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

Md. Altaf Hossain

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

On a New Graph Theory Approach to Designing Zero-aliasing Space Compressors for Built-in Self-testing

TITRE DE LA THÈSE / TITLE OF THESIS

S. Das

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

E. Petriu

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

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

A. Nayak

T. Kwasniewski

Gary W. Slater

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

ON A NEW GRAPH THEORY APPROACH TO DESIGNING ZERO-ALIASING SPACE COMPRESSORS FOR BUILT-IN SELF-TESTING

Md. Altaf Hossain

A Thesis submitted to the Faculty of Graduate Studies and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Master of Applied Science, Electrical Engineering

August 2006

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, Canada

Your file Votre référence
ISBN: 978-0-494-25785-2
Our file Notre référence
ISBN: 978-0-494-25785-2

ACKNOWLEDGEMENTS

This write up is to express the result of the research carried out by the author in the field of test data compression in space, at the School of Information Technology and Engineering of Ottawa, Ottawa, Ontario, Canada. The research work consists of implementation of various algorithm for the generation of space compaction trees of core based digital integrated circuits.

The author takes this opportunity to express his gratitude to his supervisor, **Dr. Sunil R. Das**, Professor Emeritus at the School of Information Technology and Engineering, University of Ottawa, for his valuable guidance, continuous supervision and encouragement during the progress of the research. His suggestions and ideas would be remembered for life.

The author is also thankful to his co-supervisor, **Dr. Emil M. Petriu**, Professor and University Research Chair of the School of Information Technology and Engineering, University of Ottawa for his constant support and encouragement.

I am very much thankful to **Dr. Satyendra Biswas,** Asst. Professor, Electrical Engineering Technology, Georgia Southern University, who has been constantly helping me throughout my work. His valuable support was a great help for me.

I am also thankful to **Dr. Mansoor Assaf**, who has been constantly guiding and helping me throughout my work. His timely support and clarification on my related research topics was great help.

August 2006

Md.Altaf Hossain
School of Information Technology and Engineering
University of Ottawa
800 King Edward Avenue
Ottawa, Ontario K1N 6N5, Canada

ABSTRACT

Built-in self testing (BIST) schemes that compress the test responses from k-output circuit to q signature streams, $q << k$, are known as space compactors. In this thesis, we use compression techniques in digital core based systems to facilitate deterministic or pseudorandom testing. Our objective is to minimize the storage requirement of the module under test (MUT) and also to obtain the maximum fault coverage. The objective is to achieve the same fault coverage as obtained without the compactors. We use some well known switching theory techniques, like cover table and frequency ordering, to the compression schemes. Clique detection algorithm is used to find out the maximal compatibility classes (MCCs) of the MUT outputs. In designing zero-aliasing space compressors, the concept of strong and weak compatibilities of response approach is considered, and in most cases maximal compaction is achieved.

The techniques used in this thesis are simple designs that have high or full fault coverage for single stuck- line faults and acceptable area overhead. We used ISCAS 85 combinational benchmark circuits and ISCAS 89 sequential scan circuits with ATALANTA and FSIM simulation programs to simulate the suggested approaches.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF MATHEMATICAL SYMBOLS

$\{\psi_1, \psi_2\}$	Output Sequence Pair
$\{\psi_i, \psi_j\}$	Derived Sequence Pair
ω_1 :	The 1-weight
ω_0 :	The 0-weight
L_s :	Output Sequence Length
ω_{11} :	First-order 1-weight
ω_{10} :	First-order 0-weight
ω'_{N1} :	Nth-order 1-weight of derived sequence
ω'_{N0} :	Nth order 0-weight of derived sequence
R :	Number of bit positions or residue
E_{max} :	Maximum possible errors
ε :	Detectable error probability estimate
S_1 :	Probability of single error felt at the output of the CUT
S_2 :	Probability of double error felt at the output of the CUT
B_1 :	Number of single line errors at the output of gate
B_2 :	Number of double line errors at the output of Gate
H_s :	Hamming distance
ε_N :	Nth-order detectable error Probability estimate
ε_N (OR/NOR) :	Nth- order detectable error probability estimate when OR/NOR gate is used

ε_N (AND/NAND) : Nth- order detectable error probability estimate when AND/NAND gate is used

ε_N (XOR/XNOR) : Nth- order detectable error probability estimate when XOR/XNOR gate is used

ACRONYMS

AFC	:	Average fanin in Compressor
AFM	:	Average fanin in MUT
AMB	:	Advance Microcontroller Bus Architecture
AOP	:	Area Overhead percentage
ASIC	:	Application Specific Integrated Circuit
ATE	:	Automatic Test Equipment
ATV	:	Applied Test Vectors
CR	:	Compaction Ratio
CRC	:	Cyclic Redundancy Check
CST	:	CPU Simulation Time in Second
CUT	:	Circuit Under Test
DFT	:	Design for Testability
DTC	:	Double Transition Count
LFSR	:	Linear Feed back Shift Register
MISR	:	Multiple Input Signature Register
MTC	:	Multiple Transition Count
MUT	:	Module Under Test
NAI	:	No. of AND Incompatibles
NAM	:	No. of AND MCC
NFC	:	No. of fanin in Compressor
NFI	:	No. of Fault Injected
NGC	:	No. of Gates in Compressor
NGM	:	No. of Gates in MUT
NIL	:	No. of Input line In MUT
NOAC	:	No. of Output With Compactor
NOBC	:	No. of Outputs before compaction
NOI	:	No. of OR Incompatibles
NOL	:	No. of O/P Line in MUT
NOM	:	No. of OR MCC
NXI	:	No. of XOR Incompatibles
NXM	:	No. of XOR MCC
PDA	:	Personal Digital Assistant
PLA	:	Programmable Logic Array
ROM	:	Read Only Memory
SSL	:	Single Stuck Line Fault
TPG	:	Test Pattern Generator
VLSI	:	Very Large Scale Integration

CHAPTER 1
INTRODUCTION

The modern world of automation is the outcome of the electronics, revolutionized by System-on-a-Chip (SoC). The technological boom of 21st century depends on these super chips. SoC is the heart of next generation multimedia devices, computer graphics chipsets, cell phone, video phones, and robotics. The use of SoCs is predicted to grow up to 85% within next three years. The SoC is the integration of digital, analog and memory circuits on a single piece of silicon [1]. It can be defined as an IC, designed by stitching together multiple standalone VLSI designs to provide full functionality for an application.

The use of SoC is increasing tremendously and no doubt making the life easier. The only change is with the IC manufacturer who will now produce SoCs instead of discrete devices occupying the circuit boards [2]. Sub-systems are in the form of modules on a single chip. The SoC designs are primarily characterized three forms [1]:

1. ASIC vendor design, which refers to the design in which all the components in the chip are designed and fabricated by an ASIC vendor.
2. Integrated design, which refers to a design by an ASIC vendor but not all components are designed by that vendor. It implies the use of one or multiple cores obtained from some other source such as core/IP vendor or from a foundry. The fabrication of these designs may be done by ASIC vendor or foundry company.
3. Desktop design, which refers to the designs by a company that uses cores, which for the most parts have been obtained from other IP companies, EDA companies, design services companies, or a foundry. In most of the cases, an independent foundry company fabricates these designs.

Due to increasing integration of cores and the use of embedded software in SoC, the design complexity of SoC has increased dramatically; also, last minute adjustment to the performance of the product may be difficult for the customer and the manufacturer. To start again after correction would result in millions of dollars in lost opportunities, may also miss the product cycle. By using SoC manufacturer can reduce the size of the product, number of components and as well as the over all power consumption of the product, and maximize the reliability.

Fundamental element of the SoC is the microprocessor [1]. Various blocks of the embedded random logic, memory and mixed signal circuitry is added depending on the type of application [3]. Intellectual Property (IP) blocks are mixed with IP cores from various sources for faster designing of the product, which may cut down the design cycle time from a year to months or even weeks. Unfortunately, due to increasing integration of cores, the design complexity of SoC has increased.

With the increasing demand of SoC and miniaturization, the main challenge is SoC's complexity and test cost and its accuracy becomes an issue with the manufacturer. The testability of the multifunction SoCs by single insertion must also be considered. In general, for manufacturing test, SoCs face many challenges such as high volume of test storage, long test application time through the serial paths and vector sets that are generated by third party (IC core providers) with limited information. This becomes more complicated with the high-density integration.

Testing a chip is very time consuming as it could take up to one-half of the total design time [4]. The amount of time available for manufacturing, testing, and marketing a product is decreasing [1]. As the result of global competition, customers demand lower cost and better quality products. So in order to achieve this lower cost and higher quality, testing techniques for SoCs have to be improved.

The number of pins in most of the SoCs are still under 600 [2], however, with the increase of device gate count and its complexity the number of pins will also increase. With the improvement in the fabrication technology, the future SoCs will contain over million gates, integrated embedded memory, and mixed signal digital and analog cores. For optimal test coverage, testing will move towards 1024-pin automatic test system with integrated memory and mixed signal test capabilities [3].

With the speed of data transfer, the need for frequency accuracy becomes evident. The SoC test system must provide very high, sustained data rates to cope up with today's requirements and standard of the buses.

In SoC memories, mixed signal circuits such as PLLs, digital to analog converters (DACs), analog-to-digital converters (ADCs), temperature sensors, RGB modules for color video display are integrated for communications application. Mixed signal test capability is also a critical element for a SoC test system. Measurement and testing can be done with special test setup but in order to save time, both for set up and measurement, integral instrumentation within the SoC system architecture is preferred to create a better system and cost effectiveness.

Despite of the increasing complexity of the SoC, built-in self-test techniques (BIST) is considered. In BIST techniques, individual modules on the chip have their own test circuits enabling access to circuits embedded in the device. BIST techniques rely on the system to initiate the test sequence and measure the pass- fail status of the device. The testing require test patterns, which are generated by the test pattern generator (TPG) and applied to the module under test (MUT), the output is then compared with the known correct signature or responses. When the digital system is very large, it requires increased storage space for fault free response and as a result the test procedure become quite costly, so different approach is needed to reduce the amount of storage [3],[5]. This can be achieved if we can reduce the volume of the fault-free response.

Test pattern generation, test application, and response comparison can be done with the utilization of built-in hardware [3], [6], [7], [8] in BIST. This helps to reduce the testing time by allowing different areas of the chip to be tested in parallel and to eliminate the need for external equipments. This also improves the cost efficiency of testing.

Since SRAMs, ROMs, FIFOs, and registers do not need complex hardware for test generation and compaction, BIST is successfully used for fault diagnosis. With the use of BIST techniques high fault coverage with zero-aliasing can be achieved. For regular circuits, random-logic circuits, BIST technique is not adequate, more over, test result generated by random-logic circuits seldom maintain regularity, Because of these problems random-logic circuits are tested using combination of BIST, scan design techniques, and also external test equipment [3].

Delay faults are becoming critical in digital circuits. The testing technique consists of sensitizing a path in the digital circuit under test and then incorporating it in a ring oscillator to test for delay and stuck-at faults in the path. This procedure should be exercised for all or at least critical paths in the circuit. To establish oscillations, we should make sure that there is an odd number of inverters in the loop. This technique can be used along with scan techniques or be implemented as a built-in self-test technique.

In a typical BIST environment, TPG is used to generate the test patterns, which is then injected to the MUT, the outputs from the MUT are fed into the space and time compactor, the response after compaction is then sent to the comparator. The output from the compactor is compared with the previously stored fault-free signature. A fault is detected if the circuit response is different from that of the fault-free circuit [5].

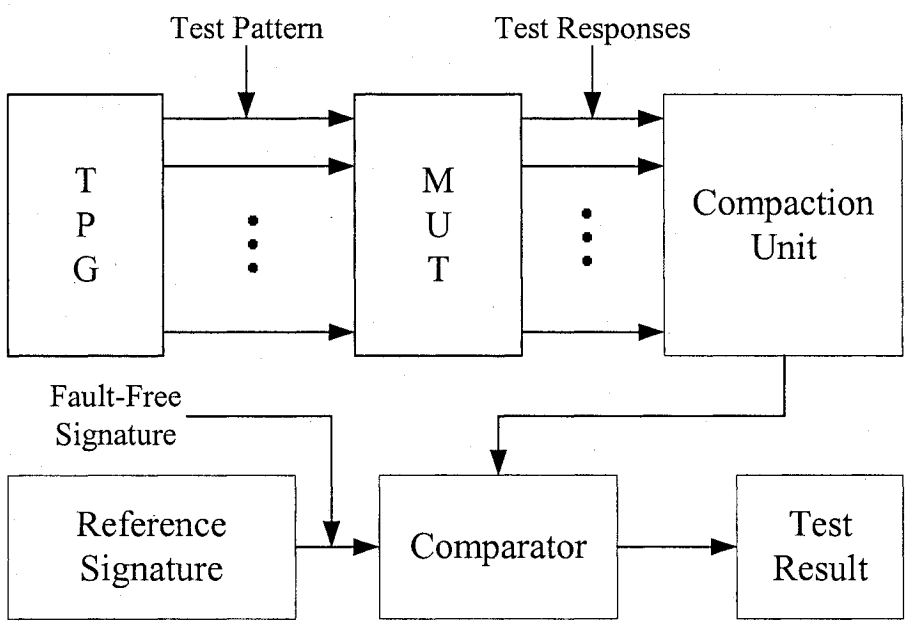


Figure 1.1 Block Diagram of BIST Environment.

To reduce the amount of fault free data and MUT responses, a compactor is used to create signature, which is short binary sequences, from the fault free data and responses of the MUT. BIST can be performed on-line or off-line, when the system is not in operation, BIST techniques may be used for off-line testing, or during the normal operation on-line testing is also possible.

Our aim in this research is the response compaction process of the MUT, objective is to implement suitable algorithm to find MCs and design the compactors for ISCAS 85 circuits to reduce the test response to a signature, which will help to avoid comparing bit-by-bit with fault free responses and it will save both time and space. With the successful compaction tree the test response of the MUT will generate a short signature, which would be then, compared with fault free signature, there by reducing storage need for the fault free responses [3], [5], and [8]. In general 's' input sequences coming out of the MUT are fed to the space compactor providing 't' out put streams of bits such that t<<s. Generally test responses are compressed into one sequence (t =1). With space compaction, it is possible to achieve high quality self-testing of complex chip and eliminates the need for large numbers of test pins. The compaction circuit must be as simple as possible to minimize the overhead cost and design with special care to avoid any signal delay. The signature must be as short as possible to minimize the amount of memory for the fault free response. In addition, the test result should be aliasing free.

Aliasing [3], [5], [9] or error masking occurs when the signature of the faulty output responses is same as the signature of the fault free responses. When aliasing occurs, it reduces the fault coverage and affects the test quality of the MUT. Fault coverage means that the number of faults detected after compaction, over the total number of faults injected. Practically high coverage, over 99% is required to make the space compaction technique feasible.

In this thesis, effort was given in designing efficient space compactor for SoC. The techniques used for this purpose based on certain inherent properties of the test response of MUT and their failure probabilities. Major emphasis was to design the compactor aliasing free, low overhead, & without interrupting the performance of the MUT.

1.1 Scope and the Thesis outline

The objective of the research is to design space compactors by implementing decomposition algorithm and using both deterministic and pseudorandom test patterns, based on the previous work done on compaction. The task and challenges are as follows:

- Use of different algorithm to find the Maximal Compatibles (MCs).
- Achieve maximal output line compaction without modifying the circuit.
- Reduce the area overhead.
- Minimize the propagation delay of the compactor.
- Analyze the results of simulation on different ISCAS 85, combinational benchmark circuits.

1.2 Organization of Thesis

Chapter 1 provides brief information of SoC, IP core and embedded systems and its testing method, BIST technique, and introduction about the dissertation.

Chapter 2 discusses about the SoC background and space compaction and time compaction techniques.

Chapter 3 introduces test generation techniques, definition and types of faults, fault models and fault injection techniques.

Chapter 4 discusses mergeability criteria for the design of space compactor.

Chapter 5 provides mathematical basis and algorithm to construct aliasing free compactor with maximal compaction.

Chapter 6 presents our experimental results on simulation on ISCAS 85 and ISCAS 89 scan benchmark circuits with brief analysis.

Chapter 7 discusses about the experimental and future work in this area.

CHAPTER 2
OVERVIEW

In this chapter, we discuss about the revolutionary development that took place in the field of semiconductor and semiconductor Industry. We also discussed SoCs and their developments and give an overview of some literatures on space and time compaction.

2.1 SoC

With the tremendous development of the wafer technology, semiconductor technology has advanced substantially and it is following the Moore's law. In 1965, Gordon Moore first postulates his theory on the rate of increasing complexity of the chip and the doubling of computing power in every 18 months. With present development of semiconductor and IC, computing power increased tremendously, like Kilobits memory become megabits and to gigabits. With the birth of SoC and its continuous development the gigahertz, microprocessor also becomes possible.

Chips are becoming complex day by day and with the increasing capacity for integration have made it possible to put together digital, analog, memory and mixed signal functions resulting an efficient, faster, but complicated system on a chip. Due to SoC smart phone, PDA, mobile robot, and HDTV are now possible. The complex digital processor, memory, high performance analog and mixed signal functions in the form of IP cores are put on a single chip to form SoCs. Application of SoC is enormous. The block diagram of a SoC for PDA is shown in Figure 2.1.

The rapid development in the field of SoCs and the shift from discrete IC to SoC have a major impact on design, the evaluation of SoC depicted in Figure 2.2 shows the IC design progress in last 20 years, it progressed from discrete ICs, to application-specific ICs, to digital SoC ICs, to today's complex and efficient SoCs.

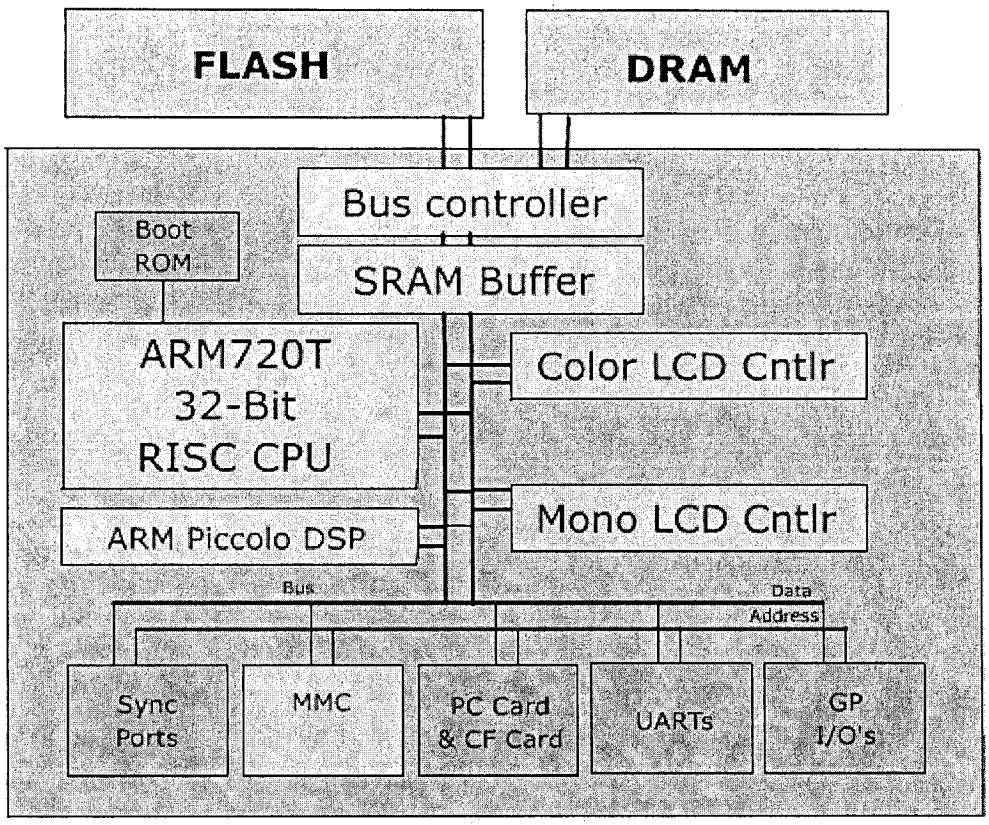


Figure 2.1 Block Diagram of SoC for PDA

Up to 1980s, there was only one design implementation methodology, where there were no distinction between analog and digital. With the presence of digital synthesis tools, the digital and analog design flow become separate, thus a major divergence occurred in the field of electronic. The divergence continued from the introduction of gate-level synthesis in 1980's, but the analog design remained mostly a handcrafted art. Due to the top-down approach at the architectural level tends to keep process technology details hidden for a long time. On the other hand, the analog design required bottom-up approach based on the physical characteristics of the process technology.

With the advancement in the SoC and mixed signal design, a turnaround of design methodology is taking place. Mixed signal architectural design can now start in a converged hardware description language, with the analog mixed-signal (AMS) extensions to both Verilog and VHDL standards for digital behavioral gate-level design and both IP blocks can be modeled behaviorally before the detail design.

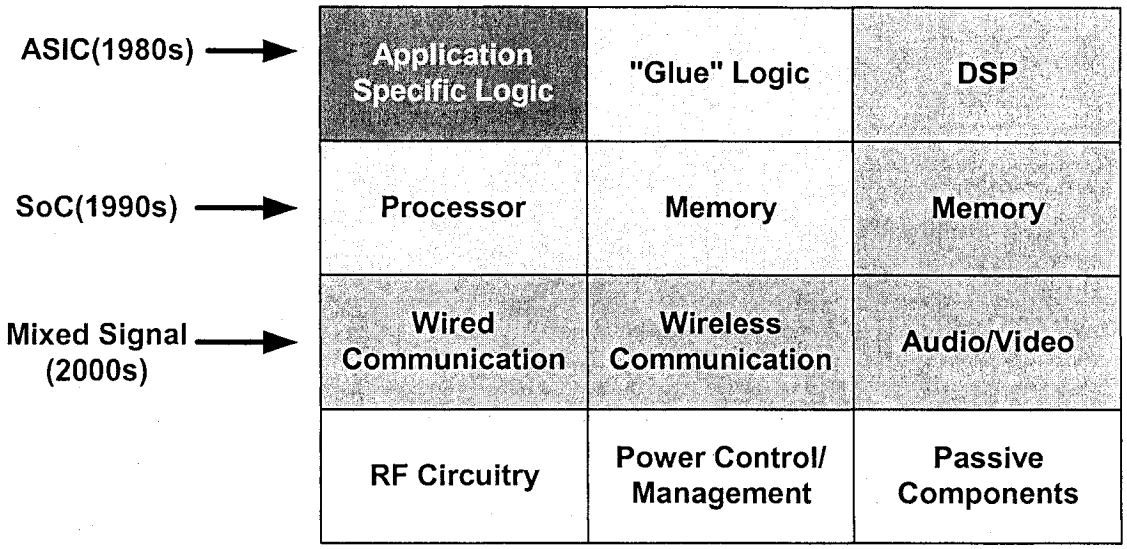


Figure 2.2 Evolution of SoC

With rapid growth in the field of semiconductor technology, the high density SoCs and ICs are now common target for designing a product. The demand of efficient, high density, mixed signal SoCs are increasing for the application in HDTV, 3rd generation Cell phone, new display technology, and for the new concept of memory modules.

2.2 SoC Related Problems

The rapid advances of the microelectronics technology in recent year have brought new possibilities to ICs design and Manufacturing. Many systems these days are designed by embedding pre-designed and pre-verified complex functional blocks, usually referred as cores, into a single die as shown in Figure 2.3. This type of design style allows designers to re-use previous designs and will lead therefore to shorter time to market and at reduced cost. Such a SoC approach is very attractive from the designer perspective. Testing of SoC, on the other hand, shares all the problems related to testing modern deep sub micron chips, and introduce some additional challenges due to the protection of intellectual property as well as the increased complexity and higher density [10].

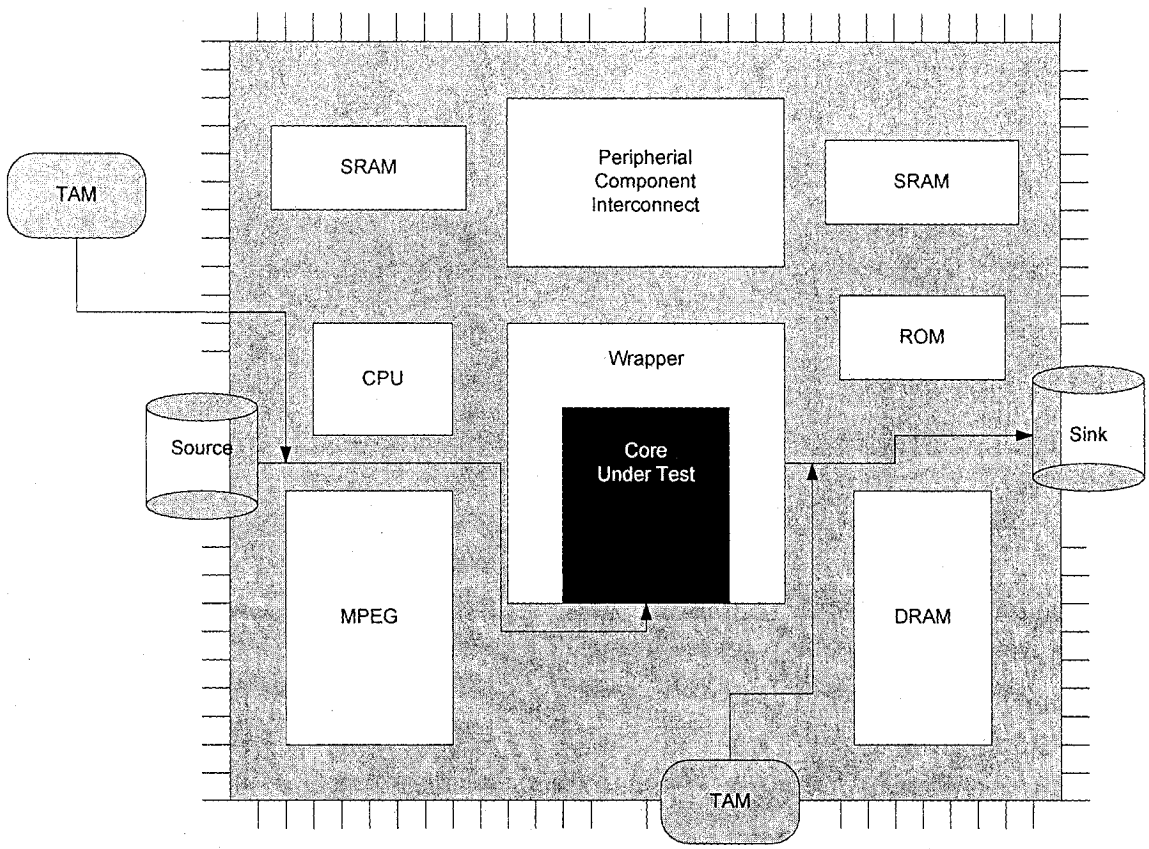


Figure 2.3 Testing a System-on-chip.

As the complexity and the density increases, a large amount of design time (approximately 60-70% of the total time) is allotted to verifying that the chip meets the specification when fabricated; otherwise, millions of dollars may be wasted in re-engineering. On the other hand, software engineers have to write greater quantities of codes to implement complex system, control strategies, signal-processing algorithms, and communication stacks. Thus, new approaches for testing chips are needed urgently but with less impact on the cost structure to make the product viable in the market. These are the most common challenges has to be faced, from the industry point of view while testing SoC.

As we discussed before, although large functional blocks such as the CPU, DSP, communication interfaces, and software stacks may be re-useable, pre-verified components can also be used with many different SoCs to reduce the amount of low-level

design work. Also in platform-based designs, most of the carefully selected basic systems are carried forward over many product designs and their derivatives in a particular field of use. Despite of many inherent challenges, these approaches are strongly advocated in major industry players. The focus on SoC design is therefore moving from low-level implementation to the specific process and the architectural level, so the critical decisions need to be made at a relatively early design process. The objective is to use tools and languages to achieve a robust initial specification, which can not only evolve as the design processed through implementation but also generate system modules against which implementation can be verified and tested at the subsequent stages in the design process. Architectural decisions have a significant impact on the system performance. In most cases, compromise have to make between the all hardware and all software extremes and design need tools with which to explore and evaluate the options, in terms of silicon efficiency and power consumption and area.

With increasing complexities and multimillion gates, the IC designs and manufacturing industries are in need to identify alternative method to minimize the increasing large costs of testing.

2.3 SoC Testing

To test the individual cores of the system the test pattern source and sink have to be available together with an appropriate test access mechanism (TAM) [11] as depicted in Figure 2.3. The processor core, on-chip memory, and other high bandwidth peripheral cores are connected together to a pipelined advanced system bus. To avoid and minimize the performance problems during testing, SoC components are connected to a separate advanced peripheral bus, which can also be used for lower performance cores. Alternatively, sub-circuits in either boundary-scan format (multiplexing approach) or another isolating technique can be used on the boundary of each core.

A traditional approach implements both source and sink off-chip and requires external Automatic Test Equipment (ATE). As the requirement for the ATE speed and memory size are continuously increasing with the increased density of the SoCs, the ATE solutions are no longer accurate and cost effective. Therefore, in order to apply at speed

test, and to keep the cost under control, on-chip test solutions are becoming more and more popular [12]. Such a solution is called BIST. Logic BIST coupled with boundary scan can take care of most of the digital core testing challenges [1]. BIST can also be applied to certain analog functions, such as phase-locked loops and even data converter [3]. The advantage of using logic BIST is that, its ability to test a circuit at full operating speed, so at full clock speed, path delays and other timing problems become prominent and which is very crucial for sub micron chips. Thus, logic BIST has become essential if timing faults are to be included in the coverage. The full scan automated test pattern generation (ATPG), which can be conducted hierarchically isolating module by module as shown in Figure 1.1, the test data analyzer, in a typical BIST environment, includes a space compaction unit followed by a time compaction unit.

A new concept of testability, called consecutive testability, in SoC was introduced in [13]. It proposed design-for-testability (DFT) method based on an integer programming problem. The test patterns and the test responses of a core are propagated to the core input through SoC inputs and the SoC outputs from the core output consecutively at the speed of the system clock, the responses are achieved consecutive transparency properties of the surrounding cores and interconnects between the cores. This method can test stuck-at faults and delay faults.

In [14], a compression method combined with scan power technique for SoC testing is also investigated where 97% of the peak power and 99% average power saving can be achieved with a compression ratio up to 95%.

A hybrid BIST solution [15] for testing SoC is also proposed for optimal balance between pseudorandom and stored test pattern. The proposed method can generate test with minimum time and memory.

2.4 Space Compaction

For testing digital cores so far there are different types of techniques, which include the parity tree space compaction, hybrid space compression, dynamic space compression, quadratic function compression, and cumulative balance testing [3][16].

XOR gates are used for parity tree compaction, XOR gates have very good signal-to-error propagation properties, which is quite useful for space compression. The form of the function realized by parity tree compactor is $Y = Y_1 \oplus Y_2 \ldots\ldots \oplus \ldots Y_k$, where 'k 'is the number of output from the module under test (MUT). However there is a limitation that the parity tree compactors propagates all error that appear on an odd number of its inputs. Errors that appear on an even numbered input of a parity tree compactor are masked. Otherwise most single stuck-at line faults can be detected using pseudorandom and deterministic test pattern.

Based on the detectable error probability estimation, hybrid space compactors (HSC) are built where AND, OR, and XOR gates are used to compress the multiple out puts of the MUT into a single line out put [17].

Dynamic space compression (DSC) is the modified version of the HSC. The basic idea of DSC is to construct the compression circuits by dynamically estimating error probabilities of the outputs. The values of the probabilities of single and double line errors are determined based on the number of single lines and shared lines connected to an output [3]. Experiment conducted in [18] by computer simulation demonstrates that DSC is an efficient method for space compression when testing VLSI circuits. The information loss compared with the syndrome counting is between 0% to 12.7%. Although DSC was later improved, it still does not provide adequate measure of fault coverage because they rely on estimation of error detection probabilities.

Programmable space compaction (PSC) is proposed in [19]. PSC can generate high fault coverage and it is low cost. In PSC, circuit specific space compactors are designed to increase the probability of error propagation. A compaction circuit that minimizes aliasing and has the lowest cost is only possible by exhaustively enumerating all $(2^k)^k$ k-input Boolean functions, where k represents the number of primary outputs of the circuit under test.

Quadratic functions compactions (QFC) is based on quadratic non-repetitive function of 2T variable over GF (2^k), Galois field of 2^k elements [20]. In QFC the observed output

responses $Z_1, \ldots, Z_k$ of the circuit under test (CUT) are processed and compressed in serial format, and the function is of the type $Z_{i0} Z_{i1} \oplus Z_{i2} Z_{i3} \oplus \ldots \oplus Z_{i(2k-2)} Z_{i(2k-1)}$, where Z_{it} and $Z_{I(t+1)}$ are blocks of length k, for t = 0,2,..., 2k-2.

In [21], a new class of space compactor was proposed, with the concept of cumulative balance testing (CBT). The method is based on multiplexed parity trees (MPTs), resulting zero-aliasing test signature. The MPTs performs space compaction of the test responses by combining the error propagation properties of the multiplexers and parity trees through multiple time steps. With this, very high fault coverage is achievable including the faults in the compactor.

For both time and space compaction, so that a single output is derived for all outputs was introduced in [22]. It is a simple approach and is suitable for built-in implementation, the best current application of such space compacted signature is PLAs and ROMs, where full testability of all single faults are achieved.

To achieve zero-aliasing compaction tree, a multiple parity functions (signatures) are merged into a narrow signature steam was presented in [23]. With this method, zero-aliasing can be achieved without changing the test sets and without any increase in the test application time.

Space and time-oriented compaction technique was proposed in [24], [3] .The proposed scheme can compress the data of a k-output circuit into single bit signature stream with zero-aliasing and without degrading the performance for single stuck-line faults. In the time oriented, a s-bits long data is compressed into r-bit with zero-aliasing where r<<s as shown in Figure 2.4.

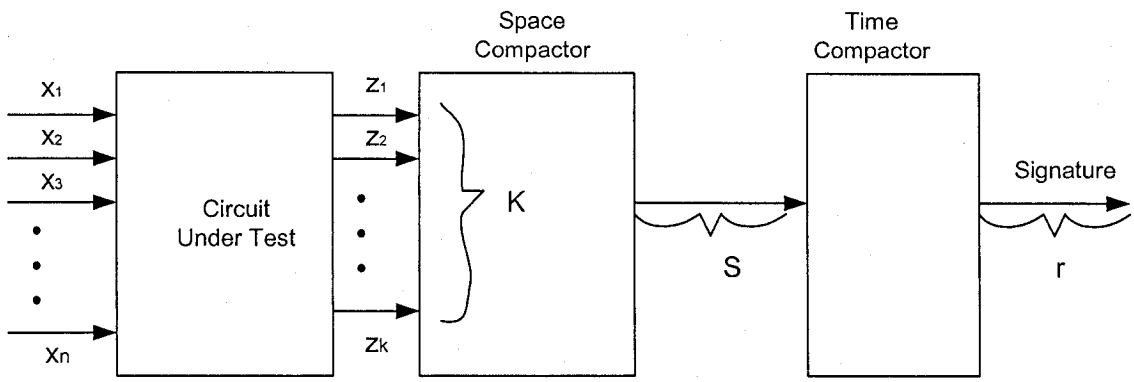


Figure 2.4 Test response Compactor (Space and Time) in BIST Environment

2.5 Time Compaction for Circuit Test

The time compaction techniques are based on parity bit checking, one's count, transition count, Walsh coefficient, linear feedback shift register, and parallel compaction analysis. These techniques can be used to evaluate the compressed response data to find out the status of the CUT [3].

With one's count technique the number of one's in the binary circuit response stream during test execution is checked [25]. The time compaction unit is independent of the CUT and it is a simple counter and depends only on the test response nature, the signature does not depend on the order in which the test patterns are applied to CUT. Signature length is logarithmic function of the output response data. If the CUT response length is r bits, then the signature length after time compaction will be of length $[\log_2 r]$. The probability distribution function of aliasing is approximately Gaussian with a mean value at r/2; r is the length of the output stream [3]. The masking probability [26] is low when the one's count of the signature is near either the minimum or maximum of its length range, so it takes the maximum value whenever the signature length approaches a maximum at the mid range of the count.

In syndrome testing ([3], [27]), it requires application of 2^n patterns to be applied at the n input combinational circuit. The fault can be detected by comparing number of ones at the output stream to the number of ones in the fault-free stored signature.

In transition counting [3], [28], the number of 0-to-1 or 1-to-0 transitions in the output bit stream is considered as signature. The compression length is less than or equal to $[\log_2 r]$, where r is the length of response stream. When the signature value is close to r/2, the masking probability takes high value, and it takes low value when the signature value is close to 0 or r. The transition counting does not guarantee the detection of all single-bit errors.

To avoid any loss of information double or multiple transition count (DTC or MTC) has recently been proposed [29], [7]. DTC can be used for testing single output circuits and multiple output circuits can be tested with MTC. DTC/MTC techniques do not require repeated input test patterns and do not use any counter. The test circuit consists of inverter, switch, OR logic gate and D flip-flop.

In parity compaction [30], the response data is compressed in to a signature only one bit. The value of the bit stream is 1 if the parity of test response sequence is odd, and 0 if the parity is even. All errors involving an odd number can be detected with parity compaction, but faults that give rise to an even number of error bits are not detected, so parity checking is a relatively ineffective compaction method. The compaction circuit consists of flip-flop and a XOR gate.

Walsh spectral compaction method requires that all possible input patterns be applied to the combinational network. In Walsh spectral analysis [31], the switching functions are represented by their spectral coefficients that are compared to known correct coefficient value. Higher percentage of fault coverage can be achieved with spectral coefficient but Walsh spectral compaction requires higher area overhead.

An alternative method to transition count testing is cyclic redundancy check (CRC). CRC can be easily implemented to detect errors in data communications, and it requires little overhead and has an extreme error detection capabilities. This technique has been used for compression of test response data [3], [32]. The signature is residue left in feedback shift register after the response from CUT has been compressed. It has two applications, one is as a source of pseudorandom binary test sequence, and the other is

response compression. The response compression is commonly known as signature analysis.

The CRC is realized by using linear feedback shift registers (LFSRs), consisting of flip-flops and XOR gates. LFSRs are used for generating pseudorandom input test patterns and most popularly used for response compaction. The nature of the generated sequence pattern is determined by the LFSR's characteristic polynomial as defined by its interconnection structure. LFSRs used for response compression is shown in Figure 2.5 and is called signature analysis. The test response from the MUT/CUT which is a bit stream is fed into the LFSR, the output stream is not observed, and only the state of the LFSR, called signature which is evaluated after the test. Let coefficient of feedback is h, the output is o, the input stream is E, and the signature is S. The feedback polynomial is,

$$h(X) = h_r + h_{r-1} X^{r-1} + \ldots\ldots\ldots\ldots + h_0$$

The input polynomial,

$$e(X) = e_n X^n + e_{n-1} X^{n-1} + \ldots\ldots + e_0$$

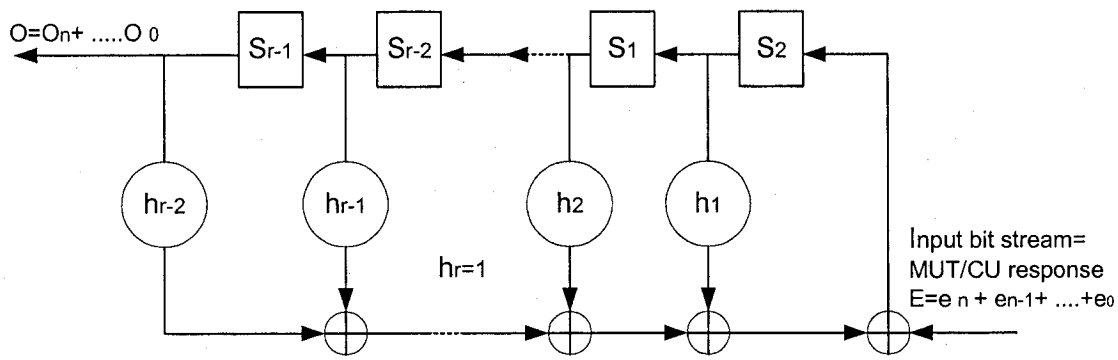


Figure 2.5 LFSR Response Compactor (Signature analysis)

The output polynomial,

$$o(X) = o_n X^n + o_{n-1} X^{n-1} + \ldots\ldots\ldots + o_0$$

The remainder polynomial which corresponds the final state of LFSR,

$$s(X) = s_{r-1} X^r + \ldots\ldots\ldots + s_0.$$

From the above, it is clear that e(X) / h(X) = o(X) + s(X) / h(X), hence the signature analysis is just a polynomial division and the signature is the remainder.

In [33] multiple-input signature registers (MISRs) were used to test the multiple output circuits. The use of MISRs in testing eliminates the need for space compactor. However, MISRs increase aliasing and require extra hardware, which may be not practical.

Recently a multiple-output parity bit signature generation method for use with non-exhaustive or compact test sets in testing digital integrated circuits was proposed in [34]. With the proposed compaction technique, high fault coverage for single stuck-line faults can be achieved with low CPU simulation time. The area overhead for the compactor is also low, but the design does not ensure zero-aliasing compression. Information loss may not be completely avoided when the size of all output responses are reduced. So, depending on the amount of information loss, the corresponding time compactor design can be affected as well. With reduced input test set but not the minimal test sets 100% fault coverage is achievable. Experiment results indicate that the new designed multiple-output parity bit signature generators are comparable in respect with conventional space-time compression, usually considered ideal for multiple-output combinational circuits [3].

CHAPTER 3

FAULT SIMULATION AND TEST GENERATION

In this chapter, we discuss about fault simulation, fault types, and definitions. We also discuss about fault simulation process, fault modeling, fault collapsing, etc.

3.1 Fault Simulation

Fault simulation is to check the response of the system/ circuit/ module from a model. Simulation may be for checking the logic or switching function, timing, circuit, and circuit faults [35]. For electronic system, both hardware and software models are used. In general, simulating a circuit in presence of fault is known as fault simulation. For fault simulation, breadboards are used in the traditional way, but with the advancement of the computing, software models are becoming more and more popular due to its accuracy, convenience, speed, and cost [35]. Simulation is an integral part of the modern digital circuit design due to its density and complexity. Through simulation, we can understand the real time performance of a system or circuit. It also shows the physical failure behavior, timing performance, and the performance at higher clock speed.

Simulation is a computer program and its inputs are circuit description and primary input signal description or the stimuli [35]. The input signal waveform is called the stimuli, which may be described as test vectors for digital circuits. A vector contains all the input signals. If we have a circuit and a set of faults in the circuit together with a set of tests (Input vectors or stimuli) then the fault simulator decides which faults will be detected by which tests. Base on the result, new test vectors are generated to detect the undetected faults and simulation is performed to obtain better fault coverage. The cycle of test generation and simulation continues until satisfactory test result is obtained or continues up until certain number of test vectors is applied. As the vectors contain the values of all the signals, whenever one or more input changes, a new vector must be specified. Most of the time the input signal changes are synchronized with some

periodically changing clock signal, so the input stimuli are a sequence of vectors applied at periodic intervals.

Since signal changes propagate from the inputs to the outputs, the circuit is labeled from inputs to outputs, and accordingly the components are evaluated. We encounter problem with the circuits with feedbacks, as labeling is not possible with the case of feedback. Problems also lie with the circuits having components with different delays.

Efficient and accurate simulation of delays, feedbacks, and the dynamically changing activity is possible by the event driven method. In this method the time, over which the circuit activity is analyzed, is divided into some suitably selected unit, and all delays are specified in terms of this time unit. The simulator contains circular stack (time wheel) and each element of this wheel is known as time slot, which is one unit of time.

Two common fault simulators are the gate-level and the behavioral fault simulation. In our thesis, we are using the gate-level fault simulation. We used logical fault model and under this model, every single line can become permanently fixed (stuck) at logic 1 or 0. Since we used software simulator, the circuits/modules are on the net list format which is circuit description. The simulation technique is shown in Figure 3.1.

Circuit File

The circuit file is the CUT, it is in the net list form. Here the design files are the benchmark ISCAS 85 combinational digital circuit or ISCAS 89 sequential digital circuits. To run the software simulation the net list format of ISCAS 85 Combinational and ISCAS 89 full-scan sequential circuit files are used.

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].

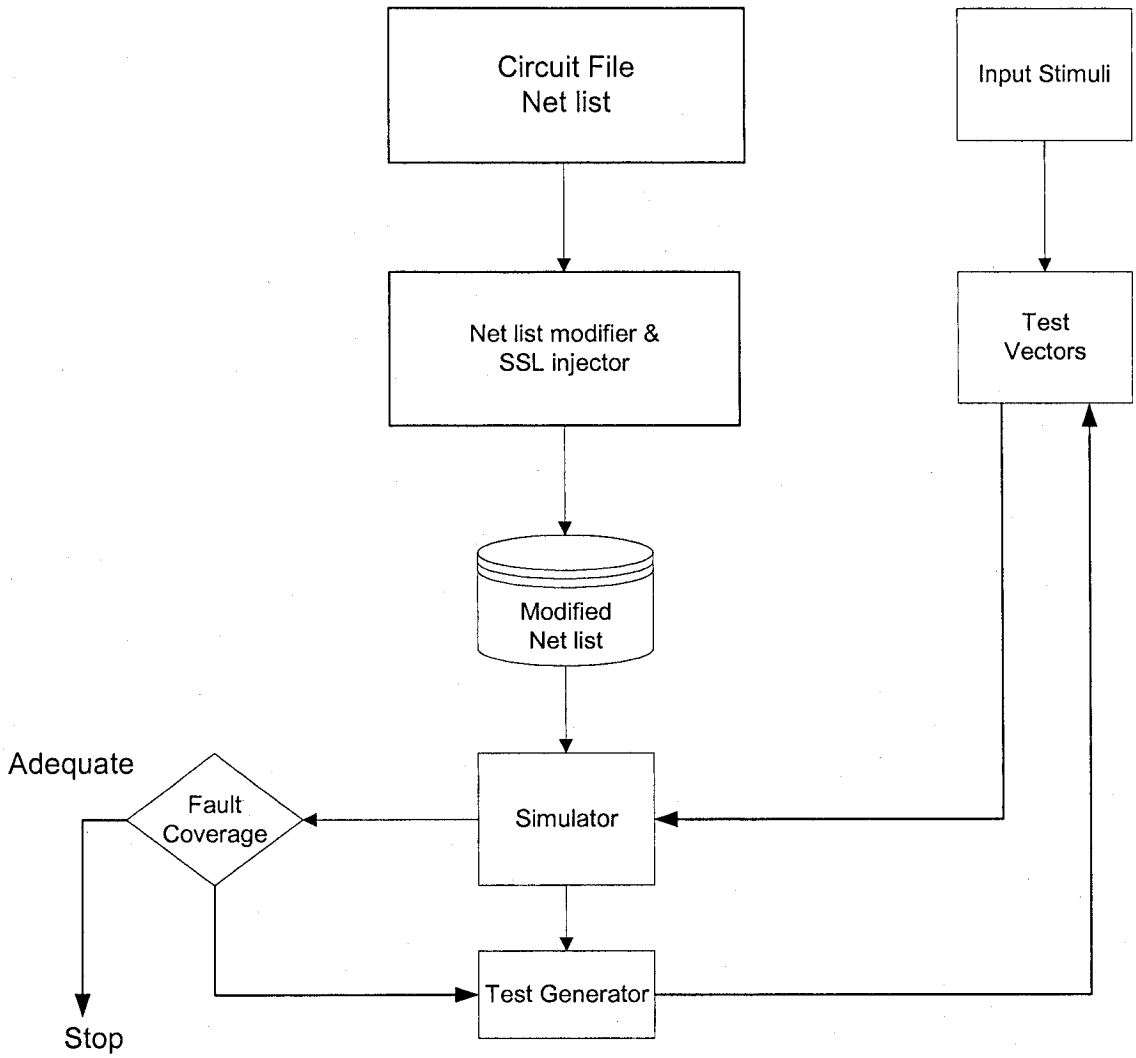


Figure 3.1 Simulation Technique

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.

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 on 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 3.2. 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.

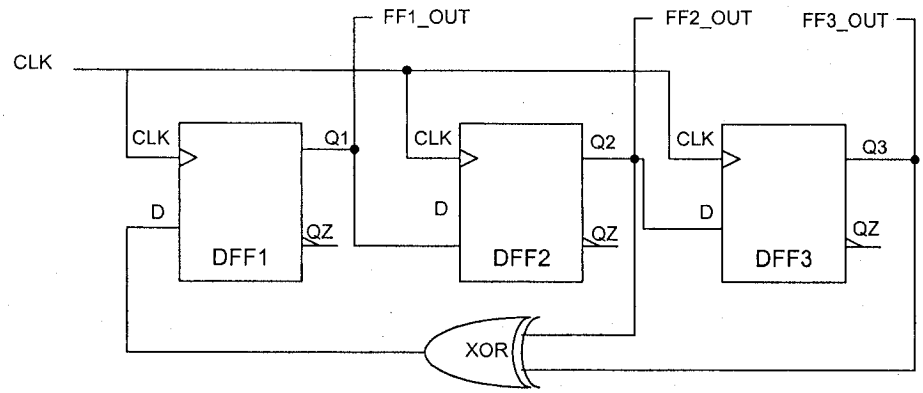


Figure 3.2 Linear Feedback Shift Register (LFSR)

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 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 is 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.

3.3 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 fault most commonly modeled are stuck-faults, which is very simple fault to analyze and proved to be very effective in representing the faulty behavior of actual devices. The stuck–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:

1. Only one line is faulty.
2. The faulty line is set to 0 or 1.
3. 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.

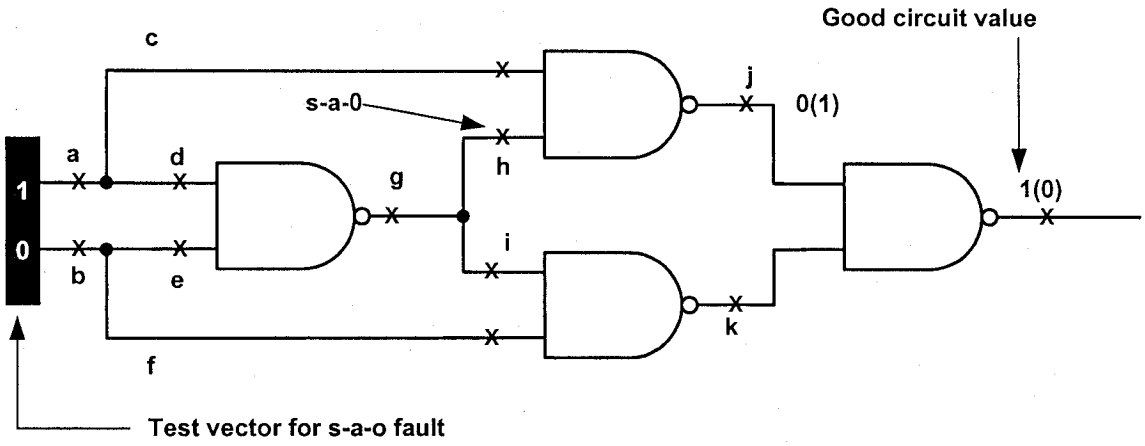


Figure 3.3: Single stuck-at Faults

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

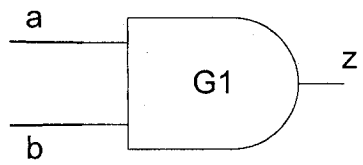
Given two faults f_1 and f_2, fault f_1 considered to dominate fault f_2 when every test for f_2 is also test for f_1. When f_1 dominates f_2, then f_1 is removed from the fault list.

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 3.4 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 3.4 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

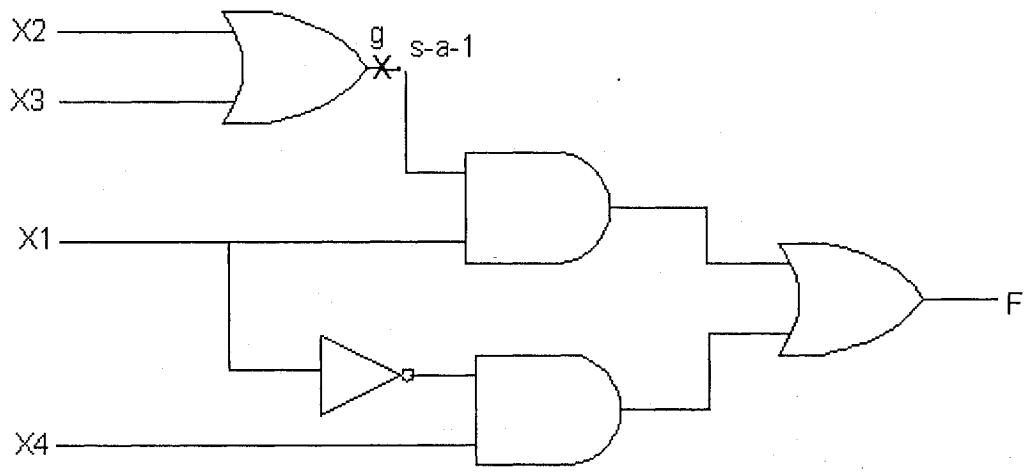


Figure 3.5 Sensitized Path

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.

3.4 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.

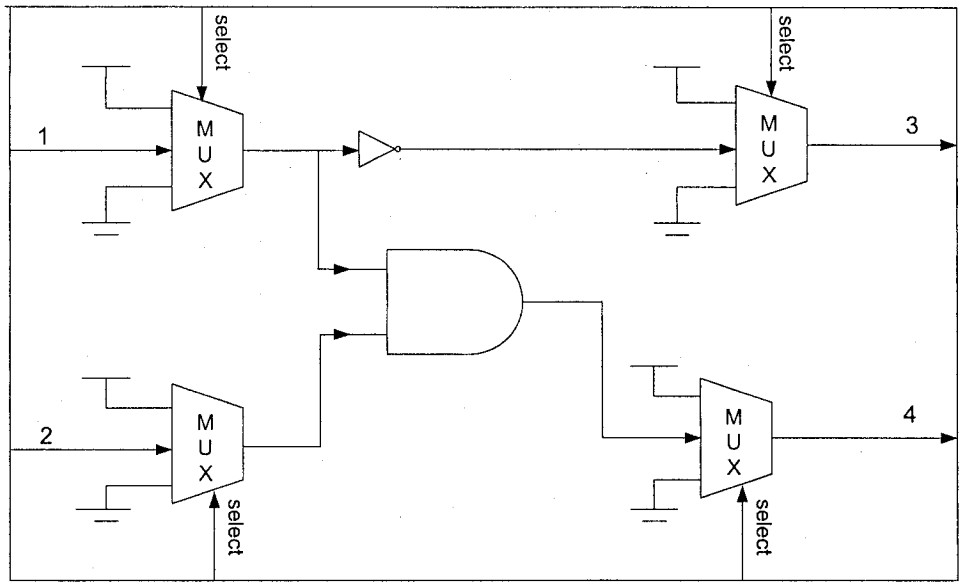


Figure 3.6 Test Set Up For Hardware Fault Injection.

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 injection 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.

CHAPTER 4

COMPACTION UNDER GENERALIZED MERGEABILITY

This chapter considers space compaction under generalized mergeability of response stream of CUT, with the objective to merge pair of lines of the CUT under stochastic dependence [39] and independence [40] of line errors to decide on the gate (AND/NAND, OR/NOR, XOR/XNOR) for zero-aliasing and achieve maximum compaction ratio for deterministic testing.

4.1 Preliminaries

Let $\Psi_1, \Psi_2, ..., \Psi_N$ represent N output sequences of length L_s, where the length is the number of bit positions in Ψ_i, $i = 1, 2, ..., N$.

The first-order 1-weight is the number of 1's in a sequence of length L_s. Similarly, the first-order 0-weight, is the number of 0's in the sequence [41].

Property 1: For any sequence Ψ_i, $L_s = \omega_{11} + \omega_{10}$ where ω_{11} and ω_{10} are first-order 1-weight and 0-weight, respectively.

Example 1: Consider an output sequence Ψ_1 = (11000101 0010) of length $L_s = 12$. The first-order 1-weight is $\omega_{11} = 5$ and 0-weight $\omega_{10} = 7$. We can extend the concept of 1-weight and 0-weight to deal with N sequences $\Psi_1, \Psi_2, ..., \Psi_N$.

Nth-Order 1-Weight: For sequences $\Psi_1, \Psi_2, ..., \Psi_N$, each of length L_s, let the sequences have 1's in m identical bit positions (the sequences each might have 1's in other positions besides these m positions). Then, the Nth-order 1-weight of the sequences in the group is given by the number of positions in which all the N sequences have 1's, which is m in this case.

Nth-Order 0-Weight: For sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$, each of length Ls, the *N*th-order 0-weight is defined in the same way as the *N*th-order 1-weight and corresponds to the number of bit positions in which all the sequences have 0's.

Example 2: Consider three sequences of length Ls = 12 as follows:

$$\Psi_1 = (1111\ 0010\ 1010)$$
$$\Psi_2 = (1111\ 0111\ 0110)$$
$$\Psi_3 = (1111\ 0101\ 1010).$$

The third-order 1-weight of and is since of Ψ_1, Ψ_2, and Ψ_3 is $\omega_{31} = 5$ since the sequences agree in bit positions 2, 9, 10, 11, 12 having 1's (counting from right). Note that each of the sequences Ψ_1, Ψ_2, and Ψ_3 has 1's in other bit positions besides positions 2, 9, 10, 11, and 12. The third-order 0-weight is $\omega_{30} = 2$ since all three sequences have 0's in bit positions 1 and 8 (counting from right).

We will denote Nth-order 1-weight and 0-weight by ω_{N1} and ω_{N0} respectively, where the first subscript corresponds to the sequence number or order, and the second subscript gives the binary digit corresponding to the weight.

Bundling: If *N* sequences are grouped together having certain *N*th-order 1-weight (or 0-weight), then the process will be termed bundling, and the grouped sequences will be termed bundled sequences. Whenever there will be bundling of a number of sequences, we will say, we are working under bundling constraint.

Nth-Order Derived Sequences: For *N* sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$, each of length L_s, the *N*th-order derived sequences, denoted by $\Psi'_1, \Psi'_2, \ldots, \Psi'_N$ respectively, are obtained by deleting the bit positions in which at least two of the sequences differ and replacing the bit positions with a dash (-).

Example 3: For the three sequences Ψ_1, Ψ_2, and Ψ_3 in the previous example, we have the derived sequences

$$\Psi'_1 = (11110 - - - - - 10)$$
$$\Psi'_2 = (11110 - - - - - 10)$$
$$\Psi'_3 = (11110 - - - - - 10).$$

Property 2: If the Nth-order 1-weight is ω'_{N1} *and* the Nth order 0-weight is ω'_{N0} of the N derived sequences in a bundle, then

$$L_s = \omega'_{N1} + \omega'_{N0} + R$$

where R is the number of bit positions in which at least two of the sequences in the bundle have different entries and the corresponding sets of derived sequences have dash (-) entries.

Property 3: Two Nth-order derived sequences may have the same Nth-order 1-weight, 0-weight, and residue R but they may represent two entirely different sets of bundled sequences.

Error Multiplicity: The error multiplicity, denoted by v, v is the number of output lines, which can be faulty simultaneously.

4.2 Merger under Stochastic Independence Line Errors

All of the following theorems are from [39], and [40].

On the assumption of stochastic independence of errors, let us consider N output sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$, each of length L_s at the output of a CUT. Let the corresponding Nth-order derived sequences be $\Psi'_1, \Psi'_2, \ldots, \Psi'_N$ having Nth order 1-weight and 0-weight, ω'_{N1} and ω'_{N0} respectively. Also, let R be the residue.

Theorem 1: For a bundled set N of output sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$, each of length L_s at the output of a CUT, the maximum possible errors with all possible multiplicities of errors v_i, is $E_{max} = L_s(2^N - 1)$.

Proof: For N sequences of length Ls, every bit position can have only one error (error multiplicity v_1), can have two simultaneous errors (error multiplicity v_2), and so on up to N possible simultaneous errors (error multiplicity v_N). The sum of all possible errors in

one bit position is then $2^N - 1$, where 2^N is the total number of binary combinations in the bit positions and 1 corresponds to the one error free pattern. Thus, for sequences of length L_s, the maximum possible errors is $E_{max} = L_s (2^N - 1)$.

Theorem 2: The total faults detected when the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ are merged by using an N–input AND (NAND) gate is

$$T(\text{AND/NAND}) = (2^N - 1) \text{ X } (\omega'_{N1})_{v1, v2, \ldots, vN} + (\omega'_{N0})_{vN} + \sum_{vj} Ri_{vj}$$

where $(\omega'_{N1})_{v1, v2, \ldots, vN}$ represents error multiplicities of 1, 2, …., N $(v1, v2, \ldots, vN)$ and Ri_{vj} represents the ith residue position of v_j 0's.

Proof: Since the derived sequences have 1-weight ω'_{N1}, having all 1's in these positions, error multiplicities 1, 2, …., N $(v1, v2, \ldots, vN)$ are detected by an N–input AND gate in each position. The sum of all these errors is $\omega'_{N1} (2^N - 1)$. Similarly, since the 0-weight is ω'_{N0}, only one fault of multiplicity N (vN) is detected in each bit position, with total errors detected being ω'_{N0}. In each of the remaining residual positions, depending on the number of 0's in a bit position, only one error of appropriate multiplicity is detected. The sum of all these errors is $R = L_s - (\omega'_{N1} + \omega'_{N0})$. Hence, the AND (NAND) gate detects $T(\text{AND/NAND}) = \omega'_{N1} \text{ X } (2^N - 1) + \omega'_{N0} + R$.

Theorem 3: The total faults detected when the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ are merged by using an N-input OR (NOR) gate is

$$T(\text{OR/NOR}) = (2^N - 1) \text{ X } (\omega'_{N1})_{v1, v2, \ldots, vN} + (\omega'_{N0})_{vN} + \sum_{vj} Ri_{vj}$$

Proof: The OR (NOR)gate detects ω'_{N1} X $(2^N - 1)$ errors corresponding to ω'_{N0} bit positions having all 0's and ω'_{N1} errors corresponding to ω'_{N1} bit positions having all 1's and $R = L_s - (\omega'_{N1} - \omega'_{N0})$ in all R residual positions. Hence, the OR (NOR) gate detects $T(\text{OR/NOR}) = \omega'_{N0} \text{ X } (2^N - 1) + \omega'_{N1} + R$.

Theorem 4: The total faults detected when the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ are merged by using an N - input XOR (XNOR) gate is

$$T(\text{OR/NOR}) = (\omega'_{N1})_{v1, v3, \ldots, vN-1} \text{ X } 2^N - 1 + (\omega'_{N0})_{v3, v4, \ldots, vN} \text{ X } 2^N - 1 + \sum_{vj} Ri_{vj} = L_s (2^N - 1)$$

where N is even.

Proof: If the number of sequences N is even, then corresponding to each of ω'_{N1} positions having all 1's, all odd multiplicities are detected at the output of the CUT. Similarly, all errors of even multiplicities are detected at the output of the CUT corresponding to each of the ω'_{N0} positions having all 0's. For the remaining R residual positions, depending on the number of 0's, either errors of all odd or even multiplicities are detected. In every bit position, the total number of either odd or even multiplicity is $2^N - 1$. Hence an XOR (XNOR) gate detects a total of $L_s (2^N - 1)$ faults.

Theorem 5: In the extreme case when $R = L_s$ (and $\omega'_{N0} = \omega'_{N1} = 0$), the total faults detected by N-input AND (NAND), OR (NOR), and XOR (XNOR) gates are $T(\text{AND/NAND}) = T(\text{OR/NOR}) = L_s$ and $T(\text{XOR/XNOR}) = L_s (2^N - 1)$ respectively.

Theorem 6: For merger of the N sequences, an N –input AND (NAND) gate will be preferable to an N -input OR (NOR) gate for maximizing the error detection if $\omega'_{N1} > \omega'_{N0}$.

Proof: The total errors detected by the N –input AND(NAND) gate and -input gate are, Respectively

$$T(\text{AND/NAND}) = \omega'_{N1} (2^N - 1) + \omega'_{N0} + R$$
$$T(\text{OR/NOR}) = \omega'_{N0} (2^N - 1) + \omega'_{N1} + R$$

Where R is the residual bit positions where at least two of the sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ have differing entries. An AND (NAND) gate is preferable to an OR (NOR) gate if

$$\omega'_{N1}(2^N - 1) + \omega'_{N0} > \omega'_{N0}(2^N - 1) + \omega'_{N1}.$$

On simplification, we have $\omega'_{N1} > \omega'_{N0}$.

Corollary 1: For merger of the N sequences, an N – input OR gate will be preferable to an N-input AND gate for maximizing the error detection if $\omega'_{N0} > \omega'_{N1}$.

Theorem 7: For merger of the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, Nth order 1-weight ω'_{N1} and Nth order 0- weight be ω'_{N0}, an N- input AND (NAND) gate will be preferable to an N – input XOR (XNOR) gate for maximizing the error detection if $\omega'_{N1} > \omega'_{N0} + R = L_s / 2$.

Proof: The total errors detected by the N –input AND (NAND) gate and N -input XOR (XNOR) gates are, respectively

$$T(\text{AND/NAND}) = \omega'_{N1}(2^N - 1) + \omega'_{N0} + R$$
$$T(\text{XOR/XNOR}) = L_s(2^{N-1})$$

where R is the residual bit positions where at least two of the sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ have differing entries. An AND (NAND) gate is preferable to an XOR (XNOR) gate if

$$\omega'_{N1}(2^N - 1) + \omega'_{N0} + R > L_s(2^{N-1}).$$

Using $L_s = \omega'_{N1} + \omega'_{N0} + R$ and simply, we get

$$2\omega'_{N1}(2^{N-1} - 1) > (\omega'_{N1} + \omega'_{N0} + R)(2^{N-1} - 1) \text{ or}$$
$$2\omega'_{N1} > \omega'_{N1}(2^N - 1) + \omega'_{N0} + R = L_s \text{ or}$$
$$\omega'_{N1} > \omega'_{N0} + R \text{ or } \omega'_{N1} > L_s/2.$$

Corollary 2: For merger of the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, an N – input XOR gate will be preferable to an N - input AND gate if

$$\omega'_{N1} < \omega'_{N0} + R \quad \text{or}$$
$$\omega'_{N1} < L_s/2.$$

Theorem 8: For merger of the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, Nth order 1-weight ω'_{N1} and Nth order 0- weight be ω'_{N0}, an N - input OR (NOR) gate will be preferable to an -input XOR (XNOR) gate for maximizing the error detection if

$$\omega'_{N0} > \omega'_{N1} + R = L_s/2.$$

Proof: The total errors detected by the N –input OR (NOR) gate and N -input XOR (XNOR) gates are, respectively

$$T(\text{OR/NOR}) = \omega'_{N1}(2^N - 1) + \omega'_{N0} + R$$
$$T(\text{XOR/XNOR}) = L_s(2^{N-1})$$

where R is the residual bit positions where at least two of the sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ have differing entries. An OR (NOR) gate is preferable to an XOR (XNOR) gate if

$$\omega'_{N1}(2^N - 1) + \omega'_{N0} + R > L_s(2^{N-1}).$$

Using $L_s = \omega'_{N1} + \omega'_{N0} + R$ and simply, we get

$$2\omega'_{N1}(2^{N-1} - 1) > (\omega'_{N1} + \omega'_{N0} + R)(2^{N-1} - 1) \text{ or}$$
$$2\omega'_{N1} > \omega'_{N1}(2^N - 1) + \omega'_{N0} + R = L_s \text{ or}$$
$$\omega'_{N1} > \omega'_{N0} + R \text{ or } \omega'_{N1} > L_s/2.$$

Corollary 3: For merger of the N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, Nth order 1-weight ω'_{N1} and Nth order 0- weight be ω'_{N0}, an N – input XOR (XNOR) gate will be preferable to an -input OR (NOR) gate if

$$\omega'_{N0} < \omega'_{N1} + R \quad \text{or} \quad \omega'_{N0} < L_s/2.$$

4.3 Merger under Stochastic Dependence Line Errors

On assumption of stochastic dependence of errors, Li and Robinson [42] determine the detectable error probability estimate ε for a two-input logic function, given two input sequences of length L_s as follows:

$$\varepsilon_2 = S_1 \times B_1 / 2L_s + S_2 \times B_2 / L_s \quad (1)$$

where :

S_1 probability of single error felt at the output of the CUT;

S_2 probability of double error felt at the output of the CUT;

B_1 number of single line errors at the output of gate;

B_2 number of double line errors at the output of gate.

Here it is shown that, in the case of stochastic dependence, the detectable error probability estimate plays a role in the selection of gates for merger.

Based on the detectable error probability estimate ε of Li and Robinson as given above, the following results can be deduced in the two-sequence case.

Theorem 9: The detectable error probability estimate (second order) when two sequences Ψ_1 and Ψ_2 of length L_s are merged by using an AND (NAND) gate is

$$\varepsilon_2(AND/NAND) = S_1 \frac{H_s + 2\omega'_{21}}{2L_s} + S_2 \frac{\omega'_{20} + \omega'_{21}}{L_s}$$

where H_s is the Hamming distance, i.e., the number of bit positions the two sequences differ.

Proof: For an AND (NAND) gate, the total number of single line errors (B1) and double line errors (B2) are

$$B_1 = H_s + 2\omega'_{21} \text{ and } B_2 = \omega'_{20} + \omega_{21}.$$

Hence, the result follows by substituting the values of B_1 and B_2 in Li and Robinson's equation (1).

Theorem 10: The detectable error probability estimate (second-order) when two sequences Ψ_1 and Ψ_2 of length L_s are merged by using an OR (NOR) gate is

$$\varepsilon_2(OR/NOR) = S_1 \frac{H_s + 2\omega'_{20}}{2L_s} + S_2 \frac{\omega'_{20}+\omega'_{21}}{L_s}.$$

Proof: In the case of an OR (NOR) gate, $B_1 = H_s + 2\omega'_{20}$ and $B_2 = \omega'_{20}+\omega_{21}$. Hence, the result follows by substituting the values of B_1 and B_2 in Li and Robinson's equation.

Theorem 11: The detectable error probability estimate (second-order) when two sequences Ψ_1 and Ψ_2 of length L_s are merged by using an XOR (XNOR) gate is ε_2 (XOR / XNOR) = S_1.

Proof: In the case of an XOR (XNOR) gate, we have $B_1 = 2L_s$ and $B_2 = 0$. Therefore, ε_2 (XOR / XNOR) = S_1.

We will now generalize Li and Robinson's result on the two-sequence case and determine detectable error probability estimate ε (Nth order) for AND/NAND, OR/NOR, and XOR/XNOR gates. ε's derived for the case when N is odd. The expressions for ε's will be very much similar when N is even.

Let $\Psi_1, \Psi_2, \ldots, \Psi_N$ each of length L_s be the N output sequences at the output of a CUT. Let the corresponding Nth-order derived sequences be $\Psi'_1, \Psi'_2, \ldots, \Psi'_N$, having Nth-order 1-weight and 0-weight, ω'_{N1} and ω'_{N0}, respectively.

Let S_i, $i = 1, 2, \ldots, N$ be the probability of i - line errors. Realistically, one can assume that $S_1 > S_2 > \cdots > S_N$ and $S_1 + S_2 + \cdots + S_N = 1$. Let R be the residue, i.e., the number of bit positions in which at least two of the sequences in the bundle have different

entries. Let A_i be the number of columns in the bundle set with vertical 1-weight equal to i. Let us denote the binomial coefficient (N_i) simply by the symbol C_i.

Example 4: Consider the following five sequences of length $L_s = 12$:

$$\Psi_1 = (\ 1111\ 00\ 010101\ )$$
$$\Psi_2 = (\ 1111\ 00\ 101000\ )$$
$$\Psi_3 = (\ 1111\ 00\ 000110\ )$$
$$\Psi_4 = (\ 1111\ 00\ 110001\ )$$
$$\Psi_5 = (\ 1111\ 00\ 011100\ )$$

In this example, the fifth-order 1-weight is $\omega_{51} = 4$ and the fifth-order 0-weight is $\omega_{50} = 2$. The residue R is six since there are six bit positions in which at least two sequences differ. The vertical 1-weights are $A_1 = 1$, $A_2 = 3$, $A_3 = 1$, $A_4 = 0$, and $A_5 = 4$.

Theorem 12: The Nth-order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, Nth order 1- weight ω_{N1} and Nth order 0- weight be ω_{N0}, are merged by using an $N-$input AND (NAND) gate is

$$\varepsilon_N(AND/NAND) = \frac{\omega_{N1}}{L_s} + \frac{1}{L_s}\left[\sum_{i=1}^{N-1}\frac{S_i}{C_i}A_i + S_N\omega_{N0}\right]$$

Proof: To proof we can be express $\varepsilon_N(AND/NAND)$ as follows:

$$\varepsilon_N(AND/NAND) = \frac{S_1}{C_1 \times L_s}(C_1 \times \omega_{N1} + A_{N-1})$$
$$+\frac{S_2}{C_2 \times L_s}(C_2 \times \omega_{N1} + A_{N-2})$$
$$+\frac{S_3}{C_3 \times L_s}(C_3 \times \omega_{N1} + A_{N-3})$$

$$+\cdots$$

$$+\frac{S_{N-1}}{C_{N-1}\times L_s}(C_{N-1}\times\omega_{N1}+A_1)$$

$$+\frac{S_N}{C_N\times L_s}(C_N\times\{\omega_{N1}+\omega_{N0}\})$$

which can be simplified as shown below:

$$\varepsilon_N(AND/NAND)=\frac{S_1}{C_1\times L_s}(C_1\times\omega_{N1})$$

$$+\frac{S_2}{C_2\times L_s}(C_2\times\omega_{N1})+\cdots$$

$$+\frac{S_{N-1}}{C_{N-1}\times L_s}(C_{N-1}\times\omega_{N1})+\frac{S_N}{C_N\times L_s}(C_N\times\omega_{N1})$$

$$+\frac{S_N}{C_N\times L_s}\times A_{N-1}+\frac{S_2}{C_2\times L_s}\times A_{N-1}$$

$$+\cdots+\frac{S_{N-1}}{C_{N-1}\times L_s}\times A_1+\frac{S_N}{C_N\times L_s}\times(C_N\times\omega_{N0})$$

From which we obtain

$$\varepsilon_N(AND/NAND)=\frac{\omega_{N1}}{L_s}(S_1+S_2+\cdots+S_N)$$

$$+\frac{1}{L_s}\left[\sum_{i=1}^{N-1}\frac{S_i}{C_i}A_i+S_N\omega_{N0}\right].$$

The result obtained by substituting $S_1+S_2+\cdots+S_N=1$.

Theorem 13: The *N*th-order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, *N*th order 1- weight ω_{N1} and *N*th order 0- weight be ω_{N0}, are merged by using an $N-$ input OR (NOR) gate is

$$\varepsilon_N(OR/NOR) = \frac{\omega_{N0}}{L_s} + \frac{1}{L_s}\left[\sum_{i=1}^{N-1}\frac{S_i}{C_i}A_i + S_N\omega_{N1}\right]$$

Proof: The *N*th-order detectable error probability estimate $\varepsilon_N(OR/NOR)$ for OR (NOR) gate can be written as follows:

$$\begin{aligned}
\varepsilon_N(OR/NOR) &= \frac{S_1}{C_1\times L_s}(C_1\times\omega_{N0}+A_1) + \frac{S_2}{C_2\times L_s}(C_2\times\omega_{N0}+A_2) \\
&\quad + \cdots + \frac{S_{N-1}}{C_{N-1}\times L_s}(C_{N-1}\times\omega_{N0}+A_{N-1}) \\
&\quad + \frac{S_N}{C_N\times L_s}(C_N\times\omega_{N0}+C_N\times\omega_{N1}) \\
&= \frac{S_1}{C_1\times L_s}(C_1\times\omega_{N0}) + \frac{S_2}{C_2\times L_s}(C_2\times\omega_{N0}) \\
&\quad + \cdots + \frac{S_{N-1}}{C_{N-1}\times L_s}(C_{N-1}\times\omega_{N0}) + \frac{S_N}{C_N\times L_s}(C_N\times\omega_{N0}) \\
&\quad + \frac{S_1}{C_1\times L_s}\times A_1 + \frac{S_2}{C_2\times L_s}\times A_2 + \cdots + \frac{S_{N-1}}{C_{N-1}\times L_s}\times A_{N-1} \\
&\quad + \frac{S_N}{C_N\times L_s}(C_N\times\omega_{N1}) \\
&= \frac{\omega_{N0}}{L_s}(S_1+S_2+\cdots+S_N) + \frac{1}{L_s}\left[\sum_{i=1}^{N-1}\frac{S_i}{C_i}A_i + S_N\omega_{N1}\right].
\end{aligned}$$

Hence, the result obtained by substituting $S_1+S_2+\cdots+S_N=1$ in the above equation.

Theorem 14: The *N*th-order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, \ldots, \Psi_N$ of length L_s, *N*th order 1- weight ω_{N1} and *N*th order 0- weight be ω_{N0}, are merged by using an $N-$ input XOR (XNOR) gate is

$$\varepsilon_N(XOR/XNOR) = S_1+S_2+\cdots+S_N.$$

Proof: The *N*th-order detectable error probability estimate $\varepsilon_N(XOR/XNOR)$ for XOR (XNOR) gate can be written as follows:

$$\varepsilon_N = (XOR/XNOR) = \frac{S_1}{C_1 \times L_s}[C_1 \times (\omega_{N0} + \omega_{N1} + R)] + \frac{S_3}{C_3 \times L_s}[C_3 \times (\omega_{N0} + \omega_{N1} + R)] + \cdots + \frac{S_N}{C_N \times L_s}[C_N \times (\omega_{N0} + \omega_{N1} + R)]$$

where R is the number of bit positions. Using the fact that $L_s = \omega_{N0} + \omega_{N1} + R$, we get $\varepsilon_N(XOR/XNOR) = S_1 + S_2 + \cdots + S_N$.

Having derived the *N*th-order detectable error probability estimate for individual gates, we can establish the generalized mergeability criteria as given in the following theorem.

Theorem 15: For two N-input gates g_1 and g_2, g_1 is preferable to g_2 if and only if $\varepsilon_N(g_1) > \varepsilon_N(g_2)$.

Proof: The proof is obvious. A point worth mentioning here is that, since the computation of the closed-form expression of the *N*th-order detectable error probability estimates can be computationally intensive, they can sometimes be avoided following a heuristic approach. To illustrate this, let us consider the preference between AND (NAND) and XOR (XNOR) gates. The difference between $\varepsilon_N(AND/NAND)$ and $\varepsilon_N(XOR/XNOR)$ gives us an expression

$$\varepsilon_N(AND/NAND) - \varepsilon_N(XOR/XNOR) = \alpha + \Omega$$

where

$$\alpha = \frac{\omega_{N1}}{L_s} - (S_1 + S_2 + \cdots + S_N)$$

$$\Omega = \frac{1}{L_s}\left[\sum_{i=1}^{N-1} \frac{S_i}{C_i} A_i + S_N \omega_{N0}\right].$$

We first compute α. If $\alpha \geq 0$, we go for the AND (NAND) gate. If $\alpha < 0$,we then compute Ω, and if $\alpha + \Omega > 0$, we choose the AND (NAND) gate. If $\alpha + \Omega < 0$ an XOR (XNOR) gate is selected. Finally, if $\alpha + \Omega = 0$, we can select either an AND (NAND) or an XOR (XNOR) gate.

Example 5: Let us consider the preference between AND(NAND) and XOR (XNOR) gate. The XOR (XNOR) gate detects only the odd line faults but AND (NAND) gate also detects even line faults. All even line faults detected by AND(NAND) gate can never be detected by XOR (XNOR) gate. So, for all odd line faults, one that detects more will be crucial in deciding the ultimate preference. AND (NAND) is preferable if and only if $\varepsilon_N(AND/NAND) > \varepsilon_N(XOR/XNOR)$. Considering only the odd line faults, we have

$$\varepsilon_N(AND/NAND) = \frac{\omega_{N1}}{L_s}(S_1 + S_2 + \cdots + S_N) + \frac{1}{L_s}\left[\frac{S_1 A_{N-1}}{C_1} + \frac{S_3 A_{N-3}}{C_3} + \cdots + S_N \omega_{N0}\right].$$

Hence of all odd line faults, if AND (NAND) detects more than what XOR (XNOR) detects, then AND (NAND) is obviously desirable. That means if

$$\frac{\omega_{N1}}{L_s}(S_1 + S_2 + \cdots + S_N) + \frac{1}{L_s}\left[\frac{S_1 A_{N-1}}{C_1} + \frac{S_3 A_{N-3}}{C_3} + \cdots + S_N \omega_{N0}\right] - (S_1 + S_2 + \cdots + S_N) \geq 0. \quad (2)$$

Then AND(NAND) is preferable, which means

$$(S_1 + S_2 + \cdots + S_N)\left[\frac{\omega_{N1}}{L_s} - 1\right] \geq 0 \qquad (3)$$

We need to calculate

$$\frac{1}{L_s}\left[\frac{S_1 A_{N-1}}{C_1} + \frac{S_3 A_{N-3}}{C_3} + \cdots + S_N \omega_{N0}\right]$$

and evaluate equation (2). If the result is ≥ 0, we stop and select AND(NAND) gate.

4.4 Mergeability Criteria

Based on Section 4.2 idea for merging a set of sequences by AND (NAND), OR (NOR) and XOR(XNOR) gates can be summarized as on of finding if this set of sequences satisfies one of the following criteria:

- If $\omega_{N1} \geq L_s/2$, the obtained group sequences are merged together with an AND(NAND) gate .
- If $\omega_{N0} > L_s/2$, the obtained group sequences are merged together with an OR(NOR) gate.
- If $\omega_{N1} < L_s/2$ and $\omega_{N0} < L_s/2$, the remaining sequences which cannot be merged by either AND (NAND) or OR (NOR) gates are merged together with XOR (XNOR) gate only at this level.

CHAPTER 5
ZERO – ALIASING SPACE COMPACTION

In this chapter, we suggest the approach to designing zero-aliasing space compaction by using graph theory [43], [44], [45] and deterministic compacted test sets as well as pseudorandom testing. Space compression hardware is proposed for single stuck-line faults, extending the well-known concepts of switching theory, together with those of strong and weak compatibilities of response data outputs, in the selection of specific gates for merger of an arbitrary but optimal number of output bit streams from the MUT.

For zero-aliasing space compactor design targeting a particular fault class, it is necessary that all the faults that are detectable at the output of the MUT must also be detectable at the output of the compactor. That means adding logic gates at the output of the MUT in a way that the overall circuits (MUT + compactor) does not include any redundancy. Therefore, the objective is to have the cascaded circuit composed of MUT followed by space compactor must be irredundant and at the same time must have fewer outputs than the MUT. It is evident that with irredundant circuits all faults detectable at the MUT output are also detectable at the compactor output. The procedure for selecting logic gates for deterministic and pseudorandom testing, using gate selection criteria as developed in [1], for double as well as multiple merger of the outputs as long as the overall circuit comprised of MUT and compactor remain irredundant until either single output is produced or few output is obtained.

The relative advantages of the developed zero-aliasing compaction method over earlier techniques are obvious zero-aliasing is achieved here without any modifications of the MUT, while keeping the area overhead and signal propagation delay low contrasted with the conventional parity tree linear compactors. The maximal compaction is achieved in most cases in reasonable time utilizing some simple heuristics. It is obvious that the suggested approach is simple and robust enough in its design methodology

based on consideration of single stuck-line faults of the MUT. The approach uses less computation in designing process in successive stages. The technique utilizes mathematically sound selection criteria of merger of a number of output lines of the MUT to decide on the gate for zero-aliasing, and achieve maximum compaction ratio in design, which is evident from the extensive simulation experiments conducted on ISCAS 85 combinational and ISCAS 89 sequential benchmark circuits.

The mathematical basis used in the design process is given next.

5.1 Mathematical Basis

Property 4 Let A and B represent two of the outputs of an MUT. Let these MUT outputs be merged by a gate from the logic family AND/NAND, OR/NOR, and XOR/XNOR, and let the gate output be z_1. Then, we can envisage the following scenarios [46]:

Case 1: Fault-free (FF) outputs = Faulty (F) outputs $\Rightarrow$ Outputs A and B of the MUT do not detect any faults, and faults are hence not detectable at z_1.

Case 2: Only those faults that occur at A and B (subject to the condition of MUT having faults) 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 MUT besides A and B.

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

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

(AB) AND (NAND) incompatible.

We can similarly define two lines (AB) being s-OR (NOR) compatible, w-OR (NOR) compatible, OR (NOR) incompatible, s-XOR (XNOR) compatible, w-XOR (XNOR) compatible, and XOR (XNOR) incompatible.

Definition 4 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to

either one of the three conditions as specified by Cases 1-3, but unknown to us. If the MUT outputs A, B are merged under these conditions by an AND (NAND), OR (NOR), or XOR (XNOR) gate, then we define output 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 16 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2, so that the outputs A, B are s-AND (NAND) compatible. Similarly, let the outputs B, C be s-AND (NAND) compatible, and outputs A, C be s-AND (NAND) compatible. Then (ABC) is s-AND (NAND) compatible and all faults are detected at z_1.

Theorem 17 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2, so that the outputs $A_1, A_2, \ldots, A_m$ are s-AND (NAND) compatible. Then all faults are detected at z_1.

Theorem 18 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2, so that the outputs A, B are s-OR(NOR) compatible. Similarly,

let the outputs B, C be s-OR (NOR) compatible, and outputs A, C be s-OR (NOR) compatible. Then (ABC) is s-OR (NOR) compatible and all faults are detected at z_1.

Theorem 19 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2, so that the outputs $A_1, A_2, \ldots, A_m$ are s-OR(NOR) compatible. Then all faults are detected at z_1.

Theorem 20 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2, so that the outputs A, B are s-XOR (XNOR) compatible. Similarly, let the outputs B, C be s-XOR (XNOR) compatible, and outputs A, C be s-OR (NOR) compatible. Then (ABC) is s-XOR (XNOR) compatible and all faults are detected at z_1.

Theorem 21 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2, so that the outputs $A_1, A_2, \ldots, A_m$ are s-XOR (XNOR) compatible. Then all faults are detected at z_1.

Theorem 22 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to

pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3, so that the outputs A, B are w-AND (NAND) compatible. Similarly, let the outputs B, C be w-AND (NAND) compatible, and 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.

Corollary 22.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 3, so that the outputs $A_1, A_2, \ldots, A_m$ are w-AND (NAND) compatible. Then all faults may or may not be detected at z_1.

Theorem 23 Let A, B, C, ... be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, ... be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3, so that the outputs A, B are w-OR (NOR) compatible. Similarly, let the outputs B, C be w-OR (NOR) compatible, and 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

Corollary 23.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 3, so that the outputs $A_1, A_2, \ldots, A_m$ are w-OR (NOR) compatible. Then all faults may or may not be detected at z_1.

Theorem 24 Let A, B, C be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3, so that the outputs A, B are w-OR (NOR) compatible. Similarly, let the outputs B, C be w-XOR (XNOR) compatible, and outputs A, C be w-OR (NOR) compatible. Then (ABC) is w-XOR (XNOR) compatible and all faults may or may not be detected at z_1.

Corollary 24.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 3, 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.

In the following section, we present an algorithm to find out all the incompatible pairs of the MUT output lines for the logic gates AND/NAND, OR/NOR and XOR/XNOR, subject to conditions of Cases 1-3 as specified above.

5.2 Incompatible Pairs of Outputs

This algorithm computes all incompatible pairs of the MUT output lines for logic AND/NAND, OR/NOR, and XOR/XNOR. It is based on finding those output pairs which do not produce 100% fault coverage while merged with AND/NAND, OR/NOR, and XOR/XNOR.

Algorithm 1

Algorithm to compute all the incompatible pairs of MUT output lines for logic AND/NAND, OR/NOR, and XOR/XNOR (pairs of lines with less than 100% fault coverage):

Generating fault lists:

Generate stuck-at-0 and stuck-at-1 fault lists by applying deterministic (compacted) input test patterns to the Module Under Test. Inject this fault list to compute all the incompatible pairs.

Step 1 Finding the number of output lines:

Calculate the total number of output lines of the MUT.

Step 2 Forming combinations of output lines:

Generate all possible combinations (v_i, v_j) of MUT output lines, taking two at a time, and store all these pairs of output lines (v_i, v_j).

Step 3 Storing the pairs of lines:

Store all pairs of output lines (v_i, v_j) in database_pairs.

Step 4 Picking up one pair of lines:

Select the first pair from the list of all pairs (v_i, v_j).

Step 5 Generating new MUTs:

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

Add a new output line to the original MUT corresponding to the outputs (v_i, v_j), one at a time.

Step 6 Deleting primary outputs and forming new MUTs:

Discard the output lines v_i, v_j from the original MUT and generate a new modified MUT.

Step 7 Testing the newly generated MUTs:

Inject stuck-at-0 and stuck-at-1 logic faults into the newly generated MUT and apply test patterns.

Step 8 Checking for all detected faults:

If the fault coverage is less than 100% for MUT_AND/NAND,

store the output pair (v_i, v_j), to database AND/NAND_incompatible.

If the fault coverage is less than 100% for MUT_OR/NOR,

store the output pair (v_i, v_j), to database OR/NOR_incompatible.

If the fault coverage is less than 100% for MUT_XOR/XNOR,

store the output pair (v_i, v_j), to database XOR/XNOR_incompatible.

Step 9 Deleting the output pair:

Delete the pair just selected from the list of output lines pairs (v_i, v_j) and go to the next pair.

Step 10 Go to Step 5 and continue until all pairs are exhausted.

5.3 Maximal Compatibles of MUT Outputs

In this section we will discuss the procedure of finding all the maximal compatibility classes (MCCs) of MUT response data outputs based on the information of the incompatible pairs utilizing the graph theory of Das[43], Lee[44]. Some relevant definitions and theorems are given below.

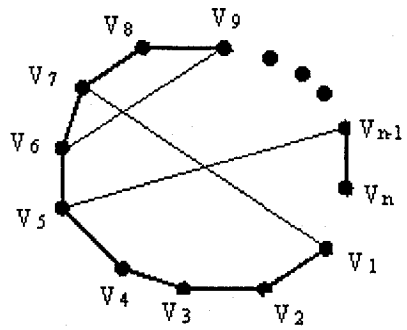


Figure 5.1 Incompatibility Graph G of n Vertices

Definition 5 Consider an undirected graph G with n vertices, v_i, i = 1, 2, ... , n. Two subgraphs G_a and G_b of G are said to be complementary to each other, if and only if, both G_a and G_b have the same set of vertices and one has edges connecting between those pairs of vertices that are not connected by edges in the other.

Definition 6 Consider a vertex v_i in an undirected graph G. The degree of v_i, $d(v_i)$, is the number of edges of G incident in v_i. The degree complement of a vertex v_i, $d'(v_i)$, is the degree of the vertex v_i in the complementary graph G'. Two vertices v_i and v_j in G are said to be minimally strongly connected, if and only if, v_i is reachable from v_j by a path of length 1. Otherwise, the vertices, if connected, are said to be nonminimally strongly connected. The degree complement of a nonminimally strongly connected pair of vertices (v_i, v_j) in G is written as $d'(v_i, v_j) = (k_1, k_2)$, where $d'(v_i) = k_1$, $d'(v_j) = k_2$.

Definition 7 A subgraph G_s of G is said to be minimally strongly connected (MSC), if and only if, every possible pair of vertices in G_s is minimally strongly connected. The subgraph G_s is said to be maximal minimally strongly connected (MMSC) if there does not exit any vertex outside of G_s which is minimally strongly connected with all the vertices of G_s.

Definition 8 Let (v_i, v_j) be a nonminimally strongly connected pair of vertices in G. Then, splitting G into two subgraphs G_i and G_j such that G_i contains the vertex v_i and G_j contains the vertex v_j is to obtain two subgraphs G_i and G_j from G such that G_i contains all the vertices of G except v_j and G_j contains all the vertices of G except v_i, both G_i and G_j having all the existing edges of G connecting between relevant pairs of vertices. Obviously, $G_i \subseteq G$; $G_j \subseteq G$.

Definition 9 For any two distinct nonminimally strongly connected pairs of vertices (v_{i1}, v_{j1}) and (v_{i2}, v_{j2}) in G, let $d'(v_{i1}, v_{j1}) = (k_1, k_2)$ and $d'(v_{i2}, v_{j2}) = (r_1, r_2)$. If $k_1 + k_2 > r_1 + r_2$, then an ordering of the degree complements of the pairs of vertices can be made as $d'(v_{i1}, v_{j1}) \geq d'(v_{i2}, v_{j2})$, whereas, if $k_1 + k_2 < r_1 + r_2$, the ordering of the degree complements of the pairs of vertices can be made as $d'(v_{i2}, v_{j2}) \geq d'(v_{i1}, v_{j1})$. If, however,

$k_1 + k_2 = r_1 + r_2$, the ordering can be made either as $d'(v_{i1}, v_{j1}) \geq d'(v_{i2}, v_{j2})$, or as $d'(v_{i2}, v_{j2}) \geq d'(v_{i1}, v_{j1})$. This kind of ordering ($\geq$) that can be established among degree complements of different nonminimally strongly connected pairs of vertices in an undirected graph is called the *magnitude ordering* of the degree complements of the pairs of vertices.

Theorem 24 Let G be an undirected graph, and let (v_i, v_j) be a nonminimally strongly connected pair of vertices in G. Let the graph G be split around (v_i, v_j) into two subgraphs G_i and G_j and let this process of splitting around nonminimally strongly connected pairs of vertices be iteratively applied to G_i and G_j and to all their subgraphs, until in the resulting subgraphs there exist no more nonminimally strongly connected pairs of vertices. The final set of these subgraphs then includes all the maximal minimally strongly connected (MMSC) subgraphs of G.

Proof: Since (v_i, v_j) nonminimally strongly connected pair of vertices in G, in no MMSC sub graph of G, both of v_i and v_j can occur. Thus, splitting G into two subgraphs G_i and G_j such that G_i contains all the vertices of G except v_j, and G_j contains all the vertices of G except v_i, eliminates the possibility of joint occurrence of v_i and v_j in any subgraph. Accordingly, by iterative application of this procedure to all the resulting subgraphs of G until in the subgraphs there does not exist any nonminimally strongly connected pair of vertices will finally lead to a set of subgraphs which will include all the MMSC subgraphs of G.

Thus, given an undirected graph G, by the application of the aforesaid iterative procedure of splitting into subgraphs, all the MMSC subgraphs of G can be obtained. But this procedure, though straight forward, has the disadvantage of needing, in general, more computation time; the procedure may generate non-MMSC and duplicate subgraphs. The computational requirements my be reduced to a great extent if, at every stage, the process of splitting into subgraphs is carried out around those nonminimally strongly connected pair of vertices which have the highest degree complements in the magnitude ordering.

Theorem 25 Let G be an undirected graph, and let (v_i, v_j) be a nonminimally strongly connected (NMSC) pair of vertices of G having the highest degree complement in the magnitude ordering. If now the graph G is split around (v_i, v_j) into two subgraphs G_i and G_j, then in the resulting subgraphs the number of nonminimally strongly connected pairs of vertices will always be less than that when G will be split into subgraphs around any other nonminimally strongly connected pair having non-highest degree complement in the magnitude ordering.

Proof: Let the degree complement of the nonminimally strongly connected pair of vertices (v_i, v_j) b e $\overline{d}\ (v_i, v_j) = (k_1, k_2)$. Let (v_s, v_t) be another nonminimally strongly connected pair of vertices of G for which the degree complement is $\overline{d}\ (v_s, v_t) = (r_1, r_2)$. We assume that $\overline{d}\ (v_i, v_j) \geq \overline{d}\ (v_s, v_t)$, that is,

$$k_1 + k_2 \geq r_1 + r_2 \qquad (4)$$

Let G_s and G_t be the subgraphs obtained on splitting G around (v_s, v_t). If now h_i, hj, h_s, and h_t be the nonminimally strongly connected pairs of vertices in the subgraphs G_i, G_j, G_s, and G_t, respectively, we like to show that

$$h_i + h_j \leq h_s + h_t. \qquad (5)$$

Observe that h_i, hj, h_s, and h_t can be expressed as

$$h_i = h\text{-}k_2, \qquad h_j = h\text{-}k_1,$$
$$h_s = h\text{-}r_2, \qquad h_t = h\text{-}r_1, \qquad (6)$$

where h is the number of nonminimally strongly connected pairs of vertices in G. Now, by substitution of (3) into (5) above, it is easy to shows that (2), that is, $h_i + h_j \leq h_s + h_t$, is equivalent to (6), that is , $k_1 + k_2 \geq r_1 + r_2$.

In order to cancel non-MMSC and duplicate subgraphs that may be generated, and also to avoid generation of these subgraphs in the process of splitting an undirected graph G into subgraphs, we use the following theorem.

Theorem 26 In the process of successively splitting an undirected graph G into subgraphs around nonminimally strongly connected pairs of vertices, let G_i and G_j are

any two subgraphs obtained at different stages such that $G_i \subseteq G_j$, but G_i is not derived from G_j. Then, in finding only maximal minimally strongly connected (MMSC) subgraphs, the subgraph G_i may be discarded in general.

Proof: When there is no nonminimally strongly connected pair of vertices in both G_i and G_j, and $G_i \subseteq G_j$, evidently G_i is either a non-MMSC subgraph of G, or a duplicate subgraph. When there exist nonminimally strongly connected pairs of vertices in G_i and G_j, even then, by further splitting of G_i, no new information can be obtained, because $G_i \subseteq G_j$. Thus, the sugraph G_i may be discarded in all cases.

Algorithm 2

Algorithm to compute all the maximal compatibility classes (MCCs) of MUT output lines for logic AND/NAND, OR/NOR, and XOR/XNOR, by using the incompatible pairs of the MUT output lines for the logic gates AND/NAND, OR/NOR and XOR/XNOR, based on the graph theory concepts as outlined above [43] [44].

Step 1 Finding the degree of complements of NMSC pairs:
From the graph G (compatibility graph) representative of the incompatible pairs, find the magnitude ordering of degree complements of the nonminimally strongly connected (NMSC) pairs of outputs of the MUT in G.

Step 2 Selecting an NMSC with highest degree of complements:
Select an NMSC pair of outputs (v_i, v_j) in G, where (v_i, v_j) has the highest degree complement in the magnitude ordering. If more than one pair of outputs has the highest degree complement, select any one of these output pairs, (v_i, v_j). Split G around (v_i, v_j) into two subgraphs G_i and G_j such that G_i contains all the outputs (vertices) of G except v_j and G_j contains all the outputs (vertices) of G except v_i.
a) Consider the subgraph G_i; check if there exists a subgraph G_k from which G_j is not derived, contains G_i. If so, discard the subgraph G_j; if not, take G_i and go to Step 1.
b) Consider the subgraph G_j; check if there exists a subgraph G_m from which G_j is not derived, contains G_j. If so, discard the subgraph G_j; if not, take G_j and go to Step 1.

Step 3 Follow Steps 1 and 2:

Follow Steps 1 and 2 iteratively until in all the resulting subgraphs there does not exist any nonminimally strongly connected (NMSC) pair of outputs (vertices). The final set of subgraphs then includes all the maximal minimally strongly connected (MMSC) subgraphs of G.

Step 4 Finding all the MCCs:

In the set of subgraphs obtained after Step 3, check if any subgraph is contained in another subgraph for possible cancellation of non-MMSC subgraphs. The resultant set, after cancellation, if any, gives all the maximal minimally strongly connected (MMSC) subgraphs of G.

In Figure 5.2 shows the flow chart of the algorithm to find out the maximal compatibility classes (MCCs).

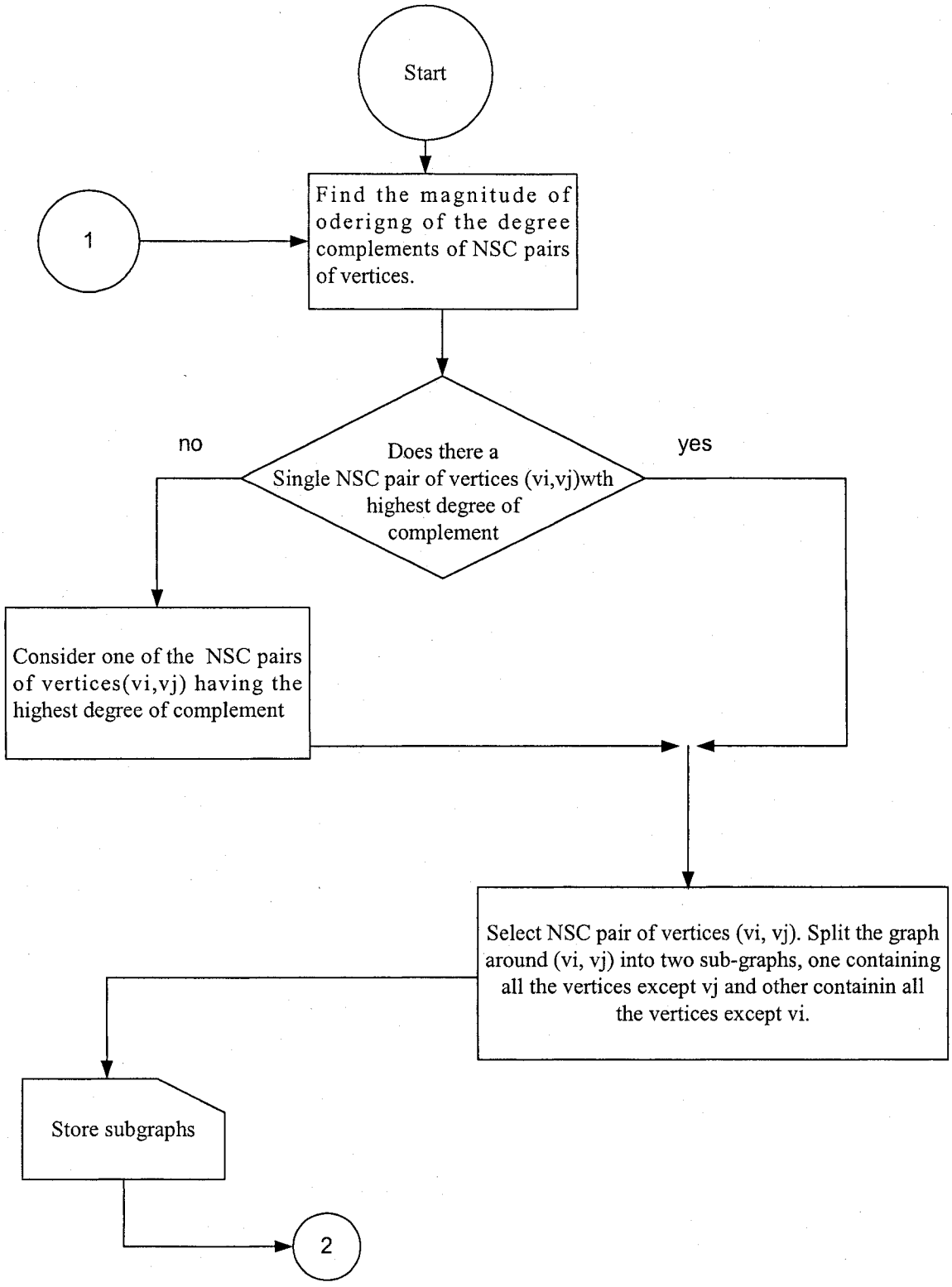


Figure 5.2 Flow chart 1 of the Algorithm 2

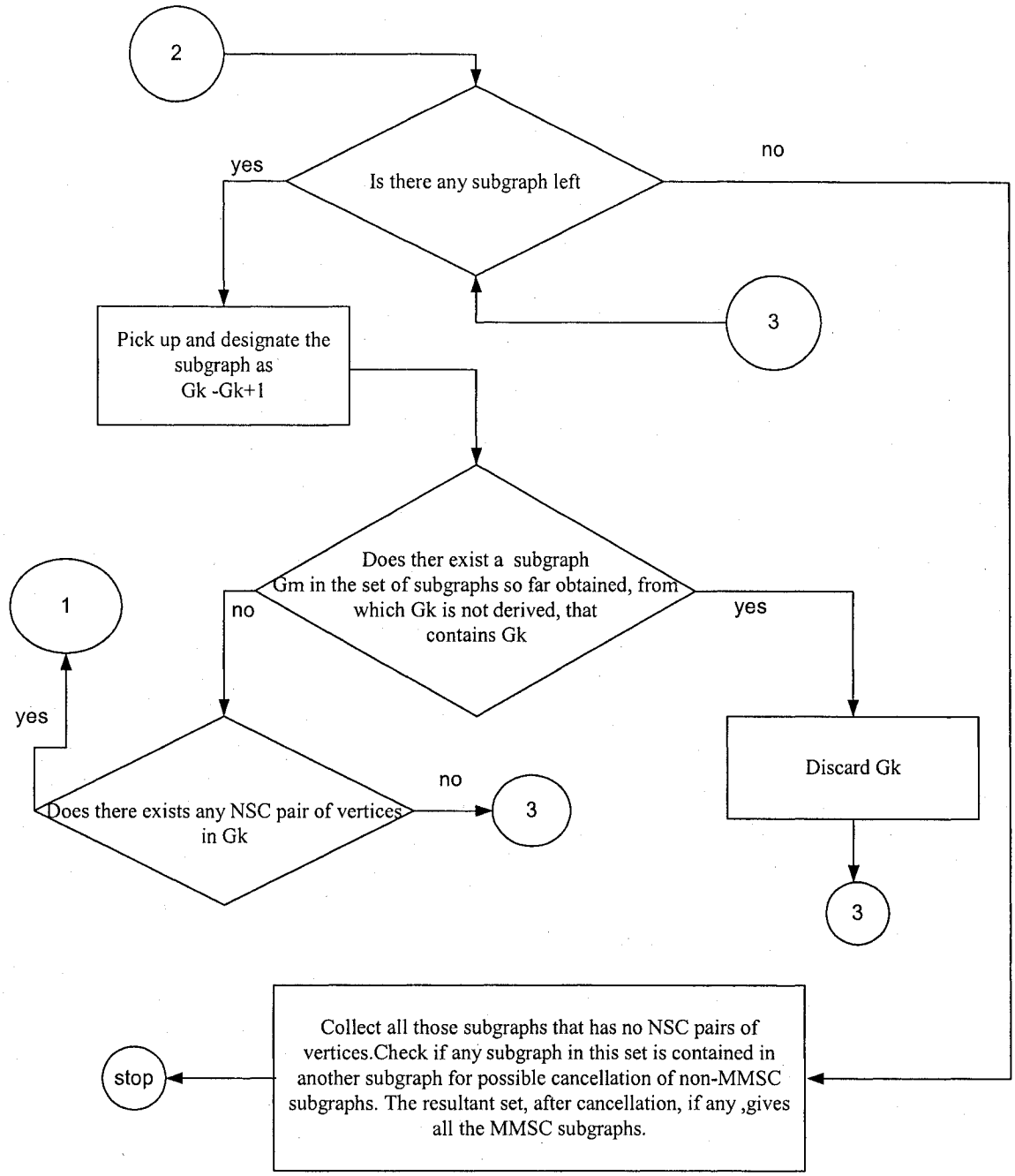


Figure 5.3 Flow chart 2 of the Algorithm 2

5.4 Constructing Aliasing-Free Compactors

This algorithm uses the knowledge of maximal compatibility classes (MCCs) to construct zero-aliasing space compressors [1].

Algorithm 3

Step 1 Defining the maximum compression ratio:
Define the possible maximum number of stages in the space compaction trees at the MUT output.

Step 2 Determining compaction tree at every stage until a minimal number of output line is obtained:
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 Computing all the MCCs for logic AND/NAND, OR/NOR and XOR/XNOR:
Find the sets of all maximal compatibility classes (MCCs) from the MUT for logic AND/NAND, OR/NOR, and XOR/XNOR, employing Algorithm 2.
Total number of MCCs = number of MCCs (AND/NAND) + number of MCCs (OR/NOR) + number of MCCs (XOR/XNOR).

Step 4 Making decision regarding which of the MCCs to pick up:
Select an MCC_i with large (possibly largest) number of output lines from the set of MCCs. Select the next large class during subsequent iteration, if 100% fault coverage is not achieved in the previous iteration from the same MUT.

Step 5 Constructing circuit with the selected MCC_i :
Merge selected output lines of the MCC_i using appropriate logic gates (AND/NAND, OR/NOR, or XOR/XNOR).

Step 6 Adding a new output line :

Add a new output line corresponding to the selected merged outputs of MCC_i.

Step 7 Discarding the MCCs already used :

Discard all the MCCs (AND/NAND) and MCCs (OR/NOR) and MCCs (XOR/XNOR) corresponding to the selected and merged lines from the database.

Remaining output lines = remaining output lines – selected output lines in MCC_is; total number of MCCs = number of MCCs (AND/NAND) + number of MCCs (OR/NOR) + number of MCCs (XOR/XNOR).

Step 8 Picking up another MCC:

From remaining output lines and remaining MCCs (AND/NAND) and MCCs (OR/NOR) and MCCs (XOR/XNOR), randomly pickup another maximal compatibility class (MCC) with the largest number of lines not being merged previously.

Step 9 Merging selected output lines:

Merge the selected output lines in MCC_j using appropriate logic gate (AND/NAND) or (OR/NOR) or (XOR/XNOR).

Step 10 Creating a new output line:

Add a new output line corresponding to the selected merged outputs of MCC_j.

Step 11 Discarding the already used MCCs:

Discard the output lines that are already used in MCC_j. Discard all the MCC_is (AND/NAND) and MCC_is (OR/NOR) and MCC_is (XOR/XNOR) corresponding to the selected and merged lines from the database. Remaining output lines = remaining output lines – selected output lines in MCC_is; total number of MCCs = number of MCCs (AND/NAND) + number of MCCs (OR/NOR) + number of MCCs (XOR/XNOR).

Step 12 Continue the loop until:

Go to Step 8 as long as there are MCCs in the sets, and enough output lines.

Step 13 Finding the remaining output lines:

Find all the remaining output lines that do not belong to any of the selected MCCs (AND/NAND) or MCCs (OR/NOR) or MCCs (XOR/XNOR).

Step 14 Merging all the remaining lines with XOR/XNOR:

If possible, merge all the remaining lines with logic gate XOR/XNOR gate.

Step 15 Adding new output lines:

Add a new output line corresponding to these selected merged outputs.

Step 16 Injecting faults to the circuit:

Inject stuck-at-0 and stuck-at-1 logic faults into the newly generated MUT (original MUT with COMPACTOR hardware).

Step 17 Appling input test patterns :

Apply input test vectors (deterministic compacted test sets with particular seed α) at the primary input of the MUT.

Step 18 Computing the fault coverage:

Compute fault coverage (FC) at the outputs of the MUT + Compactor.

Step 19 Checking for 100% fault coverage:

If the percentage of fault coverage is 100%, then replace the old MUT with the new MUT and go to Step 2 for generating the next stage of the compactor.

Step 20 Merging remaining output lines if FC< 100%

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

Step 21 Adding new lines:

Add new output lines corresponding to the selected merged outputs.

Step 22 Injecting faults to the circuit:

Inject stuck-at-0 and stuck-at-1 logic faults into the newly generated MUT (original MUT with COMPACTOR hardware).

Step 23 Appling input test patterns:

Apply input test vectors (deterministic compacted test sets with particular seed α) at the primary input of the MUT.

Step 24 Computing the fault coverage:

Compute fault coverage (FC) at the outputs of the MUT + Compactor.

Step 25 Checking the fault coverage:

If fault coverage is less than 100%, then continue to work on the same MUT. Go to Step 4 for selecting a new MCC_k.

Step 26 Checking if all the injected faults are detected:

If fault coverage is 100%, then replace the old MUT with the new MUT, and go to Step 2 for computing the next stage and subsequent stages of the compactor.

Example Let us consider c432 ISCAS 85 benchmark combinational circuit shown in Figure 5.4, it has thirty-six inputs and seven outputs and one hundred and sixty logic gates. To test the circuit we injected five hundred and twenty stuck-at-0 and stuck-at-1 faults, fifty-one compacted test patterns were injected at the primary inputs to detect all the faults. The fault-free output patterns and the corresponding detected stuck-at faults are shown below in Table 5.1.

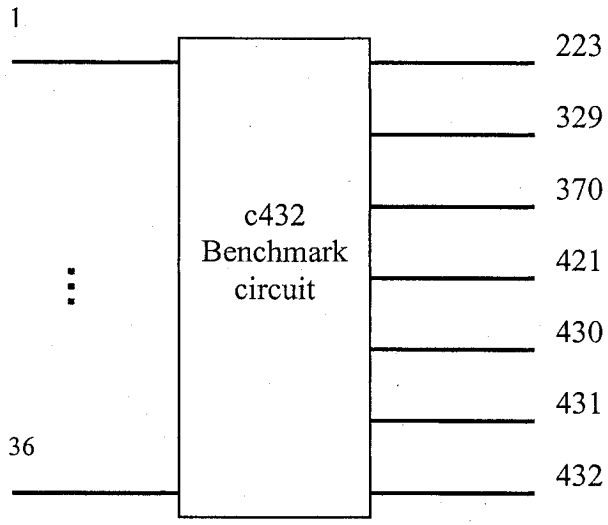


Figure 5.4 c432 Benchmark Circuit.

Five hundred twenty 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.

Table 5.1 Input Test Sets, Fault-Free Outputs, and the Corresponding Detected Stuck-at Logic Faults For c432 Benchmark Circuit.

Test Vectors	Outputs	No. of Faults detected
test 1: 110010011110001110111000110010110101	1101010	46 faults detected
test 2: 111010000001111010010111001101011000	0101001	67 faults detected
test 3: 100101001011000111010110100111001100	1111000	25 faults detected
test 4: 011000011111011110011100111110110101	1010000	42 faults detected
test 5: 001101001011010001100110101011101010	1011100	39 faults detected
test 6: 000011110111101011010011011110000010	1111100	8 faults detected
test 7: 100101111101111101010100001110110011	0111100	8 faults detected
test 8: 110100100001001101100111100111101001	1001011	29 faults detected
test 9: 101110100000011011110110011000000110	1011100	9 faults detected
test 10: 000100011011100010110101011000001000	0111011	17 faults detected
test 11: 011010010100110111000101110110011111	1101101	13 faults detected
test 12: 010010100000000011001110001110011110 0	1100000	13 faults detected
test 13: 111111110010100011110110100010001010	1011100	2 faults detected
test 14: 011110000011100010001000001011110101	1011001	14 faults detected
test 15: 011101101000100011101011001001010001	1111001	2 faults detected
test 16: 101001011111011110001010110001010001	1111010	5 faults detected
test 17: 011101111101111101101110011111100111	1010011	8 faults detected
test 18: 001010111101111100010011001000010100	0111000	3 faults detected
test 19: 111011101000111001110010001101010100	1011101	7 faults detected
test 20: 100100000011101110011001111010101000	0001010	17 faults detected
test 21: 100111011010011001111011101111001101	1101000	8 faults detected
test 22: 010001101100001100010011000001000000	1110110	12 faults detected
test 23: 110110101010111111111100111000101001	1011010	2 faults detected
test 24: 100110010110100000111011110000010010	0111010	4 faults detected
test 25: 100000100100000100000100010011111010	1111110	1 faults detected
test 26: 000010100110101111111100000110111010	1001110	9 faults detected
test 27: 000000100110001010100011001011111111	1011110	6 faults detected
test 28: 001000010010111000110101001100000111	1011101	7 faults detected
test 29: 101011111011001101011001000000001110	1011000	4 faults detected
test 30: 000000000000000000000000000000000000	0000000	18 faults detected
test 31: 000101011100000101000011001001011010	1101001	1 faults detected
test 32: 011110010000100000111000100111110100	1000000	3 faults detected
test 33: 011010011111011100101001101011111001	1000001	3 faults detected
test 34: 110101110101110010010001010101110110	0110010	1 faults detected
test 35: 111101110111000111011000001100101010	0101100	12 faults detected
test 36: 010010010110010011111111000101000110	1100101	1 faults detected
test 37: 011010111100010111010000110010011001	1101101	1 faults detected
test 38: 001000100100010101110100011101010100	1111110	1 faults detected
test 39: 000101110110111111010001011100111111	1001101	4 faults detected
test 40: 100101101011110100110010000101111011	0111101	5 faults detected
test 41: 100100110110100111001111010110001001	1011011	3 faults detected
test 42: 010001100110011100000101111110001110	1110000	2 faults detected
test 43: 010110101110111010101001110100001111	1100000	1 faults detected
test 44: 110001011001001101100010011101010000	1101111	18 faults detected
test 45: 110010001001111010100001101100110010	0100000	1 faults detected
test 46: 111000110101000101011111111101011101	0111110	4 faults detected
test 47: 010100110110111001001000000111000010	1110000	1 faults detected
test 48: 111101000011111000010011100101110110	0111111	6 faults detected
test 49: 000111000100000010101011000101111010	1111110	1 faults detected
test 50: 001001110111111001001000010010010101 0	1011111	3 faults detected
test 51: 010110110101000011110110000001110111	1100000	3 faults detected

The incompatible 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 Algorithm 1 and using deterministic input test set shown in Table 5.1. The incompatible pairs for logic AND/NAND, and OR/NOR are shown in Table 5.2 and 5.3 respectively.

Table 5.2 Incompatible Pairs for logic AND/NAND.

(O_i , O_j)
(223, 421)
(329, 421)
(370, 421)
(421, 430)
(421, 431)
(421, 432)
(430, 431)
(430, 432)
(431, 432)

Table 5.3 Incompatible Pairs for Logic OR/NOR

(O_i , O_j)
(223, 421)
(223, 432)
(329, 421)
(329, 432)
(370, 421)
(370, 432)
(421, 430)
(421, 431)
(421, 432)
(430, 431)
(430. 432)
(431, 432)

For logic XOR/XNOR we get zero incompatible pairs, thus all the pairs of output lines formed maximal compatible class (MCC) for XOR/XNOR.

To find out the maximal compatibility classes (MCCs) we used the incompatibility pairs with the graph theory [52], [53] Algorithm 2. By using the incompatibility response data in Table 5.2 and 5.3 for ISCAS85 benchmark circuit, we get the following maximal compatibility classes (MCCs) as shown in Table 5.4.

Table 5.4 Maximal Compatibility Classes for ISCAS85 Benchmark Circuit

MCCs(AND/NAND)	MCCs(OR/NOR)	MCCs(XOR/XNOR)
421 223, 329, 370, 430 223, 329, 370, 431 223, 329, 370, 432	421 432 223, 329, 370, 430 223, 329, 370, 431	223, 329, 370, 421, 430, 431, 432

To construct the aliasing free compactor Algorithm 3 is used together with the maximal compatibility classes (MCCs). A maximal compatibles set, consist of maximum output lines, is randomly picked up from the Table 5.4 and the corresponding compactor is added to the MUT. The new circuit (MUT + Compactor) is the tested for stuck-at logic faults, the faults are injected into the MUT and input test vectors is applied at the input of MUT + Compactor. The output response pattern is then compared with the corresponding fault-free signature, if all faults injected are detected at the output of the MUT + Compactor, then the constructed circuit will be retained and same process will be repeated over until a compaction tree with maximal compaction ratio is obtained. Otherwise, another MCC set or subset is randomly picked up again from the Table 5.4 for processing, finally when only two output lines are remain, then these two outputs will be merged by XOR/XNOR gate and then tested for hundred percent fault coverage.

We have generated two compaction trees for ISCAS85 benchmark circuit and both the MUT + Compactor are aliasing free and give 100% fault coverage. In both the cases maximal compaction ratio is obtained, and Table 5.5 shows both compaction trees with maximal compaction ratio.

Table 5.5 Compaction Tree for c432 ISCAS85 Benchmark Circuit with Maximal Compaction Ratio

Compaction Tree #1	**Compaction Tree #2**
433 = AND(223,329,370,430)	433 = OR(223, 329, 370, 431)
434 = XOR(431,432)	434 = XOR(421, 430)
435 = OR(421,433)	435 = XOR(432, 433)
436 = XOR(434,435)	436 = XOR(434, 435)
SINGLE OUTPUT = (436)	**SINGLE OUTPUT = (436)**

Two zero-aliasing compactors for c432 ISCAS85 benchmark circuit are shown in Figures 5.5 and 5.6.

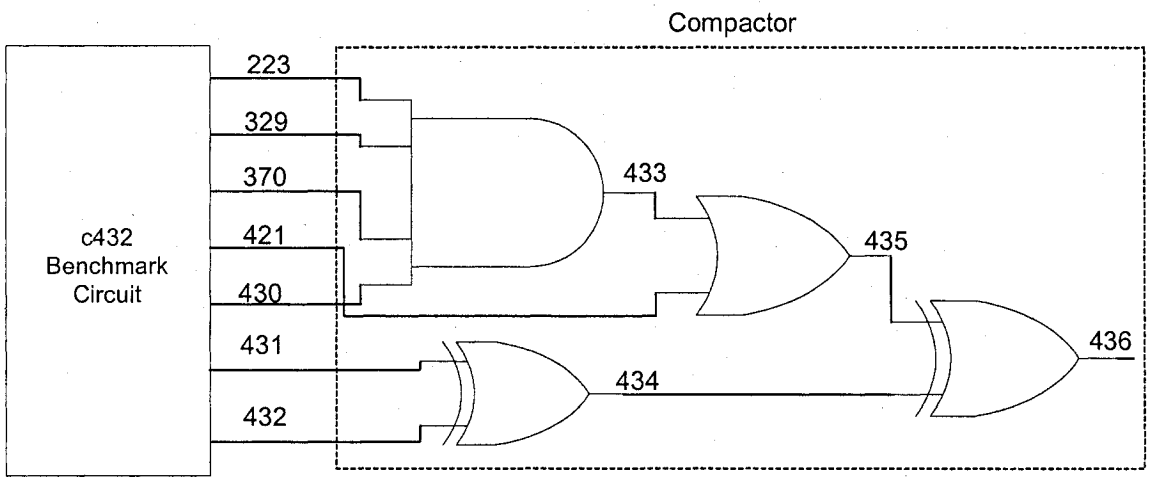


Figure 5.5 Zero-aliasing Compactor #1 for c432 ISCAS85 Benchmark Circuit

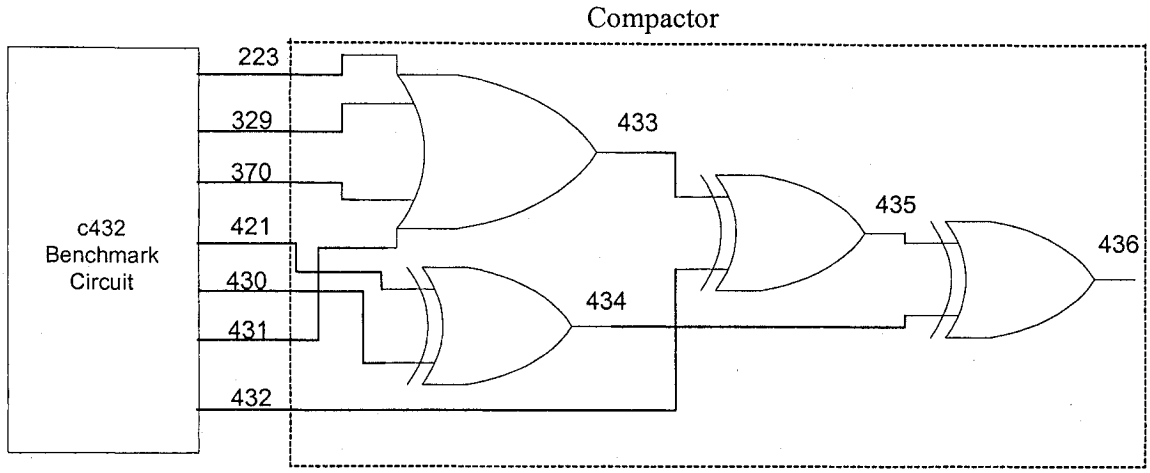


Figure 5.6 Zero-aliasing Compactor #2 for c432 ISCAS85 Benchmark Circuit

Table 5.6 Input Test Sets, Fault-Free Outputs, and the Corresponding Detected Stuck-at Logic Faults For c432 Benchmark Circuit With Compactor.

Test Vectors	Outputs	No. of Faults detected
test 1: 111111111111001000001111101001110000	0	28 faults detected
test 2: 100111011011110010001001010000111010	0	26 faults detected
test 3: 100000011001100010111111110110011011	0	7 faults detected
test 4: 000110111110000010110001011110101100	1	12 faults detected
test 5: 010000000001110001010100011010111100	1	16 faults detected
test 6: 111100001000110111001100111111001001	0	10 faults detected
test 7: 101000001010011001110000001100010011	0	4 faults detected
test 8: 011111011001101110111000101101101110	1	38 faults detected
test 9: 110100111111110011111011111100111110	1	48 faults detected
test 10: 111110010110001110110110100111101000	0	11 faults detected
test 11: 011111001001111110100011111000010001	0	8 faults detected
test 12: 110110110111010001101011001011111000	0	5 faults detected
test 13: 000000000111111011100100110111101000	1	20 faults detected
test 14: 001100001100101001110011110001001000	0	1 faults detected
test 15: 001000110111111110111110000010101110000	1	26 faults detected
test 16: 011111010011100100100010111011011010	0	5 faults detected
test 17: 000110000010101000101010110001010101	0	1 faults detected
test 18: 011011110000100110011101101000100001	1	23 faults detected
test 19: 111110100100100011010000111100010111	1	9 faults detected
test 20: 010001010111100100000011100000100110	1	3 faults detected
test 21: 001010001100100001101100010110000010	1	7 faults detected
test 22: 100110101000011011110001111011111100	1	12 faults detected
test 23: 100100110101111000000111100101110010	1	13 faults detected
test 24: 111110011111011010111100001100100111	1	5 faults detected
test 25: 010000000110110101011110111001100101	0	1 faults detected
test 26: 000001001101100001001011001000000010	1	5 faults detected
test 27: 101011111010000110111011100101001001	1	2 faults detected

Table 5.7 Input Test Sets, Fault-Free Outputs, and the Corresponding Detected Stuck-at Logic Faults For c432 Benchmark Circuit With Compactor.

Test Vectors	Outputs	No. of Faults detected
test 28: 10110000100010001010110111001110111	1	2 faults detected
test 29: 11111100100101101010101101111001 1001	1	8 faults detected
test 30: 01100100110100111001011111111 1000001	0	8 faults detected
test 31: 01100100111101011111100010100001 10010	1	6 faults detected
test 32: 01110011001001111011000111001011 1100	0	7 faults detected
test 33: 10111110111111011110010111001011010	1	2 faults detected
test 34: 11010111110000101001000110010001 0110	1	11 faults detected
test 35: 10101001101010010011010111010001 1100	0	4 faults detected
test 36: 10010101011110101110100111010110 1111	1	3 faults detected
test 37: 10010110001110000011100000001011 0011	1	9 faults detected
test 38: 10010100100101001000101000011101 0000	0	2 faults detected
test 39: 00000000000000000000000000000000 0000	0	3 faults detected
test 40: 11000001110010111101110010001 0011110	0	4 faults detected
test 41: 11100111001100101010110010000010 1110	1	4 faults detected
test 42: 11101111101001110001001111111 00000111	0	10 faults detected
test 43: 11110000111111011100000001001000 0101	0	5 faults detected
test 44: 00011001001000101110101011110100 0101	1	8 faults detected
test 45: 01010110010101001000110010000 1110011	0	1 faults detected
test 46: 00011000111010110000111010111010 0010	0	4 faults detected
test 47: 00001101101011100000001100010011 1101	1	2 faults detected
test 48: 01011011011011000001011111100 1101000	0	2 faults detected
test 49: 01001100011000010011101000000 1111100	1	1 faults detected
test 50: 11011101010110000001110101001 1001001	0	2 faults detected
test 51: 11110011011010011100101110010 0100010	1	5 faults detected
test 52: 10110110101111010001100010010 0111011	0	2 faults detected
test 53: 00000001100000101011111010110 0011111	1	10 faults detected
test 54: 01011010100000000101110000110 1000011 0	0	1 faults detected
test 55: 01011011110110101101110010100 1111001	0	4 faults detected
test 56: 10010100101100000011111100010 0010111	1	2 faults detected
test 57: 10001001110101010101100101111 1101011	0	1 faults detected
test 58: 00100011001000111110000001110 0111011	0	2 faults detected
test 59: 11001110110111101000111000001 0100011	0	2 faults detected
test 60: 10010100010101001010001101000 0011000	0	1 faults detected
test 61: 11001001100010111101010100010 1010100	0	4 faults detected
test 62: 01111111100101001011010011111 0010000	1	3 faults detected
test 63: 10100000101110101111100001001 0000110	0	2 faults detected
test 64: 01000101111010110110100001001 0111011	0	2 faults detected
test 65: 11101000001101111000001010110 0011001	1	8 faults detected
test 66: 11101000100111111011101000010 0110001	1	2 faults detected
test 67: 00011011010010000010101010001 0111100	1	3 faults detected
test 68: 10110010101100011111100101110 0110000	0	2 faults detected
test 69: 10101010101101010110011110010 1011011	1	2 faults detected
test 70: 10010000000100001110000011110 0111010	1	1 faults detected
test 71: 01111101000110110100111000110 1000000	1	2 faults detected
test 72: 01100111011111001011101110000 1100110	0	1 faults detected
test 73: 11001111011011011110100001001 1101011	0	1 faults detected
test 74: 00111000100010100001011111010 0110000	1	2 faults detected
test 75: 11111101010011101001101000110 1100001	0	1 faults detected

Tables 5.6 and 5.7 show the test result of c432 ISCAS85 benchmark circuit with compactor. To test the circuit we injected five hundred and twenty stuck-at-0 and stuck--at-1 faults, seventy five compacted test patterns were injected at the primary inputs to detect all the faults. The fault-free output patterns and the corresponding detected stuck-at faults are shown in Tables 5.6 and 5.7.

For the 36 bits input test patterns, the single bit output signature is observed and compared with the pre-computed fault-free stored signature, and the detected faults are recorded when faulty and fault-free signature mismatched.

CHAPTER 6
EXPERIMENTAL RESULTS

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 for deterministic compact and pseudorandom test sets. For each 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, number of maximal compatibility classes at the first compaction stage, and compaction ratio. The results are listed in the figures and tables in Section 6.1 for ISCAS 85 benchmark combinational circuit and in Section 6.2 for ISCAS 89 benchmark full scan sequential circuits.

6.1 Experimental Results with ISCAS 85 Benchmark Circuit

To demonstrate the feasibility of the proposed space compactor, independent simulation were performed on ISCAS 85 benchmark combinational circuit. We have used ATALANTA [47] (fault simulation program developed at Virginia Polytechnic Institute and State University) to generate fault free output sequences, which is needed to construct the space compactors. ATALANTA is also used to test the benchmark circuits using reduced test sets accompanied with random testing session. FSIM [48] fault simulation program is used to generate pseudorandom test sets. We used SUN workstation with SunOS 5.9 to do the extensive testing of the ISCAS 85 benchmark combinational circuits. The results are listed in the following tables and figures.

Tables 6.1 and 6.2 show the deterministic test results with all fault injected and with detected fault injected to the ISCAS 85 benchmark combinational circuits respectively.

Table 6.1 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	7	22	2	100.00
c432	75	524	7	99.237
c499	67	758	32	98.945
c880	118	940	26	100.00
c1355	126	1574	32	99.492
c1908	192	1879	35	99.521
c2670	192	2747	140	95.741
c3540	253	3428	22	96.004
c5315	214	5350	123	98.897
c6288	53	7744	32	99.561
c7552	368	7550	108	98.252

Table 6.2 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using ATALANTA Without Space Compactor and Detected Faults Injected

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
c17	7	22	2	100.00
c432	76	520	7	100.00
c499	66	750	32	100.00
c880	107	940	26	100.00
c1355	105	1566	32	100.00
c1908	184	1870	35	100.00
c2670	182	2630	140	100.00
c3540	253	3291	22	100.00
c5315	197	5291	123	100.00
c6288	53	7710	32	100.00
c7552	376	7419	108	100.00

Tables 6.3 and 6.4 show the pseudorandom test results 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 6.5 and 6.6 show the results with all faults injected and detected faults respectively.

Table 6.3 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	100.00
c432	1024	524	7	99.237
c499	1024	758	32	98.945
c880	6688	940	26	100.00
c1355	10016	1574	32	99.492
c1908	10016	1879	35	99.521
c2670	1000000	2747	140	95.741
c3540	100000	3428	22	96.004
c5315	10016	5350	123	98.897
c6288	192	7744	32	99.561
c7552	10000000	7550	108	98.252

Table 6.4 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	544	520	7	100.00
c499	1312	750	32	100.00
c880	5480	940	26	100.00
c1355	2124	1566	32	100.00
c1908	29472	1870	35	100.00
c2670	6378144	2630	140	100.00
c3540	38848	3291	22	100.00
c5315	4576	5291	123	100.00
c6288	128	7710	32	100.00
c7552	10000000	7419	108	99.407

Table 6.5 Simulation Results of ISCAS 85 Combinational Benchmark Circuits Using FSIM Without Space Compactor and all Faults Injected 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 6.6 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 6.1 shows the number of applied test patterns used for getting the maximum

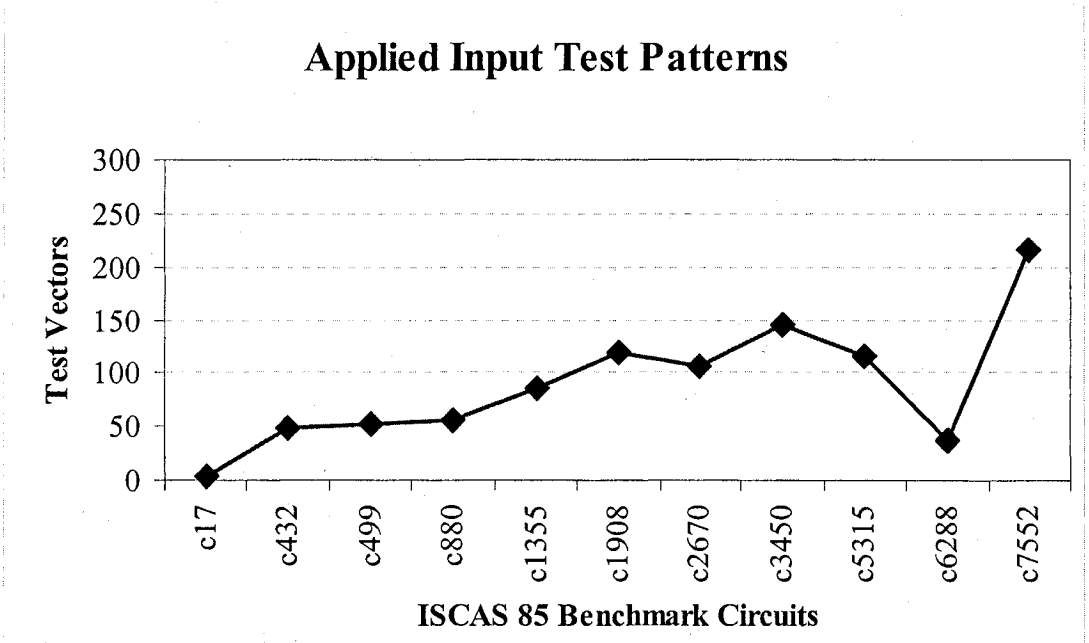


Figure 6.1 Input Test Patterns For ISCAS 85 Benchmark Circuits Using FSIM Without Space Compactor and Detected Faults Injected and Tested With Compacted Test Sets.

fault coverage with the circuit without compactor and the detected faults were injected and tested with FSIM.

To design the compactors we need the AND/NAND, OR/NOR and XOR/XNOR incompatibles and the maximal compatibility classes (MCCs). The numbers of incompatibles and maximum compatibility classes for ISCAS 85 benchmark circuits are shown in Table 6.7 for the first compaction stage, computed by using deterministic testing.

Table 6.7 Compatibility Classes for ISCAS 85 Benchmark Circuits Computed at the First Compaction Stage Using Deterministic Testing.

Circuit Name	NIL	NOL	NGM	NAI	NOI	NXI	NAM	NOM	NXM
c17	5	2	6	0	1	0	1	2	1
c432	36	7	160	9	12	0	4	4	1
c499	41	32	202	10	2	4	48	4	16
c880	60	26	383	25	30	2	64	180	4
c1355	41	32	546	55	33	34	3755	1002	256
c1908	33	25	880	23	24	0	7	2	1
c2670	233	140	1269	204	71	9	2467	962	4
c3540	50	22	1669	37	27	1	29	14	2
c5315	178	123	2485	489	293	3	21401	14809	418
c6288	32	32	2448	3	2	1	4	4	2
c7552	207	108	3719	392	251	2	11432	6480	2

To demonstrate the feasibility of the proposed zero-aliasing space compaction schemes, various testing and independent simulation were conducted with the ISCAS 85 benchmark combinational circuits and studied from various angles. One of our objectives was to keep the hardware overhead low to avoid the propagation delay and we targeted it for application to digital embedded cores-based system-on-chips. 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 6.8. The hardware overhead is calculated based on the formula below.

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

Table 6.8 Estimates of the Hardware Overhead for ISCAS 85 Benchmark Circuits Using Deterministic Testing

Circuit Name	NGC	NFC	AFC	NGM	AFM	NGC+NGM	AOP
c17	1	2	2.00	6	2.00	7	14.29
c432	4	10	2.50	160	2.10	164	2.90
c499	7	38	5.42	202	2.02	209	8.98
c880	5	30	6.00	383	1.9	389	4.05
c1355	9	40	4.44	546	1.95	555	3.69
c1908	8	32	4.00	880	1.70	888	2.12
c2670	14	153	10.93	1269	1.64	1283	7.27
c3450	9	29	3.22	1669	1.76	1678	0.98
c5315	23	145	6.30	2307	1.90	2330	3.27
c6288	3	34	11.33	2416	1.99	2419	0.70
c7552	14	123	8.75	3513	1.75	3527	1.98

With conventional switching theory concept, frequency ordering, concept of strong and weak compatibilities of the response data output and utilizing 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 in the design, it is evident from simulation experiments conducted on ISCAS 85 combinational benchmark circuits with compactors. In the following tables and figures we show the fault coverage, compaction ratio, the CPU simulation time, numbers of test vectors applied, number of outputs after compaction, testing was done with ATALANTA for deterministic and with FSIM for pseudorandom. The fault injected is same as done with the circuits without

compactors. For the sake of comparison, we tested the ISCAS 85 benchmark combinational circuits with both detected and all faults.

Table 6.9 Fault Coverage for ISCAS 85 Benchmark Combinational Circuits Using Deterministic Testing and Tested With Detected Faults.

Circuit Name	NOBC	ATV	NFI	NOAC	CR	CST [Sec]	FC [%]
c17	2	10	22	1	1/2	0.017	100.00
c432	7	124	520	1	1/7	0.130	100.00
c499	32	169	750	1	1/32	0.175	100.00
c880	26	223	940	1	1/26	0.630	100.00
c1355	32	220	1566	1	1/32	0.408	100.00
c1908	25	313	1870	1	1/25	1.010	100.00
c2670	140	496	2630	3	3/140	5.902	100.00
c3540	22	270	3291	1	1/22	3.403	100.00
c5315	123	692	5291	1	1/123	12.189	100.00
c6288	32	65	7710	1	1/32	3.265	100.00
C7552	108	349	7419	1	1/108	24.800	100.00

Table 6.10 Fault Coverage for ISCAS 85 Benchmark Combinational Circuits Using Deterministic Testing and Tested With All Faults.

Circuit Name	NOBC	ATV	NFI	NOAC	CR	CST [Sec]	FC [%]
c17	2	10	22	1	1/2	0.017	100.00
c432	7	124	524	1	1/7	0.133	99.237
c499	32	171	758	1	1/32	0.181	98.945
c880	26	226	942	1	1/26	0.633	100.00
c1355	32	229	1574	1	1/32	0.455	99.942
c1908	25	316	1879	1	1/25	1.050	99.521
c2670	140	497	2749	3	3/140	5.942	95.741
c3540	22	275	3428	1	1/22	3.500	96.004
c5315	123	698	5350	1	1/123	12.267	98.897
c6288	32	75	7744	1	1/32	3.345	99.561
C7552	108	376	7550	1	1/108	25.503	98.252

In Table 6.11 pseudorandom testing result of the ISCAS 85 benchmark combinational circuits with compactor are shown. For pseudorandom testing, we used FSIM and injected the same faults as in the case of deterministic testing. In Table 6.12, results are shown for the ISCAS 85 Benchmark circuits using FSIM with space compactors tested with compacted test vectors. From the tables it is clear that the compaction technique is feasible and gives 100 percent fault coverage for both deterministic and pseudorandom testing.

Table 6.11 Simulation Results of the ISCAS 85 Benchmark Circuits Using Pseudorandom Testing with Space Compactors.

Circuit Name	NIL	NOBC	ATV	NFI	NOAC	FC[%]
c17	5	2	45	22	1	100.00
c432	36	7	2752	520	1	100.00
c499	41	32	10929363	750	1	100.00
c880	60	26	97055712	940	1	100.00
c1355	41	32	100000000	1566	1	94.994
c1908	33	25	96283712	1870	1	100.00
c2670	233	140	100000000	2630	3	98.869
c3540	50	22	301824	3291	1	100.00
c5315	178	123	1316134912	5291	1	100.00
c6288	32	32	224	7710	1	100.00
c7552	207	108	1938989617	7419	1	100.00

Table 6.12 Simulation Results of the ISCAS 85 Benchmark Circuits Using FSIM with Space Compactors Tested with Compacted Test Vectors.

Circuit Name	NIL	NOBC	ATV	NFI	NOAC	FC[%]
c17	5	2	7	22	1	100.00
c432	36	7	80	520	1	100.00
c499	41	32	100	750	1	100.00
c880	60	26	150	940	1	100.00
c1355	41	32	124	1566	1	100.00
c1908	33	25	199	1870	1	100.00
c2670	233	140	366	2630	3	100.00
c3540	50	22	263	3291	1	100.00
c5315	178	123	686	5291	1	100.00
c6288	32	32	63	7710	1	100.00
c7552	207	108	349	7419	1	100.00

Figures 6.2, 6.3, and 6.4 show, respectively, the number of applied test patterns, compaction ratio, and the area overhead of compaction networks for all ISCAS 85 benchmark combinational circuits with our multiple line merger zero-aliasing compactors, using deterministic testing.

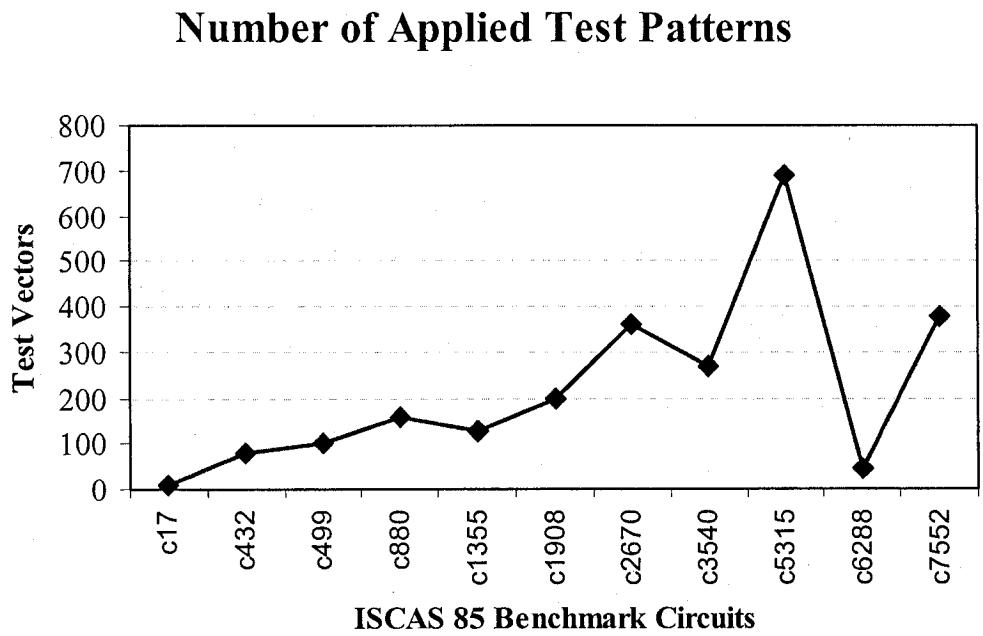


Figure 6.2 Input Test Patterns for ISCAS 85 Benchmark Circuits with Compactor Using Deterministic Testing.

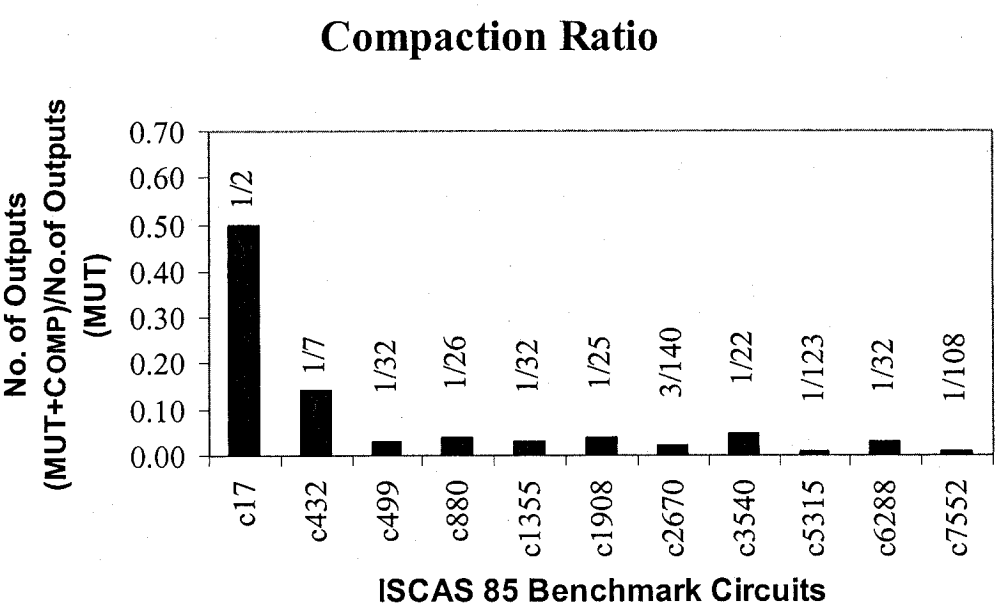


Figure 6.3 Compaction Ratio for ISCAS 85 Benchmark Circuits Using Deterministic Testing

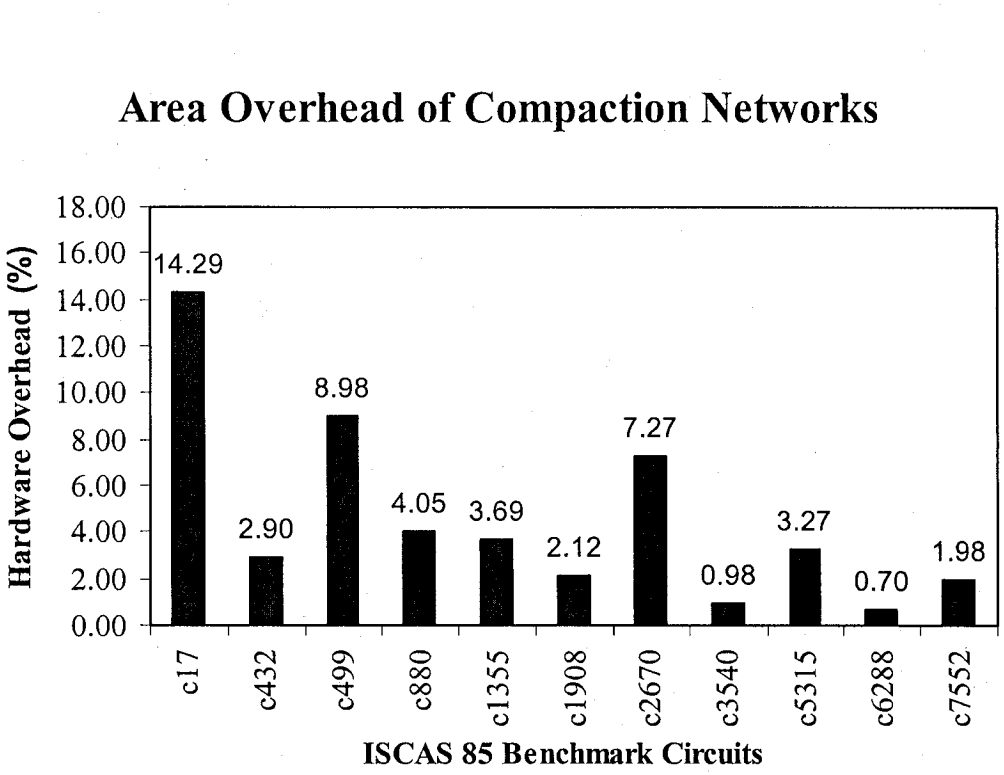


Figure 6.4 Area Overhead of Compaction Networks for ISCAS 85 Benchmark Circuits Using Deterministic Testing.

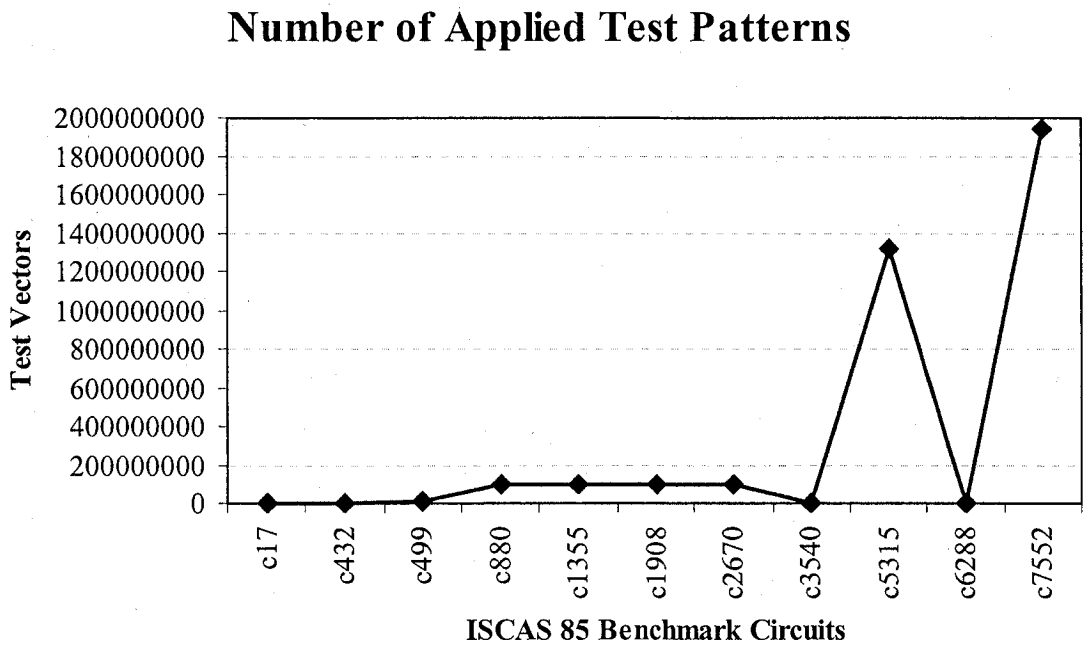


Figure 6.5 Input Test Patterns for ISCAS 85 Benchmark Circuits with Compactor Using Pseudorandom Testing.

6.2 Experimental Results with ISCAS 89 Benchmark Scan Circuit.

In order to further demonstrate the feasibility of the proposed space compactor simulation were also performed on ISCAS 89 benchmark sequential scan circuits. We have used ATALANTA to generate fault free output sequences, which is needed to construct the space compactors. ATALANTA is also used to test the benchmark circuits using reduced test sets accompanied with random testing session. FSIM fault simulation program is used to generate pseudorandom test sets. Tables 6.13 and 6.14 shows the deterministic and pseudorandom test results with detected fault injected to the ISCAS 89 benchmark sequential scan circuits respectively.

Table 6.13 Simulation results of the ISCAS 89 Benchmark Scan Circuits Using Deterministic Testing without Space Compactors.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
S27	8	32	4	100.00
S208	32	214	10	100.00
S298	55	306	20	100.00
S344	60	340	26	100.00
S349	45	348	26	100.00
S382	51	397	27	100.00
S713	98	921	42	100.00
S838	104	187	34	100.00
S953	96	81	52	100.00
S1196	197	1025	32	100.00
S1238	227	1035	32	100.00

Table 6.14 Simulation results of the ISCAS 89 Benchmark Scan Circuits Using Pseudorandom Testing without Space Compactors.

Circuit Name	Applied Test Vectors	No. of Fault Injected	No. of Output	Fault Coverage (%)
S27	96	32	4	100.00
S208	4256	214	10	100.00
S298	480	306	20	100.00
S344	128	340	26	100.00
S349	256	348	26	100.00
S382	448	397	27	100.00
S713	5615241	921	42	100.00
S838	1007311	187	34	100.00
S953	366	81	52	100.00
S1196	1003228	1025	32	100.00
S1238	1123213	1035	32	100.00

Table 6.15 Simulation Results of ISCAS 89 Benchmark Scan 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 (%)
S27	28	32	4	100.00
S208	29	214	10	100.00
S298	35	306	20	100.00
S344	22	340	26	100.00
S349	23	348	26	100.00
S382	32	397	27	100.00
S713	86	921	42	100.00
S838	97	187	34	100.00
S953	88	81	52	100.00
S1196	123	1025	32	100.00
S1238	205	1035	32	100.00

In Table 6.15, simulation results of ISCAS 89 benchmark scan circuits without compactor are shown, tested with all the detected faults. The test is done with FSIM and compacted test sets.

Figure 6.6 shows the number of applied test patterns used for getting the maximum fault coverage with the circuit without compactor and the detected faults were injected and with compacted test set, tested with FSIM.

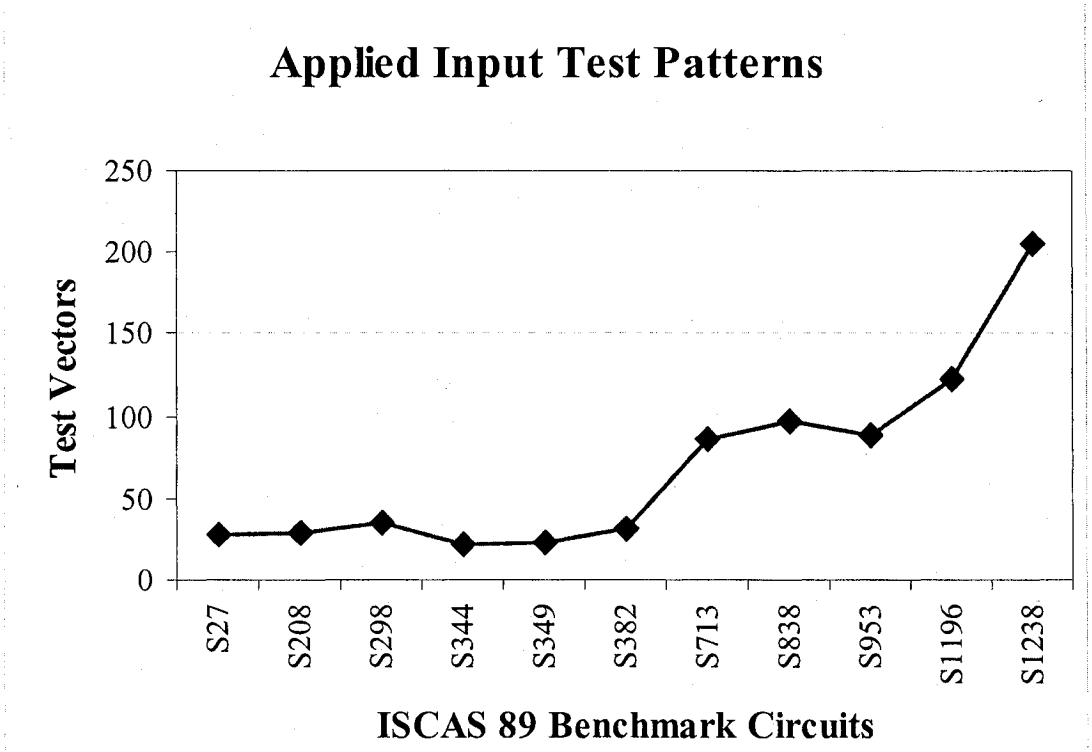


Figure 6.6 Input Test Patterns For ISCAS 89 Benchmark Scan Circuits Using FSIM Without Space Compactor and Detected Faults Injected and Tested With Compacted Test Sets.

In the following tables and figures we show the fault coverage, compaction ratio, the CPU simulation time, numbers of test vectors applied, number of outputs after compaction, testing was done with ATALANTA for deterministic and with FSIM for

pseudorandom. The fault injected is same as done with the circuits without compactors. For the sake of comparison, we tested the ISCAS 89 benchmark scan circuits with detected faults.

Table 6.16 Fault Coverage for ISCAS 89 Benchmark Sequential Scan Circuits With Compactor Using Deterministic Testing

Circuit Name	NOBC	ATV	NFI	NOAC	CR	CST [Sec]	FC [%]
S27	4	8	32	1	1/4	.0010	100.00
S208	10	38	214	1	1/10	0.033	100.00
S298	20	68	306	1	1/20	0.033	100.00
S344	26	61	340	1	1/26	0.035	100.00
S349	26	30	348	1	1/26	0.034	100.00
S382	27	40	397	1	1/27	0.037	100.00
S713	42	102	921	3	3/42	0.523	100.00
S838	34	110	187	1	1/34	0.432	100.00
S953	52	103	81	3	3/52	0.233	100.00
S1196	32	212	1025	1	1/32	1.052	100.00
S1238	32	234	1035	1	1/32	1.261	100.00

Table 6.17 Fault Coverage for ISCAS 89 Benchmark Sequential Scan Circuits With Compactor Using Pseudorandom Testing.

Circuit Name	NIL	NOBC	ATV	NFI	CST [Sec]	NOAC	FC[%]
S27	7	4	64	32	0.011	1	100.00
S208	19	10	4672	214	0.033	1	100.00
S298	17	20	88160	306	0.167	1	100.00
S344	24	26	1107232	340	2.283	1	100.00
S349	24	26	416	348	0.033	1	100.00
S382	24	27	1632	397	0.050	1	100.00
S713	54	42	7106532	921	4.067	3	100.00
S838	67	34	1196481	187	3.967	1	100.00
S953	45	52	458	81	0.087	3	100.00
S1196	32	32	1657645	1025	4.324	1	100.00
S1238	32	32	1495427	1035	6.856	1	100.00

Table 6.18 Fault Coverage for ISCAS 89 Benchmark Sequential Scan Circuits With Compactor Using FSIM and Compacted Test Sets.

Circuit Name	NIL	NOBC	ATV	NFI	NOAC	FC[%]
S27	7	4	8	32	1	100.00
S208	19	10	42	214	1	100.00
S298	17	20	65	306	1	100.00
S344	24	26	60	340	1	100.00
S349	24	26	27	348	1	100.00
S382	24	27	40	397	1	100.00
S713	54	42	101	921	3	100.00
S838	67	34	111	187	1	100.00
S953	45	52	103	81	3	100.00
S1196	32	32	208	1025	1	100.00
S1238	32	32	233	1035	1	100.00

Figures 6.7 - 6.10 show respectively, the number of applied test patterns with deterministic testing, the number of applied test patterns with pseudorandom testing, CPU simulation time with deterministic testing and compaction ratio respectively for all ISCAS 89 benchmark sequential scan circuits with our multiple line merger zero-aliasing compactors.

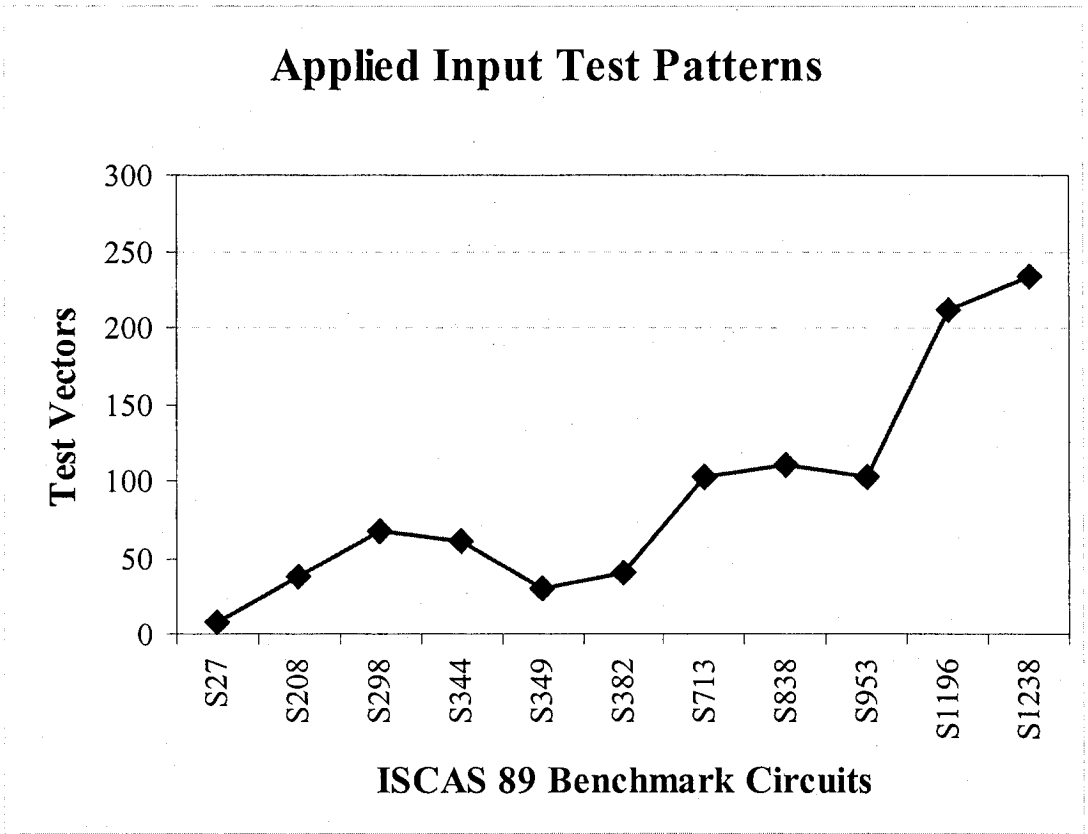


Figure 6.7 Input Test Patterns For ISCAS 89 Benchmark Scan Circuits Using Deterministic Testing.

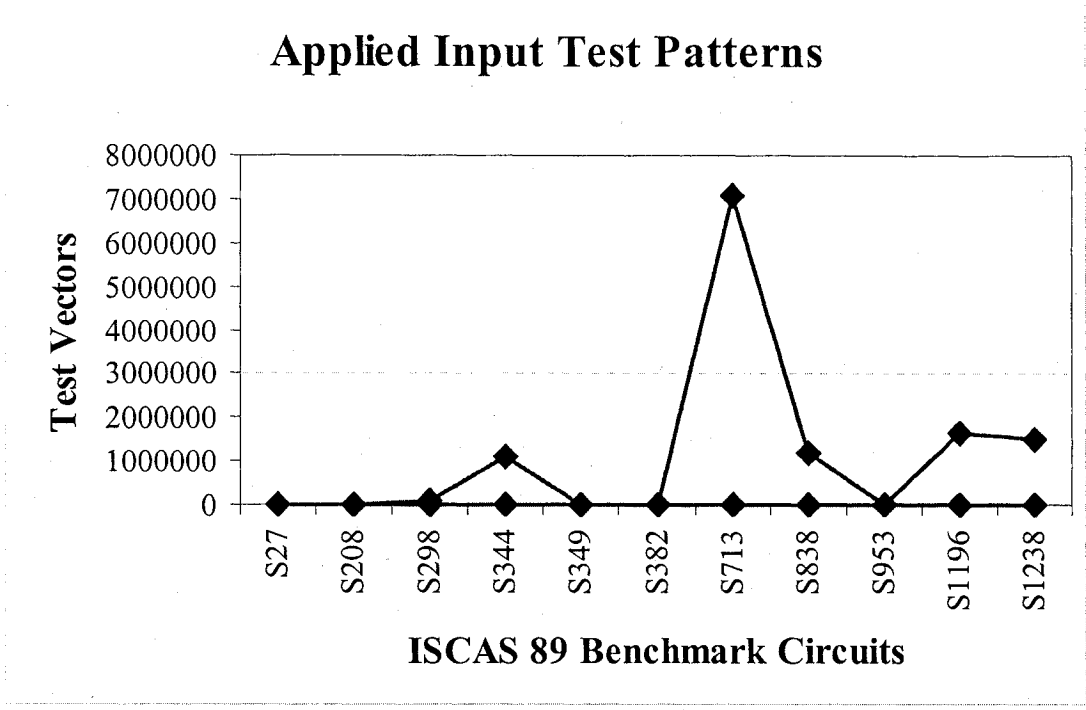


Figure 6.8 Input Test Patterns For ISCAS 89 Benchmark Scan Circuits Using Pseudorandom Testing.

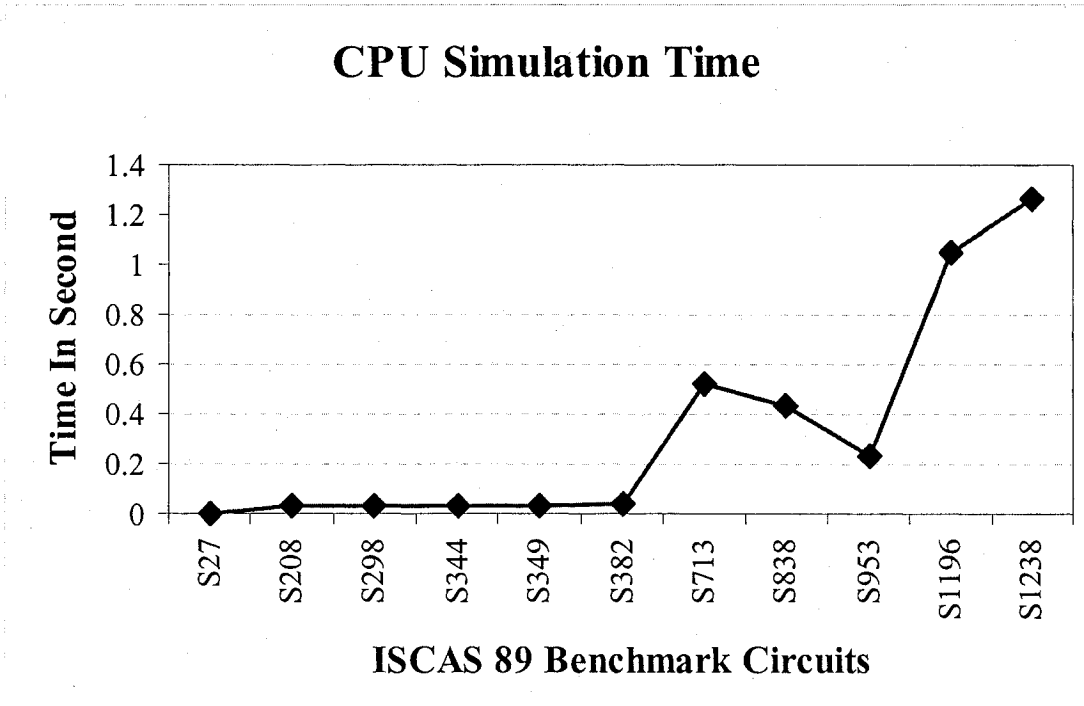


Figure 6.9 CPU Simulation For ISCAS 89 Benchmark Scan Circuits Using Deterministic Testing.

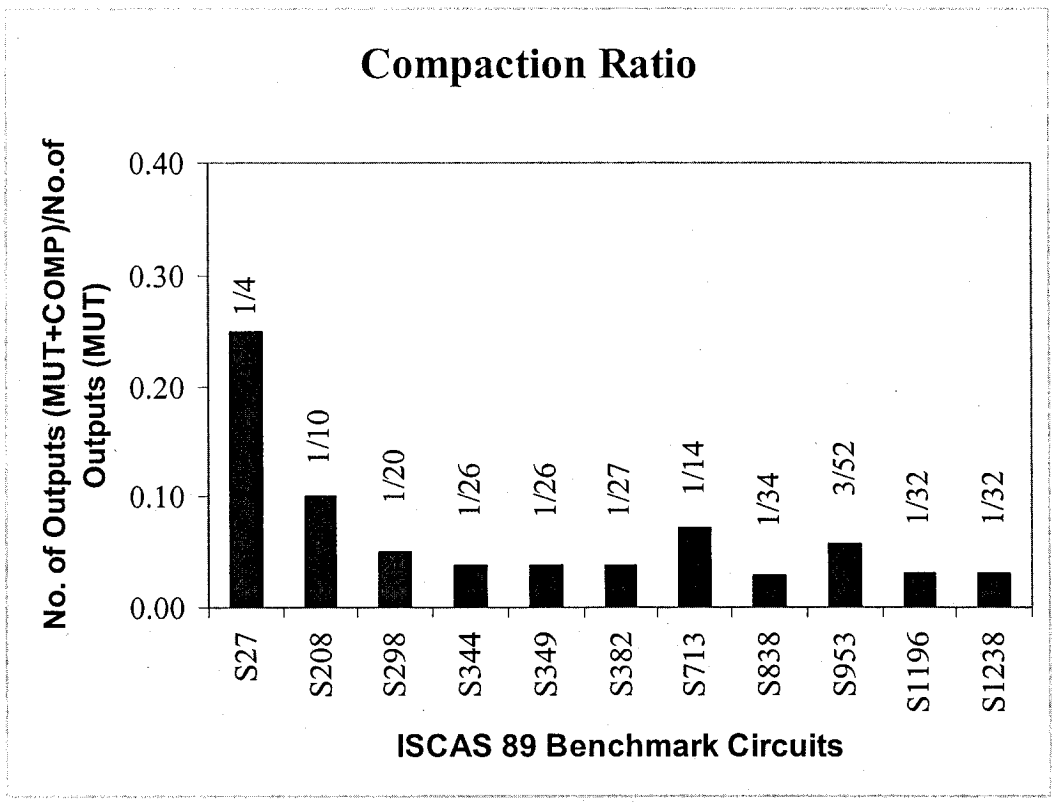


Figure 6.10 Compaction Ratio for ISCAS 89 Benchmark Scan Circuits Using Deterministic Testing.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

This thesis reported on developing a new zero-aliasing space compression approach of response data outputs specifically targeted for application to digital embedded SOCs. The suggested technique takes advantage of some well-known concepts of conventional switching theory and graph theory together with those of strong and weak compatibilities of response data outputs, in the selection of specific gates for merger of an arbitrary but optimal number of output bit streams from the MUT. 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 logic functions selected to build the compaction tree are determined solely by the characteristics of the sequences, which are inputs to the gates. The optimal mergeability criteria were obtained on the assumption of single line error, we find that error occurrence probability play a distinct role in the selection of gates for merger in most cases. The output bit stream selection is based on calculating the detectable error probability estimates or missed error probability estimates using an empirical formula developed initially by Li and Robinson.

The effectiveness of the proposed approaches is critically depend on the probability of error occurrence in different lines of the MUT, and this dependence may be effected by the circuit structure, that is, by the number of inputs, outputs, internal lines, and the type of gates the circuits are designed of. In actual situation if the circuit structure changes, then the probability value changes and obviously the corresponding compression networks that have to be designed based on optimal megeability criteria changes as well. As the objective was to devise efficient methods of synthesizing compaction networks, which provide improve fault coverage and complexity than conventional techniques, so the complexity issues were no addressed in depth in the current study. Here the major issue in involves the computation of the detectable or missed error probability estimate,

which is simple in the case of two-line mergers, compared to that in the case of generalized mergeability [5] and optimal generalized mergeability [49].

A heuristic approach is adopted for generalized mergeability, to get a result within an acceptable CPU time. The heuristic approach is very useful in this case and has been adopted and implemented in the generalized mergeability case as well, for computational reasons.

The approaches developed subsequently to designing zero-aliasing space compactors are based on sequence characterization, switching theory concepts, and also utilizing concepts of strong and weak compatibilities of response data outputs. It is an effective technique as zero-aliasing is achieved without modification of the MUT, and also the maximal compaction is achieved in most cases in reasonable time.

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.

Finally, testing can be combined with efficient input test patterns to synthesize BIST circuits that provide more than 99% and in most cases 100% fault coverage with minimum CPU simulation time and low area overhead. This is unique in the sense that zero-aliasing is achieved without any modification of the MUT, while maximal compaction is achieved in most cases in reasonable time utilizing some simple heuristics. From the experimental results, it is obvious that the suggested approach is simple and robust enough in its design methodology based on consideration of single stuck-line faults of the MUT. With increase in computational resources, in the future, this somewhat heuristic space compaction algorithm might be improved for better efficiency in respect of time and storage.

7.2 Future 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 with 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.

For hardware verification, it would be better to implement the ISCAS benchmark circuits in FPGA. Implementing the circuits under test and the compactors in FPGA to verify the feasibility and performance of the compactors may be a good idea.

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.