Parallelized Boolean Satisfiability Approach for VLSI Test Generation

A Thesis

Submitted in partial fulfillment of the requirement for Award of degree of

Master of Engineering

in

Software Engineering

Under the guidance of

Ms. Seema Bawa

Assistant Professor & Head

By

Meet Kamal Kaur

(Roll No 8023109)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

(Deemed University) Patiala-1447004

May 2004

Candidate's Declaration

I hereby certify that the thesis entitled "**Parallelized Boolean Satisfiability Approach for VLSI Test Generation**" submitted by me in the partial fulfillment of the requirement for the award of degree of Master of Engineering in Software Engineering at Thapar Institute of Engineering & Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Ms. Seema Bawa.

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

Meet Kamal

(Meet Kamal Kaur)

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

Seema Bawa 15/05/2004

(Seema Bawa)

Supervisor
Assistant Professor & Head
Computer Science & Engineering Department
Thapar Institute of Engg. & Technology
Patiala

Countersigned by

Seema Bawa 15/05/2004

(Seema Bawa)

Supervisor
Assistant Professor & Head
Computer Science & Engg. Department
Thapar Institute of Engg. & Technology
PATIALA - 147004

(D. S. Bawa)

Dean (Academic Affairs)
Thapar Institute of Engg. & Technology
PATIALA - 147004

Acknowledgement

I wish to express my deep gratitude to Ms. Seema Bawa, Head, Computer Science & Engineering Department, for providing her uncanny guidance and support throughout the course of this thesis work. I am also thankful to her for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank all the faculty and staff members and my class mates who were always there at the need of the hour to provided all the help and facilities, which I required, for the completion of my thesis.

At last but not the least I would thank the **Almighty God** for not letting me down at the time of crises and showing me the silver lining in the dark clouds.

Meet Kamal

(Meet Kamal Kaur)
Roll no 8023109

Abstract

VLSI Test Generation involves generating input patterns that give different response for a given fault in a faulty and fault-free circuit. The problem for test generation for combinational circuits is NP- Complete. 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 VLSI circuits is rapidly increasing. The need for effective and economical testing has given birth to *unconventional* methods for VLSI test generation. These include methods such as Boolean Satisfiability approach, Neural computing and Quantum approach 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 Boolean Satisfiability approach and proposed a parallel approach for VLSI test generation based on this approach. The Parallelized Boolean Satisfiability Approach for VLSI Test Generation is highly *efficient*, *scalable* and *economical* algorithm for the generation of test patterns for single-stuck-at faults in

Table of Contents

combinational circuits. If both the steps of Boolean Satisfiability approach are effectively parallelized, VLSI test generation will be much faster, effective and economic process. The parallel algorithm for both the steps has been proposed. The first step has been parallelized by concurrent computation of the CNF formulas for the components of the circuit to generate the Boolean truth function. It is proposed that the second step can be parallelized by generating the 2SAT solutions of the Boolean truth function using different orders of variables concurrently at the first level and checking the consistency with 3CNF clauses at the second level.

The effectiveness of the approach has been demonstrated by implementing it in a distributed environment. The implementation of first step i.e. extraction of the Boolean truth function has been done using PVM message-passing architecture with master-slave approach. The experimental results show the efficiency and good performance of the algorithm.

List of Figures

List of Tables and Graphs

Chapter 1

Introduction

Rapid advances in the integrated circuit technology have made it possible to fabricate digital circuits with a very large number of devices on a single chip. Very Large Scale Integration i.e. VLSI is the fabrication of millions of components and interconnections at once by a common set of manufacturing steps. The testing process detects the physical defects produced during the fabrication of the VLSI chip. Such a chip is tested by the sequence of input stimuli, known as **test vectors**, which check for the possible defects in the chip by producing observable faulty responses at primary outputs.

1.1. 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.

For a given logic circuit, a test pattern generator produces a set of input patterns that cause different responses from the faulty and fault-free circuits. Faults occur in VLSI circuits throughout their life. The different causes of faults in VLSI circuits are:

- Design Errors
- Material Defects
- Extremes in operational environment
- Deterioration due to length of operation
- Aging

Automatic test equipment applies the test vectors to input pins of the VLSI chip and compares the output pin responses with the expected responses. A normal requirement of these tests is that they detect a very high fraction of modeled faults. The detected fraction of faults is called *fault coverage*, the term commonly used to denote test quality. . Also, since the same set of test vectors will be applied to a large number of copies of chips, short test sequences are desirable.

The frequently quoted advantages of VLSI are reduced system cost, improved performance and greater reliability. These advantages, however, will be lost unless VLSI chips are economically tested. An obvious reason for testing is to separate a god product from a faulty one. The dramatic increase in the ratio of number of internal devices to input-output terminal pins of VLSI chips drastically reduces the *controllability* and *observability* of the circuit [3]. Controllability refers to the ease of producing a specific

internal signal value by applying signals to the circuit input terminals. Observability refers to the ease with which the state of internal signals can be deduced from the signals at the circuit output terminals.

A digital system consists of many printed wiring boards, each containing several VLSI chips. 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 VLSI chip further complicates the testing process. Complexity of test generation is enormous.

1.1.1. Basic Issues of Test Generation

Test Generation is a complex problem with many interacting aspects. The most important are as given in [1]:

- The cost of Test Generation
- The quality of generated test
- 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.

RTG generally works without taking into account the function or structure of the circuit to be tested. In contrast, deterministic test generation produces tests by processing the model of the circuit. Compared to RTG, deterministic test generation is more expensive, but it produces shorter and high-quality tests. Deterministic test generation can be manual or automatic.

Deterministic test generation can be fault-oriented or fault-independent. In a fault-oriented process, tests are generated for specified faults of fault-universe. Fault-independent test generation works with out targeting individual faults.

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.

1.1.2. 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. The large number and complex nature of physical failures that a practical approach to testing should avoid working directly with physical 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 [1]. 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 [30].

There are various kinds of fault models:

- Stuck-at Fault Model
- Delay Fault Model

Stuck-at Model can be further classified as:

- Single Stuck-at Fault model
- Multiple Stuck-at Fault model

Delay model can be further classified as:

- Path Delay Fault Model
- Gate Delay Fault Model
- Transition Delay Fault Model
- Segment Delay Fault Model

- Function Delay Fault Model

The single stuck-at fault model is the most commonly used fault model. The techniques discussed here are based on Single Stuck-at Fault model.

1.1.2.1. The Single Stuck-At (s-s-a) Fault Model

In the classical gate-level *stuck-at* fault model, the effects of physical failures are assumed to cause the inputs and outputs of logic gates to be permanently "stuck" at logic 0 or 1 (commonly called s-a-0 or s-a-1 faults). Much of the work in testing has used this model. The single stuck-at fault (SSF) model is also referred to as the classical or *standard* fault model because it has been the first and the most widely studied and used [1]. Although its validity is not universal, its usefulness results from the following attributes:

- It represents many different physical faults.
- It is independent of technology, as the concept of signal line being stuck at a logic value can be applied to any structural model.
- Experience has shown that tests that detect SSFs detect many nonclassical faults as well.
- Compared to other fault models, the number of SSFs in a circuit is small. Moreover, the number of faults to be explicitly analyzed can be reduced by fault-collapsing techniques.
- SSFs can be used to model other types of faults.

It is easy to see that a circuit with n signals can have as many as $3^n - 1$ possible faults since each signal can be s-a-0, s-a-1 or fault-free [3]. This is large number. For simplicity, it is assumed that only one stuck-at fault will be present at one time. This simplifying **single-fault assumption** is justified by the *frequent testing strategy*, which states that we should test a system often enough so that the probability of more than one fault developing between two consecutive testing experiments is sufficiently small. Thus if maintenance intervals for a working system are too long, we are likely to encounter multiple faults. There are situations, however, in which frequent testing is not sufficient to avoid the occurrence of multiple faults. But even when multiple faults are present, the test derived under the single-fault assumption are usually applicable for detection of multiple faults because, *in most cases, a multiple fault can be detected by the tests designed for individual single faults that compose the multiple one* [1].

With this single fault assumption, a circuit with n signals can have at most 2n faults. Further reduction in the number of faults occurs through *fault collapsing* when *equivalent classes* of faults are recognized. Two faults are said to be equivalent if they are detected by exactly the same test vectors. The single-stuck-at fault might seem artificial on the surface, but it is quite useful. Any set of test vectors that we use for detecting large percentage of single stuck-at faults, in practice, also detects a high percentage of all failures.

1.1.3. Problem Definition

The problem of test generation for a stuck-at fault in a combinational circuit with n primary input signals can be formulated as a search in the n-dimensional 0-1 state space

of primary input vectors [4]. Consider the combinational circuit of figure 1.1. The primary input signals are labeled $x_1,\ldots,x_n$. The primary output signals are labeled $f_1,\ldots,fm$. Primary input and output signals are Boolean variables.

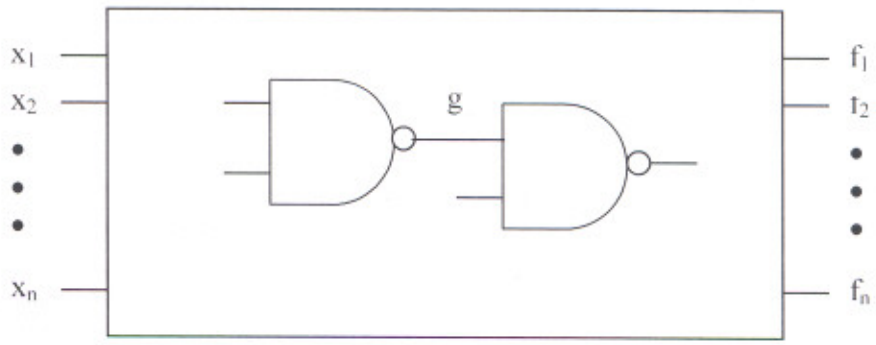

Fig 1.1. A generic combinational circuit

Suppose that the objective is to generate a test for the stuck-fault s-a-0 on some signal g that is not a primary input (fault will be referred to as g s-a-0). Signal g can be expressed as $g = G(x_1,x_2,\ldots x_n)$ of the primary inputs. Similarly, primary output signal f_j can be expressed as a Boolean function of signal g and the primary input signals as

$$f_j = F_j(g,x_1,x_2,\ldots.x_n) \text{ where } 1 <= j <= m$$

The problem of generating a test for g s-a-0 can be recast as a problem of finding a Boolean vector $(x_1,x_2,\ldots.,x_n)$ that simultaneously satisfies two Boolean equations given below:

$$G(x_1,x_2,\ldots.x_n) = 1$$
$$F_j(1,x_1,x_2,\ldots x_n) \oplus F_j(0,x_1,x_2,\ldots x_n) = 1$$

for at least one j in the range $1<=j<=m$.

The two equations for the fault g s-a-1 are similar except that the first equation becomes $G(x_1,\ldots,x_n) = 0$. This formulation is referred to as Boolean Difference.

The problem of Boolean Difference form can be stated as below:

1. Express signal with stuck-at fault as a Boolean function of the primary input signals of the circuit.
2. Express the primary output signal as a Boolean function of signal with stuck-at fault and the primary inputs.
3. Find the logic value of input signals that satisfies the Boolean False Function and the XOR equation.

Test generation is the task of generating a test set that reveals all detectable single stuck-at faults. The circuit may have certain undetectable faults called *redundant faults*. Redundant faults do not affect the normal operation of the circuit. Their identification is, however, as hard as generating a test for detectable fault. The presence of a redundant fault implies that the Boolean function realized by the combinational circuit having fewer gates.

1.2. NP-Completeness of Test Generation

The concept of NP- completeness is introduced to prove that the amount of time, that is, the number of steps required to solve some specific problems is beyond a certain practical limit. A problem that can be solved in a polynomial time on a non-deterministic machine is referred to as NP problem. Of those NP problems that are not known to have any deterministic solution, a special sub-class is referred to as the NP-complete class.

The number of primary inputs and the number of signals in the combinational circuit are generally considered as the input size of test generation problem. Theoretical study by Ibarra and Sahni has shown that the test generation problem for combinatorial circuits is NP-complete [4]. This means that no test generation algorithm with a polynomial time complexity is likely to exist [33]. The non-polynomial (or exponential) time complexity here refers to the worst-case effort of test generation for any fault in the circuit. In fact, very restricted versions of the problem are NP-complete [7], [16].

Although test generation (and identification of redundancy) is a computationally difficult problem, practical test generation algorithms usually run in polynomial time [1]. The fact that the test generation problem is NP- complete means that the polynomial time cannot be achieved in all instances, that is, any test generation algorithm may encounter a circuit with a fault that cannot be solved in polynomial time.

1.3. Use of Parallel Processing in Test Generation

As we have seen that test generation is an NP complete problem, therefore it is becoming more and more challenging as the complexity of the circuit is increasing day by day. Traditional uni-processor algorithms have reached a saturation level and can not be improved much to yield better results. Therefore there is a need to look for effective parallel/ distributed algorithms for test generation. No effective parallel architecture or algorithm for test generation has yet been found. At least two possibilities exist. First, a conventional uniprocessor algorithm can be parallelized by finding portions where it can be pipelined or directly executed in parallel[3]. Second, the testing problem can be

reformulated in another form that is inherently parallel and new parallel methods can be developed for this reformulated problem. In this thesis work, the second approach is used and a parallel approach is proposed for test generation. The proposed approach is implemented using PVM.

In the area of combinatorial optimization, good parallel methods have been designed to solve several standard types of problems. Interestingly, most of this work has concentrated on speeding up the solutions of problems that could already be solved quite efficiently (i.e. in polynomial time) on a single-processor computer. The practical need for this speed up remains to be demonstrated. Certainly, the need to solve hard (i.e. NP-complete) combinatorial problems is certainly as pressing. Here, the parallelism offers a natural opportunity that has hardly been investigated. Even though one may never be able to change the exponential time to polynomial time, there is clearly a tremendous scope for parallelism as it may be the only way to obtain a solution in practice.

In the test generation, approaches used for parallelism include fault parallelism, search parallelism and element-level parallelism.

1.4. Thesis Organization

Chapter 1 of the report gives a brief introduction about VLSI test generation, and discusses the use of parallel processing in test generation. It gives the formulation of test generation problem and puts light on the NP-completeness aspect of the problem.

The literature survey including conventional methods for VLSI test generation, need for unconventional methods and their details has been given in the second chapter. One of the unconventional methods for VLSI test generation i.e. Boolean Satisfiability approach has been explored in detail in third chapter. The parallel approach for VLSI test generation based on this method has been proposed in the fourth chapter. The design and implementation details of the parallel algorithm have been given. The experimental results show the effectiveness of the approach. Chapter 5 concludes the work and discusses the future scope of the proposed work.

Chapter 2

Literature Survey

Logic circuits are modeled as interconnections of gates [3]. A signal is a Boolean variable that may assume only one of the two possible values represented by the symbols 0 and 1. Signals that are not output signals of any gate in the circuit are called *primary inputs*. Signals that realize the output functions of the circuit are called *primary outputs*.

Logic circuits can be categorized as *combinational* or *sequential*. Consider a logic circuit C consisting of signals $x_1,\ldots\ldots,x_n$. Let H_C (V_C, E_C), where V_C is the vertex set and E_C is the edge set, be the circuit graph associated with C. $V_C = \{x_1,x_2,\ldots.x_n\}$ and there is an arc (x_i,x_j) (- E_C (1<= I<=m and 1<=j<= n) if there is a gate x_j with x_i as an input signal. If H_C has no directed cycles, C is the combinational circuit; otherwise, C is the sequential circuit.

2.1. Conventional Test Generation Techniques

There has been a lot of emphasis on Automatic Test Pattern Generation for combinatorial circuits since these techniques can also be used to test specially designed sequential circuits. A wide range of techniques has been proposed for combinational circuit test generation. At one end of the spectrum are the exhaustive and random techniques. They are described in detail below.

Exhaustive Test Generation: If the number of primary inputs of the circuit is small, exhaustive test generation that applies all possible input vectors may be an obvious candidate [3]. It ensures 100 percent fault coverage. Exhaustive techniques can be applied to more complex circuits by partitioning of the logic into smaller components. Unfortunately, the logic partitioning problem is intractable, and hence, cannot be fully automated.

Random Test Generation: Random test generation is another simple technique that probabilistically generates input vectors and verifies if they detect any faults in the circuit. If a random vector detects any fault, then it is retained as a test. The number of vectors needed for high fault coverage can be large in some cases. The main advantage of random test generation is the ease of generating vectors. Its main disadvantage is that a randomly generated test set that detects the set of faults is much larger (usually 10 times or more) than a deterministically generated test set for the same set of faults [1].

At the other end of the spectrum are deterministic techniques.

Deterministic Techniques: These can be partitioned into two groups.

- Algebraic algorithms
- Structural algorithms

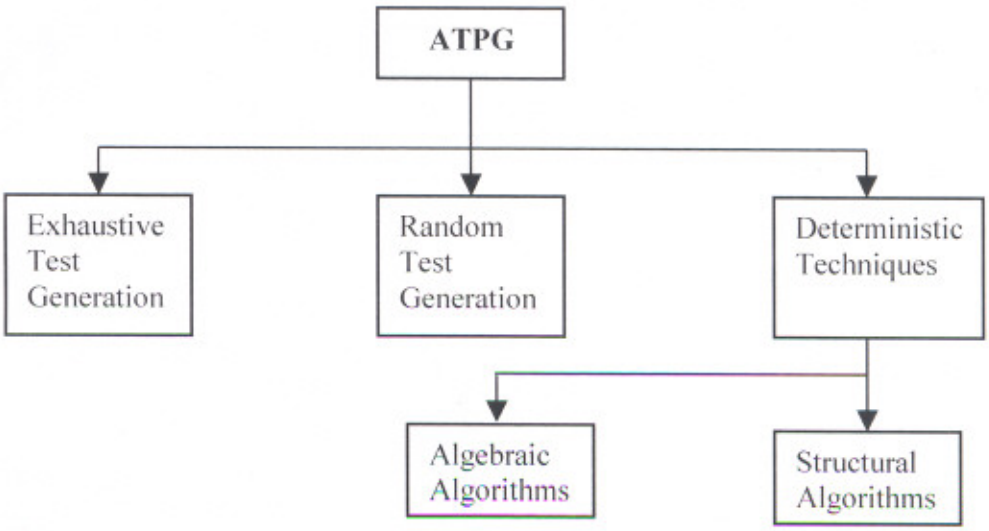

Fig 2.1. Conventional Test Generation Techniques

Algebraic Algorithms: One group consists of the *algebraic* algorithms that use the Boolean Difference formulation to symbolically solve the problem [3].

These algorithms have not been very practical for two reasons:

1. Symbolic solutions require excessive storage.
2. Heuristics required to tolerate NP-Completeness of practical test generation problems are not available. In other words, they are impractical for large circuits.

Structural Algorithms: The second group is of *structural* algorithms that exploit the transistor or gate-level representation of the circuit. These algorithms systematically enumerate the search space using branch-and bound method and employ heuristics to guide the search process. These are based on three basic operations: Fault Sensitization, Fault Propogation and Line Justification.

D - Algorithm: The first structural algorithm proposed was the D-algorithm [36]. Here, every signal in the circuits is considered as a decision point in the search process. If a decision results in some signal being assigned both logic values (0 and 1), we will have to retract that decision and make an alternative choice. i.e. using the process of backtracking [3]. The characteristic feature of D-algorithm is its ability to propagate errors on several reconvergent paths. The feature referred to as the multiple-path sensitization is required to detect certain faults, that otherwise would not be detected [1].

The D-algorithm performs poorly on circuits that require an excessive amount of backtracking. In other words, it is highly inefficient for Error Correction And Translation (ECAT) circuits with XOR gates.

PODEM: The Path Oriented DEcision Making (PODEM) algorithm offers a significant improvement. A distinctive feature of PODEM is its simplicity when compared to the D-algorithm. PODEM is a complete algorithm in that it will generate a test if it exists [26]. Here, only primary input signals are considered as decision points in the search process. Heuristics are used to dynamically determine the next decision point. PODEM implementations are known to run an order of magnitude faster than the D-algorithm on most circuits.

FAN: Fan-out Oriented test generation algorithm is faster and more efficient test generation algorithm than the PODEM algorithm [17]. It improves the PODEM algorithm through three strategies: it applies backtrace operation, performs analysis of circuit topology and reduces its search space.

TOPS and SOCRATES: TOPS (*TOP*ological *S*earch) proposed by Kirkland *etal.*[38] and SOCRATES proposed by Trischler *etal.* [37], also incorporate heuristics to accelerate the basic PODEM algorithm. TOPS performs extensive topological analyses of circuit connectivity to develop better heuristics for the branch-and-bound process. SOCRATES is developed on the basis of controllability/ observability analysis of digital circuits [28]. In another recent technique, to accelerate test generation, search-states are saved at every step. Equivalent search-states are identified and they are used to prune the search space.

CONT: The CONT algorithm pursues a different approach [39]. If the target fault is not detectable by the current input vector, CONT switches the vector to another undetected fault. In CONT-2 algorithm, all unspecified are randomly assigned prior to fault simulation. Candidate faults for target switching are identified by interleaving fault simulation steps with incremental test generation steps. In the CONT-2 algorithm, all specified inputs are randomly assigned prior to fault simulation [3].

Approximation algorithms may not guarantee a test but they have been proved to be effective for many practical circuits. Simulation-based directed search is one such technique that makes use of logic and fault simulation, and is particularly effective for sequential circuit test generation. The contest algorithm uses concurrent fault simulation and the TV – Set algorithm is based upon a new threshold – value model of the circuit. Both methods include gate delays to correctly determine tests for sequential circuits. Expert systems have also been used for test generation but with limited success.

2.2. Need for Unconventional Methods

The complexity of VLSI circuits is rapidly increasing. The VLSI circuits have, in fact, given birth to ULSI i.e. Ultra Large Scale Integration. The chips consist of so many components, number of inputs and number of outputs. For example, the Intel 80286 has over 100,000 transistors, the 80386 has 275,000 transistors, the 80486 has approximately 1,000,000 transistors. The Pentium processor of Intel has 3 million transistors. Also, there are more than 1 million chips produced in one go, that is, from a single wafer.

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 VLSI test generation, the cost and time for VLSI test generation increase considerably as the complexity of the circuit increases. This is shown in Fig 2.2 as in [10].

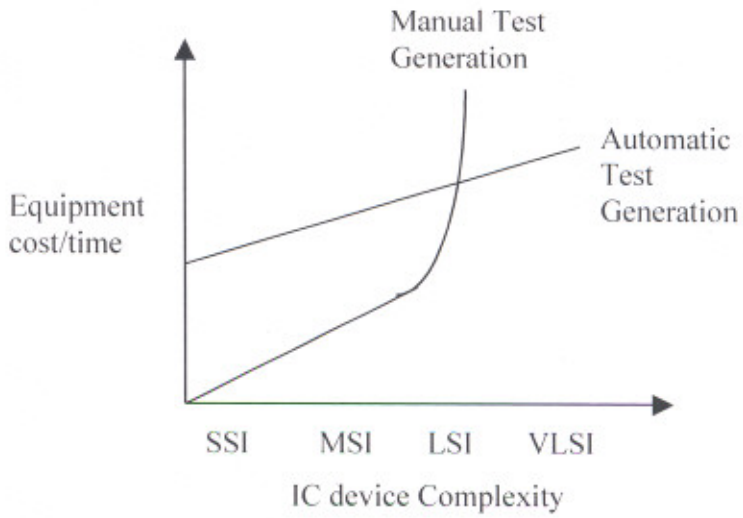

Fig 2.2. Equipment Cost/Time Vs IC Device Complexity

The frequently-quoted advantages of VLSI are reduced system cost, improved performance and greater reliability. These advantages, however, will be lost unless VLSI chips 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 VLSI chips drastically reduces the *controllability* and *observability* of the circuit [3]. Controllability refers to the ease of producing a specific internal signal value by applying signals to the circuit input terminals. Observability refers to the ease with which the state of internal signals can be deduced from the signals at the circuit output terminals.

A digital system consists of many printed wiring boards, each containing several VLSI chips. 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

VLSI chip further complicates the testing process. Complexity of test generation is enormous.

Testing is the most significant part of VLSI chip development cycle. If this takes so much time, energy and cost, then demand for VLSI chips will go down because they will not be available in the market at the right time. Moreover, this will also increase the cost of VLSI chips manifold. 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.

2.3. Unconventional Methods

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

ATG Constraint Network

The conditions for test generation are modeled by joining the circuit-under-test and its *faulty image* (a fault-injected neural network representing the circuit-under-test) back to back to their primary inputs [3]. Primary outputs of the two circuits are connected through an *output interface* to ensure that at least one primary output of the faulty circuit will differ from the corresponding fault-free circuit output.

For a single-output circuit, the output interface degenerates into a single inverter whose ports are connected to the primary outputs of the circuit-under-test and the faulty image.

It does not matter which way the inverter is connected because it is eventually replaced by a bi-directional neural network. For a circuit with n primary outputs, the output interface is a neural network consisting of n 2-input XOR gates and one n-input OR gate. The output of the interface circuit is an OR function of the outputs of n XOR gates. The inputs to the XOR are the corresponding primary outputs of the circuit-under-test and the faulty image. The output of the test generation is assigned the value 1 throughout test generation.

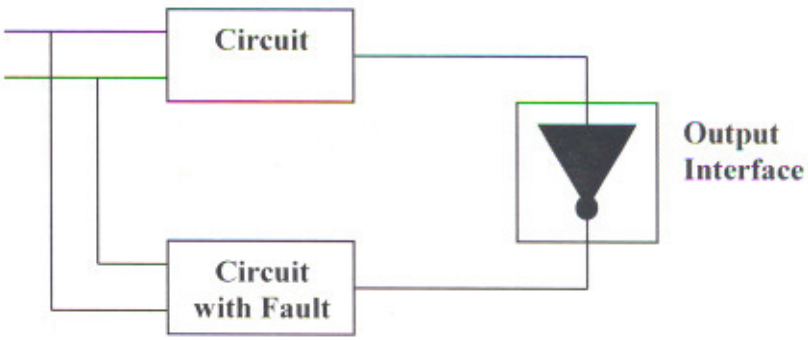

Fig 2.3. ATG Network for a Circuit with single output

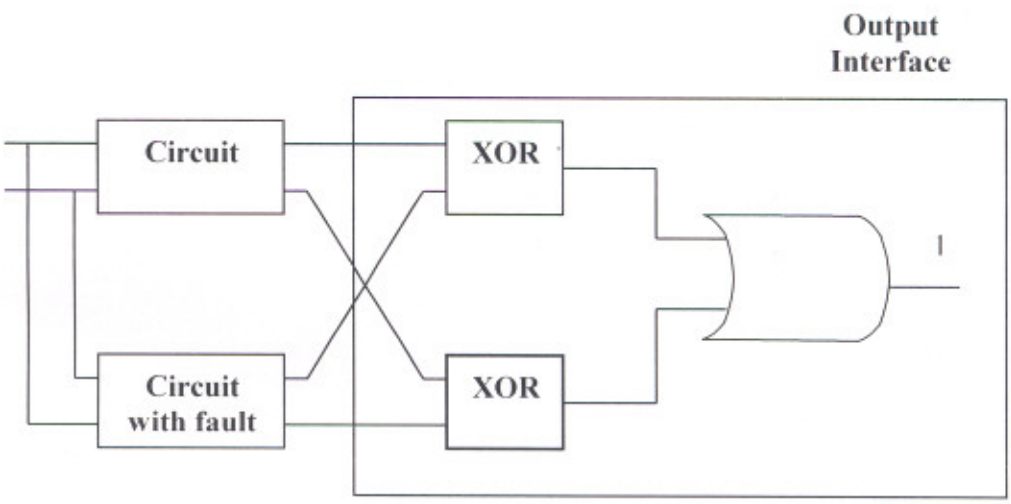

Fig 2.4. ATG Network for a Circuit with two outputs

The circuit-under-test, its faulty image and the output interface constitute the **ATG constraint network.**

A few important examples of unconventional techniques for VLSI Test Generation include:

1) Boolean Satisfiability Approach
2) Neural computing
3) Quantum Search

These are explained in brief below.

2.3.1. Boolean Satisfiability Approach

The Boolean Satisfiability approach for VLSI test generation was proposed by Tracy Larrabee in 1992. It is neither purely structural nor a purely algebraic approach[8]. 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 explored in detail in Chapter 3.

2.3.2. Neural Computing

. The circuit is modeled as a network of idealized computing elements, known as *neurons*, connected through bi-directional links. Our neuron is a binary (0-1) element. A neuron is viewed as a computing element that can assume one of two possible states: 0 or 1. Fig 2.5 shows the locality of a neuron in a network.

Associated with neuron i is a real number I_i, the threshold of the neuron and an output value V_i. A neuron is connected to its neighbors through links. A real number T_{ij} is associated with a link from neuron i to neuron j. It is assumed $T_{ij} = T_{ji}$. $T_{ii} = 0$ since the neurons do not have a self-feedback.

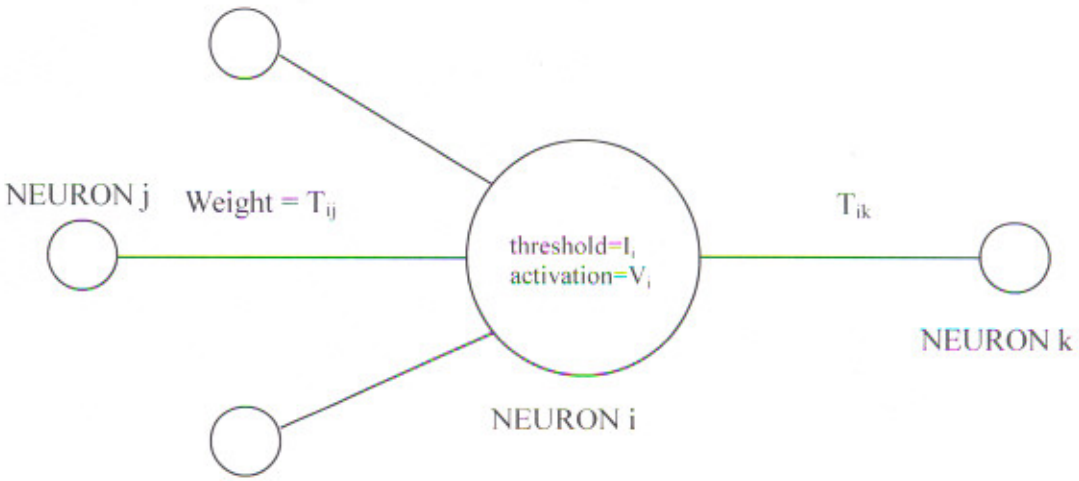

Fig 2.5. A typical Neuron

The relationship between the input and output signal states of a logic gate is expressed by an *energy function* such that the zero energy states correspond to the gate's logic function. The energy function, as given by Hopfield [31], is

$$E = -1/2 \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_{ij} - \sum_{i=1}^{N} I_i V_i + K \qquad \text{(Eqn. 3.1)}$$

where N is the number of neurons in the neural network, $T_{ij} = T_{ji}$ is the weight of the link between neurons i and j, V_i is the activation value of neuron I, Ii is the threshold value of neuron I and K is the constant, also, $T_{ii} = 0$.

Every net (signal) in the circuit is represented by a neuron and the value on the net is the activation value (0 or 1) of the neuron [3]. The neurons corresponding to the primary inputs (outputs) are called primary input (output) neurons. Neurons corresponding to the input (output) nets of the gate are called input (output) neurons. Each gate is independently mapped onto a neural network and the interconnections between the gates are used to combine the individual gate neural networks into a neural network representing the circuit. Neural networks for 2-input AND, OR, NAND, NOR, XOR and XNOR gates and the single-input inverter constitute the *basis set*. Any gates with more than two inputs are constructed from this basis set.

Massive parallelism inherent in neural networks is exploited to generate tests. The digital circuit is represented as a bi-directional network of neurons. The circuit function is coded in the firing thresholds of neurons and the weights of interconnection links. This neural network is suitably reconfigured for solving the test generation problem. A fault is injected into a neural network and an energy function is constructed with **global minima** at test vectors.

The test generation problem is formulated as an optimization problem such that the desired optima are the test vectors for some target fault. This formulation captures three necessary and sufficient conditions that any signal value must satisfy to be a test.

1. The set of values must be consistent with each gate's function I the circuit.
2. The signal in the fault-free and faulty circuits at the fault site must assume opposite values.

3. For the same primary input vector, the fault-free and faulty circuits should produce different output values.

A fault is injected by modifying the faulty image section of the ATG neural network. First of all, the ATG constraint network corresponding to the circuit-under-test is constructed. Corresponding to that ATG constraint network, an ATG neural network is constructed. This ATG neural network is simplified by neural pruning. Neural pruning collapses the neurons corresponding to the stuck-at fault since they have the same logic value.

2.3.3. Quantum Search Approach

It consists of creating a neural network for ATG constraint network. Then we minimize the energy function E to obtain the test vector. In the Quantum Search approach, specific aspects of quantum theory like superposition and quantum parallelism are applied to find the solution of energy function.

The basic variable used in quantum computing is a qubit. The difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ and $|1\rangle$ whereas a bit has only one state, either 0 or 1. The logic that is implemented with qubits is quite distinct from Boolean logic, and that is what has made quantum computing exciting by opening new possibilities [9].

For any VLSI circuit, an ATG network is created. This ATG network is then transformed into a neural network. The energy function of this neural network is then formulated. The

energy function is then minimized and a test vector can be generated. The method used here to minimize energy function is quantum search. Using quantum search algorithm, we can find these vectors at a speed much faster than classical algorithms.

The algorithm developed is so efficient that it requires only $\sqrt{N}$ (where N is the total number of vectors) iterations to find the desired test vector whereas; in classical computing, it takes N/2 iterations [9].

Chapter 3

Boolean Satisfiability Approach

Automatic Test Pattern Generation (ATPG) systems distinguish defective components from defect-free components by generating input sets that cause the outputs of the component under test to be different if the component than if it is defect-free. Existing algorithmic ATPG systems 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** 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 [8]. 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.

3.1. The Boolean Satisfiability Method

The Boolean Satisfiability is used for generating test patterns for single stuck-at faults in combinational circuits [8]. The test pattern is generated in two steps:

1) Extract the formula that defines the set of test patterns that detect the fault.

2) Run a Boolean Satisfiability algorithm to satisfy the formula.

3.1.1. Extracting the Formula

A directed acyclic graph represents the topological description of the circuit. The nodes of the graph are circuit inputs, outputs, gates and fanout points; the edges of the graph are circuit lines (wires), the sources of the graph are circuit outputs and sinks of the graph are circuit inputs. Every edge has an associated variable. Figure 4.1. shows a circuit and an associated DAG.

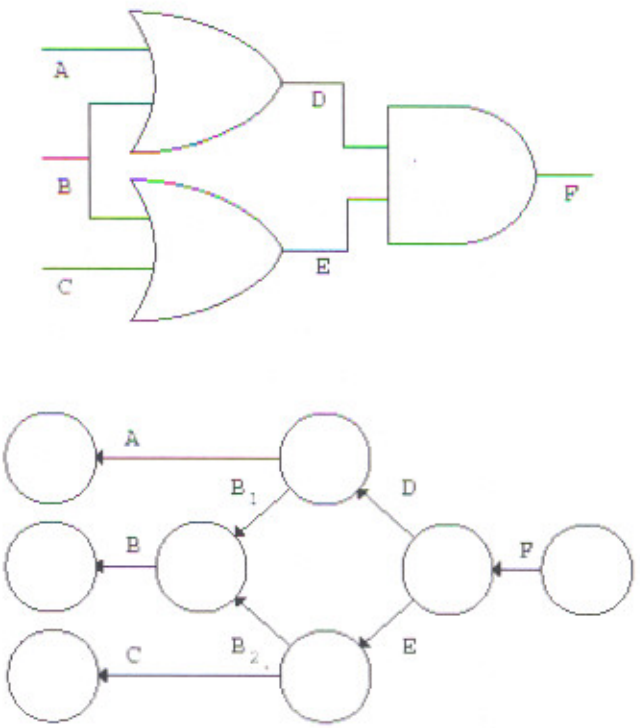

Fig 3.1. A circuit and its associated DAG

Every node of the DAG is tagged with a formula that represents the function performed by that gate or fanout point. For example, an inverter with an input X and an output Y will be tagged with the formula $\overline{Y} = X$; an AND gate with inputs X and Y and the output

Z will be tagged with the formula Z = X.Y. Every node has a formula that contains only variables for its incoming and outgoing edges.

3.1.1.1.Translating Formulas into CNF

Conjunctive Normal Form or CNF (also known as product of sums) is used. Formulas written in CNF are easily manipulated programmatically [8]. To get the CNF formula for an AND gate, we start with the formula

$$Z = X . Y$$

Because the formula P = Q is logically equivalent to $(P \rightarrow Q). (Q \rightarrow P)$, we transform our original equality into

$$(Z \rightarrow (X . Y)). ((X . Y)) \rightarrow Z)$$

Next, we transform all implications into disjunctions by using the fact that $P \rightarrow Q$ is logically equivalent to $\overline{P} + Q$ to get the formula

$$(\overline{Z} + X). (\overline{Z} + Y).(\overline{X} + \overline{Y} + Z)$$

This formula evaluates to 1 if and only if the values of the variables are consistent with the truth table of the AND gate.

In CNF formulas, one sum is called a clause. Clauses with only one, two or three elements are unary, binary or ternary clauses respectively. A formula with no ternary clauses is said to be in 2CNF (2-element conjunctive normal form).

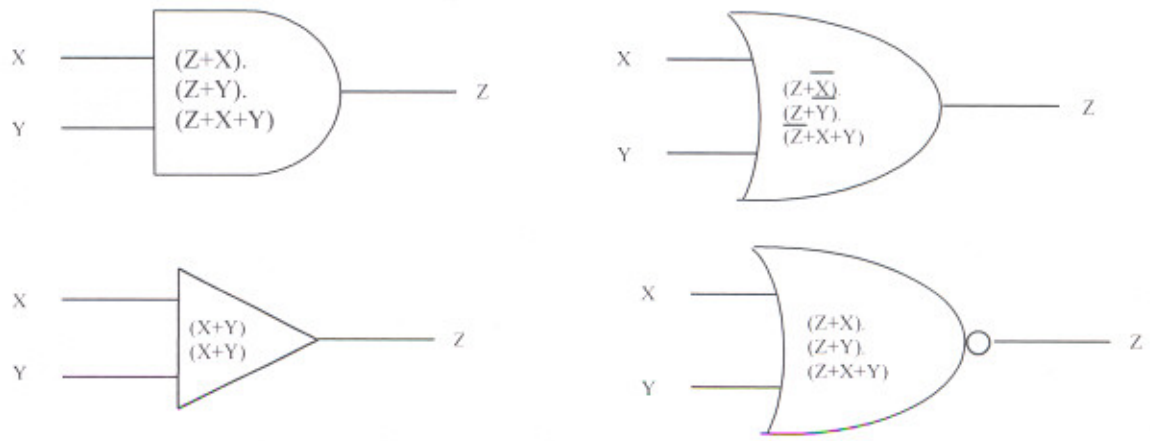

Fig 3.2. The Formulas for the basic gates

The CNF formulas for basic gates are shown in Fig 4.2 but the gates need not be basic to be included in this scheme. With the introduction of new variables, the CNF form of any formula can be produced in time and space linear in the size of original formula.

3.1.1.2. Formulas for Unfaulted and Faulted Circuits

Because each gate and fanout point must be tagged with the formula that must be independently satisfied, we can extract a characteristic formula for any circuit output (or subcircuit output) by starting at the output and walking the graph, taking the conjunction of all the formulas for the visited nodes. Since the formula for every component must be independently satisfiable, the conjunction of formulas must also be satisfiable. Fig 4.3 shows a circuit with each gate labeled by its characteristic formula. The formula for the output of this circuit is

$$(X+\overline{D}).(X+\overline{E}).(\overline{X}+D+E).(\overline{D}+A).(\overline{D}+B).(D+\overline{A}+\overline{B}).(C+E).(\overline{C}+\overline{E})$$

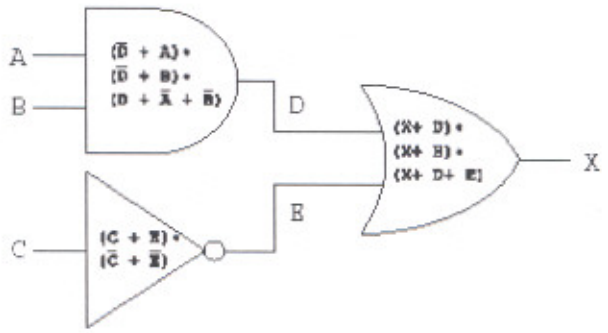

Fig 3.3. Combinational Circuit with labeled gates

We can represent the faulted version of an unfaulted circuit by making a copy of the circuit, renaming the variables and inserting two new nodes that represent the presumed disrupted connection in the faulted circuit. That is, if the circuit has the fault we want to test for, one value will be generated at the fault site, but another value will be forwarded on to the rest of the circuit. We tag the new nodes with unary clauses that indicate the behavior of the fault we are interested in. For example, Fig 4.4 shows the faulted version of the circuit in Fig 4.3. Because wire D is stuck-at 1, we add the formula (D') to the node representing the faulted behavior at the fault site, and we add the formula (D) to the node representing the correct behavior at the fault site.

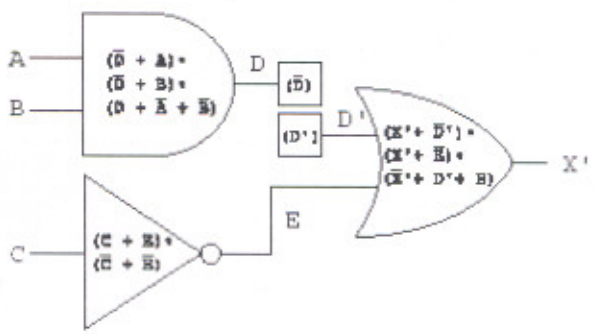

Fig 3.4.Circuit of Fig 3.3 with D stuck-at-1

Because the faulted and unfaulted circuits will have identical behavior except at these nodes that are affected by the fault, only the variables that are associated with wires that lie on a path between the fault site and a circuit output need to be renamed.

We can extract the formula for the faulted output in the same way as we extracted a formula for the unfaulted circuit: by starting at the faulted output, walking the DAG, and taking the conjunction of all the encountered nodes of the DAG.

To test for the given fault, we need only to find a set of inputs that cause the faulted output to differ from the unfaulted output. We will have a formula for all possible tests if we take the conjunction of two extracted formulas and add an additional formula for the XOR of the faulted and unfaulted output.

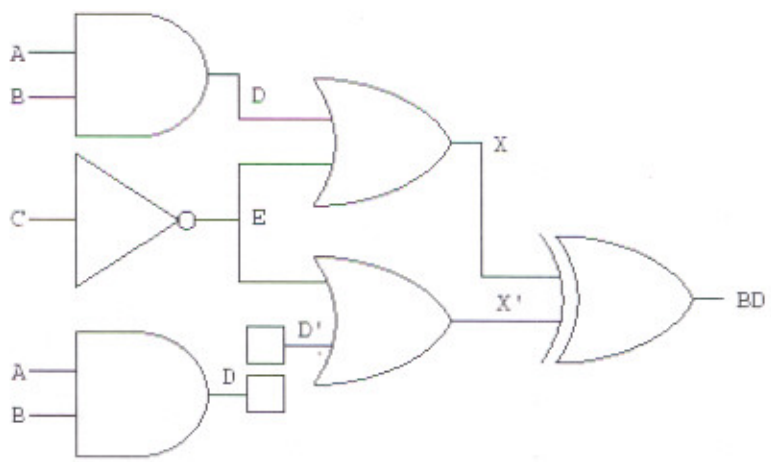

Fig 3.5. The XOR of the faulted and unfaulted circuits must be 1

The extracted formula is same as that produced by Boolean Difference method, in the sense that they are both satisfiable or unsatisfiable: Every set of satisfiable bindings for the formula produced by Boolean difference method is consistent with the satisfying

binding for the Boolean satisfiability method, and every set of satisfiable binding for boolean satisfiability formula is a superset of satisfying binding for the formula produced by Boolean difference method. The formula extracted by this system is not exactly same as that produced by Boolean difference method because this formula has extra variables in it. These redundancies will be helpful in finding a satisfying assignment for the formula.

3.1.2. Satisfying the Formula

The problem of satisfying a CNF formula, SAT, is an NP-complete problem [4]. The problem has been transformed from one problem that in worst case will take exponential time in the number of circuit inputs into another problem that in the worst case will take exponential time in the number of its variables. Fortunately, the class of formulas generated by combinational circuits is an interesting sub-class of all CNF formulas and we can use this fact to try to avoid the worst-case behavior of SAT. Many researchers have recognized that the average behavior of a SAT algorithm can be improved dramatically if the set of formulas to be solved fit a restricted profile [18], [19]. The set of formulas produced by combinational circuits fits such a profile.

At least two-thirds of the clauses generated by Boolean difference of a combinational circuit have only two disjuncts (are in CNF). This is true because each 2 – input unate gate contributes two binary clauses and one ternary clause. Unate gates with more than two inputs contribute more than two-third binary clauses and fanout points, buffers and inverters contribute only binary clauses. In practice, it has been found that 80% to 90% of the clauses are in 2CNF. The problem of satisfying the 2CNF formula, 2SAT, is

satisfiable in time linear in the number of clauses plus the number of variables [20]. There may be an exponential number of 2SAT solutions but the information from the ternary clauses can be used to guide the iteration through the 2SAT assignments.

3.1.2.1. Using 2SAT to solve SAT

An algorithm from 1970's is used for satisfying the 2CNF formula [20]. The first step is to construct an *implication graph* [8]. Each 2CNF clause (X+Y) can be viewed as two implications: $\overline{X} \rightarrow Y$ and $\overline{Y} \rightarrow X$. The implication graph for a 2CNF formula shows all the constraints imposed by 2CNF clauses on the logic values of the variables involved. The following figure (Fig 3.6) shows the circuit and the corresponding implication graph.

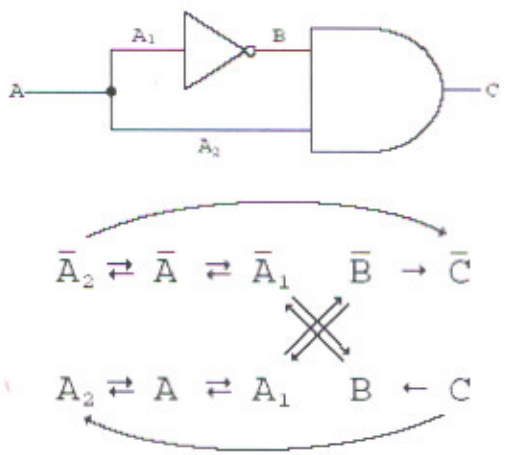

Fig 3.6. A circuit and its corresponding implication graph

More formally, for each variable X occurring in the 2CNF clauses, there are two vertices in the graph, labeled X and X. For every 2CNF clause (X + Y), there are two directed edges in the graph: one from X to Y and one from Y to X. the edge represents the logical implication between two literals. Now we can bind logic values to the variables in the

graph. Any assignment is legal as long as it does not cause a node labeled 1 (true) to precede a node labeled 0 (false).

Before we label the graph, we can simplify it by reducing each *strongly connected component* to a single node. A strongly connected component can be defined as a maximal set of nodes in a graph such that every node in the set is reachable from every other node in the set. Fig 3.6 shows the implication graph with two strongly connected components. Both of these can be reduced into a single node as shown in the following figure (Fig 3.7).

Fig 3.7. The reduced Implication Graph

If any strongly connected component contains both a literal and its negation, the formula is unsatisfiable because each strongly connected component represents a set of variable that are in an equivalence class. After each strongly connected component is reduced to a single node, the graph will not contain any cycles. Now we can find at least one satisfying binding for the 2CNF portion formula: First we visit the vertices in any topological order. For each variable, if the negated variable appears before the un-negated variable in the topological order, we bind the variable to 0 (false); otherwise, we bind the variable to 1 (true).

3.1.2.2. Iterating through 2SAT Bindings

The method for constructing a satisfying assignment for the 2CNF portion of the formula has just been described. The values were assigned so that no node bound to 1 has a directed path to a node bound to 0. Clearly, there are many such assignments but we want to construct a 2SAT assignment that is consistent with the ternary clauses. This can be done by defining an order for the 2SAT assignments and then constructing each assignment only so far as it is consistent with the ternary clauses.

We order the 2SAT assignments by ordering the variables that appear in the 2CNF clauses. This defines a total order on the 2SAT solutions: one total assignment precedes another if the n-bit binary number representing the values of the variables precedes the n-bit binary number for the other assignment. Larrabee suggests the following approach to test generation: Generate 2SAT solutions to the truth function in some fixed order until one is found that also satisfies the 3CNF clauses. There can be many such assignments but the purpose is to find that 2CNF binding that is consistent with the 3CNF or ternary clauses. There can be exponential number of 2CNF solutions to try. There are certain heuristics suggested by Larrabee so that the number of iterations is lesser.

3.1.2.3. Terminating the Search

We terminate the search for a 2SAT binding that satisfies the entire formula in one of three ways:

1) We find a satisfying binding.

2) We prove that no binding exists.

3) We exceed the amount of computational effort we are willing to spend.

Chapter 4

Parallelized Boolean Satisfiability Approach

Parallel Processing refers to the concept of speeding-up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. A program being executed across *n* processors might execute *n* times faster than it would using a single processor.

Need for Parallel Algorithms

The complexity of today's applications coupled with the widespread use of parallel processing has made the design and analysis of parallel algorithms a necessity. Many problems can be solved by massive parallelism. The emergence of high-performance, massively parallel computers demands the development of parallel algorithms to take advantage of this technology.

Parallel Processing is information processing that emphasizes the concurrent manipulation of data elements belonging to one or more processes. It involves multiple processes, which are active simultaneously, and solving a given problem, generally on multiple processors. The critical aspect here is "solving a given problem". The processes all must be concerned with the solution of one single problem. In other words, there must be interactions among the units.

Problems are parallelizable to different degrees. For some problems, assigning partitions to other processors might be more time consuming than performing the processing locally. Other problems may be completely serial. For example, consider the task of digging a hole. Although one person can dig a hole in a certain amount of time, employing more people does not reduce this time. Because, it is impossible to partition this task, it is poorly suited to parallel processing. Therefore, a problem may have different parallel formulations, which result in varying benefits, and all problems are not equally amenable to parallel processing.

4.1. Parallel Processing Concepts

In this section, the parallel processing concepts and utilities that have been used in the proposed approach (and its implementation) have been discussed in brief.

Taxonomy of Parallel Architectures

Flynn's Taxonomy of Parallel Architectures is the best-known classification scheme for parallel computer architectures. It is based on the dual concepts of data stream and instruction stream. Four classes of architectures result from the multiplicity of data and instruction schemes as shown in Fig 4.1.

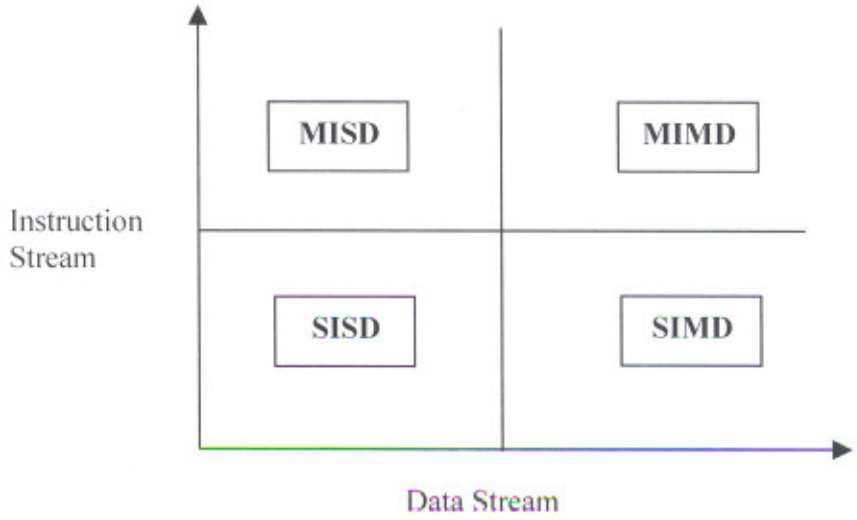

Fig 4.1. Parallel Architectures

These are as following:

- Single Instruction, Single Data Stream (SISD)
- Single Instruction Stream, Multiple Data Streams (SIMD)
- Multiple Instruction Streams, Single Data Stream (MISD)
- Multiple Instruction, Multiple Data Streams (MIMD)

In MIMD, there are further two classifications: Shared Memory MIMD Machine and Distributed Memory MIMD. The Distributed Memory MIMD Machine is shown in Fig 4.2.

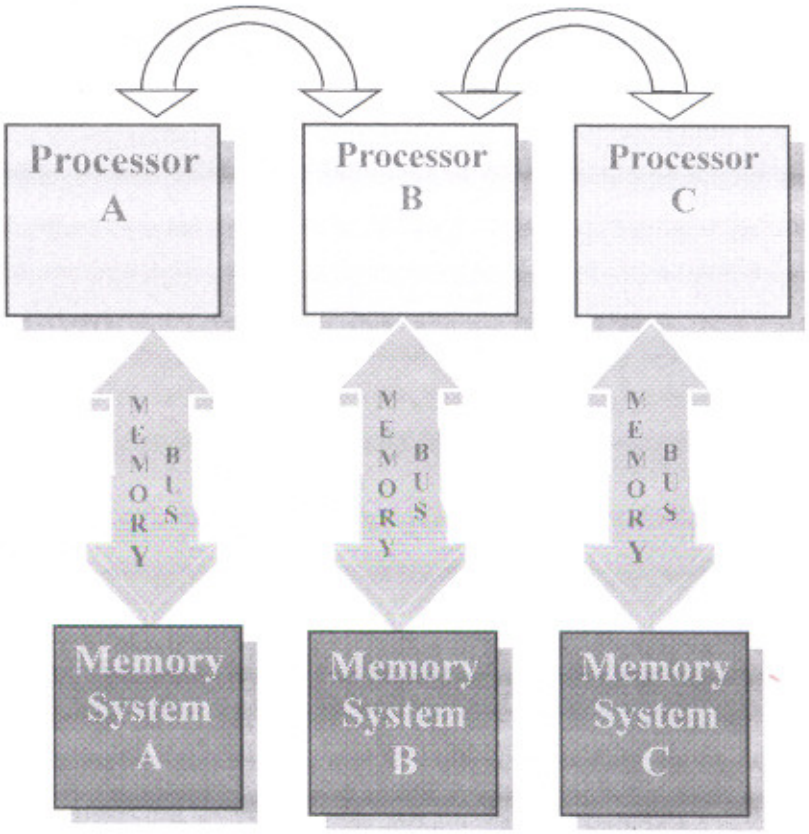

Fig 4.2. Distributed Memory MIMD Machine

PVM (Parallel Virtual Machine)

The **PVM** system is a programming environment for the development and execution of large concurrent or parallel applications that consist of many interacting, but relatively independent, components. Therefore, large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. Fig 4.3 shows the PVM system overview. It shows the PVM computation model and architectural overview.

PVM is a byproduct of an ongoing heterogeneous network computing research project involving the authors and their institutions.

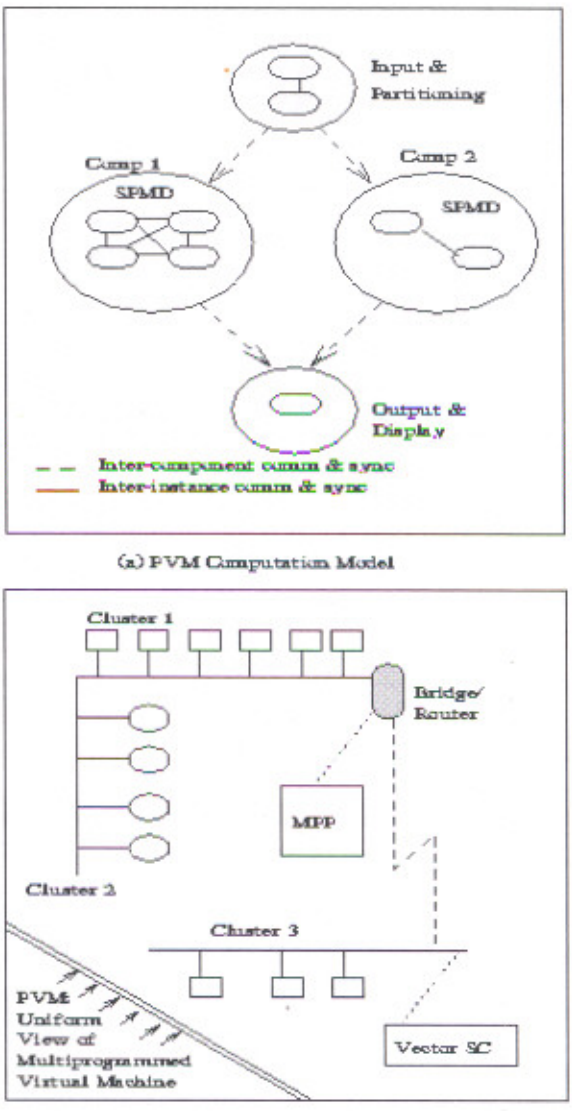

Fig 4.3. PVM System Overview

The PVM system is composed of two parts. The first part is a daemon, called *pvmd3* and sometimes abbreviated as *pvmd*, that resides on all the computers making up the virtual

machine. The second part of the system is a library of PVM interface routines. It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

The PVM computing model is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. The general paradigm for application programming with PVM is as follows. A user writes one or more sequential programs in C, C++, or Fortran 77 that contain embedded calls to the PVM library. Each program corresponds to a task making up the application. These programs are compiled for each architecture in the host pool, and the resulting object files are placed at a location accessible from machines in the host pool. To execute an application, a user typically starts one copy of one task (usually the ``master" or ``initiating" task) by hand from a machine within the host pool. This process subsequently starts other PVM tasks, eventually resulting in a collection of active tasks that then compute locally and exchange messages with each other to solve the problem. While the above is a typical scenario, as many tasks as appropriate may be started manually. Tasks interact through explicit message passing, identifying each other with a system-assigned, opaque TID.

4.2. The Proposed Parallel Approach

VLSI test generation is a highly complex process. This is because VLSI circuit consists of millions of components and interconnections. If the process of test generation is parallelized, the complexity of test generation will reduce manifold.

Boolean Satisfiability Approach is an unconventional method of VLSI test generation. This method has already reduced the complexity of test generation process to a great extent, making it solvable in linear time. But with this approach also, the number of runs possible are exponential. This can be simplified using parallel approach for the same. Also, in the first step, the extraction of the formula may take a lot of time due to the presence of millions of components and interconnections in a VLSI circuit. Therefore, a parallel approach has been proposed for both the steps.

We propose an effective scalable parallel/distributed algorithm for both the steps of Boolean Satisfiability Approach. This has been explained in detail.

The first step consists of the extraction of formula for the VLSI circuit. In this, all the nodes of the graph (the circuit is viewed as a DAG) have to be labelled with a formula, which must be converted, in a conjunctive normal form. All the inputs, outputs, gates and fanout points are treated as nodes. The parallelization of extraction of the Boolean truth function can be done to achieve the result in lesser time.

The second step consists of satisfying the Boolean truth function using a Boolean Satisfiability algorithm given by Larrabee. Larrabee suggests the following approach for the second step: Generate 2SAT solutions of the Boolean truth function in some fixed order until one is found that also satisfies the 3CNF clauses. There can be exponential number of solutions to try. There are certain heuristics given by Larrabee so that the number of iterations is lesser. At the first level, since there can be different orders in

which the assignments can be done, different slave tasks may do the processing in different orders. Larrabee suggests that the variables that impose more constraints on others should be assigned values first. When the variables place equal amount of constraints on others, then lexicographic order may be followed. So different combinations have to be tried before getting a 2SAT solution. The number of processors available can be used to try the different orders. At the second level, after performing the 2CNF computation, each 2CNF solution has to be checked to satisfy 3CNF clauses as well. The consistency with the 3CNF clauses can be checked concurrently.

The first step i.e. extraction of the function has been implemented while the second step remains to be implemented. The design and implementation details have been discussed below in detail.

4.3. Design and Implementation Details

The following sections discuss the design and implementation details of the underlying work in detail.

4.3.1. Design Details

The parallel/distributed algorithm that I am using is based not on both data parallelism and functional parallelism. Each slave works on a different piece of data and depending upon the type of data, it processes it with the corresponding set of instructions. This kind of parallelism is termed as MIMD (Multiple Instruction Multiple Data) according to Flynn's taxonomy of classification. As discussed above, in MIMD, there are further two classifications: Shared Memory MIMD Machine and Distributed Memory

MIMD Machine. In the underlying implementation, the memory is distributed with the processors; therefore, the Distributed Memory MIMD architecture has been used. This has been shown in Fig 4.2.

The *degree of concurrency* of the underlying application depends upon the size of the input. They are directly proportional. If the number of components in the VLSI circuit is large, more number of slave tasks will be spawned and hence, there will be more degree of concurrency. These slave processes run on the available number of processors. They are *mapped dynamically*. The *size of input data* associated with each task is same i.e. 4 bytes. The *size of output data* varies depending upon the computation of each task.

4.3.2. Implementation Details

PVM (Parallel Virtual Machine) *message-passing architecture* with the *master-slave approach* has been used in order to implement the algorithm.

The PVM system currently supports C, C++, and Fortran languages. We have used *C language* to develop the underlying application.

All tasks are identified by an integer *task identifier* (TID). Messages are sent to and received from tids. Since tids must be unique across the entire virtual machine, they are supplied by the local pvmd and are not user chosen.

The implementation has been done on the heterogeneous systems interconnected by a network. PVM can exploit several different types of heterogeneity as listed below:

- Architecture
- Data Format
- Computational Speed
- Machine Load
- Network Load

The parallelization has been done in a *distributed environment* consisting of *heterogeneous Linux workstations* using PVM.

The algorithm can be executed on the resources normally available in the distributed environment. A very important factor for an efficient parallel algorithm development is the dividing mechanism of the complete problem into sub-problems so that each workstation has sufficient load and takes almost an equal amount of time to process the same. This has been taken care of in the underlying algorithm.

The parallel execution times for varying number of processors are calculated and compared.

The master and slave algorithms are described below:

4.3.2.1. The Master Algorithm

i. The master task reads the circuit specifications from the input file.

ii. It then spawns the number of slaves depending upon the size of the input. The number of components varies for different circuits. If the number of components = j and the number of slaves = i, *then*

if j = n, then i = n.

iii. It puts all the data in an array A[1:m]. The master packs a different individual data in the form of a string for every slave and sends it. For each component *j*, it initialises a string that consists of its input(s), output and the type of component i.e. fanout point, AND gate, NAND gate etc.

iv. The master receives a different output *k* from all the slaves *i* and unpacks each output. The size of output varies depending upon the type of component for which the computation is done. It performs the conjunction of all the outputs because the Boolean truth function is the conjunction of all the components of the circuit.

v. The master sends the handshake parameter to all the slaves if it receives the output completely.

vi. Kills the slaves and writes the final output to a file. This is the resulting Boolean truth function that must be satisfied in order to obtain the test patterns.

vii. The total execution time *t* is calculated, where t = t2 – t1; t2 = final time and t1 = initial time.

4.3.2.2. The Slave Algorithm

i. The slaves call the parent id routine to obtain the id of the master task.

ii. Each slave *i* receives a different input *j* from the master and unpacks its own string from the buffer. Here, **data parallelism** is done because each slave receives a different data to process.

iii. Each slave then finds the component type (gate, fanout point etc.) and performs the computation of the formula for it in conjunctive normal form depending upon its type. The computation of all the components takes place concurrently. Here, **functional parallelism** takes place because there is a different computation performed for different type of a component.

iv. Each slave packs its own output k and sends it to the master by specifying the parent id.

v. The slaves receive the handshake parameter from the master. The data is resent if it has not been received by the master.

The implementation of the second step of the Boolean Satisfiability approach can also be done using message-passing architecture.

4.4. Experimental Results

The implemented algorithm has been tested using ISCAS'85 Benchmark circuits [17]. The circuits used are c17, c432, c499 and c880. Table 4.1 shows the parameters of the ISCAS'85 circuits. The results obtained have been summarized in Table 4.2. The table shows the average execution time (in seconds) taken for the computation of Boolean truth function of the considered ISCAS'85 benchmark circuits with varying number of processors. The trend of reducing average execution times taken for the Boolean truth function computation with increase in the number of processors has been shown in the following graph.

ISCAS Circuits → Parameters ↓	c17	c432	c499	c880
Lines from Primary Input Gates	5	36	41	60
Lines from Primary Output Gates	2	7	32	26
Lines from Interior Output Gates	4	153	172	357
Lines from Fanout Stems	6	236	256	437

Table 4.1. Parameters of the ISCAS'85 Circuits

ISCAS Circuits → No of Processors ↓	c17	c432	c499	c880
1 Processor	.01	.10	.15	.20
2 Processors	.01	.06	.09	.10
3 Processors	.01	.04	.06	.07
4 Processors	.02	.03	.03	.04

Table 4.2. Average Execution Time taken (in seconds) for the Boolean truth function computation

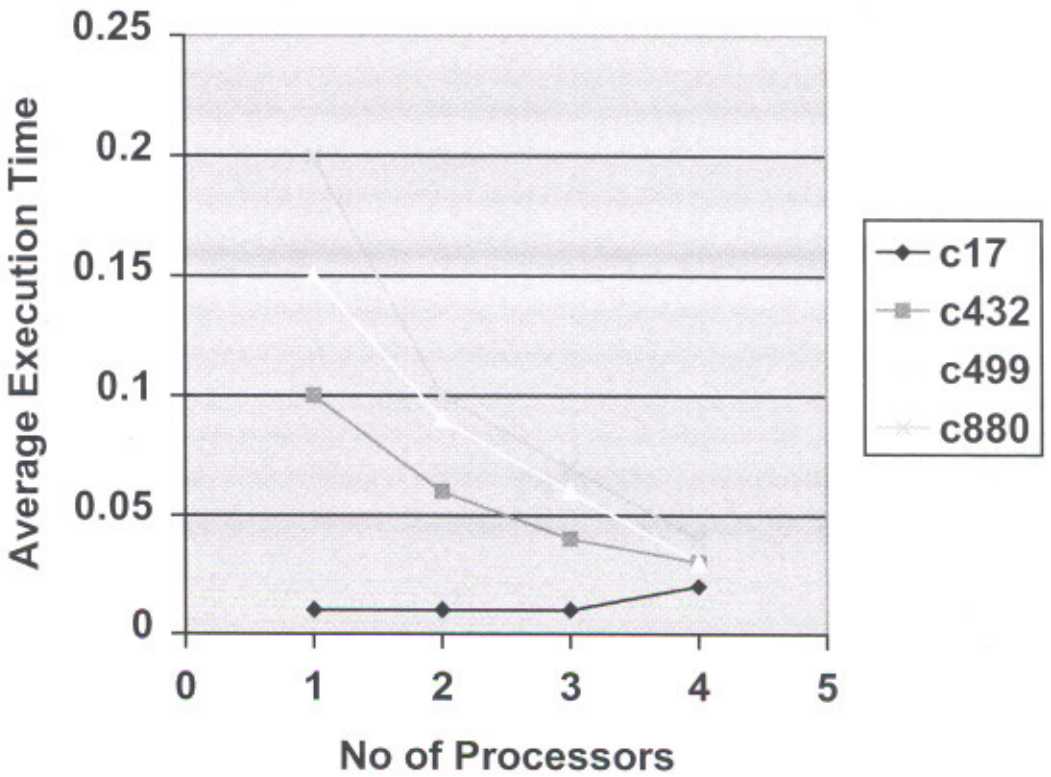

Fig 4.4. Average Execution Time (in seconds) vs. No of Processors for Boolean Truth Computation

Speedup is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processors. Speedup S is given by

$$S = T_s / T_p$$

Where T_s is execution time taken on a *single* processor

T_p is the execution time taken on p processors

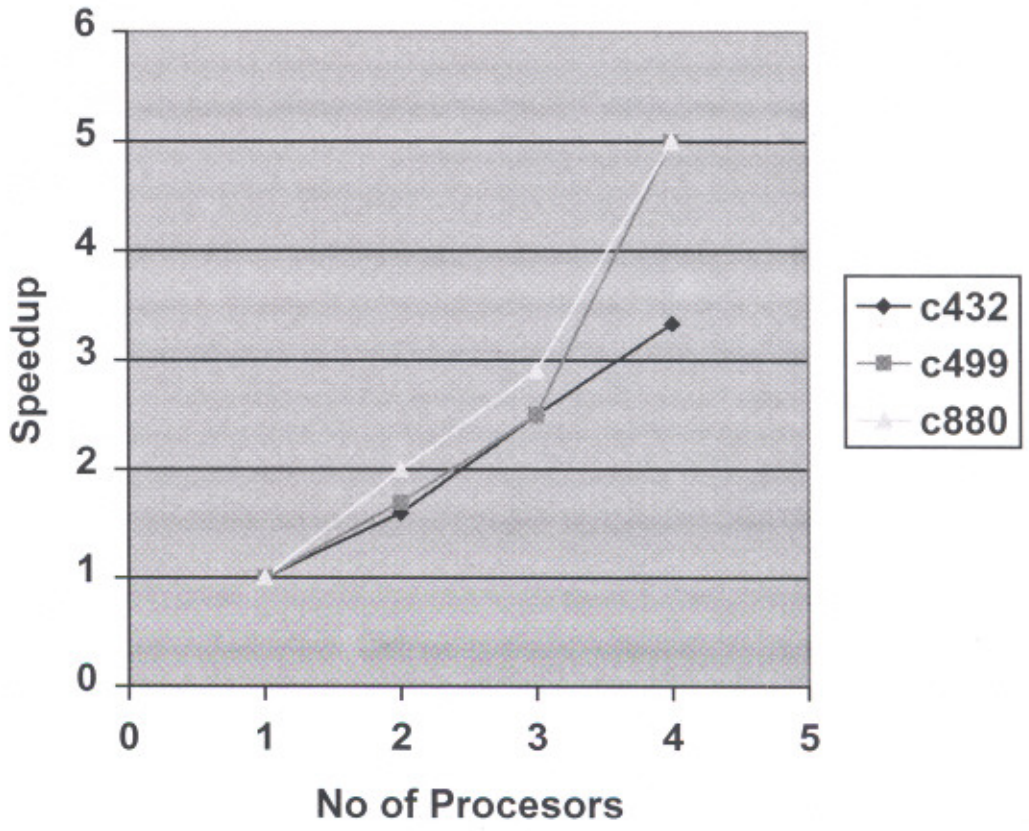

Fig 4.5. Speedup vs. No of Processors for ISCAS'85 Benchmark Circuits

The graph above (Fig 4.5) shows the speedup with increase in the number of processors for the ISCAS'85 Benchmark Circuits. In case of c432 and c499 circuits, there is s linear speedup. There is not a considerable difference between the speedup of c432 and c499 circuits. This is because their size is nearly the same. In case of c880 benchmark circuit, there is an exponential speedup. It has been observed that the *speedup* improves with increase in the size of the circuit. i.e. the bigger the circuit, the more is the speedup. Therefore, parallelization is ideally required for VLSI circuits since VLSI circuits consist of millions of components.

It has been observed that if the components in VLSI circuit are lesser as in the case of c17 circuit, adding the number of processors doesn't change the total time of computation (or it begins to increase). On the other hand, as the size if the circuit increases, the performance improves manifold. The speedup is enormous as the bigger circuits are tested with 1, 2, 3 and 4 processors. But this also has a limit. As the number of processors increase, the intercommunication overhead between the processors also increases.

The algorithm is highly *efficient* because the speed up increases considerably with the increase in the number of processors. This is obvious from the above graphs. As the efficiency E is given by

$$E = S / p$$

where S is the speedup and

p is the number of processors

The more the speedup with increase in the number of processors, the more is the efficiency.

The *cost* of solving a problem on a parallel system is defined as the product of parallel run time and the number of processors used. In our algorithm, the cost for parallel computation is almost the same for c432, c499 and c880 circuits when 4 processors are used. On the other hand, the difference in cost is considerable when a single processor is used.

4.5. Common Errors Encountered

When implementation of the application was done, there were various kinds of errors that were encountered. Some of them were the initial ones that helped me to learn more about parallel processing and PVM.

After the prototype of the application was made and run, the project was tested using different inputs and stress-tested by increasing the size of input on a single processor and more than one processor. The common errors that were encountered during this phase included *segmentation fault* and *message routing* errors.

Segmentation fault was encountered at various points as the system was tested on bigger circuits. It was encountered at a point where the master packs and sends the data in small chunks to the slaves after reading it from the file. This error was removed by making alterations in the arrays, ways of storing the data and passing the data etc. Similarly, it was encountered at other points and was removed by making suitable modifications. It was found that when the sequential version of the same program was run, there was no such error. Therefore, it was experienced that in parallel processing, memory allocation becomes more complicated and has to be taken care of.

Another fault that was encountered was message routing error. It was related to a message routing function *mxfer( )* that polls for the messages, optionally blocking until one is received or until a specified timeout. It calls function *mxinput( )* to copy messages

into a task and reassemble messages. In the generic version of PVM, *mxfer( )* uses *select()* to poll all routes (sockets) in order to find those ready for input or output. *pvmctrl( )* is called by *mxinput( )* when a control message is sent or received. The related error can be removed by following certain instructions such as not starting PVM from root and unlimiting the open file descriptors etc.

Chapter 5

Conclusion and Future Scope of Work

In this thesis, we have proposed the parallelized version of Boolean Satisfiability Approach for VLSI Test Generation. This chapter summarizes the main conclusions and suggests the future scope of work

5.1. Conclusion

The Boolean Satisfiability approach is one of the unconventional methods employed for VLSI test generation. Since VLSI test generation is a highly complex process, there is an intense need of unconventional methods for VLSI test generation that help to reduce the complexity, cost and time of test generation manifold.

The Boolean Satisfiability approach was proposed by Larrabee in 1992. It generates test patterns in two steps: Extracting the Boolean truth function and satisfying the function using Boolean satisfiability algorithm. Both these steps have been discussed. The approach is highly parallelizable and we have investigated this in detail.

The parallel version of Boolean Satisfiabilty approach has been proposed. The effectiveness of the approach has been demonstrated by implementing the first step i.e. the extraction of Boolean truth function for the VLSI circuit. The design and

implementation details of parallel algorithm have been given. The degree of concurrency of the application depends upon the size of the input i.e. the number of components in the circuit. MIMD architecture i.e. both data and functional parallelism have been used. PVM message-passing architecture with master-slave approach has been used to implement the algorithm.

The implemented algorithm has been tested using ISCAS'85 circuits. Experimental results have been shown using c17, c432, c499 and c880 circuits. The circuit was tested with different types of input and stress-tested using varying input sizes. The various errors that were encountered are discussed.

5.2. Future Scope of Work

The future scope of work in this field includes improvement in the speedup of the parallel algorithm that has been implemented. This can be done by increasing the number of processors or by improving the algorithm etc. At a certain point, the increase in number of processors also doesn't help to improve the speedup. It has a limit. For bigger VLSI circuits, however, more number of processors will improve the speedup to great extent.

The work can also be extended to implement the parallelization of the second step. This can also be done using message-passing architecture. As suggested above, the different processors can do 2CNF assignments in different orders at the first level. At second level, they can check the consistency with 3CNF parallely. If both the steps of Boolean Satisfiability approach for VLSI test generation are effectively parallelized, the test generation will be highly effective, faster and economical process.

References

1. Miron Abramovici, Melvin A. Breur, Arthur D. Friedman, "Digital Systems Testing and Testable Design", *Jaico Publishing House*, Mumbai, India, 1997.
2. Vishwani D. Agrawal and Sharad C. Seth, "Test Generation for VLSI Chips", *The Computer Society Press*, IEEE Catalog Number EH0278-2, 1988.
3. Srimat T. Chakradhar, Vishwani D. Agrawal, Michael L. Bushnell, "Neural Models and Algorithms for Digital Testing", *Kluwer Academic Publishers.*
4. O. H. Ibarra and S. Sahni, "Polynomially Complete Fault Detection Problems", *IEEE Trans. Computers*, Vol. C-24, March, 1975, pp 242 - 249.
5. S. Bawa and G.K. Sharma, " A Parallel Transitive Closure Computation Algorithm for VLSI Test Generation", *Lecture Notes in Computer Science* 2367, Applied Parallel Computing, Springer-Verlag Berlin Hiedelberg, 2002, pp. 243 - 252.
6. M. Abramovici, M.A Breuer, and A.D. Friedman, "Digital Systems Testing and Testable Design", *Computer Science Press*, New York, 1990.
7. H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problem for Combinational Circuits", IEEE Trans. On Computers, C-31(6), June 1982, pp. 555 – 560.
8. Tracy Larrabee, "Test Pattern Generation Using Boolean Satisfiability", *IEEE Trans. On Computer-Aided Design*, Vol. 7, January 1992, pp. 4 -15.

8. Tracy Larrabee, "Test Pattern Generation Using Boolean Satisfiability", *IEEE Trans. On Computer-Aided Design*, Vol. 7, January 1992, pp. 4 -15.
9. Amardeep Singh, Sarbjeet Singh, "Applying Quantum Search to Automated Test Pattern Generation for VLSI Circuits", 2003.
10. Vivek Jhamb, "Test Generation for VLSI Circuits", *BITS Research Project*, 1992.
11. W.E. Den Beste, "Using the Software Emulator to Generate and Edit VLSI Test Patterns", *Electronics Test*, March, 1984, pp. 42-52.
12. T. Larrabee, "Efficient Generation of Test Patterns using Boolean Difference", *Proceedings of International Test Conference*, IEEE, August, 1989.
13. T. Larrabee, "A Framework for evaluating Test Pattern Generation Strategies", *Proceedings of International Conference on Computer Design*, IEEE, October 1989.
14. J. A. Abraham and W.K. Fuchs, "Fault and Error Models for VLSI", May, 1986.
15. K.P. Parker, "Adaptive Random Test Generation", *J. Des. Auto. And Fault Tolerant Computing*, Vol. 1, October 1976, pp. 62-83.
16. H. Fujiwara, "Logic Testing and Design for Testability", MIT Press, Cambridge, Massachussetts, 1985.
17. H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms", *IEEE Trans. Comp.*, Vol. C-32, December, 1983, pp. 1137-1144.
18. F. Brglez and H. Fujiwara, A neutral netlist of 10 Combinational Benchmark Circuits and a target Translator in Fortran, *Proceedings of International Symposium on Circuits and Systems*, IEEE, June, 1985.
19. M. Davis and H.Putman, "A Computer Procedure for Quantification Theory", *Journal of the Association of Computing Machinery*, 7, 1960, pp. 201 – 215.

21. B. Aspwall, M. Plass and R. Tarjan, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas", *Information Processing Letters*, Vol. 8, 1979, pp. 121-123.

22. V.S.Sundaram, "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience v.2 n.4*, December 1990, pp. 315 - 339.

23. J.C.Browne, J.Dongarra, S.I.Hyder, K. Moore and P. Newton, "Visual Programming and Debugging for Parallel Computing", *IEEE Parallel and Distributed Technology: Systems and Technology*, Vol. 3, March 1995, pp. 75 - 83.

24. Tracy Larrabee, "Efficient Generation of Test Patterns using Boolean Satisfiability", *Western Research Laboratory Research Report*, February 1990.

25. M.A. Bruer and A.D. Friedman, "Diagnosis and Reliable Design of Digital Systems", *Computer Science Press Inc.*, 1976.

26. Prabhakar Goel, "An Implicit Enumeration algorithm to generate tests for combinational logic circuits", *IEEE Trans. Comput.*, Vol. C-30, March 1981, pp. 215 – 222.

27. T. Larrabee and Y.Tsuji, "Evidence for a Satisfiability Threshold for Random 3CNF Formulas", *Proceedings of the AAAI Symposium on AI and NP-Hard Problems*, 1992.

28. L.H. Goldstein, "Controllability/Observability Analysis of Digital Circuits", IEEE Trans. On Circuits and Systems, Vol. CAS-26, No. 9, September 1979, pp. 685-693.

29. Prabhakar Goel, "Test Generation Costs Analysis and Projections", Proc. 17th Design Automation Conference, June 1980, pp. 77-84.

30. 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.

31. J.J.Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of two state Neurons", May 1984.

30. 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.

31. J.J.Hopfield, "Neurons with Graded Response have Collective Computational Properties like those of two state Neurons", May 1984.

32. Al Geist, A. Beguelin, J.Dongarra, R. Manchek, V.S. Sundaram, "PVM 3 User's Guide and Reference Manual", *Oak Ridge National Laboratory*, September 1993.

33. M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W.H. Freeman and Company*, San Francisco, 1979.

34. S.T. Chakradhar, V.D. Agrwal, M.L. Bushnell, "towards massively-parallel automatic test generation", IEEE Transactions on Computer-Aided Design, Vol. 9, No. 9, September 1990, 2235 – 2258.

35. R.H. Klenke, R.D. Williams, J.H. Aylor, "Parallel-Processing Techniques for Automatic Test Pattern Generation", IEEE Computer, Vol. 25,No. 1, January 1992, pp. 71 – 84.

36. J.P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method", IBM, *Journal of Res. And Develop.*, Vol. 10, 1966, pp. 278 – 291.

37. M.H. Schulz, E. Trischler, T.M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System", *IEEE Trans. on Computer-Aided Design*, Vol. 7, January 1988, pp. 126 – 137.

38. T. Kirkland and M.R. Mercer, "A Topological Search algorithm for ATPG", *Proc. of 24th ACM/IEEE Design Automation Conf.*, June 1987, pp. 502 – 508.

39. Y. Takamatsu and K. Kinoshita, "CONT: Concurrent Test Generation System", *IEEE Trans. On Computer-Aided Design*, Vol. 8, No. 9, September 1989, pp. 966 – 972.

40. K. Hwang and F.A. Briggs, "Computer Archtecture and Parallel Processing", *McGraw-Hill*, New York, 1984.