IMPLEMENTATION OF REDUCED ORDERED BINARY DECISION DIAGRAM FOR FEATURE OPTIMIZATION

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

Degree of

MASTER OF TECHNOLOGY

(VLSI DESIGN & CAD)

Submitted by:

SUKHMANJEET KAUR

Roll No: 601161025

Under the guidance of:

Mrs. MANU BANSAL

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY
(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

July, 2013

DECLARATION

I hereby declare that the work which is being presented here in the dissertation entitled **"IMPLEMENTATION OF REDUCED ORDERED BINARY DECISION DIAGRAM FOR FEATURE OPTIMIZATION"** in partial fulfilment of the requirement for the award of degree of **Master of Technology** in **VLSI DESIGN AND CAD** 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 **Mrs. 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 my degree.

Dated: 05/07/2013

Sukhmanjeet Kaur

Sukhmanjeet Kaur

Roll No. 601161025

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

Dated: 05/07/2013

Mrs.Manu Bansal

Assistant Professor

ECED, Thapar University

Counter Signed By:

Head

ECED, Thapar University

Patiala-147004

Dean of Academic Affairs

Thapar University

Patiala-147004

ACKNOWLEDGEMENT

I would like to take the opportunity to express my gratitude to some people who were involved in this dissertation work. First, I owe my gratitude to my mentor Mrs. Manu Bansal for doing everything from the inception of the project idea to giving invaluable suggestions at every step.

I am also thankful to Dr. R. K. Khanna, Head of the Department and Dr. Kulbir Singh, PG Coordinator as well as all the faculty members and staff of the Department of Electronics and Communication Engineering for being very supportive to me. I would also like to thank all my batch-mates for motivating me all the time whenever I needed them and giving me useful tips. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would also 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.

Sukhmanjeet Kaur
601161025

ABSTRACT

The increasing speed and complexity of today's designs implies a significant increase in the power consumption of very-large-scale integration (VLSI) chips. To meet this challenge, researchers have developed many different design techniques to reduce power. For verification and checking of various hardware circuits, symbolic checking has been successfully applied and the Binary Decision Diagram representation is the core technology underlying this success and is one of the most popular data structures to represent Boolean functions. Boolean functions can be graphically manipulated to reduce the number of nodes, hence the area, when implemented as Binary decision diagrams. In BDD, the area and power consumption is determined by the total number of nodes. As number of nodes reduces, area reduces and hence it also reduces the power. A proper polarity selection of the sub-functions can not only reduce the number of BDD nodes, but also the switching activity. To optimized BDD, we have calculated the switching activity of the circuit.

Also, ordering of BDD nodes plays a very important role. Most of the algorithms for variable ordering of OBDD have focus on area minimization. Hence, for minimizing the power consumption, suitable input variable ordering is required. So, GA has been applied to find an optimal variable order that minimizes the size of BDD. Moreover, GA has been compared with various algorithms namely branch and bound algorithm, scatter search algorithm. Experimental results showed an average power reduction of 24.5 using GA and GA found to give best results in terms of power and accuracy.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

ASIC	Application Specific Integrated Circuits
BB	Branch and Bound
BDD	Binary Decision Diagram
CAD	Computer-aided design
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DVO	Dynamic Variable Ordering
EA	Evolutionary Algorithm
FPGA	Field Programmable Gate Arrays
GA	Genetic Algorithm
GUI	Graphical User Interface
IC	Integrated Circuit
LUT	Look Up Table
NC	Node Count
OBDD	Ordered Binary Decision Diagram
ROBDD	Reduced Ordered Binary Decision Diagram
RTL	Register Transfer Level
SA	Switching Activity
SOP	Sum of Product
VHDL	Very Hardware Description Language
VLSI	Very Large Scale Integration

CHAPTER 1
INTRODUCTION

1.1 BASIC INTRODUCTION

The design complexity and increasing speed of very-large-scale integration (VLSI) chips implies a significant increase in the power consumption. So, many different design approaches have been developed by researchers to reduce the power. The complexity of today's ICs, with over 100 million transistors, clocked at over 1 GHz, means manual power optimization would be hopelessly slow and all too likely to contain errors. Computer-aided design (CAD) tools and methodologies are mandatory. The advent of portable, wireless computation and communication devices imposes a very tight power budgets on the design to maximize the battery life, minimize the battery weight and allow usage of cost-efficient packaging technology. At the same time, real-time processing of audio and video data demands high computational power to meet the throughput requirements of sophisticated digital signal processing algorithms. Such algorithms typically imply high power consumption due to the number and complexity of basic operations executed. From the above, it should become clear that there is a strong necessity for minimizing power consumption when designing complex micro-electronic digital circuits and systems. Modern design methodologies offer designer tools support when taking a system specification down to a mask layout suitable for fabrication through a stepwise refinement process, across which the design is being described on different levels of abstraction. On each level through the refinement process, the designer can manually or automatically apply optimizing transformations to minimize the power consumption of the design [1].

Within the system design, digital design uses synthesis tools at the register transfer level that require a designer to explicitly include features of power reduction. Different approaches are there to reduce the power. One of them is to reduce the voltage supply or operating voltage of the circuit. The weight and cost of power supply almost depends on maximum possible power used at some instant. These are guided by a global cost function that evaluates the current configuration with respect to user-specified objectives on area, delay, and now power consumption also. The current trend towards low-power design is

mainly driven by the growing demand for long-life autonomous portable equipments, and the technological limitations of high-performance VLSI systems. For the first category of products, low-power is the major goal for which speed and dynamic range might have to be sacrificed. High speed and high integration density are the objectives for the second application category. The most efficient way to reduce power consumption of digital circuits is to reduce the supply voltage. Power optimization approaches at the High-level are significant since research results indicates that higher levels of abstraction have greater potential for power reduction [2] as shown in figure 1.1

Earlier, the major factors of the VLSI designer were area, cost, performance and reliability, power considerations were only of secondary importance. But now, this has been changed and power is being given equal importance in comparison to area and performance.
Lots of factors have contributed to this trend. This is mainly due to the remarkable and huge success of personal computing devices which demands very high speed computations and complex functionality with low power consumption.

1.2 Reduction Of Power At Different Abstraction Levels

Limits discussed so far have been fundamental since they do not depend on the technology or the choice of power supply voltage. However, there a number of obstacles or technological limitations are in the way to approaching these limits in practical circuits and ways to reduce the effect of these various limitations could be found at all levels of digital design ranging from device to system. Battery powered systems such as laptop, computers, electronics devices etc. the need of these systems arise from the need to extend battery life. Low power design is not only needed for portable applications but also to reduce the power of high performance systems with large integration density and improved speed of operation.
Emerging of portable computing and communication equipment such as laptop, palmtops, cell-phones, etc growth rate of these portable equipments are very high. Following are different levels of abstraction:

1. **System level**- The system is described in terms of a set of hardware, software and memory components and interacting algorithms they perform to provide certain functionality.
2. **Behavioral or architectural level-** The individual components such as ASICs, data paths, software processes are described in the terms of their algorithmic behavior, typically in a hardware description languages, such as behavioral VHDL, or a high-level programming language such as C. Typically, there is no timing, but only causal information is available on

this level and the interface protocols controlling the communication between the interacting processes has been already fixed.

3. **Register-transfer level**- In this, hardware is described in terms of arithmetic modules, registers and multiplexers and interconnects to steer data flow. RT level representations describes the basic building blocks of a system and their interconnects on clock-cycle level; it specifies the “micro-architecture" of the system under consideration.

4. **Gate-level**- A circuit is described in terms of a net list of gates or a set of Boolean equations reflecting its functionality.

5. **Transistor level**- A circuit is described in the terms of its transistor network structure.

6. **Physical level**- A circuit is described in terms of its mask layout as it will be used for fabrication. The basic building blocks on this level are polygons which reflect different materials used for fabrication such as metal, polysilicon, oxide etc.

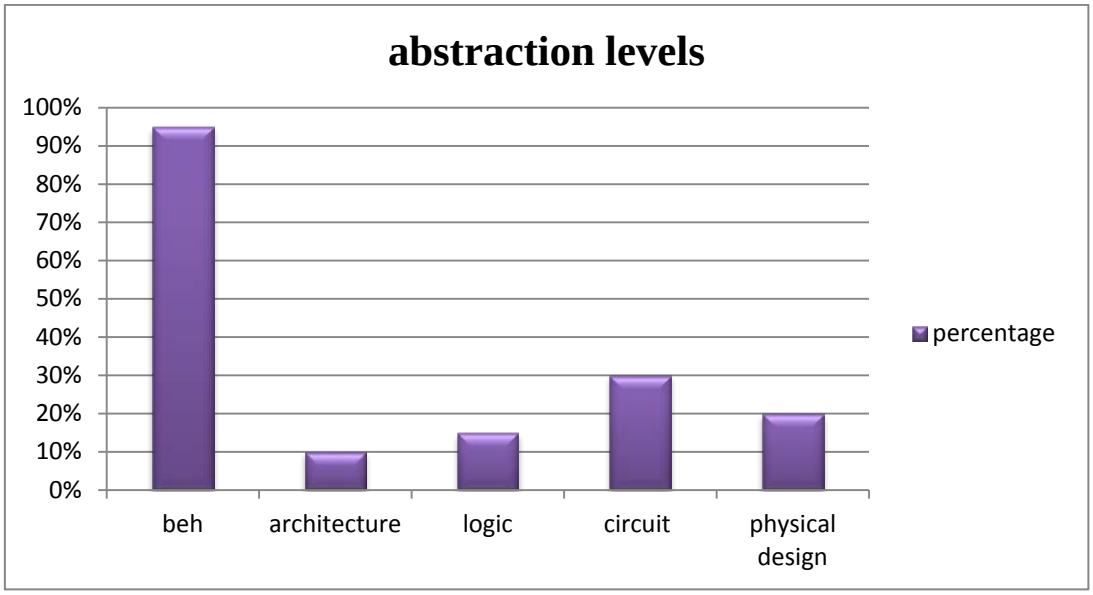


Figure1.1 Bar graph showing different abstraction levels

At system level, the first decision to be made on system level is that of algorithm selection, i.e. selecting, from a set of given algorithms, one that provides the required functionality of the system under design while best meeting design constraints. The power consumption induced by an algorithm depends on its characteristics in terms of overall complexity, complexity of the basic operations, communication requirements etc. Although the selected algorithm itself can later be optimized, subsequent optimizations are typically of a local scope and will not be able to compensate the differences in power consumption across

different algorithms, which can be up to several orders of magnitude. One of the most important factors to be taken into consideration during algorithm selection is that of memory access and management.

Behavioral level power optimization is performed during behavioral synthesis, i.e. when mapping an algorithm onto a hardware register-transfer level description. We can distinguish between optimizations which operate on the behavioral description alone and those which optimize the mapping of operations onto hardware units. Optimizations in register-transfer level operate by structurally transforming RT level net lists.

There is a general consensus that design decisions on higher levels of abstraction have the most significant impact on the overall structure and quality of the resulting design. For instance, choosing different partitioning of functionality on design components on the system level can result in vastly different requirements on the characteristics of the individual components in terms of area, performance or power. Accordingly, optimizing transformations on higher levels of abstraction have the largest impact on the power consumption of the design as a whole. It is therefore desirable to take power into account as a cost function as early as possible in the design flow, i.e. on system and behavioral level, in the design flow, since in the later stages optimizations will typically only be of a very local scope, and hence of limited effectiveness [1].

1.2.1 Contribution Of Several Power Dissipations

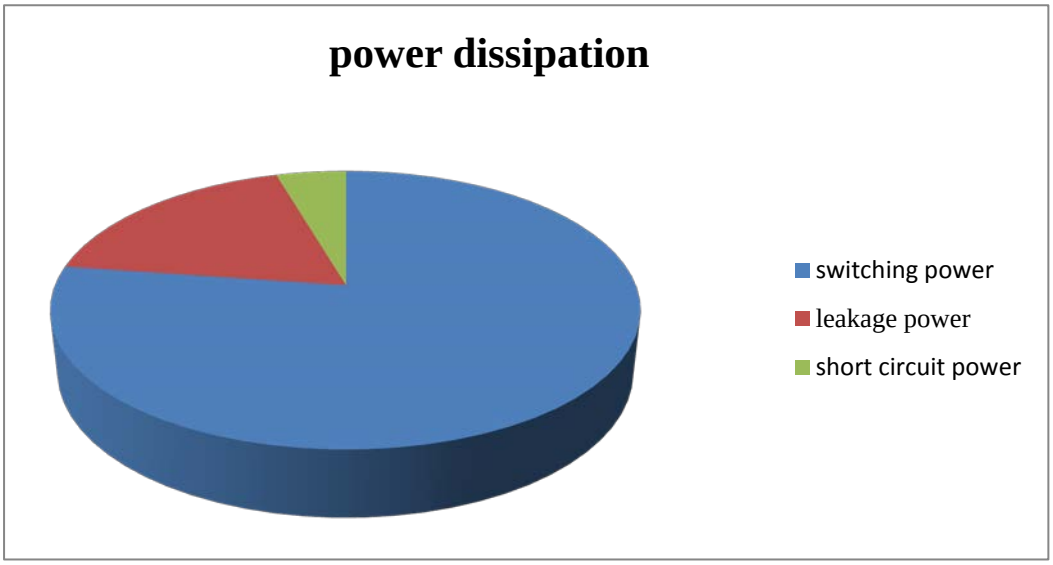


Figure 1.2 Pi chart showing several power dissipations

As these are battery operated, battery life is of primary concern. Unfortunately, the battery technology has not kept up with the energy requirement of the portable equipment. Commercial success of these products depends on weight cost and battery life.

1.3 WHY BDD?

Many problems in digital logic design and testing can be expressed as a sequence of operations on Boolean functions. Such applications would benefit from efficient algorithms for representing and manipulating Boolean functions symbolically. A variety of methods have been developed for representing and manipulating Boolean functions. Those based on classical representations such as truth tables, Karnaugh maps, or canonical sum-of-products forms are quite impractical. Since, every function of 'n' arguments has a representation of size $2n$ or more. More practical approaches utilize representations that at least for many functions, are not of exponential size. Example representations include as a reduced sum of products [3]. These representations suffer from some drawbacks. First, certain common functions still require representations of exponential size. For example, an even and odd parity functions serve as worst case examples in all of these representations. Second, while a certain function may have a reasonable representation, performing a simple operation such as complementation could yield a function with an exponential representation. Finally, none of these representations are in canonical forms, i.e. a given function may have many different representations. Consequently, testing for equivalence or satisfiability can be quite difficult.
Due to these characteristics, most programs that process a sequence of operations on Boolean functions have rather erratic behaviour. So, a new class of algorithms for manipulating Boolean functions represented as directed acyclic graphs called Binary decision diagram were introduced by Lee [4] and further popularized by Akers [11].

1.4 BDD INTRODUCTION

Binary Decision Diagrams (BDDs) are the data structures that are used to represent Boolean functions or are a direct acyclic graph representation of Boolean function introduced by Lee [4] and Akers [11]. Boolean function can be represented as a rooted, directed or acyclic graph which consists of several decision nodes and terminal nodes. Also, Boolean functions when implemented as BDDs can be manipulated to reduce the number of nodes [26].

The basic idea from which this data structure was created is the Shannon expansion. A Switching function is split into two sub-functions by assigning one variable. If such a sub-function is considered as a sub-tree, it can be represented by a binary decision tree.

In the field of synthesis and verification of digital VLSI circuits, this approach has attracted many researchers. Ability and efficiency of BDD make it popular in representing a variety of functions. The efficiency depends on the size of BDD and that is the size of graphical representations. This size depends on the variable ordering selected to build the BDD. So to find an optimal variable order is the most tiring job in OBDDS and this leads to area and power minimizations [5].

Another critical parameter while constructing of BDDs is the maximum memory requirement, which is directly proportional to number of nodes. A good variable ordering can lead to a smaller BDD and faster runtime, whereas a bad ordering can lead to an exponential growth in the size of the BDD and hence can exceed its available memory. Similarly, much attention has been devoted to techniques dedicated to finding good variable ordering. The evaluation time is also another important parameter, which uses BDDs to evaluate various logic functions. The evaluation time is proportional to the path-length in the BDD. Therefore, minimizing the path length can improve the performance of the circuit implementing a Boolean function, which will automatically enhance the performance of the final implementation. Normally, minimizing the path length in Decision Diagrams is important in database structures, pattern recognition, logic simulation and software synthesis [6]. The most successful applications for BDDs have been made in the automatic formal verification: equivalence checking for combinational circuits, equivalence checking for sequential circuits, or temporal logic model checking of sequential systems. Other applications include logic synthesis and test pattern generation.

1.5 TYPES OF BDD

1.5.1 OBDD

Ordered binary decision diagram (OBDD) was proposed by Bryant. An ordered binary decision diagram (OBDD) is a BDD with the constraint that the input variables are ordered and every source to sink path in the OBDD visits the input variables in ascending order.

Basically, OBDD is a restricted decision graph in which the ordering of variables is consistent on all paths of the graph. Because the BDD is ordered, the DAG can be levelized with all nodes with a particular top variable at a given level. With this ordering restriction, sub-graphs can be shared and the resulting diagram remains canonical. That is, the diagram is unique for a given logic function.

1.5.2 ROBDD

A reduced ordered binary decision diagram (ROBDD) is an OBDD where each node represents a distinct logic function. ROBDDs are based on a fixed ordering of variables and have additional property of being reduced. This means:

Irredundancy: The low and high successors of every node are distinct.

Uniqueness: There are no two distinct nodes testing the same variable with the same successors. Bryant was the first to prove that the ROBDD is well-defined. Bryant also showed the ROBDD is a canonical form for a logic function; that is two functions are equivalent if, and only if, the ROBDD'S for each function are isomorphic. In other words, we can compare Boolean functions by constructing their ROBDDs and checking if they are equal. ROBDDs have at most two leaf nodes, labelled 0 and 1. We sometimes draw them multiple times to avoid complex edges leading to them. [13]

1.6 OPTIMIZATION OF BINARY DECISION DIAGRAM

There are two types of terminal nodes called 0-terminal and 1-terminal. Have two successors indicated by descending lines. One of the successors is drawn as a dashed line, called 'low' and other is drawn as a solid line, called 'high'. These branch nodes define a path in the diagram for any values of Boolean variables. The '0' and '1' nodes also called the sink node. If low branch is being followed from the root, then that path will reach to sink node '0' and if high branch is being followed, then the path will reach to sink node '1'. **The BDD obeys two important restrictions**. First, it must be ordered. Second, a BDD must be reduced, in the sense that it doesn't waste space [7]. BDD's are well-known and widely used in logic synthesis and formal verification of integrated circuits. Due to the canonical representation of Boolean functions they are very suitable for formal verification problems. [8, 9]

If different variables appear in the same order on all paths from the root, such a BDD is called 'ordered'. A BDD is said to be 'reduced', if the following two rules have been applied to its graph and Reduction consists of the application of the following two rules starting from the decision tree and continuing until neither rule can be applied.

1. If two nodes are terminal and have the same label, or are internal and have the same children, they are merged.

2. If an internal node has identical children it is removed from the graph and its incoming nodes are redirected to the child. [10]

BDDs have been used in many tasks in VLSI such as equivalence checking, property checking, logic synthesis, and false paths. Synthesis, verification, and testing algorithms of VLSI circuits manipulate large number of switching functions. Therefore it is important to have efficient methods to represent and manipulate such functions. A huge class of problems in the area of VLSI CAD can be solved by adopting efficient underlying data structures. During the last decade, several methods based on Decision Diagrams have been proposed and successfully used in many industrial applications. Recently, Binary Decision Diagrams have emerged as one of the best representation methods for wide range of applications**.**

Since, BDD is relatively an old technique, but its advantages as canonical representations was only recognized and made clear by Bryant. In contrast to other techniques, Boolean functions based on BDD request the least space and hence have great computation efficiency. Hence, it is widely employed in EDA industry and academic research such as logic synthesis and optimization. For BDD, its size is a key factor to decide whether it is possible to utilize the symbolic technique based on BDD. There are many techniques for the reduction of the number of nodes [5]. It is shown that the BDD size is closely related to variable orders. The node number of reduced ordered Binary decision diagram (ROBDD) for a function is likely of linear and exponential difference [11].

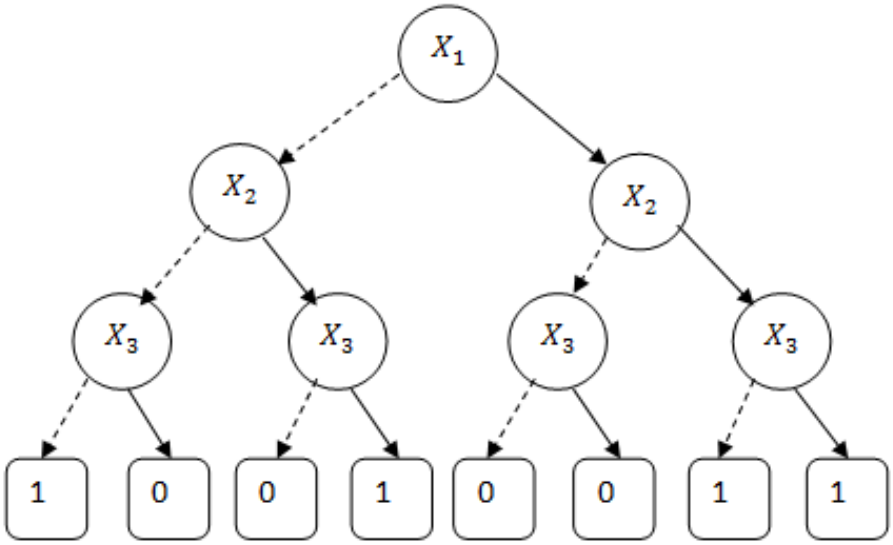
1.6.1 BINARY DECISION DIAGRAM FROM A TRUTH TABLE

The Figure 1.3 (a) below shows a binary decision tree without the reduction rules applied and a truth table, each representing the function Y (X_1, X_2, X_3).

$$Y(X_1, X_2, X_3) = \bar{X}_1 \bar{X}_2 \bar{X}_3 + X_1 X_2 + X_2 X_3$$

In the tree, the value of the function can be determined for a given variable assignment by following a path down the graph to a terminal. In the figures below, dotted lines represent edges to a low child, while solid lines represent edges to a high child. Therefore, to find ($X_1 = 0, X_2 = 1, X_3 = 1$), begin at X_1, traverse down the dotted line to X_2 (since X_1 has an assignment to 0), then down two solid lines (since X_2 and X_3 each have an assignment to one). This leads to the terminal 1, which is the value of Y ($X_1 = 0, X_2 = 1, X_3 = 1$).

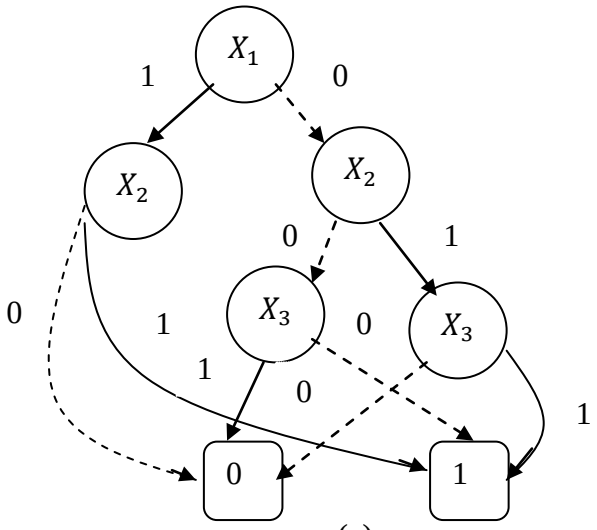
The binary decision tree of the figure 1.3 (a) can be transformed into a binary decision diagram by reducing it according to the two above mentioned reduction rules. The resulting BDD is shown in the right figure 1.3 (c).



(a)

X_1	X_2	X_3	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(b)



(c)

Figure 1.3 Binary Decision Tree from a Truth Table (a) Simple Binary decision tree (b) Truth table for the function and (c) BDD for the function Y [12]

Most of the times, the term BDD almost refers to Reduced Ordered Binary Decision Diagram (ROBDD when the ordering and reduction aspects need to be emphasized). The advantage of an ROBDD is that it is canonical (unique) for a particular function and variable order. This property makes it useful in functional equivalence checking and other operations like functional technology mapping. A path from the root node to the 1-terminal represents a

(possibly partial) variable assignment for which the represented Boolean function is true. As the path descends to a low (or high) child from a node, then that node's variable is assigned to 0 [12].

1.6.2 BINARY DECISION DIAGRAM AND BDD USING GATES

Assume that there are Boolean variables from a_1 to a_n making up the input to a function. At the root node, we test one of the variables, let it be a_1, we will have two subtrees, one for the case where a_1= 0 and one where a_1= 1. Each of the subtrees is now testing another variable, each with another two subtrees, and so on. At the leaves, we have either 0 or 1, which is the output of the function on the inputs that leads to the path from the root to the leaf. For example, for various gates, like And Gate, we can write equation as:

bool and (bool a_1, bool a_2) { return a_1 && a_2; }

In mathematical notation, we usually write $a_1 \wedge a_2$. In the diagram below for $a_1 \wedge a_2$, we indicate the 0 on left branch with a dashed line and the 1 right branch with a solid line.

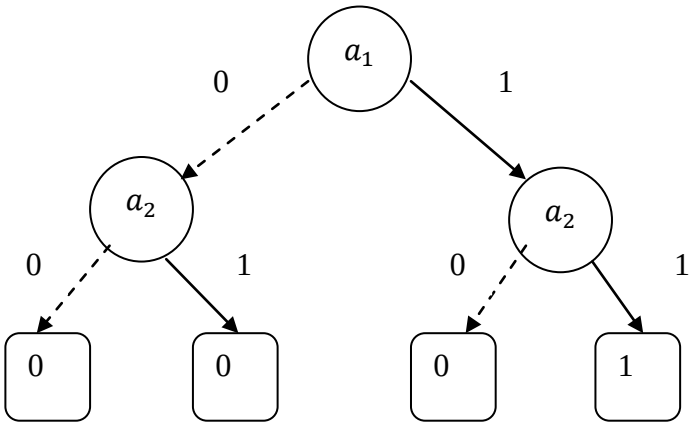


Figure 1.4 Binary Decision Tree for AND-gate [13]

Similarly, for OR gate, we can write as bool or (bool a_1 , bool a_2) { return a_1 || a_2; }

In mathematical notation, we can write as $a_1 \vee a_2$. As a binary decision tree, it would be as shown in figure 1.5.

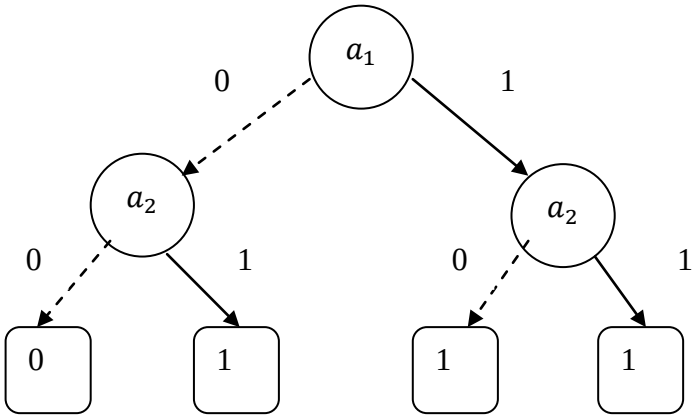


Figure 1.5 Binary Decision Tree for OR-gate [13]

But for Exclusive-Or, we cannot write $a_1 \wedge a_2$, because the Exclusive-Or operator is an operation on arguments of type int not bool. Mathematically, it is often written as $a \oplus a_2$. Nevertheless, it is easy to define.

bool xor (bool a_1, bool a_2) { return (a_1 && ! a_2) || (! a_1 && a_2); }

Binary decision trees possess some nice properties, but also have some drawbacks. The biggest problem is their size. A binary decision tree of 'n' variable will have 2_n - 1 decision nodes, and 2_n links at the lowest level, pointing to the return values 0 and 1. One of the good properties of Binary Decision Trees is Canonicity. If we test variables in a fixed order from a_1 to a_2, then the binary decision tree is unique. We can therefore test the equivalence of two Boolean functions by comparing their binary decision trees for equality. However, if it is implemented in a native may, this equality test is exponential in number of variables, because we have to compare the size of the two trees. If the order of the variables is fixed, Decision Trees are said to be ordered. Ordered binary decision trees are isomorphic to binary tries storing Booleans at the leaves. [13]

1.6.3 DECISION DIAGRAMS FROM DECISION TREES

Binary Decision diagrams (BDDs) can be distinguished from binary decision trees in two ways. First, they allow redundant test of Boolean variables to be omitted and secondly, they allow sharing of identical subtrees.

For example, in the tree for $a_1 \wedge a_2$, both branches in the test of a_2 on the left lead to 0, so there really is no need to test a_2 at all. We can simplify this to the following BDD as shown in figure 1.6

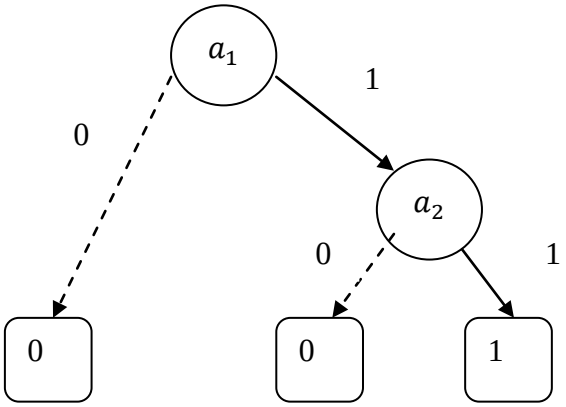


Figure 1.6 Binary Decision Diagram for AND- gate [13]

If the inputs have odd parity, that it, an odd number of 1's, then the function returns 1 and it returns 0 if the inputs have even parity, that is, an even number of 1's. Rather than 15 nodes and 2 (or 16) leaves, it only has 7 nodes and 2 leaves, exploiting a substantial amount of sharing [13]. It is shown in the figure 1.7.

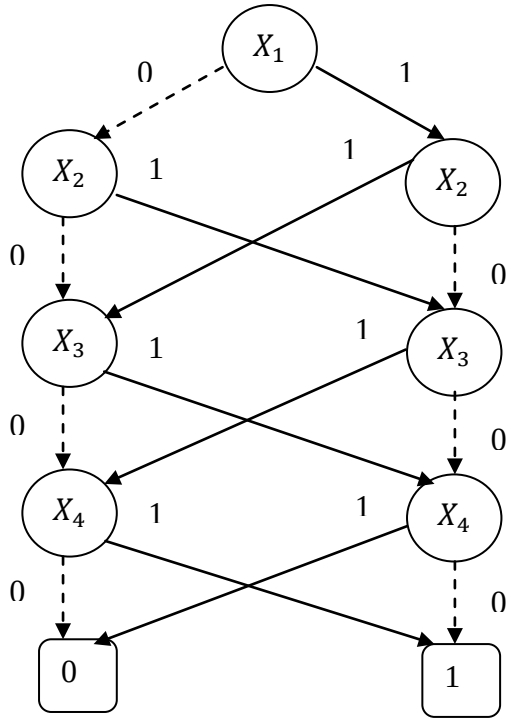


Figure 1.7: Sharing of Identical Subtrees [13]

In the worst case, the size of a BDD will still exponential in the number of variables, but in many practically occurring cases it will be much smaller. For example, independently of the number of variables, the constant function returning 0 can be represented by just a single node, as can the constant function returning 1 [13].

1.7 SOME EXAMPLES USING SWITCHING FUNCTION

As discussed above, BDD is a form of realization of multi-level logic and is resulted from direct implementation of Shannon's Expansion of a logic function. Also, each node in a BDD tells and signifies some logic function and a decision is made at each node and determines either there is a zero value or one at each node of its leg. This continues until the tree diagram reaches its leaf, Each BDD node can be easily realized using multiplexer. Thus, minimization of total number of nodes of a BDD means minimizing the number of variable present in the function, hence reduction in area [14]. A proper polarity selection of the sub-functions reduces not only the number of BDD nodes, but also the switching.

Consider the switching function,

y= $P \vee \bar{Q}R$

And assume we are interested in defining a procedure for determining the binary value of y given the binary values of P, Q, and R. One way to do this would be to begin by looking at the value of P. If P = 1, then y= 1 and we are finished. If P = 0, we look at Q. If Q = 1, then y= 0 and again we are finished. Otherwise, we look at R and its value will be the value of y. Fig.1.8 shows a simple diagram of this procedure. [14]

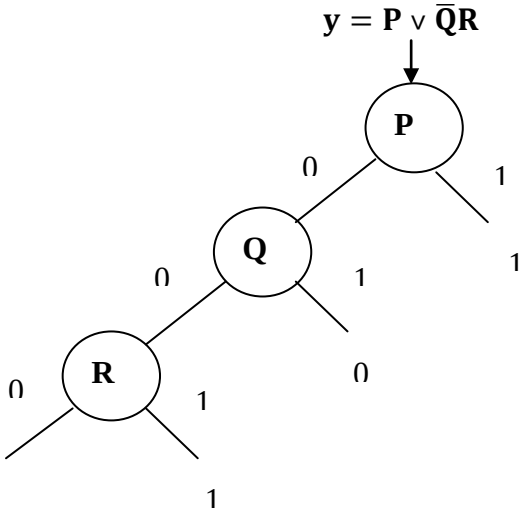


Figure 1.8 Representation of BDD for the function P + $\bar{Q}$R [14]

We enter at the node indicated by the arrow and then simply proceed downward through the diagram, noting at each node the value of its variable and then taking the indicated branch. When a 0 or 1 value is reached, this gives the value of y and the process ends.

Fig. 1.9 shows similar diagrams for some simple AND, OR, and XCLUSIVE-OR functions.

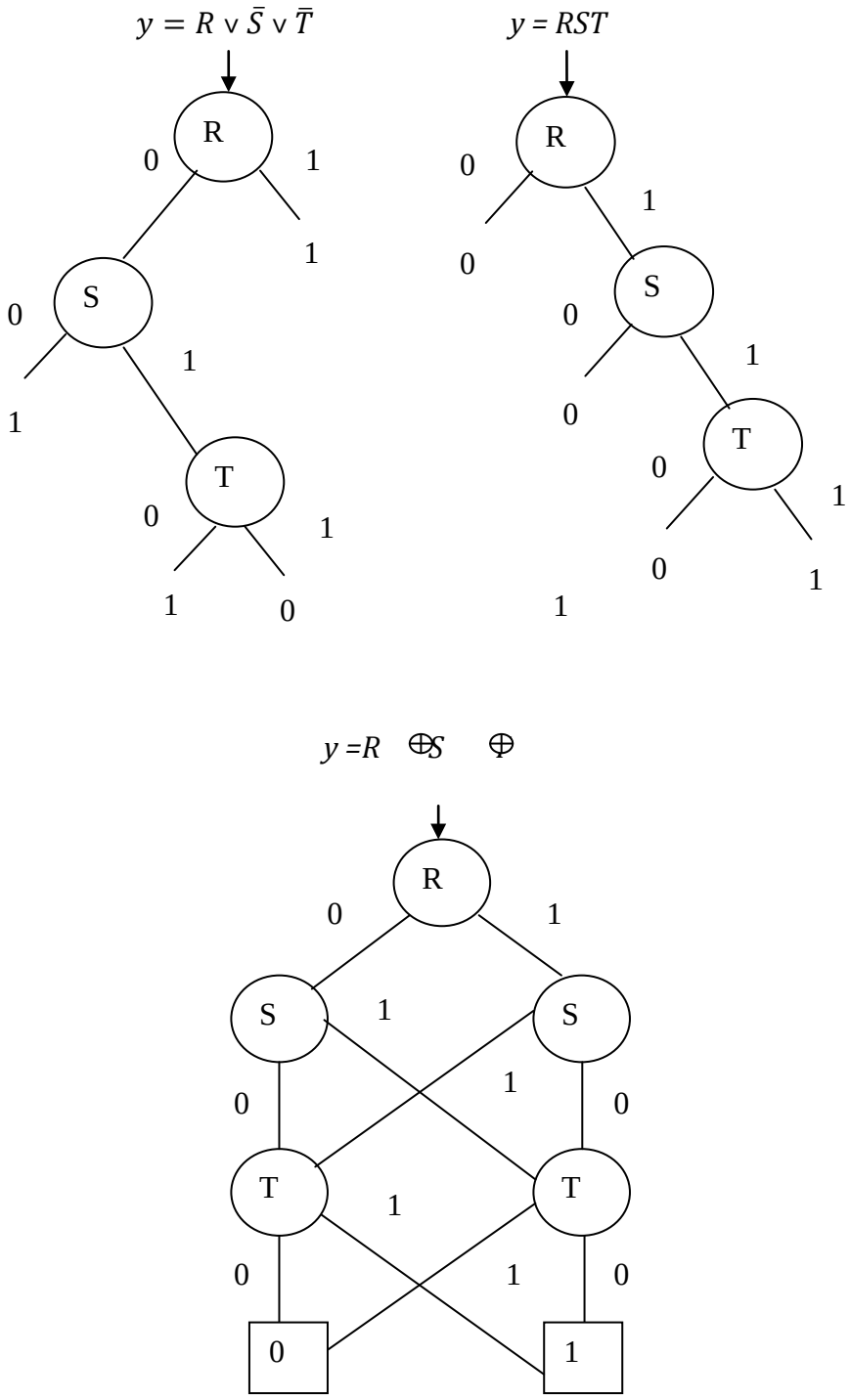


Figure 1.9: BDD representation for AND, OR and XOR [14]

Before examining some of the properties and uses of these diagrams, let us look at how they may be derived. A number of procedures are possible depending on the form in which the function y is defined. If, for example, we are given nothing more than a truth table for y, then a simple but possibly cumbersome procedure is to construct a diagram such as that shown in Fig. 1.10 which has a one-to-one correspondence between the 2" rows of the table

and the 2^n paths to the outputs of the diagram. These outputs may then be labeled with the corresponding binary values of y and the required diagram automatically results [14].

P	Q	R	y
0	0	0	a
0	0	1	b
0	1	0	c
0	1	1	d
1	0	0	e
1	0	1	f
1	1	0	g
1	1	1	h

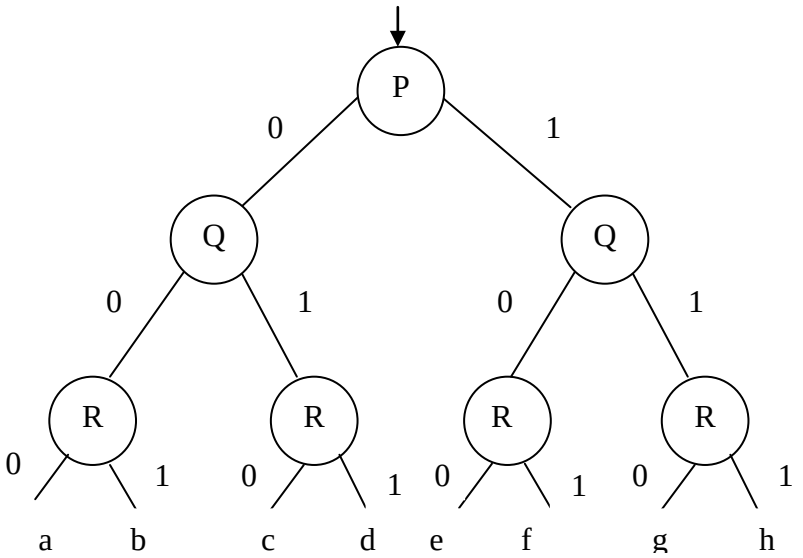
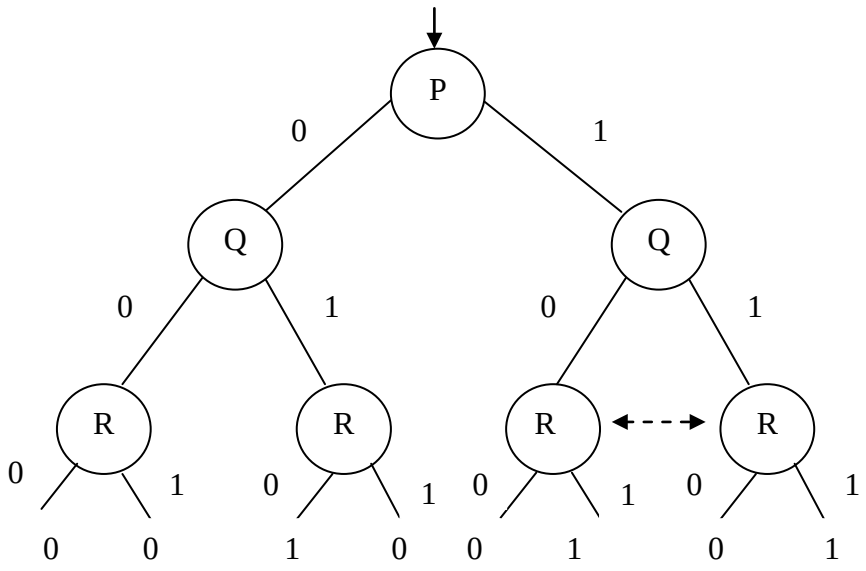


Figure 1.10: Driving a diagram from truth table [14]

With 'n' variables, there will initially be 2^n -1 nodes in such a diagram. There are, however, several obvious ways in which this number can be considerably reduced. Consider the diagram in Fig.1.11 (a) which results from the truth table for $y = \overline{P}Q\overline{R} \vee PR$.

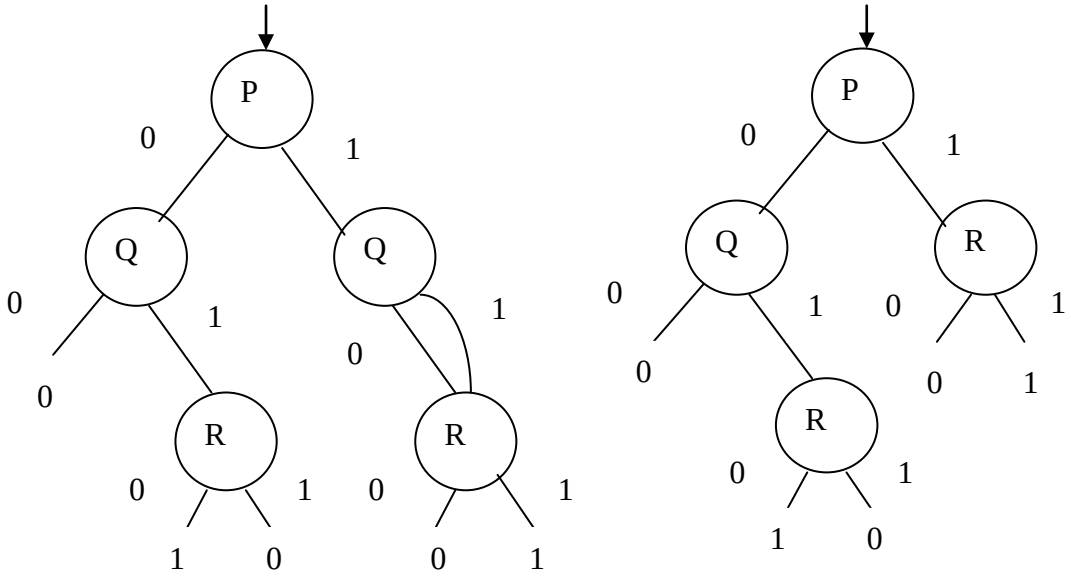
$$y = \overline{P}Q\overline{R} \vee PR$$



(a)

Figure 1.11: Optimization of BDD of function y [14]

We note that the value of y obtained at the leftmost R-node is 0 regardless of the value of R. Accordingly; we can remove this node and replace it by 0. Likewise, the two rightmost R-nodes are identical in the sense that they lead to identical output values, so we can combine them into a single node. The result is Fig. 1.11(b). But now we note that the rightmost Q-node is superfluous, since both of its branches go to the same node. Thus, we can remove it to obtain the simplified diagram of Fig. 1.11(c)



(b) **(c)**

Figure 1.11: Optimization of BDD of function y [14]

Normally, switching functions are specified in more compact forms than that afforded by the truth table with its 2^n values. One of the most common is as a Boolean expression. In this case, a "top-down" procedure can be used to derive the diagram by repeated applications of the classical Shannon expansion formula:

$$y(J, K, L, \ldots) = Jy(1, K, L, \ldots) \vee \bar{J}y(0, K, L \ldots)$$

Fig. 1.12 shows this procedure for the 5-variable function,

We begin by setting J = 0 in y to obtain the function $y_0$which must be realized below the J = 0 branch. We then do the same for J = 1 to obtain y_1 as shown in Fig. 1.12(a). Now the process is repeated for variable K to obtain the four functions in Fig. 1.12(b). Now we may note that two of the branches lead to the same function ($M\bar{N}$) so these may be directed to the same node.

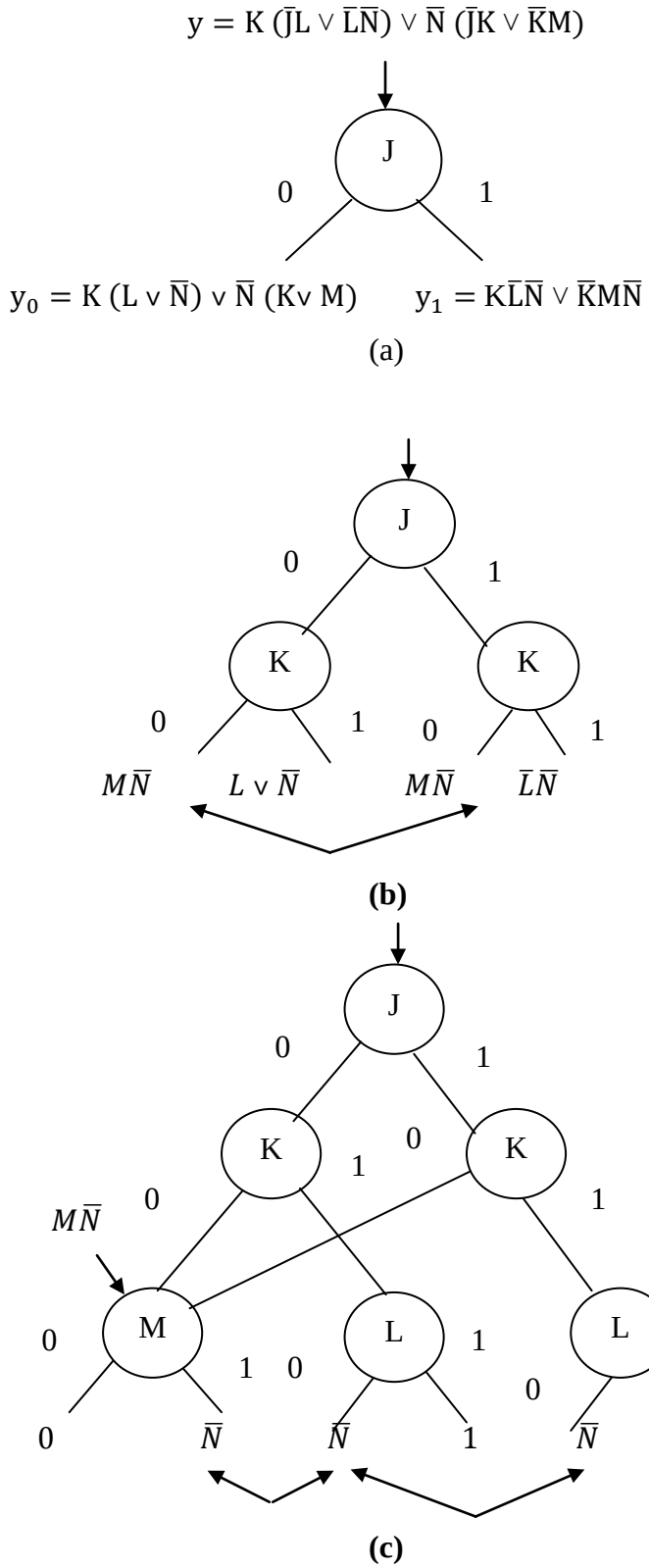
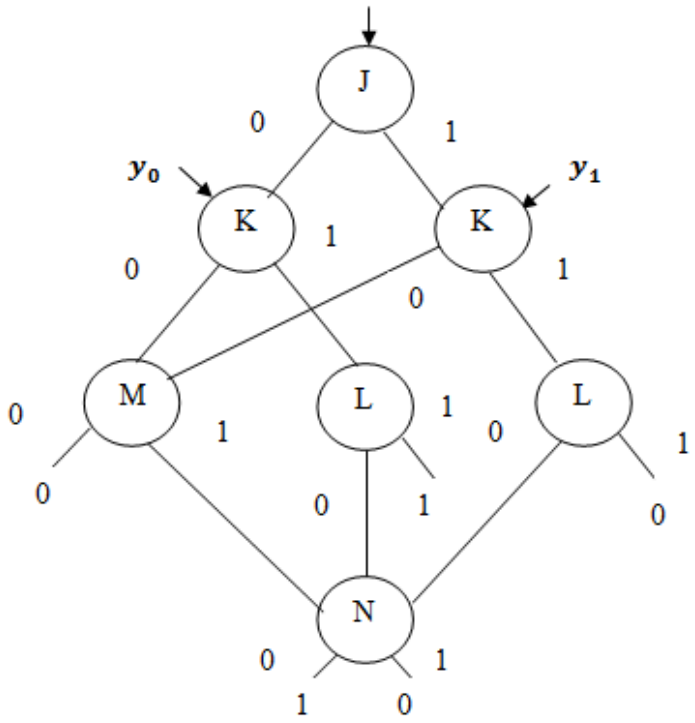


Figure 1.12 BDD representations for 5- Variable function [14]

We continue in this fashion merging identical sub functions, and then expanding each about one of its remaining variables, until all paths have terminated with a 0 or 1. Clearly, all paths

will terminate in at most n steps. Note that in the process of deriving this diagram, we have actually obtained the diagrams for a number of functions. If, for example, we enter the diagram of Fig. 2.2.5(d) at the node indicated by y_0, we will exit with the value of y_0.If we enter at the y_1-node, we will exit with the value of y_1 etc.



(d)

Figure 1.12 BDD representations for 5- Variable function [14]

Thus, we can use a single diagram to specify a number of functions depending on the node at which we enter the diagram. If, in fact, we had initially wanted to derive a diagram for the two functions, y_0 and y_1, we would have followed the same steps described above, beginning at Fig. 2.2.5(b) and ending with the diagram of Fig. 2.2.5(d) (with the J-node missing). Thus, this same procedure can be used to derive a diagram for an arbitrary number of functions.

1.8 APPLICATIONS OF BDD

BDDs are extensively used in CAD software to synthesize circuits and in formal verification. There are several lesser known applications of BDD, including Fault tree analysis, Bayesian Reasoning and Product Configuration. Every arbitrary BDD (even if it is not reduced or ordered) can be directly implemented by replacing each node with a 2 to 1 multiplexer, each

multiplexer can be directly implemented by a 4-LUT in a FPGA. It is not so simple to convert from an arbitrary network of logic gates to a BDD.

1.9 ORGANIZATION OF DISSERTATION

The main objective of this dissertation is the BDD based simulation of a digital circuit (combinational circuit) for power reduction. In BDD, the area and power consumption is determined by the total number of nodes. As number of nodes reduces, area reduces and hence it also reduces the power. A proper polarity selection of the sub-functions can not only reduce the number of BDD nodes, but also the switching activity. To optimized BDD, we have calculated the switching activity of the circuit. Moreover, GA is applied to the circuit to check the accuracy of the circuit.

The dissertation embodies following objectives:

1. To Study and implementing digital functions by means of Binary Decision Diagrams.
2. To design and implement an N- bit adder circuit over the Binary Decision Diagrams.
3. To find out the total number of bits involved into this contradiction.
4. To reduce the number of nodes and number of paths of this N-Bit adder circuit by implementing Reduced Ordered Binary Decision Diagram (ROBDD) for the circuit and calculate the Switching Activity (SA) of the reduced circuit.
5. To calculate average power reduction that we have optimized.
6. To implement Genetic Algorithm for the same and compare the GA with other techniques such as Branch and bound, scatter search and Dynamic variable ordering techniques.

Chapter 2 contains the literature review of papers related to the work. It contains the literature review of papers used in minimizing power consumptions in digital circuits. Works related to the optimization of BDD using GA to achieve low power are also discussed. Also contains paper used for ordering of BDD.

Chapter 3 discusses about our purposed work and methodology used. Also discusses how to calculate the number of paths and switching activity of a circuit. It also includes some important terms related to GA.

Chapter 4 discusses various variable ordering algorithms such as static variable ordering, dynamic variable ordering and evolutionary algorithms.

Chapter 5 discusses simulated results of the N-bit adder circuit to which GA is applied on MATLAB. It contains the results of optimized average power reduction and comparison of GA with other ordering algorithms.

Chapter 6 contains conclusions and future scope of the work.

CHAPTER 2
LITERATURE REVIEW

In this chapter, work done by many earlier researchers has been presented and discussed related to our topic.

S. B. Akers [14] presented a method for defining, analyzing, testing and implementing large digital functions by means of a binary decision diagram. This diagram explains the description of digital functions and provides a complete, concise, "implementation-free" description of the digital functions involved. In this paper, different methods have been described to derive the diagrams and examples have been given for various combinational and sequential circuits. Also, an example of carry look-ahead adder has been included. Also, those methods for deriving the diagrams are modified and extended so that a minimal number of nodes results.

S. Malik et al. [15] Binary decision diagrams as presented by Bryant were canonical representations for Boolean functions and for formal logic verification. However, the size of BDDs was sensitive to variable ordering. So, he proposed two variable ordering techniques which were based on network topology. He has presented that binary decision diagrams with automatic variable ordering were applicable for formal verification of a very large class of circuits. Also, this method proved significantly faster on benchmark circuits.

M. Fujita et al. [16] presented variable ordering methods of BDD. The variable ordering algorithm for two level circuits was based on cover patterns and selected most binate variables first and the one for multi-level circuits is based on depth first traverse of circuits. In the approach used, firstly initial variable ordering was generated and then was optimized. Initial variable ordering was generated by heuristics and the generated ordering was further optimized by exchanging variable orders. In both cases, the acquired variable orderings were optimized by exchanging a variable with its neighbour in the ordering.

N. Ishiura et al [17] proposed a new algorithm called gradual improvement methods for minimizing Binary decision diagrams (BDD's). In this algorithm, optimum order was found

by exchange of variables of BDD's. The use of BDD representation of a given function and intermediate functions makes it possible to introduce pruning into the method, which in turn reduces the computation cost. Only one problem that left was the final solution was dependent on the initial order.

R. Rudell [18] Proposed a modification to an OBDD package whereby the OBDD package itself determine and maintained the order of the variable because there was a drawback of OBDD's that there was a need for application- specific heuristic algorithms to order the variables before processing and also for many applications, heuristic algorithms were insufficient to allow OBDD operations to complete within a limited amount of memory. So, for the solution, at each garbage collection, an algorithm is applied on the OBDD to reorder the variables so as to reduce the number of nodes in the OBDD. Two OBDD minimization algorithms were implemented: the window permutation algorithm and the sifting algorithm. Dynamic variable ordering using sifting algorithm was able to complete several OBDD operations which were not able to complete without dynamic variable ordering and the resulting OBDD tends to be significantly smaller. The only drawback with the dynamic variable ordering was the runtime for OBDD operations increases significantly.

R. Drechsler et al. [19] presented a method to find a good variable ordering for OBDDs using genetic algorithms. In this paper, genetic algorithm (GA) was applied to find a variable ordering that minimizes the size of ordered binary decision diagrams (OBDDs). He compared GA to sifted and (for smaller functions) to optimal minimized OBDDs. It has been shown that the GA performed very well and often produces optimal OBDD sizes. The experimental results have shown that the GA approach is a practical alternative to the exact algorithm for variable ordering. It was also applicable to functions with more than 20 variables due to its small runtimes. The paper showed that GA approach was superior to sifting. If SIFT is applied as long as time is spent for the convergence of the GA, it yields worse results than the GA approach. This demonstrated that operations combining differing elements really provided better quality also in this context. It was also possible to use the GA with smaller populations. Then the algorithm was speeded up, but also the quality of the result is decreased.

H**. Choi et al. [20]** In this paper, size of binary decision diagram (BDD) has been reduced in the combinational circuit power estimation by using the dynamic size limit. So, basically activity of signal has been calculated through the power estimation of combinational circuit.

To reduce the size of BDD, initially a predefined static limit was often used. But static size limit does not support the varying importance of nodes. So, He presented a method called dynamic size limit of BDD to reduce the size of BDD and that also reduced the CPU time and also solves the problems of previous methods that uses the predefined static limit.

C. Meinel [21] this paper described a linear sifting algorithm which extended Rudell's variable reordering algorithm by combining it with linear transformations. He showed that linear sifting minimizes the combined size of the OBDD and the transformation in most cases, with an average improvement of 22% in the number of nodes over a large set of experiments. By using linear sifting it was possible to extract a linear filter and, hence, necessary decomposition was achieved. Using this method, functions were synthesized with standard tools which fail otherwise**.** The experimental results showed that this approach may produce good results in cases where the function cannot be synthesized without decomposition. The approach was easy to implement in all environments which have used dynamic reordering for OBDD's. It also very well supported the concept of symbolic OBDD-representations and may immediately profit from further improvements of the linear sifting algorithm.

P. Lindgren [22] He presented a low power optimization technique for BDD mapped circuits. Dynamic power dissipation characteristics of circuits from a structural mapped BDD have been estimated, by approximating the switching activity of circuit nodes. A technique for minimizing the overall sum of internal switching probabilities for a BDD based on efficient local variable exchange operations has been presented. It has been shown that the dynamic power dissipation can be reduced using this technique, when switching probability is used as an estimate for circuit switching. The paper also shows that the increase in the size of the resulting circuits was relatively small as compared to that obtained through the use of a BDD size reduction technique.

W. N. N. Hung et al. [23] proposed a technique called scatter search for minimizing the Binary decision diagram. Scatter search operated on a set of solutions, the reference set, by combining these solutions to create new ones. The main mechanism for combining solutions was such that a new solution was created from the linear combination of two other solutions. This technique gives a reasonable compromise between quality that is BDD reduction and time. So it does not blow up easily like the exact algorithm and offered a reasonable reduction in BDD size compared to other algorithms. It delivered almost optimal BDD size

with less time than exact algorithm on smaller benchmarks and delivered smaller BDD size than genetic algorithm or simulated annealing at the expense of longer runtime on larger benchmarks. The experiment demonstrated the efficiency of the approach in comparison with simulated annealing, genetic algorithm and various other methods. Experiment also compared the CPU run time (in seconds) and BDD nodes between Scatter Search and the Exact Algorithm. For larger circuits, scatter search did not perform as much reduction as the exact algorithm. That was the tradeoff between quality and speed. So far, the results were in favour of scatter search.

R. Ebendt [24] presented a new exact branch and bound technique for determining an optimal variable order and minimizing the BDD. This technique often avoided repeatedly visiting states. In contrast to all previous approaches that only considered one lower bound, this technique made use of a combination of three bounds and, by this, unnecessary computations were avoided. The lower bounds have been derived by generalization of a lower bound known from very large scale integration design. They allowed one to build the BDD either top down or bottom up. Moreover, a faster method of lower-bound computation as well as an efficient method of partial BDD reconstruction was described, which avoided many time consuming variable shifting. A comparison to the best minimization algorithm known so far showed that runtime could be reduced by up to 49%. The new bottom up approach resulted in a speed up of a factor up to two orders of magnitude compared with the best bottom up approach known so far. Experimental results were given to show the efficiency of the approach.

S. Chaudhury et al. [25] presented a BDD based optimization considering the output phase of the sub functions using Genetic Algorithm for area and power minimization. For satisfying this, he describes that area and power consumption can be determined by the total number of nodes of the BDD and the expected switching activity of the nodes. Also, a proper polarity selection of the sub functions can not only reduce the number of BDD nodes, but also the switching. When all these things were applied to a number of benchmark circuits, then number of nodes in the BDD were reduced and hence the area and power. A trade-off has also been done for combined area and power minimization, considering the node switching as the major candidate for power consumption. It has been found that the proposed method minimizes the area by about 3% and power by about 14%. Also, he concluded that the circuit could be optimized in FPGA based mapping of BDD tree, by optimizing the nodes.

S. Chaudhury, A. Dutta [26] proposed a genetic algorithm based input variable ordering to reduce the area and power. Concept of BDD was used in this paper, as Boolean logics can be graphically manipulated to reduce the number of nodes and hence the area when implemented as binary decision diagrams. Ordering of BDD nodes play a significant role in this context. For input variable ordering of OBDD, Most of the algorithms focus primarily on area minimization. However, suitable input variable ordering helps in minimizing the power consumption also. In this particular paper, a genetic algorithm based technique is implemented, to find an optimal input variable order, while node reordering is taken care by the standard BDD package. Also, a comparison of techniques is made with other standard methods of variable ordering for OBDDs and superior results were found.

A. Kumar et al. [27] proposed the optimization technique for getting optimal ordering which produces sub optimal BDD. Latest approach based on “genetic algorithm” for optimal ordering is presented in this paper. In order to get maximum benefit from BDD method, it was observed that the ordering of basic events should be optimal. So, they successfully implemented a Genetic algorithm method for the selection of best ordering. The main idea in the application of GA in BDD size optimization was to define population size and representation of ordered set of variable as chromosome. However, they have successfully developed the optimization technique for getting optimal ordering which can produce suboptimal BDD. But one thing that they observed was that during simulation of few fault trees, as the order of fault tree increases computation time increases significantly. So if this could have parallelized with a multiprocessor system, a considerable amount of run-time reduction could be achieved. One more area of improvement was the way they input the fault tree in the system. In their work, fault tree input was given as sum-of-product (SOP) in numeral form. So the future scope of this method could be to develop a graphical user interface (GUI) so that it becomes more users friendly.

O. Baradu et al. [28] presented a new double hybridized Genetic Algorithm for optimizing the Variable Ordering in Reduced Ordered Binary Decision Diagrams (ROBDDs). In this paper, the first hybridization adopted embryonic chromosomes as prefixes of variable orders instead of complete variable orders and combines a branch & bound technique with the genetic algorithm, a hybridization that leads a better trade-off between exploration and exploitation as could be seen by the better quality of solutions. The second level of hybridization was done with the existing sifting algorithm, known as one of the most effective heuristic for this problem, which is incorporated as a hypermutation operator. The

effect of hypermutation has been studied and an analysis of appropriate fitness definations has been concluded. The results showed that the use of hybridization improved the performance of resulting algorithm.

S. Chaudhury, A. Dutta [29] presented an algorithmic optimization of Binary decision diagrams and also performance evaluation of multi level circuits in terms of area and power. As discussed in earlier papers, that Binary Decision Diagrams (BDDs) can be graphically manipulated to reduce the number of nodes and hence the area. In this context, ordering of BDDs played a major role. Most of the algorithms for input variable ordering of OBDD focused primarily on area minimization. However, suitable input variable ordering helps in minimizing the power consumption also. So in this particular work, two algorithms were proposed namely, a genetic algorithm based technique and a branch and bound algorithm to find an optimal input variable order. Experimental results show a substantial saving in area and power. The techniques were compared with other optimized techniques such as scatter search technique and dynamic variable ordering and it was found that the proposed two techniques such as genetic algorithm and branch & bound techniques were superior compared to other in fulfilling the objectives.

CHAPTER 3
VARIABLE ORDERING ALGORITHMS

Most of algorithms using BDD have polynomial runtime in the size of the BDD (measured in the number of nodes). Unfortunately, in many applications, very large size BDD can induce a BDD based tool impractical or inefficient. As, the Size of BDDs is very sensitive to the chosen order on the input variables. Any carelessness in choosing orders can easily lead to exponentially large graphs even though the given function could easily have been represented as a very compact BDD under a good variable order [30].

So in this context, a number of variable ordering algorithms exists and can be mainly divided into- static variable ordering, dynamic variable ordering and evolutionary algorithms. Static variable ordering techniques attempt to establish the variable ordering prior to constructing the actual decision diagram, while dynamic variable ordering techniques attempt to adjust the ordering online during the actual construction of the decision diagram. Since static heuristics generate the final variable ordering before construction of the decision diagram even begins, there is no guarantee of good quality solutions from the resulting order. Alternatively, since dynamic variable ordering allows for adjusting the variable order during the actual construction of the decision diagram, they are generally considered more effective in providing efficient orderings; however, they are also typically much more time consuming in practice than the simpler, static heuristics, and thus, are often considered less practical [31]. Basically, Static ordering finds the node order before generating BDDs and mostly they are adopted to get the initial order of the input variables. Dynamic ordering heuristics in BDD package periodically reorder the variables to reduce the number of nodes. In dynamic ordering, variable order is automatically changed by the BDD package, transparent to the user. A brief description of these algorithms is as follows:

3.1 STATIC VARIABLE ORDERING TECHNIQUES

3.1.1 DEPTH FIRST TRAVERSAL ALGORITHM

Traversing a tree involves iterating (looping) over all nodes in some manner. The name given to a particular style of traversal comes from the order in which nodes are visited. Most

simply, does one go down first (depth-first: first child, then grandchild before second child). Depth-first traversal of a tree always starts at the root of the tree, visits a node and then recursively visits the subtrees of that node. Since a graph has no root, when we do a depth-first traversal, we must specify the vertex at which to begin. There are three types of depth-first traversal: pre-order, in-order and post-order. For a binary tree, they are defined as operations recursively at each node, starting with the root node follows:

Pre-order

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

In-order (**symmetric**)

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

Post-order:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root.

This algorithm was developed by Fujita et al. [33] that tried to find a natural static ordering for the variables. Almost all ISCAS'85 Benchmark circuits can be verified by this algorithm in several CPU minutes. It gives a natural ordering using a depth first traversal of a circuit. All inputs with more than 1 fan-out are ordered first. This algorithm performed hundred times better than other known techniques [32] and works well for all but one circuit in the benchmark circuits.

This algorithm is based on the observation that the inputs whose connections are topologically close to each other in the circuit should be near in the variable order. In the case of tree-structured circuits composed of AND, OR, NAND, NOR and NOT gates, one of the best variable orderings is the order found when traversing the circuit from an output to inputs in a depth-first way. However, we cannot obtain good variable orders in some circuits, since most circuits are not tree structured. This algorithm has been well suited for single output circuits. This algorithm has the disadvantage that the variable order generated for the highest priority output node is used for other output nodes, even if it doesn't give the best result for them. Moreover, this algorithm cannot be applied directly to the circuits having many inputs and gates with multiple fan-outs. Hence, this algorithm is not suitable for larger circuits [32].

3.1.2 VARIABLE INTERLEAVING METHOD

This Method was developed by Fujii et al. for multiple output circuits and is based on the observations that the input whose connections are closed together topological in the circuit, and should also be near in variable ordering. While conventional algorithms use variable appending, the new algorithms use variable interleaving and are based on circuit topology. The principle of the algorithm is to merge a variable order for an output into a variable order for outputs with higher priority by maintaining the order for the output as much as possible. In the conventional algorithms, a newly ordered variable is always appended to the end of the ordered variable set, while in this new algorithm, newly ordered variable are inserted into an appropriate position of the ordered variable set. We can call these variable merging methods as variable appending and variable interleaving, respectively. The order of outputs is determined in the depth first traversal way. The conventional algorithm is different from this new algorithm only in the variable merging method. These algorithms have been found to be much more effective than conventional algorithms and can be applied to wider classes of circuits since a good variable ordering can be obtained for the all the outputs of the fan-in gates, a good ordering is also obtained for the output of the gate as well. [34]

3.2 DYNAMIC VARIABLE ORDERING TECHNIQUES

Dynamic Variable Ordering algorithm was proposed by R. Rudell [18]. Techniques that iteratively improve variable orders are frequently employed “dynamically:” if the size of the BDDs grows past a given threshold during the execution of an operation, the operation itself is suspended and reordering is performed, usually by some variant of sifting. After reordering the operation that was interrupted is usually restarted.

The key idea behind this technique is to have the OBDD package which itself determines and maintains the variable order of the OBDD. This variable order is changed automatically by the OBDD package, transparently to the user, as operations are performed. Because the variable order within the OBDD is no longer static, this technique is referred to as dynamic variable ordering. When using dynamic variable ordering, a total order is defined for all variables before and after each package operation; however, the order is periodically adjusted by the OBDD package, as a consequence of an operation, to find a better order. Logically, the variable order changes in between the package operations. Thus, it maintains all advantages provided by the ordered BDD data structure, such as canonicity and efficient recursive algorithms.

This dynamic approach has proved very effective in many applications, because it allows reordering to intervene when it is actually needed. On the other hand, in sequential verification it is not uncommon that most of the time is taken by reordering, and it appears that a certain amount of control on the process by the application is beneficial.

With the help of this algorithm, a new minimizing algorithm called the shifting algorithm was proposed to minimize the graph size. The advantage of using this dynamic ordering algorithm is that many computations for finding a good ordering reach completion using this algorithm when the same computations fail using other algorithms.
Different algorithms implementing this technique are:

3.2.1 WINDOW PERMUTATION TECHNIQUE

Another minimizing algorithm that Rudell used for dynamic variable ordering is the window Permutation algorithm, proposed by Fujita et al. [33]

This technique minimizes the size of an OBDD using adjacent variable exchange. This algorithm proceeds by choosing a level i in the DAG and exhaustively searching all k! Permutations of the k adjacent variables starting at level i. This is done using k! - 1 pair wise exchanges followed by up to k (k - 1)/2 pair wise exchanges to restore the best permutation seen. It is then repeated starting from each level until no improvement in the DAG size is seen. **A** level is marked after the permutation at the level is known optimal. Because the swap of two adjacent variables is efficient, the window permutation algorithm remains practical for values of ***k*** **as** large **as** 4 or **5.** The window permutation algorithm is limited in its ability to find good variable orders. A limitation of the window permutation algorithm appears to be that several moves can be required to move a variable a long distance and these moves can be blocked by an intermediate up-hill move. Hence, it works well for less number of variables only. The complexity increases with the increase in number of inputs. Thus, there is a less benefit of using the window permutation method. So for better results, another minimizing algorithm called sifting algorithm using dynamic variable ordering was introduced, which was able to generate OBDDs for all but two of the benchmark circuits.

SHIFTING ALGORITHM

This algorithm is based on finding the optimum position for a variable, assuming all other variables remain fixed. The variable is exchanged with its successor until it becomes the next to the last variable in the directed acyclic graph (DAG). Then it is moved up to the top of the DAG. The position for the best DAG size is remembered and the variable is moved down to

the optimum position.If there are *n* variables in the DAG (excluding the constant level which is always at the bottom), then there are *n* potential positions for a variable, including its current position. Among these *n* positions, the sub-goal employed by the shifting algorithm is to find the spot which minimizes the size of the DAG. The shift algorithm has the advantage that a variable can move a long distance in the ordering. Note that the DAG-size can increase significantly after the first few variables swaps, and then eventually reduce below the starting point. This allows a type of uphill move to be taken - the acceptance of the entire sequence of pair-wise swaps is based on the best position seen regardless of any increase in the intermediate DAG size.

LINEAR SHIFTING ALGORITHM

It combines the efficiency of shifting and the power of the linear transformations. Linear transformations replace one variable, a_i, with a linear combination of variables. A linear combination is obtained by taking the exclusive-or of the arguments or its complement. Shifting is a local search algorithm that iteratively improves the variable order by a series of swaps of adjacent variables. Each variable is considered in turn and is moved up and down in the order so as to take all positions successively. The variable is then returned to one position where the minimum size of the BDD was recorded. The process then continues with another variable. The effectiveness of shifting stems from its ability to move a variable to any position in the order in a short time. Its time efficiency relies on the ability to quickly swap adjacent variables. It is indeed possible to perform such a swap by accessing only the nodes labelled by the two variables being exchanged. Each variable is considered in turn and as in sifting, it is moved up and down in the order. Let a_i be the chosen variable, and let a_j be the variable immediately following it in the order. One basic step of linear sifting consists of the following three phases:

1. Variables a_i and a_j are swapped; let the size of the BDD after the swap be s1.

2. The linear transformation $a_j \rightarrow a_i = a_{j,}$ is applied; let the resulting size of the BDD be *s2.*

3. If s1 <= *s2* then the linear transformation is undone. This is obtained by simply applying the transformation again, since it is its own inverse.

The net effect of the three-phase procedure is that x_i is moved one position onward in the order, and possibly linearly combined with a_j. Conversely, if a_j is the variable being moved, it is first swapped and then a_i is combined with it. Linear sifting is slower than normal sifting in most cases, because the cost of a linear transformation is comparable to the cost of a

variable swap. If we assume that the graphs manipulated by the two algorithms are approximately of the same size, then linear sifting will be approximately three times slower than standard sifting. [35]

3.3 EVOLUTIONARY ALGORITHMS

3.3.1 SCATTER SEARCH ALGORITHM

Scatter search was introduced by Glover in 1977. Scatter search offers a reasonable compromise between quality (BDD reduction) and time. On smaller benchmarks circuits, it delivers almost optimal BDD size with less time than the exact algorithm. For larger benchmarks, it delivers smaller BDD sizes than genetic algorithm or simulated annealing at the expense of longer runtime. Scatter search is a very aggressive search method that attempts to find high quality solutions fast. The Method 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. In contrast, here in scatter search, it chooses two or more elements of the reference set in a systematic way with the purpose of creating new solutions [23]. The two fundamental features of the scatter- search methodology are:

1. Useful information about the form (or location) of optimal solutions is typically contained in a suitably diverse collection of elite solutions.
2. Constructing combinations that extrapolate beyond the regions spanned by the solutions considered, incorporating both diversity and quality. Taking account of multiple solutions simultaneously, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

The genetic algorithms and simulated annealing also results in fairly small BDD sizes. They are both population-based optimization that bear some similarity with scatter search. In some cases, the genetic algorithm can obtain BDD sizes almost as good as scatter search. In some other cases, simulated annealing can obtain BDD sizes close to that of scatter search [36].

3.3.2 BRANCH AND BOUND ALGORITHM

Branch and Bound (BB) based algorithmic approach is an excellent optimization technique for multi-objective problems and has a finite but usually very large number of feasible solutions. A BB algorithm finds the optimal solutions of various problems, especially in discrete and combinatorial optimization. This is done by an iteration process which has three main components: selection of the solution set for bound calculation, and branching. The sequence of these may vary according to the strategy chosen for selecting the next solution set to process. Here the selection of 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. Branching scheme defines a tree structure whose nodes are obtained from the previous level by a splitting procedure i.e. subdivision of the solution space of the nodes into two or more subspaces to be investigated in a subsequent iteration. The next step of the BB technique is a procedure that computes only the lower bounds of the solution set by calculating the bounds for each of the solutions, within the given solution set. This step is called bounding. The lower bounds are calculated by setting the objective function of the proposed problem based on the fitness [29].

The key idea of the BB algorithm is, if the lower bound for some tree nodes (set of candidates say) 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. This step is called pruning, and is usually implemented by maintaining a global variable m (shared among all nodes of the tree) that records the minimum lower bound seen among all solutions examined so far. Any node whose lower bound is greater than m can be discarded. Otherwise, the bounding function for the subspace is calculated and compared with the current best solution [37].

3.3.3 GENETIC ALGORITHM

A global optimization technique known as genetic algorithm has emerged as a candidate due to its flexibility and efficiency for many optimization applications. It is a stochastic searching algorithm. This technique was developed by John Holland during 1960s. GA is inspired by the evolutionary theory explaining the origin of species. In nature, weak and unfit species within their environment are faced with extinction by natural selection. The strong ones have greater opportunity to pass their genes to future generations via reproduction. In the long run,

species carrying the correct combination in their genes become dominant in their population. Sometimes, during the slow process of evolution, random changes may occur in genes. If these changes provide additional advantages in the challenge for survival, new species evolve from the old ones. Unsuccessful changes are eliminated by natural selection [38, 39]

The genetic algorithm (GA) is a search heuristic and excellent multi-objective optimization technique that is based on the principle of the natural selection and natural genetics. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA) which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

They start with an initial population (solution space) consisting of a set of randomly generated solutions. Based on some reproductive plan especially, the crossover and mutation, they are allowed to evolve over a number of generations. After each generation, the chromosomes are evaluated based on some fitness criteria. Depending upon the selection policy and fitness value, the set of chromosomes for next generation are selected. Finally, the algorithm terminates when there is no improvement in solution over a fixed number of generations. The best solution at that generation is accepted as the solution produced by GA. The formulation of Genetic Algorithm for any problem involves the careful and efficient encoding of the solutions to form chromosomes, crossover and mutation operators and a cost function measuring the fitness of the chromosomes in a population. The flow chart for GA is shown in figure 3.2.

The process of GA follows this pattern:

1. An initial population of a random solution is created.
2. Each member of the population is assigned a fitness value based on its evaluation against the current problem.
3. Solution with highest fitness value is most likely to parent new solutions during reproduction.
4. The new solution set replaces the old, a generation is completed and the process continues at step (2).

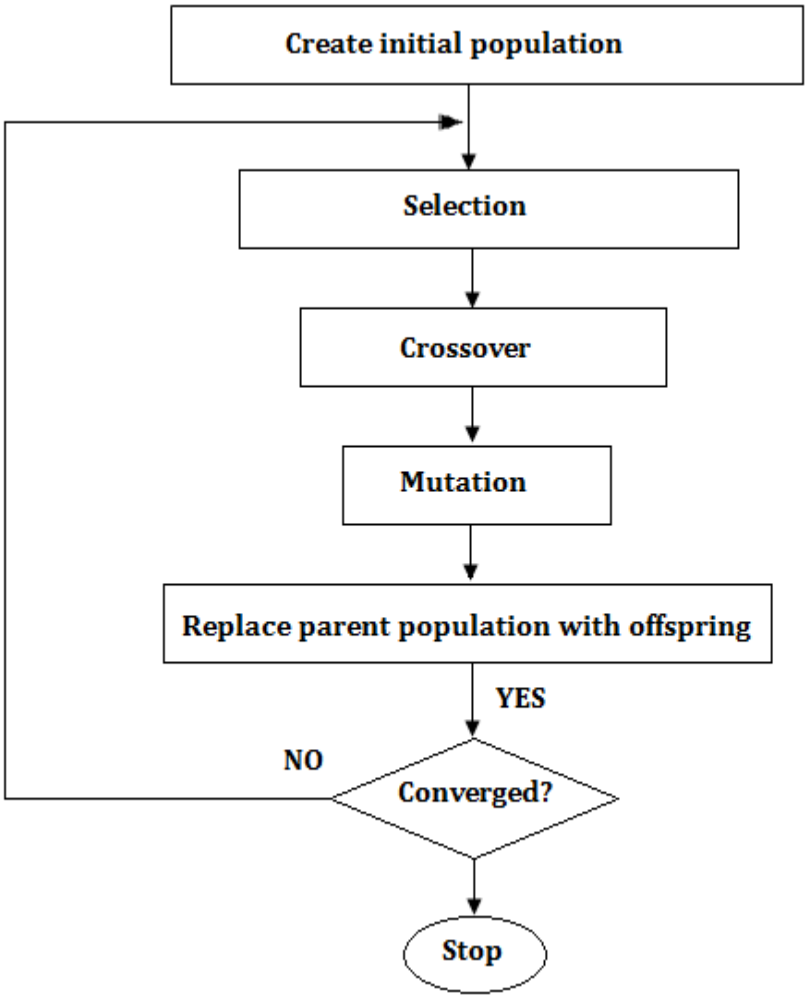


Figure 3.2 Flow chart of Genetic Algorithm

3.3.3.1 COMPONENTS NEEDED TO IMPLEMENT A GENETIC ALGORITHM

The components that are needed to implement a genetic algorithm are:

1.) Representation
2.) Initialization
3.) Evaluation Function
4.) Genetic Operators
5.) Genetic Parameters
6.) Termination

GENETIC OPERATORS- Two types of genetic operators namely the crossover and the mutation, are applied to selected parents for generating offspring. Crossover is performed between two selected individuals, called parents, by exchanging parts of their features (*i.e.* encodings) to form two new individual called offspring. The crossover point is selected

randomly, and the substrings of two parent chromosomes are exchanged to form the offspring. Care must be taken to see that neither a variable is repeated in a chromosome nor a duplicate chromosome is generated as offspring. To generate better offspring, whole population is sorted ac-cording to fitness value and the best-fit chromosomes take part in crossover [38].

The mutation operator brings variety into population by selecting a chromosome randomly from the population and modifies the chromosome at a point depending upon the value of a generated random number. To modify a chromosome, a randomly selected bit is complemented if the chromosome is a binary string. Here, as the chromosome is a string of *n* variables so it can be done by subtracting the randomly selected variable, n_i from *n* and swapping it with the variable n_i. This becomes the new chromosome. Once again the generated chromo-some cannot be a duplicate.

FITNESS MEASURE - The fitness function is an objective or evaluation function used to determine how better a particular solution is. In this problem, initially we take it as a linear combination of area (node count) and switching activity (neglecting leakage) which can be determined by using the following formula.

Fitness Value (F1) = A × Number of nodes + B × Switching Activity

Next, we consider the leakage which is no longer a negligible quantity. In fact, it is a dominant contributor to the total power consumption in today's device scaling scenario. So the modified fitness function (*F*2) becomes,

Fitness (F2) = A × Number of nodes + (1 − A) × (B × Switching Activity + (1 − B) × leakage)

where number of nodes for each BDD representation based on a particular order is given by the standard BDD package (such as, *buddy*-2.4) and switching activity is obtained by traversing through the BDD in a bottom up fashion. Since this fitness value is dominated by area (node count) a modified fitness function is taken where switching activity (for dynamic power) and number of nodes at any generation is divided by the corresponding maximum values of first generation. Lesser the value of fitness function better is the offspring [29].

3.3.3.2 GA APPROACH USED FOR BDD VARIABLE ORDERING

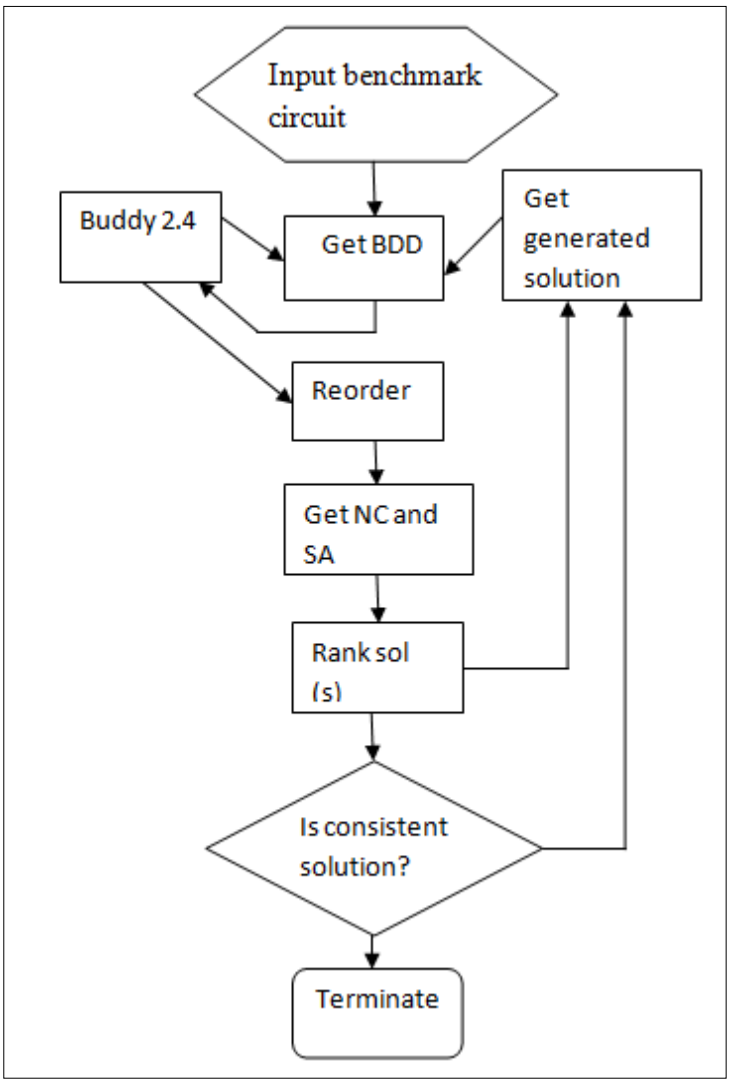


Figure 3.2 GA Approach used for BDD Variable Ordering [29]

3.3.3.3 GENETIC ALGORITHMS VERSUS TRADITIONAL METHODS OF OPTIMIZATION

Genetic algorithms are based on the principles of natural genetics and natural selection. The basic elements of natural genetics: reproduction, crossover, mutation are used in the genetic search procedure. Genetic Algorithms differ from the traditional methods of optimization in the following respect:

1. A population of points (trial design vectors) is used for starting the procedure instead of a single design point. If the number of design variables is n, usually the size of the population is taken as 2n to 4n. Since several points are used as candidate solutions, Genetic Algorithms are less likely to get trapped at a local optimum.

2. Genetic Algorithms use only the values of objective function. The derivatives are not used in search procedures.
3. In GAs the design variables are represented as strings of binary variables that correspond to the chromosomes in natural genetics. Thus the search method is naturally applicable for solving discrete and integer programming problems. For continuous design variables, the string length can be varied to achieve any desired resolution.
4. The objective function value corresponding to design vector plays the role of fitness in natural genetics.
5. In every new generation, a new set of strings is produced by using randomized parents selection and crossover from the old generation (old set of strings). Although randomized, GAs are not simple random search techniques. They efficiently explore the new combination with the available knowledge to find the new generation with better fitness or objective function value.

3.3.3.4 ADVANTAGES OF GENETIC ALGORITHM

The advantages associated with a Genetic Algorithm are**:**

1. Ease of implementation.
2. Differentiability of the objective function is not required.
3. Can handle complex, multi-nodal optimization problems.
4. Power-full search ability to attain the global optimum.
5. Computational simplicity
6. Extremely robust with respect to the complexity of the problem.
7. Diversity of solutions is maintained with mutation.
8. Takes into account the overall effect on the system.
9. Is well suited for parallel computers.
10. Simultaneously searches from a wide sampling of the cost surface.
11. Deals with a large number of variables.
12. Optimizes with continuous or discrete variables.
13. Provides a list of optimum variables, not just a single solution.
14. Can encode the variables so that the optimization is done with the encoded variables.

CHAPTER 4
PROPOSED WORK AND METHODOLOGY USED

In Binary decision diagram, an N-bit adder circuit is always taken for the consideration. In this method, we always go for the number of partitions that are involved in the circuit. A decision diagram generally means a flow which can be used to represent a function or the circuit. Now, basically what happens is if we see a binary decision diagram, it normally represents how complex the circuit is. It is not necessary that if the numbers of nodes are more, then the circuit is good. In simple language, we can say that, the way a circuit which can be represented is provided through a binary decision diagram (BDD).

In our proposed work, one node in BDD can have maximum 2 sub nodes. First of all, number of nodes which are to be considered to provide it to the circuit have been taken. Now depending upon the number of nodes taken, the BDD draws the root nodes. Now, as one root node can have 2 child nodes, hence the total number of child nodes will be:

That is, the number of paths = 2* (total number of root nodes)

once the BDD is created, we need to find the way to optimize it. For this purpose, we need to find the Switching activity (SA) at each node.

A switching activity tells us how the nodes are responding to a particular value. In VLSI, switching activity is the average number of transitions per second at a circuit node. In digital circuits, switching activity is calculated by:

SA= 2* (minterms * maxterms)

So to calculate the switching activity, we should know the number of minterms and the number of maxterms.

For example, we have a series 1011. Minterms are represented by ‘1’ and maxterms are represented by ‘0’. So here, the number of minterms will be 3, and the number of maxterms will be 1. Hence, by using the above formula, SA will be 6.

Moreover, the SA would also show that the node energy of the current format of BDD is acceptable or not. Once, we are done with the SA, we move to the optimization of the current decision diagram. To optimize the current decision diagram means we always need to check which node is not in use for longer time and without which node, we can proceed and it would not affect the output procedure of the entire system.

In this contrast, the basic problem at the optimization is that we cannot simply go for reduction of the circuit nodes. For this purpose, we need to find out some particular terms and those terms are as follows:

1. Mean
2. Variance
3. True positive
4. False positive

Now further on, we need to repair a ROBDD. First of all, in a ROBDD, any node with two identical children is removed and two nodes with isomorphic BDD's are merged. That is the number of nodes in ROBDD should be less than that of Binary decision diagram. To optimize the ROBDD, we find out the node value of each node. Then threshold values for each node will be generated while the processing of nodes. If the current value is less than that of threshold value, then we move to next node. By the end of the completion of the node, we count the total number of nodes which are predicted while the processing. In the same manner, a loop runs for each and every root nodes to the reduction of child node.

GA: GA stands for genetic algorithm. GA provides the optimal way to enhance the values of the provided input. GA used some predefined terms which are popularly known as **Genetic function and Objective function.**

A Genetic function is a function which runs as many times as the node would be there. A Genetic function always generates a value which works as an input for the Objective function.

An Objective function is something which generates high definition rates for the data sets and the circuit it has been provided with. A genetic algorithm always creates a true positive and false positive value which finally decides how much accuracy has been plotted and because it always plots at a very negotiable point value, it is quite difficult to compare the optimization technique used by GA with any other technique.

A gain is the term which provides you the input variations, on the basis of which we can finally draws a table which can be sorted out to find out which nodes we need to reduce and which don't.

CHAPTER 5
RESULTS AND DISCUSSIONS

In this chapter, a ROBDD is implemented on N-bit adder circuit with N(number of bits) using MATLAB.

For this, BDD for the circuit has been created and thus actual number of nodes and actual number of paths has been obtained. The actual number of nodes and the actual number of path variables comes out to be 63 and 126 respectively. After that, ROBDD is applied to the circuit to reduce the number of nodes and the number of paths. The reduced number of nodes and reduced number of paths were 21 and 41 respectively, while creating the new number of nodes and new number of paths. Also, the switching activity of the circuit has been reduced to 16 as shown in figure 1.

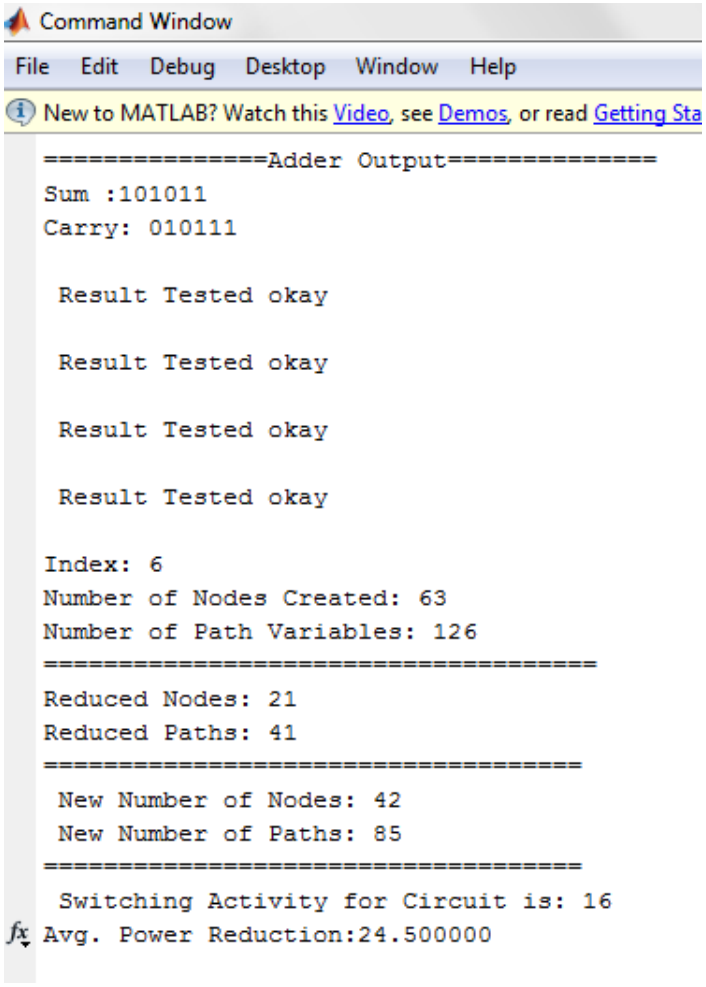


Figure 5.1 New Number of Nodes, New Number of Paths and Switching Activity

5.1 IMPLEMENTATION OF GA

Finally, after reducing the number of nodes in the circuit and calculating switching activity of the circuit, genetic algorithm is implemented on the circuit for optimizating the power and better performance. We have done optimization using matlab with N>500. The average power reduction of 24.50 has been obtained, that is the optimized value and the best mean come out to be 22.8 with area and power reduction as shown in figure 2.

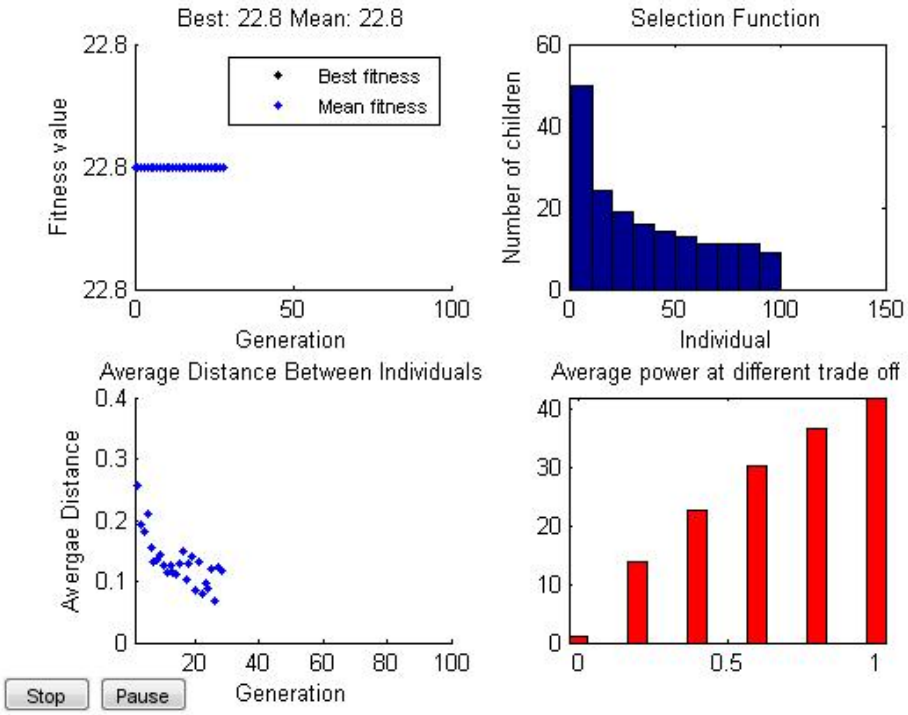
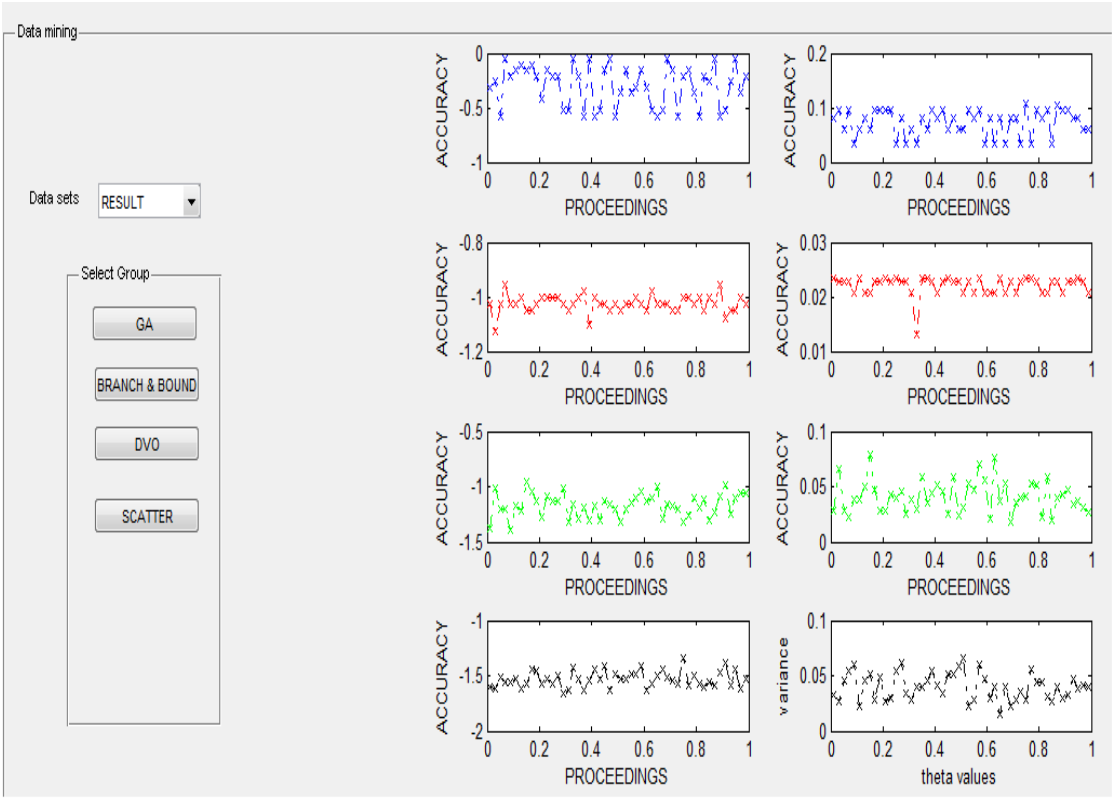


Figure 5.2 Graph of average power at different trade off and fitness value

5.2 COMPARISON OF GA WITH OTHER TECHNIQUES

After this, Genetic algorithm approach has been compared with other algorithmic techniques such as Branch & Bound, Scatter Search and Dynamic Variable Ordering and the results showed that the genetic algorithm based technique has best accuracy in terms of power and area are shown in figure 3. It shows GA attains highest accuracy in terms of power and area as compare to other techniques and scatter search has less accuracy.



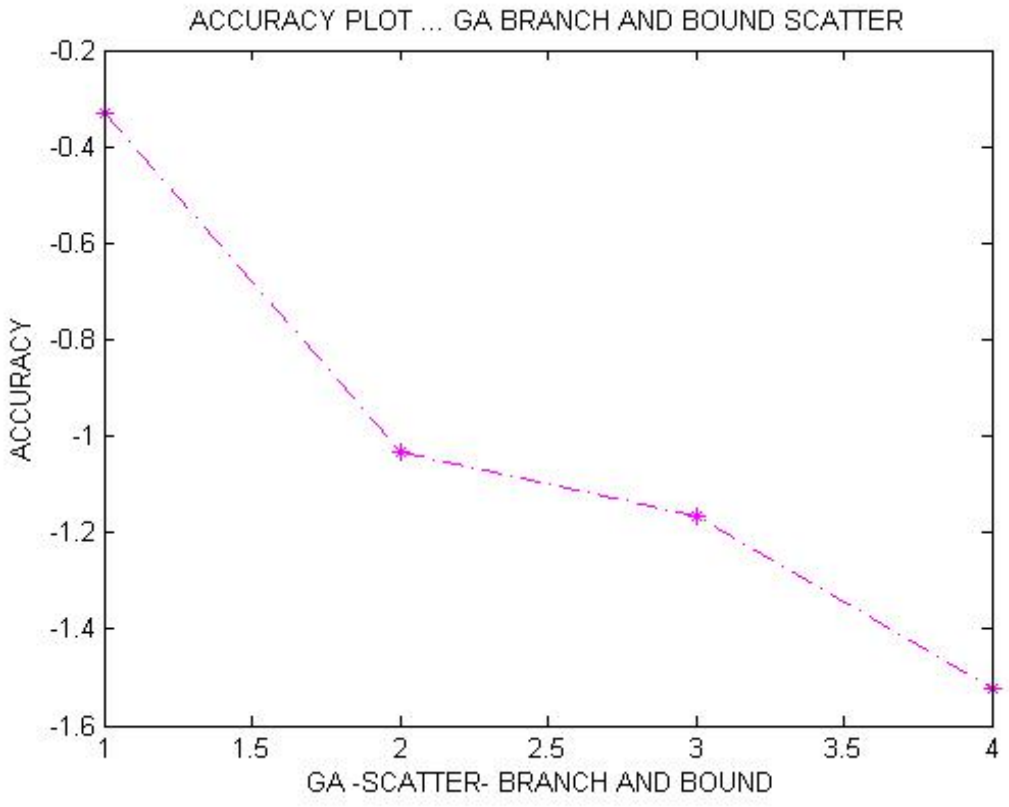


Figure 5.3 Graph showing accuracy of GA, Branch and bound, scatter search

CHAPTER 6
CONCLUSION AND FUTURE SCOPE OF WORK

Boolean functions can be graphically manipulated to reduce the number of nodes, hence the area, when implemented as Binary decision diagrams. In BDD, the area and power consumption is determined by the total number of nodes. As the number of nodes reduces, area reduces and hence it also reduces the power. A proper polarity selection of the sub-functions can not only reduce the number of BDD nodes, but also the switching activity. To optimized BDD, we have calculated the switching activity of the circuit.

Since the time, the decision diagrams have been introduced; there have been quite different methods which have been used for optimization of these decision diagrams. In our work, GA has been implemented for better accuracy and performance. GA has been also compared with different other algorithmic techniques such as branch and bound, scatter search and dynamic variable ordering. Experimental results showed an average power reduction of 24.5 which is the optimized value and GA found to give best results in terms of power and accuracy.

We have tried to optimize the entire thing using GA and the results are quite satisfactory but still there is a chance of some sort of marginal error which can be removed with some other algorithms. The other algorithms could be neural network and fuzzy logic can also be considered in the same contrast. In the same manner, some research can also be done to reduce false positive rate for this procedural behavior of BDD.

REFERENCES

[1] N. Wehn, and M. Munch, “Minimizing power consumption in digital circuits and systems: and overview”, University of Kaiserslautern, Germany, 1999.

[2] P. Singh, C. Gupta, and M. Bansal, “Power optimization in a 4-bit Magnitude comparator circuit using BDD and Pre-Computation strategy”, International Journal of Applied Engineering Research, vol.7, no.11, 2012.

[3] F. J. Hill, and G. R. Peterson, “Introduction to Switching Theory and Logical Design”, Wiley, New York, 1974.

[4] C. Y. Lee, ‘‘Representation of Switching Circuits by Binary-Decision Programs”, Bell System Technical Journal, vol. 38, pp. 985-999, Jul.1959.

[5] 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.

[6] S. Nagayama, A. Mishchenko, T. Sasao, and J. T. Butler, “Minimization of average path length in BDDs by variable reordering”, International Workshop on Logic and Synthesis, 2003.

[7] D. Sensarma, S. Banerjee, K. Basuli, and S. Naskar, “An Optimization Technique using Binary decision Diagram”, International Journal of Computer Science, Engineering and Applications (IJCSEA), vol.2, no.1, Feb. 2012.

[8] R. E. Bryant, “Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams”, ACM Computing Surveys, 1992.

[9] R. E. Bryant, “Binary Decision Diagrams and beyond: Enabling techniques for formal verifications”, International Conference on Computer- Aided Design, pp. 236-243, 1995.

[10] F. Somenzi, “Binary Decision Diagrams”, Department of Electrical and Computer Engineering, University of Colorado at Boulder.

[11] S. C. Chang, D. I. Cheng, and M. M. Sadowska, “Minimizing ROBDD size of incompletely specified multiple output function”, European Design and Test Conference, 1994.

[12] http://en.wikipedia.org/wiki/Binary_decision_diagram.

[13] F. Pfenning, “Lecture notes on Binary Decision Diagrams”, Apr.2011.

[14] S. B. Akers, "Binary Decision Diagram," IEEE Transactions on Computers, vol. c-27, no.6, Jun.1978.

[15] S. Malik, A. R. Wang, and R. K. Brayton, “Logic Verification using Binary decision diagrams in a Logic Synthesis Environment”, IEEE International Conference on Computer-Aided Design, pp.6-9, 1988.

[16] M. Fujita, Y. Matsunaga, and T. Kakuda, “On Variable Ordering of Binary decision diagrams for the Application of Multi-Level Logic Synthesis”, Proceedings of the European Conference on Design automation (EDAC), pp.50-54, 1991.

[17] N. Ishiura, H. Sawada, and S. Yajima, “Minimization of Binary Decision Diagrams based on Exchange of Variables”, IEEE International Conference on Computer- Aided Design (ICCAD), pp.472-475, 1991.

[18] R. Rudell, “Dynamic Variable Ordering for Ordered Binary Decision Diagrams”, International Conference on Computer- Aided Design, pp.42-47, 1993.

[19] R. Drechsler, B. Becker, and N. Gockel, “Genetic Algorithm for Variable Ordering of OBDDs”, IEEE Proceedings of Computer and Design Techniques, vol.143, no.6, pp.364-368, 1996.

[20] H. Choi, and S. H. Hwang, **“**Reducing the size of a BDD in the combinational circuit power estimation by using the dynamic size limit”, IEEE International Symposium on Circuits and Systems, vol.3, pp. 520-523, 1997.

[21] C. Meinel, F. Somenzi, and T. Theobald, “ Linear Sifting of Decision Diagrams and its Application in Synthesis, IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, vol.19, no.5, pp. 521-533, May 2000.

[22] P. Lindgren, M. Kerttu, M. Thornton, and R. Drechsler, “Low power optimization technique for BDD mapped circuits”, Proceedings of the Design automation conference, ASP-DAC, Asia and South Pacific, pp.615-621, 2001.

[23] W. N. N. Hung, X. Song, E. M. Aboulhamid, and M. A. Driscoll, “BDD minimization by Scatter-Search”, IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, vol.21, no.8, pp.974-979, 2002.

[24] R. Ebendt, W. Gunther, and R. Drechsler, “An Improved Branch and Bound Algorithm for Exact BDD Minimization”, IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, vol.22, no.12, pp.1657-1663, Dec 2003.

[25] S. Chaudhury, and S. Chattopadhyay, “Output phase assignment for area and power optimization in multi-level multi-output combinational logic circuits”, Annual IEEE India Conference, pp. 1-4, 2006.

[26] S. Chaudhary, and A. Dutta, “Genetic algorithm based variable ordering of BDD for multilevel logic optimization with area- power tradeoffs”, 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), pp.627-630, 2010.

[27] A. Kumar, A. Kumar, S. Choudhary, and P. V. Varde, “Optimization of Binary Decision Diagram using Genetic Algorithm”, 2nd International Conference on Reliability, Safety and Hazard (ICRESH), pp.168-175, 2010.

[28] O. Baruda, R. Ebendt, and I. Furdu, “Optimizing Variable Ordering of BDDs with Double Hybridized Embryonic Genetic Algorithm”, 12th International conference on

Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 167-173, 2010.

[29] S. Chaudhury, and A. Dutta, “Algorithmic optimization of BDDs and Performance evaluation for multi-level logic circuits with area and power trade-offs”, SciRP Journal, Online Publication,Circuits and Systems, vol.2, pp. 217-224, jul.2011.

[30] W. Mingquan, and Y. Haibin, “BDD minimization based on Genetic Tabu Hybrid Strategy”, 6th International Conference on ASIC, vol. 2, pp. 948-952, 2005.

[31] M. Rice, and S. Kulhari, “A Survey of Static variable Ordering Heuristics for Efficient BDD/MDD construction” University of California, Riverside.

[32] M. Fujita, H. Fujisawa, and Y. Matsunaga, ““Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation”, IEEE Transactions on Computer-Aided Design, vol. 12, no. 1, Jan. 1993.

[33] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluations and Improvements of a Boolean Comparison Method based on Binary Decision Diagrams," IEEE International Conference on Computer-Aided Design, Nov. 1988.

[34] H. Fujii, G. Ootomo, and C. Hori, “Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams”, IEEE/ACM International Conference on Computer-Aided Design, pp.38-41, 1993.

[35] C. Meinel, T. Theobald, and F. Somenzi, “Linear Sifting of Decision Diagrams,” Proceedings of the 34th Design Automation Conference, pp. 202-207, 1997.

[36] W. N. N. Hung, and X. Song, “BDD Variable Ordering by Scatter- Search”, International Conference on Computer Design (ICCD), pp. 368-373, 2001.

[37] M. M Rahman, and M. Chowdhury, “Examining Branch and Bound Strategy on Multiprocessor task Scheduling”, 12th International Conference on Computer and Information Technology (ICCIT), pp. 162-167, 2009.

[38] T. P. Patalia, and G. R. Kulkarni, “Behavioural Analysis of Genetic Algorithm for Function Optimization”, IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp.1-5, 2010.

[39] C. Reves, “Genetic algorithm”, School of Mathematical and Information Sciences, Coventry University.