DNA Based Boolean Satisfiable Approach For FPGA Test Generation

A Thesis

Submitted in partial fulfillment of the requirements

for the award of degree of

MASTER OF ENGINEERING

in

SOFTWARE ENGINEERING

Under the supervision of

Mr. Amardeep Singh

Senior Lecturer

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Submitted By:

SHIKHA SINGH

Roll No: 8033120

COMPUTER SCIENCE & ENGINEERING DEPARTMENT,

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(DEEMED UNIVERSITY)

PATIALA- 147004

May, 2005

Abstract

The potential of DNA computing has been used to solve many computationally hard problems. FPGA Test Generation is a compute intensive problem and involves generating input patterns that give different response for a given fault in a faulty and fault-free circuit. Conventional methods for test generation involve exhaustive, random and deterministic techniques. These methods are not very effective and become very costly and time-consuming as the size of the circuit increases. The complexity of FPGA is rapidly increasing. The need for effective and economical testing has given birth to unconventional methods for it. These include methods such as Boolean Satisfiability approach, Quantum approach, Neural Computing etc.

The Boolean Satisfiability approach for test generation was proposed by Larrabee in 1992. It is neither purely structural nor an algebraic one. The approach generates test patterns in two steps: First, it extracts the formula that defines the test patterns that detect the fault. Second, it satisfies the formula using Boolean satisfiability algorithm.

In this thesis, we have explored the computational power of DNA molecules to solve the computationally hard problems, the Boolean Satisfiability approach and proposed a DNA computing based approach for FPGA test generation. The DNA based Boolean Satisfiable approach for FPGA Test Generation is efficient, faster and economical for the generation of test patterns for single-stuck-at faults and bridging faults in FPGA. The effectiveness of the approach has been demonstrated by the theoretical comparison of DNA based Boolean Satisfiable approach for FPGA Test Generation (DSTG) results with the exhaustive search.

DNA computing, also known as molecular computing, is a new computational paradigm for parallel computation, which uses the parallelism provided by DNA molecules to solve a particular problem. This new field was launched by Leonard M. Adleman in 1994. In his groundbreaking work he showed that how the biological experiments could be used to solve an instance of Hamiltonian Path Problem (HPP), which is a compute intensive

problem The encoding of the instance of the HPP was done by DNA strands and the commonly available techniques form the molecular biology were used to manipulate this encoded information

After Adleman's experiment, Lipton used the potential of DNA computing to solve another NP – Complete problem: the Satisfiability problem. The proposed DNA based Boolean Satisfiable approach for FPGA Test Generation (DSTG) algorithm exploits the Lipton model.

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled "**DNA BASED BOOLEAN SATISFIABLE APPROACH FOR FPGA TEST GENERATION**", in partial fulfillment of the requirement for the award of degree of Master Of Engineering in Software Engineering and submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision and guidance of Mr. Amardeep Singh.

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

Shikha Singh

(Shikha Singh)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Asingh

Mr. Amardeep Singh
Senior Lecturer
Computer Science & Engineering Department
Thapar Institute of Engineering &Technology
PATIALA-147004

Countersigned By:

Rsalaria 23/06/05

Mr. R.S. Salaria
Assistant Professor & Head,
Computer Science & Engineering Department,
Thapar Institute of Engineering &Technology,
Patiala.

Dr. D.S. Bawa
Dean (Academic Affairs),
Thapar Institute of Engg. and Tech.,
Patiala.

Acknowledgement

It is my proud privilege to express my regards and sincere gratitude to Mr. Amardeep Singh, Senior Lecturer, Computer Science and Engineering Department, TIET, Patiala, for valuable suggestions, expert guidance in all the time of research for and writing of this thesis. I would like to thank him for his personal involvement in guiding me right from the start to the successful completion of thesis.

I also take the opportunity to thank Mr. R. S. Salaria, Professor and Head, Computer Science and Engineering Department, TIET, Patiala, Dr. Seema Bawa, Professor, Department of Computer Science and Engineering , Mr. Rajesh Bhatia, Assistant Professor & P.G. Coordinator and Mr. Maninder Singh, Assistant Professor, Department of Computer Science and Engineering and the entire faculty and staff for their help, inspiration and moral support, which went a long way in successful completion of my thesis.

I would like to express my gratitude to my friends whose support has been a constant source of inspiration.

Shikha Singh

Place: Patiala

(Shikha Singh)

Dated:

Roll No. 8033120

M.E. II Yr

SOFTWARE ENGINEERING

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Huge financial and intellectual investments over a half-century have made electronic computers the marvels of our age. Much of our scientific, technological, military and economic future depends on the availability of an ever-increasing supply of computational power. But the incessant demand for computational power has pushed electronic technology to the limit of physical feasibility and has raised the concern that this technology may not be able to sustain our growth in this new millennium. For this reason, it becomes important to consider the future of computation and alternative means of achieving computational power. In this regard, one of the technology which has surfaced is DNA Computing.

DNA computing uses the parallelism provided by DNA molecules to solve a particular problem. This new field was launched by Leonard M. Adleman in 1994. In his groundbreaking work [10] he showed that how the biological experiments could be used to solve an instance of Hamiltonian Path Problem (HPP), which is a compute intensive problem The encoding of the instance of the HPP was done by DNA strands and the commonly available techniques form the molecular biology were used to manipulate this encoded information. In this way DNA was shown to have massively parallel processing capabilities that might allow a DNA based computer to solve hard computational problems in a reasonable amount of time.

1.1. BACKGROUND OF NP COMPLETE CLASS PROBLEMS

Problems can be ranked in difficulty according to the length of time the best algorithm will require to solve the problem on a single computer. Algorithms whose time complexity function is bounded by a polynomial function, in terms of the size of input describing the problem, are in the polynomial time class P. Such algorithms are generally

considered efficient. Any algorithm whose time complexity function cannot be so bounded belongs to the inefficient exponential class EXP. A problem is called intractable if it is so hard that no polynomial time algorithm can possibly solve it. A special class of problems, apparently intractable, is the "nondeterministic polynomial time" class, or NP. NP contains the problems for which no polynomial time algorithm to solve them is known, but that can be solved in polynomial time on a nondeterministic computer (a computer that has the ability to pursue an unbounded number of independent computational searches in parallel). The Directed Hamiltonian Path Problem is a special kind of problem in NP known as "NP-complete". An NP-complete problem has the property that every other problem in NP can be reduced to it in polynomial time. Thus, NP-complete problems are the "hardest" problems in NP.

1.2. ORIGIN OF DNA COMPUTING

DNA computing began in 1994 when Leonard Adleman [11] has first shown that computing can be done using DNA also, without using usual machine but using test tubes etc. in a biological laboratory. For this, he has chosen Hamiltonian path problem (HPP) and obtained solution using DNA experiments. Things would not have gone further if the problem he has chosen is simple but as he has taken HPP, which is an NP-Complete problem for which there is no polynomial time algorithm using conventional computer, it created an exciting and made people to think more about DNA computing. The power of the method proposed by Adleman is in the fact that tremendous parallelism can be introduced using DNA operations and that helped Adleman to solve an NP-Complete problem.

1.3. WHY DNA COMPUTING?

This is an important question. There are three reasons for using molecular biology to solve computational problems.

1. The information density of DNA is much greater than that of silicon: 1 bit can be stored in approximately one cubic nanometer. Others storage media, such as videotapes, can store 1 bit in 1,000,000,000,000 cubic nanometer.
2. Operations on DNA are massively parallel: a test tube of DNA can contain trillions of strands. Each operation on a test tube of DNA is carried out on all strands in the tube in parallel.
3. Our computers, with more and more packed onto their silicon chips are approaching the limits of miniaturization. Molecular computing may be a way around this limitation.

1.4. UNIQUENESS OF DNA COMPUTING

DNA, with its unique data structure and ability to perform many parallel operations, allows you to look at a computational problem from a different point of view. Transistor based computers typically handle operations in a sequential manner. Of course there are multi-processor computers, and modern CPUs incorporate some parallel processing, but in general, in the basic Von Neumann Architecture computer, instructions are handled sequentially. A Von Neumann machine, which is what all modern CPUs are, basically repeat the same "Fetch and execute cycle" over and over again; it fetches an instruction and the appropriate data from main memory, and executes the instruction. It does this many, many times in a row really fast. The great Richard Feynman, in his Lecture on Computation, summed up Von Neumann computers by saying, "The inside of a computer is as dumb as hell but it goes like mad!" DNA computers, however, are non von-Neumann, stochastic machines that approach computation in a different way from ordinary computers for the purpose of solving a different class of problems.

Typically, increasing the performance of silicon computing means faster clock cycles (and larger data paths), where the emphasis is on the speed of CPU and not on the size of the memory. For example, will doubling the clock speed or doubling your RAM give you better performance? For DNA computing, the power comes from the memory capacity and parallel processing. If forced to behave sequentially, DNA loses its appeal. For example, let's look at the read and write rate of DNA. In bacteria, DNA can be replicated

at a rate about 500 base pairs a second. Biologically this is quite fast (10 times faster than human cells) and considering the low error rates, an impressive achievement. But this is only 1000bits/sec, which is a snail's pace when compared to the data throughput of an average hard drive. But look what happens if you allow many copies of the replication enzymes to work on DNA in parallel. First of all, the replication enzymes can start on the second replicated strand of DNA even before they're finished copying the first one. So, already the data rate jumps to 2000 bits/sec. But look what happens after each replication is finished-the number of DNA strands increases exponentially (2^n after n iterations). With each additional strand, the data rate increase by 1000 bits/sec. So after 10 iterations, the DNA is being replicated at the rate of about 1 Mbit/sec, after 30 iterations it increases to 1000 Gbits/sec. This is beyond the sustained data rates of the faster hard drives.

1.5. THESIS ORGANIZATION

Chapter 1 of the report gives a brief introduction about DNA Computing and puts light on the NP-completeness aspect of the problem. Chapter 2 reveals the much details of DNA, its structure and properties. Biological operations, which form the basis for DNA computing, are also discussed. Literature survey including conventional methods for FPGA Test Generation, need for unconventional methods and their details has been given in the third chapter. One of the unconventional methods for FPGA Testing i.e. Boolean Satisfiability approach has been explored in detail in chapter 4.The proposed method for FPGA testing based on this method using DNA Computation has been proposed in the chapter 5. The details and explanation of the proposed algorithm has been given. The experimental results show the effectiveness of the approach. Chapter 6 concludes the work and discusses the future scope of DNA Computation.

In this thesis the potential of DNA computing has been applied to FPGA Testing. Field programmable gate arrays (FPGAs) are programmable platforms for lots of applications such as networking, signal processing, and fault tolerant computing. In an SRAM-based FPGA, all logic elements and programmable switches can be reprogrammed by loading a configuration bitstream, giving an FPGA the flexibility to implement any digital circuit on the same piece of silicon. Generally, FPGAs can be configured in an incredibly large

number of ways, in the order of billions. Since the reconfiguration time, time to load a configuration varies between few seconds up to few minutes depending on the size of FPGA. It is impossible to map all possible configurations during testing. Therefore, a set of test configurations based on a fault list must be developed in order to ensure the part is defect-free. Hence, interconnect testing is a very challenging problem covering all faults in wires and programmable switches in a reasonably small number of test configurations. Test generation is known to be a hard (NP-Complete) problem, [27] and conventionally, the generation of tests is characterized as a search of N-dimensional (0,1) state space, where N is the number of inputs in the circuit to be tested. As the number of gates on a FPGA chip increases, the generation of test vectors become more difficult. As the number of possible set of inputs increases, the number of test vectors also increases exponentially as power of two, which makes testing of FPGAs a compute intensive problem. Since this problem can be modeled as a search problem, DNA computing suits well to provide the solution to this problem.

In this work, massive parallelism provided by DNA molecules is used to solve this problem and find a test pattern .To the best of my knowledge it is first time that DNA computing has been applied to FPGA Testing.

Chapter 2

DNA Computing Concepts

2.1. GENERAL WORKING ASPECTS

Bio-molecular computers work at the molecular level. Because biological and mathematical operations have some similarities, DNA, the genetic material that encodes for living organisms, is stable and predictable in its reactions and can be used to encode information for mathematical systems.

DNA is the major information storage molecule in living cells, and billions of years of evolution have tested and refined both this wonderful informational molecule and highly specific enzymes that can either duplicate the information in DNA molecules or transmit this information to other DNA molecules. Instead of using electrical impulses to represent bits of information, the DNA computer uses the chemical properties of these molecules by examining the patterns of combination or growth of the molecules or strings. DNA can do this through the manufacture of enzymes, which are biological catalysts that could be called the 'software', used to execute the desired calculation.

DNA computers use deoxyribonucleic acids A (adenine), C (cytosine), G (guanine) and T (thymine) [16] as the memory units and recombinant DNA techniques already in existence carry out the fundamental operations. In a DNA computer, computation takes place in test tubes or on a glass slide coated in 24K gold. The input and output are both strands of DNA, whose genetic sequences encode certain information. A program on a DNA computer is executed as a series of biochemical operations, which have the effect of synthesizing, extracting, modifying and cloning the DNA strands. Their potential power underscores how nature could be capable of crunching number better and faster than the most advanced silicon chips.

2.2. INFORMATION STORAGE AND PROCESSING CAPABILITIES

Nucleic Acids are used because of density, efficiency and speed. DNA molecules can store far more information than any existing computer memory chip. This means that DNA computing is a far denser packing of molecular information compared with silicon-based computers. A single bacterium cell measures just a micron square - about the same size as a single silicon transistor - but holds more than a megabyte of DNA memory and has all the computational structures to sense and respond to its environment. To try to put this in some understandable perspective, it has been estimated that a gram of DNA can hold as much information as a trillion CDs.

So DNA molecules would be like mega-memory. In a biochemical reaction hundreds of trillions of DNA molecules can operate in parallel. DNA computers could store a bit, 0 or 1, of data in one cubic-nanometer, one-trillionth the size of the conventional computer's electronic storage. Thus a DNA computer could store massive quantities of information in the space a standard computer would use to store much less. A pound of DNA could contain more computer memory than all the electronic computers ever made. It would be about twice as fast as the fastest supercomputer, performing more than 2,000 instructions per second. DNA computers also require miniscule amounts of energy to perform.

2.3. EFFICIENCY

In both the solid-surface glass-plate approach and the test tube approach, each DNA strand represents one possible answer to the problem that the computer is trying to solve. The strands have been synthesized by combining the building blocks of DNA, called nucleotides, with one another, using techniques developed for biotechnology. The set of DNA strands is manufactured so that all conceivable answers are included. Because a set of strands is tailored to a specific problem, a new set would have to be made for each new problem.

Most electronic computers operate linearly and they manipulate one block of data after another, biochemical reactions are highly in parallel: a single step of biochemical operations can be set up so that it affects trillions of DNA strands. While a DNA computer takes much longer than a normal computer to perform each individual calculation, it performs an enormous number of operations at a time and requires less energy and space than normal computers. 1000 liters of water could contain DNA with more memory than all the computers ever made, and a pound of DNA would have more computing power than all the computers ever made.

Obviously if you want to perform one calculation at a time, DNA computers are not a viable option. When Adleman derived an optimal solution to a seven-city traveling-salesman problem, it took approximately one week. Unfortunately, you can solve the same problem on a piece of paper in about an hour or by a digital computer in a few seconds. But when the number of cities is increased to just 70, the problem becomes intractable for even a 1000-Mips supercomputer.

The only fundamental difference between conventional computers and DNA computers is the capacity of memory units: electronic computers have two positions (on or off), whereas DNA has four (C, G, A or T). The study of bacteria has shown that restriction enzymes can be employed to cut DNA at a specific word (W). Many restriction enzymes cut the two strands of double-stranded DNA at different positions leaving overhangs of single-stranded DNA. Two pieces of DNA may be rejoined if their terminal overhangs are complementary. Complements are referred to as 'sticky ends'. Using these operations, fragments of DNA may be inserted or deleted from the DNA.

Most of the possible answers are incorrect, but one or a few may be correct, and the computer's task is to check each of them and remove the incorrect ones using restrictive enzymes. The DNA computer does that by subjecting all of the strands simultaneously to a series of chemical reactions that mimic the mathematical computations an electronic computer would perform on each possible answer. When the chemical reactions are complete, researchers analyze the strands to find the answer .For instance, by locating the longest or the shortest strand and decoding it to determine what answer it represents.

2.4. DNA OPERATIONS

All models of DNA computation apply a specific sequence of biological operations to a set of strands. These operations are commonly used by molecular biologists. Some operations are specific to certain models of DNA computation.

2.4.1 Synthesis

A desired strand of DNA can be synthesized [12] in lab. This is possible for strands up to a certain length. Longer 'random' strands are available. They consist of DNA sequences that have been cloned from many different organisms. The synthesizer is supplied with the four nucleotide bases in solution, which are combined according to a sequence entered by the user. The instrument makes millions of copies of the required oligonucleotides and places them in solution in a small vial.

2.4.2 Denaturing, annealing and ligation

Double-stranded DNA may be dissolved into single strands (or denatured [12]) by heating the solution to a temperature determined by the composition of the strand. Heating breaks the hydrogen bonds between complementary strands (Fig.2.1).

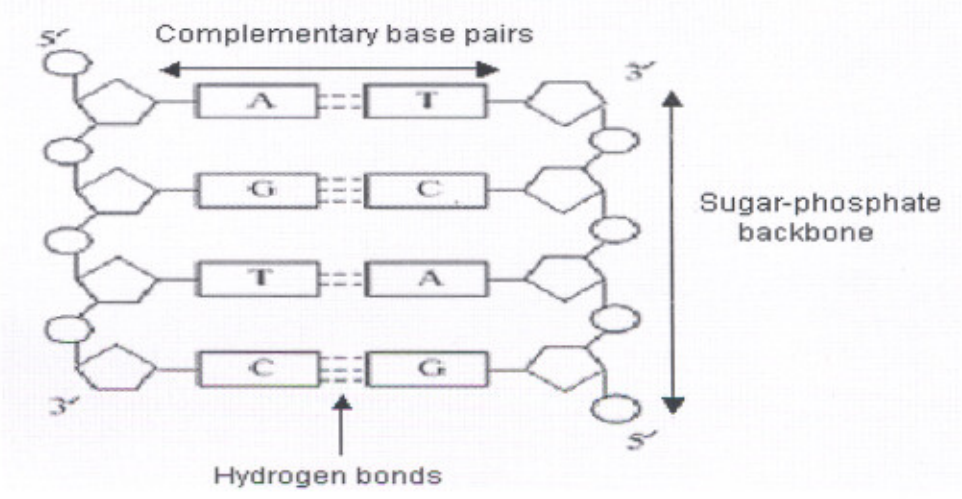

Fig. 2.1: Detailed structure of Double Stranded DNA

Since the hydrogen bonds between strands are much weaker than the covalent bonds within strands, the strands remain undamaged by this process. Since a G-C pair is joined by three hydrogen bonds, the temperature required to break it is slightly higher than that for an A-T pair, joined by only two hydrogen bonds. This factor was taken into account when designing sequences to represent computational elements.

Annealing is the reverse of melting, whereby a solution of single strands is cooled, and allowing complementary strands to bind together (Fig. 2.2).

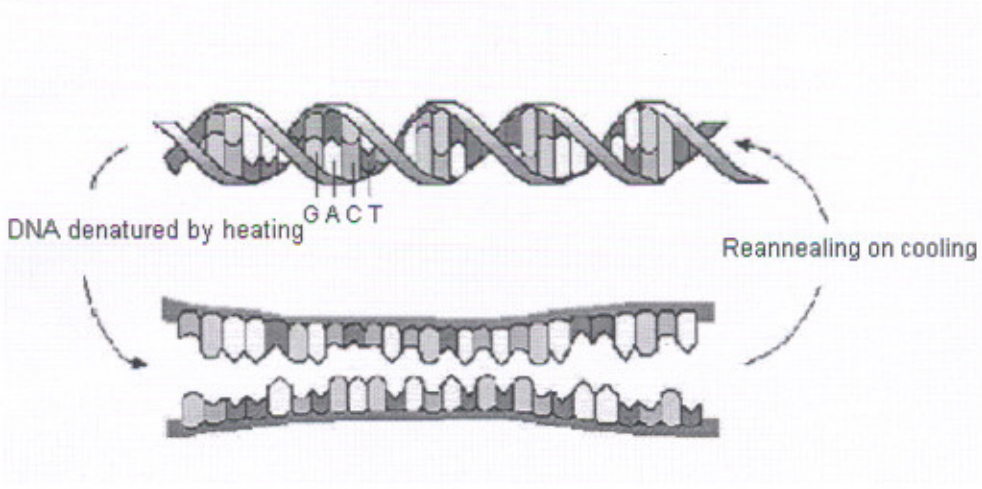

Fig. 2.2: DNA melting and annealing

In double-stranded DNA, if one of the single strands contains a discontinuity (i.e, one nucleotide is not bonded to its neighbor), then this may be repaired by DNA ligase, which is an enzyme which helps in joining two DNA strands or pairs of nucleotides. The ligation process is depicted in Fig 2.3.

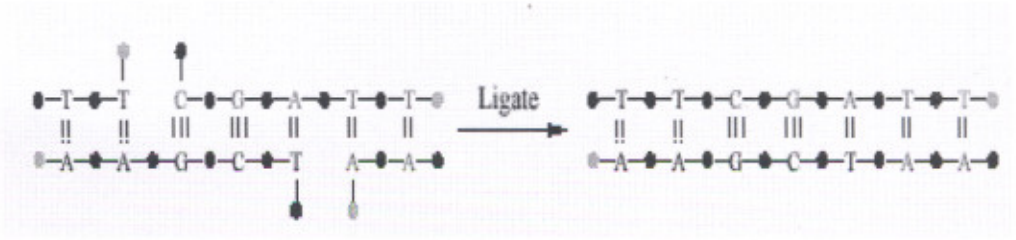

Fig. 2.3: Ligation

2.4.3 Hybridization separation

Separation by hybridization [12] is an operation often used in DNA computation, and involves the extraction from a test tube of any single strands containing a specific short sequence (e.g., extract all strands containing the sequence TAGACT). If we want to extract single strands containing the sequence X, we first create many copies of its complement. We attach to these oligonucleotides a biotin molecule which binds in turn to a fixed matrix. If we pour the contents of the test tube over this matrix, strands containing X will anneal to the anchored complementary strands. Washing the matrix removes all strands that do not anneal, leaving only strands containing X. These may then be removed from the matrix.

2.4.4 Gel-Electrophoresis

Gel electrophoresis [12] is an important technique for sorting DNA strands by size. Electrophoresis is the movement of charged molecules in an electric field. Since DNA molecules carry negative charge, when placed in an electrical field they tend to migrate towards the positive pole. The rate of migration of a molecule in an aqueous solution depends on its shape and electrical charge. Since DNA molecules have the same charge per unit length, they all migrate at the same speed in an aqueous solution. However, if electrophoresis is carried out in a gel (usually made of agarose, polyacrylamide or a combination of the two) the migration rate of a molecule is also affected by its size. This is due to the fact that the gel is a dense network of pores through which the molecules must travel. Smaller molecules therefore migrate faster through the gel, thus sorting them according to size. A simplified representation of gel electrophoresis is depicted in Fig. 2.4. The DNA was placed in a well cut out of the gel, and a charge applied. Once the gel has been run (usually overnight), it is necessary to visualize the results. This is achieved by staining the DNA with the fluorescent dye ethidium bromide and then viewing the gel under ultraviolet light. At this stage the gel is usually photographed for convenience. Gels are interpreted as follows; each lane corresponds to one particular sample of DNA.

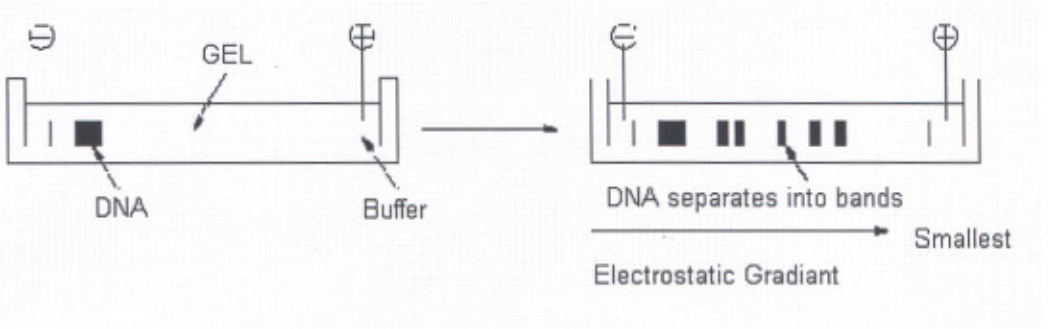

Fig. 2.4: Gel Electrophoreses Process

We can therefore run several tubes on the same gel for the purposes of comparison. Lane 7 is known as the marker lane; this contains various DNA fragments of known length, for the purposes of calibration. DNA fragments of the same length cluster to form visible horizontal bands, the longest fragments forming bands at the top of the picture, and the shortest at the bottom. The brightness of a particular band depends on the amount of DNA of the corresponding length present in the sample. Larger concentrations of DNA absorb more dye, and therefore appear brighter. One advantage of this technique is its sensitivity as little as $0{:}05^{l}g$ of DNA in one band can be detected as visible fluorescence. The size of fragments at various bands is shown to the right of the marker lane, and is measured in base pairs (bp).

2.4.5 Primer extension and PCR

The DNA polymerases perform several functions, including the repair and duplication of DNA.Given a short primer oligonucleotide, p, in the presence of nucleotide triphosphates, the polymerase extends p (always in the *50 ! 30* direction) if and only if p is bound to a longer template oligonucleotide, t. For example, in Fig. 2.5(a), p is the oligonucleotide TCA which is bound to t; ATAGAGTT. In the presence of the polymerase, p is extended by a complementary strand of bases to the 3' end of t (Fig.2.5 (b)). Another useful method of manipulating DNA is the Polymerase Chain Reaction, or PCR [12]. PCR is a process that quickly amplifies the amount of a specific molecule of DNA in a given solution using primer extension by polymerase. Each cycle of the reaction doubles the quantity of this molecule, giving an exponential growth in the number of strands. In order to target specific molecules we need to know their "start" and "end" sections. A common

problem in DNA computation is how to read-out the final solution to a problem encoded as a DNA strand, as the laboratory steps carried out may result in a very dilute solution. PCR solves this problem: if a sought after molecule is present in the solution, then it will be hugely (exponentially) multiplied so that the volume of the solution will "visibly" grow, this solves then the Detection problem. According to the PCR resource, the cycle can be repeated more than thirty times. With thirty cycles taking just three hours, one strand of DNA can turn into a million strands.

Fig. 2.5: Primer Extension by Polymerase

2.4.6 Extraction

Given a test tube $T1$ and a strand S, it is possible to extract all the strands in $T1$ that contain S as a subsequence and to separate them from those that do not contain it.

2.4.7 Union

Given two or more test tubes, say $T1; T2; : : : ; Tn$, it is possible to put in a new test tube the union [6] of all the strands contained in $T1; T2; : : : ; Tn$.

2.4.8 Detection

Confirm presence/ absence of DNA in a given test tube.

2.4.9 Restriction Enzymes

There are many kind of enzymes that are the molecules capable of operating on other molecules. Some of them are the restriction enzymes that cut DNA double strands where specific subsequence appears. Any double stranded DNA that contains the restriction site with in its sequence is cut by the enzyme [12] at that point. For example, the double stranded DNA in fig 2.6(a) is cut by restriction enzyme Sau3AI, which recognizes the restriction site GATC. The resulting DNA is depicted in fig 2.6(b). The cleavage here generates sticky (cohesive) ends. Such ends are important in DNA manipulation, because they allow catenation of DNA molecules if they have complimentary sticky ends.

Fig. 2.6: Double stranded DNA being cut by Sau3AI

2.5. SUCCESS OF DNA COMPUTING

The first applications were "brute force" solutions in which random DNA molecules were generated, and then the correct sequence was identified. The first problems solved by DNA computations involved finding the optimal path by which a traveling salesman could visit a fixed number of cities once each. Recent works have shown how DNA can be employed to carry out a fundamental computer operation, addition of two numbers expressed in binary (Bancroft) and several other problems like, Max-Clique Problem, Graph-Coloring Problem, Satisfiability Problem, Bounded Post Corresponding problem etc.

2.6. HOW DNA COMPUTERS WORKS?

DNA computers work by encoding the problem to be solved in the language of DNA: the base-four values A, T, C and G. Using this base-four number system, the solution to any conceivable problem can been coded along a DNA strand like in a Turing machine tape. Every possible sequence can be chemically created in a test tube on trillions of different DNA strands, and the correct sequences can be filtered out using genetic engineering tools.

2.7. COMPARISON OF DNA AND CONVENTIONAL ELECTRONIC COMPUTERS

As we have seen the concepts and characteristics of DNA Computer, we can now compare the DNA Computers with Conventional Electronic Computers.

2.7.1. Similarities

2.7.1.1 Transformation of Data

Both DNA computers and electronic computers use Boolean logic (AND, OR, NAND, NOR) to transform data. The logical command "AND" is performed by separating DNA strands according to their sequences, and the command "OR" is done by pouring together DNA solutions containing specific sequences. For example, the logical statement "X or Y" is true if X is true or if Y is true. To simulate that, the scientists would pour the DNA strands corresponding to "X" together with those corresponding to "Y".

2.7.1.2. Manipulation of Data

Electronic computers and DNA computers both store information in strings, which are manipulated to do processes. Vast quantities of information can be stored in a test tube. The information could be encoded into DNA sequences and the DNA could be stored. To retrieve data, it would only be necessary to search for a small

part of it - a key word, for example, by adding a DNA strand designed so that its sequence sticks to the key word wherever it appears on the DNA.

2.7.1.3. Computation Ability

All computers manipulate data by addition and subtraction. A DNA computer should be able to solve a satisfiability problem with 70 variables and 1,000 AND-OR connections. To solve it, assign various DNA sequences to represent 0's and 1's at the various positions of a 70 digit binary number. Vast numbers of these sequences would be mixed together, generating longer molecules corresponding to every possible 70-digit sequence.

2.7.2. Differences

2.7.2.1. Size

Conventional computers are about 1 square foot for the desktop and another square foot for the monitor. One new proposal is for a memory bank containing more than a pound of DNA molecules suspended in about 1,000 quarts of fluid, in a bank about a yard square. Such a bank would be more capacious than all the memories of all the computers ever made. The first ever-electronic computer took up a large room whereas the first DNA computer (Adleman) was 100 micro liters. Adleman dubbed his DNA computer the TT-100, for test tube filled with 100 micro liters, or about one-fiftieth of a teaspoon of fluid, which is all it took for the reactions to occur.

2.7.2.2. Speed

Conventional computers can perform approximately 100 MIPS (millions of instruction per second). Combining DNA strands as demonstrated by Adleman, made computations equivalent to 10^9 or better, arguably over 100 times faster than the fastest computer. The inherent parallelism of DNA computing was staggering.

2.7.2.3. Minimal Storage Requirements

DNA stores memory at a density of about 1 bit per cubic nanometer where conventional storage media requires 1012 cubic nanometers to store 1 bit. In essence, mankind's collective knowledge could theoretically be stored in a small bucket of DNA solution.

2.7.2.4. Minimal Power Requirements

There is no power required for DNA computing while the computation is taking place. The chemical bonds that are the building blocks of DNA happen without any outside power source. There is no comparison to the power requirements of conventional computers.

2.8 ADVANTAGES

1. Perform millions of operations simultaneously.
2. Generate a complete set of potential solutions.
3. Conduct large parallel searches.
4. Efficiently handle massive amounts of working memory.
5. The clear advantage is that we have a distinct memory block that encodes bits.
5. The differentiation between sub sequences denoting individual bits allows a natural border between encoding sub-strands.
6. Using one template strand as a memory block also allows us to use its compliment as another memory block, thus effectively doubling our capacity to store information.

2.9. DRAWBACKS

Generating solution sets, even for some relatively simple problems, may require impractically large amounts of memory (lots and lots of DNA strands are required).
Many empirical uncertainties, including those involving: actual error rates, the generation of optimal encoding techniques, and the ability to perform necessary bio-operations conveniently in vitro (for every correct answer there are millions of incorrect paths generated that are worthless).
This is a rather good encoding, however, as we increase the size of our memory, we have to ensure that our sub-strands have distinct complements in order to be able to "set" and "clear" specific bits in our Memory.

2.10. ERRORS IN DNA COMPUTING

The models used by scientists to study real systems. Studying usually imply many simplifications in order to be able to manage the complexity of the real systems. DNA structure is one of the most complex known elements, but with an incredible simplicity at a first look. DNA is a three-dimensional, polar, set of linked oligonucleotides in a double helix format, with different behaviour due to temperature, concentration, strands' length and other reaction's conditions. Depending on the conditions under which the DNA reactions occur, two oligonucleotides can hybridise without exact matching between their base pairs [30]. The mechanism of failures in matching bases depends on reaction conditions, the most important one being the temperature. Both Adleman and Lipton used random encoding under the basic assumption that random encoding method was adequate for their purposes. Nevertheless, this simplification might be not the best choice in order to maximise the reliability of the experiment. In the following we summarise some of the mismatching problems that can occur in a reaction.

2.10.1. False Positive Match or Partial Match

The Watson-Crick complement is not universally respected. DNA models are usually error free, but this is not realistic. Synthesis itself is a process that could result in a certain

percentage of strands imperfection: synthesizing a 20-mer, could result in (depending on the synthesis process and relative cost) some percent of 19-mer or 21-mer. Sometimes a biological manipulation could fail in 10% of cases; this error could be acceptable for some laboratories or for some kind of experiments, but not as a result of a computational process. Partial match is a typical error when the Watson-Crick complement is not respected, for example, having a double strand with (*A-T, T-A, C-C, G-C*) couples.

2.10.2. Bubble Match

We have a situation of bubble match when two strands of different length match for the external parts of the strands itself: the longest strand tend to link to the other only for the "tails" creating a bubble in the central part:

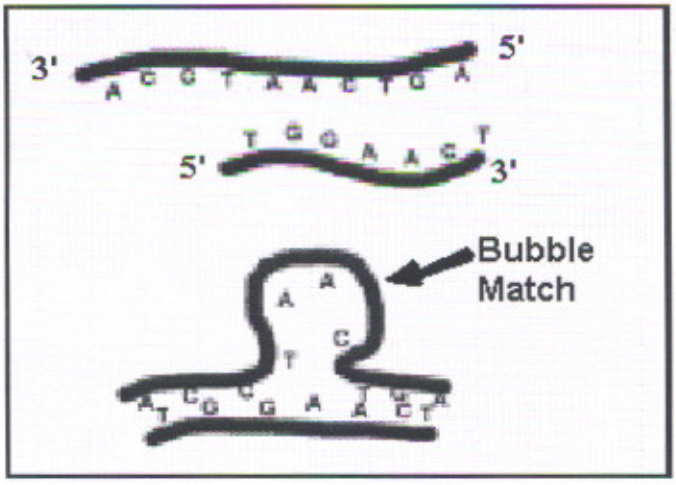

Fig. 2.7: Bubble Match

Creating a simple database of subsequences it is possible to reduce spurious matching within strands' hybridization. Since a complete search may be quite complex and the result not very significant, to improve tools' performance it is better to reduce the search space for fixed length of sub chains.

2.10.3. Cross Match

This is an abnormal situation when two double strands have a kind of intersection and are joined in a structure, such the one represented below, where two double strands are crossed to form two double helixes. The possible presence of this mismatch is not easily

detectable in the design process, since the search has to analyze DNA three-dimensional properties.

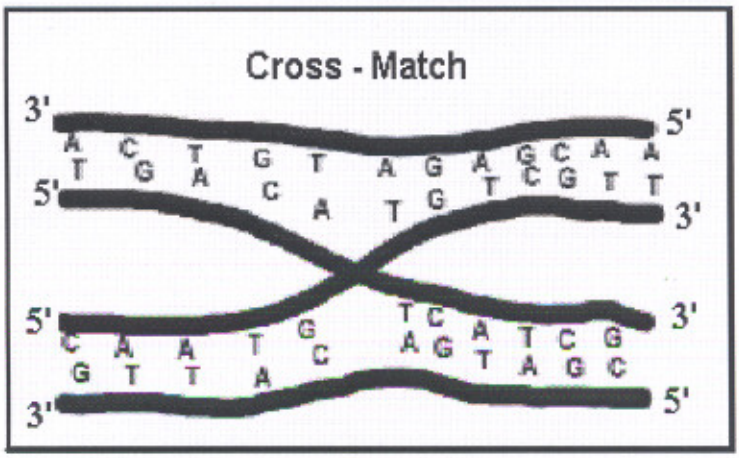

Fig. 2.8: Cross Match

2.10.4. Slide Match

This is the normal situation of two single strand matching. Problems arise when strands match for a random number of bases. We could prevent this particular situation designing encoding scheme for the oligonucleotides.

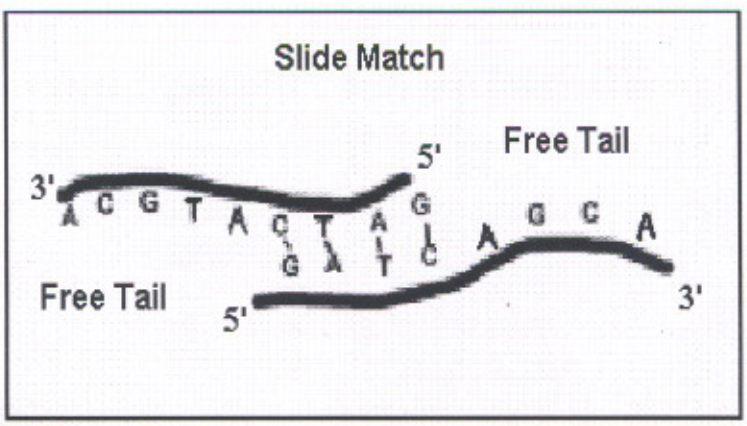

Fig. 2.9: Slide Match

2.10.5. Undesired Annealing

Most of the causes of errors in biological computations - not only in the extraction operation - are related to the possibility of undesired annealing: a strand V could anneal with one that is similar to V, but it is not the right one. In this case, we talk about partial

matches. Another kind of undesired matching could happen between two shifted" strands: for example, a strand X could partially anneal with a strand Y. Finally, a strand could anneal with itself, losing its linear structure.

Chapter 3

Literature Survey

The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. If integrated circuit testing is difficult, Field Programmable Gate Arrays (FPGAs) represent a special and particularly difficult type of digital circuit. Because FPGA devices are highly configurable, it is often difficult to isolate and exercise all elements of the architecture. In addition, modern FPGA devices represent some of the largest, most complex integrated circuit devices available.

3.1 TEST GENERATION

Test generation is the task of finding a set of test vectors that will fully test the circuit for the given set of faults. Automatic Test Pattern Generation (ATPG) algorithms can be programmed on digital computers for generating test vectors. An input vector is a set of values for all primary input signals. A fault is a fabrication-related failure. The generation of a test for a given fault involves searching among the possible input vectors of the circuit until a test vector is found. The test vector is an input vector that differentiates the fault –free circuit from the faulty one. Primary inputs are the only points where we can apply test signals to the circuit and primary outputs are the only points where we can observe the effect of the fault. Thus, a test for the given fault causes the logic value of at least one primary output of the faulty circuit to differ from its expected value in the fault-free circuit.

3.1.1 Definitions

3.1.1.1 Test Vector

An input vector for the circuit-under-test that causes the presence of a fault to be observed at a primary output.

3.1.1.2 Automatic Test Pattern Generation

The process of generating a test pattern for a specific fault using some type of algorithm.

3.1.1.3 Sensitization

The process of driving the circuit to a state where the fault causes an actual erroneous value in the device at the point of the fault. E.g., for single stuck-at faults, driving the node to the value opposite to the stuck-at value.

3.1.1.4 Propagation

The process of driving the circuit to a state where the error becomes observable at the primary outputs.

3.1.1.5 Justification

The process of determining the input combination necessary to drive an internal circuit node to a specified value (consistency).

3.1.1.6 Fault Coverage

The percentage of total faults for which test pattern have been generated.

$$\text{Fault coverage} = 100 * \frac{\text{Number of detected faults}}{\text{Total number of faults in CUT}}$$

3.1.1.7 Fault Efficiency

The percentage of faults that either are detected or PROVEN redundant (usually used to measure the effectiveness of a test generator):

$$\text{Fault efficiency} = 100 * \frac{\text{no. of detected faults + no. of redundant faults}}{\text{Total number of faults in the CUT}}$$

3.1.1.8 Controllability

It is the ability to establish a specific signal value at each node in a circuit by setting values on the circuit's input. A testability metric that measures the difficulty in driving a node to a specific value.

3.1.1.9 Observability

It's the ability to determine the signal value at any node in a circuit by controlling the circuit's input and observing its output. A testability metric that measures the difficulty in propagating the value on node to a primary output.

3.1.1.10 Predictability

It is the ability to obtain known output values in response to given input stimuli. Some factors affecting predictability are the initial state of a circuit, races, hazards, and free-running oscillators.

3.1.2 Basic Issues of Test Generation

Test generation is a complex problem with many interacting aspects. The most important are as given in [13]:

1. The cost of Test Generation
2. The quality of Test Generation
3. The cost of applying the test

The cost of test generation depends on the complexity of test generation method. Random test generation (RTG) is a simple process that involves only generation of random vectors. However, to achieve a high-quality test, we need a large set of random vectors. Even if the test generation is simple, determining the test quality may be an expensive process. Moreover, a longer test costs more to apply because it increases the time of testing experiment and the memory requirements of the tester. The test generation cost also depends on the complexity of the circuit to be tested. There are methods for reducing the complexity for testing purposes referred to as *design for testability* techniques.

3.1.3 Fault Modeling

Testing that uses all possible input vectors will, of course, reveal any faulty circuit. However, such a procedure will be too expensive for large circuits with many primary inputs (a circuit with n primary input signals will have 2^n possible input vectors).To simplify testing of large circuits we make assumptions about the possible failures. In most cases, one is not concerned with discovering the exact physical failure; what is desired is merely to determine the existence of any physical failure. One approach is to describe the effects of physical failures at some higher level (logic, register transfer, functional block etc).This abstraction is called fault modeling.

There are many benefits of modeling physical faults as logical faults[13]. First, the problem of fault analysis becomes a logical rather than a physical problem; also its complexity is greatly reduced since many different physical faults may be modeled by the same logical fault. Second, logical fault models are technology-independent in the sense that the same fault model is applicable to many technologies. Hence, testing and diagnosis methods developed for such a fault model remain valid despite changes in technology. And third, tests derived for logical faults may be used for physical faults whose effect on circuit behavior is not completely understood or is too complex to be analyzed [9].

1. Types of Fault models

a. Stuck-at Faults

- Single stuck-at fault
- Multiple stuck-at faults

b. Stuck open faults

c. Bridging faults

d. Delay faults

a. Single stuck-at Fault Model

- Only one line in the circuit is faulty at a time

- Fault is permanent as opposed to transient
- Effect of fault is as if the faulty node is tied to either Vcc(s-a-1) or Gnd (s-a-0)
- The function of the gates in the circuit is unaffected by the fault.

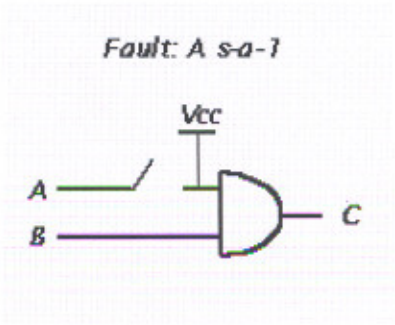

Fig. 3.1: AND gate 'A' input Stuck at-1

Table. 3.1: Truth table for AND gate s-a-1

A	B	C (Fault free O/P)	C (Faulty O/P)
0	0	0	0
0	1	0	1←
1	0	0	0
1	1	1	1

Advantages

- Can be applied at the logic level or module level
- Reasonable numbers of faults 2n(n=number of circuit nodes)
- Algorithms for automatic test pattern generation (ATPG) and fault simulation are well developed and efficient.
- Research indicates that the single stuck-at fault model covers about 90% of the possible manufacturing defects in CMOS circuits.
- Source-drain shorts, oxides pinholes, missing features, diffusion contaminants, metallization shorts etc.

- Other useful fault models (stuck-open, bridging faults) can be mapped into(sequences of) stuck-at faults

Disadvantages

- Does not cover all defects in CMOS or other devices
- Requires a lower level circuit description (transistor level), at least for development of fault list.

b. Multiple Stuck-at Fault Model

- Same as single stuck-at faults except 2 or more lines in the circuit can be faulty at the same time.
- If used in conjunction with single stuck-at faults, it covers a greater percentage of physical defects.

Disadvantages

- A larger no of faults 3^n-1 (n= no. of circuit nodes).
- Algorithms for ATPG and fault simulation are much more complex and not as well developed.

c. Stuck-open Fault Model

- A single physical line in the circuit is broken
- The resulting unconnected node is not tied to either Vcc or Gnd

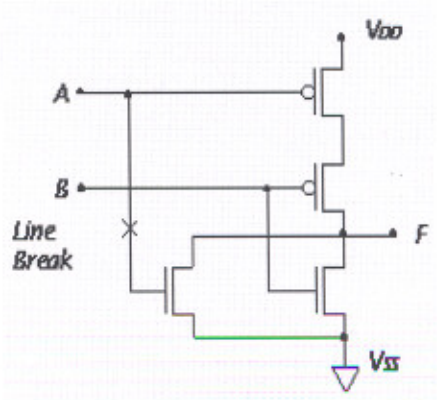

Fig.3.2: 'A' Input to NMOS is broken (stuck-open) resulting a "memory effect"

The stuck-open fault is popular for use in CMOS circuits. When a defect occurs that breaks a line internal to a CMOS gate, it can result in a "memory effect." For example, in this circuit, if the output is driven high with a 00 input, and then a 10 input is applied, then the output will stay high (its previous value). However, if the output is driven low with a 01 input, and then the 10 input is applied, then the output will stay low (its previous value) which appears OK.

d. Bridging Faults Model

- Two nodes of the circuit are shorted together
- Usually assumed to be a low resistance path (hard short).

Bridging fault models can be used to increase defect coverage. The major disadvantage is, again, the number of faults considered and the impact on test generation time.

e. Delay Fault Model

- The logic function of the circuit-under-test is error free
- Some physical defects, such as process variations, etc., make some delays in the circuit-under-test greater than some defined bounds.

- Two delay fault models are typically used:
- Gate delay or transitional fault model
- Path delay fault model

Even though a circuit doesn't have any defects that cause incorrect function, it may contain defects that cause slow function. The delay fault models are attempts to model these types of defects and their effect on the circuit. The two models currently used are the gate delay or transitional fault model, where a single gate is assumed to take too long to produce an output, and the path delay fault model, where certain paths in the circuit may take too long to be exercised.

f. Path Delay Fault Model

This fault model considers paths in the circuit and tests to see if any path delays exceed some DPmax. This algorithm overcomes a potential problem with the transitional fault mode. That is, that the delay of a faulty gate can be compensated by gates in the propagation path that have faster-than-typical delays.

This delay fault model is also consistent with a statistical design philosophy. A statistical design philosophy recognizes that the delays of gates in a circuit are usually not all worst case, but that they fall within a small "typical" range. Using this knowledge, a greater clock speed can be specified by determining the typical delays for all paths in the circuit. This delay testing method is a form of performance verification.

3.1.4 Conventional Test generation Techniques

A wide range of techniques has been proposed for combinational circuit test generation.

3.1.4.1 Exhaustive Approach

In this approach, all the possible combinations of input test vectors are generated and applied to the circuit under test (CUT). If a circuit has n number of inputs, then the possible combination of input vectors will be 2^n. This technique offers 100% fault coverage with a low computational cost. However, this method is not practicable for circuits with higher number of inputs (e.g. >20). If the number of inputs is high, then test length becomes very long and it takes a much longer time for test generation. So exhaustive approach is not efficient test generation method for fpgas.

3.1.4.2 Pseudo-exhaustive approach

In pseudo-exhaustive approach, circuits are logically partitioned into smaller parts and then each part is tested exhaustively by much fewer test vectors. Different algorithms have been proposed for circuit-partitioning in order to reduce hardware overhead and testing time. Although pseudo-exhaustive method achieves the benefits of exhaustive testing by using far fewer test patterns but with the increase of complexities in circuits, it leads to large test sets. Sometimes physical segmentation of the circuit is necessary in this approach. Moreover, hardware implementation of pseudo-exhaustive pattern generator is difficult and most of the design does not lead to minimal test set.

3.1.4.3 Deterministic approach

In deterministic approach, CUT is analyzed at the beginning of the test and then test vectors are generated using any suitable algorithm for deterministic test pattern generation. Different algorithms have been proposed for deterministic test pattern generation such as Exclusive-OR, **D-algorithm**, **Path-Oriented Decision Making (PODEM)** and **FAN** oriented test pattern (FAN). Deterministic test pattern enables error signals, generated due to presence of faults, and propagates them to some observable outputs from the faulty nodes or lines. This method guarantees full fault coverage but the

increasing densities in the circuit lead to large test data volumes and computational complexities.

3.1.4.4 Pseudo-random approach

Pseudo-random approach is now an established technique for testing ICs at low cost. In this approach, a set of pseudo-random vectors (PRV) is generated randomly from 2^n possible input patterns (n=number of inputs). LFSR is commonly used for test pattern generation because it has a simple structure and can also be used as output response analyzer. IC testing using random test pattern has been investigated. The main advantage of this approach is that random pattern generation circuitry is simple and a large number of tests can be generated using smaller data storage. The disadvantage of this approach is the length of the test set that detects a set of faults is much larger (usually 10 times or more) than a deterministically generated test set for the same faults. Besides this, the quality of the LFSR-generated test set depends on the CUT. Fig. 9 shows the relationship between fault coverage and number of applied random test vectors.

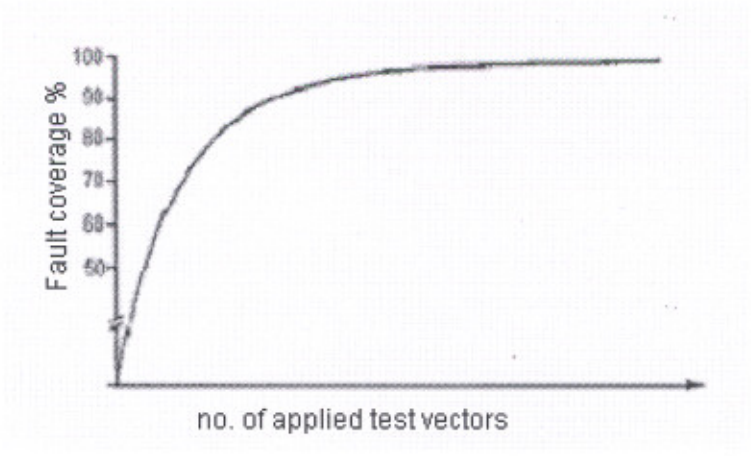

Fig. 3.3: Fault coverage versus no. of random test vectors

It is seen from Fig. 9 that almost 90% fault coverage can be achieved using a few test vectors. A large number of test vectors are then needed to detect the remaining faults. Another implication of this approach is that there are faults in some circuits (for example,

high fan-in AND gate) known as random pattern resistant faults where acceptable fault coverage cannot be achieved even after applying a large number of test patterns.

3.1.4.5 Weighted random approach

To overcome the drawbacks of low-fault coverage due to random pattern resistant faults in IC testing, weighted random approach has been proposed. It has been shown that biased or weighted PRV can test PRV resistant faults more efficiently at a lower number of test vectors. The disadvantage of this approach is that pre-processing is necessary to calculate the signal probabilities for every input and to generate necessary weight set. Complex and additional hardware circuitry is necessary to design weighted random pattern generator. Moreover, for complex circuits, multiple sets of weight are necessary to achieve acceptable fault coverage. This leads to a large volume of data to be stored and manipulated.

3.1.4.6 Mixed-mode approach

In order to have a better solution to the problems of low-fault coverage due to random-resistant faults, researchers have proposed the mixed-mode approach. They claimed that this approach ensures complete fault coverage while offering reduced storage requirements, shorter test application time and smaller area overhead compared to the weighted random approach. This approach exploits the advantages of both pseudo-random and deterministic test pattern generation techniques. Generally most of the faults in a circuit are easily testable. Experiments show that a certain maximum number of faults in an IC are easily detectable using a lower number of test patterns whereas a large number of test patterns are needed to detect the rest of the hard-to-detect faults. In mixed-mode testing approach, PRV is generated using LFSR and is applied to the CUT to detect all the random pattern testable faults and then deterministic test sets are generated using the same LFSR to target the random pattern resistant faults from compacted test data named as seed.

3.2. FPGA INTERCONNECT TESTING

FPGA interconnect testing is done for the following:

1. Stuck –at – Fault
2. Bridging Fault
3. Complete Test Set

1. Stuck –at Fault

A short between two wires can be both Wired-AND and Wired-OR. Also it is to be noted that a stuck– closed CIP will create a short between two wires and a stuck-open CIP will create an open between the wires. The wires are divided into several bunches, where a bunch is a group of wires that may have a pair-wise shorts, but not every wire is necessarily adjacent with every other wire in the bunch and wires in the different bunches are not adjacent [6][29]. For example the entire horizontal wires located between two adjacent PLB rows can be treated as a bunch, even though some of the shorts are not physically feasible.

This approach makes interconnect testing easy to handle and makes the method layout independent. In order to detect the above mentioned faults, the applied tests must check if every wire and CIP is able to pass both 1 and 0 correctly and that every pair of wire segments that may be shorted can pass all four combinations of 1 and 0 namely (1,0); (1,1); (0,1); (0,0).

This will make sure that both wired AND and wired OR are tested.

2. Bridging Faults

It is not appropriate to consider bridging faults between all possible pairs of nets in the design. First, the size of the fault list becomes intractable. Second, bridging faults between some pairs of nets are improbable due to physical neighborhood obtained from layout information. Inductive fault analysis (IFA) [4][14] techniques are proposed to extract the fault list from the physical layout information [Ferguson 88]. These techniques are very time consuming for a medium to large size designs. The presented technique supports both internally generated fault list for bridging fault or any user-specified fault lists. All the nets in the design are partitioned into some groups. All the nets in the same

group are tested for all possible pair-wise bridging faults. A group of m nets has m(m-1) bridging faults.

The bridging fault between two nets in different groups is not included in the fault list. In the internally generated fault list, all the inputs of a look-up table (LUT) are considered in the same group. Hence, the number of the groups is equal to the number of LUTs [6]used for the design. As a result, the size of the fault list is a constant factor of the number of logic blocks used in the design. Note that this fault list exploits physical neighbourhood, since the nets in different blocks have less probability of bridging faults than those in the same block.

3. Complete Test Set

In this section, a fault list is considered which is more comprehensive than stuck- at fault list. The fault list consists of stuck-at and all combinations of bridging faults for all the nets of the mapped design. Note that the number of all combinations of bridging faults for a design with n nets is equal to n(n-1)/2 faults. In conventional bus testing at board level, only [log2 (M+2)] test vectors are sufficient to test for all possible stuck-at, open and bridging faults for M bus lines [1][14][29].

The test vectors correspond to the Walsh code, which are the columns of binary representation of the number from 1 to M using [log2 (M+2)] bits. Fig 3.4 shows the test vectors for 6 wires. The number of required test vectors is [log2(6+2)]=3. Note that all -0 and all-1 patterns are not used,as test vectors; otherwise one stuck-at-1 and one stuck-at-0 fault cannot be detected. These test vectors are used in conjunction with single-term functions to generate test configurations for the presented fault list. As a result, adding this technique to single-term functions, all the bridging faults can be detected.

001_____

010_____

011_____

100_____

101_____

110_____

Fig. 3.4: Complete Test Set for 6 Wires

So, for a mapped design with N nets, we transform each of [log2 (N=2)] test vectors for all the nets in the mapped design into the activating inputs for LUTs. Based on the activating inputs for each LUT, the single-term logic function of the LUT is determined. Note in this approach [29], no ATPG algorithm is needed for test configuration generation. As a result, test configuration generation is very fast.

3.3 NEED FOR UNCONVENTIONAL METHODS

The frequently-quoted advantages of FPGA are reduced system cost, improved performance and greater reliability. These advantages, however, will be lost unless FPGA circuits are economically tested. An obvious reason for testing is to separate a good product from a faulty one. The dramatic increase in the ratio of number of internal devices to input-output terminal pins of FPGA drastically reduce the controllability and observability of the chip.

A digital system consists of many printed wiring boards, each containing several FPGAs. Testing such systems can be an overwhelming task and faster and more efficient test generation algorithms are needed. The variety of subtle failures that can occur in a FPGA chip further complicates the testing process. Complexity of test generation is enormous. Testing is the most significant part of FPGA development cycle. If this takes so much time, energy and cost, then upcoming demand for FPGAs will slow down because they will not be available in the market at the right time. Moreover, this will also increase the cot of FPGAs manifold. Conventionally, to find a fault in the circuit, ATE(Automatic Test Equipment) is used. If the chip has n inputs, then considering the chip as a black box, we will have to test it 2^n times to find the fault. So for million chips that are produced from a single wafer, we have to test

$$1 \text{ million} * 2^n \text{ times}$$

In real life, the number of inputs are also very large in number e.g. 50 or more. So the value of n is also very high. Using the conventional methods of test generation, the cost and time for test generation increase considerably as the complexity of the circuit increases. This is shown in fig.3.5 as in [27].

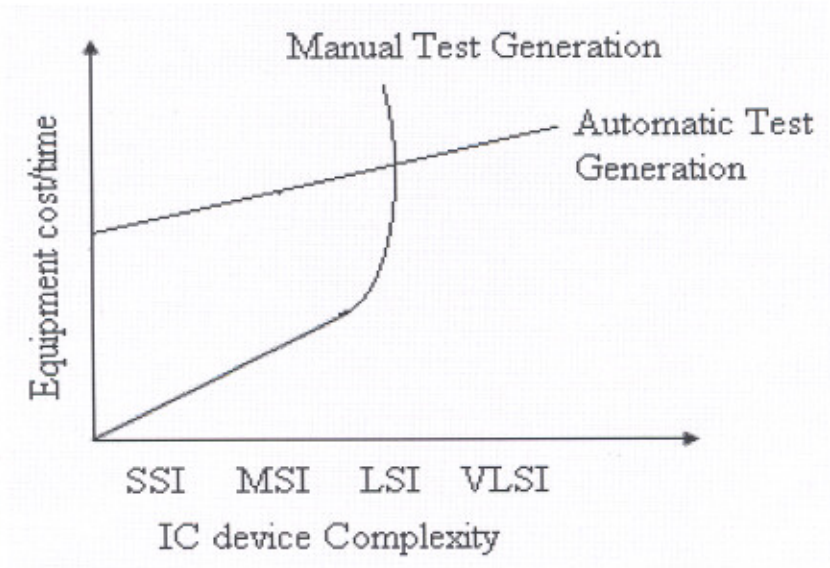

Fig. 3.5: Equipment Cost/Time Vs IC Device Complexity

Conventional techniques are not able to serve the current demand. This makes use of conventional methods out of trend. Hence, the birth of Unconventional methods.

3.4. UNCONVENTIONAL METHODS

The unconventional methods for FPGA test generation are those, which do not view the chip in the form of its components such as gates and transistors rather formulate the circuit in its own form. Most of the unconventional methods use the concept of ATPG Constraint Network. This has been explained below in detail.

3.4.1. ATPG

Automated Test Pattern Generation (ATPG) for digital circuits is a major research topic. As far as the single stuck-at fault model is considered, efficient algorithms have been devised for combinational networks. The economical importance of ATPG tools for digital circuits is continuously growing, and the demand for efficient algorithms and tools able to handle the current circuits is thus very strong. Correspondingly, there have been significant research efforts in this field ,which produced tens of proposals in terms of ATPG algorithms and techniques.

In the last years, one of the main goals of these efforts was to develop effective algorithms for sequencial circuits. Due to the great increase in the circuit size and complexity, this task is now critical from the point of view of the required computational power, and a significant amount of research activities has been devoted to it in the past years. The stuck-at-fault model has generally been adopted, and efficient algorithms have been proposed for combinational networks. On the opposite side, the problem of effectively dealing with very large sequencial circuits is still open, even if some commercial tools are already available. In fact, no proposed method is able to successfully handle the complete set of real-world circuit's test engineers have to face with: they either definitely get off or generate very poor results.

Very large sequential circuits still constitute a problem. None of the methods proposed can fully handle all the real-world circuits test engineers have to deal with, since they either definitely get off or generate sequences with very low fault coverage's. Three type of approaches have been suggested:

1. Topological Approach
2. Symbolic Approach
3. Simulation based Approach

3.4.1.1. Topological Approach

The traditional topological [24], is based on extending to sequential circuits the branch and bound techniques developed for combinational circuit by adopting Huffman's Iterative Array Model, and the method's effectiveness heavily relies on the heuristics adopted to guide the search. The approach uses a complete algorithm, and therefore it can be exploited to identify untestable and redundant faults. Thresholds are normally introduced to avoid the exploration of the whole search space, and redundancy identification often fails when applied to large circuits, where the search space is excessively large to be explored.

Random search and exhaustive search techniques are used in Topological approach. The most widely used ATPGs usually adopt topological techniques, which suffice for circuits of moderate size and complexity. When highly sequential circuits are considered, the backtracking [24] activity inherent in the search space exploration reduces the quality of

tests considerably because of the increased number of aborted. Most topological ATPG algorithms generate the test sequence for every fault by performing the following three phases:

1. **Fault Excitation:** This phase is performed at time frame t0 to find a state and a set of values to be applied to the circuit Primary Inputs that excite the fault. The state is typically partially specified.
2. **Fault Propagation:** Fault effects are propagated up to a Pimary Output, either in time frame t0 or in a succeeding time frame. Further assignments may be made to the state in time frame t0.
3. **Fault Justification:** A sequence of vectors that justify the state required in time frame t0 is obtained. If conflicts are found during any of the three phases, the test generator backtracks to a previous decision point and makes an alternative decision. State justification is performed by reverse time processing [24]. Time frames are processed in reverse order until states containing all don't care values is reached. If a desired state is determined to be unreachable, then backtracking is performed. If a state is not determined to be unreachable, then the topological algorithms for fault propagation or state justification are used.

3.4.1.2. Symbolic Approach

The recently explored symbolic approach exploits techniques, which were initially developed for formal verification, based on the extraction and manipulation of the Boolean functions implemented by the circuit. This approach is based on a complete algorithm, too, and is very effective when small- and medium-sized circuits are considered. Being an exact method, it can identify all un testable faults for circuits it is able to deal with. Unfortunately, it is completely inapplicable when dealing with circuits having more than some tens of Flip-Flops. This greatly limits its usefulness in practice.

Boolean satisfiability is the technique used in the symbolic approach. Symbolic Techniques can be used to reason about the behavior of a macro, regardless its Topology. In this case, the circuit is modeled as a Finite State Machine, and is represented by the Boolean state transition function computing the next state y from the current state s and the current input x, $y= f(s,x)$, and by the output function computing the output z starting

from the same information, z = f(s,x). Such functions are extracted from the circuit net list, and completely describe its input/output behavior.

The characteristic function of the state transition function y=f(s,x) is a compact representation of all the triples(s,x,y) satisfying function: the characteristic function $\chi^{f}_{(s,x,y)}$ takes the value 1 for all the triples (s',x',y') such that y'= f(s',x').Using such a representation, it is extremely easy to extract the information items that are needed during ATPG:

1. **Propagation:** given a set of input constraints and state constraints, compute the corresponding permitted output values. If the sets of allowed inputs and states are modeled by their characteristic functions, fχ(x) and f s(s),respectively, then the characteristic function of the output set of values is fy(y)= χ s [fχ(x),fs(s),χf(s,x,y)].
2. **Justification:** the same idea applies when trying to compute input constraint starting from output knowledge.
3. **State traversal:** knowing which states are accessible to a circuit is essential to avoid trying to justify an invalid state. These equations can be efficiently computed when all the functions are represented by their respective BDD.

These equations can be efficiently be computed when all the functions are represented by their respective BDD. Symbolic approaches alone, however, do not represent a general solution to the ATPG problems; reasonable size can be computed for small and medium circuits only. Symbolic techniques are applied to small macros. Thus, the symbolic techniques are used to provide an early backtrack indication only.

3.4.1.3. Simulation based Approach

The simulation based approach consists of generating pseudo random sequences; fault simulating them, and then modifying their characteristics to increase the obtained fault coverage.

DNA, Genetic Algorithm, Simulation Annealing and Quantum approaches can be used in Simulation based approach. The test generation process can be organized as two consecutive phases: fault excitation and fault propagation. To excite a Target Fault, TF, the ATPG has to drive an appropriate set of values on the circuit Primary Inputs (PIs) and on the circuit Pseudo Primary Inputs (PPIs), i.e., on the flip-flops outputs. Therefore, to

excite a target fault TF we need first to bring the circuit to a state where it is excited. We refer to this state as a fault excitation state. For a given fault TF, several fault excitation states may exist; hard-to-test faults typically have a small set of excitation states. On the other hand, in order to propagate a TF toward a Primary Output (PO), the ATPG has to compute a sequence of input patterns to be able to drive the circuit from the fault excitation state through several fault propagation states [24] which allow the fault to reach PO.

Although the notion of fault excitation/propagation state is important to improve the ATPG capabilities, it is not enough to attain high fault coverage. Simulation based approach is stopped when the four phases have been repeated for a given number of iterations. Given such assumptions, let us explain how the phases interact:

Phase (1) has the objective of maximizing the fault-free gate activities, by exploiting information coming from logic simulation; it starts from a random population of sequences, and outputs a population which maximizes the circuit gates activity.

Phase (2) computes sequences in which the fault free circuit reaches as many states as possible; it works on a population coming from phase (4), trying to add as many new states as possible to the set of already reached states.

Phase (3) relies on fault simulation only, since it performs fault dropping and selects the faults which will be addressed by the following phase; its input is a population of sequences coming either from phase (1) or phase (2), while its outputs are the set of the least active gates within the circuit and the population of sequences coming from the previous phase.

Phase (4) tries to propagate the previously selected faults toward the circuit outputs by means of both logic and fault simulation; its inputs are a population of vectors and a set of gates on which its attention is to be kept; its outputs are a set of test sequences and the initial population for the next phase.

3.4.2. Boolean Satisfiability Approach

The Boolean Satisfiability Approach for test generation was proposed by Tracy Larrbee in 1992. It is neither purely structural nor a purely algebraic approach [23]. It generates test patterns in two steps: First, extraction of the Boolean truth function for the circuit and

second, satisfying the function using Boolean satisfiability algorithm. This approach is applied to FPGA and is explored in detail in Chapter 4.

Chapter- 4

Boolean Satisfiability Approach

Existing algorithms for single stuck-at faults in combinational circuits fall into two classes: the structural methods, which perform a topological search of the circuit under test and the algebraic methods, which generate test patterns by manipulating algebraic formulas. The Boolean Satisfiability method [23] is a new algorithm for test pattern generation for single stuck-at faults in combinational circuits that is neither purely structural one nor an algebraic one. This method is not only practical but also performs better than most systems now in use. The Boolean Satisfiability Approach is flexible, effective and economical.

4.1. SATISFIABILITY FORMULATION

4.1.1 Configuration for the Entire FPGA

In the presented approach, all the nets in the design are partitioned into two groups, *plus* and *minus*, such that for every LUT L with n inputs to be partitioned, $n/2$ of the inputs are assigned into plus group, and the other $n/2$ inputs are assigned to minus group. This process is called *dichotomization*. The fanout branches and stem of the same net must be assigned to the same group. By assigning 0 (1) to all the nets in the plus group and 1 (0) to all the nets in the minus group, the desired activating inputs for all the LUTs will be obtained. Based on the logic value of the activating input and the output of the LUTs, the configuration of the LUT and the preset value of the flip-flops are determined. The logic values of the nets corresponding to primary inputs are the test vectors.

For the next test configuration, we set $n = n/2$, and the same dichotomizing algorithm is performed for each of the groups obtained in the previous configuration. This process recursively continues for all the $log_2 n$ test configurations.

4.1.2 SAT Formulation

The configuration generation problem is converted into a satisfiability problem by constructing a Boolean function F [14]. A solution to F=1 can be converted to the desired test vectors and configurations. Consider the symmetric logic function $S_k^n(x1, x2, ..., xn)$, where $k = 1$ if and only if exactly k inputs of S_k^n are 1. For example, $S_1^3 = x_1x_2'x_3' + x_1'x_2x_3' + x_1'x_2'x_3$. S_n^k can be recursively defined as follows (π is the logical AND function):

$$S_k^n(x_1,\ldots,x_n) = \begin{cases} \prod_{i=1}^{n} x_i & ,k = n \\ \prod_{i=1}^{n} \overline{x_i} & ,k = 0 \\ x_1 . S_{k-1}^{n-1}(x_2,\ldots,x_n) + \overline{x_1} . S_k^{n-1}(x_2,\ldots,x_n) & ,0 < k < n \end{cases}$$

Consider the set of the nets in the design that must be dichotomized. Let M be the number of LUT inputs that participate in this grouping algorithm. $M = N/2^{i-1}$ at i^{th} configuration for N-input LUTs. We assign one variable per each net. All the fanout stem and branches of the same net share the same variable. For each LUT l, let x_1^l, x_2^l ,..., x_M^l be the M inputs of l which have to be dichotomized.

Let $F_l = S^M_{M/2}(x_1^l, x_2^l, x_3^l, ..., x_M^l)$.

The solution for $F_l = 1$ is an equal-size dichotomization of those M inputs of l, a sub-solution to the problem for only one LUT. The function that is the SAT formulization for the entire FPGA is:

$F = \pi F_l$ for each LUT l.

In the variable assignment that satisfies F, every variable whose with logic value 0 is put in the minus group, otherwise is assigned to the plus group. All the nets are assigned with unique variables. For the first configuration, S_2^4 must be used for all LUTs, since $M = 4$. The SAT function for the first configuration is as follows:

$F = S_2^4(x_1,x_2,x_3,x_4) . S_2^4(x_5,x_6,x_7,x_8) . S_2^4(x_7,x_9,x_4,x_{10}) . S_2^4(x_4,x_9,x_{10},x_8)$ where:

$S_2^4(x_1,x_2,x_3,x_4)= [(x_1'+x_2'+X_3')(x_1'+x_2'+x_4')(x_1'+x_3'+x_4')(x_2'+x_3'+x_4')$
$(x_1+x_2+x_3)(x_1+x_2+x_4)(x_1+x_3+x_4)(x_2+x_3+x_4)]$

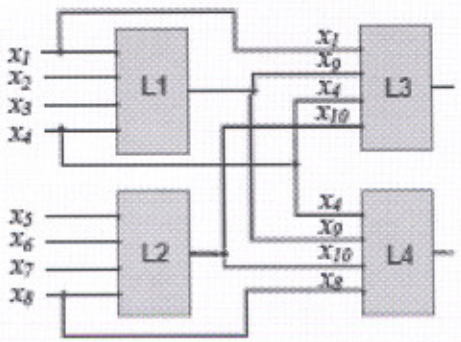

Fig. 4.1: An example design implemented with four LUTs

One variable assignment that satisfies *F* is:

$<x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}>=<0,0,1,1, 0,1,1,0,0,1>$.

It implies that plus group = $\{x_3,x_4,x_6,x_7,x_{10}\}$ and minus group = $\{x_1,x_2,x_5,x_8,x_9\}$. For the second configuration, each of the above groups must be partitioned. Hence, there are two SAT functions on two disjoint sets of different variables. These functions are as follows:

$F_1= S_1^2(x_1,x_2).S_1^2(x_5,x_8).S_1^2(x_1,x_9).S_1^2(x_8,x_9)$

$F_2= S_1^2(x_3,x_4).S_1^2(x_6,x_7).S_1^2(x_4,x_{10}).S_1^2(x_4,x_7)$

Where $S_1^2(x_1,x_2) = (x_1'+x_2')(x_1+x_2)$

One variable assignment that satisfies *F1* is: $<x_1,x_2,x_5,x_8,x_9> =<1,0,0,1,0>$. Therefore the groups are $\{x_2,x_5,x_9\}$ and $\{x_1,x_8\}$. Similarly, a variable assignment that satisfies *F2* is: $<x_3,x_4,x_6,x_7,x_{10}> =<0,1,0,1,0>$ and the groups are $\{x_3,x_6,x_{10}\}$ and $\{x_4, x_7\}$. The results interpreted in terms of the activating inputs and LUT configurations for two test configurations are shown in Fig. 4.2.

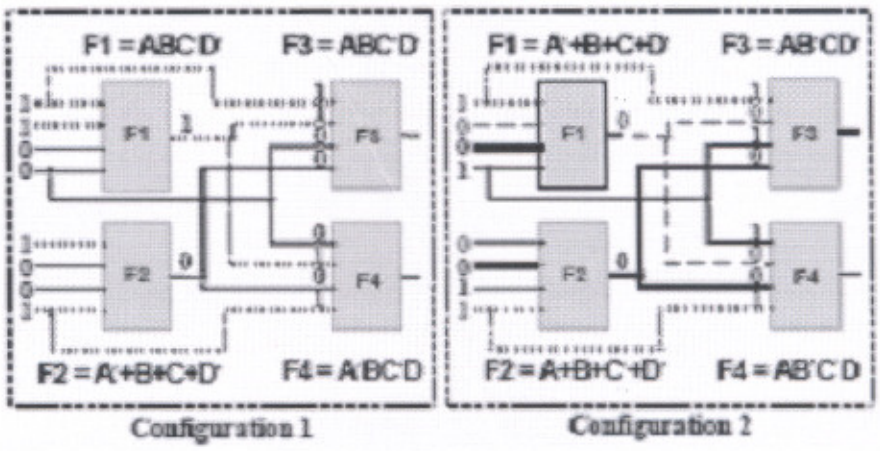

Fig. 4.2: Two configuration for 4-input LUTs

4.2 Conventional SAT Solvers

There are number of SAT solver in the market that are used to solve the Boolean SAT equation and among them that are most common are GRASP, Chaff etc. Lets discuss these solvers one by one.

4.2.1 Grasp

GRASP [8] is an integrated algorithmic framework for SAT that unifies several search pruning techniques and facilitates identification of additional one. GRASP is premised on the inevitability of the conflicts during search and its most distinguishing feature is the augmentation of basic backtracking search with powerful conflict analysis procedure. Analyzing conflicts to determine their causes enables GRASP to backtrack non-chronologically to the earlier levels in the search tree, potentially pruning large portion of the search space. In addition to recording the causes of conflicts, GRASP can recognize and preempt the occurrence of the similar conflicts later in the search. Finally, straightforward bookkeeping of the causality chains leading up to conflicts allows GRASP to identify assignment that are necessary for a solution to be found.

Conflicts analysis in GRASP is tightly coupled with BCP and the causes of conflicts need not necessarily correspond to decision assignment. Also clause can be added to the original set of clauses, and the number and the size of added clauses is user-

controlled. GRASP employs techniques to prune the search by analyzing the implication structure generated by BCP.

4.2.2 Chaff

The other efficient SAT solver is Chaff, which achieves significant performance, gains through careful engineering of all aspects of the search-especially a particularly efficient implementation of Boolean constraint propagation(BCP) and a novel low overhead decision strategy. Chaff [2] employs a conflict resolution scheme that is philosophically very similar to GRASP, employing the same type of conflicts analysis, conflicts clauses addition. There are some differences that have dramatically enhanced the simplicity and elegance of the Chaff. First is the clause deletion, like other solver Chaff supports the deletion of added conflict clauses to avoid a memory explosion. Chaff also employs a feature referred to as restarts. Restarts in general consist of a halt in the solution process, and a restart of the analysis included in the new one. There are two versions available of Chaff: mChaff and ZChaff. Both are instances of Chaff, but from the viewpoint of implementation they are totally different since they were developed independently by M.~Moskewicz (mChaff) and L.Zhang (ZChaff).

Chapter-5

Problem Undertaken

As a part of my thesis I have solved FPGA test configuration and generation task by transforming it into a Boolean satisfiability equation with the property that any assignment of input variables that satisfies the equation specifies the fault. Satisfying assignment for particular test will result in a fault and the absence of a satisfying assignment implies that the layout does not have fault. After this is done, DNA computing algorithm is applied on this Boolean equation for solving the test configuration alternatives utilizing the properties of DNA computation.

To make the clear picture let us take a brief look at what is Boolean satisfiability (SAT). SAT is the problem of finding a solution (if one exists) to the equation f = 1, where f is a Boolean function to be satisfied. The formula (f) can be represented in CNF (Conjunctive Normal Form). An instance of the problem is defined by a Boolean expression written using an AND, OR, NOT, variables and parentheses. The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true. There is no known algorithm that is faster than the exponential one. Each Boolean variable can be assigned either 'true' or 'false'. If we have 1 variable we have only two possibilities. When we have 20 variables the number of possible assignments grows exponentially to $2^{20} = 1048576$. One can check whether the first variable is 'true' or 'false' and in each case an appropriate action is done.

In classical approach the computer checks every possible combination one by one. The idea behind DNA Computing is different. It works in parallel. It does not try every single possibility but tries criteria one by one, eliminating all false solutions that does not satisfy the criteria. It starts by the first criterion, deletes all solutions which do not suit the condition, then checks solutions whether or not they satisfy the next criterion and if they do not ,it deletes them as well as so on until all criteria are checked. It is so powerful because every "if" acts on several variables at the same time.

We can imagine every variable in the problem as a node in the tree and two values can take (either 'true' or 'false') as branches. The classical computation would rely on searching the tree using certain algorithm (like depth-first search). In contrast DNA computation will depend on checking few nodes and cutting out the appropriate branches, not checking nodes one by one. At the end of the procedure , only the solution(s) will survive. The last thing one needs to do is to read off the solution.

FPGA test configuration problem can be solved by transforming it into Boolean equation. By representing the test configuration problem as Boolean function one can also prove that particular testing alternatives does not exist or the net list is not testifiable. Modern SAT solvers are enriched with clause-learning and backtracking techniques to help prune the solution space. There are two classes of high-performance algorithm for solving instances of SAT in practice: modern variants of the David-Putnam-Loveland algorithm, such as GRASP ,Zchaff, and stochastic local search algorithms such as WalkSAT. I have used DNA based algorithm to solve the problem of FPGA Test Configuration because it is based on the principles of DNA Computing and DNA Computing offers the remarkable concept – parallel computation. Using this concepts we can develop DNA based algorithm that can find testing solutions more quickly and effectively than is possible on a classical computer.

Chapter 6

Proposed Method

FPGA Test generation is a highly complex process. This is because FPGA circuit consists of millions of components and interconnects. If the process of test generation is parallelized, the complexity of test generation will reduce manifold. The proposed method is an unconventional method of FPGA test generation. This method can reduce the complexity of test generation process to a great extent, making it solvable in linear time.

The test pattern is generated for bridging faults of FPGA in two steps:

1. Extract the formula that defines the set of test patterns that detect the fault.
2. Apply dna based algorithm to satisfy the Boolean formula.

The proposed method has been explained in detail as follows:

The first step consists of the extraction of formula for the FPGA test configurations. In this, the circuit has to be labeled with a formula, which must be converted in a conjunctive normal form (CNF).

The second step consists of satisfying the Boolean truth function using the algorithm explained later.

The proposed model includes various biological notations developed to solve the problem, the problem formulation framework and the respective algorithm developed to solve the problem. The operations in the suggested algorithm are considered to be error free and the ideal conditions are assumed to be there though it may not be possible in practice. But these limitations may certainly be overcome by introducing certain amount of replication in the information manipulated. The advantage of the proposed method is that the output values are computed and stored parallely. The presented approach makes use of the Lipton's Encoding scheme for the solution that has the obvious advantage that the DNA strands used in the process are reusable. The strands which represent the output can be used repeatedly any number of times.

6.1. NOTATIONS FOR THE BASIC OPERATIONS OF PROPOSED METHOD

1. Append (A,σ) : For each single DNA strand in the test tube A, append pattern σ.
2. B ← Extract (A,σ) : Extract from the test tube A all the single DNA strands having pattern σ and put the extracted strands into another test tube B.
3. Detect (A) : Test whether A contains at least a single DNA strand. The result is either true or false.
4. Polymer (A) : It implies polymerization. Create a complementary strand for each DNA strand A.

   In addition to the above , we will use the following:

   B ← Mix (A_1, A_2, ...A_m) : Mix test tubes A_1, A_2,.....A_m and name the mixture as B.

6.2. PROPOSED APPROACH

Let n and m denote the number of variables and clauses respectively. For each of the n variable, two distinct value sequences of DNA (A,T,C,G) is designed- one representing true (T), $_{i=1}{}^{n}\, t_i$ and one representing false(F), $^{n}_{i=1} f_i$, satisfying the following conditions:

1. All of these value sequences have the same length (say t bases).
2. For every i, t_i and f_i are complementary to each other.

Each of the 2^n truth assignments are represented by a library sequence consisting of the concatenation of one value sequence of each variable. DNA molecules with library sequences are termed library strands and a combinational pool containing library strands is termed a library.

To represent all possible variable assignments for the n variable SAT problem, Lipton encoding [15] is used.

The Boolean formula derived can be encoded by a graph G_n.

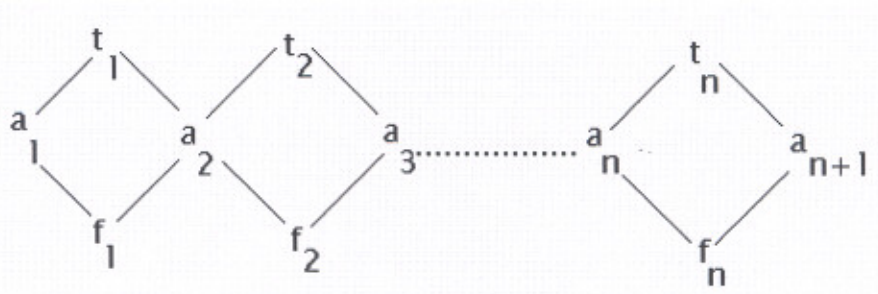

Fig. 6.1: The graph Gn which encodes n-bit numbers (binary)

Notation: t=1 ⟹ True

f=0 ⟹ False

Vertex is denoted by a_i and Edge by $E_{ai\ ti}$, $E_{ai\ fi}$, $E_{ti\ ai+1}$, $E_{fi\ ai+1}$

For example $a_1t_1\ a_2t_2\ a_3$ denotes binary number 10.

6.2.1. Procedure

Step 1: Design and synthesis of the library strands

1. Create DNA strands to encode vertices and edges

Design 2^n DNA Strands, each with n bit regions to encode vertices and edges. Since the result tubes are to be constructed for a CNF formula with n input variables, so we start with 2^n identical memory strands each with n bit regions. The n bit regions serve as the input. The strands taken are initialized so that each strand represent a different possible combination of states of variables in a clause C of CNF formula. This can be done by pouring together many copies of 5'→3' DNA sequence p_iq_i (representing each vertex) and 3'→5' DNA sequences of the form $\hat{p}_i$ $\hat{q}_i$ (representing each edge where $\hat{X}$ denotes Watson-Crick Complement of X). Besides this again add 3'→5' sequences ½ for a_1 uptill a_{n+1}.The strands randomly anneal with each other making use of Watson Crick complementarily of DNA. Care should be taken to see that this random annealing process obtains all possible combinations of n input variables. This random annealing of strands help to form the initial tube, which contains the strands representing all possible combinations.

The above step is depicted pictorially as shown below:

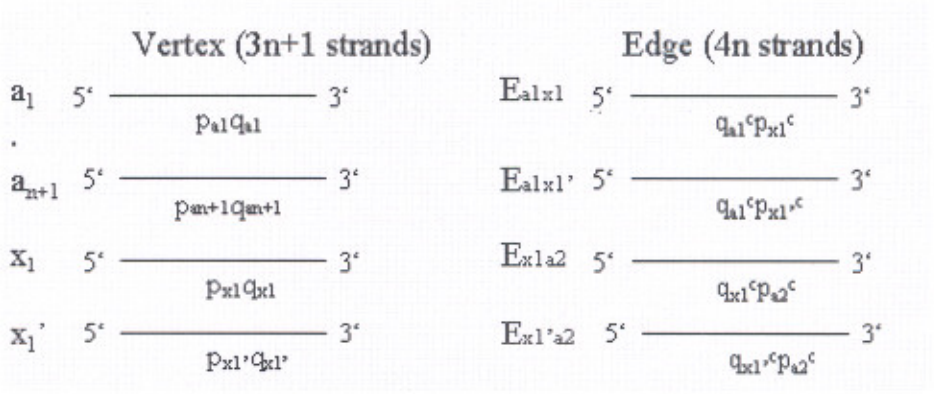

Fig. 6.2: DNA strands to encode vertices and edges

Total number of Vertex = 3n+1 strands

Total number of edges= 4n strands

2. Hybridization

A path V-E-V-E-V.....V ➔ denote an n bit binary number

Example: a path $a_1x_1a_2x_2'a_3$ denote a 2 bit binary number,10

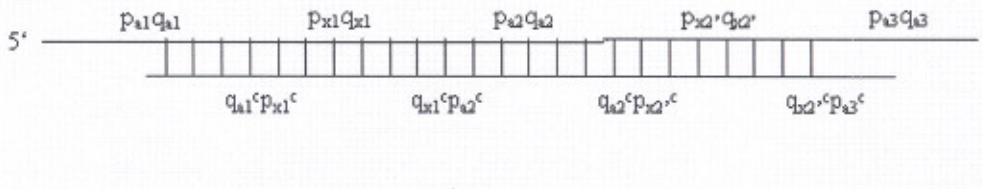

Fig. 6.3: Hybridization

Step 2: Extraction of the required library strands

The extraction of required library strands is done by the elimination series of the strands. The extraction proceeds in repeated fashion for each clause of CNF formula. In this Step , the strands prepared in previous step are taken in tube T. For every un-complemented variable t_i in a clause C, the strands present in the tube T

having bit as *0* are retained in T and rest are separated in another tube each time. Then the strands of resulting tubes are combined into T_t. This step is repeated for every uncomplemented variable of clause *C*. In the Step 3, again the strands in tube T are taken and for every complemented variable f_i in a clause *C*, the strands present in this tube having bit as *1* are retained in T and rest are separated in the other tube each time. Then the strands of resulting tubes are combined into T_f. This step is repeated for every complemented variable of clause *C*. The Step 2 and Step 3 are repeated for every clause *C* of CNF formula.

In the Step 4, again the resulting tubes T_t and T_f are combined into tube T. Now the strands in T represent those combinations of input variables which satisfies every clause of the CNF formula and thus in turn satisfying the whole CNF formula. The extraction process is depicted as an illustration as shown below:

Table 6.1: Extraction Process

Test Tube	Operation	Value Present
t0	Create DNA Strands	00,01,10,11
t1	E (t0,1,1)	10,11
~t1	Remainder of t1	00,01
t2	E (~t1,2,1)	01
t3	Merge t1 U t2	10,11,01 -> T
t4(to remove 11)	E (t3,1,0)	01
~t4	Remainder of t4	10,11
t5	E (~t4,2,0)	10
t6	Merge t4 U t5	01,10

6.3. PROPOSED ALGORITHM

Step 1: Design 2^n DNA strands each with n bit regions and initialize it into the tube T

For variable i = 1 to n do

A← extract (T, ti);

A'← extract (T,fi);

T← mix (A,A');

Step 2: For each clause C of the form(t_1 V t_2 V.....t_k V f_1 V f_2 V.....f_k)

T_{1t}← extract (T,t_1);

T_{2t} ← extract (T_{1t}',t_2);

.....

.....

T_{kt} ← extract (T_{kt-1}',t_k);

T_t ← mix (T_{1t},T_{2t},.....T_{kt});

Step 3: For each clause C of the form(t_1 V t_2 V.....t_k V f_1 V f_2 V.....f_k)

T_{1f} ← extract (T ,f_1);

T_{2f} ← extract (T_{1f}',f_2);

.....

.....

T_{kf} ← extract (t_{kf-1}', f_k);

T_f ← mix (T_{1f},T_{2f},.....T_{kf});

Step 4: T ← mix (T_t , T_f);

Step 5: Execute detect (T)

If the result is positive, the formula is satisfiable , that is if there is any DNA strand present in T then the FPGA circuit to be tested is faulty. The DNA sequence of strand(s) present in the test tube will give the required test vectors.

If T is empty, then no assignment to the variables of the CNF formula exists. It implies that the circuit to be tested is fault free.

6.4. EXPERIMENTAL RESULTS

Table 6.2: Comparison among the number of iterations required by the Exhaustive Search and DSTG

n	m	ES	DATPG
8	13	256	138
10	17	1024	212
12	20	4096	290
14	25	16384	380
16	28	65536	460
18	33	262144	668
20	37	1048576	782
22	40	4194304	970
24	43	16777216	1130
26	45	67108864	1276
28	50	268435456	1514

n –input variables (2^n search space size), m – no. of clauses,

DSTG - DNA** based Boolean Satisfiable FPGA Test Generation **Algorithm

The number of necessary biochemical operations is n-1 for connecting patterns, 3n for extracting strands that represent assignments , n.m for Step 2 of algorithm, again n.m for Step 3 of the algorithm, 1 for Step 3 and finally 1 for final Step 4. Thus, we conclude that our algorithm runs in time n.m + 4n + 2.

A theoretical comparison has been made between Exhaustive Search and DSTG as shown in Table 6.2. In exhaustive search, we have to go through each possible combination to find the test vector, hence search space size comes out to be $O(2^n)$. On the other hand, it

is clear from the algorithm that the total complexity of our new algorithm comes out to be $O\ (n.m + 4n + 2)$, for some boolean formula with n variables and m clauses, which is too much improved over classical methods used for test case generation

Chapter- 7

Conclusion and Future Scope

In this thesis we have proposed the DNA version of Boolean Satisfiability Approach for FPGA Test Generation. This chapter summarizes the main conclusions and suggests the future scope of work in the field of DNA Computation.

7.1 CONCLUSION

In this thesis a new concept of DNA Computing to solve the Boolean Satisfiability (SAT) based detailed FPGA testing problem is illustrated. The Boolean Satisfiability approach is one of the unconventional methods employed for FPGA test generation. Since FPGA test generation is a highly complex process, there is an intense need of unconventional methods for FPGA test generation that help to reduce the complexity, cost and time of test generation manifold.

The DNA based Boolean Satisfiable approach for FPGA Test Generation (DSTG) has been proposed. The effectiveness of the approach has been demonstrated by the theoretical comparison of DSTG results to exhaustive search.. Results shows that the developed approach DSTG is taking n.m + 4n + 2 iterations to solve SAT instances and shows that the proposed algorithm is able to find out the required result more quickly than is possible on a classical computer. The complexity of this algorithm in terms of number of iterations is quite improved over other approaches used for testing FPGAs. This improved efficiency is achieved due to the ability of DNA computing to evaluate all possible combinations simultaneously. This massive parallelism and recombination properties possessed by DNA molecules are responsible for solving a problem, like FPGA test case generation, with exponential complexity, in a polynomial time [$O\ (n.m\ +4n\ +2)$].

7.2 FUTURE SCOPE

In the field of FPGA, there are many problems that can be solved using DNA Computing. DNA based search algorithms can be applied in modified form in the problems like circuit-partitioning, placement and its application in test generation can be enhanced. DNA Computing can be applied to those problems that can be converted into a search based problems. As the field of DNA Computing is being explored, the application of this field in the FPGA can be enhanced. Many FPGA layout problems can also be eliminated using the benefits of DNA Computing.

DNA computing is just over three years old, and for this reason, it is too early for either great optimism or great pessimism about the technology. DNA computers will become more common for solving very complex problems; Just as DNA cloning and sequencing were once manual tasks, DNA computers will now become automated. The first model, of small scale, could restore memory and calculate twice as fast as the worlds leading super-computer, for a very economical price. Thus there are endless possibilities of the use of DNA computing for big business, government and many other types of organizations. Once the bulk of its possibilities have been studied and learned the dawn of DNA based computers would cause the super-computers of tomorrow to be able to handle far more tasks and information than the computers of today.

References

[1] A. Krasniewski, "Application-dependent testing of FPGA delay faults," in EUROMICRO'99,1999,pp. 204-209,1996.

[2] CHAFF: Engineering an Efficient SAT Solver, *In Proc. Design Automation.*, June 2001, pp. 530-535.

[3] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of FPGA interconnect. In International Test Conference, pages 404-411, October 1998.

[4] D. Das, N.A. Touba, A Low Cost Approach for Detecting, Locating, and Avoiding Interconnects Faults in FPGA-Based Reconfigurable Systems, Proc. Int'l Conf. On VLSI Design, 1999.

[5] H. Michinishi, T. Yokohira, T.Okamato, T.Inoue, and H.Fujiwara. A test methodology for interconnect structures of LUT-based FPGAs. Proceedings of the Fifth Asian Test Symposium, 1996. Page(s): 68-74

[6] Huang, W.K., X.T. Chen, and F. Lombardi, "On the Diagnosis of Programmable Interconnect Systems: Theory and Application," IEEE FPGA Test Symp., pp. 204-209, 1996.

[7] I.G. Harris, and R. Tessier. Interconnect testing in cluster-based FPGA architectures. Proceedings Design automation Conference, 2000. Page(s): 49-54

[8] J.P.M. Silva and K.A. Sakallah. GRASP: A New Search Algorithm for Satisfiability. *In Proc. ACM/IEEE ICCAD.*, Nov. 1997.pp 156-168

[9] J.P. Hayes, " Modeling Faults in Digital Logic Circuits", Rational Fault analysis, R. Saeks and S.R. Liberty, eds, Marcel Dekker, New York, 1977, pp. 78-95.

[10] Leonard M. Adleman, "Computing with DNA", Scientific American, August 1998.

[11] Leonard M. Adleman. "Molecular Computation of Solutions to Combinatorial Problems". Science, 266:1021-1024, November 1994.

[12] Martyn Amos, Gheorghe Paun, Grzezorg, Rozenberg and Arto Salomaa, "Topics in the theory of DNA computing," Theoretical computer science, 287, 2000, 3-38.

[13] Miron Abramovici, Melvin A. Breur, Arthur D. Friedman, "Digital Systems Testing and Testable Design", *Jaico Publishing House*, Mumbai, India, 1997

[14] M.B. Tahoori, Application-Dependent Testing of FPGA Interconnects, CRC Technical Report 02-4, Stanford University, 2002.

[15] Richard Lipton. "Speeding Up Computation via Molecular Biology", November 1994.

[16] L. M. Adleman, 1996. "On Constructing a Molecular Computer". 1st IMACS workshop on DNA based computers. Princeton. In DIMACS series. vol.27(1996) . 1-21.

[17] M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. Testing the interconnect of RAM-based FPGAs. IEEE Design & Test of Computers, 15(1):45-50, January-March 1998.

[18] Richard Lipton "DNA Solution of Hard Computational Problems". Science, 268:542-545, April 1995.

[19] Richard Lipton. "Using DNA to solve NP-complete problems". Technical report, Princeton University, USA, January 1995.

[20] S.M. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, Boston, MA, 1994.

[21] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories", Ph.D Dissertation, University of Toronto, 1997.

[22] S.J. Wang, and C.N. Huang. "Testing and diagnosis of interconnect structures in FPGAs. Proceedings of the Seventh Asian Test Symposium, 1998. ATS '98. Page(s): 283-287.

[23] T. Larrabee, 1992 "Test pattern generation using Boolean Satisfiability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 4-15.

[24] T. Kirkland, M.R. Mercer. A Topological Search Algorithm for ATPG. Proc. ACM/IEEE 24th Design Automation Conf.,p. 502-508. June, 1987

[25] Lila Kari. "DNA computing: the arrival of biological mathematics". http://www.csd.uwo.ca/ lila/amsn.ps,1995

[26] M. Ogihara, A. Ray. "Simulating Boolean Circuits on DNA Computer". Technical Report 631, University of Rochester, August 1996.

[27] Vivek Jhamb, "Test Generation for VLSI Circuits", *BITS Research Project*, 1992

[28] Xilinx, Inc., " The Programmable Gate Array Data Book", 1992

[29] X. Sun, J. Xu, B. Chan, P. Trouborst. Novel technique for built-in self-test of FPGA interconnects. Proceedings of the International Test Conference, 2000. Page(s): 795-803, 1998.

[30] J. Gray, T. Frutos, A. Berman, A. Condon, M. Lagally, L. Smith, and R. Corn. "Reducing Errors in DNA Computing by Appropriate Word Design". Draft, 1996.

Papers Accepted/Communicated

1. Shikha Singh, Aavriti Goyal. "**Biological Approach for Generating Cipher-Text Using DNA Computing**". National Conference of Bioinformatics Computing, held at T.I.E.T, Patiala, on 18th March -19th March. **(Published)**

2. Amardeep Singh, Shikha Singh. "**DNA based Boolean Satisfiability Approach for FPGA Test Generation**". Neural Information Processing Systems Conference 2005 (NIPS 2005). **(Communicated)**