"IMPLEMENTATION OF MEMETIC ALGORITHM FOR ESTIMATION OF NODE COUNT AND POWER USING VARIOUS CROSSOVER OPERATORS"

Submitted in Partial Fulfilment of the Requirements for the award of the degree of

MASTER OF TECHNOLOGY

In

VLSI Design

Submitted By

Anamika

Roll no. 601261007

Under the guidance of

Mrs Manu Bansal, Assistant Professor, ECED, T.U, Patiala

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR UNIVERSITY, PATIALA

July 2014

CERTIFICATE

I hereby certify that the work which is being presented in this thesis entitled "**Implementation of Memetic Algorithm for estimation of node count and power using various Crossover operators**" by me in partial fulfilment of requirements for the award of degree of Master of Technology in VLSI Design from Thapar University, Patiala, is an authentic record of my study carried under the guidance and supervision of Mrs Manu Bansal (Assistant Professor), ECED.

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

Date: 11th July, 14

ANAMIKA

Roll No. 601261007

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

Date: 11th July, 14

11/7/14

Mrs Manu Bansal

Assistant Professor

ECED, Thapar University

Countersigned by:

(Dr. Sanjay Sharma)

Professor and Head

ECED, Thapar University

Patiala-147004

Dean of Academic Affairs

Thapar University

Patiala-147004

ACKNOWLEDGEMENT

This report is completed with prayers of many and love of my family and friends. However, there are a few people that I would like to specially acknowledge and extend my heartfelt gratitude who have made the completion of this report possible. With the biggest contribution to this report, I would like to thank **Mrs Manu Bansal** had given me her full support in guiding me with stimulating suggestions and encouragement to go ahead in all the time of the report.

I am also thankful to **Dr. Sanjay Sharma**, Professor and Head, Electronics and Communication Engineering Department, for providing us with the adequate infrastructure for carrying out the work.

I am also thankful to **Dr. Kulbir Singh**, P.G Coordinator, Electronics and Communication Engineering department, for the motivation and inspiration that triggered me for the 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.

ANAMIKA

ABSTRACT

Low power consumption has emerged as a key design parameter for digital VLSI systems. Therefore, accurate methods are required to estimate the switching activity at the internal nodes of the logic circuits to determine average power dissipation. Since, manipulation of Boolean function is an important element of many logic synthesis algorithms including logic optimization and logic verification of sequential and combinational circuits, therefore, it is important to have efficient methods to represent and manipulate such functions. A major problem with binary decision diagram (BDD) based manipulation is the need for application-specific heuristic algorithms to order the input variables before processing. Therefore, finding a good variable order for ordered binary decision diagrams (OBDDs) is an essential part of OBDD-based CAD tools.

For the current problem of variable ordering and for determining signal activity, the technique that has been proposed and used in this thesis work is a Modified Memetic Algorithm (MMA) based technique with different crossover operators (namely order crossover, cycle crossover and partially mapped crossover operator) that finds an optimal input variable order with an aim to reduce node count for a multi-input multi-output (MIMO) Boolean function and also aim to reduced power dissipation by determining the signal activity using BDD-based probabilistic technique.

TABLE OF CONTENTS

CHAPTER CONTENTS PAGE NO.

CHAPTER CONTENTS **PAGE NO.**

LIST OF ABBREVIATIONS

ABBREVIATION	MEANING
BB	Branch and Bound
BDD	Binary Decision Diagram
DAG	Directed Acyclic Graph
DBRS	Differential bidirectional random search
DMA	Differential Memetic Algorithm
EA	Evolutionary Algorithm
GA	Genetic Algorithm
HGA	Hybrid Genetic Algorithm
LS	Local Search
MA	Memetic Algorithm
MIMO	Multiple Input Multiple Output
MMA	Modified Memetic Algorithm
MUX	Multiplexer

ABBREVIATION	MEANING
NP	Non-Polynomial in Time
OBDD	Ordered Binary Decision Diagram
PMX	Partially mapped
PSO	Particle Swarm Optimization
PTL	Pass Transistor Logic
RB1eX	Randomized blending crossover
ROBDD	Reduced Ordered Binary Decision Diagram
SDD	Shared Binary Decision Diagram
WIN	Window Permutation Algorithm

LIST OF FIGURES

LIST OF TABLES

1. Introduction

Boolean function manipulation is an important component of many logic synthesis algorithms including logic optimization and logic verification of combinational and sequential circuits. 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 large 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 Diagram have been proposed and successfully used in many industrial applications. A Boolean function that describes a digital circuit can be represented as a Binary Decision Diagram (BDD) which is a directed acyclic graph with respect to an order and satisfying a set of properties. The number of its non- terminal nodes gives their size. For multiplexer based design styles such as Pass Transistor Logic (PTL), a smaller number of nodes directly transfers to a smaller chip area. A BDD can be implemented either in canonical form, reduced order binary decision diagram (ROBDD) or in any other specific order (OBDD). If all identical nodes are shared and all redundant nodes are eliminated, the OBDD is said to be reduced (an ROBDD).

The size of a BDD is strongly dependent on the order of input variables. It is not unusual for the size of an OBDD to increase exponentially with the circuit size using one variable order, but linearly using another variable order. Since the memory requirement and processing time increase as the OBDD size increases, it is important to keep the size of the OBDD as small as possible. Hence a number of heuristics and algorithms have been developed to determine the optimal ordering for input variables.

1.1 Introduction to Binary Decision Diagram (BDD)

BDD is a multi-rooted (shared) directed acyclic graph (DAG) which is used to represent a set of Boolean functions [1]-[2]. Each node in the DAG represents a Boolean function F and has an associated variable x_i and pointers to two other nodes (functions) in the DAG. The node F is written as the tuple (x_i, G, H) where x_i is called the top variable of the function F, G is the positive cofactor of F with respect to x_i i.e. G = F_{xi}, and H is the negative cofactor of F with respect to x_i i.e. H = $F_{xi'}$. Shannon decomposition is carried

out in each node and node F is thus represented as $F = x_iG + x_i'H$. The constant functions 0 and 1 are represented by terminal nodes. BDDs are used for hardware verification. BDDs are also used by techniques for logic synthesis. These applications benefit from optimization of BDDs with respect to different criteria, i.e. objective functions. In this, BDD optimizations happen at a deep and abstract logic level. In a design flow, this is an early stage before the step of technology mapping. After BDD optimization, it is often possible to directly transfer the achieved optimizations to the targeted digital circuits. Given below are few examples of BDD optimization and their applications:

i.) In BDD-based functional simulation, a simulator evaluates a given function by direct use of the representing BDD. A crucial point here is the time needed to evaluate a function. Hence, BDD optimizations are desired to minimize evaluation time [3]-[4].

ii.) One way to synthesize a circuit for a given function is to directly map the representing BDD into a multiplexor circuit. It is known that optimizations of the BDD (e.g. BDD size, expected or average path length in BDDs) directly transfer to the derived circuit (e.g. area and delay minimization). With that, the targeted applications are in the field of logic synthesis.

1.2 Ordered Binary Decision Diagram (OBDD)

Ordered binary decision diagram (OBDD) was proposed by Bryant [2]. An Ordered Binary Decision Diagram (OBDD) is a pair (π, G) where π denotes the variable ordering of the OBDD and G is a finite DAG i.e. G = (V, E) where, V denotes the set of vertices and E denotes the set of edges of the DAG, with exactly one root node and the following properties:

i.) A node in V is either a non-terminal node or one of the two terminal nodes in {1, 0}.

ii.) Each non-terminal node v is labeled with a variable in X_n, denoted var(v), and has exactly two child nodes in V which are denoted then(v) and else(v).

iii.) On each path from the root node to a terminal node the variables are encountered at most once and in the same order.

Thus, OBDD [2] is a restricted decision graph in which the ordering of variables is consistent on all paths of the graph. The order selected for decision variables can significantly affect the number of nodes required in a BDD. The BDD in Fig. 1.1 (a) uses variable order a < b < c < d, while the BDD in Fig. 1.1 (b) represents the same function, with variable order a < c < d < b. Both figures have same function but different number of nodes. 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 and for a given variable order.

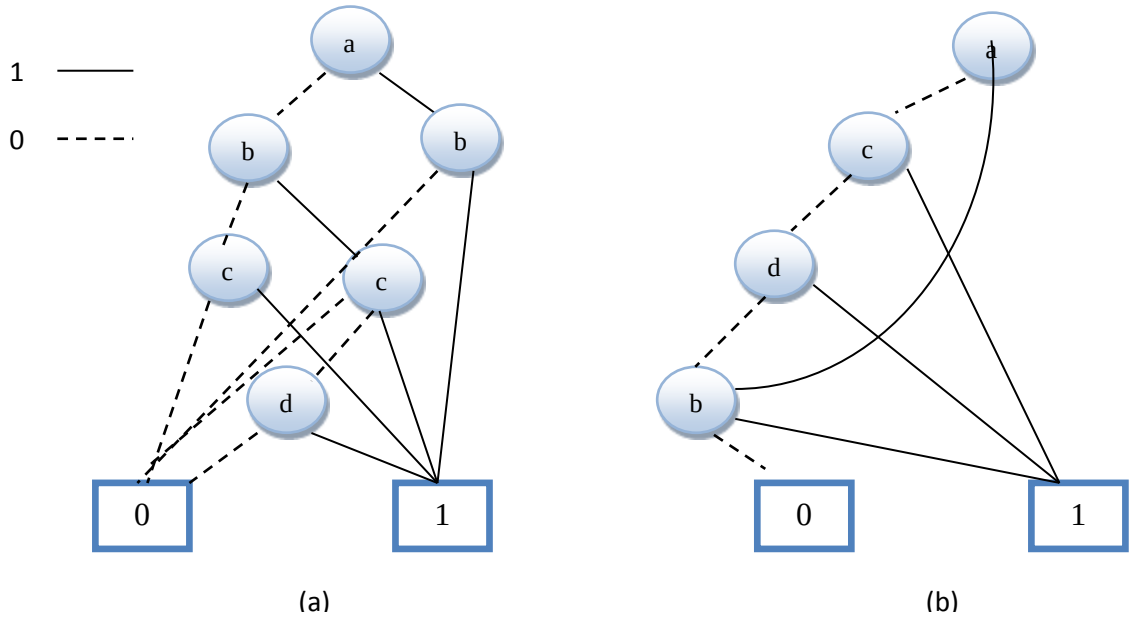

Fig. 1.1 Different OBDDs for same Boolean function F with different variable order

F=ab + a'c + bc'd

1.3 Reduced Ordered Binary Decision Diagram (ROBDD)

The ROBDD representation is defined by imposing restrictions on the BDD specification of Akers [1] such that the resulting form is canonical. A BDD is reduced if it contains no isomorphic sub graphs, and every vertex has distinct children. Reduced, ordered BDDs [2] are canonical representations of Boolean functions: for a fixed variable order, two functions are equivalent if and only if their BDDs are identical. An OBDD is converted to an ROBDD using the following reduction rules:

a. Merging equivalent nodes
b. Merging isomorphic nodes

c. Eliminating redundant tests

Redundant nodes in the graph, i.e. nodes not needed to represent a Boolean function, can be eliminated. ROBDDs allow for a good trade-off between efficiency of manipulation and compactness. Compared to other techniques to represent Boolean functions, e.g. truth tables or Karnaugh maps, ROBDDs often require much less memory and faster algorithms for their manipulation do exist. Fig. 1.2 shows the truth table for a Boolean function and the corresponding OBDD and ROBDD.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

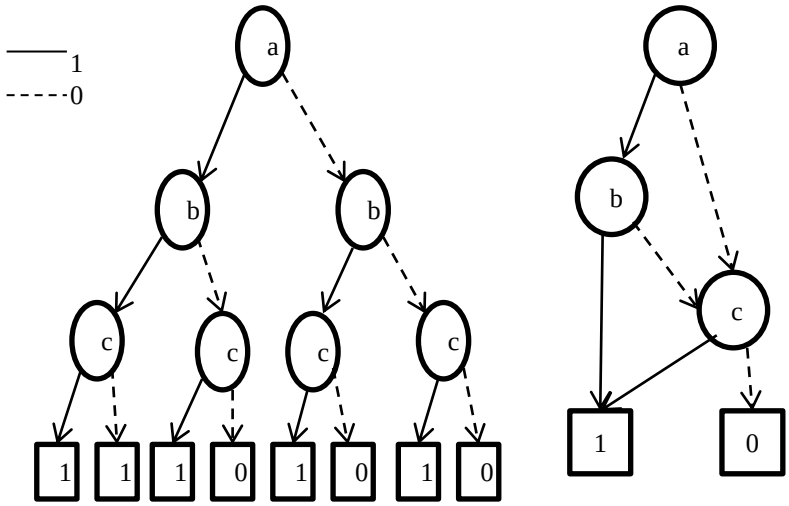

Fig. 1.2 Truth Table, Ordered BDD and corresponding ROBDD for the Boolean function, f = ab+c

1.4 Applications of BDDs

Binary Decision Diagrams (BDDs) have achieved great popularity as data structures for representing Boolean functions in solving most of the combinational problems which arise in synthesis and verification of digital systems [1]-[3]. The two important properties of BDDs which form the basis for applications of BDDs in different areas are:

i.) BDDs are canonical. This property is useful when verifying the equivalence between two circuits [3]. Accordingly, two circuits are equivalent if and only if their BDDs are identical for a specific variable ordering.

ii.) BDDs are effective structures for the representation of large combinatorial sets.

1.4.1 Functional Synthesis: BDDs as MUX Circuits

Boolean functions, represented by BDDs can be realized using multiplexer based design styles such as Pass Transistor Logic (PTL). Pass Transistor Logic uses only two transistors to realize a multiplexor a wired OR of two MOS transistors. The advantages of PTL circuits as an alternative to static CMOS design are higher energy efficiency (low power), less circuit area and higher circuit speed. Each node in the corresponding BDD for a Boolean function represents a 2x1 MUX. In 2x1 multiplexer when select line '*s'* is set to zero then output *y* is equal to *a* and when *s* is set to one then output is *b* which is shown in Fig 1.3 and is expressed logically as $y = s'a + sb$. The MUX representation is obtained by back propagation from leaf (terminal) nodes to the root nodes of the BDD, as shown in Fig 1.4. Hence, lesser node count is desirable in order to reduce the net area for realization and fabrication of a digital circuit.

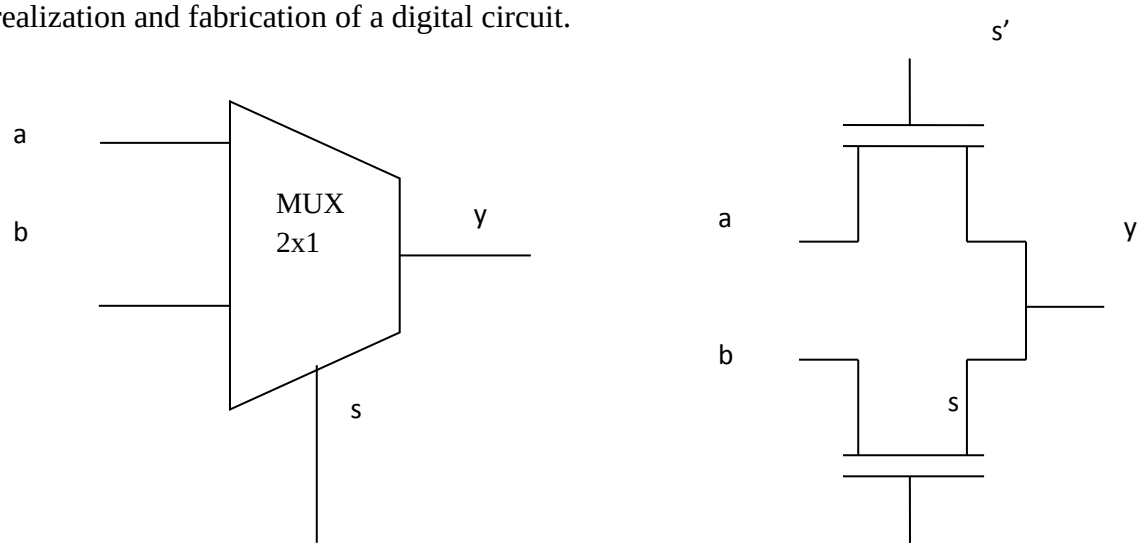

Fig 1.3 A 2x1 Multiplexer and corresponding transistor realization

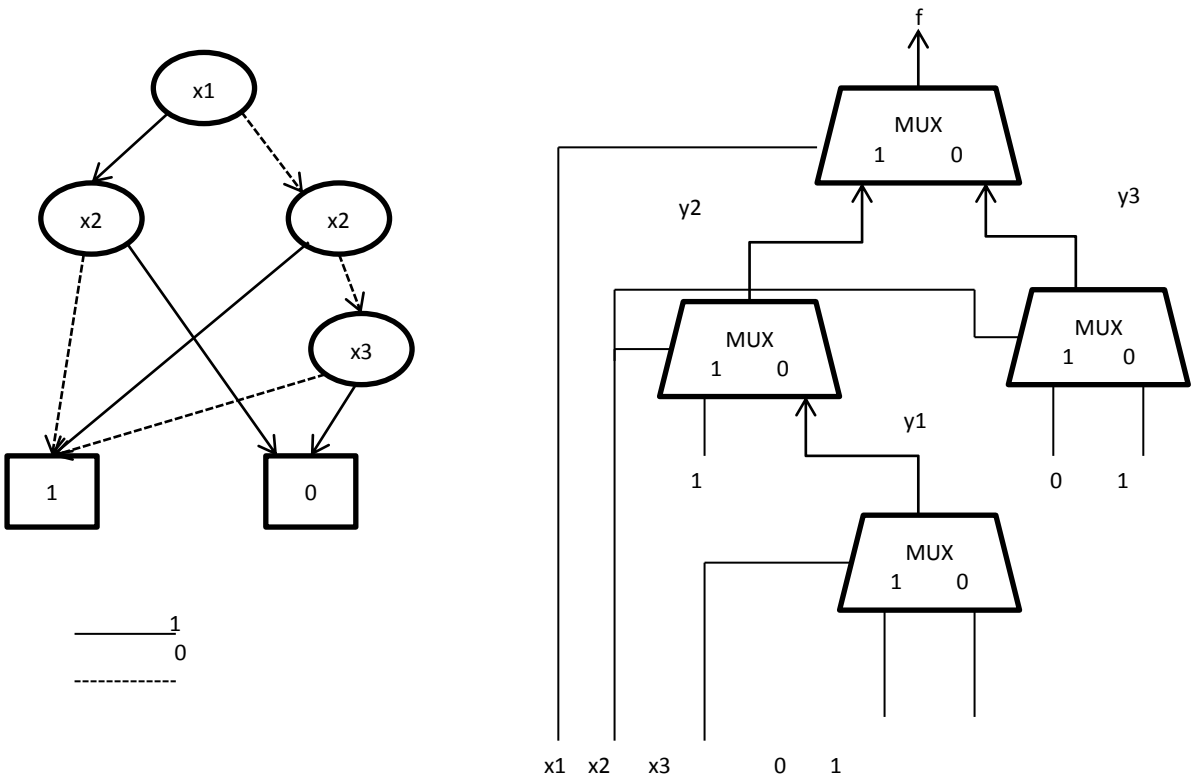

Fig 1.4 ROBDD and corresponding MUX representation for Boolean function, f = x1x2'+ x1' (x2'x3'+ x2)

1.4.2 Functional Verification of Logic Circuits

Binary decision diagrams (BDDs) were originally invented for hardware verification to efficiently store a large number of states that share many commonalities [2]. The canonical property of BDDs makes them an efficient alternate approach for logic circuit comparison. The basic aim for hardware verification is to compare a new design to a *known good* design [5]. Two logic circuits are equivalent if and only if their compact representations in the form of BDDs are identical provided the same variable ordering is used in both representations as illustrated in Fig. 1.5. Fig. 1.5 (a) shows the BDD for Boolean function f1 = ab+a'c+bc. On applying the reduction rules, i.e. eliminating redundant test, ROBDD for function f1 is obtained with variable order a < b < c as shown in Fig.1.5 (b). ROBDD for function f2 = ab+a'c with order a < b < c is shown in Fig. 1.5 (c). The two figures being identical show that the two Boolean functions f1 and f2 are identical.

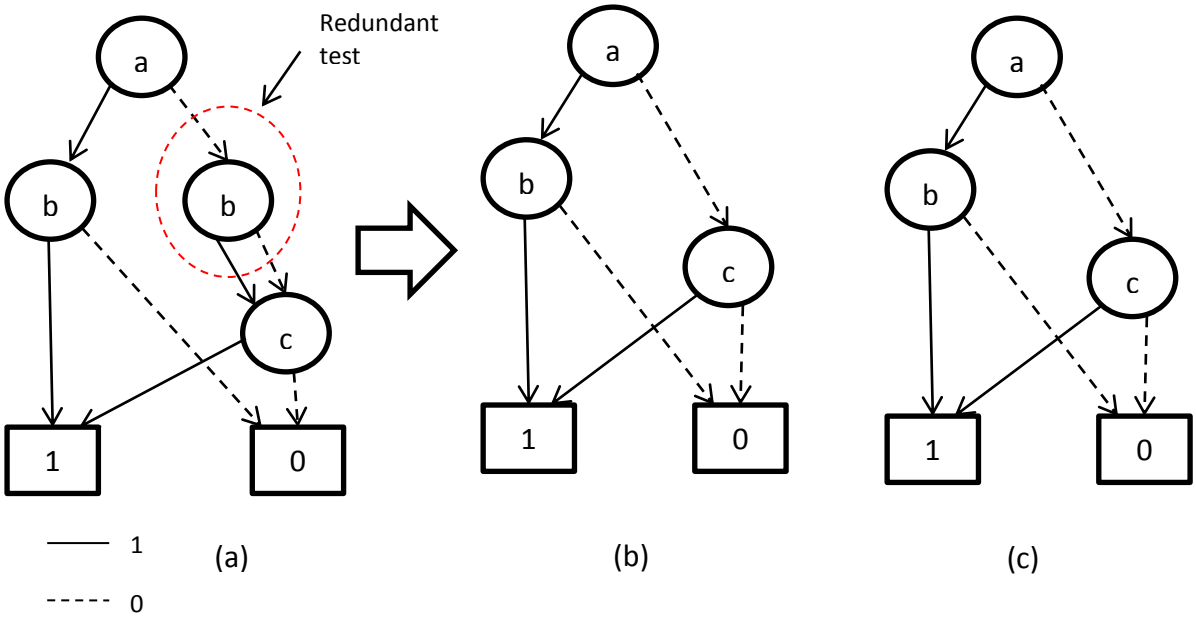

Fig. 1.5 Equivalence Verification of Boolean functions f1 = ab+a'c+bc and f2 = ab+a'c

1.5 Objective of Thesis

The objectives of this thesis work are:

- To study the effect of variable ordering on BDD size corresponding to a given Boolean function.
- Formulation of an improved Memetic Algorithm combining features of different algorithms with an aim to provide lesser BDD size in terms of reduced node count, reduced execution time and reduced power dissipation.
- Implementation of the modified Memetic Algorithm using C++ codes on a Linux platform. C++ Language allows incorporating the features of standard BDD package such as *Buddy-2.4*.
- Experimentation of the implemented algorithm with a number of Benchmark circuits taken from the LGSynth93 benchmark circuit suites.
- Analysis of the results obtained using proposed Memetic Algorithm.

1.6 Organization of Thesis

The thesis report is organized as follows:

- The thesis report begins with a brief introduction to Binary Decision Diagrams (BDDs) and its different types i.e. Ordered Binary Decision Diagram (OBDD) and Reduced Ordered Binary Decision Diagram (ROBDD).
- Chapter 2 provides Literature Review for Binary Decision Diagram and techniques using crossover operator.
- Chapter 3 provides an overview of Memetic algorithm and also a detailed description of the Modified Memetic Algorithm (MMA) that has been proposed in this thesis work, its important features, advantages flow chart of the proposed algorithm and its implementation for BDD optimization using standard BDD package. It also estimates power using probabilistic technique.
- In Chapter 4 experimental results for different Benchmark circuits are given that demonstrates the efficiency of the proposed method by comparing it with other existing methods.
- Chapter 5 gives the conclusion and future scope of the work proposed in this thesis.

2. Literature review

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

S. B. Akers [1] suggested a method for defining, analyzing, testing and implementing large digital function such as carry look-ahead adder by means of a binary decision diagram (BDD). BDD provides a complete and concise description of the digital function involved. Different methods have been suggested to derive the diagrams and examples for various combinational and sequential circuits. Methods used for deriving BDD are modified and extended in order to achieve minimum number of nodes.

S. Malik *et al.* [6] proposed two variable ordering techniques which were based on network topology. As the size of BDD's are sensitive to variable ordering, binary decision diagrams with automatic variable ordering applicable for formal verification of a very large class of circuits is presented. The proposed method proved significantly faster on benchmark circuits.

M. Fujita *et al.* [7] suggested 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 multi-level circuits are based on depth first traverse of circuits. In the approach used, initial variable ordering was generated by heuristics and was optimized by exchanging variable orders. In both cases, the acquired variable orderings were optimized by exchanging a variable with its neighbor in the ordering.

N. Ishiura *et al.* [8] proposed a new improvement method for minimizing binary decision diagrams (BDD's). In the algorithm, optimum order was found by exchanging 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. The disadvantage of the method was that the final solution was dependent on the initial order.

R. Rudell [9] proposed a modification to an ordered binary decision diagram (OBDD) package whereby the OBDD package itself determine and maintain the order of the variable because there were few drawbacks of OBDD's like that they require 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 at each garbage collection, an algorithm is applied on the OBDD to reorder the variables 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 completed 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.* [10] suggested a method to find a good variable ordering for OBDD's using genetic algorithms. Genetic algorithm (GA) was applied to find a variable ordering that minimizes the size of ordered binary decision diagrams (OBDD's). He compared GA to sifted and (for smaller functions) to optimal minimized OBDD's. 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 suggested method 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. The results 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 so the algorithm was speeded up, but the quality of the result is decreased.

H. Choi *et al.* [11] proposed a method called dynamic size limit of BDD to reduce the size of BDD and also to reduce the CPU time. Activity of signal has been calculated through the power estimation of combinational circuit. Initially a predefined static limit was often used to reduce the size of BDD but static size limit does not support the varying importance of nodes, thus proposed method called dynamic size limit was used to solve the problems of previous methods that uses the predefined static limit.

C. Meinel [12] suggested 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 improvement in the number of nodes over a large set of

experiments. By using linear sifting algorithm it was possible to extract a linear filter and hence, necessary decomposition was achieved. Using the suggested 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 suggested approach is 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 [14] 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 BDD based on efficient local variable exchange operations has been presented. It is presented that the dynamic power dissipation can be reduced using this technique, when switching probability is used as an estimate for circuit switching. The presented technique 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.* [13] 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 favor of scatter search.

M.T.V. Baghmisheh *et al.* [15] proposed a differential memetic algorithm (DMA) using randomized blending crossover (RB1eX) with the differential bidirectional random search (DBRS) local search algorithm which aims to scatter the new born offspring more diversely in the whole search space. A comparison of the performance of the DMA and seven other evolutionary algorithms reported demonstrates better performance of proposed DMA algorithm in most of the cases.

M. Kaya [16] proposed two new crossover operators namely sequential and random mixed crossover in genetic algorithm. The fitness values for existing and developed crossover techniques were compared for two stages using two different benchmark problems. For the first stage, both the techniques were applied to reinforced concrete beam problem and the space truss problem. For the second stage, both the techniques were applied to deep beam problem and concrete mix design problem. The results show that higher fitness values are achieved using random mixed crossover technique when compared to existing crossover techniques.

A. Banerjee [17] proposed a probabilistically guided context sensitive crossover operator for evolutionary clustering applications. The cluster-wise comparison is performed in two partitions represented by two parents by using the restricted growth function as the representation for the genotype. Clusters are compared using a statistical basis for spatial randomness on the assumption that natural groupings in data are compact and isolated. The proposed crossover operator has good exploitation properties and is heavily biased against an exploratory genetic search because it identifies and necessarily passes good schemas to the offspring. The proposed and existing crossover methods are also compared for statistical and runtime which are presented on synthetic and real data sets. The results show that the proposed crossover combines the diversity preserving characteristics of an unguided context-sensitive operator with the fast convergence properties of a greedy context-insensitive operator.

M. Maruniak *et al.* [18] proposed a binary decision diagram optimization method combining residual variable with basic BDD reduction method. Since, multiplexer uses logical structure similar to binary decision diagram, therefore, BDD optimization methods are be used to optimize multiplexer tree in combination with multiplexer optimization

methods. The results show reduced total amount of multiplexers in optimized multiplexer tree in comparison to non-optimized multiplexer tree.

O. Hasancebi *et al.* [19] proposed two crossover techniques to obtain better efficiency of Genetic algorithm (GA). These techniques were tested on two truss size optimization problems, and were evaluated with respect to exploration and exploitation aspects of the search process. With the comparative study carried out between the common crossover techniques and the proposed crossover techniques, the paper shows that the direct design variable exchange produces best solution.

3. Modified Memetic Algorithm

3.1 Introduction to Memetic Algorithm

Memetic algorithms (MA) represent one of the recent growing areas of research in evolutionary computation. The term MA is now widely used as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. Quite often, MA is also referred to in the literature as Baldwinian evolutionary algorithms (EA), Lamarckian EAs, cultural algorithms, or genetic local search [20].

Not quite surprisingly in a rapidly expanding field as this is, one can also find the term MA used in the context of particular algorithmic subclasses, arguably different from those grasped in the initial definition of MAs. We can say that a MA is a search strategy in which a population of optimizing agents synergistically cooperate and compete [21].

A particular feature of MAs is greatly responsible for this unlike traditional Evolutionary Computation (EC) methods, MAs are intrinsically concerned with exploiting all available knowledge about the problem under study; this is something that was neglected in EAs for a long time. The exploitation of problem-knowledge can be accomplished in MAs by incorporating heuristics, approximation algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc.

MAs are like EAs population-based metaheuristics. This means that the algorithm maintain a population of solutions for the problem at hand, i.e., a pool comprising several solutions simultaneously. Each of these solutions is termed individual in the EA jargon, following the nature-inspired metaphor upon which these techniques are based. Each individual or agent represents a tentative solution for the problem under consideration. These solutions are subject to processes of competition and mutual cooperation in a way that resembles the behavioral patterns of living beings from a same species. To make clearer this point, it is firstly necessary to consider the high-level template of the basic population event: a generation.
This is shown below in Fig. 3.1.

```
Process Do-Generation (↑↓ pop: individual [ ])
Variables
Breeders; newpop: Individual [ ];
begin
breeders ←Select-From-Population (pop);
newpop ←Generate-New-Population (breeders);
pop← Update-Population (pop, newpop)
end
```

Figure 3.1: The basic generational step

As it can be seen, each generation consists of the updating of a population of individuals, hopefully leading to better and better solutions for the problem being tackled. There are three main components in this generational step: selection, reproduction, and replacement. The first component (selection) is responsible (jointly with the replacement stage) for the competition aspects of individuals in the population. Using the information provided by an *ad hoc* guiding function (*fitness* function in the EA terminology), the goodness of individuals in *pop* is evaluated; subsequently, a sample of individuals is selected for reproduction according to this goodness measure. This selection can be done in a variety of ways. The most popular techniques are fitness-proportionate methods (the probability of selecting an individual for breeding is proportional to its fitness1), rank-based methods (the probability of selecting an individual depends on its position after ranking the whole population), and tournament-based methods (individuals are selected on the basis of a direct competition within small sub-groups of individuals).

Replacement is much related to this competition aspect, as mentioned above. This component takes care of maintaining the population at a constant size. To do so, individuals in the older population are substituted by the newly-created ones (obtained from the reproduction stage) using some specific criterion. Typically, this can be done by taking the best (according to the guiding function) individuals both from *pop* and *newpop* (the so-called “plus" replacement strategy), or by simply taking the best individuals from *newpop* and inserting them in *pop* substituting the worst ones (the “comma" strategy). In the former case, if |pop| = |newpop| then the replacement is termed generational; if |newpop| is small (say |newpop| = 1), then we have a steady-state replacement. May be the

most interesting aspect in this generation process is the intermediate phase of reproduction. At this stage, we have to create new individuals (or agents) by using the existing ones. This is done by utilizing a number of reproductive operators. Many different such operators can be used in a MA, as illustrated in the general pseudo code [25] shown in Fig. 3.2.

Nevertheless, the most typical situation involves utilizing just two operators: recombination and mutation.

Algorithm. Memetic Algorithm.
Compute an initial population of *N* random individuals
Apply local search on these individuals
while the number of iterations is lower than *N*1 and an improvement has occurred since less than *N*2 iterations
do for *i* = 1 to *k*
do Choose 2 individuals randomly
Construct 2 children with crossover
Apply local search on both children
Add children to the population
end
for Keep the *N* best individuals in the population
Apply mutation with a probability μ to every individual of the population.
end while

Figure 3.2: Pseudo code for Memetic Algorithm

3.2 Modified Memetic algorithm

In this chapter, Modified Memetic Algorithm based approach that has been used to identify a good variable ordering in a shared BDD (SDD) is presented. This chapter explains the functioning of basic mechanisms that are used to implement this algorithm. The algorithm incorporates global search exploration feature of GA and local search exploitation of MA with various hybrid operations in a way that provides momentum to the search operation and at the same time prevents the solution to land in local optimum situation.

3.2.1 Solution or Meme Representation

The operation of MMA begins with a population (P) of randomly generated chromosomes representing potential solutions to the variable ordering problem in BDDs. A chromosome, representing a Boolean function of *n* variables, itself is a set of *n* memes (*i*1, *i*2, *i*3, *i*4, *i*5 … *i*n), each represented by an integer number between 1 to *n* without repetition. For example, let us consider a combinational logic circuit consisting of 7 variables (inputs), then according to above, the chromosome can be represented by any of these forms:

3	4	6	1	7	5	2

011	100	110	001	111	101	010

Fig 3.3Solution or Meme Encoding in Chromosome Form

This chromosome identifies an ordering of the input variables to be used in constructing the shared BDD for the multi-output function. The input variables are ordered according to the numbers appearing in the chromosome and the corresponding SDD is generated.

3.2.2 Fitness Function Calculation

Fitness function is calculated to determine the quality of each solution. Fitness of each individual is evaluated in terms of the number of nodes in the corresponding SDD that represents the fitness function for the problem, as determined by the standard BDD package *Buddy-2.4.*

$$Fitness\ Function, F\ =\ Number\ of\ nodes\ in\ SDD$$

3.2.3 Local Search Operation for Meme Improvement

All solutions are sorted in ascending order and the best and the worst *S* solutions among them undergo the local-search (LS) mechanism [22]. The LS mechanism used for optimizing the best *S* solutions performs pair-wise swap of memes in a chromosome, as depicted in Fig. 3.4 , until either a better local optimum result is obtained or for maximum

$$M = n\,X\,(n\text{-}1)\,/\,2$$

where, n is the number of input variables.

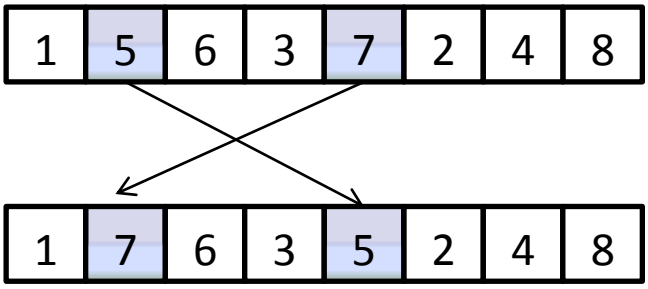

Fig. 3.4 Local Search LS mechanism for improving the performance of best*s* solutions

The LS mechanism used for optimizing the worst *S* solutions is different. These solutions are made to mimic the global best solution. Any worst solution *w* is obtained by randomly selecting a subsequence from the best solution to replace the corresponding position in worst w^{th} solution, while keeping the other positions unchanged. However, if the constraint violation occurs, then the memes are randomly relocated to form a new feasible solution as illustrated in Fig. 3.5. The new solution is retained if the fitness improves, else it is reversed.

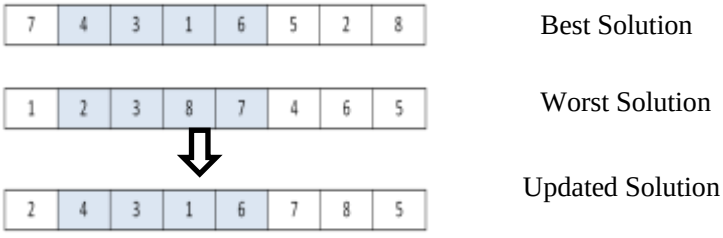

Fig. 3.5 Local Search LS mechanism for improving the performance of worst *s* solutions

3.3 Operators Used

The new local optimum solutions obtained after LS operation are again sorted and allowed to evolve by crossover and mutation operations. This process continues until an optimum variable order is obtained. The GA technique incorporated in this algorithm is hybrid GA as proposed in [23] in which important parameters such as population size, number of generations, crossover rate, mutation rate are not fixed, rather, these are made dynamic and change depending upon the improvement found in every generation.

3.3.1 Mutation Operator

This operation is used to introduce variety in solution. In other words, mutation is defined as a process of randomly disturbing genetic information. This is done to prevent all solutions in a population to fall into a local optimum of solved problem. Mutation operation depends upon the type of encoding scheme used to represent the chromosomes. For example, for binary encoding a few randomly chosen bits are switched from 1 to 0 or from 0 to 1. Since integers are used to represent memes in this algorithm, the mutation operation incorporated is similar in a way that memes are complimented and replaced with their corresponding integer compliments as shown in Fig. 3.6. Also other memes are modified so as not to violate constraints.

$$\text{Compliment of integer } n \;=\; \text{ELEMENTS} - n + 1$$

where, ELEMENTS represent the total number of input variables.

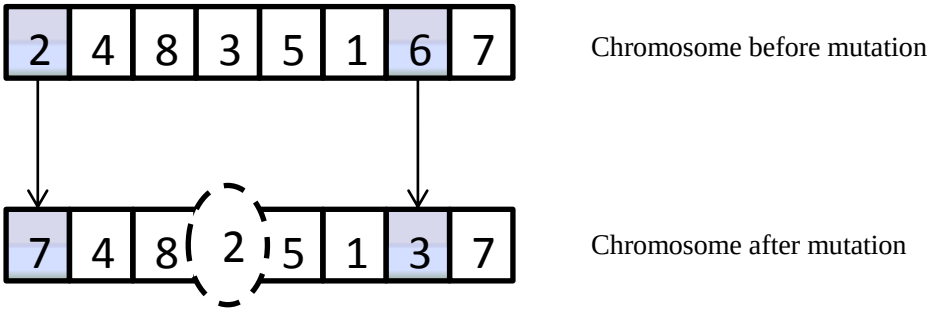

Fig. 3.6 Mutation Operation incorporated in MMA

3.3.2 Cross-over Operator

A modified alternating crossover mechanism is adopted to generate next generation as shown in Fig. 3.7.

Fig. 3.7 Modified Alternating Crossover Mechanism

This crossover mechanism enhances the performance of the algorithm as compared to conventional crossover schemes. A new solution is generated from two randomly selected parents in such a way that memes are copied alternately from these parents. This technique is a hybrid version of the alternating crossover mechanism as presented in [24], the difference being that memes are copied from opposite ends of both the parents without allowing the constraints violation.

3.4 Theoretical Review of Cross-over Operators

Two main strategies that a solid crossover technique should use to locate the optimum are exploration and exploitation. Exploration ability of a crossover technique encourages the search for a rapid and thorough discovery of the design space. A robust exploration prevents the search from locating a local peak. However, the mere use of it becomes inefficient in making use of the already obtained points to reach better points. On the other hand, a solid exploitation employs the previously found points to reach the optimum. However, in this case a slow convergence is observed accompanied by an increasing risk of locating a local peak. In fact, these two strategies are contradictory, and a balanced use of them is vital for achieving an efficient search through crossover. [20]

3.5 Types of Cross-over Operators

Crossover along with selection-reproduction and mutation is one of the main operators of any MA. It provides an exchange of design characteristics between paired individuals (designs). Several crossover operators have been devised to accomplish this task; amongst these are order cross-over, cycle cross-over and partially mapped cross-over operators.

MA can rapidly identify discrete zones within a large search space area to concentrate the search for an optimum solution. This technique changes mutually defined parts of two members selected and obtains different members that give new points in the search space [19]. The crossover operators used in the current study are summarized below.

The crossover point is randomly selected (represented by dashed line) between 1 and L−1 where L is the length of the chromosome. Two new members were obtained by relocating parts, after this the cut-off point is matched in two members.

3.5.1 Order Crossover Operator

This operator conveys a sub-placement from the first parent without any changes and then to resolve conflicts, compresses the second parent by eliminating the cells conveyed by the first parent and then shifting the rest of the cells to the left without changing their order. It then copies this compressed second part into the remaining part of the offspring array. Firstly the crossover point is randomly selected and then the offspring is obtained by relocating parts of the parents as shown in the Fig 3.8.

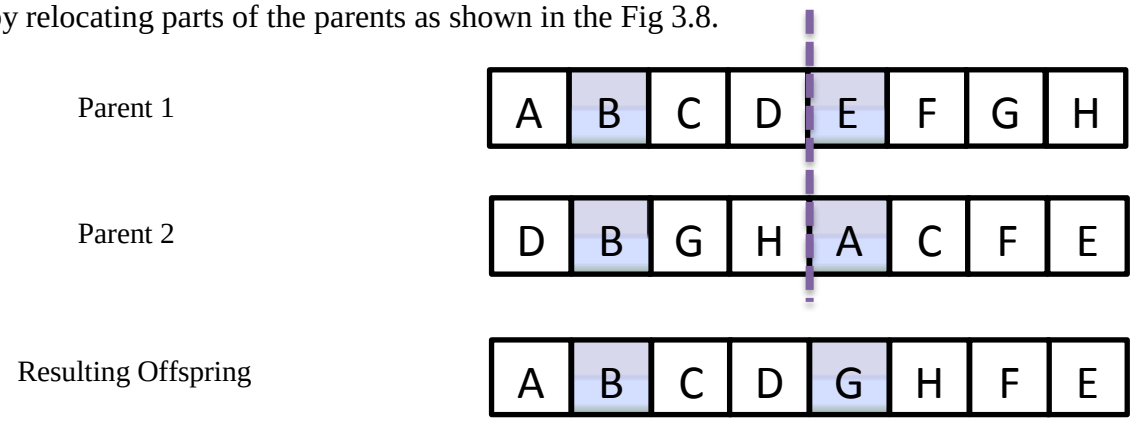

Fig. 3.8Order Crossover Mechanism

3.5.2 Cycle Crossover Operator

In the offspring generated by the cycle crossover, every cell is in the same location as in one parent or the other. The basic idea behind the cycle crossover is to avoid the cell conflicts by finding non-overlapping sets of cells to pass from two parents. The offspring is generated by relocating the elements from parent1 in the offspring taking care of the fact that the element present at the same position in parent2 will be prioritize to get relocate in the offspring. As shown in the Fig 3.9, 'A' gets relocated in the offspring at the first position then instead of relocating 'B', 'D' is prioritized and is relocated at the fourth position in the offspring and similarly other elements are relocated in a cycle.

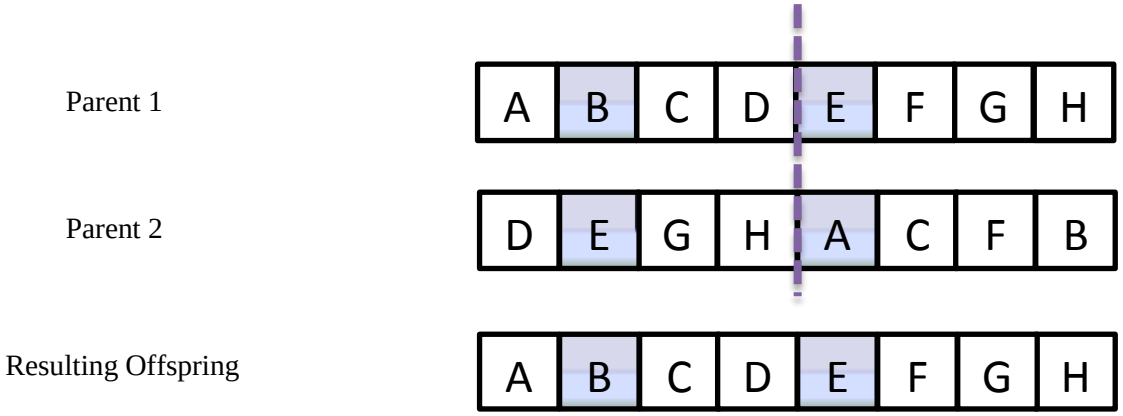

Fig. 3.9 Cycle Crossover Mechanism

3.5.3 Partially Mapped Crossover Operator

In Partially mapped cross-over mechanism (also known as PMX crossover mechanism), with the randomly selected crossover point, the offspring is obtained by partially mapping the elements of the parents as shown in the Fig 3.10. It proceeds with position wise exchange. A cell in the segment of the first parent and a cell in the same location of the second parent define which cells in the first parent have to be exchanged to generate the offspring.

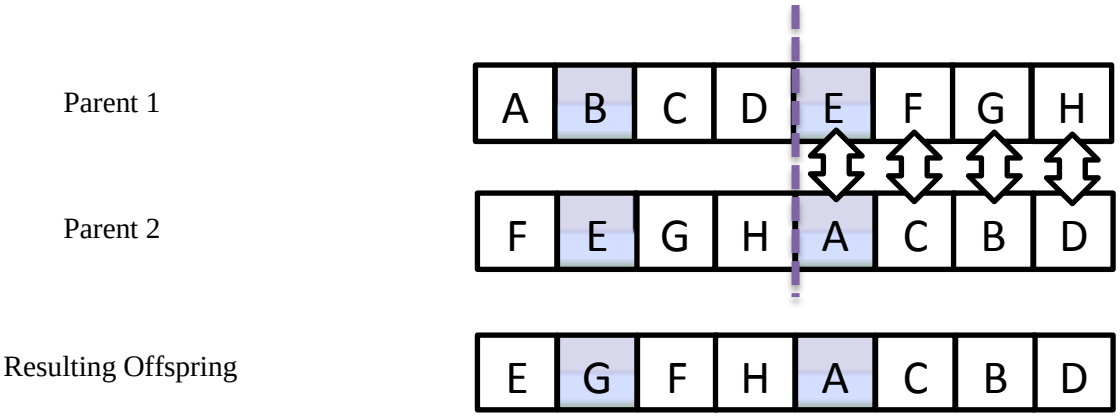

Fig. 3.10PMX Crossover Mechanism

3.6 Parameter Settings and Flow Chart

This algorithm takes a population size of 20 chromosomes per generation with 80% crossover rate and 4% mutation rate. The generation size begins with 20 generations and constants δ_1, δ_2, γ_1, and γ_2 as given in [23] have been fixed at 10%. The constant μ for change in mutation rate is set as 0.05. The constants MAX_POP and MAX_GEN are kept to be 50 and the constant MIN_POP is fixed at 5. For Simple Genetic Algorithm SGA, population size and number of iterations is kept fixed at 50, other parameters being same.

A complete flowchart of the proposed algorithm is presented below in Fig. 3.11.

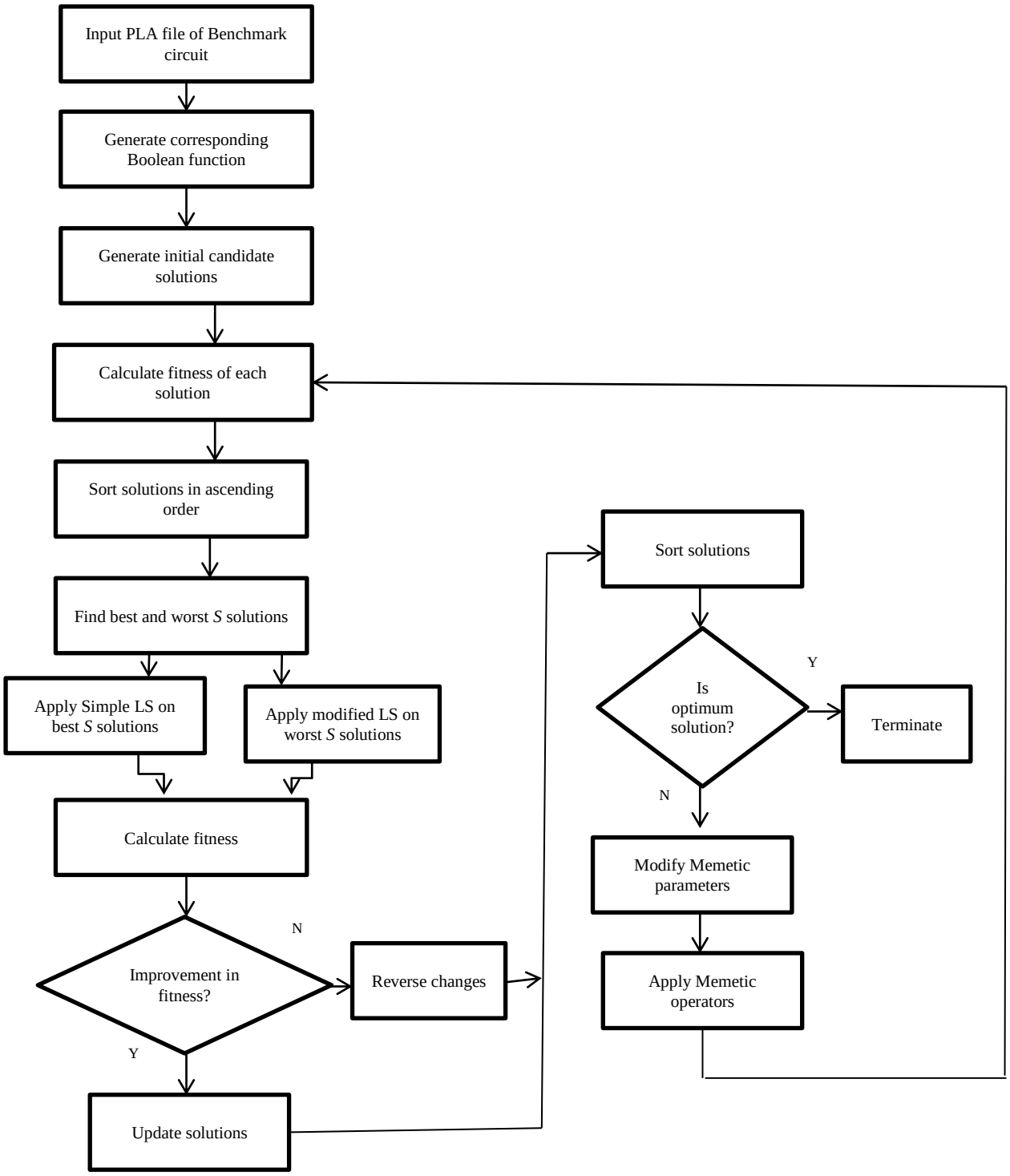

Fig. 3.11 Flow Chart for proposed Modified Memetic Algorithm

3.7 Overview of BDD Package

A BDD package has three main components: the BDD algorithm component, the dynamic variable ordering component and the garbage collection component. The common features of these components are described as under:

3.7.1 BDD Algorithm

This component computes the resultant BDDs for different Boolean functions. The implementation of these algorithms is typically based on depth-first traversal. The unique tables are hash tables with the hash collisions resolved by chaining. A separate unique table is associated with each variable to facilitate the dynamic-variable-reordering process. The computed cache is a hash-based direct mapped (1-way associative) cache. BDD nodes support complement edges where for each edge, an extra bit is used to indicate whether or not the target function should be complemented. The advantage of this encoding is that a function and its complement can be represented by the same BDD and use this extra bit in the reference edge to interpret the BDD either in the positive or the negated form. Implementation-wise, this extra bit is typically encoded in the least significant bit of the address pointer (the reference edge) to avoid incurring extra memory cost. This encoding exploits the property that address pointers in modern machines are always at least 4-byte aligned, which means the least significant bit is always 0. Thus it can be used to encode the complement information.

3.7.2 Dynamic Variable Ordering

Since the size of a BDD graph is sensitive to the order selected on input variables, dynamic variable reordering is an essential part of all modern BDD packages. This component aims to dynamically establish a good variable order as the computation progresses. Typically, when a variable reordering algorithm is invoked, all top-level operations that are currently being processed are aborted. When the variable reordering algorithm terminates, these aborted operations are restarted from the beginning. The dynamic variable reordering algorithms are generally based on the *sifting* algorithm; i.e., the variable orders are changed by exchanging nodes in one level with nodes in the adjacent level. This process is illustrated in Fig. 3.12. This figure shows the sifting process for exchanging the orders of the variable *a* and the variable *b*. Each node is

tagged with a number so that these can be referred easily. Let *f* be the function representing a BDD in terms of its cofactors.

$$f = v'.f|v<-0 + v.f|v<-1$$

where, *v* is the variable in b's root node and the 0-cofactor *f |v<-0* is recursively defined by the reachable sub graph of b's 0-branch child. Similarly, the 1-cofactor *f |v<-1* is recursively defined by the reachable sub graph of b's 1-branch child. Using this definition, the BDD in Fig. 3.12(a) can be represented as

$$a'.(b'.f0 + b.f1) + a.(b'.g0 + b.g1)$$

By rearranging this formula, we get

$$b'.(a'.f0 + a.g0) + b.(a'.f1 + a.g1)$$

New BDD after the performing sifting operation is shown in Fig. 3.12(b). Implementation-wise, node 4 and node 5 are created to represent the new children of node 1. Node 1 is updated to reference these new children and is relabeled with variable b. As for node 2 and node 3, they remain unchanged and if there are no references to them, they will be garbage collected later. Note that because node 1 might be referenced by others, it is important that node 1 is reused with its reachable graph representing the same function. Without this reuse, a new node will have to be created in place of node 1 and all the references to node 1 will be needed to be relocated and updated in order to reference this new node, which is very inefficient. The node-reuse technique allows the sifting algorithm to be a local operation involving only the node and its children.

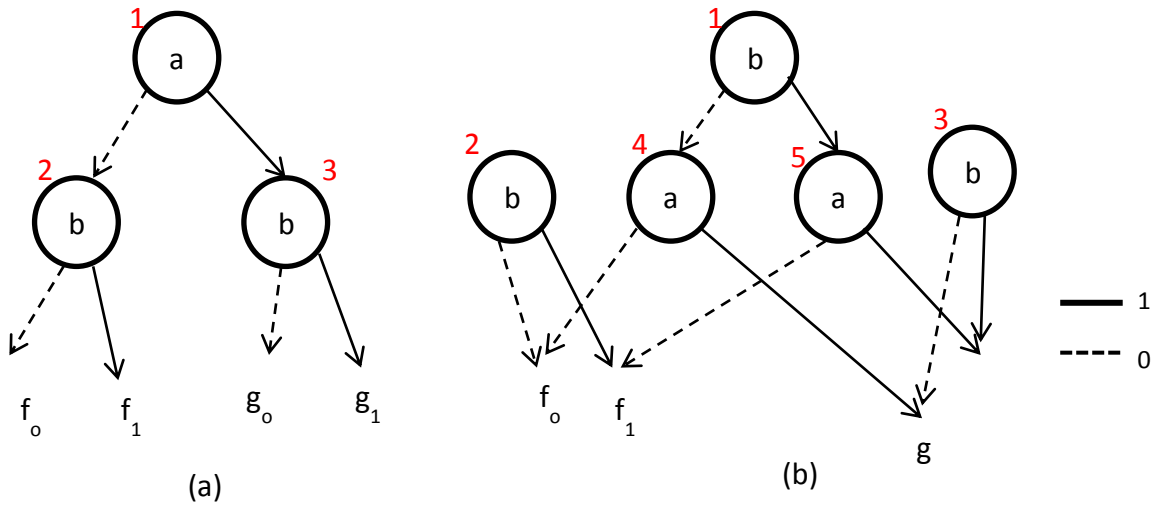

Fig 3.12 Shifting process for Dynamic Variable Reordering

3.7.3 Garbage Collection

BDD computations are inherently memory intensive because after all, it is all about traversing and constructing graphs. Furthermore, in verification, many intermediate BDD results are created to arrive at a simple final answer—true or false. Thus, it is important to have a good garbage collector to automatically remove BDD nodes that are no longer useful. A BDD node is referred to as *reachable* if it is in some BDD that external user has a reference to. As external users free references to BDDs, some BDD nodes may no longer be reachable (*deaths*). These nodes are referred to as *unreachable* BDD nodes. Hence all the *unreachable* nodes are garbage collected by the package leading to the fast manipulation and construction of BDDs by reducing the memory requirement.

3.8 Estimation of Power using Probabilistic techniques

When designing VLSI circuits, the designers accurately estimate the silicon area and the expected performance before the circuit goes into fabrication. However, with the requirements of low power, the designers are faced with the added burden of qualifying their designs with respect to power dissipation. Hence, work has started in earnest to accurately estimate power dissipation at different levels. 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.

3.8.1 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 this portable equipment are very high. Following are different levels of abstraction:

- **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.
- **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.
- **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.

- **Gate-level**- A circuit is described in terms of a net list of gates or a set of Boolean equations reflecting its functionality.
- **Transistor level**- A circuit is described in the terms of its transistor network structure.
- **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.

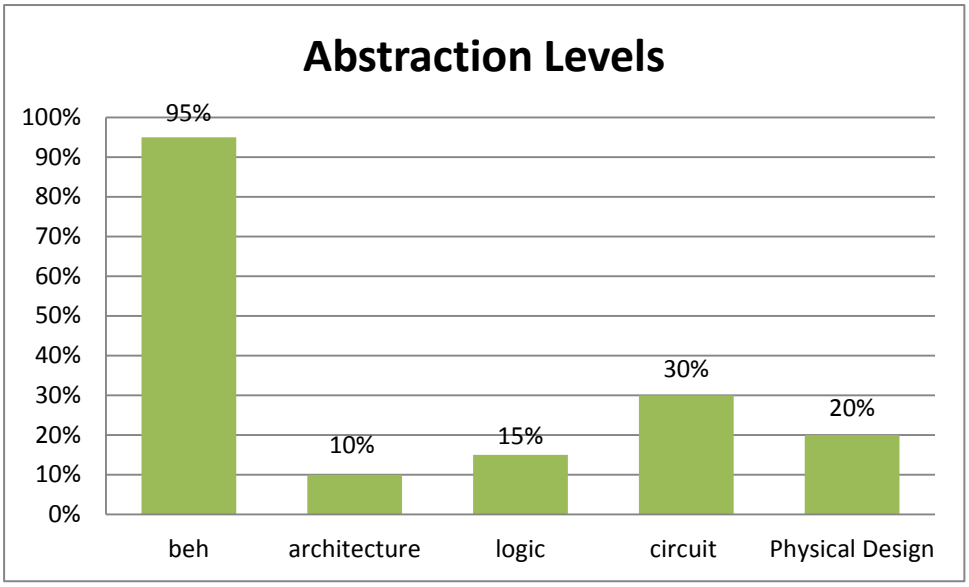

Figure3.13 Bar graph showing different abstraction levels

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].

3.8.2 Contribution Of Several Power Dissipations

It has been noted that the power dissipation in CMOS circuits is mainly due to dynamic (switching and short circuit) current and the sub threshold leakage current. With the scaling down of device sizes and transistor threshold, the sub-threshold current will increase and can become a sizable component of total power dissipation. However, in the current-day technology, the dominant component power is still the switching component (about 80% of the total power dissipation) as shown in the figure. Therefore, we will devote a considerable effort in estimation of switching power in VLSI circuits.

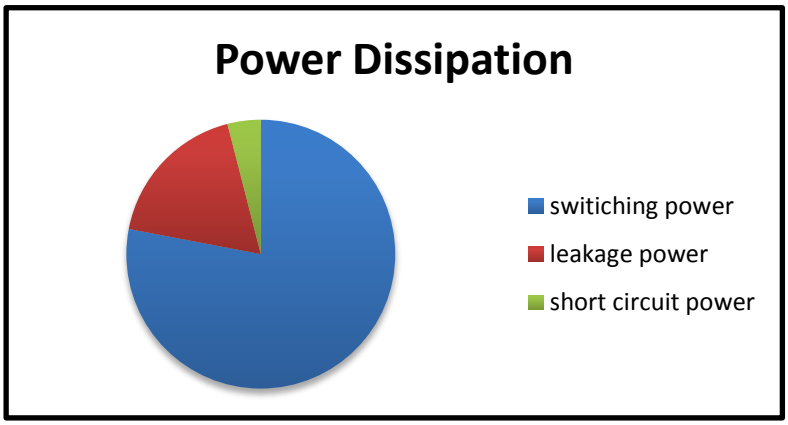

Figure 3.14 Pi chart showing several power dissipations

3.8.3 WHY ESTIMATE POWER DISSIPATION?

Average power estimation involves accurately estimating the average switched capacitances at the internal nodes of the circuits. Accurate estimation of switched capacitance at the architecture or behavioral level is quite difficult, while they can be more accurately estimated at the transistor circuit level. An accurate estimate of power dissipation during various phases of VLSI design can verify the power dissipation requirements and thereby avoid complicated and expensive design changes that might be required due to power constraint violations. Power estimates can also be used during the circuit, logic and high-level synthesis to accurately trade off power versus other design parameters such as area, performance and noise margin.

3.8.4 PROBABILISTIC TECHNIQUE FOR SIGNAL ACTIVITY ESTIMATION

The majority of power dissipation in VLSI circuits is due to signal transitions at circuit nodes. Therefore, accurate methods are required to estimate the switching activity at the internal nodes of the logic circuits to determine average power dissipation. The problem of determining when and how often the transitions occur at a node in the digital circuit is difficult because they depend on the applied input vectors and the sequence in which they are applied. Hence, the easiest solution is to use circuit simulation. Based on the given sets of inputs, the power supply current can be monitored to determine the average power dissipation. The major problem with this approach is that the circuit simulation is too slow for large circuits. Besides being slow, the technique is strongly input pattern dependent. Hence, pattern-independent probabilistic techniques are required to quickly estimate the average number of transitions per circuit node.

3.8.5 SIGNAL PROBABILITY USING BDD

Signal probability of any arbitrary Boolean expression can be easily calculated using BDDs. It is assumed that the signals are spatially uncorrelated and that the signals can be modeled as strict-sense stationary (SSS) and mean-ergodic with zero delay; that is, all the switching is carried out simultaneously, and signal probability values and switching activity do not vary over time.[14]

P(f) denotes the probability that f = 1(the output probability of f) and a(f) denotes the activity for f (the probability that f will change in value from one cycle to the next).

To avoid the high computational complexity of an exact method, it is assumed that there is no spatial correlation between the Shannon cofactors of the function of interest. The

approximation technique provides the exact result for the case where co-factors are spatially uncorrelated. In the case where cofactors are positively correlated an overestimate is obtained, since the switching of a top variable is less prone to cause a true switching of the node's output. The opposite holds for negatively correlated factors. This observation allows for the application of Theorem 3.1 from [7]. The formula in (1) can then be derived using the multiplexer-based circuit model:

$$a(f) = \left(\frac{P(f_0) + P(f_1) - 2\,P(f_0)\,P(f_1)}{1-(a(f_0)+a(f1)-a(f_0)a(f_1))} - \frac{\frac{1}{2}(a(f_0)+a(f_1)-a(f_0)a(f_1))}{1-(a(f_0)+a(f_1)-a(f_0)a(f_1))}\right) * a(v)(1-a(f_0))(1-a(f_1))$$

$$+ \left(\frac{1-P(v)-\frac{a(v)}{2}}{1-a(v)}\right) * a(f_0)(1-a(v))(1-a(f_1))$$

$$+ \left(\frac{P(v)-\frac{a(v)}{2}}{1-a(v)}\right) * a(f_1)(1-a(v))(1-a(f_0))$$

$$+ \frac{1}{2}a(f_0)a(v)(1-a(f_1)) + \frac{1}{2}a(f_1)a(v)(1-a(f_0))$$

$$+ a(f_0)a(f_1)(1-a(v)) + \frac{1}{2}a(f_0)a(f_1)\,a(v)$$

……(1)

In (1), v is the input variable, f_0 is the low cofactor and f_1 is the high cofactor. This formula is used recursively in a bottom-up approach to calculate the activity for each node in BDD.

The switching activity estimation method described is analyzed further to show various properties and demonstrate how it can be applied to low-power synthesis for BDD mapped circuits. The total power dissipation of the mapped circuit is computed as

$$PD_{tot} = \sum_{\forall_n} a(n) * driver\big(fanout(n)\big) + leakage(n)$$

……(2)

In (2), the power dissipation due to leakage is ignored as its contribution to total power dissipation is very less; therefore, *leakage (n)* is ignored.

4. Results and Design Methodology

In this chapter, experimentation results by applying the proposed MMA with a number of Benchmark circuits taken from the LGSynth91 benchmark circuit suites have been presented. The MMA based technique, as defined above, is implemented with C++ codes and experimented by running on Ubuntu Tool. The result of experimentation for different Benchmark circuits has been shown in Table 4.1, Table 4.2 and Table 4.3.

4.1 Comparison with Dynamic Algorithms

Table 4.1 and Table 4.2 performs comparison with the reordering heuristics that have been incorporated in the BDD package *Buddy-2.4*, such as Window Permutation WIN2, WIN2ite, WIN3 and Sifting algorithm[7], [8]. The results indicate that in 75.55% cases, our MMA based approach has resulted in lesser number of node counts for the SDDs. Also, for all circuits, number of iterations to generate optimum variable order is lesser than the corresponding MMA.

Table 4.1 Comparison of MMA results for Multi-input Adder with Dynamic Algorithms

Benchmark Circuits	**I/P**	**O/P**	**Initial Node Count**	**WIN2**	**WIN2ite**	**WIN 3**	**SIFT**	**RANDOM**	**Proposed MA with crossover**		
									order	cycle	PMX
1-adder	3	2	8	8	8	8	8	8	**8**	**8**	**8**
2-adder	5	3	17	17	17	17	17	17	**14**	**14**	**14**
3-adder	7	4	32	32	34	32	26	34	**20**	**20**	**20**
4-adder	9	5	63	65	63	46	46	61	**30**	**38**	**32**
5-adder	11	6	126	128	126	61	55	78	**32**	**32**	**32**
6-adder	13	7	253	255	253	85	85	93	**68**	**88**	**118**
7-adder	15	8	508	510	508	94	94	264	**209**	**177**	**176**
8-adder	17	9	1027	1001	1001	87	283	845	**255**	**277**	**207**

Table 4.2 Comparison of MMA results for LGSynth93 benchmark circuits with Dynamic Algorithms

Benchmark Circuits	I/P	O/P	Initial Node Count	WIN2	WIN2ite	WIN 3	SIFT	RANDOM	Proposed MA with crossover		
									order	cycle	PMX
b12	15	9	91	86	86	65	65	76	**29**	**28**	**29**
sao2	10	4	154	149	109	91	92	132	**76**	**63**	**61**
5xp1	7	10	88	87	82	82	82	88	**50**	**53**	**51**
bw	5	28	118	113	112	106	106	118	**106**	**105**	**106**
con1	7	2	18	20	18	16	18	18	**16**	**16**	**15**
inc	7	9	81	83	80	74	73	86	**47**	**47**	**47**
sqrt8	8	4	42	40	39	37	37	48	**30**	**30**	**30**
squar5	5	8	38	38	38	38	38	38	**19**	**19**	**19**
clip	9	5	105	254	178	108	105	105	**87**	**96**	**92**
xor5	5	1	9	9	9	9	9	9	**9**	**9**	**9**
pdc	16	40	696	666	658	640	640	709	**700**	**736**	**719**
duke2	22	29	976	825	777	360	358	414	**497**	**478**	**442**
misex1	8	7	47	43	42	41	41	47	**19**	**26**	**18**
misex2	25	18	99	140	126	111	86	83	**86**	**84**	**80**
misex3c	14	14	750	719	490	454	527	844	**451**	**460**	**454**
table5	17	15	873	776	738	717	712	974	**827**	**990**	**873**
t481	16	1	32	32	32	32	32	74	**70**	**85**	**32**

From the data in Table 4.2, it is observed that MMA resulted in an average reduction of 24.03% in SDD sizes. Also the proposed MMA provides a 2.75% average improvement in node count of Benchmark circuits as compared to the best known dynamic algorithm i.e. Sift Algorithm.

In BDD each node is represented as a single 2x1 multiplexer. After synthesization of a 2x1 multiplexer in Synopsys tool, the power calculated is 762.3125 nW as shown in Table 4.3

Table 4.3 Report for Power calculation using Synopsys tool

```
****************************************
Report : power
-analysis_effort low
Design : mux_2
Version: Y-2006.06-SP4
Date : Thu May 24 12:18:01 2012
****************************************
Library(s) Used:
fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a
_c/
2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db)
Operating Conditions: TCCOM Library: fsa0a_c_generic_core_tt1p8v25c
Wire Load Model Mode: enclosed
Design Wire Load Model Library
---------------------------------------------
mux_2 G5K fsa0a_c_generic_core_tt1p8v25c
Global Operating Voltage = 1.8
Power-specific unit information:
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = Unitless
Cell Internal Power = 765.1176 nW (87%)
Net Switching Power = 97.1948 nW (13%)
---------
Total Dynamic Power = 762.3125 nW (100%)
***** End Of Report **
```

Total power dissipation of the circuit can be calculated by multiplying the power calculated for a BDD (represented as 2x1 Mux) with the node count. It can be represented as below

Total power = node count * 762.3125nw

Table 4.4 performs comparison of the Power calculated using Probabilistic techniques [14] with the existing algorithm (values calculated using Synopsys tool).The results indicate that in 80% cases, our MMA based approach using probabilistic analysis has shown lesser power dissipation.

Table 4.4 Comparison of MMA results for LGSynth93 benchmark circuits with existing algorithm

Bench-mark Circuits	I/P	O/P	Order Crossover Operator(in mW)		PMX Crossover Operator(in mW)		Cycle Crossover Operator(in mW)	
			Proposed algorithm	Existing algorithm	Proposed algorithm	Existing algorithm	Proposed algorithm	Existing algorithm
b12	15	9	**0.000330105**	0.022107	**0.000004756**	0.022107	**0.54750 x 10^-6**	0.021345
5xp1	7	10	**0.000045783**	0.038116	**0.00004578**	0.038878	**45.783 x 10^-6**	0.040403
bw	5	28	**0.015 x 10^-6**	0.080805	**0.015 x 10^-6**	0.080805	**0.015 x 10^-6**	0.080043
inc	7	9	**0.000041003**	0.035829	**0.000041003**	0.035829	**41.0032 x 10^-6**	0.035829
xor5	5	1	**0.000866615**	0.006861	**0.000866615**	0.006861	**0.000866615**	0.006861
misex2	25	18	**0.015 x 10^-6**	0.065559	**0.0297 x 10^-6**	0.060985	**0.01501 x 10^-6**	0.064034
t481	16	1	**0.126 x 10^-12**	0.053362	**2.2704 x 10^-6**	0.024394	**0.2520 x 10^-12**	0.064797
sao2	10	4	**0.0316 x 10^-6**	0.057936	**0.1238 x 10^-6**	0.046501	**0.00316 x 10^-6**	0.048026
clip	9	5	**96.88 x 10^-12**	0.065559	**0.0667 x 10^-6**	0.070133	**0.0014 x 10^-12**	0.073945
1-adder	3	2	**0.00157471**	0.006099	**0.00157471**	0.006099	**0.00157471**	0.006099
2-adder	5	3	**6.2263 x 10^-6**	0.010672	**6.2263 x 10^-6**	0.010672	**6.2263 x 10^-6**	0.010672
3-adder	7	4	**12.530 x 10^-6**	0.015246	**12.530 x 10^-6**	0.015246	**12.5303 x 10^-6**	0.015246
4-adder	9	5	**20.786 x 10^-6**	0.022869	**0.40629759**	0.024394	**0.40629759**	0.028968
5-adder	11	6	**2.5533 x 10^-6**	0.024394	**0.0316 x 10^-6**	0.024394	**0.00019 x 10^-6**	0.024394
6-adder	13	7	**2.52 x 10^-16**	0.051837	**5.131 x 10^-15**	0.089953	**1.138 x 10^-15**	0.067084

4.2 Graphical Analysis of results

A graphical view of reduction in BDD sizes in terms of node count by using different optimization algorithms has been shown in Fig. 4.1. Accordingly, it can be observed that in most of the circuits, the results provided by the proposed MMA are better as compared to other *state-of-art* algorithms.

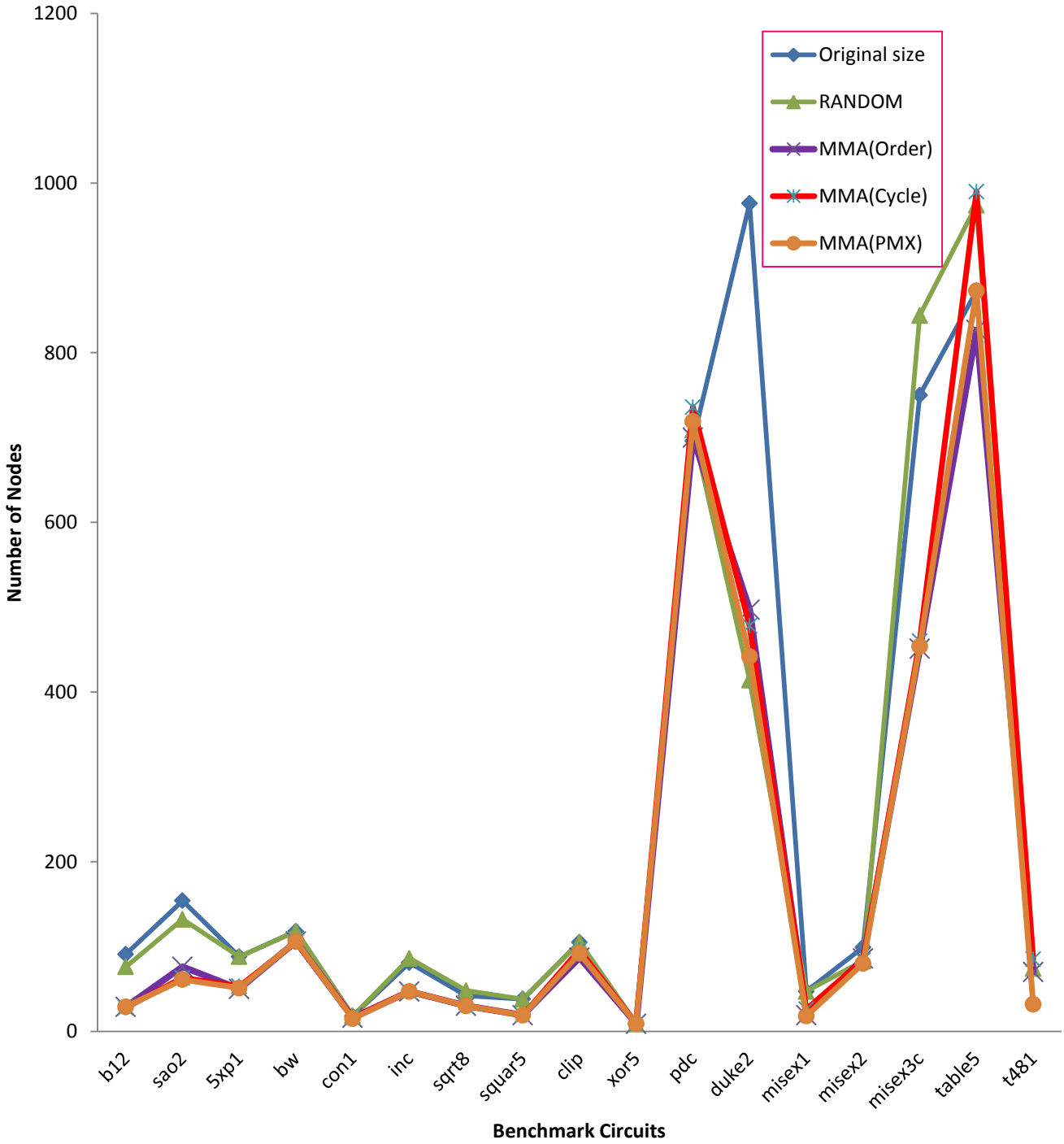

Fig 4.1 Graph showing reduction in BDD size in terms of node count using different algorithms

A graphical view of reduction of power dissipation using probabilistic technique [14] in comparison to values calculated using Synopsys tool has been shown in Fig. 4.2 and Fig. 4.3. Accordingly, it can be observed that in most of the circuits, the results provided by the proposed MMA are better as compared to other techniques implemented.

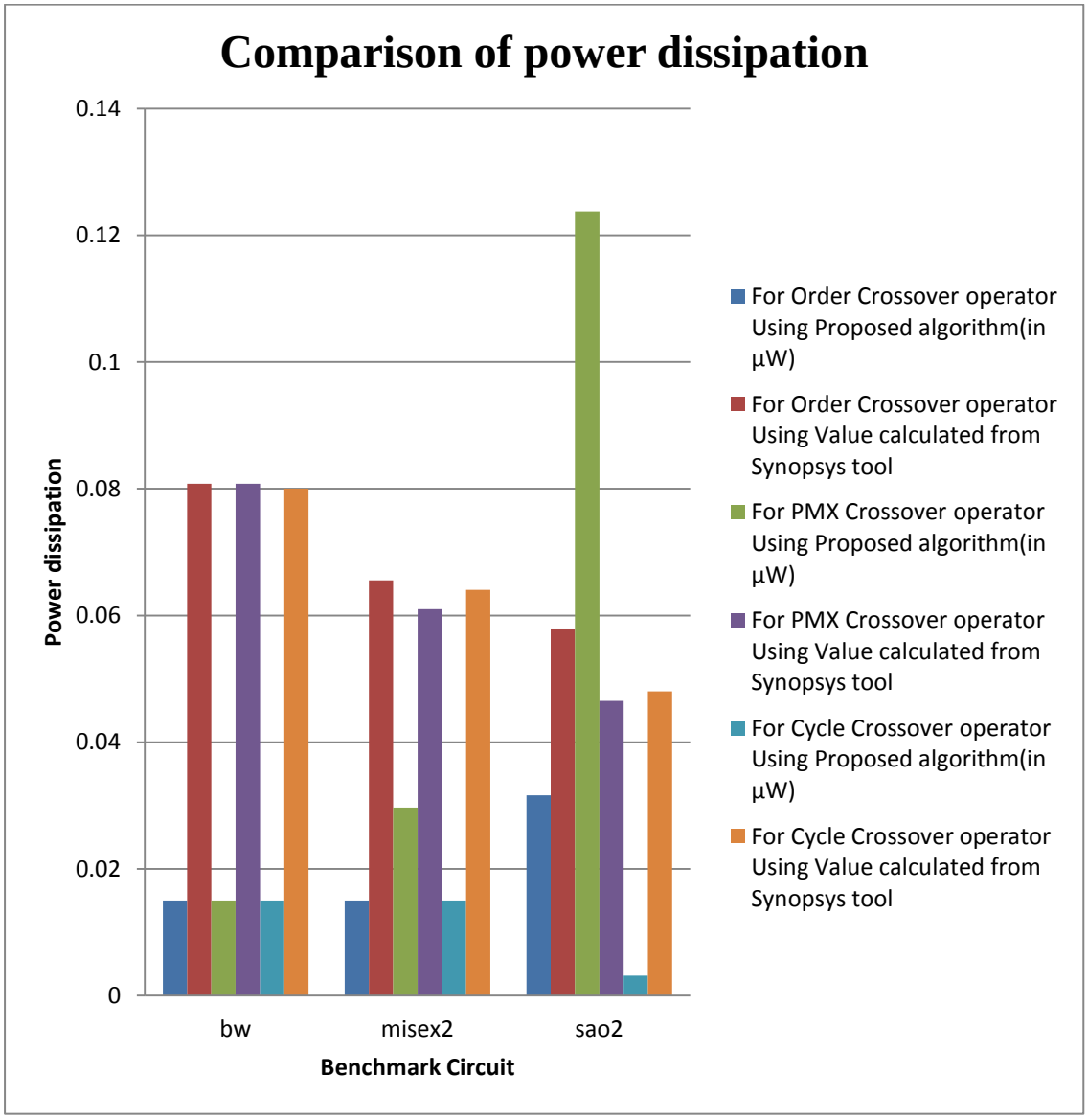

Fig 4.2 Bar Chart showing reduction in power dissipation using different crossover operator in comparison to existing algorithm for various benchmark circuits

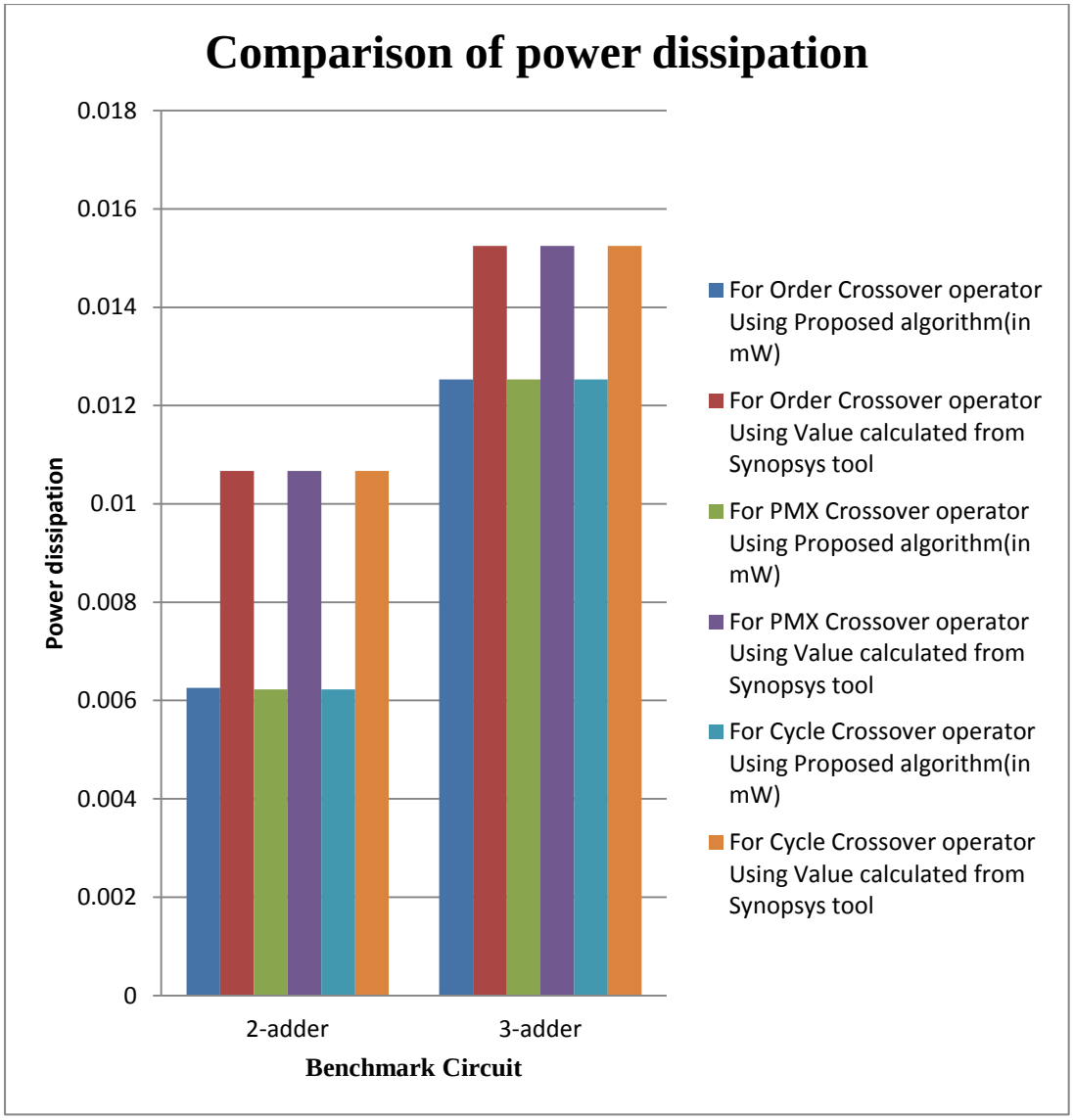

Fig 4.3 Bar Chart showing reduction in power dissipation using different crossover operator in comparison to existing algorithm for multi-input adder circuit

5. Conclusion and Future Work

5.1 Conclusion

In this thesis work, a Modified Memetic Algorithm (MMA) has been presented as one of the variable reordering algorithms for reducing the node count in shared BDDs for multi-output functions using three crossover operators and power is also estimated using Probabilistic techniques.

Through experiments, it has been observed that the results have improved significantly compared to those obtained using the Dynamic Algorithms and other existing algorithms, in terms of node count reduction, reduced power dissipation and the number of iterations. By using the proposed algorithm, an average improvement of 24.03% has been observed in the results when compared to original node count of the circuits and 2.75% average improvement w.r.t. the best known dynamic algorithm i.e. Sift algorithm. Thus, using this algorithm not only reduces the node count in SDDs as compared to other variable reordering algorithms, but at the same time, requires lesser iterations and population size to obtain the optimum result. This makes it an efficient variable reordering method.

Using Probabilistic analysis, it is observed that in more than 80% of the circuits, the results provided by the proposed MMA are better as compared to other techniques implemented.

5.2 Future Work

- BDD-based probabilistic technique can be implemented for other evolutionary and dynamic algorithms.
- Performance improvement in terms of memory requirement will be another agenda in future work.
- More Power optimization techniques can be implemented using proposed crossover operators.

References

[1] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509-516, June 1978.

[2] R.E. Bryant, "Graph based Algorithms for Boolean Function Manipulations," *IEEE Transactions on Computers*, vol. 35, no. 1, pp. 667-691, August 1986.

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

[4] Y.Y. Liu, K.H. Wang, T.T. Hwang and C.L. Liu, "Binary Decision Diagram with Minimum Expected Path Length," *Design, Automation and Test*, pp. 708-712, 2001.

[5] Y. Iguchi, T. Sasao and M. Matsuura, "Evaluation of Multiple-Output Logic Functions using Decision Diagrams," *Asian South Pacific Design Automation Conference*, pp. 312-315, 2003.

[6] 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.

[7] 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*, pp.50-54, 1991.

[8] N. Ishiura, H. Sawada and S. Yajima, "Minimization of Binary Decision Diagrams based on Exchange of Variables" *IEEE International Conference on Computer-Aided Design*, pp.472-475, 1991.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] M.T.V. Baghmisheh and M.A. Ahandani, "A Differential Memetic Algorithm," A*rtificial Intelligence Review*, vol. 41, Issue 1, pp. 129-146,January 2014.

[16] M. Kaya, "The Effects of two New Crossover Operators on Genetic Algorithm Performance," *Applied Soft Computing*, pp.881-890, January 2010.

[17] A. Banerjee, "A Novel Probabilistically-Guided Context-Sensitive Crossover Operator for Clustering," *Swarm and evolutionary computation*, pp.47-62, June 2013.

[18] M. Maruniak and P. Pistek, "Binary Decision Diagram Optimization Method based on Multiplexer Reduction Methods," *IEEE International conference on system Science and engineering*, pp.395-399, July 2013.

[19] O. Hasancebi and F. Erbatur, "Evaluation of Crossover Techniques in Genetic Algorithm based Optimum Structural Design," C*omputers and Structures*, pp.435-448, July 1999.

[20] http://en.wikipedia.org/wiki/Memetic_algorithm.

[21] M. Norman, and P. Moscato, "A Competitive and Cooperative Approach to Complex Combinatorial Search," *Proceedings of the 20th Informatics and Operations Research Meeting, Buenos Aires (20th JAIIO)*, pp. 3.15-3.29, August 1991.

[22] E. Elbeltagi, T. Hegazy and D. Grierson, "Comparison among Five Evolutionary-Based Optimization Algorithms," *International Journal of Advanced Engineering Informatics*, pp. 43-53, January 2005.

[23] N. Zhuang, M.S.T. Benten and P.Y.K Cheung, "Improved Variable Ordering of BDDs with Novel Genetic Algorithm," *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 414-417, 12-15 May 1996.

[24] W. Lenders and C. Baier, "Genetic Algorithms for the Variable Ordering Problem of Binary Decision Diagrams," *Foundations of Genetic Algorithms*, Japan: Springer, pp. 1-20, August 2004.

[25] B. Bontoux, C. Artigues and D. Feillet, "A Memetic Algorithm with a Large Neighborhood Crossover Operator for the Generalized Traveling Salesman Problem," *Computers and Operations Research*, pp. 1844-1852, May 2009.

Publications

- "Reducing node count and computational time using different Crossover Operators in Memetic Algorithm" in 3rd National Conference on Advances in Metrology, Feb 2014.
- "Effect of various Crossover operators in Memetic algorithm on Multi-input adders" in International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering, Vol. 2, Issue 3,pp.1230-1232, March 2014.