nodes. The example function shown in Figure 2.16 is not a complex one and has only four variables. In complex functions where the number of variables is large, the effect of ordering also varies exponentially with the optimal size.

2.3 SIFT ALGORITHM FOR ORDERING OF VARIABLES:

The sift algorithm is one of the best algorithm to find the good variable ordering in BDD [10]. The result obtained by sift algorithm is very near to the optimal ordering. In this algorithm all the variable traverse to the every position and then kept fixed in its optimal position.

Sift Algorithm is as follows:

The sifting first check for the best position of a variable say x[i] of a variable array x[n], where n is the number of variables and i is the current position of variable x[i] in ordering, without changing the position of other variables. In detail the following two steps for the current variable x[i] are performed:

1. The variable is moved through the whole order, and the minimum number of nodes is recorded for each position of x[i] (Figure 2.17).
2. The variable is placed at the position in the order where the minimum in the first step was observed.

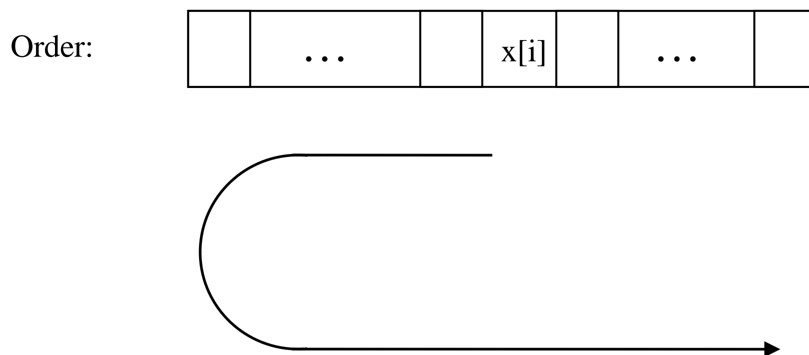


Figure 2.17 Idea of the Sift Algorithm [10]

Let for a variable order x1 x2 x3 x4. The sift algorithm is performed to find the optimal position of x2. The different permutations are shown in Table 2.1 below.

Table 2.1 Sift of variable x2 for different positions

Order number	Ordering	Swapping
1	x1 x2 x3 x4	initial
2	x1 x3 x2 x4	Swap (x2 ,x3)
3	x1 x3 x4 x2	Swap (x2 ,x4)
4	x2 x3 x4 x1	Swap (x2 ,x1)

We will find number of nodes for each order from 1 to 4 and the position of the x2 for the minimum number of nodes is the optimal position of x2. This position of x2 will be fixed for next loop when we will check the optimal position for next unused variable.

The complete sift algorithm is as follows:

sift algorithm()

{

1. Generate a random number between 0 to N-1 and let assign it to i.
2. Take the ith variable and swap it to $(i+1)^{th}$ variable and find the node count.
3. Likewise i^{th} variable will acquire every position by fixing the other variable at their order.
4. Find the optimum position of ith variable for which node count is minimum.
5. By fixing this current variable at its optimum position swap the next unused variable with all variables except the previous fixed ones.
6. Repeat above step for each variable by fixing the optimum position for all previous calculated variables and for each ordering count the number of nodes and if node count is less than previous calculated node count then update the minimum number of nodes and variable order with current values

}

2.5 BDD USING MUX

As discussed above that for multiplexers based design of the digital logics the nodes of the BDD will be represented by a 2×1 MUX. An example function has been taken whose truth table is given in Table 2.2.

Table 2.2 Truth Table of an example function

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

The BDD and the corresponding multiplexer based diagram of the example function of Table 2.2 are shown in Figure 2.18.

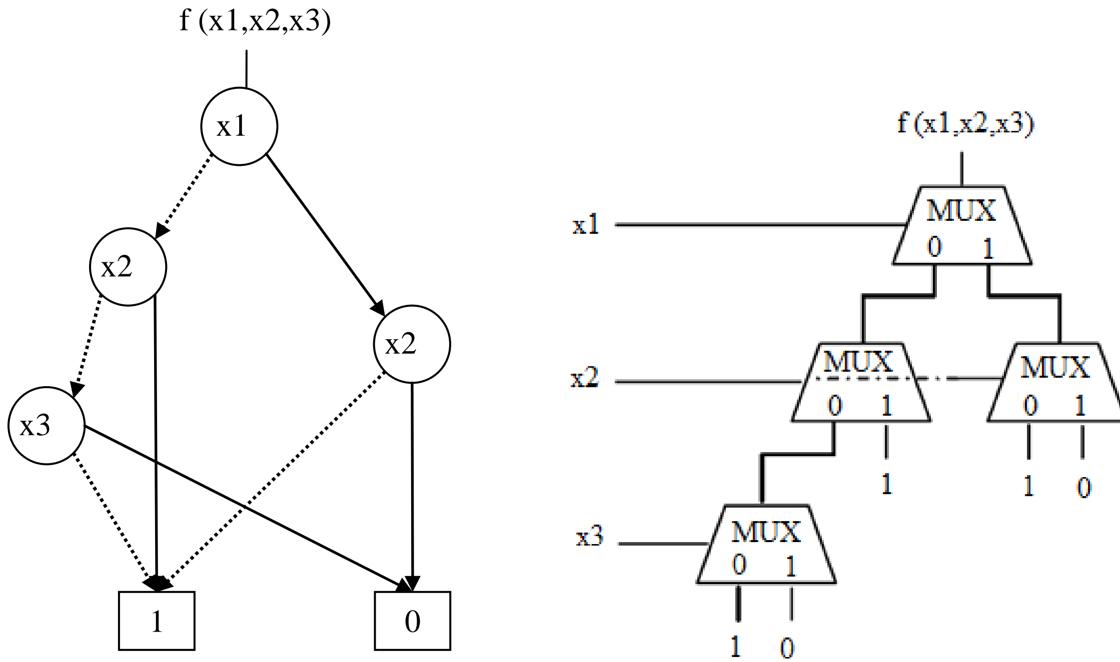


Figure 2.18 Implementation of BDD using MUX

2.6 BDD USING PASS TRANSISTOR LOGIC (PTL):

A Boolean function represented by BDDs can also be implemented by using Pass Transistor Logic. Pass transistor logic (PTL) can be a promising alternative to static CMOS for deep sub-micron design [29]. So the Boolean function represented by a BDD can be easily and effectively transforms for physical design using Pass Transistor Logic. A BDD node and its PTL are shown in Figure 2.19.

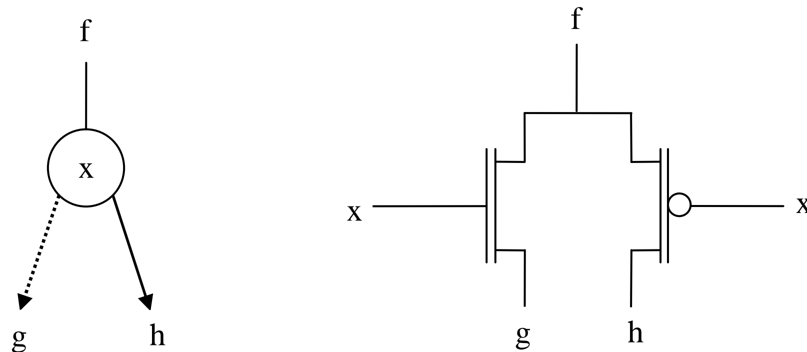


Figure 2.19 Implementing a BDD node in PTL [29]

The 'g' and 'h' shown in Figure 2.19 are the f[1/x] and f[0/x] nodes of the function 'f' respectively. A complete PTL implementation of a function f = a' + bc' from its BDD is shown in Figure 2.20.

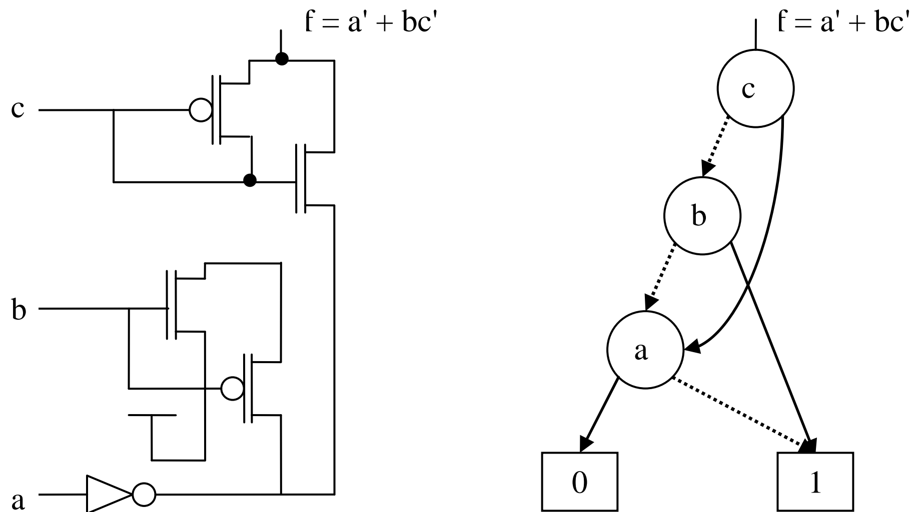


Figure 2.20 PTL implementation of the function f = a' + bc' in PTL [29]

2.7 APPLICATIONS OF BDD

It is observed that BDD represents the canonical form of given function for a particular variable ordering. If the chosen variable ordering is a good one, the resulting BDD is a compact representation of the function.

BDDs have several applications in circuit verification. They can be used to test Equivalence of expressions, validity of Boolean expressions, Satisfiability of Boolean expressions and absence of redundant variable etc. The detail of the application of BDD is as follows:

• **Expression Equivalence:** Instead of converting expressions to be compared to SOP, POS or truth table form to test equivalence, we can construct BDD for the expressions using a common order. If the BDD's obtained for a function by using the same ordering are same then expressions are equivalent.

• **Validity of Boolean Expression:** A Boolean expression is valid if it returns TRUE for all possible variable assignments. This can be tested by noticing occurrence of terminal node '0' in the BDD. If there is no '0' in the BDD then the expression is valid, else not.

• **Satisfiability of Boolean Expression:** A Boolean expression is satisfiable if it returns TRUE for some variable assignment. This can be tested by noticing occurrence of terminal node '1' in the BDD. If there is a '1' in the BDD then the expression is satisfiable, else not.

• **Absence of Redundant Variable:** In case there is a redundant variable in the expression, it can be found out by constructing BDD of the expression. If the diagram does not have any occurrence of a particular variable, then the variable is redundant to the expression.

2.8 BDD PACKAGES:

There are many BDD packages available for manipulation of Binary Decision Diagrams. CUDD is developed by Boulder in University of Colorado [19]. BuDDy is another popular package [20] which is developed by Jorn Lind-Nielsen. These packages use standard ordering method like windows method, Sift algorithm, random method etc...

CHAPTER 3

Literature Review on Algorithms for BDD Ordering

Existing methods for the determination of good variable orderings can be classified into three categories based on the approach. The first is initial heuristics starting from a circuit [12-14], the second is gradual improvement heuristics based on the exchange of variables in the BDD [10] and the third are exact methods [11] to find an optimal ordering. Obviously, it is desirable to determine the exact result. In some applications it is even more important to find the best variable ordering since it turned out by experiments that the best greedy approaches are up to a factor of two worse than the optimal result. But exact methods were so far only applicable to functions with less than twenty variables, due to their exponential runtime behaviour. Since for larger functions no exact method can be applied, in the past few years several methods have been proposed that get high quality results but are time consuming, e.g. A method based on simulated annealing [17] and evolutionary algorithms [12-14]. The drawbacks of these methods were that they had very bad runtime behaviour and/or were only applicable to small functions. Additionally, these probabilistic approaches are sensitive to several chosen parameters and for this are not very robust [22].

E. Felt *et al.* [24] have proposed a method in which he showed an improvement over existing heuristics in the term that it focus on the size of a BDD which represent multiple functions. They have addressed the issue of shared BDD in which the order of variables used for one function may not be optimal for any or all of the other functions. Secondly when a good initial order is selected, the same is used throughout the entire BDD operations which may result in a go off situation because if this good initial order is suitable for some functions it may be non desirable or a bad order for some other functions represented by some intermediate node of the BDD. These two issues have been improved by using this proposed dynamic iterative improvement algorithm. They have taken windows of variable sizes during the ordering operation. For choosing the window

size they have implemented two strategies. If the size of the window is small then it results in faster operation. And if the size of the window is large then it results in a good reduction in BDD size. This variables covered by the window is reorder within the window and then the window position is changed which termed as sliding window. The concept of sliding window is shown in Figure 3.1.

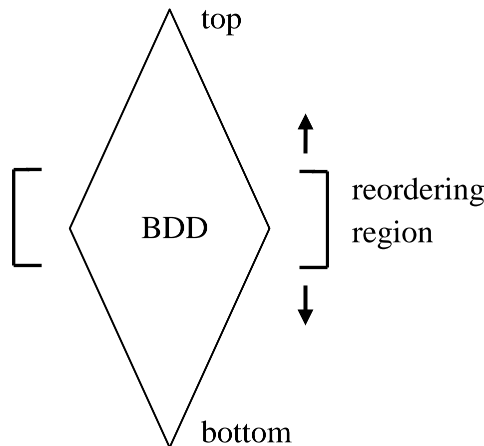


Figure 3.1 Sliding window strategy [24]

This method starts with a small window of variables at the top of the BDD. The best order of these variables in this current window is computed and this best order is used for further computation of BDD. Now the window slides down by one step, i.e. down by one variable. Similarly the best order for this new window is computed and used for further computation and again the window slides down by one variable. When this sliding window covers all the intermediate variables and reaches the bottoms variable it turns around and slides back to the top of the BDD.

In this fashion the reordering window slides up and down according to the requirement i.e. when the stop criteria reaches or the minimum desirable size is found. To traverse the search space they have used some special criteria like they have constructed the variable ordering permutations in bottom-up fashion to eliminate duplicate calculations. The result obtained by their method is compared with the other heuristic techniques. The result showed a good improvement on other techniques.

B. Bollig *et al.* [17] have proposed an algorithm which is based on the 'Simulated Annealing' process. This is a randomized technique. The name and inspiration of annealing came from the process of annealing in metallurgy. In metallurgy this is a technique involving heating and controlled cooling of a material to increase the size of its

crystals and reduce their defects. The Simulated annealing algorithm is a well known heuristic which reduce computation time which in turn reduces the total cost. The method proposed here is used for long run optimal solutions for optimization problems. In this method the swap operation between the two variables in the neighbourhood is performed and if the size is improved then the variables are exchanged between their positions. In this way by using two jump operations the swapping of two variables is done.

N. Zhuang *et al.* [25] have proposed the use of the basic operations in genetic algorithms in the computation of BDD. They used three dynamic parameters of a genetic algorithm: population size, mutation rate and stop criteria. Each of the variable orders is represented by a chromosome and the variables in the chromosome are represented by genes. The crossover of two parent chromosome is done by combining the genes of the two parent chromosome. A random cut point on the both parents is determined using suitable algorithm and the first part of the child genes are given by first parent and the remaining genes i.e. variables are given by the second parent. The process is shown in Figure 3.2.

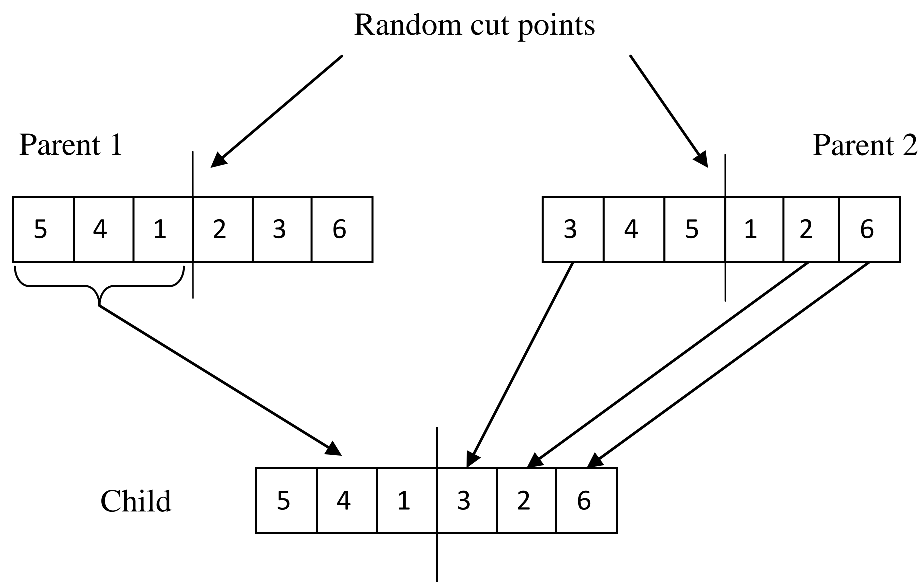


Figure 3.2 Variable Ordering Crossover Procedure [25]

The new child chromosome in generated in the scheme shown in Figure 3.2. The population size is defined as the total number of chromosomes in each stage. In this paper they have taken dynamic population size. Their stop criteria are also depends on the result. If improvement in result found then it spends more time in searching for a more optimal solution whereas if the result is not improving further then it terminates the

search. The new approach to introduce the genetic algorithm in BDD computation has shown good improvement on previous paper in term of time saving and also the result is near to the optimal value than previous methods.

R. Drechsler *et al.* [11] have proposed an exact minimization technique using upper bound. In this technique they have used the concept of branch and bound. First, the BDD for a given Boolean function is constructed using sifting operations during symbolic simulation. The present BDD size is used as an initial upper bound for the exact minimization procedure. For each subset of total order, the best variable ordering and the corresponding BDD size are stored in the hash table. If the lower bound of subset is larger or equal than the upper bound then the subset has no entry in the hash table. Now the remaining variables, which are not in subset, are then shifted to the next level according to predefined computation strategy. Now the total cost of the new order is computed and stored in a cost table. In this way by keeping upper bound fixed, the cost for all the variable orders, which satisfy the criteria, is computed. Authors have marked that while in their work they have fixed the upper bound but if this upper bound is determined by pre-processing based on evolutionary algorithm principle then this proposed algorithm can speed up. Now after completing this stage the next stage will start which use the minimum BDD size of the previous stage as a new upper bound. They have checked their algorithm on CUDD package and compared with previous exact method and found some improvement.

J. Jain *et al.* [23] have proposed the method of finding BDD order by using a sampling of the Boolean function. In this work they have proposed a new methodology to find out ordering by sample the Boolean function into small sub functions. Then they have applied different existing strategy for reordering in these sampled parts of the actual function instead of applying on the whole function.

In their method, for a given function they computed, and then analyzed various subspaces of function using a greedy ordering technique. More precisely, they obtained subspaces and analyzed sub-functions. From such reordered-BDDs, variable orders of individual samples, they computed functional information to deduce the final variable order for the complete Boolean space. For obtaining the sub-functions they used a random selection approach. Now after obtaining sub-functions, they have applied sift algorithm which is a greedy based ordering procedure. This operation on these sub functions generates variety of variable orders. Out of these orders the good ones are picked and then

further used to build complete variable order. The further process used four phases namely sampling for estimation, sampling for candidate order determination and sampling for testing and sampling for evaluation. The first phase, sampling for estimation heuristically decides the parameters related to the size of subspaces used for calculating the candidate variable orders by first examining the behaviour of the given function on some very small subspaces. The same time, if the given function is very difficult to analyze then such behaviour may be apparent in even very small subspaces. The second phase sampling for candidate order determination generates a set of candidate variable orders.

Essentially, it generates a different number of simpler functions say 't', and for each function they independently call an ordering algorithm like sifting based reordering procedure. This provides up to 't' different candidate variable orders. The third phase, sampling for testing weeds out the ineffective sample orders using an independently generated sample space on which each order generated in the previous phases is used to create BDDs. The candidate orders that perform poorly are rejected. The fourth and last phase, sampling for evolution allows for the given candidate sample orders to be further improved, by reordering them in independently obtained subspaces, to increase their chance of being suitable for the complete Boolean space. Given m variable orders as input to this phase, the variable order that gave the smallest graph, on the subspace chosen for evolution, is selected as the sampling-computed variable order.

Rolf Drechsler [26] has proposed the technique of using lower bounds in the BDD minimization process. In this paper, the author proposed that the lower bounds computed during the minimization process can speed up the computation significantly. First they studied lower bounds from a theoretical point of view. Then they have incorporated these techniques in dynamic minimization algorithms. They have shown that by the computation of good lower bounds, large parts of the search space can be pruned in the many dynamic approaches like sift algorithm, resulting in very fast computations. Their experimental results showed that by using lower bounds, the number of exchanges can be reduced by more than 50% on average, resulting in a speedup of nearly 70% and hence the proposed approach is highly efficient in computing compact BDDs when compared with the normal sift operation.

W. Hung *et al.* [15] have proposed the use of scatter search in finding good variable ordering in BDD. Scatter search is very aggressive and attempts to find high quality

solutions fast. Scatter search operates on a set of solutions, the reference set, by combining these solutions to create new ones. The main mechanism for combining solutions is such that a new solution is created from the linear combination of two other solutions. Unlike a population in genetic algorithms, the reference set of solutions in scatter search tends to be small.

In genetic algorithms, two solutions are randomly chosen from the population and a crossover or combination mechanism is applied to generate one or more offspring. A typical population size in a genetic algorithm consists of hundred elements, which are randomly sampled to create combinations. In contrast, scatter search chooses two or more elements of the reference set in a systematic way with the purpose of creating new solutions. Since the combination process considers at least all pairs of solutions in the reference set, there is a practical need for keeping the cardinality of the set small. Typically, the reference set in scatter search has twenty solutions or less. The scatter search process consists of diversification generation, improvement, and reference set update, subset generation and solution combination.

The idea of maintaining reference sets in Scatter Search is to maintain both quality and diversity. In this paper the authors have partitioned the reference set into two subsets RefSet1 and RefSet2. RefSet1 is the high quality subset of size b1, it contains the b1 orders so far encountered with the smallest BDD size. RefSet2 is the high diversity subset of size b2, it contains the b2 orders so far encountered that do not qualify for RefSet1 but with the largest distance. In a reference set update, new variable orders are first considered for membership in RefSet1. If not, they are considered for membership in RefSet2. They have defined quality in terms of BDD size i.e. the smaller the number of nodes in BDDs, the better the quality.

They calculated diversity in terms of distance. The degree of diversity is defined in terms of minimum distance. To calculate the distance between two variable orders, they have taken the difference in positions between two orders with regard to the same variable. In Scatter Search, Improvement is to transform a trial solution into one or more enhanced trial solutions. If no improvement can be made, the enhanced solution is considered to be the same as the input solution. In this paper for implementation they used the sifting algorithm. In subset generation process, the algorithm operates on the reference set to produce a subset of its solutions as a basis for creating combined

solutions. At the end the variable orders are combined to give the improved result in terms of fast computation time when compared with the other method of ordering.

W. Mingquan *et al*. [27] have proposed a new BDD minimization algorithm based on a genetic tabu hybrid strategy. It combines the global space search of genetic algorithm with the local neighbourhood search and the gradual global optimizing of tabu search. In this approach first variables are swapped and then the uniform distribution is utilized to generate neighbourhood solutions in the search space. The key of global search is that its solution should be able to avoid falling into a local suboptimal trap whatever initial conditions. In local search the optimal solution is greedily sought in the neighbourhood of the current solution, but especially is prone to converge to the local minimum. Its efficiency is completely depended on the neighbourhood structure and initial solution.

Fred Glover proposed tabu search to overcome local optimum. Its basic principle is to pursue local search whenever it encounters a local optimum by allowing non-improving moves; cycling back to previously visited solutions is prevented by the use of memories, called tabu list, that record the recent history of the search, a key idea that can be linked to artificial intelligence concepts. At initialization, it makes a coarse examination of the solution space, known as diversification, but as candidate locations are identified the search is more focused to produce local optimal solutions.

In this paper, the authors have used tabu search strategy to improve the Genetic Algorithm based BDD minimization algorithm by combining the inherent parallelism and global space search of Genetic Algorithm with the local neighbourhood search and the gradual global optimizing of tabu search. The genetic tabu hybrid algorithm proposed in this paper is partitioned into two sections: the first-level selection and the second-level selection. The whole scheme for the genetic tabu hybrid algorithm is shown in Figure 3.3. Initially they have taken a guess set of N feasible candidate solutions (parents which represent one variable order each) constructing a population is chosen randomly in the search space which is all permutations of BDD variables. Each parent can generate children to form a family. The children in the same family become the objects being selected in the first level where tabu search is used to choose the optimal child as parent for the next generation.

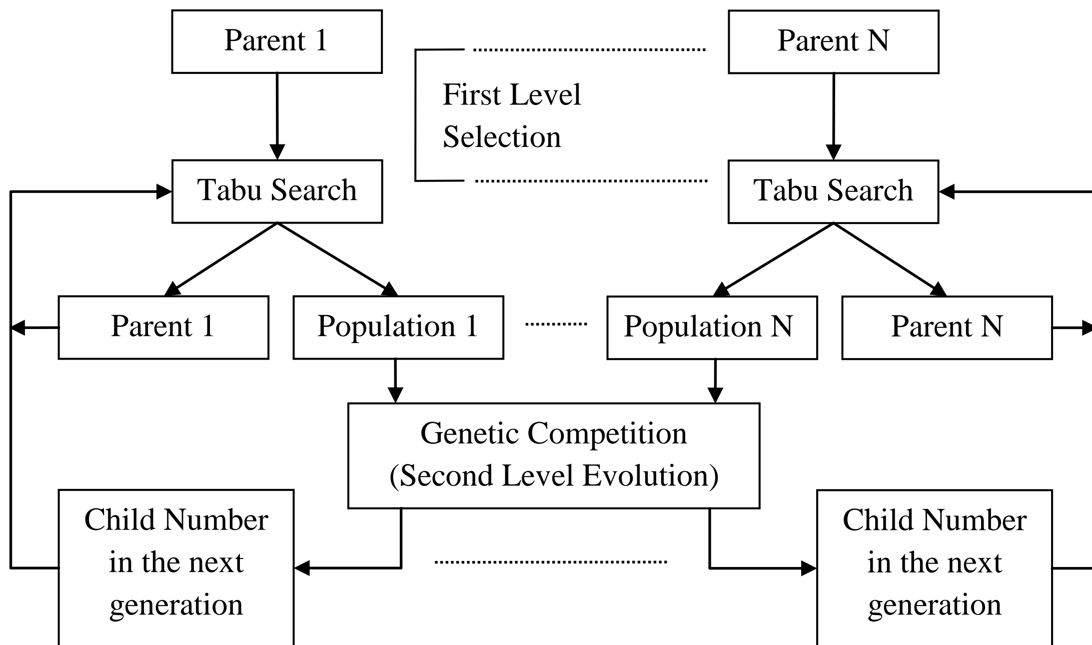


Figure 3.3 Framework of genetic tabu hybrid algorithm [27]

The competition between all families is performed in the second-level selection, which actually provides a measure of the fitness of every family. Then they have applied genetic operations in the second-level. Instead of the objective values of a single point which might be biased, a fitness value based on the objective values of all children in the same family has been taken.

They have taken the number of children allocated to each family for the next generation is proportional to their fitness values, but the total number of children in the next generation is constant. In this way they ensured that the fitter individuals have a better survival chance, and eventually only one optimal family can survive. In the second level selection they have measured the regional information of the search space. Here the fitness value indicates the distribution of good solutions in the search space. They have focused more attention on the region where the higher fitness value is found. For speeding up algorithm convergence, the group sifting algorithm has been used to optimize the initial population for better objects selected. In initialization, N initial guessed solutions can be considered as N basic points in the search space. The space spanned from those points is just the set of all variable ordering. Their experiment result showed that their strategy of using a genetic tabu hybrid algorithm has shown a good improvement in terms of time efficiency.

R. Ebendt *et al.* [16] have proposed the method of combining ordered best first search with the well known Branch and Bound technique. The method proposed here is used mainly in the field of artificial intelligence. The so called A^* algorithm. The A^* algorithm prunes large parts of state spaces and has become an integral part of standard artificial intelligence search techniques. Using this successful artificial intelligence technique the authors have shown that the problem of finding an optimal variable ordering is reduced to the problem of computing an optimal path in a state space. This state space is much smaller than the space of all orderings. It is explored by expanding states to its successors until a goal state is found.

In this process of state space exploration they have shown that the new method always determines the best, i.e., the most promising state before the next expansion. By this, they have effectively pruned large parts of the state space and observed that this pruning effect is higher than in any other approach for exact BDD minimization used before. The best-first ordering of the states further speeds up the run time of their operation significantly.

To prune the search space more effectively, they have combined their new approach with already existing branch and bound (B&B) techniques. Their approach of combining A^* and B&B is an improvement of A^* itself, i.e., it can be transferred to other applications as well. In this proposed method they have found another improvement of A^* which addresses the memory requirement of the approach, which in this application yielded reductions in memory consumption of more than 60%. Their experiment result showed that there is significant run time reductions observed on benchmark functions.

P. Prasad *et al.* [13] have proposed a static variable ordering technique which uses the input Boolean expression to find the variable order which is further used to minimize the average path length. Their method is based on the single level Boolean function; hence, for a multilevel Boolean function, they have converted to single level function prior to applying their proposed method. Their proposed method consists of selecting a variable based on the complexity of sub-functions derived by assigning logic 1 and logic 0 for that variable. In their method they have given priority to the variable that produces sub-functions with minimum complexity over other variables. The complexity of sub-functions is evaluated based on the number of variables, number of product terms and the number of variable occurrences. Using this method, the BDD graph with shortest possible paths among each node including the terminal nodes has been produced, which eventually

reduced the average path length. This method proposed by them showed a good variable ordering over other earlier methods.

P.W.C. Prasad *et al.* [18] have proposed another method of finding an optimal variable ordering which is based on the manipulation on the graphical representation of the Boolean functions. The graph representation of an example function is shown in Figure 3.4. In Figure nodes x1, x2, x3 and x4 are the input nodes and node 1,3 are the output nodes. Nodes 1, 3, 4, 6, 7, 9, 11 and 13 represent NOT operations and nodes 2, 5, 8, 10, and 12 represent AND operations.

The proposed method uses the graph topology to find the best variable ordering. Therefore the input Boolean function is converted into a unidirectional graph. They have used three levels of graph parameters to increase the probability of having a good variable ordering. The initial level used the parameters: the total number of nodes in all the paths, the total number of paths and the maximum number of nodes among all paths. Since the variable with the highest number of nodes has the greatest effect on the circuit, it is placed first in the variable ordering.

In their second and third levels they used two extra parameters named 'the shortest path among two variables' and the 'sum of shortest path from one variable to all the other variables'. They have performed the permutation of the graph parameters at each level for each variable order and the number of nodes is recorded.

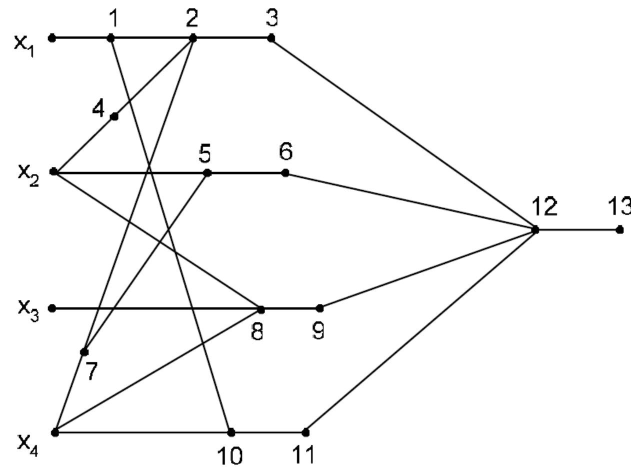


Figure 3.4 Graph Representation for function $f = \left((x_1'.x_2'.x_4')'.(x_1'.x_4)'.(x_2.x_4')'.(x_4.x_2.x_3)'\right)'$ [18]

In their experiment they showed that the proposed algorithm based on three level permutations of graph parameters is capable of handling multiple output benchmark circuits. They have implemented their algorithm and it has been tested using benchmark

circuits and their results have been compared with three selected CUDD reordering methods. They found that this algorithm is deterministic in the sense that there is no heuristic involved in any of the primary parameters of the algorithm. This algorithm is a promising alternative to existing reordering methods which uses many heuristics to reduce the number of nodes in BDD.

U. Kuhne *et al.* [28] have proposed the use of genetic programming which is an improvement over previous genetic algorithms used. The approach presented in their paper is based on Genetic Programming. It works directly on the graph structure of the BDDs. The flow is of the algorithm is shown in Figure 3.5.

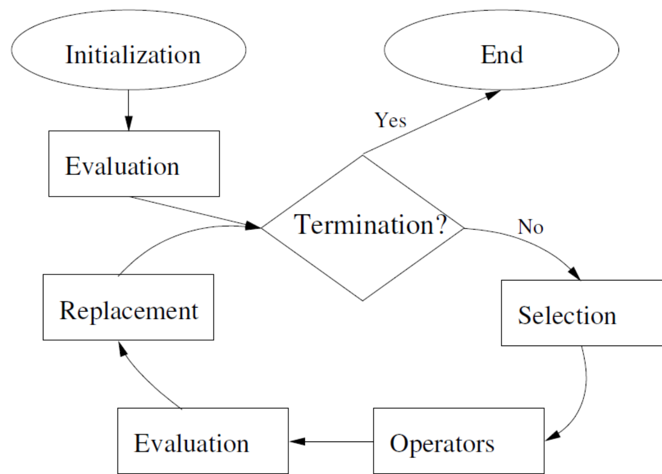


Figure 3.5 The flow of the Genetic Programming algorithm [28]

In initialization they have used standard functions of the BDD packages. In evaluation phase they have taken two objectives: correctness and compactness. With correctness being the more important one shows whether an individual represents the target function correctly or not. They have used crossover and mutation techniques in the evaluation phase to generate the new improved populations. For different stages they have used different functions from BDD package. Finally they have carried their experiment in different initial order. The first one starts with a population of randomly initialized graphs, the second one starts with correct OBDDs. The result showed is not very effective in the first method with random initial population but the result with the correct BDDs is quite good.

O. Brudaru *et al.* [6] have proposed the double hybridization in which the first hybridization adopts embryonic chromosomes as prefixes of variable orders instead of complete variable orders and combines a branch & bound technique with the basic

genetic algorithm. The second hybridization is done with the existing sifting algorithm, known as one of the most effective heuristic for this problem, which is incorporated as a hyper-mutation operator. Their results showed that this kind of hybridization remarkably improves the performance of the resulting algorithm.

S. Chaudhury *et al*. [14] have used the Branch and Bound method with an improved branching and bounding scheme. They have introduced an iteration process which has three main components: selection of the solution set for bound calculation, and branching. In this the selection of the next solution set is based on the bound value of the solution sets obtained after branching from the previous level. For each of these iterations, it is checked whether the subspace consists of a lower bound, in that case, it is compared with the current best solution thereby retaining the better of the two while pruning the other sets.

The efficiency of the their proposed method depends strongly on the node-splitting procedure and on the lower bound estimations, for that they started the search from a set of sorted non-duplicate potential solutions and applied variable's sequence inversion of a solution. Accordingly, to insure maximum non-over-lapping subsets or no overlapping subsets, they proposed three different types of solutions space, namely left-side_inverted, rightside_inverted and bothside_inverted. The technique applied is shown in Figure 3.6.

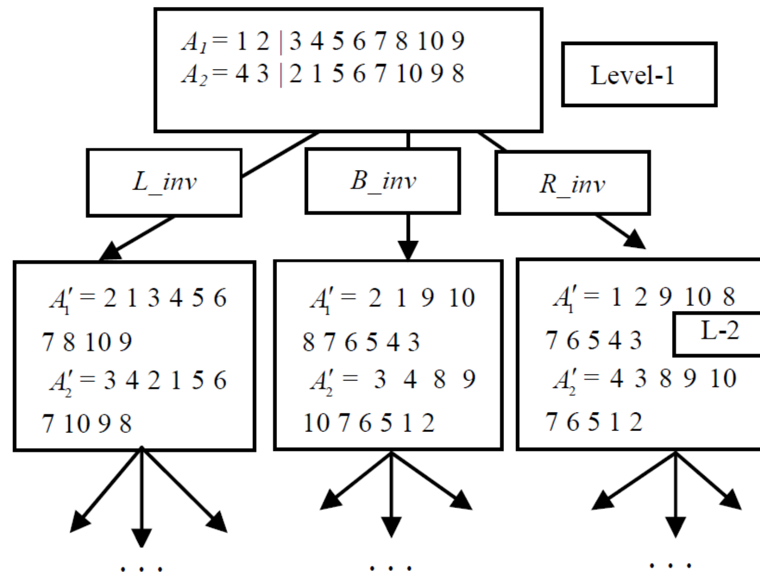


Figure 3.6 Branching scheme for B&B algorithm [14]

For bounding, the lower bounds are calculated by setting the objective function of the proposed problem based on the some fitness as defined in the paper. Their key idea is that

if the lower bound for some tree nodes (set of candidates) A is greater than the lower bound for some other node B in the same level, then A may be safely discarded from the search. Otherwise, the bounding function for the subspace is calculated and compared with the current best solution.

The search terminates when we reach Level-(n-2), where n is defined as the number of variables and the optimal solution is then the one recorded as 'current best'. Their procedure stops when all nodes of the search tree are either pruned or solved. To check whether the solution obtained converges to the local minimum or not, they have repeated the above algorithmic steps by starting the search from a different solution set obtained by reversing the set of Level-1.they have implemented their algorithm in C language and compared with a genetic algorithm based approach. Their experimental results showed that when the area reduction is given higher priority, this method showed good result than many other heuristics.

CHAPTER 4

BDD Using Improved Initial Order

Many methods proposed so far use a random initial ordering with the different heuristic based algorithms. The BDDs constructed using the random initial ordering is then improved iteratively by applying new orderings. These new orderings are generated by using different approaches. The initial ordering used in these approaches is called a seed ordering. This seed ordering has a great impact on the iterative process of finding an optimal ordering in many reordering techniques. If an arbitrary ordering is used as initial ordering then the solution obtained may be trapped in local minima where as the optimal solution are the global minima.

This work proposed a method of finding an initial ordering which is an improved seed ordering. This new improved seed ordering is used in the different reordering approached.

The method to find the improved seed ordering is divided into three parts. The first is to find the cost of the each variable by moving it to a different position. The cost of the variable at a position is the size of the BDD at that position. The small size means that the cost is small and if the size is large then cost is large. In the second part we choose the position corresponding to a variable where it has the minimum cost i.e. maximum weight. In this way we find the minimum cost for every variable based on its weight and combine them. In the third step the variables are ordered according to their corresponding maximum weights. All the three parts of the proposed approach is discussed in details in the following sections.

Finding Cost of Each Variable:

In this approach first a variable is taken randomly and swapped with the variables at different positions. The number of these variables at different position is fixed. We call this fixed number of different position say '*l*'. In order to reduce run time for large circuits, the method proposed here use the different value of '*l*'. The value of '*l*' depends upon the number of input variables. For large circuits the value of '*l*' is generally kept very small compare to the number of variables. For small circuits the value of '*l*' is high with

compare to the number of variables. The cost of each of the positions of this current variable is calculated and stored in the first row of an array of size l×n. Here n is the number of input variables. After swapping with each of the 'l' variables the current variable is stored back to its original position. Now next variable is taken and again swapped with variables at 'l' different positions. In this way each the n variable is swapped and restored after 'l' times of swapping. The cost of each variable for each of the 'l' positions is calculated from the BDD package and stored in the corresponding cell of the matrix. The cost matrix for this is shown in the Table 4.1.

Table 4.1 Cost matrix

Variables	Cost			
v1	Cost $_{v1,vl1}$	Cost $_{v1,vl2}$	. . . .	Cost $_{v1,vln}$
v2	Cost $_{v2,vl1}$	Cost $_{v2,vl2}$	. . . .	Cost $_{v2,vln}$
v3	Cost $_{v3,vl1}$	Cost $_{v3,vl2}$	. . . .	Cost $_{v3,vln}$
v4	Cost $_{v4,vl1}$	Cost $_{v4,vl2}$	. . . .	Cost $_{v4,vln}$
: : : : :	: : : : :	: : : : :		: : : : :
vn	Cost $_{vn,vl1}$	Cost $_{vn,vl2}$	. . . .	Cost $_{vn,vln}$

In the Table 4.1, the rows of the cost matrix represent cost of each variable. The cell entry 'Cost $_{vn,vln}$' shows the cost when the current variable 'v_n' is swapped with the variable at position corresponding to 'l_n' In this way the cost for all the n variables is calculated.

Finding minimum cost:

From the cost matrix shown in Table 4.1 the minimum cost of each of the variable is calculated and the position of the variable corresponding to that minimum cost is stored. The minimum cost for a variable shows that it has the high weight at that position. So by finding the minimum cost, the position at which the variable has highest impact on the BDD size is calculated. For this in this one another matrix of minimum cost is used. Each

row of this minimum cost matrix stores the minimum cost of the variable and the position corresponding to that minimum cost. This new matrix is shown in the Table 4.2.

Table 4.2 Minimum cost matrix

Variables	Minimum cost	Position
v1	min_cost $_{v1}$	P_1
v2	min_cost $_{v2}$	P_2
v3	min_cost $_{v3}$	P_3
v4	min_cost $_{v4}$	P_5
: : : :	: : : :	: : : :
vn	min_cost $_{vn}$	P_n

Generating the Seed Ordering:

The seed ordering proposed here is generated from the minimum cost matrix shown in Table 4.2. The minimum cost column of the matrix is sorted in ascending order of their cost. This ensures that the variable which has lowest cost comes first in the proposed seed ordering and so on. The column position is also sorted with the minimum cost matrix based on the cost of the variables. This sorted position matrix is our improved seed ordering.

This proposed method can be divided into seven steps.

Step 1. In this step each variable is moved to 'l' different position. The distance between two successive positions is the distance function 'd'. We have taken the following assumption to find the number of positions 'l':

$l = \text{n}$, for $\text{n} \leq 10$;

$l = 10$, for $10 < \text{n} \leq 30$;

$l = 15$, for $30 < \text{n} \leq 50$;

$l = 20$, for $50 < \text{n} \leq 100$;

$l = 30$, for $\text{n} > 100$;

where 'n' is the number of variables,

This assumption insures that for large circuits i.e. circuits having more number of input variables the algorithm do not take a large run time.

Step 2. In this step the variables 'V_i', which has to be swapped with current variable is determined using equation (4.1).

$$V_i = (i * n)/l \quad (4.1)$$

here i varies from 0 to n, n is the number of input variables.

Step 3. In this step the current variable' x_n' is swapped with each 'V_i' variables and the cost are calculated. This swapping scheme is shown in Figure 4.1.

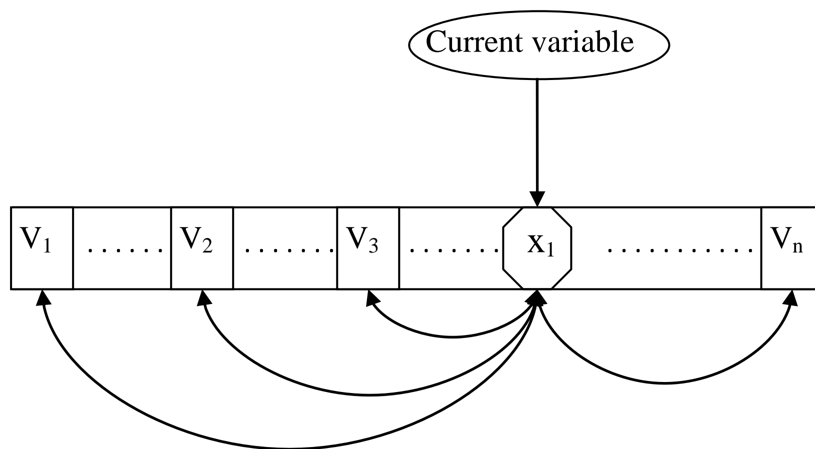


Figure 4.1 Swapping Scheme for the proposed method.

Step 4. In this step the cost from each swapping is stored in the cost matrix proposed in Table 4.1.

Step 5. In this step the minimum cost for each of the variable and the position corresponding to the minimum cost is stored in the minimum cost matrix shown in Table 4.2.

Step 6. In this step the minimum cost and the column position of the minimum cost matrix is sorted in ascending order. The new position of the ith element of the array 'minimum cost[i]' in the sorted array is the position of variable 'x_i' in our proposed improved initial ordering.

Step 7. This new initial order is used as a seed order in the different reordering techniques available in the BDD packages.

The flow chart of the proposed method is shown in the Figure 4.2

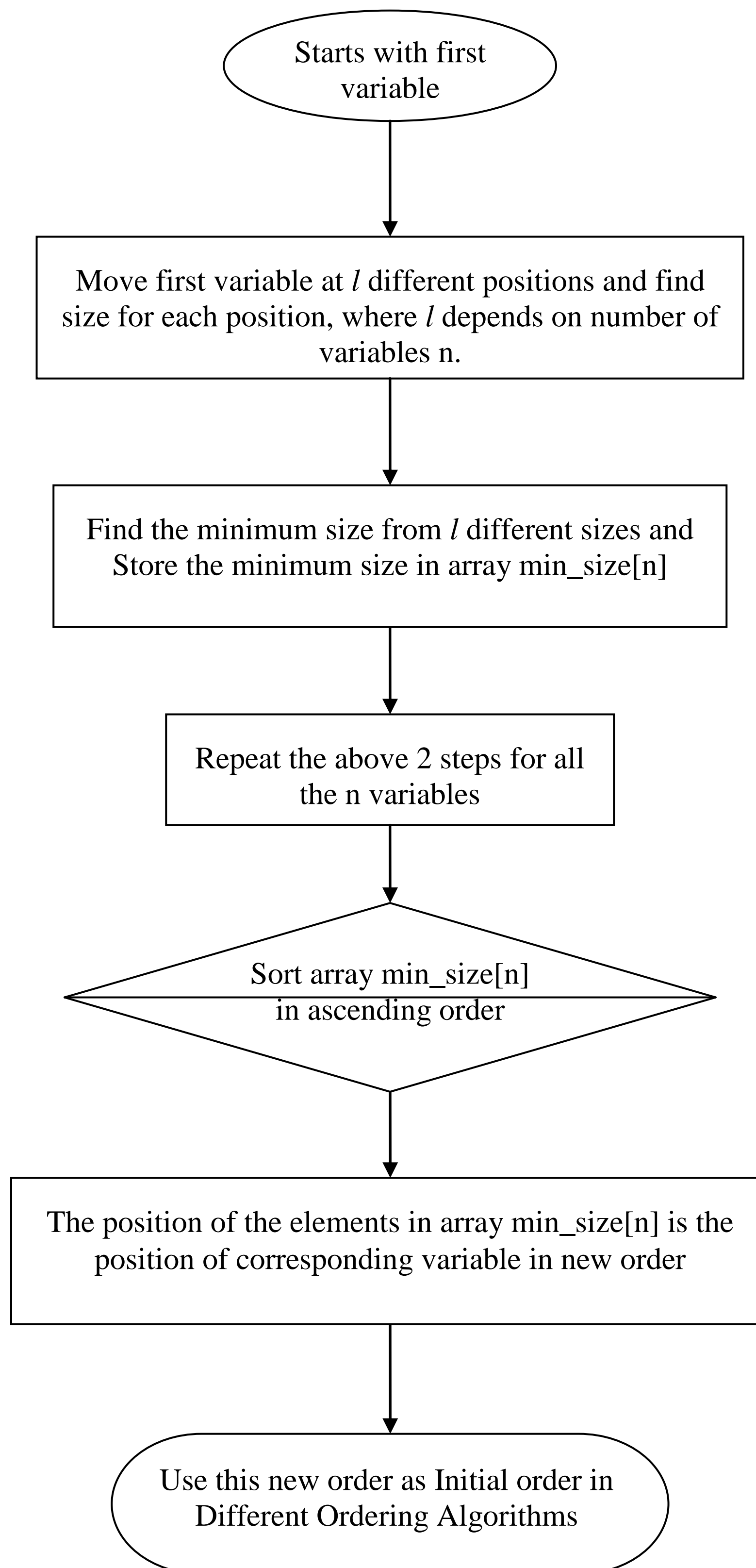


Figure 4.2 Flow chart of the proposed method

CHAPTER 5

RESULTS

The proposed method is implemented in C on a UNIX based Ubuntu machine. The different benchmark circuits available in the LGSynth93 benchmark package and of the adder's package are used to test this proposed method. The proposed improved seed ordering is generated by the algorithm written in C. This generated improved ordering is used with two standard reordering techniques available in the BuDDy package: sift algorithm and win2ite. The BDD size is calculated by BuDDy. The total area of the circuits is calculated by assuming that every node is implemented by a 2×1 MUX. Then, the area of a single MUX is used [21]. The total area is calculated by multiplying the number of nodes in BDD with the area of a single MUX.

The number of nodes in the BDD is calculated by using both the initial orderings. In Table 5.1 and Table 5.2 the comparison between the BDD size with a random initial ordering and with proposed initial ordering of benchmark circuits from LGSynth93 benchmark suite is shown. In the tables the column #i and #o shows the number of inputs and outputs in the benchmark circuits respectively. Table 5.1 shows the comparison between the BDD sizes using sift algorithm method. Table 5.2 shows the comparison between the BDD sizes using win2ite method.

Table 5.1 BDD Size Comparison Using Sift Algorithm

Benchmark Circuits (LGSynth93)	#i	#o	Sift Algorithm		
			Simple Initial Ordering	Proposed Initial Ordering	**Improvement (%)**
vg2	25	8	286	169	**40.96**
cordic	23	2	39	39	**0**
b12	15	9	65	65	**0**
sao2	10	4	93	85	**8.60**
5xp1	7	10	78	78	**0**
clip	9	5	105	105	**0**
misex3c	14	14	467	445	**4.71**
misex2	25	18	84	83	**1.19**

Table 5.2 BDD Size Comparison Using win2ite Method

Benchmark Circuits (LGSynth93)	#i	#o	win2ite Method		
			Simple Initial Ordering	**Proposed Initial Ordering**	**Improvement (%)**
vg2	25	8	325	**295**	**9.23**
cordic	23	2	48	**40**	**16.66**
b12	15	9	61	**60**	**1.64**
sao2	10	4	95	**88**	**7.37**
5xp1	7	10	88	**89**	**1.14**
clip	9	5	94	**93**	**1.06**
misex3c	14	14	455	**452**	**0.66**
misex2	25	18	107	**81**	**24.30**

Table 5.3 shows the number of nodes in BDD of different adder circuits with the win2ite method.

Table 5.3 BDD Size Comparison of Adder Circuits Using win2ite Method

Adder Circuits	#i	#o	win2ite Method		
			Simple Initial Ordering	**Proposed Initial Ordering**	**Improvement (%)**
02adder	5	3	25	**23**	**8**
03adder	7	4	34	**34**	**0**
04adder	9	5	35	**35**	**0**
05adder	11	6	94	**63**	**32.98**

From the above tables Table 5.1, Table 5.2 and Table 5.3 it is observed that there is an improvement in the size of BDD in many circuits. From Table 5.1 for the sifting algorithm method there is improvement in four circuits out of eight. Maximum improvement is in 'vg2', which has 25 input variables. Similarly from Table 5.2 it can be easily observed that for LGSynth93 benchmark circuits there is an improvement in BDD size in almost every circuit. Maximum improvement is in 'misex2' which is almost 25 %. Also for adder circuits, using win2ite method there is an improvement in BDD size of 2 bit and 5 bit adder. This is tabulated in Table 5.3. The improvement in BDD size in different circuits can be easily observed by using line diagrams. In Figure 5.1, Figure 5.2 and Figure 5.3 we have shown the comparison in BDD size of different circuits by using the proposed improved initial ordering.

In Figure 5.1 and Figure 5.2 we have shown the comparison of size of LGSynth93 benchmark circuits by using the method sift and win2ite respectively.

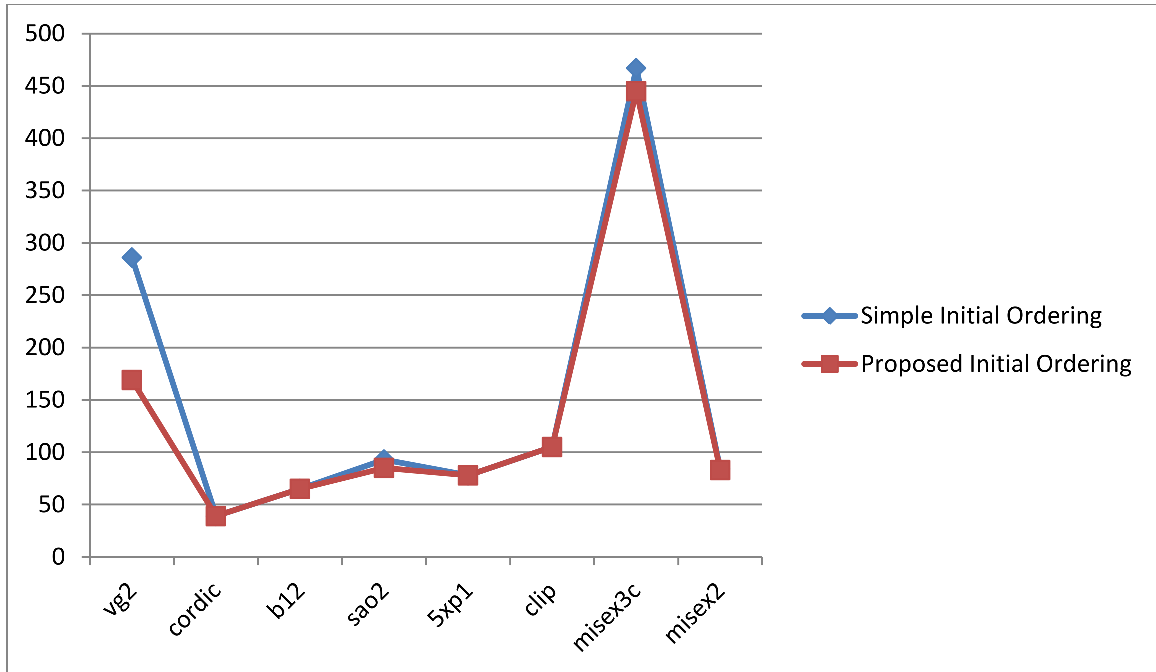


Figure 5.1 Comparison of BDD size using both initial ordering in sift algorithm.

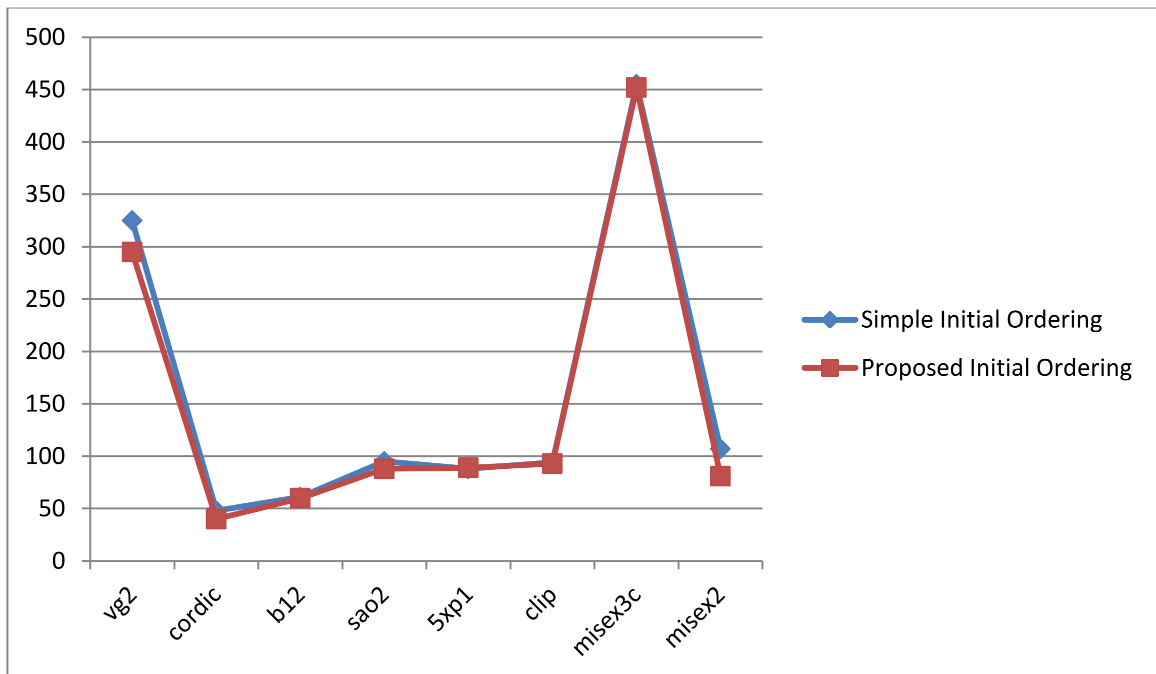


Figure 5.2 Comparison of BDD size using both initial ordering in win2ite method

In Figure 5.3 the comparison of BDD sizes of different adder circuits have shown. The ordering method used in Figure 5.3 is win2ite. The maximum improvement is for 5-bit adder circuit which has 11 input variables.

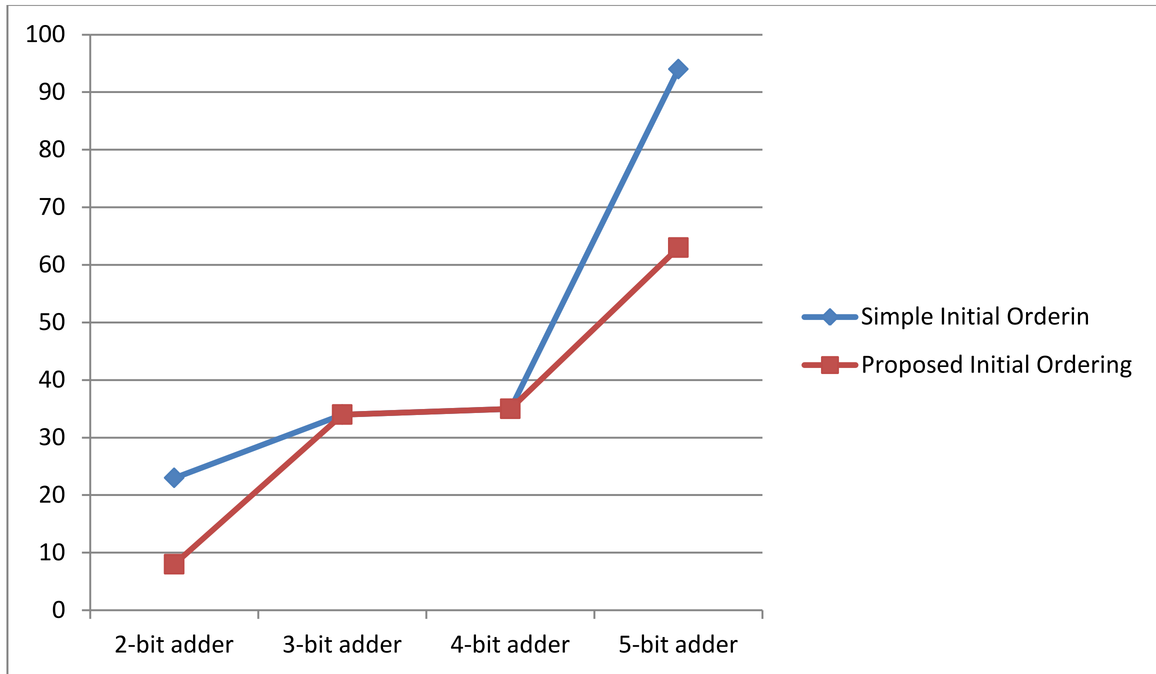


Figure 5.3 Comparison of BDD size of adder circuits using win2ite method

Area Calculation:

The total area is calculated by assuming that every node of BDDs is implemented by a 2×1 MUX. The area of a single 2×1 MUX is used [21]. The total area is then calculated by multiplying the area of a single 2×1 MUX to the total number of nodes in BDD. The total area is calculated for all the benchmark circuits from LGSynth93 using both the reordering methods. The average improvement is calculated by summing the area of all the benchmark circuits and taking average of it. The area of a 2×1 MUX calculated using Synopsis tool is shown in Table 5.4.

Table 5.4 Area of a 2×1 MUX using Synopsys [21]

```
****************************************
Report : area
Design : mux_2
Version: Y-2006.06-SP4
Date   : Thu May 24 12:24:56 2012
****************************************
Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a_c/
2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)

Number of ports:             4
Number of nets:              4
Number of cells:             1
Number of references:        1

Combinational area:       14.873600
Noncombinational area:     0.000000
Net Interconnect area:    undefined  (Wire load has zero net area)

Total cell area:          14.873600
Total area:               undefined

***** End Of Report **
```

The area of a 2×1 MUX is calculated using 180 nm technology. The total cell area of a 2×1 MUX is 14.8736 units. The total area of a circuit is calculated by assuming each node of the BDD as a 2×1 MUX and multiplying the area of the single 2×1 MUX to the total number of nodes in the BDD. The area of different benchmark circuits from LGSynth93 benchmark suite is tabulated in Table 5.5. In Table 5.5 area is calculated by using both the method of ordering then average improvement is calculated by taking improvement of all the circuits together. In Figure 5.4 the total average improvement is compared by using bar diagrams. The total area is scaled by 100 for better comparison. The total average improvement in the area is shown in percentage in between the bars.

Table 5.5 Area calculation of different benchmark circuits

Benchmark circuits **LGsynth93**	#i	#o	Area (in Units)			
			Sift		Win2ite	
			Simple Initial Order	**Proposed Initial Order**	Simple Initial Order	**Proposed Initial Order**
cordic	23	2	580.07	**580.07**	713.93	**594.94**
b12	15	9	966.78	**966.78**	907.29	**892.42**
5xp1	7	10	1160.14	**1160.14**	1308.88	**1323.75**
clip	9	5	1561.73	**1561.73**	1398.12	**1383.24**
vg2	25	8	4253.85	**2513.64**	4833.92	**4387.71**
sao2	10	4	1383.24	**1264.26**	1412.99	**1308.88**
misex3c	14	14	6945.97	**6618.75**	6767.49	**6722.87**
misex2	25	18	1249.38	**1234.51**	1591.48	**1204.76**
Total			18101.16	15899.88	18934.10	**17818.57**
% Improvement			**12.16 %**		**5.89 %**	

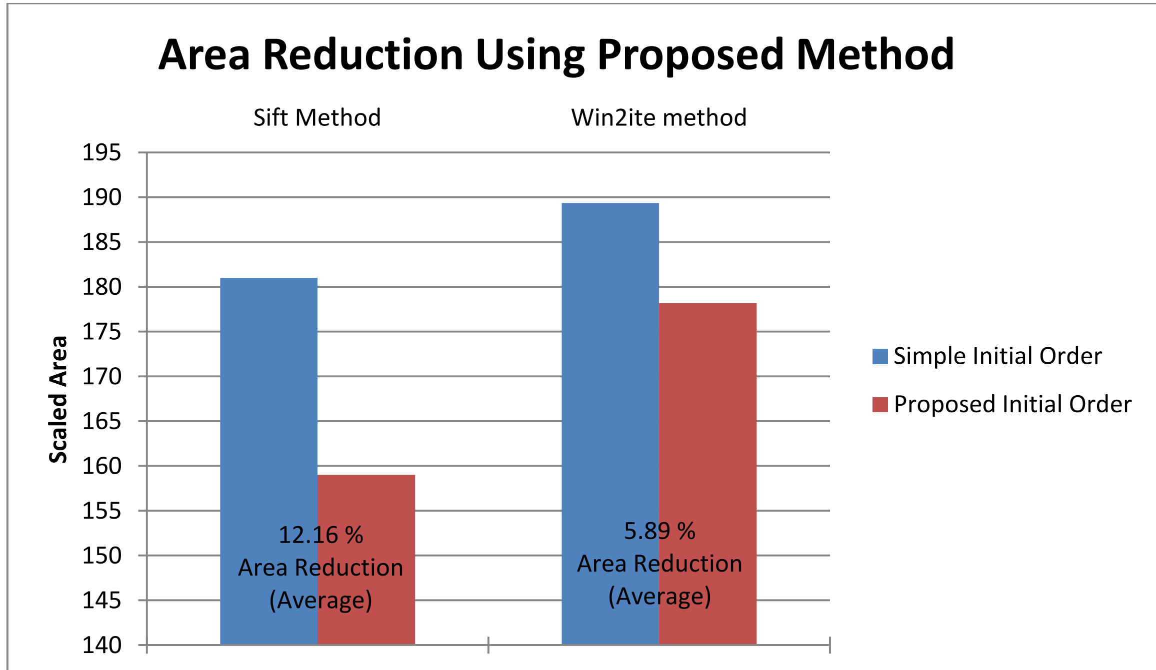


Figure 5.4 Comparison of average area of benchmark circuits using both the ordering method.

CHAPTER 6

CONCLUSIONS & FUTURE SCOPE

Symbolic model checking has proven to be a powerful paradigm to automatically verify real world applications. In this dissertation a use of the improved initial ordering in the existing BDD minimization techniques has been presented and found to achieve our objective of node minimization which in turn leads to area minimization.

In this paper a method has been proposed to find an improved initial order for existing reordering techniques of 'sift algorithm' and 'win2ite'. The improvement is good for both the circuits i.e. for the circuits which are having less input variables and the circuits having more input variables. The results are compared with the current use of a random simple initial ordering. The significant improvement is found in term of area by improving node count in BDDs.

The future works can use this method in other techniques with slight modification in finding the cost matrix. Since much attention at run time is not given in this work, the future work can be focused on run time.

LIST OF PUBLICATIONS

Md. Balal Siddiqui and Manu Bansal," BDD Ordering: A Method to Minimize BDD Size by Using Improved Initial Order," International Journal of VLSI and Embedded Systems (IJVES), Vol. 04, Issue 03, pp. 400-403, May 2013.

REFERENCES

1. C. Y. Lee, "Representation of Switching Circuits by Binary-Decision Programs," *Bell Systems Technical Journal*, Vol. 38, pp. 985–999, 1959.
2. R. E. Bryant, "Graph-based algorithms for Boolean function manipulations," *IEEE Transaction on Computers*, pp. 677-691, 1986.
3. G. Cabodi, P. Camurati, and S. Quer, "Improving the efficiency of BDD-based operators by means of partitioning," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 18, pp. 545-556, 1999.
4. S. I. Minato, "Binary Decision Diagrams and Applications for VLSI CAD," *Springer International Series in Engineering and Computer Science*, 1995.
5. J. R. Burch, E. M. Clarke and K. L. McMillan, "Sequential circuit verification using symbolic model checking," *IEEE Design Automation Conference*, pp. 46-51, 1990.
6. O. Brudaru, R. Ebendt and I. Furdu, "Optimizing Variable Ordering of BDDs with Double Hybridized Embryonic Genetic Algorithm," *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing,* pp. 167-173, 2010.
7. M. Fujita, Y. Matsunaga and T. Kakuda, "On Variable Ordering Of Binary Decision Diagrams for the Application Of Multi-Level Logic Synthesis," *Proc. of European Design Automation Conference*, pp. 50–54, 1991.
8. S. J. Friedman and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *IEEE Conference on Design Automation*, pp. 348-356, 1987.
9. M. Fujita, H. Fujisawa and Y. Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 12, pp. 6-12, 1993.
10. R. Rudell, "Dynamic variable ordering ,for ordered binary decision diagrams," *IEEE conference on CAD*, pp. 42-47, 1993.
11. R. Drechsler, B. Becker and N. Gockel, "Genetic algorithm for variable ordering of OBDDs," *IEEE Proceedings-Computers and Digital Techniques*, Vol. 143, pp. 364 –368, 1996.
12. H. Sakanashi, T. Higuchi, H. Iba and Y. Kakazu, "Evolution of binary decision diagrams for digital circuit design using genetic programming," *Lecture Notes in Computer Science,* Vol. 1259, pp 470-481, 1997.

13. P.W.C. Prasad, M. Raseen, A. Assi, A. Elchouemi and S. M. N. A. Senanayake, "Minimum Average Path Length in BDDs based on Static Variable Ordering," *IEEE Midwest Symposium on Circuits and Systems*, 2005.
14. S. Chaudhury and A. Dutta, "Algorithmic Optimization of BDDs and Performance Evaluation for Multi-level Logic Circuits with Area and Power Trade-offs," *Scientific Research*, Vol. 2, pp. 217-224, 2011.
15. W. N. Hung and X. Song, "BDD Variable Ordering By Scatter Search," *IEEE Conference on Computer Design*, pp. 368-373, 2001.
16. R. Ebendt and W. Gunther, "Combining Ordered Best-First Search With Branch and Bound for Exact BDD Minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, pp. 1657-1663, 2003.
17. B. Bollig, M. Lobbing, I. Wegener, "Simulated annealing to improve variable order in BDD," *International Workshop on Logic Synthesis*, 1995.
18. P. W. C. Prasad, A. Assi, A. Harb and V. C. Prasad, "Binary Decision Diagrams: An Improved Variable Ordering using Graph Representation of Boolean Functions," *International Journal of Electrical and Computer Engineering 2006*, Vol. 1, pp. 1-7, 2006.
19. F. Somenzi, "CU Decision Diagram Package Release 2.4.0," *University of Colorado at Boulder,* 2004.
20. J.Lind-Nielsen, "BuDDy, a binary decision diagram package," http://sourceforge.net /projects/buddy (online).
21. P. Singh, "Synthesis and Optimization of A 4-Bit Magnitude Comparator Circuit Using BDD and Pre-Computation Based Strategy for Low Power," M.Tech thesis, ECED, Thapar University, Patiala, 2012.
22. R. Drechsler, N. Drechsler and W. Giinther, "Fast Exact Minimization of BDDs," *IEEE Transaction on CAD*, pp. 384-389, June 1998.
23. J. Jain, W. Adams and M. Fujita, "Sampling Schemes for Computing OBDD Variable Orderings," *IEEE International Conference on Computer-Aided Design,* pp. 631-638, 1998.
24. E. Felt, G. Yorkt, R. Brayton, A. S. Vincentelli, "Dynamic Variable Reordering for BDD Minimization," Design Automation Conference, *European Design and Automation Conference,* pp. 130-135, Sep. 1993.

25. N. Zhuang, M. S. T. Benten and P. Y. K. Cheung, "Improved Variable Ordering Of BDDs With Novel Genetic Algorithm," *IEEE International Symposium on Circuits and Systems,* Vol. 3, pp. 414-417, May 1996.
26. R. Drechsler, "Using Lower Bounds During Dynamic BDD Minimization," *IEEE Conference on Design Automation*, pp. 29-32, June 1999.
27. W. Mingquan and Y. Haibin, "BDD Minimization Based on Genetic Tabu Hybrid Strategy," *International Conference on ASIC,* pp. 948-952, Oct. 2005.
28. U. Kuhne and N. Drechsler, "Finding Compact BDDs Using Genetic Programming," Springer *Applications of Evolutionary Computing*, pp. 308-319, 2006.
29. P. Buch, A. Narayan, A. R. Newton and A. S. Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits," *IEEE conference on computer aided design*, pp. 663-670, Nov. 1997.