uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Sujoy Mukherjee

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Space Compactor Design in BIST with Maximal Compaction Ratio using Concepts of Strong and Weak Compatibilities of Response Data

TITRE DE LA THÈSE / TITLE OF THESIS

Professor Sunil R. Das

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

V. Groza

T. Kwasniewski

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Space Compactor Design in BIST with Maximal Compaction Ratio Using Concepts of Strong and Weak Compatibilities of Response Data

By

Sujoy Mukherjee

A Thesis Submitted to the School Of Graduate Studies and Research

in Partial Fulfillment of the Requirements for the degree of

Master Of Applied Sciences

In Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School Of Information Technology and Engineering

(Electrical and Computer Engineering)

Faculty of Engineering

University of Ottawa

April, 2007

Your file Votre référence
ISBN: 978-0-494-34094-3
Our file Notre référence
ISBN: 978-0-494-34094-3

To my parents

ABSTRACT

Built-in self-testing (BIST) is a design process that provides the capability of solving many of the problems otherwise encountered in testing today's complex system on a chip circuits. With the increasing number of I/O counts in system on a Chip circuits it is almost impossible to think of BIST with out a space compactor. The fundamental problem with space compactors is error masking or aliasing, which occurs when the signatures from faulty output responses map into fault free signatures resulting reduced test quality. This theses develops a new approach for implementing space-efficient BIST support hardware, extending the well known concepts of conventional switching theory, and of compatibility relation as employed in the minimization of incomplete sequential machines, using Paull-Unger algorithm of finding all the maximal compatibility classes in the design. The theses utilizes mathematical selection criteria of merger of an optimal number of output lines of the MUT to decide on the logic for zero-aliasing, achieving maximal compaction in the design, as is evident from the simulation experiments conducted on ISCAS 85 and ISCAS 89 benchmark circuits.

ACKNOWLEDGEMENT

This thesis has become a reality due to the generous help and support of many individuals. It is my pleasure to acknowledge and thank them for their contributions, which are embodied in concept, thoughts and goals of this dissertation.

I sincerely thank my advisor, Dr. Sunil R. Das, Professor, School of Information Technology and Engineering, University Of Ottawa, whose technical advice, guidance and encouragement made my post graduate studies a meaningful learning experience.

Finally, I owe special thanks to my family for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ATE: automatic test equipment.

ATG: automatic test generation.

BIST: built-in self-testing

BIT: built-in test

CRC: cyclic redundancy checking

CUT: circuit under test.

DFT: design for testability

DSC: dynamic space compression

DTC/MTC: double/multiple transition counting

HSC: hybrid space compactor

LFSR: linear feedback shift register

LSSD: level sensitive scan design

MDSC: modified dynamic space compression

MISR: multiple-input signature register

MPT: multiplexed parity tree

PODEM: path oriented decision making

PSC: programmable space compaction

QFC: quadratic function compaction

RCU: response compaction unit

TPG: test pattern generator

Chapter 1

Introduction

Recent advancement in CMOS (Complimentary Metal Oxide Semiconductor) technology has enabled the fabrication of high density Integrated circuits which has the capability to operate at a much higher speed than ever before. With the growing size and complexity of Very Large Scale Integration (VLSI) circuits, it has become increasingly challenging to deliver a circuit that is free from any defects or flaws. Test is an essential step that helps to ensure that the final product in the form of a fabricated Integrated Circuit (IC) meets high quality requirements. Test addresses issues related to the screening of fabricated devices for manufacturing related defects. This thesis makes a primary contribution to the area of integrated circuit testing.

1.1 Role of Testing in VLSI Design

Recent advancement in Electronic design automation and the application of design for testability concepts help to produce an error free design. Even though a circuit is designed as error-free, manufactured circuits may not function correctly. Since the manufacturing process is not perfect, some defects such as short-circuits, open-circuits, open interconnections, pin shorts etc. might be introduced by the manufacturing process itself. Williams stated [68] that the cost of detecting a faulty component increases ten times at each step between pre-packaging component test and system warranty repair (see Figure 1.1). It is

important to identify a faulty component as early in the manufacturing process as possible. Therefore, testing has become a very important aspect of any VLSI manufacturing system.

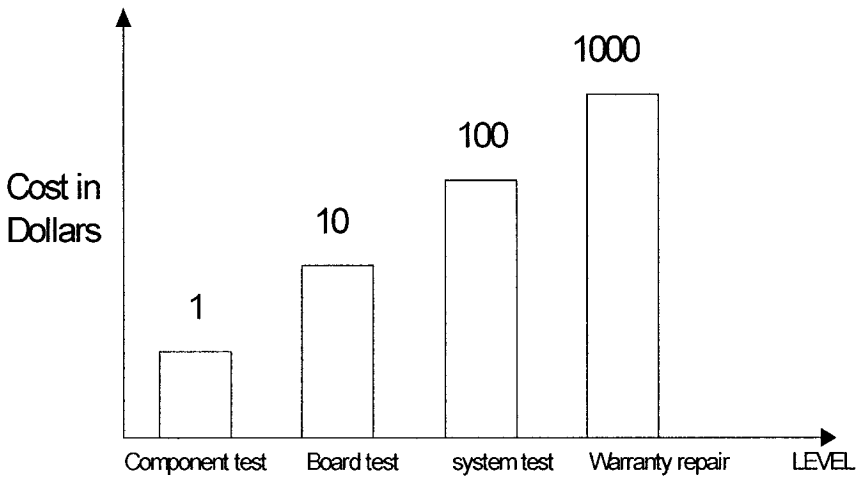


Figure 1.1. Cost of Failure [68]

Manufacturing defects tend to alter the circuit behavior. Hence the testing process involves verification of circuit behavior by applying a set appropriate test patterns and comparing the device under test response with a set of known good response.

1.2. Test Vector Generation And Fault Modeling

Test Vector Generation is an important part of test. Miczo [59] stated that the early test vector generation approaches involved either the creation of all possible combinations for device inputs or the creation of stimuli targeted to verify certain functional features of the device. Hence a circuit with n inputs would require 2^n test vectors. This is effective if n is small and the device does not

contain memory elements. Modern VLSI circuits are large and complex; they have outgrown both of these restrictions. An exhaustive test pattern set for a modern VLSI circuit could result into a hundred million or more vectors. Each circuit is tested for a given set of test patterns using Automatic Test Equipment (ATE). This equipment is very expensive and contains a limited amount of memory to store test patterns. The length of time that each circuit requires for testing is also an important factor in determining the overall test cost. In order to reduce the test application cost (which primarily comprises of the tester time and the cost of test equipment), it is important to keep the number of test vectors to a minimum.

Eldred [67] suggested an efficient test generation approach that targets hardware faults rather than the function. In this case test patterns are created for specific faults. Logical fault models can be used to represent common physical faults. Logical faults represent the effect of physical faults on the behaviour of the digital circuit. Then, input stimuli are created to distinguish between the fault-free and faulty circuits. Test pattern generation based on the logical fault model assumes the presence of a single fault in the circuit at a given time. Test patterns derived under the single-fault assumption are generally considered useful for detecting multiple faults because a test derived for an individual single fault can detect multiple faults containing that single fault as a component. There are, however, specific multiple faults where the components can mask each other and detection by a single fault test is not guaranteed [50].

Among many different types of logical fault models, the "stuck at" model is prominently used for test vector generation and evaluation. When the faulty logic level is 0, the fault is called stuck-at-zero (s-a-0) and when the faulty logic level is 1, the fault is called stuck-at-one (s-a-1). Under the single fault assumption, only one fault is applied at a time when a test set is being created. This approach is called "Single Stuck Fault (SSF)" modeling. The concepts of stuck at fault modeling will be discussed in detail in a later chapter.

Popular test generation approaches based on the SSF model can be classified into random and deterministic methods. Random test generation depends only on the number of circuit inputs and works without taking into account the topology of the circuit to be tested. Deterministic test generation produces tests by processing a model of the circuit. Deterministic test generation can produce shorter tests of a higher quality in comparison to the random test generation approach. Deterministic test generation can be fault-oriented or fault independent. Tests are generated for specific faults from a given fault universe in fault-oriented deterministic test generation. The goal of the fault-independent deterministic test generation approach is to derive a set of tests that detect a large set of faults in the fault universe without targeting individual faults. The test generation process can be manual or automatic.

1.3 Fault Simulation And Test Evaluation

Fault simulation normally refers to obtaining the response of a system from a model under faulty condition and comparing the response with known good response or signature. If the response does not match, the fault is considered

detected by the given set of test patterns. Thus, fault simulation can be used to evaluate or grade test patterns and testability analysis of a design prior to production.

Fault coverage is a test evaluation metric that indicates the effectiveness of a test. It is defined as the ratio of the number of detected faults to the total number of simulated faults. The higher the fault coverage the better is the quality of that particular test.

1.4 Design-For-Test

The prime objective of any testing system is to achieve high fault coverage (95%-100%) with the application of minimum number of test vectors. The higher the number of test vectors, the higher the cost associated with test. The high quality test offers product reliability at a minimum test application cost. However, there is a cost associated with generating such good quality tests. The test generation cost depends on the complexity of the circuit. Methods of reducing the complexity of a circuit for testing purposes are referred to as design-for-testability techniques. Circuits that are highly testable offer easier test access (controllability from primary inputs and observable at primary outputs) for internal nodes. Structural analysis techniques are often used to compute numerical measures for the testability of the internal nodes of circuits. Designs with insufficient testability measures rarely attain high fault coverage even with a large number of test patterns. The lack of good coverage results in the poor screening of manufacturing defects. As a result, the compromised quality of the manufactured ASICs reduces the reliability of the product that uses those

components. Such products have higher failure rates, increasing overall costs. Circuits that are not test-friendly require architectural and/or structural design changes to improve testability. These changes may be guided by the testability measure, which can often point to areas of poor testability in the circuit.

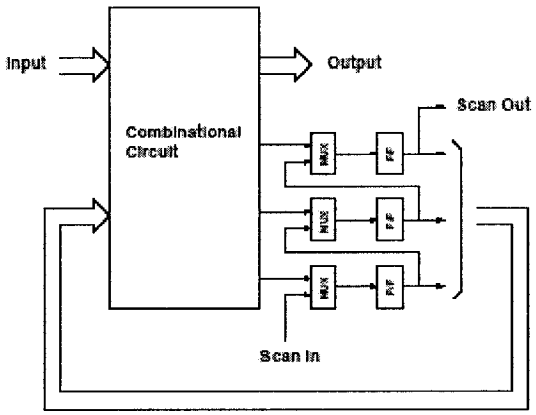


Figure 1.2. Block diagram of a scan design

To name a popular DFT technique used is Scan design [56]. In this Technique memory elements are separated from combinational circuits during testing. Usually filp-flops are chained together into a shift register during the selected testing mode and the circuit is partitioned into a combinational blocks so that the test generation can be executed easily. The tests can be scanned in and the test responses scanned out for verification. Although reduced test generation costs and improved fault coverage are the advantages of this technique it has some drawbacks too. This technique adds up more area overhead in design and increases in test application time.

1.5 Built In Self-Test

Built-in self-testing (BIST) is a design methodology that has the capability of solving most of the problems encountered in testing digital systems. It combines the concepts of both built-in test (BIT) and self-test (ST) in one, termed built-in self-test (BIST). In BIST, test generation, test application, and response verification procedures are accomplished through built-in hardware [54], [5], [8], [32]. It allows different parts of a chip to be tested in parallel, reducing the required testing time. It also simplifies the need for external test equipment. As the cost of verification testing is becoming the major component of the manufacturing cost of a new product, BIST tends to reduce manufacturing test and maintenance costs and also improves diagnosis.

Several companies such as Motorola, AT&T, IBM, and Intel have incorporated BIST in many of their products [57], [8]. AT&T, for example, has incorporated BIST into more than 200 of their chips. The three large PLAs and microcode ROM in the Intel 80386 microprocessor were built-in self-tested [8]. The general-purpose microprocessor chip, Alpha AXP21164, and Motorola microprocessor 68020, were also tested using BIST techniques [57], [8].

1.6 Test Technology Road Map

With the rapid advancement in process technology, gate count or logic density of present Integrated Circuit has been increased. With the increasing complexity, test continues to be a major expense in the IC development and manufacturing chain, with up to 35% of nonrecurring engineering (NRE) costs attributed to test

development and debug, and with ATE cost per transistor expected to remain flat. Changing processes and design methods are pushing testability beyond economic limits. Rapid improvements must be made to improve overall testability and test economics. Below here are some of the predictions made by international technology road map for semiconductors [58] [70]:

- *Built-in-self-test* will be needed as chip performance begins to outpace tester timing accuracy, and as overall test data volume exceeds ATE and chip boundary capability. BIST needs to be made usable in short-design cycle environments by novice designers. Logic BIST methods must be developed that provide high coverage for all required fault types. Power management and sequencing of tests must have to be addressed by BIST tools. The concept of BIST needs to be extended to include additional on-chip parametric measurements currently performed by ATE.
- *Divide-and-conquer* techniques such as scan wrappers around hierarchical blocks and localized BIST will be needed to manage test complexity, as the task of developing adequate tests will become impossible for flat designs.
- *Reuse of cores* requires the encapsulation and reuse of test as well. Standardized interfacing and access methods for core testing are needed, as are composition methods to assemble complete chip tests for chips with multiple cores, including analog cores. Methods to test the interconnect between cores must also be developed, and signal integrity standards developed to assure that embedded cores function as expected

within the larger chip, even in the presence of noisy power, grounding, and interconnect. In addition, standardized test data formats, such as standard test interface language (STIL), will be needed to ensure portability.

- New fault types must be identified, and test methods developed for them, as net timing and signal integrity problems increase dramatically and introduce new modes of chip failure.
- *Design for testability* must be tightly integrated into all steps of the overall design flow, including synthesis, validation, and physical design, and include automatic test generation with high coverage for all relevant fault types (such as stuck-at faults, timing, bridging, signal integrity). These DFT techniques must apply to complex chips with multiple time domains and must include methods to simplify probing requirements at wafer test.
- *Signal integrity* and electromagnetic (EM) phenomena will become an increasingly important test issue as chips and test equipment become complex. New fault models (including soft error models) that incorporate the effects of EM fields must be developed. Relationships between design constraints and manufacturability and testability must be developed for different design domains. Test generators must be sensitive to signal integrity issues.
- *Timing tests* are impacted by interconnect delays, slow synthesis system drivers, increased frequencies, multiple timing domains and clock skew. Automatic test generation will be necessary to accommodate the large

number of near-critical paths, and BIST systems will have to guarantee high coverage of timing faults.

- *Quiescent current (IDDQ) testing* requires extensions for manufacturing test as background currents increase, although it will remain a valuable technique for failure analysis. Single threshold IDDQ testing is no longer viable for many CMOS processes, and other approaches will be needed to ensure device reliability.
- *Hardware/software co-design* will provide opportunities for system software to be used as an aid for test. All high-level design methodologies must be sensitive to the fact that they may target unknown libraries and processes with special test problems and unknown fault statistics.
- *Analog/RF systems* present continuing challenges for test, primarily because they require test to specifications rather than structural test. As a result, there is inherent difficulty in presenting any simplification to create a meaningful fault model. Embedded analog blocks will likely require the development of BIST approaches.
- *Yield improvement and failure analysis* tools and methods will be needed to provide for rapid yield learning with very complex chips containing embedded cores (on the actual chips, not simplified test devices), and for highly automated failure analysis where faults can no longer be visually detected. Design and synthesis for diagnosis must be included if short failure analysis and yield improvement cycles are to be realized.

Other issues include the insertion of testability circuitry in systems limiting the Operating speed of the system. Figure 1.3 and 1.4 shows the increasing trend in test technology as projected by ITRS 2005 [70].

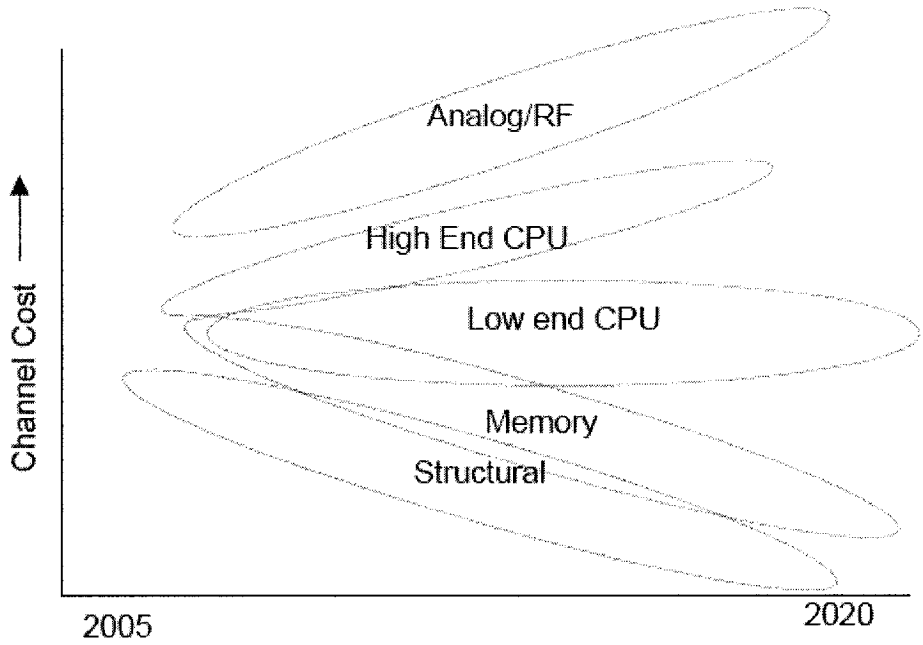


Figure 1.3. Channel cost range for product segments [ITRS'2005]]

Power consumption during test must be carefully considered so test programs do not put the system into modes that consume excessive power. Application of test programs must also not cause excessive noise with the possibility of soft errors.

1.7 Organization of Thesis

Chapter 1 provides brief information about fault tolerance in system design techniques and applications. This chapter also unveils the motivation and objective of the dissertation.

Chapter 2 reviews literature on system on a chip testing. Includes BIST techniques, review of space compactors etc.

Chapter 3 Fault Simulation and related concepts have been reviewed. Fault simulation approaches and approach used in this thesis have been discussed.

Chapter 4 Theory for Compactor design utilizing the concept of compatibility relation has been formulated. A detailed treatment of the proposed algorithm has been presented.

Chapter 5 we present our experimental results on simulation on ISCAS 85 and ISCAS 89 benchmark circuits. A Comparative study has been done with the parity tree space compactor to demonstrate the suitability of proposed algorithm.

Chapter 6 Concluding remarks on present work have been presented.

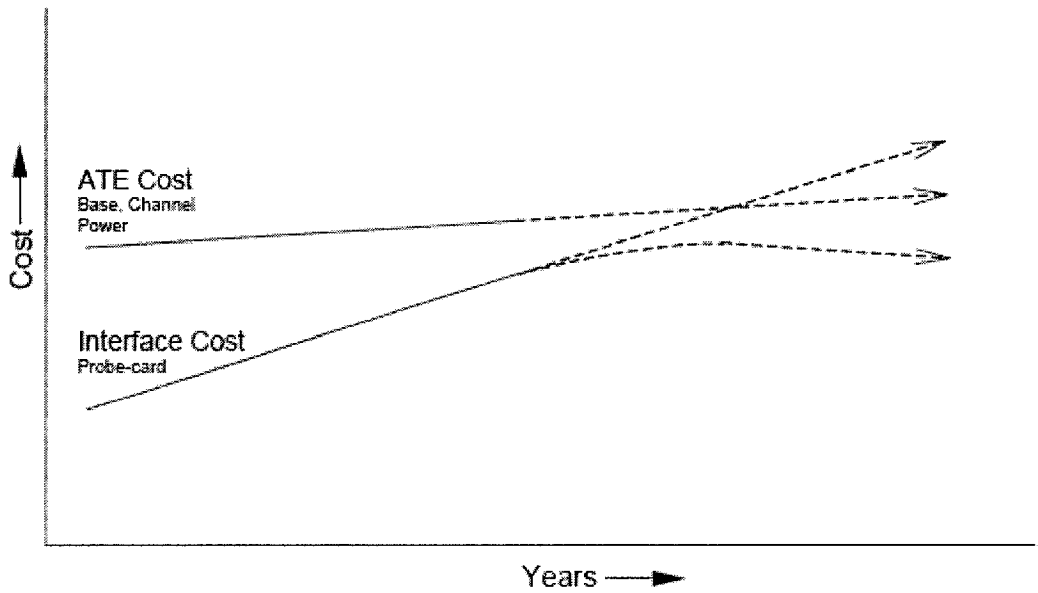


Figure 1.4 Test Cell Cost / Unit vs. Interface Cost Trend [ITRS'2005]]

Chapter 2

Literature Review

In this chapter, we review BIST (Built In Self Test) as an effective and efficient test mechanism. One of the important aspects of any BIST technique is response data compaction. In this Chapter we review response compaction techniques and problems associated with response compaction. As evident, a reliable and practical compression technique should have the following characteristics:

a) Realization of data compression hardware must have to be easy and as part of CUT (Circuit Under Test)'s BIST logic.

b) Data compaction circuit should not introduce signal delays affecting either the normal behavior or the test execution time of the CUT.

c) Moreover, the length of the test signature should be short to reduce the amount of storage required for fault free signatures.

d) The faulty and fault free signatures should be different; otherwise, error masking or aliasing will result introducing information loss that penalizes the test quality of BIST.

2.1 Built In Self Test- An Introduction

Built-in self-testing (BIST) is a design approach that can significantly improve the testability of digital circuits and save testing time. It combines concepts of both built-in test (BIT) and self-test (ST) in one termed as built-in self-test (BIST). In BIST, test generation, test application, and response verification are all done

through built-in hardware, which allows different parts of a chip to be tested in parallel, reducing the required testing time, besides eliminating the necessity for external test equipments. A typical BIST environment, as shown in Fig. 2.1, uses a test pattern generator (TPG) that sends its outputs to a circuit under test (CUT), and the resulting output streams from the CUT are fed into a response data analyzer. A fault is detected if the CUT response is shown to be different from that of the fault-free circuit. The test data analyzer is comprised of a response compaction unit (RCU), storage for the fault-free responses of the CUT and a comparator.

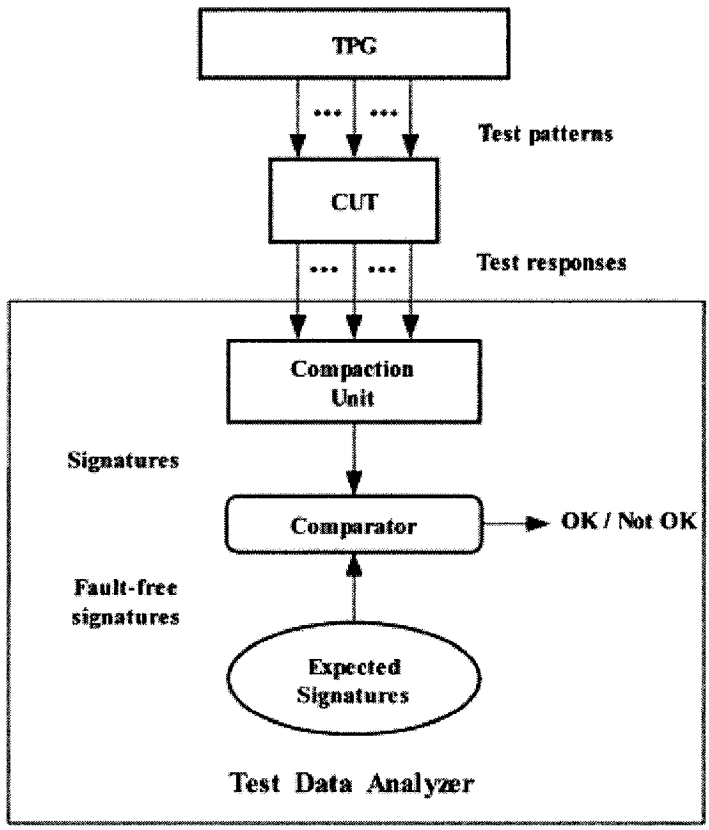


Figure 2.1 A Typical BIST Environment

In order to reduce the amount of data represented by the fault-free and the faulty CUT responses, data compression is used to create signatures from the CUT and its corresponding fault-free circuit. BIST techniques use pseudorandom, pseudo exhaustive and exhaustive test patterns, or even sometimes on-chip storing of reduced test sets. The standard response compaction unit is comprised of a space compression unit and a time compression unit. In general, S responses coming out of a CUT are first fed into a space compressor, providing output streams such that $t << S$. Most often, test responses are compressed into only one sequence (t ==1). Space compression generates a solution to the problem of achieving high test quality for built-in self-testing of complex digital circuits without the requirement of monitoring a large number of internal test points, by merging test responses coming from the internal test points into a single bit stream, reducing in the process the test time and the resulting area overhead. This single bit stream of length r is eventually fed into a time compressor and a shorter sequence of length q, $q < r$ is obtained at the output.

Below here is a list of advantages of BIST [59]:

1. A major argument for the use of BIST is the reduced dependence on expensive testers. Modern day testers represent a major investment. To the extent that this investment can be reduced or eliminated, BIST grows in attractiveness as an alternative approach to test. It is not even necessary to completely eliminate testers from the manufacturing flow to economically justify BIST. If the duration

of the test can be reduced by generating stimuli and by computing a response on chip, it becomes possible to achieve the same throughput with fewer and possibly less expensive testers.

2. One of the problems associated with the testing of ICs is the interface between the tester and the IC. Cables, contact pins and probe cards all require careful attention because of the capacitance, resistance and Inductance introduced by these devices, as well as the risk of failure to make contact with the pins of device under test (DUT), possibly resulting in false rejects. These interface devices not only represent possible technical problems, they can also represent a significant incremental cost. BIST can eliminate or significantly reduce these costs.
3. Modern complex System on a Chip consists of RAM, ROM, register banks and scratch pads etc. These are often quite difficult to access from the I/O pins of an IC. BIST can directly access these memories and BIST controller can often be shared by some or all of the embedded memories.
4. Test data generation and management can be very costly. It includes the cost of creating, storing and otherwise managing test patterns, response data, and any diagnostic data needed to assist in the diagnosis of defects. BIST can substantially reduce the data management problem.

Another advantage of BIST is that many thousands of pseudo –random vectors can be applied in BIST mode in the time that it takes to load a scan path a few hundred times. A further benefit of BIST is the ability to run test at speed, which improves the likely hood of detecting delay errors.

2.2 Response Compaction

As the name BIST (Built In Self Test) suggests that the test circuit must have to be built in or embedded with in the DUT (Design Under Test). Hence the test circuit must have to be simple with less area overhead. With the increasing number of I/O counts in present day Integrated Circuits, it has become much more challenging to produce test circuit with less area overhead. Hence it brings an important building block with in BIST architecture, commonly known as response compactor. A response compactor performs the task of response compaction, their by reducing the volume of information that need to be analyzed or processed to produce a signature of a fault free circuit and faulty circuit. The testability of the circuit is being determined by comparing the signatures under faulty condition and fault free condition. Hence the efficiency of response compactor justifies the economics of a BIST system.

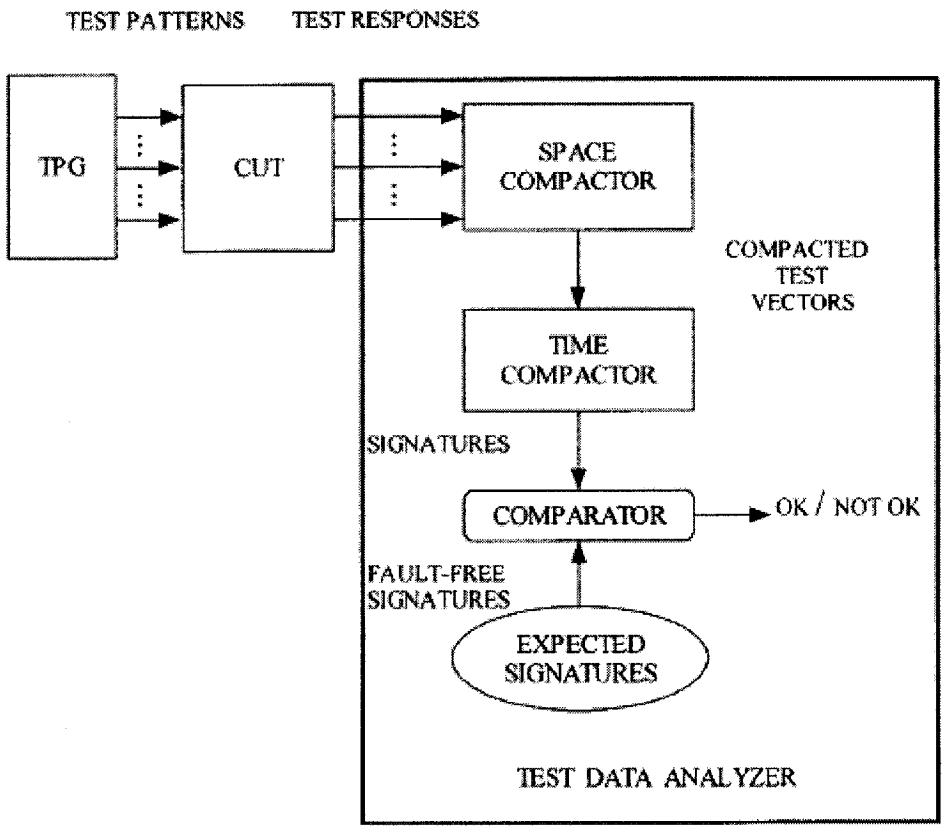


Figure 2.2 Response Compaction Scheme

As shown in figure 2.2 response compaction can be divided into two categories. One is space compaction and the other is time compaction. In case of space compaction multiple output lines are compressed to fewer number of output lines. Time compressors are used to generate a shorter signature to minimize storage requirements further down.

2.3 Space Compaction Techniques

The basic idea of space compression is to produce a limited number of signatures (usually a single signature) for multiple output circuits instead of producing an individual sequence for each output (see figure 2.3). This results in

achieving high quality BIST of complex chips without the necessity of monitoring a large number of internal test points. It also reduces testing time and area overhead by grouping test sequences coming from a CUT into a single stream of bits. Sequence coming out of the space compressor is stored in a time compressor as a signature.

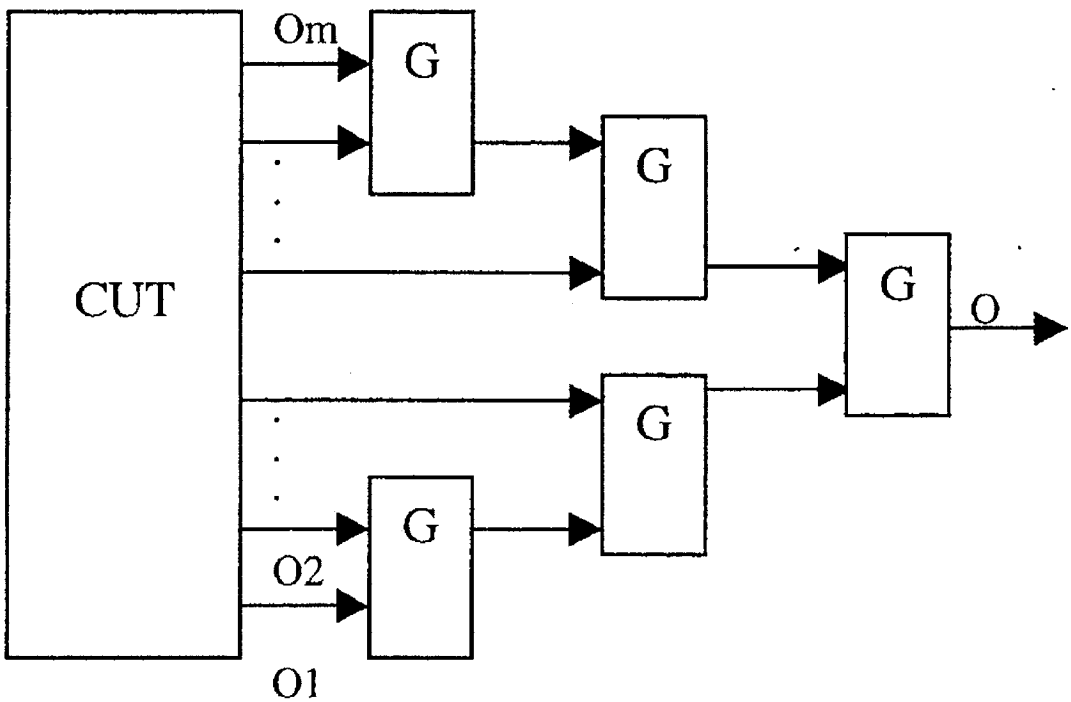


Figure 2.3 Space Compaction using Logic reduction Network

Space compression thus reduces the number of pins monitored by the tester as well as the memory space required for reference signature, with loss of some useful information generally leading to a reduction in fault coverage.

A brief discussion of various space compaction techniques as proposed in the literature or used in industries like parity tree compaction, hybrid space

compression, dynamic space compression, modified dynamic space compression, modified hybrid space compression, quadratic functions compaction, programmable space compaction, multiplexed parity tree compaction in the context of BIST is provided in the following sub sections.

2.3.1 Parity Tree Compaction

The parity tree space compaction circuits consists of Exclusive-OR (XOR) or Exclusive-NOR (XNOR) gates only. The parity tree circuit realize functions of the form z= z1 ⊕z2⊕...zk. The parity tree space compactor propagates all errors that appear on an odd number of its inputs z1, z2, zk. However aliasing occurs if error appears on an even number of its inputs. If a parity tree is used for space compaction, pseudorandom or compact test patterns can be used to detect most of the single stuck at faults [60]. A parity tree space compaction scheme is shown in Figure 2.4

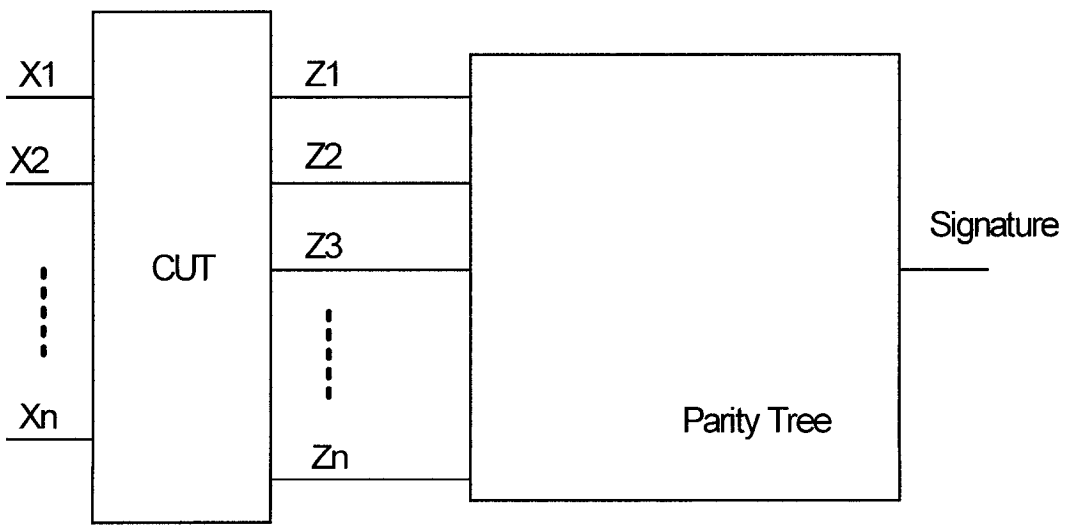


Figure 2.4 A Parity Tree Compaction

2.3.2 Hybrid Space Compactors (HSC)

Li and Robinson have proposed a space compression method called hybrid space compression (HSC) which uses two-input AND,OR and XOR gates to construct a compression tree to compress the multiple outputs of the CUT into a single line, rather than using XOR/XNOR gates as an output compression tool [16]. The Compaction tree is constructed based on the detectable error probability estimate $\in$ defined as:

$$\in = S_1 \frac{R_1}{2L} + S_2 \frac{R_2}{L}$$

Where

S1 is the probability of single error felt at the output of CUT.

S2 is the probability of double error felt at the output of CUT.

R1 is the number of single line errors detected at the output of that logic gate.

R2 is the number of double line errors detected at the output of that logic gate.

L is the input sequence length.

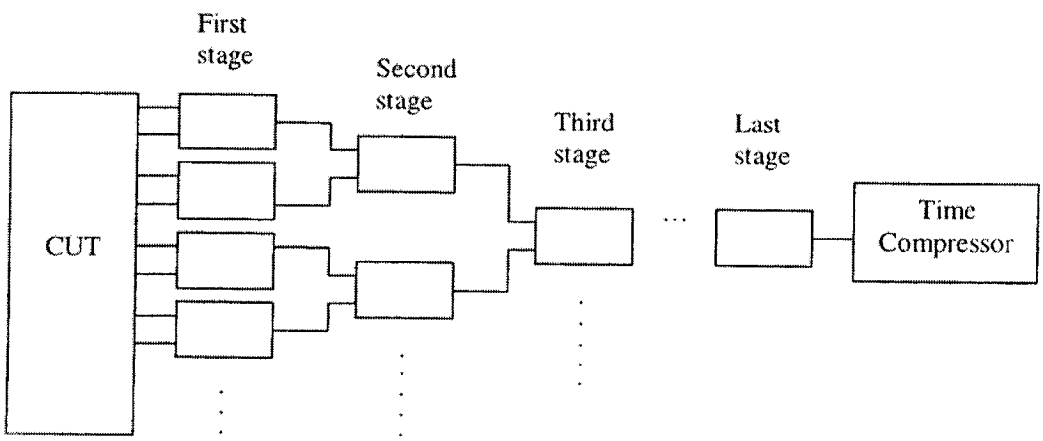


Figure 2.5 Realization of Hybrid Space Compaction

The circuit realization of HSC is shown in figure 2.5. Space Compression in HSC is achieved by deriving $\in$ for AND, OR and XOR gates with the maximum $\in$ being selected to merge the candidate outputs. The process is repeated stage by stage until the outputs have been merged into a single line.

2 .3.3 Dynamic Space Compression

Jone and Das subsequently proposed a modified method called dynamic space compression, where the probabilities S1 and S2 are dynamically estimated based on the structure of the CUT during the entire derivation process of the space compressor, instead of a fixed value used in case of HSC. Experimental results show that the information loss in general is between 0% and 12.7%. A theory to predict the performance degradation of general space compression methods was also developed by Jone and Das [18].

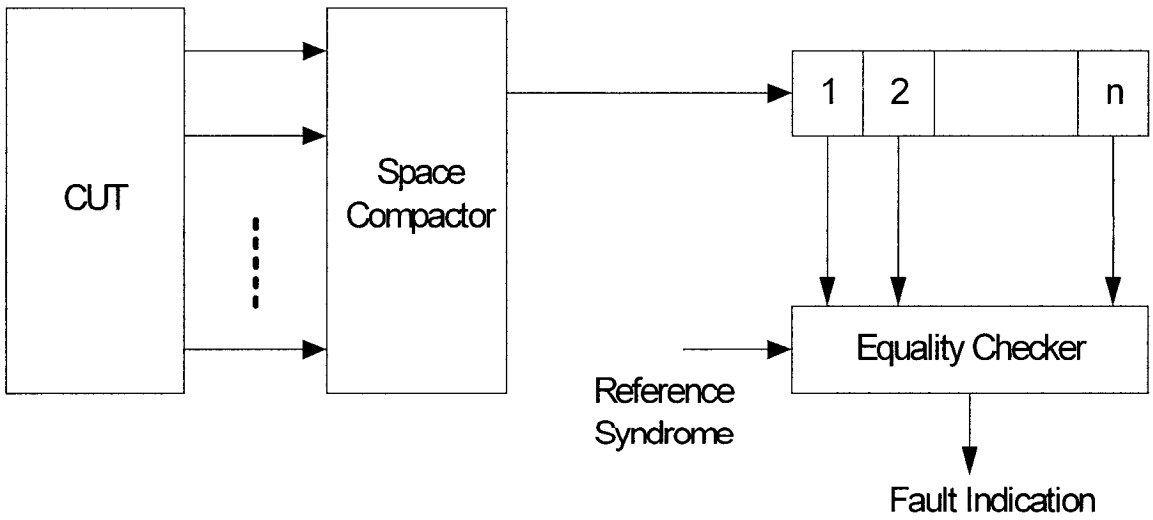


Figure 2.6 Syndrome Counter Used as a Time compressor

2.3.4 Modified Dynamic Space Compression

A refinement of DSC technique called modified dynamic space compression (MDSC) was proposed in [26]. For a CUT, a compaction tree is constructed based on its structure. In this technique, the detectable error probability estimates are calculated by using the Boolean difference method. The output data modification is used to minimize the number of faulty output data patterns, which have the same compressed form as the fault free patterns. The compressed outputs are fed into a syndrome counter to derive the signature of the circuit as shown in figure 2.6. Experimental results indicate that the loss of information is in the range of 0% to 10 %. However the Boolean difference method used to derive the number of detectable faults in a circuit may require high storage and thus the method may not be suitable for circuits with a very high density of gates because of involved computations.

2.3.5 Modified Hybrid Space Compaction

Using the concepts of hamming distance and sequence weights, a new space compression technique was proposed by [5] in which two of the output sequences are merged by logic gates AND/NAND, OR/XOR, XOR/XNOR to construct a compaction tree to minimize the storage of CUT. Figure 2.7 illustrates an outline of the proposed scheme.

Some of the advantages inherent in this method are its simplicity and low hardware overhead. Besides, the method uses non-exhaustive test sets for the compaction tree design. Later Das and Barakat [5] modified this scheme

introducing generalized mergeability to merge an arbitrary number of output streams from CUT. The proposed method uses compact test sets and achieves reasonably high fault coverage (i.e. 100 %) with low overhead (2% -5%).

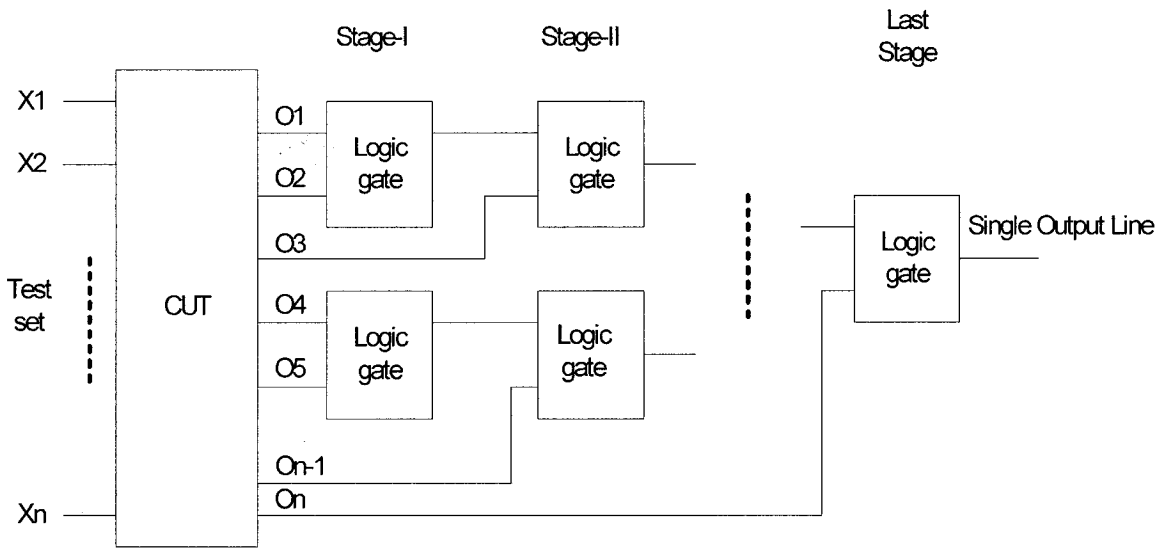


Figure 2.7 Modified Hybrid Space Compaction

2.3.6 Programmable Space Compaction

Programmable space compaction (PSC) is a technique for designing low cost space compactors with high fault coverage [61]. In PSC, a circuit specific space compactor is designed to increase the likelihood of error propagation. However, PSC as well does not guarantee zero aliasing. An optimal compaction circuit with minimized aliasing and lower cost can be found by exhaustively enumerating all the 2^{2k} k-input Boolean functions for a k-output CUT can be practically implemented using complex search techniques.

2.3.7 Multiplexed Parity Tree

A new approach termed multiplexed parity tree (MPT) based on parity tree circuits has been proposed in [8] to achieve high fault coverage. MPTs (shown in figure 2.8) perform zero aliasing space compaction of test responses by combining the error propagation properties of multiplexers and parity trees. Experimental results indicate that the associated hardware overhead is moderate (7% -10%) and very high fault coverage (95%) is obtained for faults in the CUT including the compaction circuit. A different approach to achieve zero aliasing was also proposed by the authors. The technique is based on constructing multiple output space compactors. It is shown that a two output circuit implementing two parity functions is adequate for aliasing free compaction.

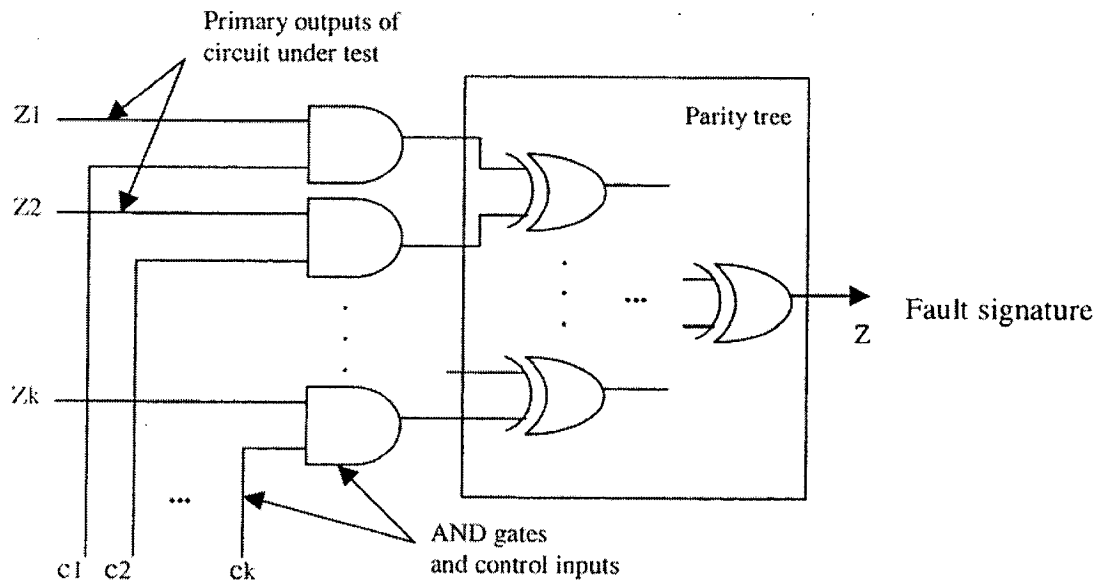


Figure 2.8 The MPT compaction scheme

2.4. Time Compaction Techniques

The Space compressor reduces the width of the test responses, while a time compressor is used to derive the signature in order to minimize the amount of storage needed to store information of the fault free signatures. Following subsections provide a brief overview of different time compaction techniques.

2.4.1 Ones Counting

Ones counting [25] use its signature the number of ones in the binary circuit response stream. The hardware that represents the compaction unit consists of a simple counter, and is independent of the CUT; it only depends on the nature of the test response. Signature values do not depend on the order in which the input test patterns are applied to the CUT.

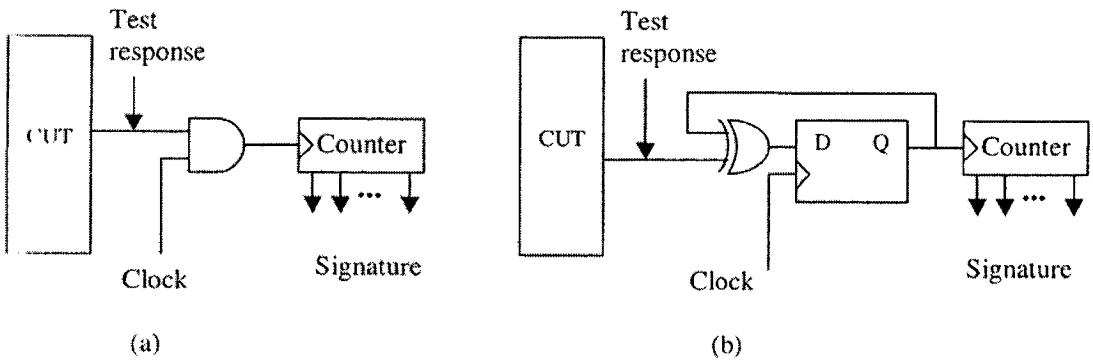


Figure 2.9 Ones Counting and Transition Counting Technique

2.4.2 Syndrome Counting

In syndrome counting [27], all the input patterns are exhaustively applied to an n-input combinational circuit. The syndrome, which is given by the normalized number of 1s in the response stream, is defined as, $S = K/2^n$ being the number of min terms of the function being implemented by the single-output CUT. Any switching function can be so realized that all its single stuck-line faults are syndrome-testable.

2.4.3 Transition Counting

Transition counting [28] counts the number of times the output bit stream changes from 1 to 0 and vice versa. In transition counting, the signature length is less than or equal to $\lceil \log_2 r \rceil$, where r being the length of a response stream. The error masking probability takes high values when the signature value is close to r/2, and low values when it is close to 0 or r.

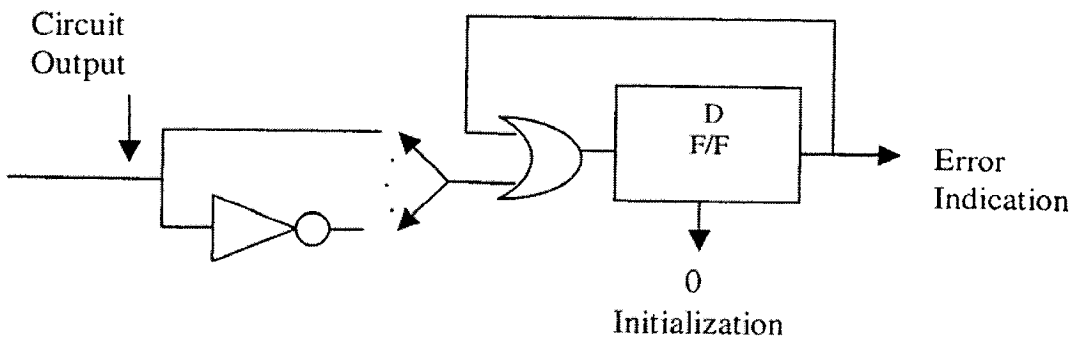


Figure 2.10 (a) A DTC tester for single output case

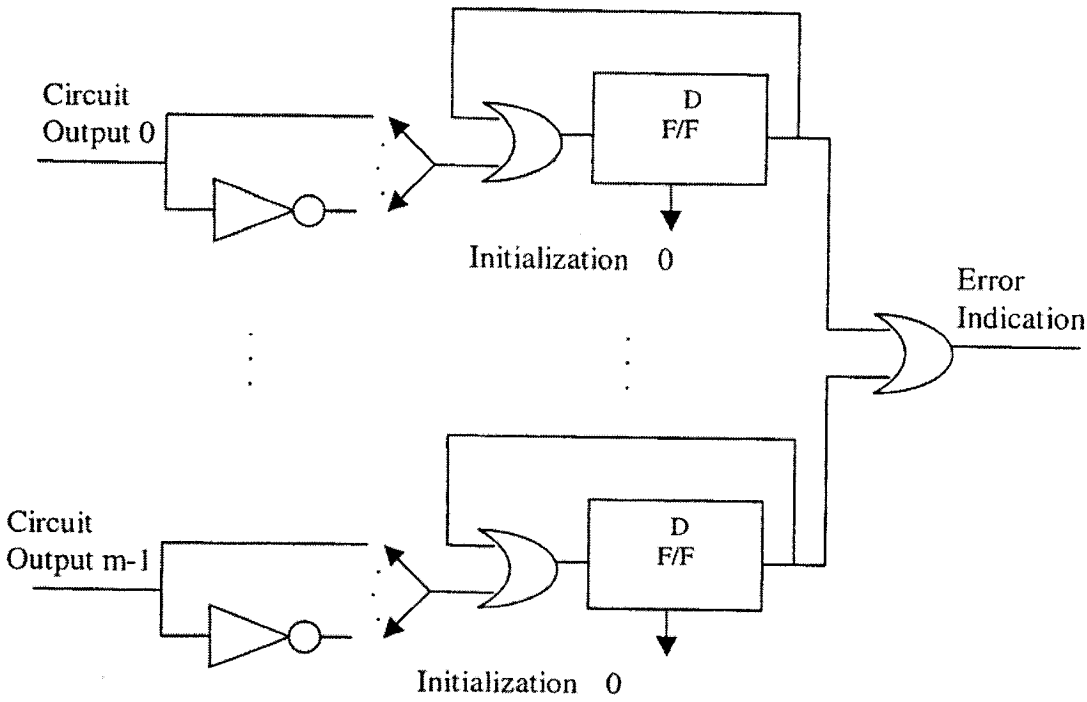


Figure 2.10 (b) A MTC Tester For Multiple Output Case

2.4.4 Walsh Spectral Analysis

In Walsh spectral analysis [62], switching functions are represented by their spectral coefficients that are compared to known correct coefficients values. In a sense, in this method, the truth table of the given switching function is basically verified. The process of collecting and comparing a subset of the complete set of Walsh functions is described as a mechanism for data compaction. The use of spectral coefficients ensures higher percentage of error coverage, whereas the resulting higher area overhead for generating them is deemed as a disadvantage [31].

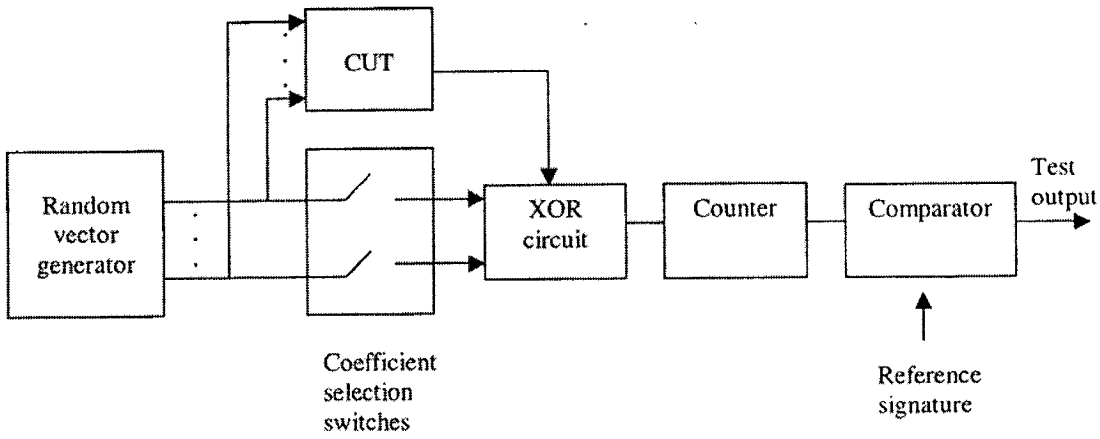


Figure 2.11 Implementing the RW compact testing scheme

2.4.5 Parity Checking

In parity checking [30], the response bit stream coming from a CUT reduces a multitude of output data to a signature of length 1 bit. The single bit signature has a value that equals the parity of the test response sequence. Parity checking detects all errors involving an odd number of bits, while faults that give rise to an even number of error bits are not detected. This method is relatively ineffective since a large number of the possible response bit streams from a faulty circuit will result in the same parity as that of the correct bit stream. All single stuck-line faults in fanout-free circuits are detected by the parity check technique.

2.4.6 Signature Analysis

Signature analysis [63] is probably the most popular time compaction technique currently available. It uses linear feedback shift registers (LFSRs) consisting of

flip-flops and exclusive-or gates. The signature analysis technique is based on the concept of cyclic redundancy checking (CRC). LFSRs are used for generating pseudorandom input test patterns, and for response compaction as well. The nature of the generated sequence patterns is determined by the LFSR's characteristic polynomial as defined by its interconnection structure. A test-input sequence is fed into the signature analyzer, which is divided by the characteristic polynomial of the signature analyzer's LFSR. The remainder R(x) obtained by dividing P(x) by G(x) a Galois field such that P(x) = G(x). Q(x) + R(x) represents the state of the LFSR, Q(x) being the corresponding quotient. In other words, R(x) represents the observed signature. Signature analysis involves comparing the observed signature R(x) to a known fault-free signature $R_0(X)$. An error is detected if these two signatures differ. Suppose that P(x) is the correct response and P'(x) = P(x) + E(x) is the faulty one, where E(x) is an error polynomial; it can be shown that aliasing occurs whenever E(x) is a multiple of G(x).

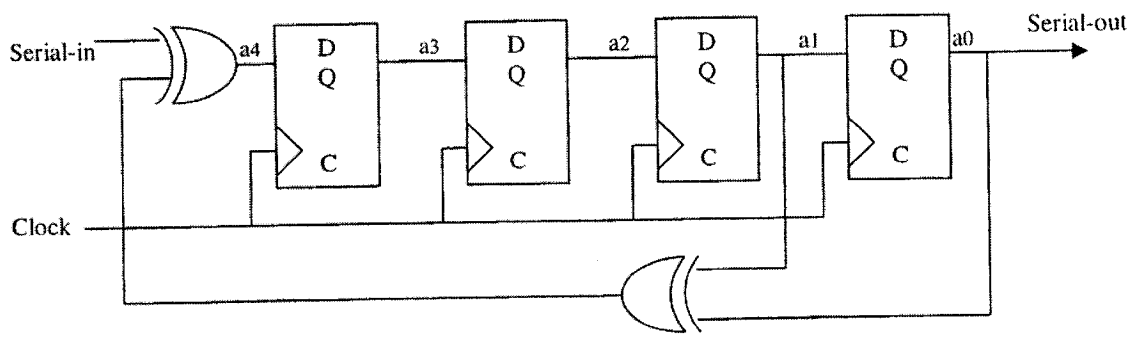


Figure 2.12 SISR logic diagram

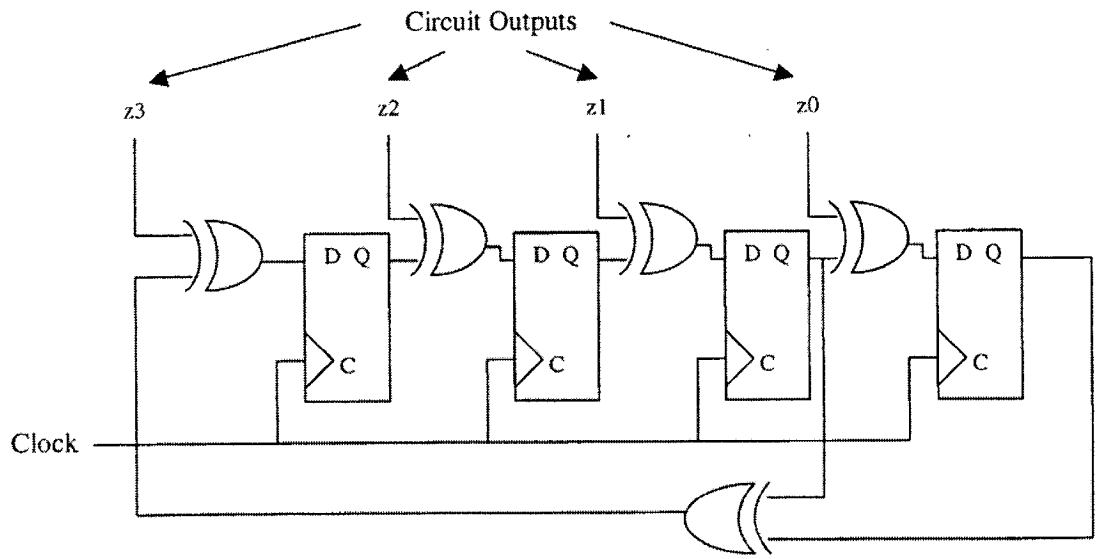


Figure 2.13 Multiple Input Signature register

Different methods for computing and reducing the aliasing probability in signature analysis have been proposed, *viz.* the signature analysis model proposed by Williams *et al.* [64] which uses Markov chains and derives an upper bound on the aliasing probability in terms of the test length and probability of an error occurring at the output of the CUT. Another approach to the computation of aliasing probability is presented in [66]. An error pattern in signature analysis causes aliasing, if and only if, it is a codeword in the cyclic code generated by the LFSR's characteristic polynomial. Unlike other methods, the fault coverage in signature analysis may be improved without changing the test set. This can be done by playing with the length of the LFSR, or by using a different characteristic polynomial As demonstrated in [65], for short test length, signature analysis detects all single-bit errors. However, there is no known theory that characterizes fault detection in signature analysis.

CHAPTER 3

Fault Simulation Environment

In this chapter, we review concepts associated with fault simulation. The detail treatment of a HDL based fault simulator will be presented. Development of an improved HDL based fault simulation approach will be discussed in detail. The main motivation behind this new approach is to explore strong compatible output pairs of the CUT (circuit under test). The strong compatible output pairs will be used in the design of space compactor.

3.1 Fault Simulation And Fault Simulator

Unlike simulation at design verification stage, Fault simulation relates to manufacturing test. The main objective here is to create a test program that uncovers defects and performance problems that occur during the manufacturing process [59]. The test program has to be through and efficient. If the manufacturing test program involves creation of redundant test stimuli, there is delay in migrating the test program to the tester. However stimuli that do not improve test thoroughness also add recurring costs at the tester. This is due to the fact that those stimuli results in higher cost of storage in test pattern and expected outputs as well as higher test application time.

Fault simulation normally refers to obtaining the response of a system from a model. For electronic systems, both hardware and software models are used. While hardware models or breadboards are the traditional way of verifying

designs, software models are becoming more popular due to their economy and accuracy derived largely from advances in the field of computing. A typical software simulation environment consists of the building blocks as shown in Fig. 3.1.

The test pattern generator (TPG) is the block responsible for generating stimulus vectors for the circuit under test (CUT). The fault injection block is responsible for injecting faults to the circuit under test (CUT). The injections of faults are based on the covered fault models by the fault simulator as a whole. It is not necessary for a fault simulation scheme to cover all types of available fault models. The objective and purpose of the fault simulator determine the supported fault models by a particular simulation approach. Another important building block of a fault simulator is the response analyzer. This block consists of a comparator unit and some buffer to store fault-free responses from the CUT. The response analyzer unit stores the fault-free responses of the CUT in buffer memory. These stored fault-free responses are then compared with faulty CUT responses in a comparator to determine detectability of certain faults. Though implementation of fault simulation environments may vary in practice, the concepts described herein is independent of implementations.

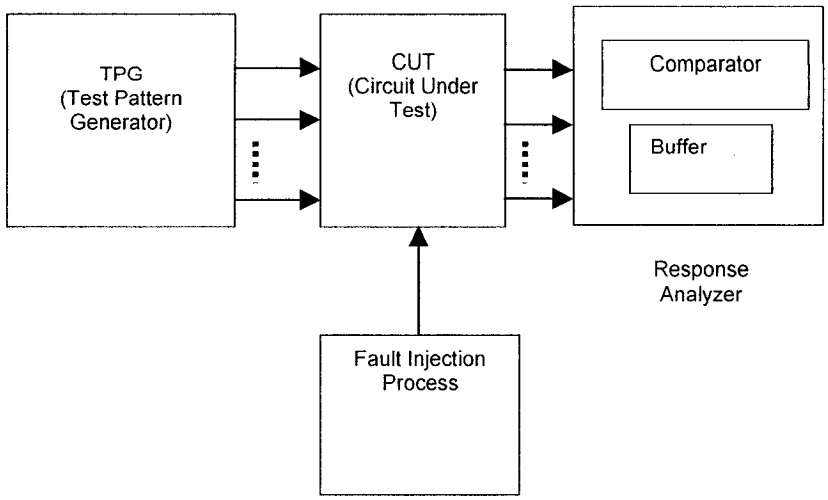


Figure 3.1 Block Diagram of a Fault Simulator

3.2 Conventional Fault Simulator

Fault simulator for electronic systems/ circuits/ modules, both hardware and software models are used. Hardware models are the traditional way of verifying the designs, software models are becoming more popular due to their economy, accuracy derived largely from the advances of in computing. With the increase of the density and complexity of the digital circuits, the software simulation is more effective and efficient method for design validation [35].

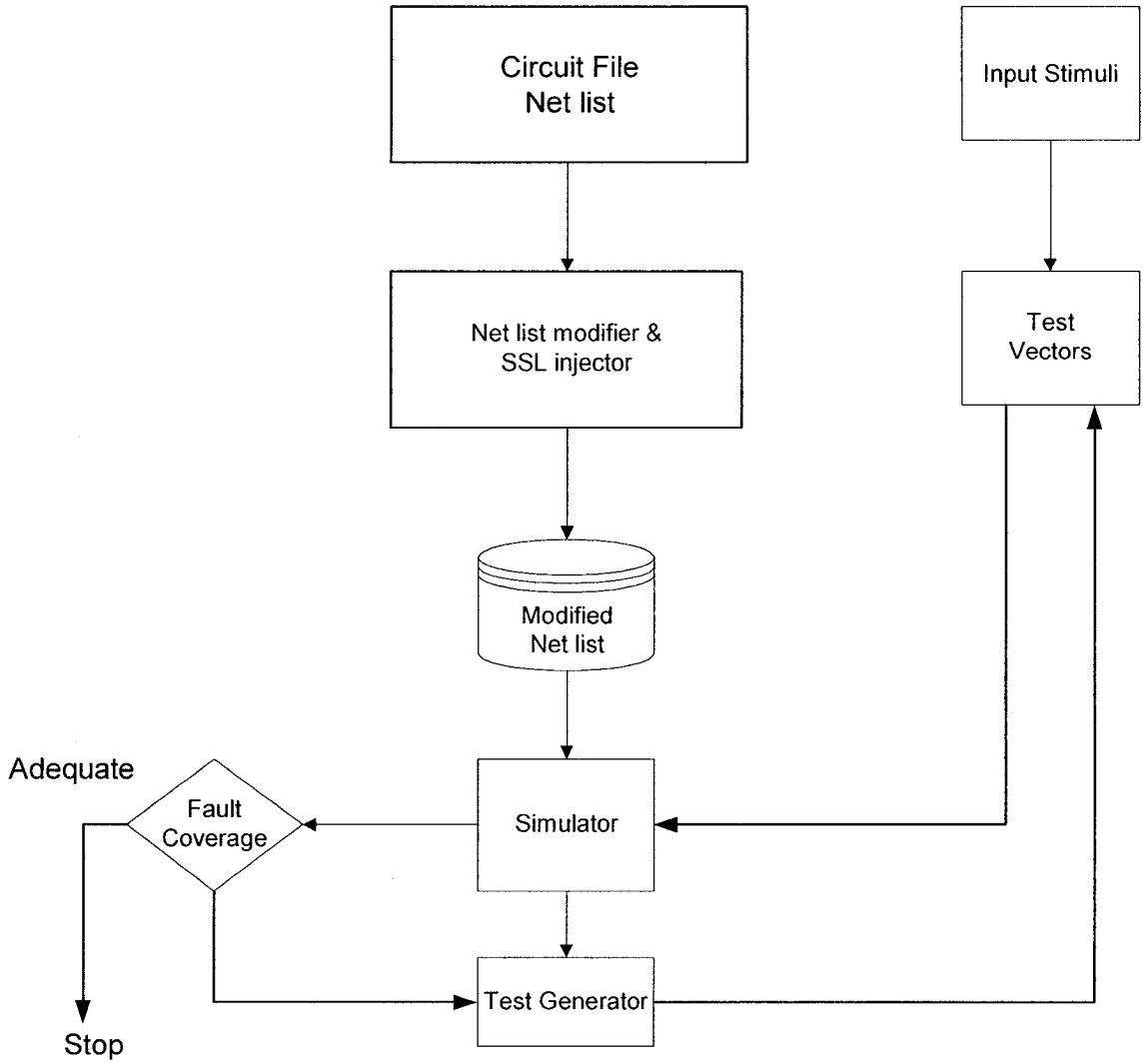


Figure 3.2 Conventional fault simulator

The fault simulator is the main component of the whole simulation process. Here the total process is written in C language. In the whole process, there are many steps before injecting test vectors or running the test on the circuit. When a circuit file is selected, the net list modifier first modifies the circuit by injecting single-stuck line faults (SSL). So modified net list circuit file with fault is crated and then the test patterns are generated and applied to the circuit to find out the fault coverage. If fault coverage is too low then it generates new sets of test

vectors to detect the fault at the output end of the circuit. When satisfactory fault coverage is obtained, the simulation process stops. Pseudorandom or deterministic test can be run depending on the type of the test vectors selected. The software simulator have the provision to select pseudorandom, deterministic, number of test vectors to be applied and also the initial seed value for test generation could be selected. Our task is to develop aliasing free compactor, which we will put at the output to minimize the number of outputs. We used ATALANTA and FSIM for deterministic and pseudorandom testing respectively.

3.2.1 Test Vector Generation

Test generator generates pseudo-exhaustive and pseudorandom test vectors. Number of test vectors and the sequence of the patterns depend on the type and level of complexity of the circuits. Some circuits need very few test vectors for full fault coverage with single stuck-faults. For BIST application, pseudorandom test sets can be generated on-chip with low hardware cost and which can generate test vectors for high fault coverage.

For hardware implementation of the random TPG shift registers are used. A LFSR, when clocked, advances the signal through the register from on bit to the next most significant bit. Some of the outputs are combined in exclusive–OR configuration to form the feedback mechanism. The LFSR can be formed by performing exclusive-OR operation to the outputs of two or more of the flip-flops together and feeding those outputs back into the input of one of the flip-flops as shown in Figure 3.3. The circuit can generate pseudorandom test patterns when

the outputs of the flip-flops are loaded with seed values, any value except all 0s and should be clocked.

A maximal-length of LFSR produces the maximum number of patterns, the pattern count is equal to 2^{n-1}, where n is the number of register elements in the LFSR.

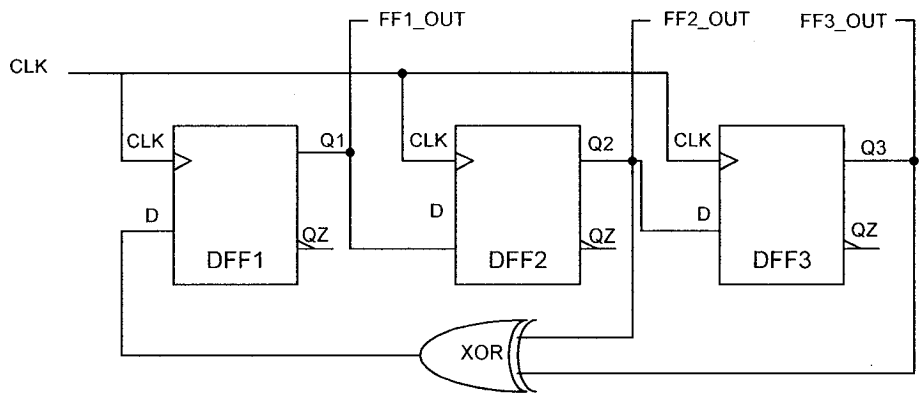


Figure 3.3 Linear Feedback Shift Register (LFSR)

For predicting the maximum length of LFSR, Peterson Weldon has compiled a table as shown in Table 3.1 of maximum length of LFSR. A three bits LFSR is shown in the table.

Table 3.1 Maximum length of LFSR

CLOCK PULSE	Q1	Q2	Q3	COMMENTS
1	1	1	1	Seed Value
2	0	1	1	
3	0	0	1	
4	1	0	0	
5	0	1	0	
6	1	0	1	
7	1	1	0	
8	1	1	1	Starts repeat

In our thesis, we implemented the test signal generator by software simulation rather using hardware technique. The random signal is generated by the simulator, which is written in C language. Initializing the simulator with a certain seed produces exactly the same series of random numbers on all platforms.

3.2.2 Types of Faults

In digital circuits, enormous number of various failures could be present and not feasible to analyze them all together, so failures are grouped according to their logical fault effect on the functionality of the circuit and which leads to construction of fault models as the basis for testing algorithms. Fault denotes the physical failure of the circuit, the fault effect denotes the logical effect of the fault

on the signal being processed by the circuit, and an error is defined as the condition (state) of a circuit/ system with a fault. Faults can be *permanent faults*, which are not removable without repair or re-design, and also can be temporary faults (transient or intermittent), *temporary faults* appear and disappear within a short intervals of time. There is another faults, which is known as *delay faults*, it is the effect of the operating speed of the circuit and can be controlled by varying the clock speed, better lay out, short inter-connection etc.

Permanent faults are classified as the catastrophic faults, could be due to open or short. Due to permanent faults, the topology of the circuit/ system is changed. Permanent faults are also needs to be tested because these could be the result of processing/ layout disturbance in the manufacturing process. The performance parameter of each manufactured circuit will be deviated from the specification due to fault and therefore corresponds to a different point in each parameter space. If it is within fault-free space then circuit is treated as fault free, otherwise it is considered as faulty circuit.

3.2.3 Fault Modeling

For analysis of the circuit and circuit fault, fault modeling is required. Fault models help to generate tests, and they help to evaluate test quality defined in terms of fault coverage [35]. An effective fault model is one that is simple to analyze and also closely represents the behavior of physical faults in the digital circuits. Physically there are many faults are possible, like short interconnection, open circuit path etc. but these are very complex to model. The faults most

commonly modeled are stuck-at-faults, which is very simple fault to analyze and proved to be very effective in representing the faulty behavior of actual devices. The stuck–at-faults are often referred to as logical faults, and are assumed to effect only the interconnections. Possible fault locations are at the inputs or outputs of the gates, each line can have two types of stuck-faults: stuck-at-0 or stuck-at-1. A stuck-at-1 fault always has a logical value irrespective of changes in the inputs. Normally several stuck-faults can be simultaneously present in the circuit. A circuit with n lines can have 3n-1 possible stuck-at-1 and stuck-at-0 fault combinations.

Fault models are used in test generation and fault diagnosis, and it is important that the tests that cover the target faults are able to detect the defective parts, single stuck-at-tests are successful in this area. The defects cannot be modeled directly, and it is never the less possible to successfully identify defects using fault simulation, but to achieve this, fault models are made as realistic as possible i.e. fault behavior must conform closely to defect behavior [37].

The fault models can be grouped into the following fault models: Single stuck-at-fault model, functional fault model, component open and short fault model, memory fault model, and delay fault model. As we mentioned already that, the stuck-at-fault model is the most common fault model. We create stuck-at faults in our CUT to find out the fault coverage with compactor. In this chapter, we will discuss briefly about the stuck-at fault model and the terms used in our fault simulation process.

Single Stuck-at-faults

Stuck-at-faults can have three properties:

Only one line is faulty.

The faulty line is set to 0 or 1.

The fault can be at the input or output of the gate.

In Figure 3.4 below a XOR circuit is shown, the circuit has 12 fault locations(X) and 24 stuck-at-faults, stuck-at -0 or stuck-at-1, twelve faults of stuck-at-1 fault and twelve faults of stuck-at-0 fault.

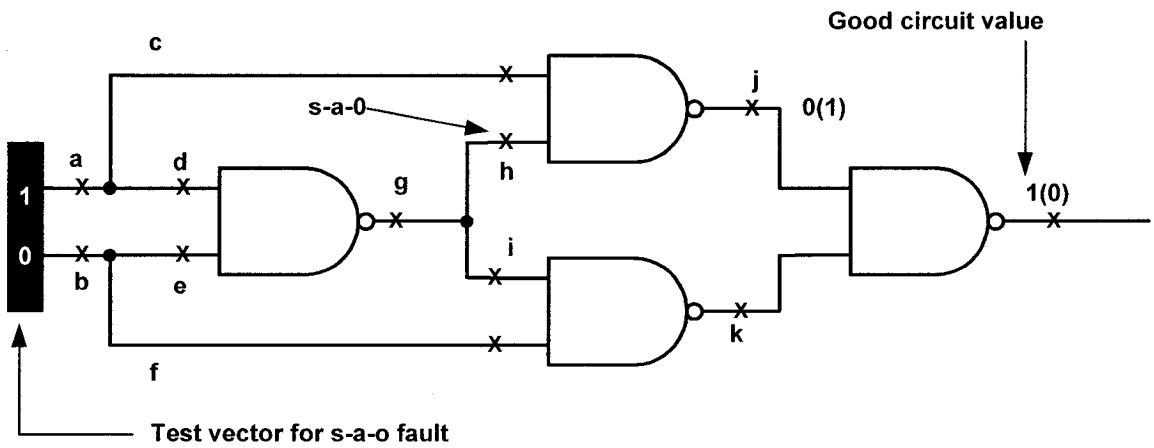


Figure 3.4: Single stuck-at Faults

Fault Equivalence

Two faults f_1 and f_2 are equivalent when all the tests detecting f_1 also detect f_2, and the faults are indistinguishable and have exactly the same set of tests.

Fault Dominance

Given two faults f_1 and f_2, fault f_1 considered to be dominant over fault f_2 when every test for f_2 is also test for f_1. When f_1 dominates f_2, then f_1 is removed from the fault list.

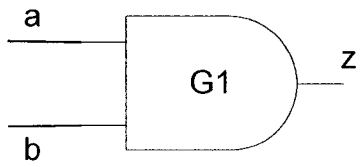
Fault Collapsing

When the effect of two faults are indistinguishable at the output of a circuit it is called equivalent and any tests detecting one of them will also detect the other [35]. If one representative fault is selected from each class of equivalent faults then it is called equivalence fault collapsing.

In physical fault testing, physical defects are abstracted into a logical fault model. We know that the most widely used fault model is Single stuck-line (SSL) model [36], and every single line can become permanently fixed (stuck) at logic 1 or 0 values. Although the model is simple and technology independent, it represents a large number of physical faults.

Fault collapsing reduces the number of faults using two relationships among faults. Fault equivalence and fault dominance. As we described above that two faults are equivalent if the faulty functions produced by the two faults are equal and alternatively the two faults are equivalent if they can be detected by the same tests. In this case, there is no way to distinguish between the two faults. In the example shown in Figure 3.4, the SSL fault stuck-at-0 represented by a/0 is equivalent to the fault z/0. If two faults are equivalent then one of the

faults can be dropped from the fault list since the detection of the other fault guarantees the detection of the fault dropped.



Inputs		Correct	Faulty functions					
a	b	functions(z)	a/0	a/1	b/0	b/1	z/0	z/1
0	0	0	0	0	0	0	0	1
0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1
1	1	1	0	1	0	1	0	1

Figure 3.5 Fault Collapsing for 2 Input Gate.

Fault f_1 is considered to dominate another fault f_2 when every test for f_2 is also test for f_1 [46]. In figure 3.4, the fault z/1 dominates the fault a/1 since the only test vector 01 for a/1 (shaded in figure) is also a test for z/1. If a fault f_1 dominates a fault f_2, then the fault f_1 can be dropped from the fault list as the detection of f_2 guarantees the detection of f_1. By applying fault collapsing to the AND gate in figure 3.4, the number of faults can be reduced from six to three [38]. The fault a/0 and b/0 are dropped since they are equivalent to z/0. The fault z/1 is also dropped since it dominates both a/1 and b/1, so the collapsed fault list is {a/1, b/1, z/0}. A test set that detects the faults in the collapsed list can be

obtained from the table in figure 3.5 as {01, 10, 11} and the test detects all faults in the collapsed fault list and consequently all six faults in the AND gate.

Fault collapsing can be divided in to two approaches: local and global. The local fault collapsing method computes the collapsed fault list for individual gates and then collects the collapsed fault lists for the gates to form the overall collapsed fault list for the circuit. Global fault collapsing is similar to local fault collapsing, except that we perform the same process of fault collapsing on the entire circuit as opposed to individual gates [38].

Path Sensitization

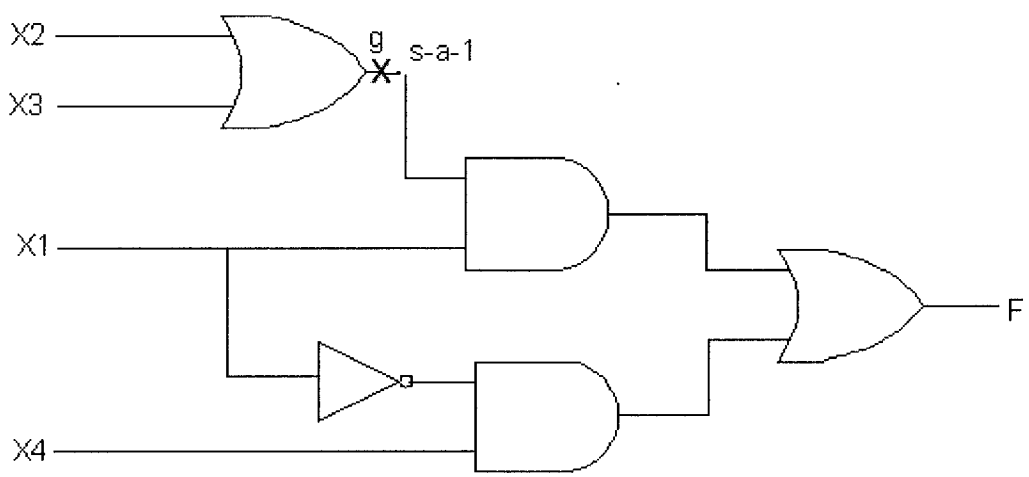


Figure 3.6 Sensitized Path

A line whose value in the test X changes in the presence of a fault j is said to be sensitized to j by X. A path composed of sensitized lines is called a sensitized path. To detect a fault s-a-j, j=0, 1, by the input vector X, the input X must cause

the signal a in the normal (fault-free) circuit to take the value j. For example, in circuit of Figure 3.5, to detect **g** s-a-1, wee need to satisfy the condition X2 + X3 = 0. The error signal must be propagated to the output.

3.2.4 Fault Injection Technique

The most important and the vital part of the test generation process is the fault injection. The efficiency of the fault injection technique plays vital role in test generation. Fault injection can be done in both hardware and software levels.

Hardware Fault Injection

To implement hardware fault injection, additional hardware is used to introduce faults to the target system or circuit. Depending on the faults and their locations, hardware – implemented fault injection method falls into two categories:

Hardware fault injection with contact

Hardware fault injection without contact

Hardware fault injection with contact: The injector has direct physical contact with the target system/circuit, producing voltage or current changes externally to the target system/circuit.

Hardware fault injection without contact: The injector has no direct physical contact with the target system/circuit. An external source produces some

physical phenomenon, such as heavy-ion radiation and electromagnetic interference, causing spurious currents inside the target.

The hardware fault injection technique is to inject faults iteratively to every single line (wire), to test for both stuck-at-0 and stuck-at-1 faults.

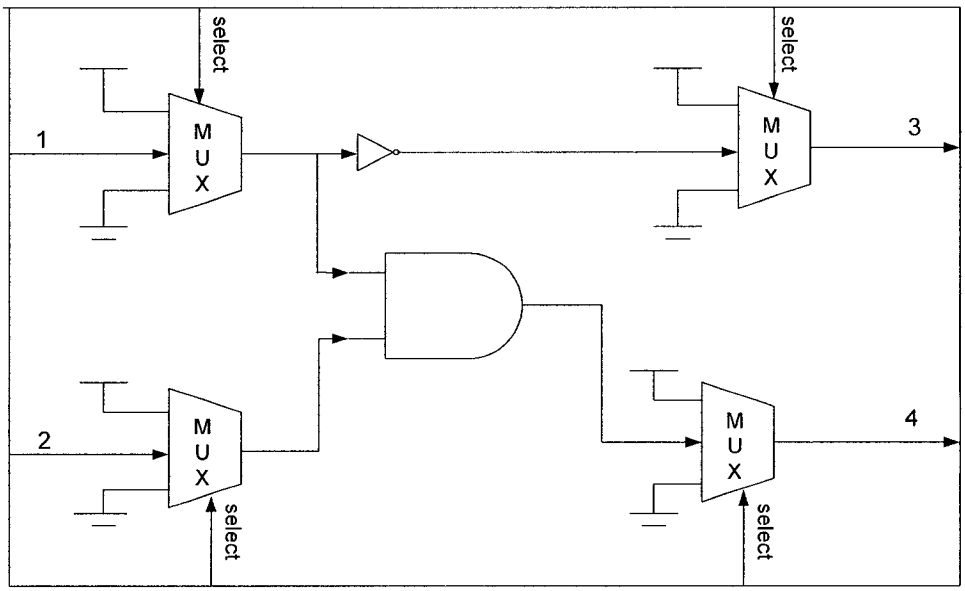


Figure 3.7 Test Set Up For Hardware Fault Injection.

To do so multiplexers are used as shown in Figure 3.6. In fault injection scheme, every mutually exclusive wire now has a 3 input multiplexer (MUX) introduced within it, which allows us to either run the wire as it is (normal operation), or inject stuck-at-0 or stuck-at-1 faults. If the select signal is "00", then the wire runs as it is, but if the values are at "01", then a stuck-at-1 fault is injected, if the values are "10" then stuck-at-0 fault is injected, and finally, if the values are at "11", the again the wire is at normal operation.

Software Fault Injection

These techniques are the most attractive because they do not require expensive hardware. Furthermore, they can be used to target applications and operating systems, which is difficult to do with hardware fault injection. In our thesis, we used software fault injection scheme. Since we are dealing with ISCAS85 combinational circuits, the fault injections in the circuits are done in three phases. The first phase injects faults in the input wires, second phase injects the faults in the internal wires of the circuit, and finally, in third phase faults are injected output of the circuits. The circuits are in the net list format and the simulation software in every steps starting from the inputs, internal wires and the output wires injected the faults. The software generates the faults internally; the user also can inject the faults by using fault file. We used FSIM and ATALANTA for simulation. The software first injects the faults (either stuck-at-0 or stuck-at-1) and then test patterns are injected to check the response of at the output, the output response is compared with the fault free signature previously stored with good circuit. Finally, when faults are injected for all the wires and tested, software then calculate the fault coverage by dividing the number of faults detected by numbers faults injected.

$$\text{Fault Coverage} = \frac{\text{Total numbers of faults detected}}{\text{Total numbers of faults injected}} \text{ X } 100$$

Where the total numbers of faults injected = 2 X total numbers (input + out put + internal wires). In Table 3.2 a fault list file is shown, here gate_A and gate_B are the name of the gates. The first line, "gate_A -> gate_B/1" describes the stuck-at

1 fault on the gate_B input line, which is connected to gate_A. Similarly, the second line, "gate_A->gate_B/0" describes the stuck-at 0 fault on the gate B input which is connected to gate A.

Table 3.2 Fault Injection File.

gate_A -> gate_B /1 gate_A -> gate_B /0 gate_A /1 gate_B /1

The third and the forth lines describe the stuck-at 1 faults on the gate_A output and the gate_B output, respectively.

3.3 Fault Simulation In HDL

3.3.1 Stuck-At Fault Modeling In Verilog

A circuit composed of resistors, diodes and transistors can be represented as an interconnection of logic gates. If the gate –level model is altered so as to represent a faulted circuit, then the behavior of the faulted circuit can be analyzed and tests developed to distinguish it from the fault -free circuit. As a minimum, a fault model must possess the following four properties [59]:

1. It must correspond to real faults.
2. It must have adequate granularity.
3. It must be accountable.

4. It must be easily automated.

Fault simulation and analysis require modeling the faulty behavior of the circuit under test. Fault models serve two purposes. First, they help generate tests and second, they help evaluate test quality defined in terms of coverage of modeled faults. A good fault model is one that is simple to analyze and yet closely represents the behavior of the physical faults in the circuit. Fault modeling is strongly related to circuit modeling. As the primary objective of HDLs was to model the circuit hardware, so they are also suitable for fault modeling with various levels of abstraction.

If we consider a digital circuit as the interconnection of logic gates, numerous fault possibilities, viz. missing gates, wrong gate types, missing interconnections, added interconnections, shorted interconnections, etc. are possible. Most of these faults, although physically real, are too complex to model. The faults most commonly modeled are stuck-at faults. Stuck-at faults are not only the simplest faults to analyze, but they also have proved to be very effective in representing the faulty behavior of the actual devices. The simplicity of stuck-at faults is derived from their logical behavior. That is why these faults are often referred to as logical faults. Stuck-at faults can be classified as stuck-at-1 (s-a-1) and stuck-at-0 (s-a-0) faults. Fault situation in which the node under consideration will have a logical value of 1, no matter how the circuit under test excitation changes is called stuck-at-1 fault. This situation is similar to forcing a logical value of 1 to that particular node. On the other hand, fault situation in which the node under consideration will have logical value of 0, no matter how the circuit under test

excitation changes is called stuck-at-0 fault. This situation is equivalent to forcing a logical value of 0 to that particular node. We next discuss the modeling of these fault types in Verilog.

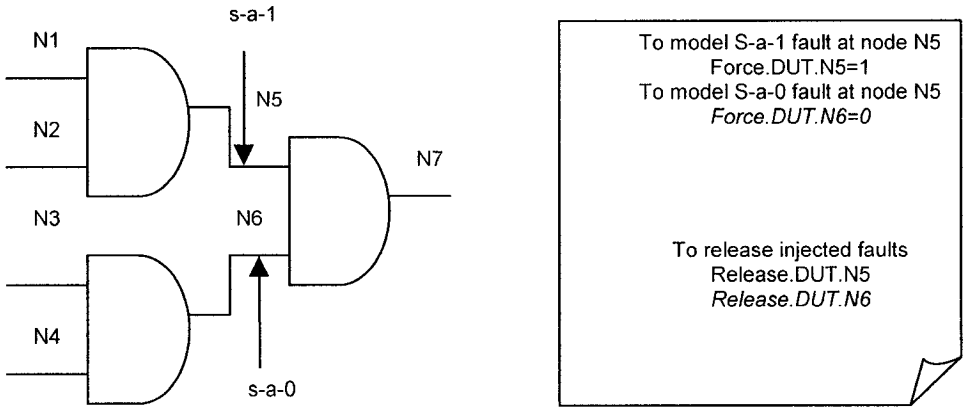


Figure3.8.Modelling S-A-1 and S-A-0 faults in verilog

Force statement in Verilog is used to override assignments on both registers and nets, and a specific logic value can be assigned to that net or register, while *release* statement releases the node from any forced value in effect. For example, *force dff.q=1'b1* will force a logic value of 1 to that specific node q of dff as shown in Fig. 2. Thus, it models the stuck-at-1 fault to that particular node. A simple statement like *release.dff.q* will release the forced value from that particular node and will bring that circuit to its normal condition. A stuck-at-0 fault can also be modeled similarly. In the next section, we will show how this kind of fault modeling technique can be used to implement a fault simulator.

3.3.2 Fault Injection Strategy:

The core of any fault simulator is the fault injection procedure. Different approaches for injection of permanent faults in HDL descriptions are possible, some of which have already been proposed in the literature.

Changing the HDL code: Here the original HDL instructions are enriched by the addition of necessary code to inject the fault, or new input signals are added to control fault injection [55], [35]. This technique, however, significantly slows down the simulation process, because the additional source lines are always simulated, even when they are not used to inject the fault.

Modifying the simulator: The code necessary to inject and detect faults is added into the simulator source code. This technique is probably the fastest fault injection methodology, and promises to simulate each faulty circuit as fast as a fault-free circuit. It is also extremely powerful, since one may change any parameter or register during simulation. The success of this technique depends on the availability of the source code of a good simulator or on the amount of effort it might require to develop such a simulator.

Interacting with the simulator using simulation command script: The faults are injected through the simulator user interface using simulation commands. This technique is less powerful than modifying the simulator, but during simulation it is nearly as fast. In fact, no additional source code is present, and commands are active only when the fault is injected. However, as this approach uses specific simulator command scripts, it suffers from the drawback of being vendor-dependent.

Use of force and release in Verilog: Stuck-at-faults can be modeled using Verilog construct *force* and *release* as discussed earlier. This opens up the opportunity of using these constructs as fault injection constructs. This approach has all the advantages of the third strategy as described before, with the added advantage of being simulator-independent. This certainly increases the portability of the simulation environment from one simulator to another.

The fault injection strategy used in this thesis work belongs to the fourth category. *Force* and *release* constructs have been used from Verilog test bench to inject faults to the CUT. The module instance parameter access has been used to access internal nodes of the CUT.

3.3.3 Fault Simulation Environment

The basic concept for simulation is based on any standard Verilog digital circuit simulation procedure where a DUT (design under test) is being tested through the application of stimuli from a test bench module. The test bench module is responsible for generating the entire stimuli for the circuit under test and collecting the responses from the circuit. In this case, to inject faults, our test bench not only applies the stimulus to the circuit under test, it also accesses the internal node points using module instance in Verilog to insert fault. After insertion of a stuck-at-fault, the response of the circuit is then compared against a pre-fault condition response to determine the detectability of that fault condition.

Our fault simulation environment is comprised of two steps. Step1 is to build a generic fault simulation environment for the ISCAS 85 combinational benchmark circuits to automate the test bench generation process. We have developed a C application program, which parses through the Verilog description of ISCAS 85 circuits, identifies all inputs, outputs, and internal nodes, and generates a Verilog test bench following the above-mentioned procedure to insert faults obeying appropriate Verilog syntax. The faults are injected to all the inputs and internal nodes. The internal nodes are identified from wire declaration portions of Verilog source files. In Step 2, this test bench and CUT are being fed to a standard Verilog (in our case ModelSim) simulator and simulated using routine simulation and compilation procedure. The main advantage of this procedure is that the main circuit description never gets modified to simulate a fault. Besides, recompilation is not necessary for each of the faults, which allows simulation to run at full speed. Figs. 3.9 and 3.10 show block diagrams that best describe this fault simulation environment.

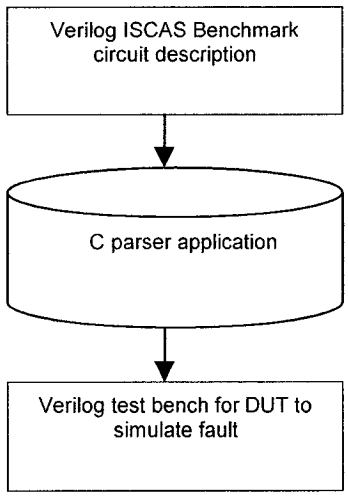


Figure 3.9 Fault Simulation Environment

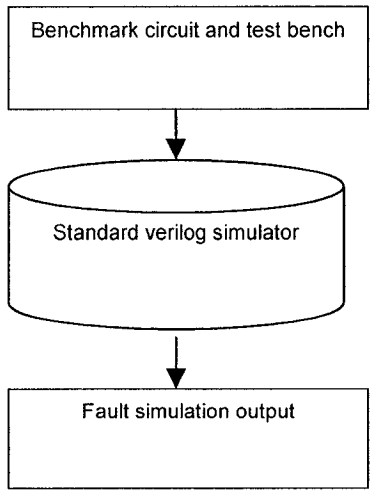


Figure3.10 Fault Simulation Environment Step2

3.4 Conclusion

This work develops approach to fault simulation of ISCAS 85 combinational benchmark circuits using HDL description of gate level netlists. The suggested method is based on exploitation of the existing *force* and *release* features available

in Verilog hardware description language. With a relatively moderate effort, an effective fault simulator can be built by proper utilization of Verilog syntax and its application to fault modeling. The implemented fault simulation system is able to emulate all of the ISCAS 85 combinational circuits. The fault coverage obtained using the simulator is comparable to the fault coverage by other simulation approaches, as evidenced by experimental results.

The experimental results as shown in chapter 5 also demonstrate that access to the source code of a HDL simulator or its modifications is not essential in order to compute faulty responses from the digital circuit. The strong fault modeling capability inherent in Verilog and its universality in design industry open up the possibility of greatly optimizing the efficiency of the proposed approach. One of the advantages of this fault simulation approach over others is that it is simulator independent, and any standard Verilog simulator can be used as fault simulator. This makes the test benches for the benchmark circuits portable so that they can be used under any Verilog digital simulation environment.

One of the main motivations behind this effort is to extract information of fault densities at different output lines of the Circuit Under test (CUT). This is normally referred as fault grading with respect to output. As in this simulation approach we exploited the digital logic simulation capabilities of a standard verilog simulator, we were able to extract the desired information by customizing our test bench. In Chapter 4 of this thesis, it is explained how this information is used in designing aliasing free space compactor.

Though the simulation environment used in this work uses gate level model of benchmark circuits, the same concept can possibly be extended to simulate faults at RT level as well. One of the prime reasons that support the previous statement is that most of the present day complex chip designs are done using RT level description. This in turn might prove to be economical to IC production test, yielding better results.

Chapter 4

Design Of Space Compactor

In Previous chapter we have discussed different aspects of fault simulation and related concepts. We also have outlined an improved fault simulation scheme based on standard verilog logic simulator. The main motivation behind this work was to extract the strong and weak compatible output lines for Circuit Under Test. In this Current Chapter we will show how these extracted property can be beneficial in designing an efficient space compaction algorithm. First, The mathematical basis for the algorithm will be formulated followed by a description of the proposed algorithm.

4.1 Mathematical Basis

Property 1 Let A and B represent two of the outputs of a CUT. Let these CUT outputs be merged by an AND (NAND) gate and let the AND (NAND) gate output be z_1. Then we can envisage the under noted scenarios:

Case 1: Fault-free (FF) outputs = Faulty (F) outputs $\Rightarrow$ Outputs A and B of the CUT do not detect any faults.

Case 2: Only those faults that occur at A and B are detectable at $z_1 \Rightarrow FF \neq F$.

Case 3: Faults occur at A and B but not all or some are detectable at $z_1 \Rightarrow FF \neq F$. That is, these missed faults at z_1 are detected additionally at other outputs of the CUT besides A and B.

Definition 1 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the CUT outputs A, B are merged by an AND (NAND) gate, we define lines A, B to be **strongly** AND (NAND) compatible, written as

(AB) s-AND (NAND) compatible.

Definition 2 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above (but not Cases 1-2). If the CUT outputs A, B are merged by an AND (NAND) gate, we define lines A, B to be **weakly** AND(NAND) compatible, written as

(AB) w-AND(NAND) compatible.

Definition 3 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs

A, B conforms to none of the conditions as specified by Cases 1-3 above. If the CUT outputs A, B are merged by an AND(NAND) gate, we define lines A, B to be AND (NAND) incompatible, written as

(AB) AND(NAND) incompatible.

Definition 4 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the CUT outputs A, B are merged by an OR (NOR) gate, we define lines A, B to be **strongly** OR (NOR) compatible, written as

(AB) s-OR (NOR) compatible.

Definition 5 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above (but not Cases 1-2). If the CUT outputs A, B are merged by an OR (NOR) gate, we define lines A, B to be **weakly** OR (NOR) compatible, written as

(AB) w-OR (NOR) compatible.

Definition 6 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to none of the conditions as specified by Cases 1-3 above. If the CUT outputs A, B are merged by an OR(NOR) gate, we define lines A, B to be OR(NOR) incompatible, written as

(AB) OR(NOR) incompatible.

Definition 7 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the CUT outputs A, B are merged by an XOR (XNOR) gate, we define lines A, B to be **strongly** XOR (XNOR) compatible, written as

(AB) s-XOR (XNOR) compatible.

Definition 8 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs

A, B conforms to conditions of Case 3 above (but not Cases 1-2 . If the CUT outputs A, B are merged by an XOR (XNOR) gate, we define lines A, B to be **weakly** XOR (XNOR) compatible, written as

(AB) w-XOR (XNOR) compatible.

Definition 9 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to none of the conditions as specified by Cases 1-3 above. If the CUT outputs A, B are merged by an XOR (XNOR) gate, we define lines A, B to be XOR (XNOR) incompatible, written as

(AB) XOR (XNOR) incompatible.

Definition 10 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to either of the three conditions as specified by Cases 1-3 above, but unknown to us. If the CUT outputs A, B are merged under these conditions by an AND(NAND), OR(NOR) or XOR(XNOR) gate, then we define lines A, B to be **simply** AND(NAND), OR(NOR) or XOR (XNOR) compatible, written as

(AB) AND (NAND), OR(NOR) or XOR(XNOR) compatible.

Theorem 1 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-AND (NAND) compatible. Similarly, let the outputs B, C be s-AND (NAND) compatible, and the outputs A, C be s-AND (NAND) compatible. Then (ABC) is s-AND (NAND) compatible and all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 2 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs A_1, $A_2, \ldots, A_m$ are s-AND(NAND) compatible. Then all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 3 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs

A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-OR (NOR) compatible. Similarly, let the outputs B, C be s-OR (NOR) compatible, and the outputs A, C be s-OR (NOR) compatible. Then (ABC) is s-OR (NOR) compatible and all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 4 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $A_1, A_2, \ldots, A_m$ are s-OR(NOR) compatible. Then all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 5 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests) or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-XOR (XNOR) compatible. Similarly, let the outputs B, C be s-XOR (XNOR) compatible, and the outputs A, C be s-XOR (XNOR) compatible. Then (ABC) is s-XOR (XNOR) compatible and all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 6 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $A_1, A_2, \ldots, A_m$ are s-XOR(XNOR) compatible. Then all faults are detected at z_1.

Proof: The proof follows readily.

Theorem 7 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-AND (NAND) compatible. Similarly, let the outputs B, C be w-AND (NAND) compatible, and the outputs A, C be w-AND (NAND) compatible. Then (ABC) is w-AND (NAND) compatible and all faults may or may not be detected at z_1.

Proof: The proof follows readily.

Corollary 7.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but

complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A_1, A_2, ... , A_m conforms to conditions of Case 3 above, so that the outputs A_1, A_2, ... , A_m are w-AND(NAND) compatible. Then all faults may or may not be detected at z_1.

Theorem 8 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-OR (NOR) compatible. Similarly, let the outputs B, C be w-OR (NOR) compatible, and the outputs A, C be w-OR (NOR) compatible. Then (ABC) is w-OR (NOR) compatible and all faults may or may not be detected at z_1.

Proof: The proof follows readily.

Corollary 8.1 Let A_1, A_2, ... , A_m be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A_1, A_2, ... , A_m be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A_1, A_2, ... , A_m conforms to conditions of Case 3 above, so that the outputs A_1, A_2, ... , A_m are w-OR(NOR) compatible. Then all faults may or may not be detected at z_1.

Theorem 9 Let A, B, C, ... be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-XOR (XNOR) compatible. Similarly, let the outputs B, C be w-XOR (XNOR) compatible, and the outputs A, C be w-XOR (XNOR) compatible. Then (ABC) is w-XOR (XNOR) compatible and all faults may or may not be detected at z_1.

Proof: The proof follows readily.

Corollary 9.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output CUT. Let the faults detected at the CUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the CUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$ (τ might not be a minimal or non-minimal but complete set of tests), or pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Case 3 above, so that the outputs $A_1, A_2, \ldots, A_m$ are w-XOR(XNOR) compatible. Then all faults may or may not be detected at z_1.

4.2 Algorithms Development

The developed zero-aliasing space compaction approach is comprised of three different algorithms—one for finding the strong and weak compatible pairs of response data outputs of the MUT for logic families AND/NAND, OR/NOR, and XOR/XNOR, one to find Maximal Compatible (MC)s classes using paul-unger method, and one for constructing the compaction networks based on the knowledge of the generated MCs classes.

4.2.1 Algorithm A

Figure 4.4 provides a flow chart of the approach utilized in the design of our proposed zero-aliasing space compactors. A detailed formulation of the various steps involved in the implementation is given below.

Step 1) Define the possible maximum number of stages in the space compaction trees at the MUT output.

Step 2) Compute the total number of output lines in the MUT. Continue the following steps unless there is only a single output line (possibly).

Step 3) Find the sets of all MCs classes from the MUT for logic AND/NAND, OR/NOR, and XOR/XNOR by employing Algorithm B.

Step 4) Select an MC class MC_i based on largest number of output lines, from the sets of all MCs. Select the second largest class during subsequent iteration, if 100% fault coverage is not realized in the preceding iteration from the same MUT.

Step 5) Merge the selected output lines of the MC_i using appropriate logic gates AND/NAND, OR/NOR or XOR/XNOR.

Step 6) Add a new output line corresponding to the selected merged outputs in MC_i.

Step 7) Discard all the output lines already used in MC_i.

Step 8) Search for another MC class MC_j from the remaining output lines.

Step 9) Merge the selected output lines in MC_j using appropriate logic gates.

Step 10) Add a new output line corresponding to the selected merged outputs in MC_j.

Step 11) Discard all the output lines already used in MC_j.

Step 12) Repeat step 8 as long as there are MCs in the sets and enough output lines. Step Step13) Calculate all the remaining output lines that do not belong to any of the selected MCs.

Step 14) Merge all the remaining lines with XOR/XNOR gate.

Step 15) Add a new output line corresponding to the selected merged outputs.

Step 16) Inject stuck-at logic faults into the newly generated MUT (original MUT +compactor hardware).

Step 17) Compute fault coverage by applying input test patterns.

Step 18) If the fault coverage is 100%, then replace the old MUT with the new MUT, and repeat step 2 for computing the second stage of the compactor.

Step 19) If the fault coverage is less than 100%, then merge all the remaining lines with two-input XOR/XNOR gates, two output lines at a time.

Step 20) Add a new output line corresponding to the selected merged outputs.

Step 21) Inject stuck-at logic faults into the newly generated MUT (original MUT +compactor hardware).

Step 22) Compute fault coverage by applying input test patterns.

Step 23) If the fault coverage is less than 100%, then continue to work on the same MUT and repeat step 4 for selecting a new MC class MC_k.

Step 24) If the fault coverage is 100%, then replace the old MUT with the new MUT, and repeat step 2 for computing the second stage and subsequent stages of the compactor.

4.2.2 Algorithm B

The selection criteria for a possible merger candidate are based on finding a non-disjoint property among the strong compatible pairs. This procedure is supported by the theory as stated in mathematical basis section. In the event that such non-disjoint property does not exist, the maximum compatible class is formed. Formation of maximal compatible classes is required to find merger candidates at a certain stage of compactor formation. The methodology as stated by paull-unger [69] has been used to find the maximal compatible classes. A brief review of the paull-unger method is presented below due to its relevance with subject matter.

Some step-by-step processes for obtaining the maximal compatibles are given by Paul and Unger. A simple method using Boolean algebra is given here:

(1) For every incompatible row-pair, form the Boolean sum of the corresponding row designations, and write the Boolean product of all of these sums.

(2) Multiply out this expression to obtain an equivalent sum of products, eliminating all redundancy in the process.

(3) For each resultant product, write the set of all missing row designations; these sets constitute all of the maximal compatibles.

The method can be well illustrated using an example from Paull and Unger[69]. The implication table from that example is reproduced in Fig. 4.1 showing only the ***X*** entries that denote the incompatible row-pairs.

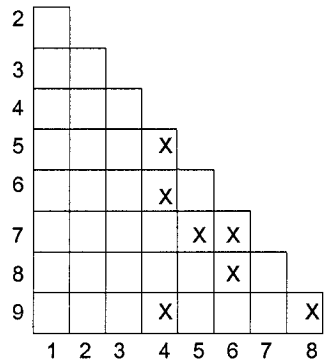


Figure 4.1 Paull-Unger Method-Implication table with X entries showing incompatible row pairs.

Step(1) : (4+5) (4+6) (4+9) (5+7) (6+7) (6+8) (8+9)

Step(2) : = 4568 + 4679 + 478 + 569

Step(3): MC's : (12379) (12358) (123569) (123478)

Once the maximum compatible classes are found, the next step is to form compaction networks using maximal compatible classes. This process exhaustively achieve maximum compaction ratio with high fault coverage.

4.2.3. Algorithm C

The main phenomenon used in this approach was based on the grading of output lines into two compatible classes – namely strong compatible and simply compatible. The output lines that satisfy the conditions as set by definition 1 of mathematical basis section are considered to be the strong compatible output pairs. Each pair of output lines is chosen sequentially as possible merger candidate and a

fault simulation is being performed with the modified CUT. If all the faults that appear in pre-merged output line pair are also detectable on the merged output line, then the pair is identified as strong compatible pair. The figure below best describes the strong compatible test procedure.

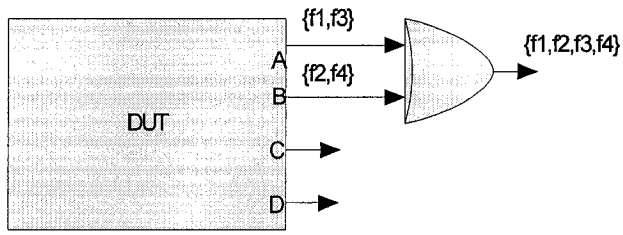


Figure 4.2 Strong Compatible test case

However in case of simply compatible scenario all the faults that are detectable at a certain pair of output lines may or may not be detectable at their corresponding merged output line. However those undetected faults may well be detected at other output lines of CUT. They are simply compatible pair of output lines as they combined with rest of the output lines yield 100% fault coverage.

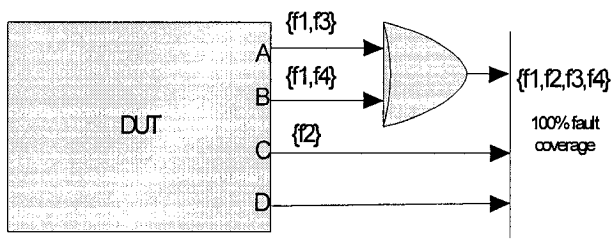


Figure 4.3 Simply Compatible test case

In each stage of the compactor formation, the output lines are first tested to find strong compatible output pairs for possible mergers. In the event that such strong compatible output pairs do not exist then simply compatible output pairs are considered for further post processing.

The following algorithm [51], [56] computes all incompatible pairs (less than 100% fault coverage) of the MUT output lines for logic AND/NAND, OR/NOR, and XOR/XNOR.

Step 1) Calculate the total number of output lines of the MUT.

Step 2) Generate all possible combinations (A_i,A_j)of output lines, taken two at a time, and store all pairs of the output lines (A_i,A_j).

Step 3) Select the first pair from the list of combined output lines (A_i,A_j).

Step 4) Merge the selected pair of output lines (A_i,A_j)using logic gates AND/NAND, OR/NOR, and XOR/XNOR, respectively, using only one logic gate at a time.

Step 5) Add a new output line to the original MUT corresponding to the outputs (A_i,A_j), one at a time.

Step 6) Discard the output lines (A_i,A_j)from the original MUT, and generate a new modified MUT.

Step 7) Inject stuck-at logic faults into the newly generated MUT and apply test patterns.

Step 8) If the fault coverage is less than 100%, then store the output pair (A_i,A_j) in the incompatible pairs database of logic AND/NAND, OR/NOR, and XOR/XNOR, respectively.

Step 9) Delete the pair just selected, from the list of combined output lines (A_i,A_j), and select the next pair.

Step 10) Repeat step 4 and continue until all pairs are selected

A flow chart diagram of the proposed approach is shown in Fig.4.4.

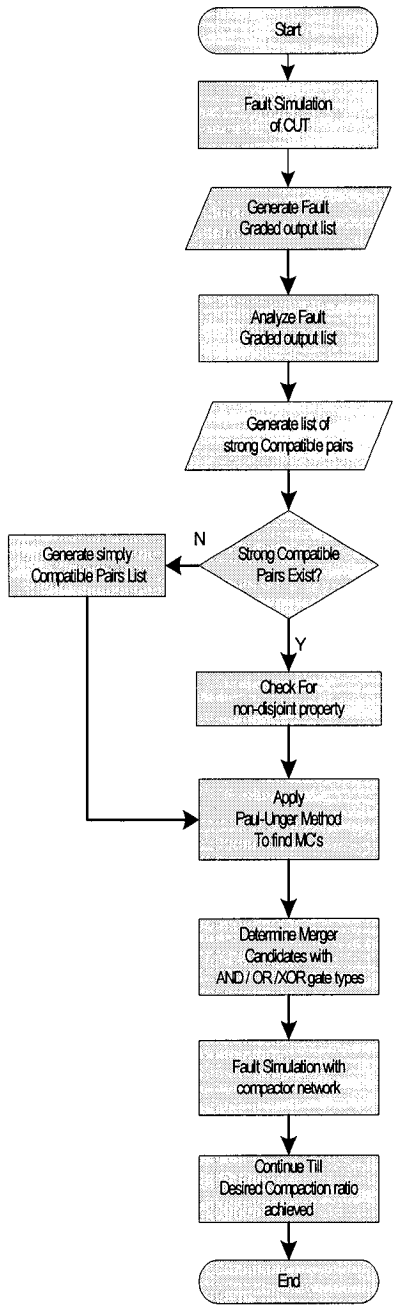


Figure 4.4 Flow diagram of algorithm

4.3 Example

Let us consider c432 ISCAS 85 benchmark combinational circuit shown in Figure 4.5, it has thirty-six inputs and seven outputs and one hundred and sixty logic gates. To test the circuit we injected stuck-at-0 and stuck-at-1 faults, compacted test patterns were injected at the primary inputs to detect all the faults.

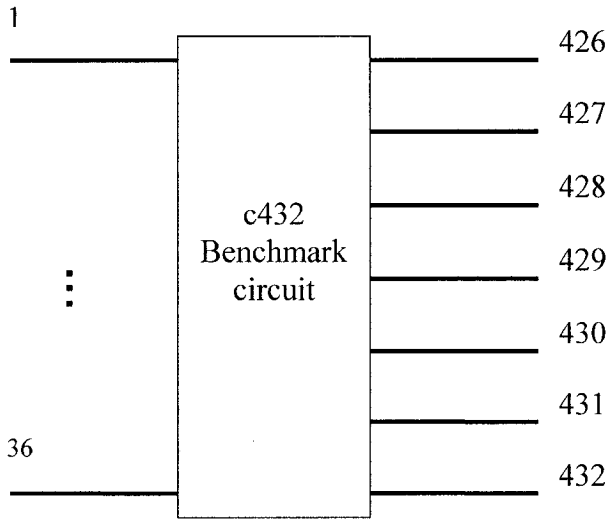


Figure 4.5 c432 Benchmark Circuit.

Stuck-at faults were injected, the seven bits output patterns were observed and compared with the pre-computed fault-free stored output patterns, the detected faults are recorded when faulty, and fault-free output patterns mismatched.

4.3.1 Compactor formation –Stage 1:

The Strong compatible pairs of the c432 ISCAS 85 benchmark circuit (MUT) output lines (O_i, O_j) for logic AND/NAND, OR/NOR and XOR/XNOR are obtained by applying developed algorithm and using deterministic input test set. There was no strong compatible pair obtained at this stage using AND logic. The Strong compatible pairs for logic XOR, and OR/NOR are shown in Table 4.1 and 4.2 respectively.

Table 4.1 Strong compatible Pairs for logic XOR.

(O_i , O_j)
(426, 427)
(426, 429)
(426, 430)
(426, 432)
(427, 430)
(427, 431)

Table 4.2 Strong compatible Pairs for Logic OR/NOR

(O_i , O_j)
(426, 427)

As evident from the list of strong compatible pairs of information, we check for non-disjoint property among the strong compatible output pairs. As it is seen pairs (426,427), (427,429) and (426,429) are strong compatible pairs and satisfies the condition for non-disjoint. Hence the lines (426,427 and 429) can be merged together using a 3-input XOR gate to achieve 100 % fault Coverage. So at the end of stage 1 the compactor circuit takes the form as shown in figure.

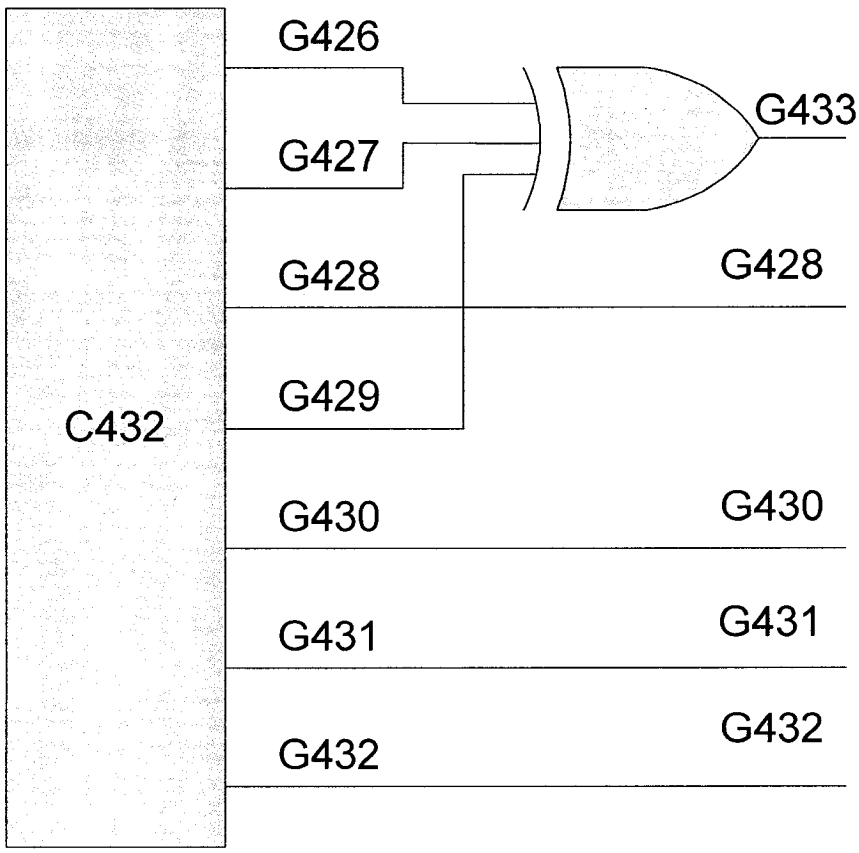


Figure 4.6 Designed Space Compactor-stage1

4.3.2 Compactor formation –Stage 2:

At stage 2, simulation experiments show that there is no strong compatible output pairs exist. Hence we have to deal with the information of simply compatible pairs of output lines. Tables 4.3 – 4.4 shows the simply compatible pairs of output lines. There was no compatible pairs of Output line using logic gate of AND type.

Table 4.3 Simply compatible Pairs for logic XOR.

(O_i, O_j)
(433, 430)
(433, 431)
(428,430)
(428, 431)
(430, 431)

Table 4.4 Simply compatible Pairs for logic OR.

(O_i, O_j)
(431, 430)

To find out the maximal compatibility classes (MCCs) we used the Paull-Unger Method. Obtained maximal compatibility classes (MCCs) as shown in Table 4.5

Table 4.5 Maximal Compatibility Classes for ISCAS85 Benchmark Circuit

MCCs(XOR/XNOR)
(430,432)
(428,432)
(433,431)
(436,430)

The above process continues until maximum compaction ratio is achieved. Figure Shows a Compaction network that achieved high fault coverage with maximal compaction ratio.

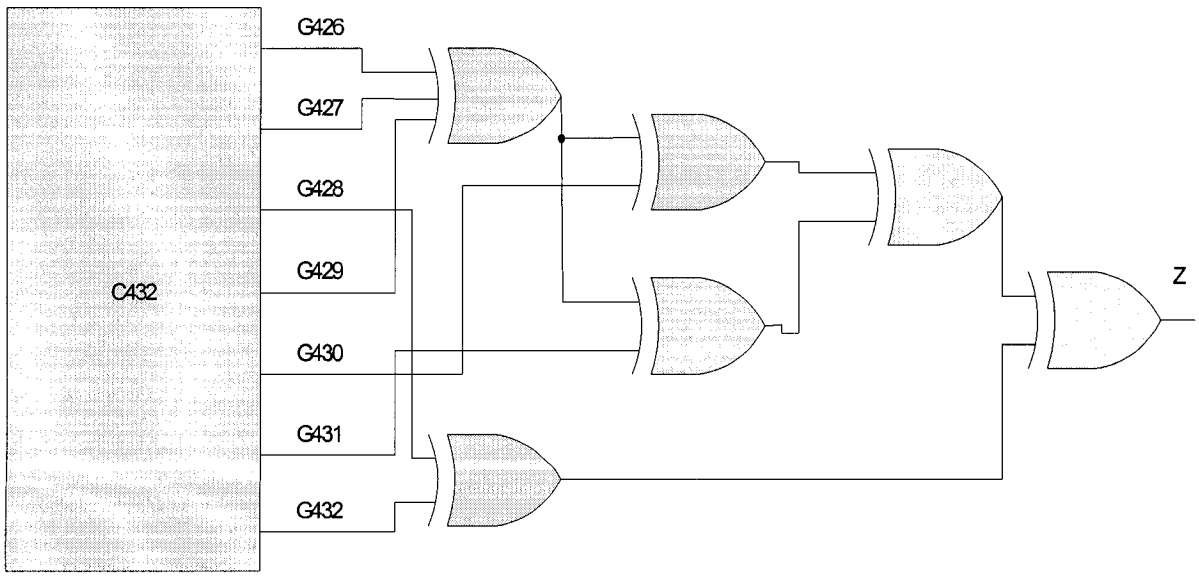


Figure 4.7 Designed Space Compactor

Chapter 5

Experimental Results

5.1 Simulation Methods

In the previous chapters, we proposed a new approach for implementing space-efficient BIST support hardware, extending the well known concepts of conventional switching theory, and of compatibility relation as employed in the minimization of incomplete sequential machines, using Paull-Unger algorithm of finding all the maximal compatibility classes (MCCs) in the design, based on optimal generalized sequence mergeability. This work utilizes mathematically sound selection criteria of merger of an optimal number of output lines of the MUT to decide on the logic for zero-aliasing, achieving maximal compaction ratio in the design, as is evident from the simulation experiments.

Extensive simulations were carried out to demonstrate the feasibility of the proposed zero-aliasing space compression scheme using ISCAS 85 combinational benchmark and ISCAS 89 sequential benchmark circuits. We used simulation programs ATALANTA [47] and FSIM [48] as our test generation tools. ATALANTA (a logic and fault simulation program developed at the Virginia Polytechnic Institute and State University) is used to generate reduced test sets those detects most of the single stuck-at-line (SSL) faults. FSIM fault simulation program is used to generate pseudorandom test sets for the circuits.

For a detailed analysis of proposed algorithm, several observation metrics were set during the simulation experiment. These include the number of test vectors to be used to construct the compaction tree, simulation time, fault coverage obtained and hardware overhead of formed compaction circuit. We determined the fault coverage before adding the compactor, fault coverage with the compactor, number of test vectors used to construct the compaction tree, simulation CPU time, hardware overhead and compaction ratio by running ATALANTA and FSIM programs respectively. The results are listed in the figures and tables in Section 5.2 for ISCAS 85 benchmark combinational circuit and in Section 5.3 for ISCAS 89 benchmark full scan sequential circuits.

For comparison purpose, we used the parity tree space compactor composed of XOR gates only. Parity tree space compactors are generally considered ideal as space compactor, since it propagates all errors appearing on an odd number of inputs. Having the parity tree space compactor as a reference we were able to study the suitability of the proposed algorithms for constructing space compaction network.

5.2 Experimental Results with ISCAS 85 Benchmark circuit

5.2.1 Experimental results without space compactors

To get a realistic estimate of the fault coverage, we first carried out the simulation with out compactor. This allowed us to determine highest achievable fault coverage for the circuit under test. In this process we also generate a list of

detectable faults for the circuit under test. In next step we perform fault simulation with only the detectable faults to achieve 100 % fault coverage.

Tables 5.1 and 5.2 show the deterministic test results with all fault injected and with detected fault injected to the ISCAS 85 benchmark combinational circuits respectively.

Table 5.1 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using ATALANTA Without Space Compactor and all Faults Injected.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	5	22	2	100
c432	51	524	7	99.237
c499	53	758	32	98.945
c880	54	940	26	100
c1355	84	1574	32	99.492
c1908	118	1879	35	99.521
c2670	111	2747	140	95.741
c3540	156	3428	22	96.004
c5315	115	5350	123	98.897
c6288	34	7744	32	99.561
c7552	212	7550	108	98.265

Table 5.2 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using ATALANTA Without Space Compactor and detectable Faults Injected

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	5	22	2	100.00
c432	51	520	7	100.00
c499	50	750	32	100.00
c880	51	940	26	100.00
c1355	84	1566	32	100.00
c1908	108	1870	35	100.00
c2670	111	2630	140	100.00
c3540	151	3291	22	100.00
c5315	115	5291	123	100.00
c6288	34	7710	32	100.00
c7552	212	7419	108	100.00

Tables 5.3 and 5.4 show the pseudorandom test results using FSIM with all fault injected and with detected fault injected to the ISCAS 85 benchmark combinational circuits respectively. We have also done the testing with compacted test vectors for both all faults injected and with detected faults injected. Tables 5.5 and 5.6 show the results with all faults injected and detected faults respectively.

Table 5.3 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM Without Space Compactor and all Faults Injected.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	32	22	2	100
c432	224	524	7	97.519
c499	224	758	32	97.625
c880	224	940	26	95.011
c1355	224	1574	32	92.440
c1908	224	1879	35	82.863
c2670	224	2747	140	82.235
c3540	224	3428	22	86.669
c5315	224	5350	123	96.374
c6288	224	7744	32	99.561
c7552	224	7550	108	90.887

Table 5.4 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM Without Space Compactor and Detected Faults Injected.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	32	22	2	100.00
c432	224	520	7	100.00
c499	224	750	32	100.00
c880	224	940	26	100.00
c1355	224	1566	32	100.00
c1908	224	1870	35	100.00
c2670	224	2630	140	100.00
c3540	224	3291	22	100.00
c5315	224	5291	123	100.00
c6288	224	7710	32	100.00
c7552	224	7419	108	99.407

Table 5.5 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM Without Space Compactor and all Faults Injected and Tested With Compacted Test Sets.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	4	22	2	100.00
c432	52	524	7	99.237
c499	54	758	32	98.945
c880	54	940	26	100.00
c1355	84	1574	32	99.492
c1908	121	1879	35	99.521
c2670	108	2747	140	95.741
c3540	148	3428	22	96.004
c5315	127	5350	123	98.897
c6288	30	7744	32	99.561
c7552	212	7550	108	98.252

Table 5.6 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM Without Space Compactor and Detected Faults Injected and Tested With Compacted Test Sets.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	4	22	2	100.00
c432	48	520	7	100.00
c499	53	750	32	100.00
c880	55	940	26	100.00
c1355	86	1566	32	100.00
c1908	119	1870	35	100.00
c2670	107	2630	140	100.00
c3540	145	3291	22	100.00
c5315	115	5291	123	100.00
c6288	37	7710	32	100.00
c7552	216	7419	108	100.00

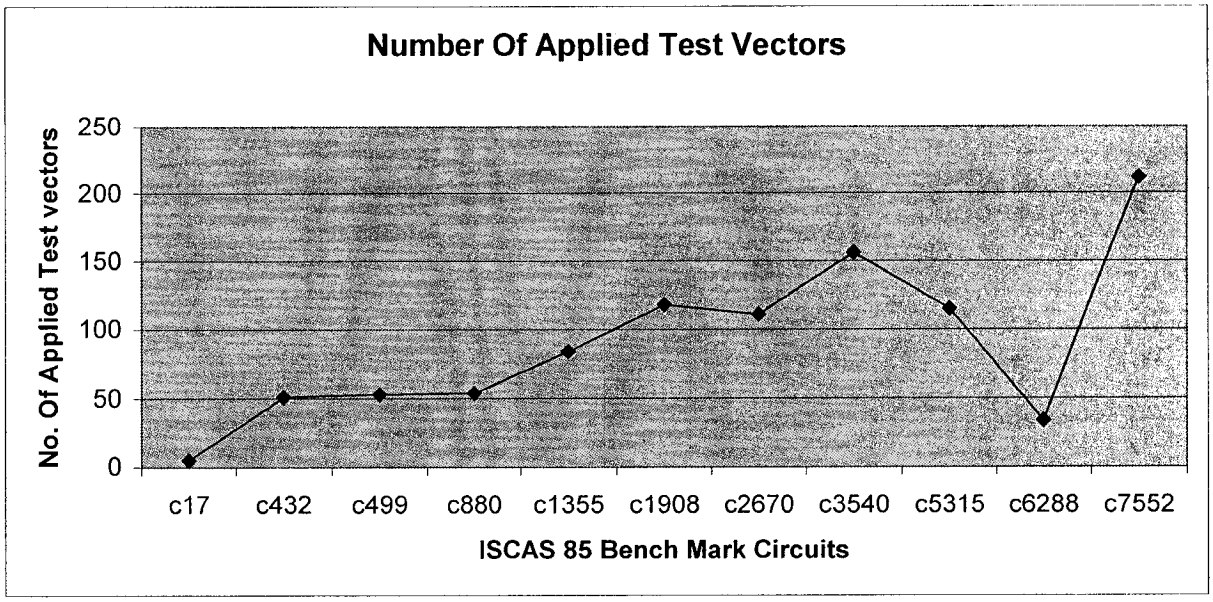


Figure 5.1 Input Test Patterns For ISCAS 85 Benchmark Circuits Using ATALANTA Without Space Compactor

Figure 5.1 shows the number of applied test patterns used to obtain the maximum fault coverage without compactor.

We also made a comparative study of simulation results obtained with FSIM and ATALANTA. The Figures 5.2,5.3 and 5.4 depicts the study.

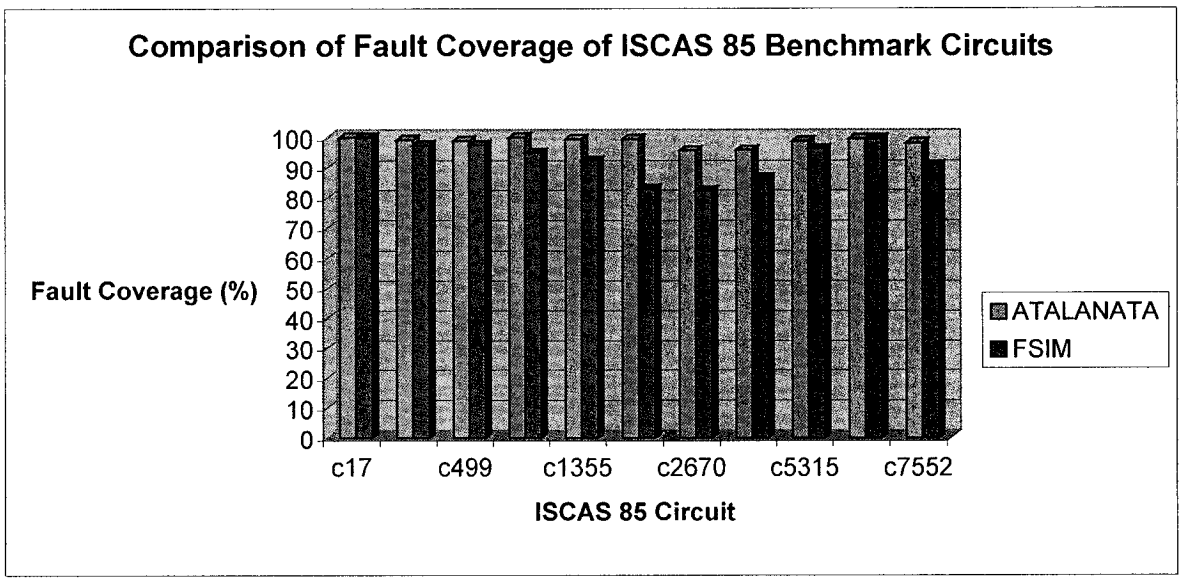


Figure 5.2 Fault Coverage For ISCAS 85 Benchmark Circuits Using ATALANTA Without Space Compactor

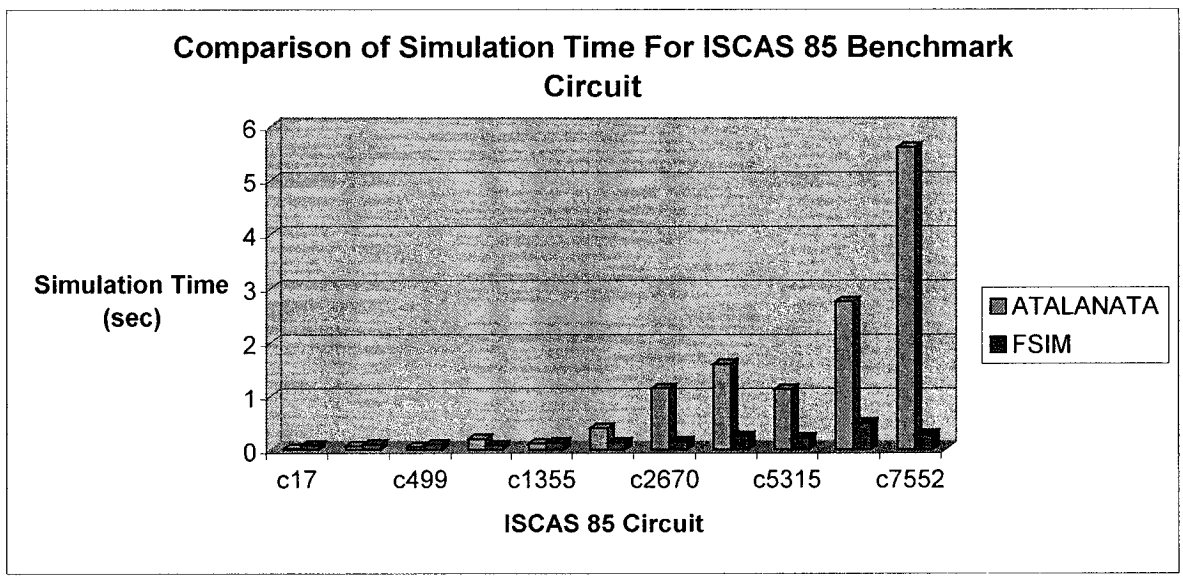


Figure 5.3 Simulation time For ISCAS 85 Benchmark Circuits Using ATALANTA Without Space Compactor

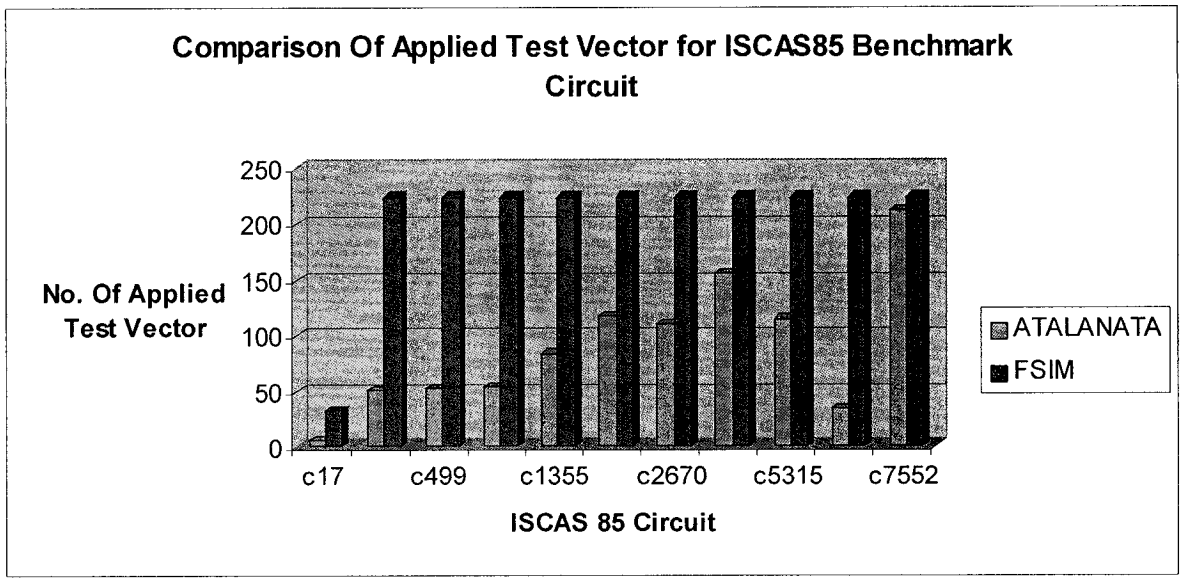


Figure 5.4 Input Test Patterns For ISCAS 85 Benchmark Circuits Using ATALANTA Without Space Compactor

5.2.2. Experimental Results With Space Compactors

The space compactor design was done based on the algorithm proposed in previous chapter. This algorithm utilizes the concept of strong and weak compatible output pairs of circuit under test. Formation of maximal compatible class was done using Paull-Unger method. The designed compactor greatly reduces the problem of aliasing with maximal compaction ratio and acceptable hardware overhead as evident from the simulation results shown in tables 5.7 and 5.8 respectively.

Table 5.7 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using ATALANTA With Space Compactor and detectable Faults Injected

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Compaction ratio	Fault Coverage (%)
c17	9	22	.033	2	1/2	100.00
c432	70	520	0.699	7	1/7	100.00
c499	56	750	4.899	32	1/32	100.00
c880	147	940	65.49	26	1/26	100.00
c1355	189	1566	4.17	32	1/32	100.00
c1908	177	1870	27.75	35	1/35	100.00
c2670	512	2630	120.20	140	1/140	100.00
c3540	109	3291	280.24	22	1/22	100.00
c5315	75	5291	55.28	123	1/123	100.00
c6288	69	7710	115.15	32	1/32	100.00
c7552	199	7419	210.63	108	1/108	100.00

Table 5.8 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM With Space Compactor and detectable Faults Injected

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Compaction ratio	Fault Coverage (%)
c17	101	22	.082	2	1/2	100.00
c432	3253	520	1.920	7	1/7	100.00
c499	3943	750	7.345	32	1/32	100.00
c880	75045	940	85.785	26	1/26	100.00
c1355	7529	1566	7.34	32	1/32	100.00
c1908	142000	1870	81.595	35	1/35	100.00
c2670	714392	2630	238.204	140	1/140	100.00
c3540	375423	3291	340.94	22	1/22	100.00
c5315	9843	5291	71.28	123	1/123	100.00
c6288	10074729	7710	725.95	32	1/32	100.00
c7552	20000045	7419	1064.695	108	1/108	100.00

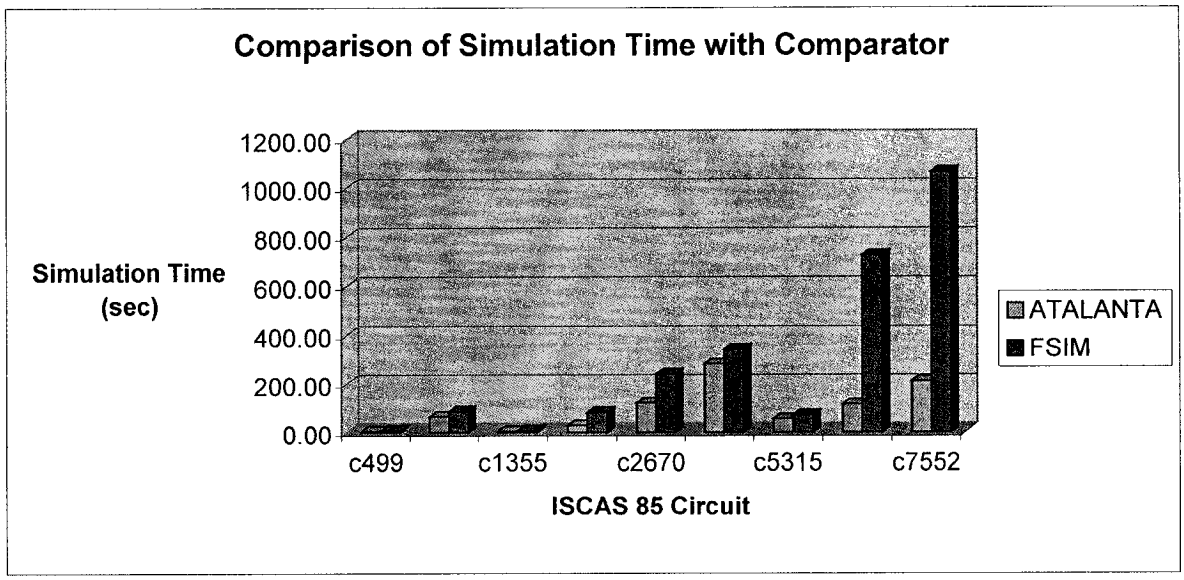


Figure 5.5 Simulation time for ISCAS 85 Benchmark Circuits Using ATALANTA With Space Compactor

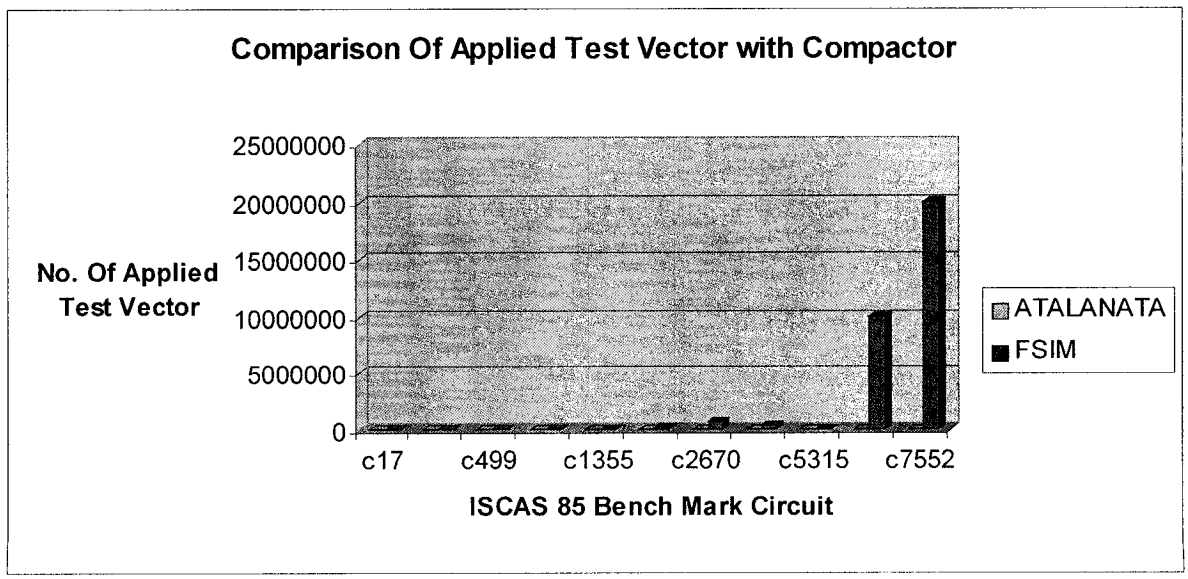


Figure 5.6 Input Test Patterns For ISCAS 85 Benchmark Circuits Using ATALANTA With Space Compactor

One of our objectives was to keep the hardware overhead low to avoid the propagation delay and to achieve area efficient hardware. The hardware overhead is calculated based on the formula below.

$$\text{Hardware Overhead} = \frac{\text{Total number of gates in the compressor X Average fanin in the compressor}}{\text{(Average fanin in the CUT) X (Total gate in CUT + Total gates in compressor)}}$$

The hardware overhead for all ISCAS 85 benchmark combinational circuits with our multiple line merger zero-aliasing compactors, using deterministic test sets is shown in Table 5.9

Table 5.9 Estimates of the Hardware Overhead for ISCAS 85 Benchmark Circuits

Circuit Name	Total No of gates in the compactor (1)	Total No of Fanin In the Comapctor	Average Fanin in the compactor	Total No of gates in the CUT (2)	Average Fanin in the CUT	(1)+(2)	Hardware Overhead (%)
c17	1	2	2.00	6	2.00	7	14.29
c432	4	8	2.00	160	2.10	164	2.32
c499	4	32	8.00	202	2.02	206	7.69
c880	5	30	6.00	383	1.9	388	4.069
c1355	14	45	3.21	546	1.95	560	4.12
c1908	8	32	4.00	880	1.70	888	2.12
c2670	10	149	14.9	1269	1.64	1279	7.103
c3540	4	25	6.25	1669	1.76	1673	0.84
c5315	20	135	6.75	2307	1.90	2327	3.053
c6288	8	39	4.75	2416	1.99	2424	0.80
c7552	11	118	10.73	3513	1.75	3524	1.88

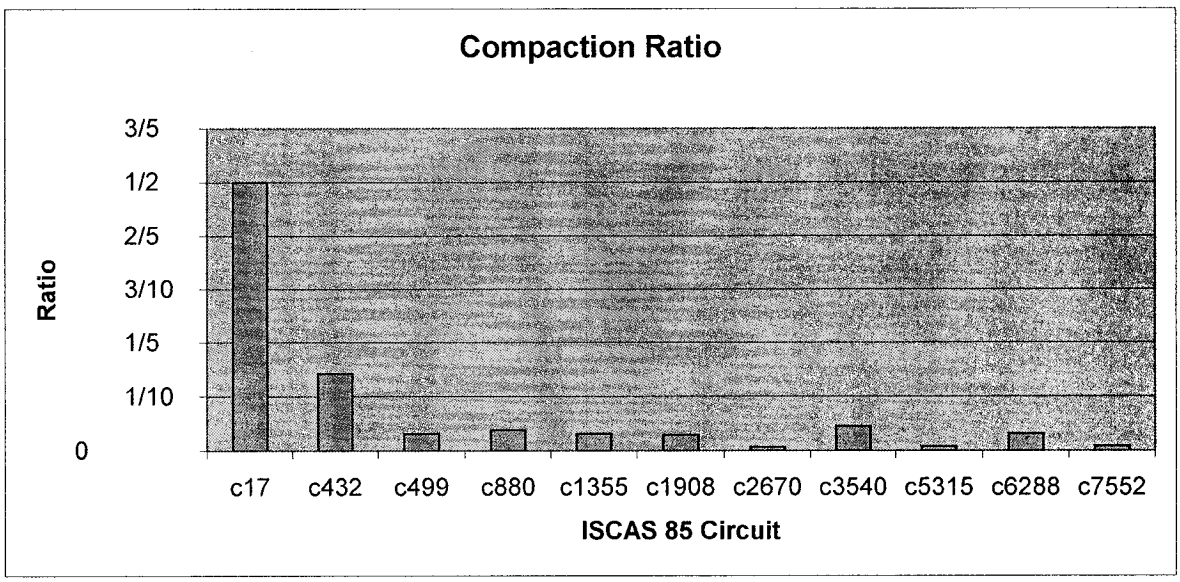


Figure 5.7 Compaction Ratio for ISCAS 85 Benchmark Circuits

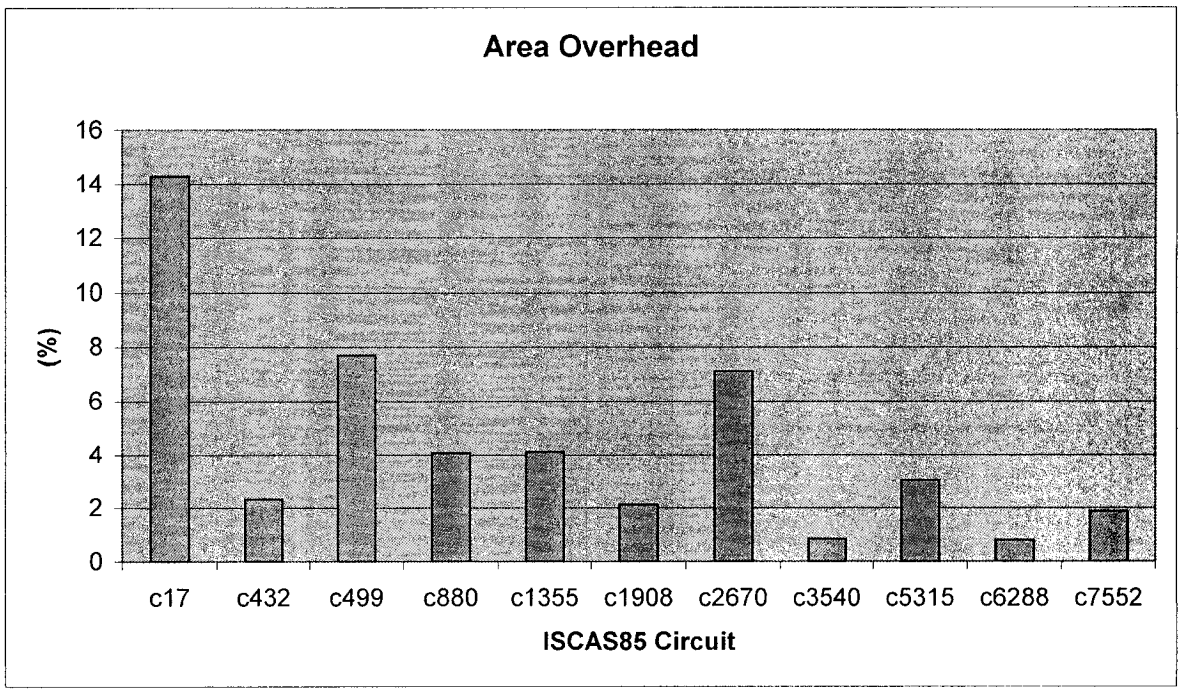


Figure 5.8 Area Overhead of Compaction Networks for ISCAS 85 Benchmark Circuits.

5.3 Experimental Results With ISCAS 89 Benchmark Scan Circuit.

In order to further demonstrate the feasibility of the proposed space compactor we have also conducted some simulation experiments on ISCAS 89 benchmark sequential scan circuits. We have used both ATALANTA and FSIM in these simulation experiments. Tables 5.10,5.11,5.12 and 5.13 shows the deterministic and pseudorandom test results with detected fault injected to the ISCAS 89 benchmark sequential scan circuits respectively.

Table 5.10 Simulation results of the ISCAS 89 Benchmark scan Circuits Using ATALANTA without Space Compactors

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Fault Coverage (%)
S27	7	32	0.000	4	100.00
S208	32	215	0.017	10	100.00
S298	35	300	0.017	20	100.00
S344	23	342	0.033	26	100.00
S349	27	350	0.033	26	99.429
S382	33	399	0.017	27	100.00

Table 5.11 Simulation results of the ISCAS 89 Benchmark scan Circuits Using FSIM with out Space Compactors

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Fault Coverage (%)
S27	64	32	0.083	4	100.00
S208	2592	215	0.067	10	100.00
S298	352	308	0.100	20	100.00
S344	416	342	0.100	26	100.00
S349	9024	350	0.083	26	99.429
S382	544	399	0.083	27	100.00

Table 5.12 Simulation results of the ISCAS 89 Benchmark Scan Circuits Using ATALANTA with Space Compactors.

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (Sec)	Compaction Ratio	Fault Coverage (%)
S27	9	32	0.015	1/4	100.00
S208	27	215	0.027	1/10	100.00
S298	33	300	0.048	1/20	100.00
S344	20	342	0.040	1/26	100.00
S349	22	348	0.039	1/26	99.429
S382	31	399	0.028	1/27	100.00

Table 5.13 Simulation results of the ISCAS 89 Benchmark Scan Circuits Using FSIM with Space Compactors

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	Compaction Ratio	Fault Coverage (%)
S27	60	32	0.023	1/4	100.00
S208	2448	215	0.205	1/10	100.00
S298	379	300	0.297	1/20	100.00
S344	476	342	0.342	1/26	100.00
S349	11429	348	0.329	1/26	99.429
S382	549	399	0.122	1/27	100.00

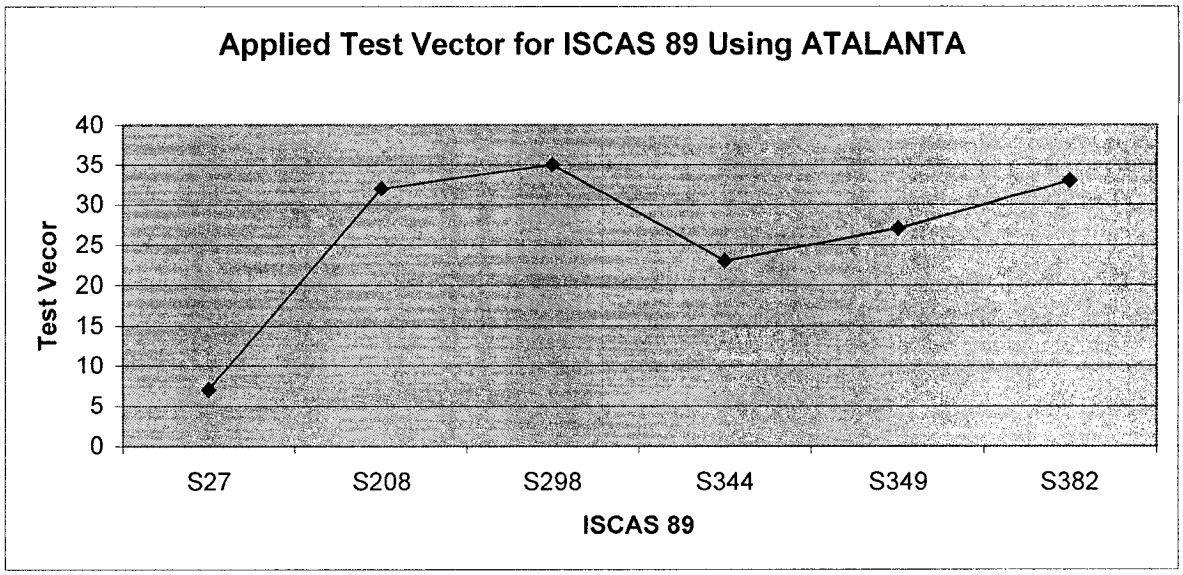


Figure 5.9 Input Test Patterns For ISCAS 89 Benchmark Circuits using ATALANTA Without Space Compactor

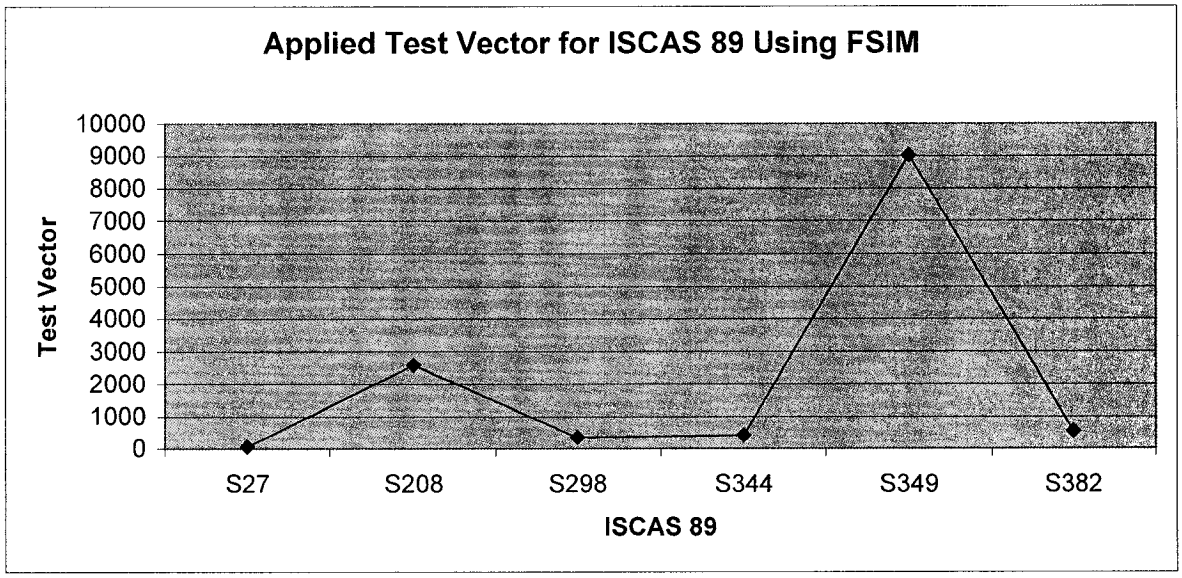


Figure 5.10 Input Test Patterns For ISCAS 89 Benchmark Circuits using FSIM Without Space Compactor

5.4 Comparison of Proposed Approach With Parity Tree Space Compactor.

For Comparison purpose, We also used the parity tree space compactor composed of XOR gates, which propagates all errors appearing on an odd number of inputs and is therefore, considered ideal for space compaction. The Simulation results of parity tree as space Compactor with ISCAS 85 benchmark circuits using ATALANTA and FSIM simulators are given next.

Table 5.14 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using ATALANTA With Parity tress as Space Compactor

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Comp. ratio	Fault Coverage (%)
c17	6	22	0.000	2	1/2	100
c432	65	520	0.117	7	1/7	100
c499	53	750	0.200	32	1/32	96.800
c880	50	940	0.533	26	1/26	99.294
c1355	85	1566	1.017	32	1/32	98.044
c1908	143	1870	0.750	35	1/35	99.251
c2670	108	2630	30.867	140	1/140	70.34
c3540	175	3291	2.367	22	1/22	95.101
c5315	122	5291	5.867	123	1/123	98.659
c6288	46	7710	5.933	32	1/32	99.961
c7552	208	7419	64.933	108	1/108	94.977

Table 5.15 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM With parity tree as Space Compactor

Circuit Name	Applied Test Vectors	No. of Fault Injected	Simulation Time (sec)	No. Of Output	Comp. ratio	Fault Coverage (%)
c17	32	22	0.083	2	1/2	100
c432	224	520	0.117	7	1/7	96.346
c499	224	750	0.100	32	1/32	94.800
c880	224	940	0.083	26	1/26	93.952
c1355	224	1566	0.117	32	1/32	89.731
c1908	224	1870	0.150	35	1/35	80.802
c2670	224	2630	0.283	140	1/140	67.603
c3540	224	3291	0.300	22	1/22	83.285
c5315	224	5291	0.517	123	1/123	95.138
c6288	224	7710	0.817	32	1/32	99.500
c7552	224	7419	0.967	108	1/108	87.816

Table 5.16 Estimates of the Hardware Overhead for ISCAS 85 Benchmark Circuits With Parity Tree as Space Compactor

Circuit Name	Total No of gates in the compactor (1)	Total No of Fanin In the Compactor	Average Fanin in the compactor	Total No of gates in the CUT (2)	Average Fanin in the CUT	Hardware Overhead (%)
c17	1	2	2.00	6	2.00	14.29
c432	6	12	2.00	160	2.10	3.44
c499	31	62	2.00	202	2.02	13.19
c880	25	50	2.00	383	1.9	6.42
c1355	31	62	2.00	546	1.95	5.50
c1908	24	48	2.00	880	1.70	3.10
c2670	139	278	2.00	1269	1.64	11.78
c3540	21	42	2.00	1669	1.76	1.40
c5315	122	244	2.00	2307	1.90	5.27
c6288	31	62	2.00	2416	1.99	1.27
c7552	107	214	2.00	3513	1.75	3.36

To have an overall view, we compared the fault coverage and hardware overhead between proposed compactor and parity tree compactor as shown in Figure 5.11 and 5.12 respectively.

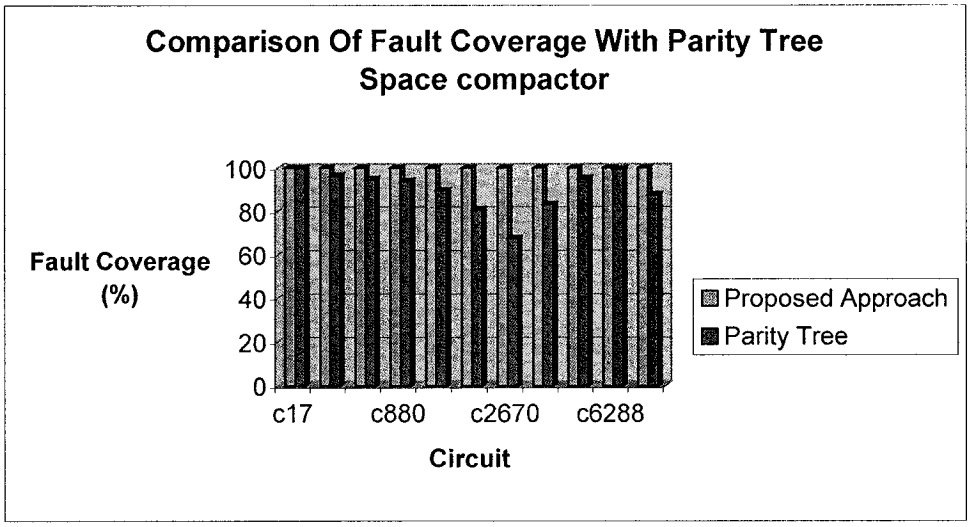


Figure 5.11 Comparison Of Fault Coverage between Proposed approach and Parity tree Space Compactor

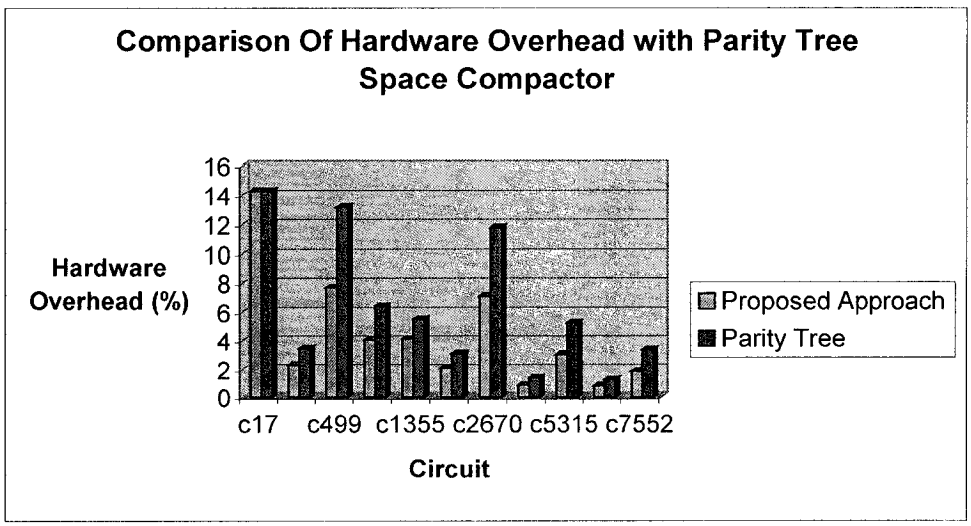


Figure 5.12 Comparison Of Hardware Overhead between Proposed approach and Parity tree Space Compactor

5.5 Summary

In this chapter we provided the results of our simulation experiments on both ISCAS 85 and ISCAS 89 circuits using simulators like FSIM and ATALANTA. We also conducted some simulation experiments on space compactors formed with parity tree as compaction method to conduct a comparative study with the proposed algorithm.

From extensive simulation and comparative study, the advantage of the proposed algorithm is evident in terms of fault coverage and hardware overhead. For most of the benchmark circuits we obtained better fault coverage with reduced CPU simulation time and hardware overhead using our space compactor than what we have using parity tree space compactor.

Chapter 6
Conclusions And Future Work

6.1 Conclusions

This Thesis Considers the general problem of designing and analyzing efficient space compaction techniques for built in self testing of VLSI circuits using compact test sets. The extra logic representing the compression circuit must be as simple as possible, to be easily embedded within the CUT, and should not introduce signal delays to affect the test execution time or normal functionality of the circuit. Besides, the length of the signature must be as short as possible in order to minimize the amount of memory needed to store the fault-free responses. Furthermore, the signatures derived from faulty output responses and their corresponding fault-free signatures should not be the same, which unfortunately is not always the case. A fundamental problem with compression techniques is error masking or aliasing, which occurs when the signatures from faulty output responses map into fault free signatures. Aliasing causes loss of information, which in turn affects the test quality of BIST, reducing the total fault coverage.

This thesis reports on developing zero-aliasing compression techniques of test data outputs for digital integrated circuits that facilitate the design of space efficient BIST support hardware using pseudorandom and compact test sets. Specifically, utilizing the concept of fault graded output merger based on the information of strong or simply compatible output pairs, this thesis proposes a space compactor design method for use in SOC digital integrated circuits. This proposed techniques use

AND (NAND), OR (NOR), and XOR (XNOR) gates as appropriate to construct an output compaction tree that compresses the outputs of the MUT to ideally a single output line. The suggested compaction technique, as evidenced by experiments on ISCAS 85 combinational benchmark circuits and ISCAS 89 Circuits, using fault simulation programs like FSIM and ATALANTA, provides 100% Fault coverage for single stuck-line faults yielding zero aliasing, with low CPU simulation time, and acceptable area overhead for the compactor.

The advantages of proposed method of space compression are the following. It is simple, has low area overhead, has high or full fault coverage for single stuck-line faults, and requires low CPU simulation time. This may be suitable for VLSI design environment as BIST supported hardware. Design algorithms are proposed in this thesis, and the implementations are demonstrated with some examples. Space compactors have been designed for all ISCAS 85 benchmark combinational and some ISCAS 89 scan sequential benchmark circuits and tested with ATALANTA and FSIM to verify the feasibility and usefulness of the suggested approach.

6.2 Future Research

The success of the proposed algorithm in designing aliasing free space efficient hardware for BIST of System On a Chip Circuit opens new avenues for further research. For future research, we like to make some suggestions which will produce better results in terms of, design efficiency, effective design, more reduction in hardware overhead, less test patterns, and less testing time with better results.

For efficient and effective multiple line merger, better algorithms can be developed for circuit under test.

Design process took longer time for circuits consist of large number of gates, also there was a limitation of using complex algorithm. So use of better and faster computational resources to speedup the design process.

A heuristic approach is adopted for generalized mergeability to get a result within an acceptable CPU time. The performance of the design process may be improved by using better and more efficient heuristics.

Bibliography

[1] R. Rajsuman, *System-on-a-Chip: Design and Test.* Boston, MA: Artech House, 2000.

[2] M. H. Assaf, "Digital Core Output Test Data Compression Architecture Based on Switching Theory Concepts", *Ph.D. Dissertation*, School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada., 2003.

[3] S. R. Das, C. V. Ramamoorthy, M. H. Assaf, E. M. Petriu, and W. B. Jone, "Fault tolerance in systems design in VLSI using data compression under constraints of failure probabilities", *IEEE Trans. Instrum. Meas.*, vol.50, pp. 1725-1747, December 2001.

[4] M. Seuring, and K. Chakrabarty, "Space compaction of test responses for IP cores using orthogonal transmission functions", *Proc. IEEE VLSI Test Symp.*, pp. 213-219, 2000.

[5] S. R. Das, T. Barakat, E. M. Petriu, M. H. Assaf, and K. Chakrabarty, "Space compression revisited", *IEEE Trans. Instrum. Meas.*, vol. 49, pp. 690-705, June 2000.

[6] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-in Test for VLSI: Pseudo random Techniques.* John Wiley, New York, 1987.

[7] S. Mourad and Y. Zorian, *Principles of Testing Electronic Systems.* New York, NY: Wiley 2000.

[8] K. Chakrabarty, "Test Response Compaction for Built-In Self-Testing", *Ph.D. Dissertation*, Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, 1995.

[9] S. R. Das, M. H. Assaf, E. M. Petriu, W. -B. Jone, and K. Chakrabarty, "A novel approach to designing aliasing-free space compactors based on switching theory formulation", in *Proc. IEEE Inst. Meas. Tech. Conf.*, pp. 198-203, 2001.

[10] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core Based System ICs", *IEEE Communication Magazine*, pp. 104-109, June 1999.

[11] Y. Zorian, E. J. Marnissen, S. Dey, "Testing Embedded Core based System Chips", *IEEE Int. Test Conf. (ITC)*, Washington DC, IEEE computer Society press, pp. 130-143, October 1998.

[12] G. Jerven, Z. Peng, Raimul Ubar, Helena Kruus, "A Hybrid BIST Architecture and its optimization for SoC Testing", *IEEE Proc. Int. Symp. Quality Electronic Design*, pp. 273-279, 2002.

[13] T. Yoneda, and Fujiwarna, H., A DFT Method for Core-based System-on-a-chip Based on Consecutive Testability", *Proc. Asian Test Symp.*, pp. 193-198, Kyoto, Japan, 2001.

[14] Rosinger, P. Gonciari, P.T. Al-Hashimi, B. M. and Nicolici, N., "Simultaneous Reduction in Volume of Test Data and Power Dissipation for System-on-a-chip", *IEE Electronics Letters*, pp. 1434-1436, Vol. 37, Issue 24, 22 Nov. 2001.

[15] G. Jervan, Z. Peng, and R.Ubar, "Test Cost Minimization for Hybrid BIST", *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 283-291, Yamanashi, Japan, 2000.

[16] K. K. Saluja and M. Karpovsky, "Testing computer hardware through data compression in space and time", *Proc. Int. Test Conf.*, pp.83-88, 1983.

[17] Y. K. Li and J. P. Robinson, "Space Compression Methods With Output Data Modification", *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 290-294, March, 1987.

[18] W.-B. Jone and S. R. Das, "Space compression method for built-in self-testing of VLSI circuits", *Int. Journal of Computer-Aided Design*, vol. 3, pp. 309-322, September 1991.

[19] Y. Zorian and V. K. Agarwal, "A general scheme to optimize error masking in built-in self-testing", *Proc. 1986 Int. Symp. Fault-Tolerant Computing*, pp. 410-415, 1986.

[20] M. Karpovsky and P. Nagvajara,"Optimal robust compression of test responses", *IEEE Trans. Computers,* vol. c-39, pp. 138-141, January 1990.

[21] K. Chakrabarty and J. P. Hayes, "Cumulative balance testing of logic circuits", *IEEE Trans. VLSI Systems*, vol. 3 , Issue 1, pp. 72-83, March 1995.

[22] M. Serra, and J. C. Muzio, "Space Compaction for Multiple-output Circuits", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* pp.1105-1113, Issue 10, Oct. 1988.

[23] K. Chakrabarty and J. P. Hayes, "Zero-aliasing Space Compaction of Test Responses using Multiple Parity Signature", *IEEE Trans. VLSI Systems,* pp. 309-313, June 1998.

[24] Jin Ding, D. Moloney, and Xiaojun Wang, "Aliasing-free Space and Time Compaction with Limited Overhead", *Proc. IEEE First Int. Symp. Quality Electronic Design,* pp. 355-360, San Jose, CA, USA, March 2000.

[25] J.P. Hayes, "Checksum methods for test data compression", *Journal of Design Automation and Fault-Tolerant Computing*, vol. 1, pp. 3-7, January 1976.

[26] S. R. Das, H. T. Ho, W. B. Jone, and A. R. Nayak, "An improved output compaction modification technique for built-in self-test in VLSI circuits", *Proc. 1994 Int. Conf. VLSI Design*, pp. 403-407, 1994.

[27] J. Savir, "Syndrome-testable design of combinational circuits", *IEEE Trans. Computers,* vol. C-29, pp. 442-451, June 1980.

[28] J. P. Hayes, "Transition count testing of combinational logic circuits", *IEEE Trans. Computers,* vol. C-25, pp. 513-620, June 1976.

[29] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-in Test for VLSI: Pseudorandom Technique*. John Wiley, New York, 1987.

[30] S. B. Akers, "A parity bit signature for exhaustive testing", *IEEE Trans. Computer Aided Design,* vol. 7, pp. 333-338, March 1988.

[31] A. K. Susskind, "Testing by verifying Walsh coefficients", *Proc. Int. Symp. Fault-Tolerant Computing,* pp. 206-208, 1981.

[32] E. J. McCluskey, "Built-in self-test techniques", *IEEE Design & Test of Computers*, vol. 2, pp. 21-28, April 1985.

[33] J. Savir, "Reducing the MISR size", *IEEE Trans. Computers*, vol. C-45, pp. 930-938, August 1996.

[34] S. R. Das, M. Sudarma, E. M. Petriu, M. H. Assaf, and W. B. Jone, "Parity Bit Signature in Response Data Compaction and Built-in Self-Testing of VLSI Circuits with Nonexhaustive Test Sets", *43rd Midwest Symp. Circuits and Systems*, Lansing, Michigan, August 2000.

[35] S. R. Das, Chuan Jin, Liwu Jin, M. H. Assaf, Emil M. Petriu, W. B. Jone and Mehmet Sahinoglu, "Implementation of Testing environment for digital IP cores", *IEEE Int. and Meas. Conf.*, Como, Italy, pp. 18-20, May 2004.

[36] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Digital System Testing and Testable Design", *Computer Science Press*, New York, 1990.

[37] Naim Ben-Hamida, Khaled Saab, David Marche, and Bozena Kaminska, "FaultMaxx: A Perturbation Based Fault Modeling and Simulation for Mixed Signal Circuits", *Proc. of the 10th Anniversary Compendium of Papers from Asian Test Symp.*, pp. 189, 2001.

[38] Hussain Al-Assad, Raymond Lee, "Simulation-Based Approximate Global Fault Collapsing", Department of Electrical & Computer Engineering, University of California, 2002.

[39] S. R. Das, E. M. Petriu, T. Barakat, M. H. Assaf, and A. R. Nayak, "Generalized detectable error probability estimate and output data compression under multiplicities of error—Mathematical analysis," *Proc. 3rd*

World Conf. Integrated Design and Process Technology, Berlin, Germany, vol. 6, pp. 92–99. 1998.

[40] S. R. Das, T. Barakat, A. R. Nayak, and M. H. Assaf, "Generalized mergeability in space compressor design in built-in self-test of VLSI circuits,"in *Proc. IEEE Inst. and Meas. Technology Conf.*, vol. 2, pp. 1448–1453, 1997.

[41] S. R. Das, Emil M. Petriu, T. Barakat, M. H. Assaf, and A. R. Nayak ," Space compaction under generalized mergeability", *IEEE Trans. Inst. and Meas.*, vol. 47, pp 1283 – 1293, October 1998.

[42] Y. K. Li, and J. P. Robinson, "Space compression methods with output data modification", *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 290–294, 1987.

[43] S. R. Das, C. L. Sheng, Z. Chen, and T. Lin, "Magnitude ordering ofdegree complements of certain node pairs in an undirected graph and an algorithm to find a class of maximal subgraphs", *Computers Electrical Engineering*, vol. 6, pp. 139-151, Sep. 1979.

[44] S. Y. Lee, "On a Family of Novel Clique Detection Algorithms and Some Studies on Their Performance Characteristics and Order of Complexities in the Context of Existing Algorithms", *Ph.D. Dissertation, Department of Electronics*, National Chiao Tung University, Hsinchu, Taiwan, ROC, 1982.

[45] S. R. Das, S. Y. Lee, Z. Chen, S. M. Wu, and T. Lin, "On the Complexity and Performance Evaluation of a Novel Clique Detection Algorithm in Undirected Graphs", *Int. Journal,Policy and Information*, Vol. 4, pp. 21-32, 1980.

[46] S. R. Das, Altaf Hossain, E. M. Petriu, M. H. Assaf, Mehmet Sahinoglu, W. - B. Jone, "On a New Graph Theory Approach to Designing Zero-Aliasing Space Compressors for Built-In Self-Testing" *Proc. of Inst. and Measurement Tech. Conf.*, pp. 1890 – 1895, 2006.

[47] H. K. Lee, and D. S. Ha, "An Efficient Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation", *Proc. of the Int. Test Conf.*, pp. 946-955, 1991.

[48] H. K. Lee, and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits", *Technical Report 12-93,* Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 1993.

[49] S. R. Das, M. H. Assaf, J. Liang, E. M. Petriu and W. B. Jone, "A New Apprisal of Response Compaction Efficiency in SoC Based on Detectable Single Stuck Line Fault Subject to Complete or Near Complete Set of Tests Using Nth Order Missed Error Probability Estimates", *IASTED Int. Conf. Modeling, Identification and Control*, pp. 913-918, Innsburk, Austria, Feb., 2001.

[50] Abramovici, M., Breuer, M., and Friedman, A., *Digital Systems Testing andTestable Design*, IEEE Press, New York, NY., 1990.

[51] Agrawal, V.D., "Sampling Techniques for Determining Fault Coverage in LSI Circuits," *J. Digital Systems*, vol. 5, Sept 1981, pp. 189-202.

[52] Agrawal, V.D., Seth, S.C., and Agrawal, P., "Fault Coverage Requirement in Production Testing of LSI Circuits," *IEEE Journal of Solid-State Circuits,* vol. SC-17, no. 1, Feb 1982, pp. 57-61.

[53] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self-test—Part II," *IEEE Des. Test Comput.*, vol. 10, pp. 69–77, June 1993.

[54] P. H. Bardell,W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Technique*. New York: Wiley, 1987.

[55] Mansour H. Assaf, Sunil R. Das, Emil M. Petriu, Liwu Jin, Chuan Jin, Dhruv Biswas, Voicu Groza, and Mehmet Sahinoglu,"Hardware and Software Co-Design in Space Compaction of Cores-Based Digital Circuits" IMTC 2004 - Instrumentation and Measurement Technology Conference, Corno, Italy, 18-20 May 2004.

[56] E.B. EICHELBERGER and T.W.Williams, "A logic design structure for LSI testability", *Proc.1997 Design Automation Conference*, pp. 462-468,1977.

[57] R. G. Daniels and W. B. Bruce, "Built-in self-test trends in Motorola microprocessors," *IEEE Des. Test Comput.*, vol. 2, pp. 64–71, Apr. 1985.

[58] Internaltional Technology Road map For semiconductors, 1999 Edition

[59] Alexander Miczo, Digital Logic Testing and Simulation ,Second Edition,Wiley-interscience, a john Willey & sons, Inc publications.

[60] H.Fujiwara and A. Yamamoto."Parity-scan design to reduce the cost of test application". IEEE transaction on computer Aided Design,Vol.12.pp.1604-1611,October 1993.

[61] Y. Zorian and Ivanov,"Programmable Space compaction for BIST", Proc.1993 Int.Symp on Fault Tolerent Computing,pp.340-349,1993.

[62] T.-C. Hsiao and S. C. Seth, "An analysis of the use of Rademacher–Walsh spectrum in compact testing," IEEE Trans. Comput., vol. C-33, pp. 934–937, Oct. 1984.

[63] R. A. Frohwerk, "Signature analysis—A new digital field service method," *Hewlett-Packard J.*, vol. 28, pp. 2–8, May 1977.

[64] T. W. Williams, W. Daehn, M. Gruetzner, and C. W. Starke, "Aliasing errors in signature analysis registers," *IEEE Des. Test Comput.*, vol. 4, pp. 39–45, Apr. 1987.

[65] N. R. Saxena and J. P. Robinson, "A unified view of test response compression methods," *IEEE Trans. Comput.*, vol. C-36, pp. 94–99, Jan. 1987.

[66] D. K. Pradhan and S. K. Gupta, "A new framework for designing and analyzing BIST techniques and zero aliasing compression," *IEEE Trans. Comput.*, vol. C-40, pp. 743–763, June 1991.

[67] Eldred, R.D., "Test Routines Based on Symbolic Logic Statements", *J. ACM*, vol.6, no. 1, Jan 1959, pp. 33-36.

[68] T. W. Williams, "Design for testability," in *Computer Design Aids for VLSI circuits*, NATO AIS series, The Netherlands: Martinus Nijhoff Publishers,pp.359-416,1986.

[69] M. C. Paull and S.H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions," *IRE Transactions on Electronic Computers EC-8*, NO. 3, 356-367 (1959).

[70] Internaltional Technology Road map For semiconductors, 2005 Edition

List Of Publications By The Candidate

1. Sujoy Mukherjee, "Space Compaction for Embedded Cores-Based System-on-Chips (SOCs) Using Fault Graded Output Merger", IMTC 2007 – Instrumentation and Measurement Technology Conference Warsaw, Poland 1-3 May 2007.(Accepted)

   (With Sunil R. Das , Emil M. Petriu, Mansour H Assaf, Altaf Hossain)

2. Sujoy Mukherjee," An Improved Fault Simulation Approach Based on Verilog With Application to ISCAS Benchmark Circuits" IMTC 2006 – Instrumentation and Measurement Technology Conference Sorrento, Italy 24-27 April 2006.

   (With Sunil R. Das, Emil M. Petriu, Mansour H. Assaf, Mehmet Sahinoglu, and Wen-Ben Jone)

3. Sujoy Mukherjee, "Design of Aliasing Free Space Compressor in BIST with Maximal Compaction Ratio Using Concepts of Strong and Weak Compatibilities of Response Data Outputs and Generalized Sequence Mergeability", IWDC 2002: 234-245

   (with Sunil R. Das, Mansour H. Assaf, Emil M. Petriu)