Université d'Ottawa · University of Ottawa

PERMISSION DE REPRODUIRE ET DE DISTRIBUER LA THÈSE

PERMISSION TO REPRODUCE AND DISTRIBUTE THE THESIS

NOM DE L'AUTEUR / NAME OF AUTHOR:	Mansour Assaf
ADRESSE POSTALE / MAILING ADDRESS:	171 du Voilier Gatineau, Quebec J8P 7Z2
GRADE / DEGREE:	**ANNÉE D'OBTENTION / YEAR GRANTED**
Ph. D - Electrical Engineering	2003
TITRE DE LA THÈSE / TITLE OF THESIS: Digital Core Output Test Data Compression Architecture Based on Switching Theory Concepts	

L'auteur permet, par la présente, la consultation et le prêt de cette thèse en conformité avec les règlements établis par le bibliothécaire en chef de l'Université d'Ottawa. L'auteur autorise aussi l'Université d'Ottawa, ses successeurs et cessionnaires, à reproduire cet exemplaire par photograhie ou photocopie pour fins de prêt ou de vente au prix coûtant aux bibliothèques ou aux chercheurs qui en feront la demande.

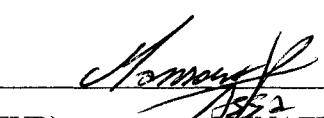
Les droits de publication par tout autre moyen et pour vente au public demeureront la propriété de l'auteur de la thèse sous réserve des règlements de l'Université d'Ottawa en matière de publication de thèses.

The author hereby permits the consultation and the lending of this thesis pursuant to the regulations established by the Chief Librarian of the University of Ottawa. The author also authorizes the University of Ottawa, its successors and assignees, to make reproductions of this copy by photographic means or by photocopying and to lend or sell such reproductions at cost to libraries and to scholars requesting them.

The right to publish the thesis by other means and to sell it to the public is reserved to the author, subject to the regulations of the University of Ottawa governing the publication of theses.

N.B. LE MASCULIN COMPREND ÉGALEMENT LE FÉMININ

June 20, 2003
DATE


(AUTEUR) SIGNATURE (AUTHOR)

Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

ASSAF, Mansour

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Ph. D. (Electrical Engineering)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Digital Core Output Test Data Compression Architecture Based on Switching Theory Concepts

S. Das

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

E. Petriu

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

A. Adler

V. Groza

T. Kwasniewski

S. Mourad

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

DIGITAL CORE OUTPUT TEST DATA COMPRESSION ARCHITECTURE BASED ON SWITCHING THEORY CONCEPTS

Model Implementation and Analysis

By

Mansour Hanna Assaf, M.A.Sc.

A dissertation submitted to the Faculty of Graduate and Postdoctoral Studies of the University of Ottawa in partial fulfillment of the requirements for the Degree of Doctor of Philosophy (Ph.D.)

School of Information Technology and Engineering
Department of Electrical and Computer Engineering
University of Ottawa

July 2003

ACKNOWLEDGMENTS

The present dissertation embodies the results of investigations carried out by the author in the general area of built-in self-testing (BIST) of VLSI circuits, and specifically in the field of test data compression in space, during the period 1997-2002 at the School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada. The research works reported in the dissertation on the performance characteristics and complexities of the various algorithms for the generation of space compaction trees of core-based digital integrated circuits are particularly significant because of the novelty of the approaches and of the very intriguing nature of the problem.

The author takes this opportunity to express his heartfelt gratitude to his supervisor, **Dr. Sunil R. Das**, Professor Emeritus at the School of Information Technology and Engineering, University of Ottawa, for his patient guidance, numerous help, and continued encouragement during the progress of the work. The research training he gave him will be an invaluable and unforgettable experience throughout his whole life. The quality of the dissertation also improved remarkably as a result of meticulous reading and many constructive suggestions from Dr. Das.

The author is particularly thankful to his esteemed co-supervisor, **Dr. Emil M. Petriu**, former Director of the School of Information Technology and Engineering, University of Ottawa, for his constant encouragement and immense support throughout the progress of this research. He gave the present author absolute freedom in choosing his direction of research and working on his dissertation, that rendered the whole business not only extremely interesting, but at the same time truly enjoyable.

The author's special thanks are due to Professor **C. V. Ramamoorthy** of the University of California, Computer Science Division, Berkeley, CA, USA, for his many helpful suggestions during the preparation of the dissertation. His qualities as an esteemed scientist and researcher should be a glaring example to everybody.

The author also takes the present opportunity of recording his indebtedness to **Dr. Krishnendu Chakrabarty** of the Duke University, Durham, NC, USA, and to **Dr. Wen-Ben Jone** of the University of Cincinnati, Cincinnati, OH, USA, for their helpful comments and cooperation in every phase of the work.

Thanks are also due to **Dr. Dong S. Ha** of the Department of Electrical Engineering at the Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, and **Professor Irith Pomeranz** of the University of Iowa, Iowa City, IA, USA, for kindly providing the author with their test pattern generation and fault simulation tools.

The author owes special thanks to several research colleagues at the University of Ottawa: **Jing Yi Liang, Tony Barakat**, and **Made Sudarma** for many stimulating discussions and research collaboration, and **Rami Abielmona**, and **Payam Abolghasem** for their help in the JBits hardware realization.

He also takes this opportunity of expressing his deep appreciation to his mother, **Mrs. Laurice N. Kanaan**, for her constant encouragement and inspiration during the whole period of his studies. His grateful thanks are specially due to his family members for their support and help during the past several years he was occupied with his academic pursuits.

The financial support from the Natural Sciences and Engineering Research Council of Canada under **Grant A 4750** is also gratefully acknowledged.

July 2003

M. H. Assaf
School of Information Technology and Engineering
University of Ottawa
800 King Edward Avenue
Ottawa, Ontario K1N 6N5, Canada

ABSTRACT

The design of space-efficient support hardware for built-in self-testing (BIST) is of critical importance in the design and manufacture of VLSI circuits. This dissertation reports new space compression techniques for particular use in digital core based systems which facilitate designing compression networks using compact or pseudorandom test sets, with the target objective of minimizing the storage requirements for the module under test (MUT), while maintaining the fault coverage information. The suggested techniques take advantage of some well known concepts of conventional switching theory, particularly those of cover table and frequency ordering as commonly utilized in the minimization of switching functions, besides knowledge of Hamming distance, sequence weights, and derived sequences in the selection of specific gates for merger of an arbitrary number of output bit streams from the MUT. The outputs coming out of the space compactor may eventually be fed into a time compressor (*viz.* syndrome counter) to derive the MUT signatures. The approaches developed to designing zero-aliasing space compressors utilizing additionally concepts of strong and weak compatibilities of response data outputs are novel in the sense that zero-aliasing is achieved without modification of the MUT, while maximal compaction is achieved in most cases in reasonable time utilizing some simple heuristics.

The techniques proposed in the dissertation guarantee simple design with a high or full fault coverage for single stuck-line faults, with low CPU simulation time, and acceptable area overhead. Design algorithms are proposed in the dissertation, and the simplicity and ease of their implementations are demonstrated with numerous examples. Specifically, extensive simulation runs on ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits with FSIM, ATALANTA, HOPE, and COMPACTEST programs confirm the usefulness of the suggested approaches under conditions of both stochastic independence and dependence of single and multiple line errors.

A performance comparison of the designed space compressors with conventional linear parity tree space compactors as benchmark is also presented in the dissertation, where zero-aliasing is not realized, which demonstrates improved tradeoff for the new circuits between fault coverage and the MUT resources consumed contrasted with existing designs, thereby aiding to fully appreciate the enhancements. However, for the zero-aliasing compactors, advantages are clearly obvious.

TABLE OF CONTENTS

TABLE OF ABBREVIATIONS

SoC:	System-on-a-Chip
SLI:	System Level Integration
MCM:	Multi-Chip Modules
IC:	Integrated Circuit
IP:	Intellectual Property
BIST:	Built-In Self-Test
TPG:	Test Pattern Generator
MUT:	Module Under Test
BIT:	Built-in Test
ST:	Self Test
DFT:	Design for Testability
VSIA:	Virtual Socket Interface Alliance Bus
AMBA:	Advanced Micro-Controller Bus Architecture
TAM:	Test Access Mechanism
ATPG:	Automatic Test Pattern Generator
HSC:	Hybrid Space Compression
DSC:	Dynamic Space Compression
QFC:	Quadratic Functions Compaction
PSC:	Programmable Space Compaction
MPT:	Multiplexed Parity Trees
DTC:	Double Transition Counting
MTC:	Multiple Transition Counting
CRC:	Cyclic Redundancy Check
LFSR:	Linear Feedback Shift Register
MISR:	Multiple Input Signature Register
CUT:	Circuit Under Test
s-a-1:	Stuck-at-1 Faults
s-a-0:	Stuck-at-0 Faults
MCSET	Modified Cut-Set Algorithm
COMP:	Compactor
JBits:	Development Framework for Xilinx FPGAs Based on the Java Language
API:	Application Programming Interface
XHWIF:	Xilinx Hardware Interface
Virtex DS:	Virtex Device Simulator
NIF:	Number of Injected Faults
PIC:	Number of Input Lines in MUT
POC:	Number of Output Lines in MUT
NGC:	Number of Gates in MUT
NGP:	Number of Gates in Compactor
NGCP:	Number of Gates in MUT + Compactor
TV:	Number of Applied Input Test Vectors
POCP:	Number of Output Lines in MUT+COMPACTOR
NL:	Number of Levels in COMPACTOR
AO:	Area Overhead (%)
AVFC:	Average Fanins in MUT

AVFP:	Average Fanins in Compactor
NFP:	Number of Fanins in Compactor
CR:	Compaction Ratio
CPU:	CPU Simulation Time (secs)
FC:	Fault Coverage (%) (MUT+COMPACTOR)
NAIP:	Number of AND Incompatible Pairs Used in the Computation of the AND Compatibility Classes
NOIP:	Number of OR Incompatible Pairs Used in the Computation of the OR Compatibility Classes
NXIP:	Number of XOR Incompatible Pairs Used in the Computation of the XOR Compatibility Classes
NMCA:	Number of Maximal AND Compatibility Classes
NMCO:	Number of Maximal OR Compatibility Classes
NMCX:	Number of Maximal XOR Compatibility Classes

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

System-on-a-Chip (SoC) is revolutionizing the electronics industry. These super chips will power the next generation of cell phones, multimedia devices and PC graphics chipsets. Accounting for about 60% of today's semiconductor market, SoC is predicted to rise to 90% within the next five years. SOC is known by several names: System Silicon, System Level Integration (SLI) and Multi-Chip Modules (MCM). Regardless of how it is referenced, it has the same meaning: the growing integration of digital, analog and memory circuits on a single piece of silicon [6].

System-on-a-Chip will, no doubt, make everyone's life easier, with the exception of the average IC manufacturer who will now have to produce their customer's whole system on one piece of silicon. Rather than discrete devices occupying a circuit board, subsystems are now virtual components on a single chip. Due to this amount of integration, last minute adjustments to the performance of a product may be difficult for the customer and the system manufacturer. Additional turns of silicon would result in millions of dollars in lost opportunities, possibly even missing a product cycle altogether. By incorporating SoC designs, manufacturers can minimize the size of devices and products, use fewer components, consume less power, and maximize reliability, which will increase value for device and product manufacturers and consumers.

The microprocessor is the fundamental element in SoC. Depending on the specific application, various blocks of embedded random logic, memory and mixed-signal circuitry are added [21]. In designing SOC, predefined Intellectual Property (IP) blocks are usually mixed together from different sources and proprietary IP cores are added to create a specific application. By utilizing existing IP cores, the need for designing an entire chip from scratch is eliminated, which accelerates time-to-market by reducing design cycle time from a year or more to several months, or even weeks. Unfortunately, the mixing of major functional blocks from different sources can make integration not so easy a job.

As ICs expand geometrically with shrinking interconnect line widths and gate lengths, the challenge begins with SoC's complexity. With SoCs being applied to volume markets, the question of test cost becomes an issue with manufacturers as well as its test accuracy. The test economies of the new multifunction SoC test system that can test the digital, analog and memory content of an SoC device in a single insertion must be considered as well.

Data rates, accuracy, integrated memory, analog test capabilities, and number of single pins available are all included in the evaluation criteria for SoC testing. The specifications can vary greatly, but as SoC device capability expands, device manufacturers will be looking for increased digital pins, faster digital speeds, more integrated analog, low voltage differential, high speed serial, dense embedded DRAM, vastly deep and wide scan as well as the ability to test more devices in parallel, with quad site becoming the production standard [6].

The majority of today's SoC devices are still under 600 pins; however, but as the device gate count and complexity evolve, so will be the number of pins. The future SoCs will contain well over one million gates and integrate embedded memory and the highest performance mixed signal digital and analog cores. To utilize optimal test coverage and full production, testing will move towards 1024-pin automatic test systems with fully integrated memory and mixed signal test capabilities.

Also increasing are the requirements around frequency and accuracy. It is necessary for SoC test systems to provide very high, sustained data rates to support the requirements of today's standard buses which run at speeds of over 100 MHz and specialty interfaces that are exceeding 500 Mbps with edge placement accuracy of less than 100 picoseconds.

To facilitate audio, video, graphics and communications applications within SoC, mixed-signal IP cores are integrated throughout. With SoC providing the links between the electronic and physical worlds, mixed-signal test capability is a critical element for an SoC test system.
Instruments can be implemented to a special purpose test system to provide some measurements; however, in order to save set-up time as well as measurement time during testing, integral instrumentation within the SoC system architecture is preferred. This ultimately creates a better system with better test economies.

With the complexity of SoC increasing, manufacturers have chosen to use built-in self-test techniques (BIST) where individual modules on the chips themselves have their own test circuits enabling access to circuits embedded within the device and minimizing the demand on the test system. BIST techniques still rely on the system to initiate input test patterns and measure the pass-fail status of the device.

Conventional testing techniques, as considered from a digital design perspective, require application of test patterns enabled by a test pattern generator (TPG) to the module under test (MUT) and comparison of the responses with known correct responses. With a larger digital system, however, because of increased storage demands for fault-free responses, the test procedure becomes quite costly, and alternative measures are needed to reduce the amount of required storage [3], [7].

BIST is a design methodology that provides the capability of solving a large number of difficulties otherwise encountered in testing digital systems. It combines two concepts: that of built-in test (BIT) and of self-test (ST), resulting in what is now known as built-in self-test (BIST). By utilizing built-in hardware [3], [25], [69], [56], [59], [7], [62], BIST can accomplish test generation, test application, and response verification, which helps to reduce required testing time by allowing different areas of a chip to be tested in parallel. This eliminates the need for external test equipment. With the cost efficiency of testing becoming a major factor of the manufacturer's expense of new products, the use of BIST thus tends to reduce manufacturing and maintenance expenses through superior diagnosis.

Several companies such as Motorola, AT & T, IBM, and Intel have incorporated BIST in many of their products [29], [59]. AT & T, for example, has incorporated BIST into more than 200 of their IC chips. The three large PLAs and microcode ROM in the Intel 80386 microprocessor were built-in self-tested [59]. The general-purpose microprocessor chip, Alpha AXP21164, and Motorola microprocessor, 68020, were also tested using BIST techniques [29], [59].

BIST is widely used to test embedded regular structures that exhibit a high degree of periodicity such as memory arrays (SRAMs, ROMs, FIFOs, and registers). This type of circuits does not require complex extra hardware for test generation and response compaction. Also, including BIST in these circuits can guarantee high fault coverage with zero aliasing. Unlike regular circuits, random-logic circuits cannot be adequately tested only with BIST techniques, since

generating adequate on-chip test sets using simple hardware is a difficult task to be accomplished. Moreover, since test responses generated by random-logic circuits seldom exhibit regularity, it is extremely difficult to ensure aliasing-free compaction. Therefore, random-logic circuits are usually tested using a combination of BIST, scan design techniques, and external test equipment.

A typical BIST environment, as shown in Figure 1.1, uses a test pattern generator (TPG) that sends its outputs (test vectors) to a module under test (MUT), and output streams from the MUT are fed into a test data analyzer. A fault is detected if the test sequence is different from the response of the fault-free module. The test data analyzer is comprised of a response compaction unit, a storage for the fault-free responses of the MUT, and a comparator.

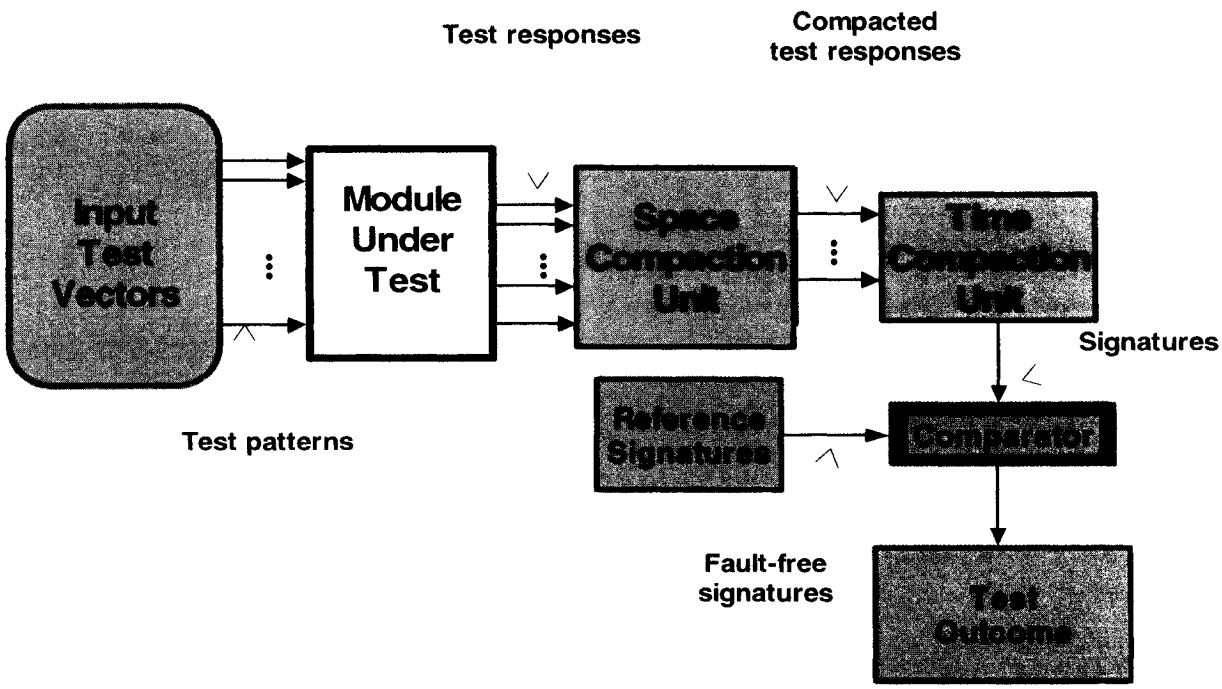


Figure 1.1 BIST Environment.

In order to reduce the amount of data represented by the fault-free and faulty MUT responses, data compression is used to create signatures (short binary sequences) from the MUT and its corresponding fault-free module (circuit). Signatures are compared and faults are detected if a

match does not occur. BIST techniques may be used during normal functional operating conditions of the unit under test (on-line testing), as well as when a system is not carrying out its normal functions (off-line testing). In the case where detecting real-time errors is not that important, systems, boards, and chips can be tested in off-line BIST mode. BIST techniques use pseudorandom, or pseudoexhaustive test patterns, or sometimes on-chip storage of reduced test sets. Today, testing logic cores exhaustively is seldom used, since only a few test patterns are needed to ensure full fault coverage for single stuck-line faults [29]. Reduced pattern test sets can be generated using existing algorithms such as FAN, and others. Built-in test generators can often generate such reduced test sets at low cost, making BIST techniques suitable for on-chip self-testing.

The subject research aims at the response compaction process of built-in self-testing techniques that translates into a process of reducing the test response from the MUT to a signature. Instead of comparing bit-by-bit the fault-free responses to the observed outputs of the MUT as in conventional testing methods, the observed signature is compared to the correct one, thereby reducing the storage needed for the correct circuit responses [3], [7], [59], [62]. The response compaction in BIST is carried out through a space compaction unit followed by time compaction. In general, s input sequences coming from a MUT are fed into a space compactor, providing t output streams of bits such that $t << s$. Most often, test responses are compressed into only one sequence ($t = 1$). Space compaction brings a solution to the problem of achieving high-quality built-in self-testing of complex chips without the necessity of monitoring a large number of internal test points, thereby reducing testing time and area overhead by merging test sequences coming from these internal test points into a single stream of bits. This single bit stream of length r is eventually fed into a time compactor, and finally a shorter sequence of length q ($q < r$) is obtained at the output. The extra logic representing the compaction circuit, however, must be as simple as possible, to be easily embedded within the MUT, and should not introduce signal delays to affect either the test execution time or normal functionality of the circuit being tested. Moreover, the length of the signature must be as short as it can be in order to minimize the amount of memory needed to store the fault-free response signatures. Also, signatures derived from faulty output responses and their corresponding fault-free signatures should not be the same, which unfortunately is not always the case.

A fundamental problem with compaction techniques is error masking or aliasing [3], [7], [9], [59], [62], which occurs when the signatures from faulty output responses map into the fault-free

signatures, usually calculated by identifying a good circuit, applying test patterns to it, and then having the compaction unit generate the fault-free references. Aliasing causes loss of information, which affects the test quality of BIST and reduces the fault coverage (the number of faults detected, after compaction, over the total number of faults injected). Several methods have been suggested in the literature for computing the aliasing probability. The exact computation of this aliasing probability is known to be an NP-hard problem [63]. In practice, a rather high fault coverage, over 99%, is required and hence, any space compression technique that maintains more percentage error coverage information is considered worthy of investigation.

The subject research deals with the general problem of designing space-efficient support hardware for built-in self-testing (BIST) of VLSI circuits using knowledge of compact test sets. The techniques developed for the purpose are primarily based on identifying certain inherent properties of the test responses of the MUT, together with the knowledge of their failure probabilities. A major objective to realize in space compaction is to provide techniques that are simple, suitable for on-chip self-testing, require low area overhead, and have little adverse impact on the MUT performance.

1.1 Scope of the Present Work and Thesis Outline

The objective of this research is to develop new approaches to designing non-zero-aliasing and and zero-aliasing space compactors using both deterministic or pseudorandom test patterns.
Based on the review of the related works and our initial investigation, it was concluded that conventional switching theory concepts such as those of Hamming distance, sequence weights, and derived sequences together with those of cover table, and frequency ordering, commonly utilized in minimization of switching functions, present advantages over other methods to fulfill this task. There remain, however, several challenges to complete a zero-aliasing space compactor model:

- Achieve maximal output lines compaction without any modifications of the MUT;
- Establish a mathematically sound selection criterion of merger of a number of lines of the MUT;
- Reduce the area overhead and signal propagation delay;
- Analyze the results of simulations on different ISCAS 85 combinational and ISCAS 89 full scan benchmark circuits.

The material in this dissertation is organized as follows:

Chapter 1 provides an overview of SoC, IP core based embedded systems, digital IP core testing methods, and a brief introduction of what is going to be discussed in the present dissertation.

Chapter 2 discusses SoC test, built-in self-test for digital IP core test, space and time compaction techniques.

Chapter 3 introduces certain important properties of circuit responses to develop pairwise mergeability criteria for the design of space compactor on the assumption of stochastic independence and stochastic dependence of single and double line errors.

Chapter 4 introduces more important properties of circuit responses to develop generalized mergeability criteria for the design of space compactor on the assumption of stochastic independence and stochastic dependence of single and double line errors.

Chapter 5 introduces more important properties of circuit responses to develop optimal generalized mergeability criteria for the design of space compactor on the assumption of stochastic independence and stochastic dependence of single and double line errors.

Chapter 6 introduces important properties of circuit responses to construct aliasing-free compactors with maximal compaction ratio.

Chapter 7 provides experimental results on extensive simulations of the ISCAS 85 combinational and ISCAS 89 full scan benchmark circuits.

Chapter 8 discusses the experimental results and future research.

CHAPTER 2

LITERATURE OVERVIEW

Unlike traditional test approaches, Systems-on-chips (SoCs) make it impossible to establish the demarcation lines between design and test. For mixed-signal devices and complex digital cores, design test engineers must use design tools that let them incorporate testability early in the design process [6]. Incorporating testability includes two options: design-for-test (DFT) techniques, which ensure that all circuit elements within a chip system can be stimulated and observed, built-in-self-test (BIST) techniques, which allow a chip to generate input test stimuli needed to test itself and measure the corresponding output test responses. Test's importance in SoC design has encouraged design automation vendors such as Mentor Graphics, Cadence, and Synopsys to integrate DFT tools into their simulation and synthesis packages. Testability tools for designs that exist in a standard design-description format such as Verilog, VHDL, EDIF, or Spice are also offered by other firms such Fluence Technology, Genesys Testware, InterHDL, LogicVision, and Syntest. A device's testability is evaluated using these tools enhanced Verilog, VHDL, or Spice descriptions that incorporate BIST functions are first produced, input test vectors are automatically generated, faults are simulated and so on.

A variety of test space compaction techniques for BIST have been proposed in the literature [1], [38], [41], and some of which are in actual use. We describe some of them, concentrating only on some of their relevant properties like the area overhead, fault coverage, error masking probability, etc. There also exist a number of efficient time compaction techniques including syndrome testing, transition counting, signature analysis, and others [34], [59]. Since our concern in this research is exclusively space compaction, we will refrain from discussing time compression techniques in details. Interested readers can get a good amount of information on the subject in [59], [62], [3].

2.1 System-on-Chip and Built-In Self-Test

The number of components that system engineers can incorporate into a single chip is growing rapidly [71]. This rapid growth of SoC design has led to the marketability of intellectual property (IP) cores, available in various forms ranging from soft cores to hard cores [72]. By integrating IP cores into a system, the core's usefulness can be enhanced.

In order to prevent failure upon power-up of core-based systems which can contain several types of logic, hundreds of memory elements (RAMs, ROMs, EPROMs etc.), dozens of functional blocks attained from as many sources, user-defined logics, and other analog circuitries, system engineers should take the core integration issue into consideration in the design stages and look for ways to integrate the various cores into a unified testability scheme [6]. A typical complex multimillion gate wired ASIC design block diagram with cores, embedded random logic, memories (RAMs, ROMs, etc.), user-defined logics, mixed-signal and communication interface circuitry is shown in Figure 2.1.

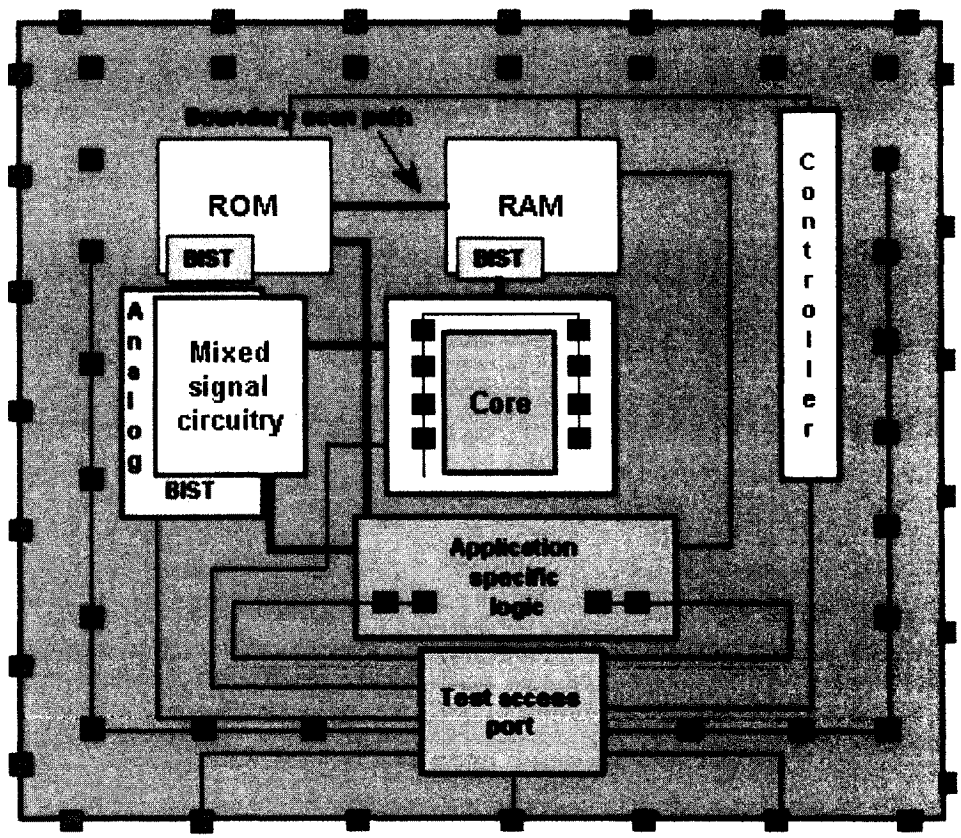


Figure 2.1 Typical ASIC Design Block Diagram with Cores.

Even if each core comes with its own testability bus and plan, there is no guarantee of overall testability. Knowing that some cores come with no testability complicates the problem further. A master plan for observation and control is needed to target this issue. Suppose that one core carries the Virtual Socket Interface Alliance (VSIA) bus, another the Advanced Microcontroller Bus Architecture (Amba), and suppose other cores have no provision for test other than a set of test vectors. Consequently, the key issue here is how to test cores together, synchronized, or isolated from one another, so that problems can be identified and located at the whole chip stage. One possible solution is to use externally driven diagnostics, which is feasible by planning in advance such that all cores have connections to the outside of the chip. Another possible solution is to let each core diagnose itself; therefore self-test and built-in self-test (BIST) is a good and even a much better choice. External and self-test diagnostics are both possible if planned in advance and they both can go a long way towards helping, if not completely solving, the SoC test challenge.

In general and for obvious reasons, IP core providers do not give away source codes to users. In fact what users buy is a black box which can be tampered with simulation and testing. On the other hand, a set of test vectors is provided by many IP providers when users buy their IP cores. The end users can use the test vectors to do the challenging task of core test.

A possible solution could be to use a standardized test bus on every core and chip; however, there is no such standard, and the fight for one unique standard continues with no sign of early resolution. Several standard bus contenders have emerged, including a standard bus from VSIA and another one from ARM's proprietary Amba. IEEE's Test Technical Committee is working on a standard for Embedded Core Test and a method of wrapping cores for the purpose of isolation, called P1500 SECT [5]. Core users have to deal with multiple buses or isolation methods, because IP providers and chip makers are still free to choose whichever bus they like. For instance, Xilinx, Inc. (San Jose, CA) has adopted an IBM test bus, while Altera Corp. (San Jose, CA) has adopted the Amba bus, as have a number of ARM's partners and others in the semiconductor industry. Amba bus could be established as an SoC de facto global standard, if the number of adopters keeps growing and reaches a critical mass. Amba also defines a test methodology that permits fast testing of memory cache RAM and modular testing. Large telecom companies such Motorola Inc. (Schaumburg, IL), IBM Corp. (Armonk, NY), and others have developed their own test

integration plans and bus schemes. Others act as mediator between the chip makers and the core vendors, trying to find a generic SoC test architecture, so that all cores on a chip will be synchronized. Core vendors may also provide testable IPs that can be hooked up to various buses by settling on full scan, boundary scan approaches or, logic BIST. Regardless of the methodology chosen, in order to locate faults in each core individually or on a core-by-core basis, cores must be isolated. The input test patterns provided by the core vendor can be integrated into an overall test plan once the cores are isolated. Logic BIST can also be integrated once core isolation is achieved.

Figure 2.2 shows generic conceptual test access architecture and a generally used nomenclature for embedded IP cores that was introduced by Zorian et al. [21]. The architecture consists of three elements. The first element of the architecture is the test source/sink. The test source is the place where real-time stimulus generation takes place, while the test sink is responsible for carrying out real-time response evaluation. Source and sink can either be implemented on chip (Built-In Self-Test), off-chip (Automatic Test Equipment), or in combination of both. The test access mechanism (TAM) is the second element of the architecture. It bridges the physical distance between source and core as well as between core and sink, and therefore serves as 'test data highway. Examples of TAM include TESTRAIL [73], Test Bus [74], and Addressable Test Ports [75]. The wrapper, a thin shell around the core that connects the TAMs to the core is the third architectural element. The wrapper provides the switching between normal functional access and both core-internal and core-external test access via the TAM. Furthermore, wrappers may provide width adaptation in case of a mismatch between core I/O width and TAM width, that is, by means of serial-parallel and parallel-serial conversion. Examples of wrappers include TestShell [73], and Test Collar [74]. One of the elements of the IEEE P1500 SECT [76], is a standardized, but scalable wrapper [77].

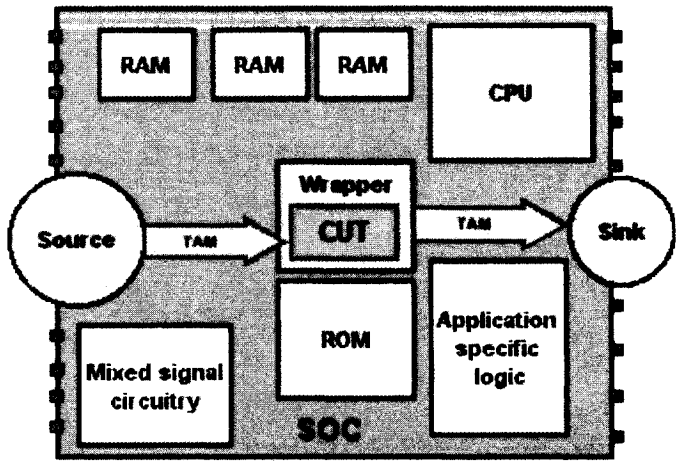


Figure 2.2 Generic Conceptual Test Architecture [1] Consisting of Three Elements: (1) Source/Sink, (2) Test Access Mechanism, and (3) Wrapper.

In Amba, peripheral cores are isolated using an on-chip system bus so that each core can be accessed separately. The processor core, on-chip memory, and other high-bandwidth peripheral cores are connected together to a pipelined advanced system bus. To minimize the potential performance penalties posed by test circuitry and to reduce loading on the system bus, SoC components are also connected to a separate advanced peripheral bus, which can also be used for lower-performance cores. Alternatively, subcircuits either in boundary-scan format (multiplexing approach) or another isolation technique can be used on the boundary of each core.

Logic BIST coupled with boundary scan can be an attractive solution that can take care of most of the digital cores testing challenge [6]. BIST has also been applied to certain analog functions, such as phase-locked loops and even data converters. One advantage of using logic BIST is its ability to test a circuit at its full operating speed, which can reach today even gigahertz stage, something that escapes the external automated test equipment systems. Path delays and other timing problem have become crucial to operation; therefore, at full clock-rate, testing has become essential for submicron chips. Thus, logic BIST has become essential if timing faults are to be included in the coverage. The full scan automatic test pattern generation (ATPG) which can be conducted hierarchically isolating module by isolated module is an attractive solution, but BIST, included on an IP core, has proved to be far simpler. Again, 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 of core-based systems-on-a-chips (SoCs) called consecutive testability was introduced in [86]. It proposes a design-for-testability (DFT) method based on an integer programming problem. The input test patterns and test responses of a core are propagated to the core inputs from the SoC inputs and to the SoC outputs from the core outputs consecutively at the speed of the system clock. The propagation of test patterns and responses is achieved by using the consecutive transparency properties of surrounding cores and interconnects between cores. The proposed method can test logic faults such as stuck-at faults and timing faults such as delay faults.

A new compression method combined with scan power minimization technique for system-on-chip testing is investigated in [87]. Savings of up to 97% in peak power and 99% in average power as well as compression ratios of up to 95% can be achieved. That way, a space and time compaction methodology that guarantees a single-bit bandwidth and maximizes parallelism among core tests was proposed in [88].

A geometric shape-based compression/decompression scheme was introduced in [92] that uses reordering of the test vectors in can enable generation of geometric shapes which can be highly compressed via perfect loss-less compression was introduced in [92]. In [93], a compression/decompression mechanism was proposed which avoids test access mechanism (TAM) performances degradation.

A hybrid BIST solution for testing systems-on-chip [95] is also proposed to find the optimal balance between pseudorandom and stored test patterns to perform core test with minimum time and memory, without losing test quality. A DFT technique that allows core vendors to reduce the test complexity of a core they are trying to market was presented in [96]. A smaller number of test vectors resulting in less test data as well as less test time is achieved by carefully combining a parallel "test per clock" BIST scheme inside the core, together with the normal external testing scheme using a tester.

2.2 Space Compaction for Digital Core Test

Some of the common space compression techniques include the parity tree space compaction, hybrid space compression, dynamic space compression, quadratic functions compaction, programmable space compaction, and cumulative balance testing [3], [45].

The parity tree compactor circuits [44, 45] are composed of only XOR gates. An XOR gate has, very good signal-to-error propagation properties that are quite desirable for space compression. Functions realized by parity tree compactors are of the form $Z = Z_1 \oplus Z_2 \oplus \ldots \oplus Z_k$. The parity tree space compactor propagates all errors that appear on an odd number of its inputs. Thereby, errors that appear on an even number of parity tree circuit inputs are masked. As demonstrated experimentally, most single stuck-at line faults can be detected using pseudorandom input test pattern generators and reduced test sets.

The hybrid space compression (HSC) technique, originally proposed by Li and Robinson [41], uses AND, OR, and XOR logic gates as output compression tools to compress the multiple outputs of the CUT into a single line. The compaction tree is constructed based on the detectable error probability estimates.

A modified version of the HSC method, called dynamic space compression (DSC), was subsequently proposed by Jone and Das [37]. Instead of assigning static values for the probabilities of single and double line errors, the dynamic space compression (DSC) method dynamically estimates those values based on the CUT structure during the computation process. 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. A general theory to predict the performance of the space compression techniques was also developed in [37]. Experimental results show that the information loss, combined with the syndrome counting as time compactor, is between 0% and 12.7 %.

DSC was later improved in [30], in which some circuit-specific information was used to calculate the probabilities. However, neither HSC nor DSC does provide an adequate

measure of fault coverage because they both rely on estimates of error detection probabilities.

Quadratic functions compaction (QFC) [54] uses quadratic functions to construct the space compaction circuits, and has been shown to reduce aliasing. In QFC, the observed output responses $Z_1, \ldots, Z_k$ of the CUT are processed and compressed in a serial fashion based on a function of the type $Z_{i\,0} Z_{i\,1} \oplus Z_{i\,2} Z_{i\,3} \oplus \ldots \oplus Z_{i\,(2\,k-2)} Z_{i\,(2\,k-1)}$, where $Z_{i\,t}$ and $Z_{i\,(t+1)}$ are blocks of length k, for $t = 0, 2, \ldots, 2k-2$.

A new approach termed programmable space compaction (PSC) has recently been proposed for designing low-cost space compactors that provide high fault coverage [52]. In PSC, circuit-specific space compactors are designed to increase the likelihood of error propagation. However, PSC does not guarantee zero aliasing. A compaction circuit that minimizes aliasing and has the lowest cost can only be found by exhaustively enumerating all $(2^k)^k$ k-input Boolean functions, where k represents the number of primary outputs of the CUT.

A new class of space compactors based on parity tree circuits was recently proposed by Chakrabarty and Hayes [20]. The method is based on multiplexed parity trees (MPTs), and introduces zero aliasing. Multiplexed parity trees perform space compaction of test responses by combining the error propagation properties of multiplexers and parity trees through multiple time-steps. The authors show that the associated hardware overhead is moderate, and very high fault coverage is obtained for faults in the CUT including even those in the compactor.

An approach that implements both time compaction and space compaction such that a single signature is derived for all outputs using a simple scheme with suitable built-in implementation was introduced in [108]. The best current application of such a space-compacted signature is programmable logic arrays and read-only memories, where full testability of all single faults is achieved.

A parity-based space compaction technique that eliminates aliasing for any given fault model was presented in [110]. The test responses are merged into a narrow signature stream using a multiple-output parity tree.

In [111], a structural method for linear output space compaction is presented. This method is based on simple estimates for the probabilities of the existence of sensitized paths from the signal lines to the circuit outputs. Output partitions are determined without fault simulation.

A structure-independent space compaction method for testing embedded cores was proposed in [113]. A special code checker is used to design the compactor. Time compaction is also achieved using this method.

In [114], space- and time-oriented compaction techniques was proposed. The proposed scheme can compress the data of a k-output circuit into single bit signature stream with zero-aliasing and zero-performance-degradation for single stuck-line faults. In the time-oriented compaction approach developed in the paper, an s-bit long data stream can be compressed into an r-bit signature with zero-aliasing.

2.3 Time Compaction for Digital Core Test

The well known time compaction techniques include parity bit checking, transition count, one's count, Walsh coefficients, linear feedback shift register, and parallel compaction analysis. Based on these approaches, the compressed response data can be used to evaluate the correctness of the circuit under test (CUT).

One's count was proposed by Hayes [34] and is given by the number of ones in the binary circuit response stream during test execution. The hardware that represents the compaction unit consists of a simple counter, and is independent of the circuit under test. It only depends on the nature of the test response. The signatures values do not depend on the order in which the input test patterns are applied to the CUT. The length of the signature is a logarithmic function of the length of the output response data. Therefore, if the circuit response is of length r bits, then its signature will be of length $\lceil \log_2 r \rceil$. The probability distribution

function of aliasing is approximately Gaussian with a mean value at r/2, r being the length of the output stream. The masking probability [30] is low when the one's count of the signature is near either the minimum or maximum of its length range. Therefore, it takes a maximum value whenever the signature length approaches a maximum at the midrange of the count.

Syndrome testing [46] differs from the ordinary one's counting technique. The syndrome testing requires application of 2^n patterns to be applied to an n-input combinational circuit. A fault can be detected by comparing the number of ones at the output stream to the number of ones of the previously stored fault-free signature.

In transition counting [35], on the other hand, the signature is the number of 0-to-1 or 1-to-0 transitions in the output bit stream. The response compression length is less than or equal to $\lceil \log_2 r \rceil$, where r is the length of response stream. The masking probability takes high values when the signature value is close to r/2, and low values when it is close to 0 or r. In other words, masking in transition count depends on the number of faulty circuits that have the same transition count as the fault-free circuit. Unlike one's counting, transition counting is sensitive to the order of the bits in the response vector. However, transition counting does not guarantee the detection of all single-bit errors.

Double or multiple transition counting (DTC or MTC) [25-69] has recently been proposed as a new compaction technique that does not result in any loss of information. Single-output circuits can be tested using DTC testing, while MTC testing is used for multi-output circuits. DTC/MTC detects many faults that can also be detected by conventional testing methods. However, DTC/MTC techniques do not require repeated input test patterns and they do not use a counter. Test circuitry consists of an inverter, a switch, an OR logic gate, and a D flip-flop.

Parity compaction [61] on the response bit stream is realized by compression of the output data to a signature of length of only one bit. The value of this bit stream is 1 if the parity of the test response sequence is odd, and 0 if the parity is even. Parity compaction detects all errors involving an odd number of bits, while faults that give rise to an even number of error bits are not detected. This fact shows that parity checking is a relatively ineffective compaction method. The hardware implementation consists of a flip-flop and an XOR gate.

Walsh spectral compaction method is similar to syndrome analysis, which requires all possible input patterns to be applied to the combinational network. In Walsh spectral analysis [49], the switching functions are represented by their spectral coefficients that are compared to known correct coefficient values. Testing procedure checks the correctness of Walsh coefficients, which requires both an exhaustive and verification test of all Walsh coefficients. The spectral coefficient guarantees higher percentage of error coverage of the tested circuit. However, it also requires higher area overhead for generating them.

Testing using two different compaction schemes in parallel has been extensively investigated. The combination of signature analysis and transition counting has been analyzed in [47]. However, analysis shows that using simultaneously both techniques leads to a very small overlap in their error masking. As a result, the fault coverage can be improved while the fault signature size and the hardware overhead are increased.

Cyclic redundancy check (CRC) is an alternative method to transition count testing and is more popular. The CRC is easily implemented to detect errors in data communications. The CRC requires little overhead and it has extreme error detection capabilities. The CRC technique has been used for the compression of test response data [56]. The signature is the residue that is left in the feedback shift register after the response from the circuit under test has been compressed. One of its applications is as a source of pseudorandom binary test sequences; other applications are as means to carry out response compression. This latter process is commonly known as signature analysis [32].

Signature analysis is a compression technique based on the concept of cyclic redundancy checking (CRC) and realized in hardware using linear feedback shift registers (LFSRs), consisting of flip-flops and XOR gates. Signature analysis currently is the most popular time compaction technique. LFSRs are used for generating pseudorandom input test patterns and for response compaction as well. The nature of the generated sequence patterns is determined by the LFSR's characteristic polynomial as defined by its interconnection structure. Response sequence $P(x)$ is fed into the signature analyzer, and then divided by the characteristic polynomial $G(x)$ of the signature analyzer's LFSR. The remainder $R(x)$ obtained by dividing $P(x)$ by $G(x)$ over a Galois field such that $P(x) = G(x) \cdot Q(x) + R(x)$ represents the state of the LFSR. In other words, $R(x)$ represents the observed signature. The signature analysis involves comparing the

observed signature $R(x)$ to a known fault-free signature $R_0(x)$. An error is detected if these two signatures differ from each other. Suppose that $P(x)$ is the correct response and $P'(x) = P(x) + E(x)$ is the faulty one, where $E(x)$ is an error polynomial. It can be shown that aliasing occurs whenever $E(x)$ is a multiple of $G(x)$ or $E(x) = hG(x)$.

A method for computing and reducing the aliasing probability in signature analysis has been proposed by William et al. [50], which uses Markov chains and derives an upper bound on the aliasing probability in terms of the test length and probability of an error occurring at the output of the CUT.

An approach to the computation of the aliasing probability is presented in [43]. An error pattern in signature analysis causes aliasing, if and only if, it is a codeword in the cyclic code generated by the LFSR's characteristic polynomial.

Unlike other methods, the fault coverage in signature analysis may be improved without changing the test. This process can be implemented by changing the length of the LFSRs or by using different characteristic polynomial $G(x)$. As presented in [48], for short test lengths, signature analysis detects all single-bit errors. However, there is no known theory that characterizes fault detection in signature analysis.

Multiple-output circuits can be tested [58] using multi-input signature registers (MISRs). The use of MISRs in testing eliminates the need for a space compactor. However, MISRs increase aliasing and require extra hardware, which make it far from being practical.

A new approach based on modification of the test response for reducing aliasing probability has been proposed by Zorian et al. [3, 59]. This method suffers from two drawbacks: it involves large hardware overhead and increases testing time without ensuring zero aliasing.

Das et al. have recently proposed a multiple-output parity bit signature generation method for use with nonexhaustive or compact test sets in testing digital integrated circuits [78]. The suggested compaction technique provides high fault coverage for single stuck-line faults, with low CPU simulation time, and acceptable area overhead for the compactor. However, the design does not ensure a zero-aliasing compression. Information loss may not be completely avoided when the size of all output responses is reduced. Therefore, depending on the amount of information loss, the corresponding time compactor design will be affected

as well. Even though the method uses reduced input test sets which are *not* the minimal test sets needed to ensure a 100% fault coverage, experimental results indicate that the new designed multiple-output parity bit signature generators are comparable in all respects with conventional space-time compression, usually considered ideal for multiple-output combinational circuits.

CHAPTER 3

COMPACTION UNDER PAIRWISE MERGEABILITY

The subject chapter discuses space compaction techniques under pairwise mergeability of output bit streams of the CUT. The principal idea of the proposed approaches is to compress m functional test outputs of the CUT into one single test output line without sacrificing too much information. Generally, space compression has been accomplished using XOR gates in cascade or in a tree structure. We will here adopt a combination of both cascade and tree structures (cascade-tree) for our framework with AND (NAND), OR (NOR), and XOR (XNOR) operators. The logic function to be selected to build the compaction tree will be determined by the characteristics of the sequences, which are inputs to the gates based on some mergeability criteria [3]. We also assume a syndrome counter [43] at the output of the two-input gates of the compaction tree.

3.1 Pairwise Merger Under Stochastic Independence of Line Errors

The basic idea here is to select a suitable gate to merge two candidate output lines of the CUT. The selection is essentially based on certain mergeability criteria that use the properties of Hamming distance and sequence weights [36].

A mathematical s basis for the proposed approach is given below.

3.1.1 Mathematical basis for the proposed approach

3.1.1.1 Derived sequences

Let $\{\Psi_i, \Psi_j\}$ represent a pair of output sequences of length L_s, where the length is the number of bit positions in Ψ_i and Ψ_j. Let $H_s(\Psi_i, \Psi_j)$ represents the Hamming distance between Ψ_i and Ψ_j (the number of bit positions in which Ψ_i and Ψ_j differ).

Definition 1 The 1-weight, denoted by $W_1(\Psi_i)$, of a sequence Ψ_i, is the number of 1s in the sequence. Similarly, the 0-weight, denoted by $W_0(\Psi_i)$, of a sequence Ψ_i, is the number of 0s in the sequence.

Example 1 Consider an output sequence pair $\{\Psi_1, \Psi_2\}$ with Ψ_1 = (00101010), and Ψ_2 = (00001110). The length of both output streams is 8 (L_s= 8). The Hamming distance between Ψ_1 and Ψ_2 is 2 ($H_s(\Psi_1, \Psi_2) = 2$). The 1- and 0- weights of Ψ_1 and Ψ_2 are $W_1(\Psi_1) = 3$, $W_1(\Psi_2) = 3$, $W_0(\Psi_1) = 5$, and $W_0(\Psi_2) = 5$, respectively.

Property 1 For any sequence Ψ_i, it can shown that $L_s = W_0(\Psi_i) + W_1(\Psi_i)$.
For the output sequence Ψ_1 = (00101010) given in Example 1, we have found that $W_0(\Psi_1) = 5$ and $W_1(\Psi_1) = 3$. Therefore, it is obvious that $L_s = 5 + 3 = 8$.

Definition 2 Consider an output sequence pair $\{\Psi_i, \Psi_j\}$ of equal length L_s. Then the sequence pair derived by discarding the bit positions in which the two sequences differ (indicated by a dash -) is called its derived sequence pair, $\{\Psi'_i, \Psi'_j\}$.

In rest of the thesis, we will denote the derived sequence of a sequence Ψ_i by Ψ'_i, its 0-weight by $W'_0(\Psi'_i)$, and its 1-weight by $W'_1(\Psi'_i)$.

Example 2 For $\{\Psi_1, \Psi_2\}$ given in Example 1, $\{\Psi'_1, \Psi'_2\}$ = (00-01-10 , 00-01-10}. The 1- weight and 0-weight of the derived pair $\{\Psi'_1, \Psi'_2\}$ are: $W'_1(\Psi'_1) = 2$, $W'_1(\Psi'_2) = 2$, $W'_0(\Psi'_1) = 4$, and $W'_0(\Psi'_2) = 4$.

Property 2 For any derived sequence pair $\{\Psi'_i , \Psi'_j\}$, we have $W'_1(\Psi'_i) = W'_1(\Psi'_j)$ and $W'_0(\Psi'_i) = W'_0(\Psi'_j)$. i.e. as shown in Example 2, for the same output sequence pair $\{\Psi_i , \Psi_j\}$ given in Example 1, we have shown that $W'_1(\Psi'_1) = W'_1(\Psi'_2) = 2$ and $W'_0(\Psi'_1) = W'_0(\Psi'_1) = 4$.

By the above property, when no ambiguity arises, we will denote 0- and 1- weight for the derived sequence pair by simply W'_0 and W'_1, respectively. The length of the derived sequence pair will be denoted by L'_s, where $L'_s = L_s - H_s(\Psi_i , \Psi_j)$. Also, since $H_s(\Psi'_i , \Psi'_j)$ is always zero, we will simply use H_s to denote $H_s(\Psi_i , \Psi_j)$.That is, for $\{\Psi_i , \Psi_j\}$ in Example 1, Ψ'_1 = (00-01-10), Ψ'_2 = (00-01-10), $L_s = 8$ and $H_s(\Psi_1 , \Psi_2) = 2 = H_s$. Therefore, $L'_s = 8 - 2 = 6$ and $H_s(\Psi'_1 , \Psi'_2) = 0$.

Property 3 For every distinct pair of output sequences $\{\Psi_i , \Psi_j\}$ at the output of a CUT, the corresponding derived pair $\{\Psi'_i , \Psi'_j\}$ of equal length L'_s is distinct.
Two derived sequence pairs may have the same length L'_s but they are still distinct and not identical.

Property 4 Two derived sequence pairs $\{\Psi'_i , \Psi'_j\}$ and $\{\Phi'_i , \Phi'_j\}$ of original output streams pairs $\{\Psi_i , \Psi_j\}$ and $\{\Phi_i , \Phi_j\}$ having the same length L'_s are identical (in the set-theoretic sense). Thereby, $\{\Psi'_i , \Psi'_j\} = \{\Phi'_i , \Phi'_j\}$ if $\{\Psi_i , \Psi_j\} = \{\Phi_i , \Phi_j\}$.

Consider Ψ_1 = (00001010), Ψ_2 = (00101110), Φ_1 = (00001111), and Φ_2 = (00000011) as four output sequence streams of a CUT. Let $\{\Psi_1 , \Psi_2\}$ and $\{\Phi_1 , \Phi_2\}$ be two distinct output pairs, and $\{\Psi'_1 , \Psi'_2\}$, $\{\Phi'_1 , \Phi'_2\}$ their corresponding derived sequence pairs, respectively, such that $\{\Psi'_1 , \Psi'_2\}$ = {00-01-10 , 00-01-10} and $\{\Phi'_1 , \Phi'_2\}$ = {0000--11 , 0000--11}. Both derived sequence pairs have the same length $L'_s = 6$ but they are not identical.

However, in general, it is not expected that any two distinct pairs of sequences at the output of the CUT will be identical and hence the possibility of the corresponding derived pairs being identical is also remote.

3.1.2 Mergeability criteria and gate selection

Definition 3 The maximum expectation of error (E_E) for two-input logic functions, given two input sequences of length L_s is defined as the sum of all single line errors ($2L_s$) and all double line errors (L_s).

This definition assumes stochastic independence of single line and double line errors and thus includes all of the possibilities of error occurrence. We will define later the mergeability criteria assuming distinct probabilities for single and double line errors.
Now let $E_s(g)$ be the number of single line errors and $E_d(g)$ be the number of double line errors detected at the output of a gate g.
For the output sequence pair $\{\Psi_1, \Psi_2\}$ given in Example 1, the maximum expectation of error (E_E) for two-input logic functions is $E_E = 2 \times 8 + 8 = 24$.

Definition 4 The maximum detectable error ($E_{max}(g)$) for two-input logic functions, given an input sequence pair $\{\Psi_i, \Psi_j\}$ of length L_s when the gate type is g, is defined as $E_{max}(g) = E_s(g) + E_d(g)$, as given in Table 3.1.

Gate Type (g)	Maximum Detectable Error	Detectable Error
AND/NAND	$E_{max}(AND/NAND) = 2L_s + L_s$	$E(AND/NAND) \leq 2L_s + L_s$
OR/NOR	$E_{max}(OR/NOR) = 2L_s + L_s$	$E(OR/NOR) \leq 2L_s + L_s$
XOR/XNOR	$E_{max}(XOR/XNOR) = 2L_s + 0$	$E(XOR/XNOR) = 2L_s + 0$

Table 3.1 The Detectable and Maximum Detectable Error Using Different Type of Gates.

$E_{max}(XOR/XNOR) = E_s(XOR/XNOR) + E_d(XOR/XNOR) = 2L_s + 0 = E(XOR/XNOR)$ where $E_s(XOR/XNOR)$ and $E_d(XOR/XNOR)$ are, respectively, the single and double line detectable errors when using XOR or XNOR gates.

Theorem 1 For any derived output sequence pair $\{\Psi'_1 , \Psi'_2\}$ of length L'_s, if an AND (NAND) gate is used to merge the derived sequence pair, the total number of detected errors will be $3W'_1$, $2W'_1$ being single line errors and W'_1 double line errors (assuming $W'_0 = 0$).

For $\{\Psi_1 ,\Psi_2\}$ in Example 1, $W'_1(\Psi'_1) = 2$ and $W'_1(\Psi'_2) = 2$. Therefore, the total number of errors detected at the output of an AND (NAND) gate is 6 (all errors are detected).

Theorem 2 For any derived output sequence pair $\{\Psi'_i , \Psi'_j\}$ of length L'_s, if an OR (NOR) gate is used to merge the derived sequence pair, the total number of detected errors will be $3 W'_0$, $2W'_0$ being single line errors and W'_0 double line errors (assuming $W'_1 = 0$).

For $\{\Psi_1 ,\Psi_2\}$ in Example 1, $W'_0(\Psi'_1) = 4$ and $W'_0(\Psi'_2) = 4$. Therefore, the total number of errors detected at the output of an OR (NOR) gate is 12 (all errors are detected).

Theorem 3 If an output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s is merged with an AND (NAND) gate, the maximum errors detected $E_{max}(AND/NAND)$ will be as follows:

$E_{max}(AND/NAND) = H_s$ *single line errors* + $3W'_1$ *single and double line errors* + W'_0 *double line errors.*

Proof: Since the Hamming distance is H_s, if one sequence has a 0, the other has a 1 in the corresponding position for all the H_s bits. If an AND (NAND) gate is used, only single fault that makes both sequences 1 will be detectable, and thus H_s single line faults are detected. Similarly, $3W'_1$ single line and double line faults are detected if W'_1 is the 1-weight of the derived sequence pair. Likewise, if W'_0 is the 0-weight of the derived sequence pair, only W'_0 double line faults making both sequences 1 in the corresponding positions will be detected.

Theorem 4 If an output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s is merged using an OR (NOR) gate, the maximum errors detected $E_{max}(OR/NOR)$ will be as follows:

$E_{max}(OR/NOR) = H_s$ *single line errors* + $3W'_0$ *single and double line errors* + W'_1 *double line errors.*

Proof: As before, since the Hamming distance is H_s, if one sequence has a 0 the other has a 1 in the corresponding position for all the H_s bits. If an OR (NOR) gate is used, only single fault that makes both sequences 0 will be detectable, and hence H_s single line faults are

detected. Similarly, $3W'_0$ single line and double line faults are detected if W'_0 is the 0-weight of the derived sequence pair. Likewise, if W'_1 is the 1-weight of the derived sequence pair, only W'_1 double line faults making both sequences 0 in the corresponding positions will be detected.

Theorem 5 An output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s is XOR (XNOR) gate mergeable if the maximum number of errors detected is $2L_s$, which is greater than or equal to the maximum number of errors detected by AND (NAND) gates $E_{max}(AND/NAND)$ or OR (NOR) gates $E_{max}(OR/NOR)$, when used for merger.

Proof: The theorem is a direct consequence of Theorems 4 and 5.

Theorem 6 For any output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s, an OR (NOR) gate or an AND (NAND) gate may be used for merger if the maximum number of errors detected is E_{max} where $2L_s \leq E_{max} \leq 3L_s$.

Proof: For sequences of length L_s, the maximum expectation of error E_E is $3L_s$ ($2L_s$ single line and L_s double line errors). An XOR (XNOR) gate can only detect $2L_s$ single line errors. Hence, if the error detection count is between $2L_s$ and $3L_s$, we need to use OR (NOR) or AND (NAND) gates based on whether E_{max}(AND/NAND) or E_{max}(OR/NOR) has a greater value.

Theorem 7 An AND (NAND) gate may be selected for merging an output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s with Hamming distance H_s if and only if, $W'_1 > W'_0$ and $E_{max}(AND/NAND) = H_s$ *single line errors* $+ 3W'_1$ *single and double line errors* $+ W'_0$ *double line errors* $\geq 2L_s$

Proof: Since E_{max}(AND/NAND) is $H_s + 3W'_1 + W'_0$ and E_{max}(OR/NOR) is $H_s + 3W'_0 + W'_1$, if AND/NAND is preferable to OR/NOR, then obviously $E_{max}(AND/NAND) - E_{max}(OR/NOR) > 0$, or $H_s + 3W'_1 + W'_0 - H_s + 3W'_0 + W'_1 > 0$, or $W'_1 > W'_0$. Further, if $E_{max}(AND/NAND) \geq 2L_s$, total faults detected by XOR/XNOR gate, then evidently AND/NAND is selected for merger.

Theorem 8 An OR (NOR) gate may be selected for merger of an output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s with Hamming distance H_s if and only if $W'_0 > W'_1$ and

$E_{max}(OR/NOR) = H_s$ *single line errors* $+ 3W'_0$ *single and double line errors* $+ W'_1$ *double line errors* $\geq 2L_s$.

Proof: Follows as in Theorem 7.

Corollary 8.1 However, if $W'_0 = W'_1$, either OR (NOR) or AND (NAND) gate may be used for merger.

Theorem 9 An output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s needs to be merged with an AND (NAND) gate if and only if $E_{max}(AND/NAND)$ is maximized over $m(m-1)/2$ sequence pairs for an m-output CUT.

Proof: Since $E_{max}(AND/NAND)$, considering all output sequence pairs for an m-output CUT (m(m-1)/2), has the greatest value, obviously, AND/NAND gate must be selected.

Theorem 10 An output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s and Hamming distance H_s needs to be merged with an OR (NOR) gate if and only if $E_{max}(OR/NOR)$ is maximized over $m(m-1)/2$ sequence pairs for an m-output CUT.

Proof: If $E_{max}(OR/NOR)$ is maximized over all m(m-1)/2 output sequence pairs for an m-output CUT, then evidently OR/NOR gate should be selected since OR/NOR detects the largest number of faults.

Theorem 11 An output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s is AND (NAND) gate mergeable if and only if $W'_1 > L_s/2$.

Proof: $E_{max}(AND/NAND) = H_s + 3W'_1 + W'_0$ whereas $E_{max}(XOR/XNOR) = 2L_s$. If AND/NAND is selected over XOR/XNOR, then $E_{max}(AND/NAND) - E_{max}(XOR/XNOR) > 0$,

$$\text{or } H_s + 3W'_1 + W'_0 - 2L_s > 0,$$

$$\text{or } (H_s + W'_1 + W'_0) + 2W'_1 - 2L_s > 0,$$

$$\text{or } L_s + 2W'_1 - 2L_s > 0, \text{ or } W'_1 > L_s / 2.$$

Besides, if $W'_1 > L_s / 2$, W'_1 is also greater than W'_0 and hence the selection of AND/NAND gate is unique.

Theorem 12 An output sequence pair $\{\Psi_i , \Psi_j\}$ of length L_s is OR (NOR) gate mergeable if and only if $W'_0 > L_s/2$.

Proof: The theorem follows in the same way as in Theorem 11.

Algorithm 1

The algorithm for implementation of the proposed method is given below in a pseudocode format.

stage = total_number_of_outputs
current_stage = 0
while (current_stage < stage-1)
{
for (i = 0; i < total_number_of_outputs; i++)
    for (j = 1; j < total_number_of_outputs, j ≠ i; j++)
    {
    Determine the derived sequence pair $\{\Psi'_i , \Psi'_j\}$ that corresponds to the sequence pair $\{\Psi_i , \Psi_j\}$.
    Compute its corresponding Hamming distance H_s, 0-weight W'_0, and 1-weight W'_1 .
    Select a gate for merger based on the mergeability criteria discussed earlier (AND/NAND gate if $W'_1 > (L_s / 2)$;
    OR/NOR gate if $W'_0 > (L_s / 2)$; XOR/XNOR gate otherwise) and compute E_{max} value for the selected gate.
    }
Select the sequence pair $\{\Psi_i , \Psi_j\}$ for merger that has the maximum E_{max} value, choosing one arbitrarily when there is a tie.
Compute the corresponding new output sequence using the chosen gate.
Discard sequence pair $\{\Psi_i , \Psi_j\}$ which will no longer be used as the result of this selection.
current_stage++
}

Example 3 Consider the Ten-line decimal-to-8421 BCD converter shown in Figure 3.1. The circuit has 9 primary inputs X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, and X_9, and four primary outputs O_1, O_2, O_3, and O_4. For the ten grouped input test patterns, the fault-free output patterns are shown below in Table 3.2.

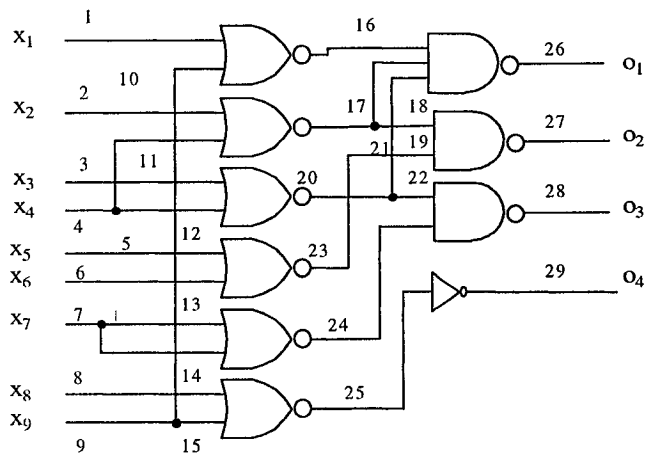


Figure 3.1 Ten-Line Decimal-to-8421 BCD Converter.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	O_1	O_2	O_3	O_4
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	1	1	0	0	1

Table 3.2 Fault-Free Output Patterns for the Ten-Line Decimal-to-8421 BCD.

There are only six output sequence pairs $\{O_i, O_j\}$ $i = 1,...,6; j = 1,...,6; i \neq j$ to consider in the first stage. The Hamming distances of all sequence pairs are computed and given as follows: $H_s(O_1, O_2) = H_s(O_1, O_3) = H_s(O_1, O_4) = 5$; $H_s(O_2, O_3) = 4$; $H_s(O_2, O_4) = H_s(O_3, O_4) = 6$.

Furthermore, W'_0 and W'_1, the 0-weight and 1-weight, respectively, of the derived sequences corresponding to every sequence pair are as follows: $W'_0(O_1, O_2) = W'_0(O_1, O_3) = 3$; $W'_1(O_1, O_2) = W'_1(O_1, O_3) = W'_1(O_2, O_3) = 2$;
$W'_0(O_1, O_4) = W'_0(O_2, O_3) = W'_0(O_2, O_4) = W'_0(O_3, O_4) = 4$; $W'_1(O_1, O_4) = 1$; $W'_1(O_2, O_4) = W'_1(O_3, O_4) = 0$.
Since, $E_{max}(XOR/XNOR) = 2L_s = 20$ ($L_s = 10$) is greater than $E_{max}(AND/NAND)$ and $E_{max}(OR/NOR)$ for every pair of sequences, therefore every pair is XOR (XNOR) gate mergeable. An XOR gate is selected to merge O_1 and O_2 chosen arbitrarily. The remaining sequences O_3 and O_4 are also merged with an XOR gate. The two output sequences in the subsequent stage turn out to be also XOR gate mergeable; so they are merged with an XOR gate to complete the compression tree. The circuit with the compression tree is shown in Figure 3.2.

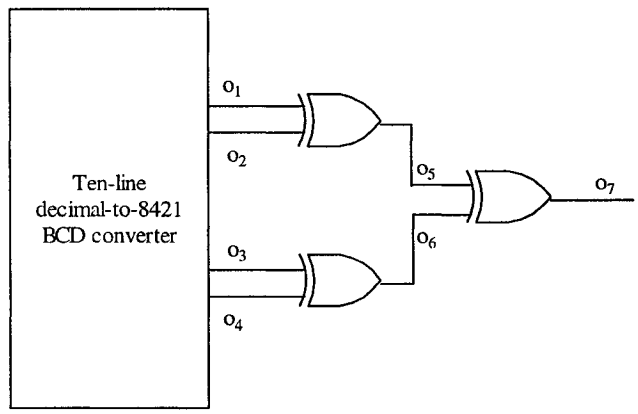


Figure 3.2 The Corresponding Compaction Tree.

The Ten-line decimal-to-8421 BCD converter has got 58 single stuck-line faults, 2 undetectable faults at ${}_{11}{}^{0}O_1$ and ${}_{12}{}^{0}O_1$ (the effect of lines 11 and 12 being stuck-at-0 is not visible at output O_1), 34 equivalent faults, and 22 single stuck-line faults in its collapsed fault set. All detectable single stuck-at faults are injected into the circuit.

The fault effect for this circuit at the outputs with the test set of Table 3.2 is shown in Table 3.3. The fault-free space compactor response for the 10 input test vectors is the following single 10-bit stream $O_7 = (0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0)$.

If our space compactor is followed by a time compactor (i.e. Syndrome Counter), the resulting fault-free signature for the output stream O_7 is simply the number of ones that will be equal to 5 in this case. This fault-free signature is saved in memory to be compared to the faulty signatures in order to determine the percentage of loss.

Fault Effect	$_1{}^0O_1$	$_1{}^1O_1$	$_2{}^0O_1$	$_2{}^0O_2$	$_2{}^1O_2$	$_3{}^0O_1$	$_3{}^0O_3$	$_3{}^1O_3$	$_4{}^0O_1$	$_4{}^0O_2$	$_4{}^0O_3$	$_5{}^0O_2$	$_6{}^0O_2$	$_6{}^0O_3$	$_7{}^0O_3$	$_8{}^0O_4$	$_8{}^1O_4$	$_9{}^0O_1$
For The	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
Same	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1
Grouped	1	1	0	0	1	1	0	1	1	1	0	1	1	0	0	0	1	1
Input Test	1	1	1	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1
Patterns	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	1	1
Used to	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	1	0
Calculate	0	1	0	1	1	0	1	1	0	1	1	1	0	0	1	0	1	0
The	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	1	0
Fault-free	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
Responses	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	1	1	0

Fault Effect	$_9{}^0O_4$	$_{16}{}^1O_1$	$_{17}{}^1O_1$	$_{17}{}^1O_2$	$_{20}{}^1O_1$	$_{20}{}^1O_3$	$_{23}{}^1O_2$	$_{24}{}^1O_3$	$_{25}{}^1O_4$	$_{26}{}^0O_1$	$_{27}{}^0O_2$	$_{28}{}^0O_3$
For The	0	0	0	0	0	0	0	0	0	0	0	0
Same	0	0	1	0	1	0	0	0	0	0	0	0
Grouped	0	1	0	0	1	0	1	0	0	0	0	0
Input Test	0	1	1	0	0	0	0	1	0	0	0	0
Patterns	0	1	0	0	0	0	1	1	0	0	0	0
Used to	0	0	0	1	0	0	0	0	0	0	0	0
Calculate	0	0	0	1	0	1	0	0	0	0	0	0
The	0	0	0	0	0	1	0	0	0	0	0	0
Fault-free	1	0	0	0	0	0	0	0	0	0	0	0
Responses	0	0	1	0	1	0	0	0	0	0	0	0

Table 3.3 The Fault Effect at The Outputs of The Ten-Line Decimal-to-8421 BCD Converter.

The faulty signatures corresponding to the single line faults ${}_{1}{}^{0}O_{1}$, ${}_{1}{}^{1}O_{1}$, ... , ${}_{27}{}^{0}O_{2}$, ${}_{28}{}^{0}O_{3}$ are, respectively, the following: 4, 10, ... , 0,0. Comparing all the faulty signatures to the expected signature, we notice that two of the faulty signatures match the fault-free signature. Therefore, 56 detectable faults are injected into the circuit and the percentage of loss before compaction is 0.00 %. Furthermore, we calculate percentage of fault coverage at the output of the space compactor by comparing the fault-free stream bits to the faulty stream bits at the output of the compactor. The number of undetected faults turns out to be 5 at the output of the space compactor. Hence the percentage of fault coverage at the output of the compressor equals to 91.07 %, which is equivalent to 8.92 % fault loss.

3.2 Pairwise Merger Under Stochastic Dependence of Line Errors

In the previous section, we presented the new compaction technique assuming stochastic independence of single line and double line errors. In the absence of stochastic independence, we observe that the probability plays an important role in the selection of gates for merger. In the following section we deal with the construction of the compaction tree when single and double line errors are considered stochastically dependent [3].

3.2.1 Effect of error probability in selection criteria

Li and Robinson [41] determine the detectable error probability estimate ε for a two input logic function, given two input sequences of length L_s as follows:

$$\varepsilon = S_1 \frac{2R_1}{2L_s} + S_2 \frac{R_2}{L_s}$$

where S_1: Probability of single error effect felt at the output of the CUT;

S_2: Probability of double error effect felt at the output of the CUT;

R_1: Number of single line errors at the output of gate *i*, if gate *i* is used;

R_2: Number of double line errors at the output of gate *i*, if gate *i* is used.

Based on the detectable error probability estimate ε of Li and Robinson as given above, we deduce the following results that profoundly influence the selection of gates for merger.

Theorem 13 For an output sequence pair $\{\Psi_i, \Psi_j\}$ of length L_s and Hamming distance H_s, an AND/NAND gate is preferable to an XOR(XNOR) gate if

$$S_1 < \frac{[2W'_1 + 2W'_0]}{[H_s + 4W'_0 + 2W'_1]}$$

Proof: The detectable error probability estimate when an AND/NAND gate is used is:

$$\varepsilon\ (A/N) = S_1 \frac{[H_s + 2W'_1]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

While the corresponding one when an XOR/XNOR gate is used is:

$$\varepsilon\ (X/X) = S_1 \frac{[\ 2L_s\ ]}{2L_s} = S_1$$

An AND/NAND gate is preferable to an XOR/XNOR gate if:

ε **(A/N)** > ε **(X/X)** , Or, ε**(A/N)** - ε**(X/X)** > 0 ,

$$S_1 \frac{[H_s + 2W'_1]}{2L_s} + \left[S_2 \frac{[W'_0 + W'_1]}{L_s} - S_1 \right] > 0,$$

$$S_1 \frac{[H_s + 2W'_1]}{2L_s} + \left[[1-S_1] \frac{[W'_0 + W'_1]}{L_s} - S_1 \right] > 0,$$

$$S_1 \frac{[H_s + 2W'_1 - 2L_s]}{2L_s} + \left[\frac{[1-S_1][W'_0 + W'_1]}{L_s} \right] > 0,$$

$$S_1 \frac{[H_s + 2W'_1 - 2L_s - 2W'_0 - 2W'_1]}{2L_s} + \left[\frac{[W'_0 + W'_1]}{L_s} \right] > 0,$$

$$S_1 \frac{[-H_s - 4W'_0 - 2W'_1]}{2L_s} > \left[\frac{-[W'_0 + W'_1]}{L_s} \right],$$

$$S_1 < \frac{2\,[W'_0 + W'_1]}{[H_s + 4W'_0 + 2W'_1]}.$$

Corollary 13.1 On the other hand, an XOR/XNOR gate is selected if

$$S_1 > \frac{[2W'_0 + 2W'_1]}{[H_s + 4W'_0 + 2W'_1]}.$$

Theorem 14 For an output sequence pair $\{\Psi_i, \Psi_j\}$ of length L_s and Hamming distance H_s, an OR (NOR) gate is preferable to an XOR/XNOR gate if

$$S_1 < \frac{2[W'_0 + W'_1]}{[H_s + 4W'_1 + 2W'_0]}.$$

Proof: The detectable error probability estimate when an OR/NOR gate is used is:

$$\mathcal{E}(O/N) = S_1 \frac{[H_s + 2W'_0]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

while the corresponding one when an XOR/XNOR gate is used is:

$$\mathcal{E}(X/X) = S_1 \frac{[2L_s]}{2L_s} = S_1$$

An OR/NOR gate is obviously preferable to an XOR/XNOR gate if:

$$\varepsilon(\mathbf{O/N}) > \varepsilon(\mathbf{X/X}),$$

$$\varepsilon(\mathbf{O/N}) - \varepsilon(\mathbf{X/X}) > \mathbf{0},$$

$$S_1 \frac{[H_s + 2W'_0]}{2L_s} + \left[S_2 \frac{[W'_0 + W'_1]}{L_s} - S_1\right] > 0,$$

$$S_1 \frac{[H_s + 2W'_0]}{2L_s} + \left[[1-S_1] \frac{[W'_0 + W'_1]}{L_s} - S_1\right] > 0,$$

$$S_1 \frac{[H_s + 2W'_0 - 2L_s]}{2L_s} + \left[\frac{[1-S_1][W'_0 + W'_1]}{L_s}\right] > 0,$$

$$S_1 \frac{[H_s + 2W'_0 - 2L_s - 2W'_0 - 2W'_1]}{2L_s} + \left[\frac{[W'_0 + W'_1]}{L_s}\right] > 0,$$

$$S_1 \frac{[-H_s - 4W'_1 - 2W'_0]}{2L_s} > \left[\frac{-[W'_0 + W'_1]}{L_s}\right],$$

$$S_1 < \frac{2[W'_0 + W'_1]}{[H_s + 4W'_1 + 2W'_0]}.$$

Corollary 14.1 On the other hand, an XOR/XNOR gate is selected if

$$S_1 > \frac{[2W'_0 + 2W'_1]}{[H_s + 4W'_1 + 2W'_0]}.$$

However, the probability does not play any role in the selection between AND (NAND) and OR (NOR) if we use the empirical formula of Li and Robinson. The under noted theorem states the condition that determines the selection criteria between AND/NAND and OR/NOR gates.

Theorem 15 For an output sequence pair $\{\Psi_i, \Psi_j\}$ of length L_s and Hamming distance H_s, an AND (NAND) gate is preferable to an OR (NOR) gate if $W'_1 > W'_0$.

Proof: The detectable error probability estimate when an AND/NAND gate is used is:

$$\mathcal{E}\ (A/N) = S_1 \frac{[H_s + 2W'_1]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

and the corresponding one when an OR/NOR gate is used is:

$$\mathcal{E}\ (O/N) = S_1 \frac{[H_s + 2W'_0]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

An AND/NAND gate is surely preferable to an OR/NOR gate if:

$$\mathbf{\varepsilon(A/N) - \varepsilon(O/N) > 0},$$

$$S_1 \frac{[H_s + 2W'_1 - H_s - 2W'_0]}{2L_s} + S_2 \frac{[W'_0 + W'_1 - W'_0 - W'_1]}{L_s} > 0,$$

$$S_1 \frac{[W'_1 - W'_0]}{L_s} > 0,$$

$$W'_1 > W'_0 \cdot$$

Corollary 15.1 An OR/NOR gate is selected if on the other hand

$$W'_1 < W'_0 \cdot$$

Theorem 16 Sequence merger following the Li and Robinson criterion may result in a compression network that is different from the one that will result based on the selection criteria given above even if the same probability estimate of Li and Robinson is used as the guide to selection.

Proof: The proof is provided by the following example. Consider the following three sequences Ψ_1 = (0000001000000000), Ψ_2 = (0000000100000000), and Ψ_3 = (0000110000000000). By Li and Robinson criterion, we will merge Ψ_1 and Ψ_3, while according to our selection criteria, Ψ_1 and Ψ_2 will be merged with an OR (NOR) gate because E_{max} *(OR/NOR)* for $\{\Psi_2, \Psi_3\}$ equals 0.81, E_{max} *(OR/NOR)* for $\{\Psi_1, \Psi_3\}$ also equals 0.81, whereas E_{max} *(OR/NOR)* for $\{\Psi_1, \Psi_2\}$ equals 0.87 which is obviously the greatest probability estimate, assuming $S_1 = 0.0$ and $S_2 = 1.0$.

The next theorem establishes that the gate selection based on the aforementioned results provide a stricter guideline as opposed to the criteria used by Li and Robinson.

Theorem 17 The use of the detectable error probability estimate ε as suggested by Li and Robinson and discussed earlier gives a weaker condition in gate selection for output merger in the construction of the compaction network compared to the selection criteria proposed by Theorem 13 through 16.

Proof: Consider the following four output sequences:

W:	0000100000000000	X:	0000010000000000
Y:	0000001000000000	Z:	0000000100000000

All sequence pairs {W , X}, {W , Y}, {W , Z}, {X , Y}, {X , Z}, and {Y , Z} have the same characteristics

H_s = 2, W'_0 = 14, and W'_1 = 0. For S_1 = 0.0 and S_2 = 1.0, the corresponding probability estimates of all sequence pairs are the following:

ε(AND/NAD) = 0.87 ε(OR/NOR) = 0.87 ε(XOR/XNOR) = 0.0 .

Since sequence pairs have the same probability estimate, any one can be selected for merger (i.e. {W, X}). Now, if we use Li and Robinson formula to merge the selected output sequences, then we will have to decide between choosing AND or OR gate (both have got the same probability estimate). But, according to our mergeability criteria, an OR gate should be used for merger because $W'_0(W, X) = 14$ is greater than $W'_1(W, X) = 0$. This shows that our condition in gate selection is stronger than Li and Robinson selection criteria for output merger in the construction of the compaction network.

We now prove an important theorem concerning gate selection for merger with certain particular but generally chosen values of probability S_1 and S_2 for single and double line errors, respectively.

Theorem 18 The gate selection criteria established earlier under condition of stochastic independence of single line and double line errors at the output of a CUT remains unchanged even under condition of stochastic dependence of single line and double line errors based on computation of detectable error probability estimate ε, the empirical formula of Li and Robinson, if the values of the probabilities S_1 and S_2 are chosen as S_1 = 2/3 and S_2 = 1/3.

Proof: The detectable error probability estimates, as used earlier, for AND/NAND, OR/NOR and XOR/XNOR gates are respectively,

$$\mathcal{E}(A/N) = S_1 \frac{[H_s + 2W'_1]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

$$\mathcal{E}(O/N) = S_1 \frac{[H_s + 2W'_0]}{2L_s} + S_2 \frac{[W'_0 + W'_1]}{L_s}$$

$$\mathcal{E}(X/X) = S_1 \frac{[2L_s]}{2L_s} = S_1$$

These estimates are for output sequences of length L_s, Hamming distance H_s, 1-weight W'_1 and 0-weight W'_0, with probability of single and double line errors being, respectively, S_1 and S_2, as usual. Evidently, an AND/NAND gate is preferable to an XOR/XNOR gate, if ε (A/N) - ε (X/X) > 0, and similarly, an OR/NOR gate is preferable to an XOR/XNOR gate, if ε (O/N) - ε (X/X) > 0. With values of S_1 and S_2 being, respectively, 2/3 and 1/3, we have :

$$\varepsilon_{(A/N)} = \frac{[H_s + 3W'_1 + W'_0]}{3L_s} ,$$

$$\varepsilon_{(O/N)} = \frac{[H_s + 3W'_0 + W'_1]}{3L_s} ,$$

$$\varepsilon_{(X/X)} = 2/3 .$$

Then

$$\varepsilon_{(O/N)} - \varepsilon_{(X/X)} = \frac{[H_s + 3W'_1 + W'_0]}{3L_s} - 2/3 > 0,$$

$$[H_s + 3W'_1 + W'_0] - 2L_s > 0,$$

$$[H_s + 3W'_1 + W'_0 - 2H_s - 2W'_1 - 2W'_0] > 0,$$

$$[W'_1 - W'_0 - H_s] > 0,$$

$$[W'_1 - W'_0 - L_s + W'_1 + W'_0] > 0,$$

$$2W'_1 > L_s ,$$

$$W'_1 > L_s/2 .$$

This is exactly the same as the one established previously for AND/NAND gate selection under condition of stochastic independence of errors. It can be shown likewise that an OR/NOR gate is preferable to XOR/XNOR gate, if

$$W'_0 > L_s/2 ,$$

and AND/NAND gate is preferable to OR/NOR, if

$$W'_1 > W'_0 \cdot$$

We thus see that our gate selection criteria need not to be changed even under condition of stochastic dependence of errors when their probabilities are $S_1 = 2/3$ and $S_1 = 1/3$, which considerably simplifies the computational problem involved with detectable error probability estimate ε.

Algorithm 2

The pseudocode description of the algorithm that constructs the compaction tree assuming different probability values for single and double line errors is given below:

Define the probability of single (S_1) and double (S_2) error effect felt at the output of the CUT.
stage = total_number_of_outputs
current_stage = 0
while (current_stage < stage-1)
{
for (i = 0; i < total_number_of_outputs; i++)
for (j = 1; j < total_number_of_outputs, $j \neq i$; j++)
{
Determine the derived sequence pair $\{\Psi'_i , \Psi'_j\}$ that corresponds to the sequence pair $\{\Psi_i , \Psi_j\}$.
Compute its corresponding Hamming distance H_s, 0-weight W'_0, and 1-weight W'_1 .
Compute its corresponding number of single (R_1) and double (R_2) line errors at the output of the three basic type of gates (AND/NAND, OR/NOR,and XOR/XNOR).
Select the sequence pair $\{\Psi_i , \Psi_j\}$ for merger based on the detectable error probability estimate formula $\varepsilon = S_1 (R_1 / 2L_s) + S_2 (R_2 / L_s)$ discussed earlier, choosing one arbitrarily when there is a tie.
}
Select a gate to merge the chosen sequence pair $\{\Psi_i , \Psi_j\}$ based on the mergeability criteria, an XOR/XNOR gate is preferable to an AND/NAND gate if $S_1 > [2W'_1 + 2W'_0] / [H_s + 4W'_0 + 2W'_1]$ an XOR/XNOR gate is preferable to an OR/NOR gate if $S_1 > [2W'_1 + 2W'_0] / [H_s + 4W'_1 + 2W'_0]$ an AND/NAND gate is preferable to an OR/NOR gate if $W'_1 > W'_0$.
Compute the corresponding new output sequence using the chosen gate.
Discard sequence pairs $\{\Psi_i , \Psi_j\}$ which will no longer be used as a result of this selection.
current_stage++
}

3.2.2 Input sequence length

Apparently, the length of the test input patterns L applied to the CUT plays an important role in the construction of the circuit's compaction tree. The nature of its elements (line selection and type of logic gates used) changes simultaneously with the modification of the input test length L. Furthermore, as will be shown in the results section, the input test length has a significant effect on the percentage of faults coverage for the circuit.

Example 4 Consider the three sequences used in Example 1, Ψ_1 = (111111000), Ψ_2 = (010101000), and Ψ_3 = (000100010). All sequences are of length L = 9. We have shown that according to our selection criteria, Ψ_2 and Ψ_3 will be used for merger. If we now consider the same sequences, each increased by 6 bits of 1 (L = 15), we have the new sequences as follows: Ψ_1 = (111111000111111),), Ψ_2 = (010101000111111), and Ψ_3 = (000100010111111); and now Ψ_1 and Ψ_2 will be selected for merger (assuming S_1 = 0.666 and S_2 = 0.333).

Example 5 Once again, consider the Ten-line decimal-to-8421 BCD converter shown in Figure 4. The circuit has got six sequences pairs $\{O_i, O_j\}$, $i = 1,...,6; j = 1,...,6; i^1j$ to be considered in the first stage. The Hamming distances of all sequence pairs are calculated as follows: $H_s(O_1, O_2) = H_s(O_1, O_3) = H_s(O_1, O_4) = 5; H_s(O_2, O_3) = 4; H_s(O_2, O_4) = H_s(O_3, O_4) = 6$. The 0-weight and 1-weight of the derived sequence pairs are: $W'_0(O_1, O_2) = W'_0(O_1, O_3) = 3; W'_1(O_1, O_2) = W'_1(O_1, O_3) = W'_1(O_2, O_3) = 2; W'_0(O_1, O_4) = W'_0(O_2, O_3) = W'_0(O_2, O_4) = W'_0(O_3, O_4) = 4; W'_1(O_1, O_4) = 1; W'_1(O_2, O_4) = W'_1(O_3, O_4) = 0.$ Each pair of sequences has got different number of single (R_1) and double (R_2) line errors at the output of gate i, if gate i is used for merger. The pair $\{O_2, O_3\}$ has got R_1 (AND/NAND) = 8, R_2 (AND/NAND) = 6, R_1 (OR/NOR) = 12, R_2 (OR/NOR) = 6, R_1 (XOR/XNOR) = 20, and R_2 (XOR/XNOR) = 0. The corresponding probability estimate ε (OR/NOR) = 0.6 is the highest value (assuming S_1 = 0.05 and S_2 = 0.95). Therefore, O_2 and O_3 are selected for merger in the first stage. They are merged with an OR gate because both mergeability equations

$S_1 > [2W'_1 + 2W'_0] / [H_s + 4W'_1 + 2W'_0]$ and $S_1 > [2W'_1 + 2W'_0] / [H_s + 4W'_1 + 2W'_0]$ are not satisfied, and $(W'_1 = 2) < (W'_0 = 4)$. The two output sequences O_1 and O_4 in the subsequent stage turn out to be also OR/NOR mergeable (merged with an OR gate), while

in the last stage, O_5 and O_6 are merged with an AND gate, completing the compaction tree. The circuit with the compression tree is shown in Figure 3.3.

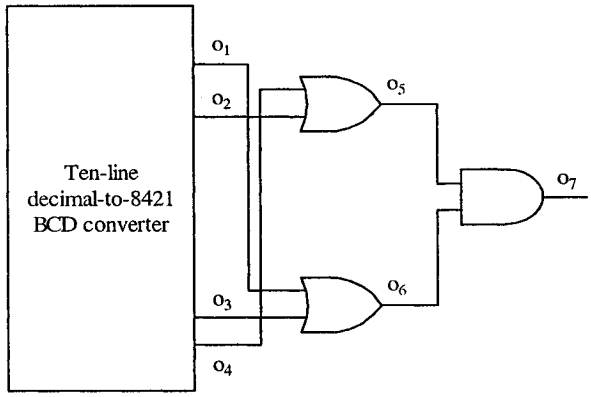


Figure 3.3 The Converter Circuit with its Corresponding Compaction Tree.

Table 3.4 summarizes the results on the construction of compaction trees for different values of the probability of single and double error effects felt at the output of the CUT.

S_1	S_2	COMPACTION TREE
0.00	1.00	$O_5 = OR(O_2 , O_3)$ $O_6 = OR(O_1 , O_4)$ $O_7 = AND(O_5 , O_6)$
1.00	0.00	$O_5 = XOR(O_1 , O_2)$ $O_6 = XOR(O_3 , O_4)$ $O_7 = XOR(O_5 , O_6)$
0.66	0.33	$O_5 = XOR(O_1 , O_2)$ $O_6 = XOR(O_3 , O_4)$ $O_7 = XOR(O_5 , O_6)$
0.33	0.66	$O_5 = OR(O_2 , O_3)$ $O_6 = OR(O_1 , O_4)$ $O_7 = AND(O_5 , O_6)$
0.10	0.90	$O_5 = OR(O_2 , O_3)$ $O_6 = OR(O_1 , O_4)$ $O_7 = AND(O_5 , O_6)$
0.90	0.10	$O_5 = XOR(O_1 , O_2)$ $O_6 = XOR(O_3 , O_4)$ $O_7 = XOR(O_5 , O_6)$
0.95	0.05	$O_5 = XOR(O_1 , O_2)$ $O_6 = XOR(O_3 , O_4)$ $O_7 = XOR(O_5 , O_6)$
0.05	0.95	$O_5 = OR(O_2 , O_3)$ $O_6 = OR(O_1 , O_4)$ $O_7 = AND(O_5 , O_6)$

Table 3.4 The Ten-Line Decimal-to-8421 BCD Converter's Compaction Trees for Different Values of S_1 and S_2.

All 56 detectable single stuck-at line faults are injected into the circuit and their effect at the outputs of the circuit is shown as in the previous section in Table 3.3. Depending on the choice of S_1 and S_2, we get different lines selection and different types of gate to use when constructing the corresponding space compactor. The fault-free space compactor response for the 10 input test vectors and for (S_1 , S_2) equalling (0.0 , 1.0), (0.33 , 0.66), (0.1 , 0.9), and (0.05 , 0.95) is the following single 10-bit stream O_7 = (0 0 1 1 1 0 0 0 0 0), while it is the bit-stream O_7 = (0 1 0 0 1 1 0 1 1 0) when (S_1 , S_2) equals (1.0 , 0.0), (0.66 , 0.33), (0.9 , 0.1), and (0.95 ,

0.05), respectively. The fault-free signatures using Syndrome Counter as time compressor are 3 and 5 respectively. For (S_1 , S_2) with the following values (0.0 , 1.0), (0.33 , 0.66), (0.1 , 0.9), and (0.05 , 0.95), by comparing the fault-free streams at the circuit's outputs to the faulty bit streams when injecting all 56 detectable faults, we find that the percentage of fault loss is 0.00 % before compaction. Moreover, 17 out of 56 stuck-at line faults turns out to be undetectable after space compaction and thus, we get about 69.64 % fault coverage or 30.35 % fault loss. For the probability values (1.0 , 0.0), (0.66 , 0.33), (0.9 , 0.1), and (0.95 , 0.05), the compressor is simply a parity tree circuit which is a much better compressor for this circuit (91.07 % fault coverage) as seen previously in the case when stochastic independence of single and double line errors is assumed.

CHAPTER 4

COMPACTION UNDER GENERALIZED MERGEABILITY

This chapter considers space compaction under generalized megeability of response streams of a CUT, where the basic is to select a suitable gate to merge multiple candidate output lines. The selection is essentially based on certain mergeability criterion that also uses the properties of Hamming distance and sequence weight as discussed earlier.

4.1 Space Compaction Under Generalized Mergeability Based on Stochastic Independence of Multiple Line Errors

4.1.1 Gate selection under generalized mergeability

We now consider the case of generalized sequence mergeability under conditions of both stochastic independence and stochastic dependence of multiple line errors occurring at the output of a CUT in the design of the desirable compaction tree. The following definitions are relevant.

Definition 5 If N sequences at the output of a CUT are grouped together having certain Nth 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 that we are working under *bundling constraint*.

Definition 6 The *error multiplicity*, denoted by ν_N, is the number N of simultaneous errors that can occur at the outputs of a CUT or the number of lines at the output of a CUT that be faulty simultaneously.

We next discuss our pertinent results in the case of generalized mergeability of response data output assuming stochastic independence of line errors (multiple).

4.1.2 Generalized mergeability under stochastic independence of multiple line errors

Theorem 19 On the assumption of stochastic independence of errors for a bundled set of N output sequences $\psi_1, \psi_2, \ldots, \psi_N$ each of length L_s at the output of a CUT, the maximum number of possible errors with all possible multiplicities of errors ν_N is $E_{max} = L_s^{[2N-1]}$.

Theorem 20 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences $\psi'_1, \psi'_2, \ldots, \psi'_N$, have Nth order 1-weight and 0-weight, W'_{N1} and W'_{N0}, respectively, and let R be the residue. The total number of faults detected when N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged together by using an N-input AND (NAND) gate is

$$E(\text{AND/NAND}) = [(W'_{N1})_{n1, n2, \ldots, nN}]^{(2N-1)} + (W'_{N0})_{nN} + \sum_{\nu_j} (R_i)_{nj}$$

where $(W'_{N1})_{n1, n2, \ldots, nN}$ represents the number of error multiplicities of 1, 2, ..., N, and $(R_i)_{nj}$ represents the number of errors corresponding to the ith order residue position of $_{nj}$ 0's.

Theorem 21 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences $\psi'_1, \psi'_2, \ldots, \psi'_N$, have Nth order 1-weight and 0-weight, W'_{N1} and W'_{N0}, respectively, and let R be the residue. The total number of faults detected when the N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged by using an N-input OR (NOR) gate is

$$E(\text{OR/NOR}) = [(W'_{N0})_{n1, n2, \ldots, nN}]^{(2N-1)} + (W'_{N1})_{nN} + \sum_{\nu_j} (R_i)_{nj}$$

where $(W'_{N0})_{n1, n2, \ldots, nN}$ represents the number of error multiplicities of 1, 2, ..., N and $(R_i)_{nj}$ represents the number of errors corresponding to the ith order residue position of n_j 0's.

Theorem 22 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences $\psi'_1, \psi'_2, \ldots, \psi'_N$, respectively, have Nth order 1-weight and 0-weight, W'_{N1} and W'_{N0}, respectively, and let R be the residue. The total number of faults detected when the N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged by using an N-input XOR (XNOR) gate is:

$$E(\text{XOR/XNOR}) = [(W'_{N1})_{n1, n3, \ldots, nN-1}]^{(2N/2)} + [(W'_{N0})_{n2, n4, \ldots, nN}]^{(2N/2)} + \sum_{V_j} (R_i)_{nj}$$

$$= [L_s]^{(2N/2)} = [L_s]^{(2N-1)}$$

where $(W'_{N1})_{n1, n3, \ldots, nN-1}$ represents the number of 1-error multiplicities of 1, 3, ... , N-1, $(W'_{N0})_{n2, n4, \ldots, nN}$ represents the number of 0-error multiplicities of 2, 4, ... , N, and $(R_i)_{nj}$ represents the number of errors corresponding to the ith order residue position of n_j 0's, N being even.

Theorem 23 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences be $\psi'_1, \psi'_2, \ldots, \psi'_N$, respectively. Let the Nth order 1-weight be W'_{N1} and Nth order 0-weight be W'_{N0}. For merger of the N sequences, an N-input AND (NAND) gate will be preferable to an N-input OR (NOR) gate for maximizing error detection if $W'_{N1} > W'_{N0}$.

Corollary 23.1 For the merger of N sequences $\psi_1, \psi_2, \ldots, \psi_N$ of length L_s, Nth order 1-weight W'_{N1} and Nth order 0-weight W'_{N0}, an N-input OR (NOR) gate is preferable to an N-input AND (NAND) gate if $W'_{N0} > W'_{N1}$.

Theorem 24 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences be $\psi'_1, \psi'_2, \ldots, \psi'_N$. In addition, let the Nth order 1-weight be W'_{N1} and the Nth order 0-weight be W'_{N0}. For merger of N sequences, an N-input AND(NAND) will be preferable to an N-input XOR (XNOR) gate for maximizing error detection if $W'_{N0} > (W'_{N1} + R) = L_s/2$.

Corollary 24.1 For merger of N sequences $\psi_1, \psi_2, \ldots, \psi_N$ of length L_s, Nth order 1-weight W'_{N1} and Nth order 0-weight W'_{N0}, an N-input XOR(XNOR) gate is preferable to an N-input AND(NAND) gate if $W'_{N1} < (W'_{N0} + R)$, or $W'_{N1} < L_s/2$.

Theorem 25 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences be $\psi'_1, \psi'_2, \ldots, \psi'_N$. In addition, let the Nth order 1-weight be W'_{N1} and Nth order 0-weight be W'_{N0}. For merger of N sequences, an N-input OR (NOR) gate will be preferable to an N-input XOR (XNOR) gate for maximizing error detection if $W'_{N0} > W'_{N1} + R$ or $W'_{N0} > L_s/2$.

Corollary 25.1 For merger of N sequences $\psi_1, \psi_2, \ldots, \psi_N$ of length L_s, Nth order 1-weight W'_{N1} and Nth order 0-weight W'_{N0}, an N-input XOR (XNOR) gate is preferable to an N-input OR (NOR) gate if $W'_{N0} < (W'_{N1} + R)$ or $W'_{N0} < L_s/2$.

Theorem 26 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the derived sequences be $\psi'_1, \psi'_2, \ldots, \psi'_N$. In addition, let the Nth order 1-weight be W'_{N1} and Nth order 0-weight be W'_{N0}. In the extreme case when $R = L_s$ and ($W'_{N1} = W'_{N0} = 0$), the total faults detected by N-input AND (NAND) OR (NOR) and XOR (XNOR) gates are, respectively, as follows:

$$T(\text{AND/NAND}) = T(\text{OR/NOR}) = L_s \text{ and } T(\text{XOR/XNOR}) = L_s^{(2N-1)}.$$

The theorems are very important in the selection of the best subsets of output sequences in maximizing error detection at the CUT output during space compaction. The proofs of these theorems are intentionally avoided for the sake of brevity though the theorems are retained for completeness. Some of the results in the said context can also be found in [53].

For the implementation of the generalized mergeability criteria as outlined in the above results, to construct the space compactor, algorithm was developed. The algorithm was executed on ISCAS 85 combinational benchmark circuits using fault simulation and fault detection programs ATALANTA, FSIM and COMPACTEST. The steps of the algorithm are given next followed by a simple illustrative example.

Algorithm 3

A) Separation Criteria of Sequences for AND(NAND) and OR(NOR) Gates

A1) All original sequences are separated in two lists : ANDlist, Orlist as follows (NAND and NOR gate listings with AND and OR gates, respectively, are omitted in subsequent descriptions of the algorithm).
All sequences that have number of 1s > $L_s/2$ are sent to the ANDlist (where L_s is the length of the sequence).
All sequences that have number of 1s < $L_s/2$ are sent to the ORlist.

A2) Sequences { x_i } where i = { 0,1,2,... } that have number of 1s = $L_s/2$ are sent to the appropriate list where there is a potential grouping. This is done according to the following algorithm:

Let us consider the case where i = 1.

Execute the operation : [x_1] bitwise AND [all sequences of ANDlist], then retain and count the outputs that have number of 1s = $L_s/2$; let "andMatch" be that number.

Execute [x_1 inverted] bitwise AND [all sequences of ORlist inverted], then retain and count the outputs that have number of 1s = $L_s/2$; let "orMatch" be that number.

Now, compare "andMatch" and "orMatch" numbers, send x_1 to the largest list among them, where a potential grouping exists.

Here inversion means one's complement. For example, if x_1 = 011001111, then [x_1 inverted] = 100110000. L_s is the length of a sequence.

B) Stage 1

B1) Obtaining the list of sequences having Best 1's or 0's Grouping :
From the ANDlist, group sequences according to an algorithm that takes into consideration (W_{N1}, N_1), i.e. , in case of AND gate :
W_{N1} : number of matching 1s in the same bit positions of all candidate sequences.
N_1 : number of sequences that have W_{N1} matching of 1's bits in common.
Detailed explanation of this Algorithm is given as follows :

B1.1) W_{N1} overrides N_1. In other words, the group that has the maximum value of W_{N1} is the best group as long as $N_1 \geq 2$.

B1.2) The best grouping between two groups that have the same value of W_{N1} is the one that has the bigger N_1 value.

B2) The same Algorithm can be applied to ORlist but we should replace 1 by 0.

B3) Mergeability Criteria :

Assume the following :
W_{N1} : Number of matching 1s in the same bit positions of all candidate sequences.
W_{N0} : Number of matching 0s in the same bit positions of all candidate sequences.

B3.1) For $W_{N1} > 0.6L_s$: the best grouping of sequences obtained from B1) are merged together with an AND gate only. The resulting output is sent back to the AND sub list.

B3.2) For $0.5L_s \leq W_{N1} \leq 0.6L_s$: the best grouping of sequences obtained from B1) are merged together with AND and XOR gates (XNOR is omitted from discussions). The resulting output is sent either to AND, OR, or XOR (when $L_s/2$ case exists)

sub lists. All the resulting trees of gates are found by using a recursive algorithm.

B3.3) For $W_{N0} > 0.6L_s$: the best grouping of sequences obtained from B2) are merged together with an OR gate only. The resulting output is sent back to the OR sub list.

B3.4) For $0.5L_s \leq W_{N0} \leq 0.6L_s$: the best grouping of sequences obtained from B1) are merged together with OR and XOR gates. The resulting output is sent either to AND, OR, or XOR (when $L_s/2$ case exists) sub lists. All the resulting trees of gates are found by using a recursive algorithm.

B4) Sorting the output sequence after merging with an XOR gate :

If the output sequence has :

Number of 1s > $L_s/2$: the sequence is sent to AND sub list of candidates.

Number of 1s < $L_s/2$: the sequence is sent to OR sub-list of candidates.

Number of 1s = $L_s/2$: the sequence is sent either to AND or OR sub-lists according to the algorithm described in A2).Stage

C) Two and Intermediate Stages

Repeat the same steps as in Stage A.

Intermediate stages exist as long as there are still sequences which can be grouped by either AND, AND (XOR), OR, OR (XOR) gates.

D) Last Stage (XOR Processing)

The obtained sequences that cannot be grouped by AND or OR gates are merged together by XOR gate. At this point, we get the last output sequence of the whole list of sequences.

The following rules are important for the clarification of the first algorithm.

Rule :

Assume we have two 1's grouping of sequences with (W_{N1},N_1) and (W'_{N1},N_1').

The following rule is applied to obtain the optimal solution (best grouping out of both of them) :

Weight "W" takes advantage over (overrides) number of sequences "N". In other words, the group that has the maximum value of W is the best solution.

If $W_{N1} = W'_{N1}$, our optimal solution is the group that has the maximum value of N.

If $W_{N1} = W'_{N1}$ and $N_1 = N_1'$, our optimal solution is either one.

This rule can be generalized for several groups of sequences, i.e., N sequences. Consider the following sequences : 0, 1, 2, 3, 4, ..., N. G_k is the group of sequences which has K sequences ($k < N$) that have matching of 1's bits, and G_i is the group of sequences which has i sequences ($i < N$) that have matching of 1's bits. Three cases can be considered as follows :

If $W_{k1} > W_{i1} \wedge$ group G_k is better than group G_i.

If $W_{k1} < W_{i1} \wedge$ group G_i is better than group G_k.

If $W_{k1} = W_{i1}$ and $N_k \geq N_i \wedge$ group G_K is better than group G_i.

The same procedure can be followed for matching 0s of bits.

Example 6 Find the FIRST best gate selection for the following six bit streams.

1 1 1 1 1 1 0 0 1 ∧ seq 0

1 1 1 1 1 1 0 1 0 ∧ seq 1

1 1 1 1 1 1 0 1 0 ∧ seq 2

1 1 1 1 1 1 0 1 1 ∧ seq 3

1 1 0 0 0 0 0 1 0 ∧ seq 4

1 1 0 0 0 0 1 0 1 ∧ seq 5

1 1 0 1 0 1 0 1 0 ∧ seq 6

In the above example, we have 3 groups of 1s and 5 groups of 0s to be considered, where $L_s = 9$ and ($0.6L_s = 5.4$). All possibilities of 1's groupings are :

First 1's group : $W_{71} = 2 > 0.6L_s$, $N_1 = 7$ (seq 0, ..., seq 6)

Second 1's group : $W_{41} = 6 < 0.6L_s$, $N_2 = 4$ (seq 0, ... , seq 3)

Third 1's group : $W_{31} = 7 > 0.6L_s$, $N_3 = 3$ (seq 1, ..., seq 3)

All possibilities of 0's groupings are :

First 0's group : $W_{40} = 1 < 0.6L_s$, $N_1 = 4$ (seq 0, ..., seq 3)

Second 0's group : $W_{20} = 4 < 0.6L_s$, $N_2 = 2$ (seq 4, seq 5)

Third 0's group : $W_{20} = 2 < 0.6L_s$, $N_3 = 2$ (seq 1, seq 2)

Fourth 0's group : $W_{20} = 4 < 0.6L_s$, $N_4 = 2$ (seq 4, seq 6)

Fifth 0's group : $W_{30} = 2 < 0.6L_s$, $N_5 = 3$ (seq 4, seq 5, seq 6)

From all the above possibilities, according to the highest weight, only two groups are to be considered:

- Second 1's group: $W_{41} = 6 > 0.6L_s$, $N_1 = 4$ (seq 1, seq 2, seq 3) and
- Third 1's group: $W_{31} = 7 > 0.6L_s$, $N_3 = 3$ (seq 0, ..., seq 6).

Solution : By applying the general rule and since ($W_{31} = 7$) > ($W_{41} = 6$), we select the third 1's group (seq 1, seq 2, seq 3) as the BEST group to be merged by an AND gate.

Implementation of the Selection of the Best Group Sub-Algorithm

In order to understand the process to obtain the "Best Group", the following definitions, categories of grouping and stages of processing are provided.

Definition and initialization of parameters :

maxW : a variable which denotes the maximum number of matching bits (can be 0 or 1 according to the kind of grouping taken into consideration) in the same bit positions obtained so far in the process of getting the BEST GROUP.

max N : a variable which denotes the maximum number of sequences that have maxW matching of (1s or 0s according to the kind of grouping taken into consideration) bits in common obtained so far in the process of getting the BEST GROUP.

As a first step, we initialize both maxW and maxN to zero.

Stages of processing :

At the first stage, "Selecting the Best Group" sub-algorithm (explained later) is applied by starting with seq0 going through each of : seq1, seq2, ... , seqN.

At the second stage, "Selecting the Best Group" sub-algorithm is applied by starting with seq1 going through each of the following sequences : seq2, seq3, ... , seqN. At the ith stage, "Selecting the Best Group" algorithm is applied by starting with seq(i-1) going through each of the following sequences : seqi, seq(i+1), ... , seqN.

The last stage of processing during the execution of the algorithm starts at seq(N-1) going through seqN.

Categories of groupings :

The various kinds of groupings are defined and classified as follows :

Cat [1] Bad Grouping :

If resulting W < maxW, it means that we must exclude this sequence from grouping under the same stage.

Cat [2] Good Grouping :

If resulting W > maxW or [resultingW = maxW and resultingN ≥ maxN], it means that we continue processing in the same stage.

Cat [3] Better Grouping :

It has the same conditions as in Good Grouping. In addition, it is the last good grouping found in a stage. The grouping is considered as a candidate for the "best grouping".

Cat [4] Best Grouping :

It has the maximum of W and N parameters found after going through all the stages and so it is the best group of all the stages.

Sub-Algorithm to select the "BEST GROUP" :

In stage 1, we start processing the operation with seq0.

1) To obtain the matching of 1s in each bit position we do : (seq0 AND seq1) = R_1 which has (W_{N1}, N_1). Now we compare W_{N1} (resultingW) with maxW and if needed (when W_{N1} = maxW), we compare N_1 (resultingN) with maxN and record the highest of both, i.e., MAX[W_{N1},maxW]. We put it in maxW, and we put MAX[N_1,maxN] in maxN.

If resultingW < maxW then (seq0 , seq1) is a bad grouping according to Cat [1], and we restart 1) for seq0 and seq2.

If (seq0, seq1) is a good grouping according to Cat [2], we can say that (seq0,seq1) is the best group so far. Then we go to 2).

2) Now we execute : (R_1 AND seq2) = R_2 which has (W_2, N_2). Then we compare W_2 (resultingW) with maxW and if needed (when W_{N1} = maxW), we compare N_2 (resultingN) with maxN and record the highest of both, i.e. , MAX[W_2,maxW]. We put it in maxW and we put MAX[N_2,maxN] in maxN.

If resultingW < maxW then (seq0,seq1,seq2) is a bad grouping according to Cat [1] and we restart 2) for R_1 and seq3.

If (R_1, seq2) is a good grouping according to Cat [2], we can say that (R_1, seq2) is the best group so far. Then we go to 3).

3) Now we execute : (R_2 AND seq3) = R_3 which has (W_3, N_3). Then we compare W_3 (resultingW) with maxW (obtained in step 2) and if needed (when W_3 = maxW) we compare N_3 (resultingN) with maxN (obtained in step 2) and record the highest of

both, i.e. MAX[W_2, maxW]. We put it in maxW and we put MAX[N_2, maxN] in maxN.

- If resultingW < maxW (seq0, seq1, seq2, seq3) is a bad grouping according to Cat [1], we restart 3) for R_2 and seq4.
- If (R_2, seq3) is a good grouping according to Cat [2], we can say that (R_2, seq3) is the best group so far. Then we go to the next step.

4) We repeat the above steps until we exhaust all sequences starting with sequence 0. Whenever we find the "better grouping" according to Cat [3] in the first stage (steps : 1, 2, 3), we compare its resulting parameters to the first grouping of the second stage which starts with sequence 1.

5) We repeat step 4) for all the existing stages. At the last stage, we obtain the *optimal* grouping which is the "best grouping for the whole list of sequences" according to Cat [4]. This step is achieved using a recursive method.

4.1.3 Mergeability criteria

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

- If $W_{N1} \geq L_s/2$, the obtained group sequences are merged together with an AND (NAND) gate only.
- If $W_{N0} > L_s/2$, the obtained group sequences are merged together with an OR (NOR) gate only.
- If $W_{N1} < L_s/2$ and $W_{N0} < L_s/2$, 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.

4.2 Space Compaction Under Generalized Mergeability Based on Stochastic Dependence of Multiple Line Errors

In this section, we will present generalized mergeability criteria for merging an arbitrary number of response data outputs of the CUT under stochastic dependence of multiple line errors [7]. In order to do that, we first extend the definition of second-order detectable error probability estimate E_2 as defined earlier to cover the case of N sequences. We state the following obvious theorems without proof.

Theorem 27 The second order detectable error probability estimate E_2 when two sequences Ψ_1 and Ψ_2 of length Ls are merged by using an AND (NAND) gate is

$$E_2\ (AND / NAND) = S_1 \frac{H_s + 2W_{21}}{2L_s} + S_2 \frac{W_{20} + W_{21}}{L_s}$$

where H_s is the Hamming distance between the two sequences.

Theorem 28 The second order detectable error probability estimate when two sequences Ψ_1 and Ψ_2 of length L_s are merged by using an OR (NOR) gate is

$$E_2\ (OR / NOR) = S_1 \frac{H_s + 2W_{20}}{2L_s} + S_2 \frac{W_{20} + W_{21}}{L_s}$$

H_S being the Hamming distance between the sequences.

Theorem 29 The second order detectable error probability estimate E_2 when two sequences Ψ_1 and Ψ_2 of length Ls are merged by using an XOR (XNOR) gate is

E_2 (XOR/XNOR) = S_1.

4.2.1 Derivation of the n^{th} order detectable error probability estimate for individual gates

In this section, we will determine the Nth order detectable error probability estimate E_N for AND (NAND), OR (NOR) and XOR (XNOR) gates. They are derived for the case when N is odd. The expressions for E_Ns will be very much similar when N is even.

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

Let S_i , i = 1, 2, ...,N be the probability of i-line errors. Realistically, one can assume that $S_1 > S_2 > ... > S_N$ and $S_1 + S_2 + ... + S_N = 1$. Let R be the residue, that is, 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 $\binom{N}{i}$ simply by the symbol C_i or ^NC_i.

Example 7 Consider the following 5 sequences of length $L_s = 12$ each.

ψ_1 = 1 1 1 1 0 0 0 1 0 1 0 1

ψ_2 = 1 1 1 1 0 0 1 0 1 0 0 0

ψ_3 = 1 1 1 1 0 0 0 0 0 1 1 0

ψ_4 = 1 1 1 1 0 0 1 1 0 0 0 1

ψ_5 = 1 1 1 1 0 0 0 1 1 1 0 0

$W_{51} = 4$ $W_{50} = 2$ R

Vertical 1-weights

2 3 2 3 1 2

In this example, the 5^{th} order 1-weight is $W_{51} = 4$ and the 5^{th} order 0-weight is $W_{50} = 2$. The residue R is equal to 6 since there are 6 bit positions in which at least two sequences differ. The vertical 1-weights are $A_1 = 1$, $A_2 = 3$, $A_3 = 2$, $A_4 = 0$, and $A_5 = 4$.

Theorem 30 The Nth order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, ..., \Psi_N$ are merged by using an N-input AND (NAND) gate is

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

Proof:

$$E_N\ (AND/NAND) = \frac{S_1}{C_1^N \times L_s}\left[C_1^N \times W_{N1} + A_{N-1}\right] + \frac{S_2}{C_2^N \times L_s}\left[C_2^N \times W_{N1} + A_{N-2}\right] + \cdots +$$
$$\frac{S_{N-1}}{C_{N-1}^N \times L_s}\left[C_{N-1}^N \times W_{N1} + A_{N-(N-1)}\right] + \frac{S_N}{C_N^N \times L_s}\left[C_N^N \times W_{N1} + C_N^N \times W_{N0}\right]$$

$$E_N\ (AND/NAND) = \frac{S_1}{C_1^N \times L_s}\left[C_1^N \times W_{N1}\right] + \frac{S_2}{C_2^N \times L_s}\left[C_2^N \times W_{N1}\right] + \cdots +$$
$$\frac{S_{N-1}}{C_{N-1}^N \times L_s}\left[C_{N-1}^N \times W_{N1}\right] + \frac{S_N}{C_N^N \times L_s}\left[C_N^N \times W_{N1}\right] +$$
$$\frac{S_1}{C_1^N \times L_s}\left[A_{N-1}\right] + \frac{S_2}{C_2^N \times L_s}\left[A_{N-2}\right] + \cdots +$$
$$\frac{S_{N-1}}{C_{N-1}^N \times L_s}\left[A_1\right] + \frac{S_1}{C_N^N \times L_s}\left[C_N^N \times W_{N0}\right] =$$

$$E_N\ (AND/NAND) = \frac{W_{N1}}{L_s}\left(S_1 + S_2 + \cdots + S_{N-1} + S_N\right) +$$
$$\frac{1}{L_s} \times \left[\frac{S_1}{C_1^N}\left[A_{N-1}\right] + \frac{S_2}{C_2^N}\left[A_{N-2}\right] + \cdots + \right.$$
$$\left. \frac{S_{N-1}}{C_{N-1}^N}\left[A_1\right] + \frac{S_N}{C_N^N}\left[C_N^N \times W_{N0}\right] \right]$$

$$E_N\ (AND/NAND) = \frac{W_{N1}}{L_s} + \frac{1}{L_s} \times \left[\frac{S_1}{C_1^N}\left[A_{N-1}\right] + \frac{S_2}{C_2^N}\left[A_{N-2}\right] + \cdots + \right.$$
$$\left. \frac{S_{N-1}}{C_{N-1}^N}\left[A_1\right] + S_N\left[W_{N0}\right] \right] = \frac{W_{N1}}{L_s} + \frac{1}{L_s} \times \left[\sum_{i=1}^{N-1} \frac{S_i}{C_i}[A_i] + S_N\ W_{N0} \right]$$

$$E_N\ (AND/NAND) = \frac{W_{N1}}{L_s} + \frac{1}{L_s}\left[\ \frac{1}{C_1}\left(\ S_1\ A_{N-1} + S_{N-1}\ A_1\ \right) + \frac{1}{C_2}\left(\ S_2\ A_{N-2} + S_{N-2}\ A_2\ \right) + \cdots + S_N\ W_{N0}\ \right].$$

Theorem 31 The Nth order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, ..., \Psi_N$ are merged by using an N-input OR (NOR) gate is

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

Proof:

$$E_N(O/N) = \frac{S_1}{C_1^N \times L_s}\left[C_1^N \times W_{N0} + A_1\right] + \frac{S_2}{C_2^N \times L_s}\left[C_2^N \times W_{N0} + A_2\right] + \cdots + \frac{S_{N-1}}{C_{N-1}^N \times L_s}\left[C_{N-1}^N \times W_{N0} + A_{N-1}\right] + \frac{S_N}{C_N^N \times L_s}\left[C_N^N \times W_{N0} + C_N^N \times W_{N1}\right]$$

$$E_N(O/N) = \frac{W_{N0}}{L_s}\left(S_1 + S_2 + \cdots + S_{N-1} + S_N\right) + \frac{1}{L_s} \times \left[\ \frac{S_1}{C_1^N}[A_1] + \frac{S_2}{C_2^N}[A_2] + \cdots + \frac{S_{N-1}}{C_{N-1}^N}[A_{N-1}] + \frac{S_N}{C_N^N}\left[C_N^N \times W_{N1}\right]\ \right]$$

$$E_N(O/N) = \frac{W_{N0}}{L_s} + \frac{1}{L_s} \times \left[\ \frac{S_1}{C_1^N}[A_1] + \frac{S_2}{C_2^N}[A_2] + \cdots + \frac{S_{N-1}}{C_{N-1}^N}[A_{N-1}] + S_N[W_{N1}]\ \right]$$

$$= \frac{W_{N0}}{L_s} + \frac{1}{L_s} \times \left[\ \sum_{i=1}^{N-1} \frac{S_i}{C_i}[A_i] + S_N\ W_{N1}\ \right]$$

$$E_N(O/N) = \frac{W_{N0}}{L_s} + \frac{1}{L_s}\left[\frac{1}{C_1}\left(S_1 A_1 + S_{N-1} A_{N-1} \right) + \frac{1}{C_2}\left(S_2 A_2 + S_{N-2} A_{N-2} \right) + \cdots + S_N W_{N1} \right].$$

Theorem 32 The Nth order detectable error probability estimate when N sequences $\Psi_1, \Psi_2, ..., \Psi_N$ are merged by using an N-input XOR (XNOR) gate is:

$$E_N(XOR/XNOR) = S_1 + S_3 + S_5 + \cdots + S_N.$$

Theorem 33 In general, for two N-input gates G_1 and G_2, G_1 is preferable to G_2 if and only if: $E_N(G_1) > E_N(G_2)$.

Theorems 25 and 26 are obvious and proofs are omitted.

A heuristic approach has been adopted and implemented in the following algorithm in order to get results within an acceptable CPU time. The heuristic approach is very useful in this case since a closed-form expression of the n^{th} order detectable error probability estimates can be computationally intensive.

Algorithm 4

A1) From all the output sequences, select a group of N_0 sequences having largest the Nth order 0-weight. Then select another group of N_1 sequences having the largest Nth order 1-weight.

A2) Choice between selected 1-weight and 0-weight groups

In order to make a choice between the largest group based on 1's grouping or 0's grouping, let us consider the following example.

Suppose we obtained from step A1) a 0's grouping and a 1's grouping which have as weights and numbers of sequences (W_{N0}, N_0) and (W_{N1}, N_1), respectively. The selection of the best group between both groups is done according to the following rule :

a. W (weight) overrides N (number of sequences) which means the group that has the maximum value of W is the best solution.
b. If $W_{N0} = W_{N1}$, the best group is the group that has the largest value of N.
c. If $W_{N0} = W_{N1}$ and $N_0 = N_1$, the best group is either one. In the implementation, the Nth order 1-weight is chosen.

A3) Choice between (AND/NAND, OR/NOR) gates

A3.1) If 1-weight is chosen

- Compute W_{N1} and R/N :
- If $W_{N1} > R/N$, then AND gate is chosen over OR, but it still has to be compared with XOR.
- If $W_{N1} \leq R/N$, then compute both the formulas :

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

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

- Then compare them and :

  *If E_N[AND/NAND] > E_N[OR/NOR], then AND/NAND gate is chosen over OR/NOR gate, but it still has to be compared with XOR/XNOR gate.

  *If E_N[AND/NAND] < E_N[OR/NOR] then OR/NOR gate is chosen over AND/NAND gate, but it still has to be compared with XOR/XNOR gate.

  *If E_N[AND/NAND] = E_N[OR/NOR] then either AND/NAND or OR/NOR gate can be chosen, but the chosen gate still has to be compared with XOR/XNOR gate. In the implementation, AND gate was chosen in this situation.

A3.2) If 0-weight is chosen

- Compute W_{N0} and R/N :
- If W_{N0} > R/N, then OR/NOR gate is chosen over AND/NAND gate, but it still has to be compared with XOR/XNOR gate.
- If $W_{N0} \leq$ R/N, then compute both the formulas :

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

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

- Then compare them and :

  *If E_N[AND/NAND] > E_N[OR/NOR] then AND/NAND gate is chosen over OR/NOR gate, but it still has to be compared with XOR/XNOR gate.

  *If E_N[AND/NAND] < E_N[OR/NOR] then OR/NOR gate is chosen over AND/NAND gate, but it still has to be compared with XOR/XNOR gate.

  *If E_N[AND/NAND] = E_N[OR/NOR] then either AND/NAND or OR/NOR gate can be chosen, but the chosen gate still has to be compared with XOR/XNOR gate. In the implementation, AND gate was chosen in this situation.

A4) Choice between (above chosen gate, XOR/XNOR) gates

A4.1) Case 1 (AND/NAND, XOR/XNOR) :

a) Compute : $A = W_{N1}/L_s - (S_1 + S_3 + \ldots + S_N)$ for N odd.

$A = W_{N1}/L_s - (S_1 + S_3 + \ldots + S_{N-1})$ for N even.

b) If $A \geq 0$ then use AND/NAND gate (non-exhaustive approach is fulfilled).

c) if $A < 0$ then

$$\text{Compute B} = \frac{1}{L_s} \times \left[\sum_{i=1}^{N-1} \frac{S_i}{C_i}[A_i] + S_N W_{N0} \right]$$

then compute (A+B) = C,

d) If $C > 0$ then use AND/NAND gate.

e) If $C < 0$ then use XOR/XNOR gate.

f) If $C = 0$ then use either AND/NAND or XOR/XNOR gate. In the implementation XOR gate was chosen in this situation.

A4.2) Case 2 (OR/NOR, XOR/XNOR) :

a) Compute $A = W_{N0}/L_s - (S_1 + S_3 + \ldots + S_N)$ for N odd.

$A = W_{N0}/L_s - (S_1 + S_3 + \ldots + S_{N-1})$ for N even.

b) if $A \geq 0$ then use OR/NOR gate (non-exhaustive approach is fulfilled).

c) if $A < 0$ then

$$\text{Compute B} = \frac{1}{L_s} \times \left[\sum_{i=1}^{N-1} \frac{S_i}{C_i}[A_i] + S_N W_{N1} \right]$$

then compute (A+B) = C,

d) If $C > 0$ then use OR/NOR gate.

e) If $C \leq 0$ then use XOR/XNOR gate.

f) If $C = 0$ then use either OR/NOR or XOR/XNOR gate. In the implementation XOR gate was chosen in this situation.

A5) Apply the above procedure to the merging sequence and all the remaining sequences until we end up with only one sequence at the output.

CHAPTER 5

COMPACTION UNDER OPTIMAL GENERALIZED MERGEABILITY

We develop here techniques that deal with the general problem of designing space-efficient support hardware for built-in self-testing (BIST) of VLSI circuits and digital core based system using knowledge of compact test sets. The techniques developed for the purpose are primarily based on identifying certain inherent properties of the test responses of the CUT, together with the knowledge of their failure probabilities. With that objective in view, the optimal mergeability criteria of output sequences are first developed taking advantage of some well-known concepts of conventional switching theory, *viz.* those of cover table and frequency ordering [66, 67], as commonly utilized in the minimization of switching functions, together with notions of Hamming distance, sequence weights, and derived sequences as used by the authors earlier in sequence characterization; later, the effects of failure probabilities on the optimal mergeability criteria [62] of output sequences are analyzed using all the aforesaid concepts but based on a new measure of failure probability introduced in this chapter.

5.1 Optimal Generalized Mergeability Under Stochastic Independence of Multiple Line Errors

More details are given here, as a basis for understanding the new techniques.

Definition 7 Given a set of response sequences at the output of a CUT, a table can be readily formed in the following manner. Designate the different output sequences as $\alpha_1, \alpha_2, \ldots, \alpha_k$ where k represents the total number of sequences, and the individual bit positions of the different sequences as $\mu_1, \mu_2, \ldots, \mu_{L_s}$ (starting from left), L_s being the total

number of bits in each sequence or sequence length. Write all the bit positions in a row, and underneath each of the bit positions make an entry of the set of sequences if their values in this bit position is a 1, in a columnar form. Each of the columns headed by a specific bit position will then give a collection of the sequences of which the bit value is a 1 in this particular bit position. Such a table will be called a 1-cover-table.

Definition 8 Given a set of response sequences at the output of a CUT, a table can be readily formed in the following manner. Designate the different output sequences as $\alpha_1, \alpha_2, \ldots, \alpha_k$ where k represents the total number of sequences, and the individual bit positions of the different sequences as $\mu_1, \mu_2, \ldots, \mu_{Ls}$ (starting from left), L_s being the total number of bits in each sequence or sequence length. Write all the bit positions in a row, and underneath each of the bit positions make an entry of the set of sequences if their values in this bit position is a 0, in a columnar form. Each of the columns headed by a specific bit position will then give a collection of the sequences of which the bit value is a 0 in this particular bit position. Such a table will be called a 0-cover-table.

Example 8 Consider the following 8 sequences : α_1= 1101 1100 ; α_2= 1101 0110 ; α_3= 1111 1101 ; α_4= 0111 1001 ; α_5= 1111 1111 ; α_6= 0000 0001 ; α_7= 1011 1111 ; α_8= 1111 1101 . The 1-cover table and 0-cover table are shown in Table 5.1 and Table 5.2, respectively.

μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8
α_1	α_1	α_3	α_1	α_1	α_1	α_2	α_3
α_2	α_2	α_4	α_2	α_3	α_2	α_5	α_4
α_3	α_3	α_5	α_3	α_4	α_3	α_7	α_5
α_5	α_4	α_7	α_4	α_5	α_5		α_6
α_7	α_5	α_8	α_5	α_7	α_7		α_7
α_8	α_8		α_7	α_8	α_8		α_8
			α_8				

Table 5.1 1-Cover-Table.

μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8
α_4	α_6	α_1	α_6	α_2	α_4	α_1	α_1
α_6	α_7	α_2		α_6	α_6	α_3	α_2
		α_6				α_4	
						α_6	
						α_8	

Table 5.2 0-Cover-Table.

Definition 9 For any two distinct sequences α_i and α_j appearing in a cover table (1-cover table or 0-cover table), if the sequence α_i appears in more columns in the table than the sequence α_j, then an ordering of the sequences can be made as $\alpha_i \geq \alpha_j$. If both the sequences α_i and α_j appear in the same number of columns in the cover table, the ordering can be made either as $\alpha_i \geq \alpha_j$ or $\alpha_j \geq \alpha_i$. This kind of ordering ($\geq$) that can be established amongst different sequences in a cover table is called the frequency ordering of the sequences.

Example 9 Consider the 1-cover- table as given in Table 5.1. We find the one-frequency ordering of the sequences $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8$ as follows:

$$\alpha_5 \geq \alpha_3 \geq \alpha_7 \geq \alpha_8 \geq \alpha_1 \geq \alpha_2 \geq \alpha_4 \geq \alpha_6.$$

Consider the 0-cover- table in Table 5.2 now. We find the zero-frequency ordering of the sequences $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8$ as follows:

$$\alpha_6 \geq \alpha_1 \geq \alpha_2 \geq \alpha_4 \geq \alpha_3 \geq \alpha_7 \geq \alpha_8 \geq \alpha_5 .$$

We reiterate the definitions for the following terms again, though they have been discussed in the previous section.

w_{N1} Number of matching 1s in the same bit positions of all candidate sequences.

w_{N0} Number of matching 0s in the same bit positions of all candidate sequences.

R The residue (Hamming distance of multiple sequences).

L_s The length of the sequence.

We now give below our optimal generalized mergeability criteria under conditions of stochastic independence of multiple line errors.

5.1.1 Generalized optimal mergeability criteria

The main idea behind optimally merging a bundled set of sequences by a logic gate (AND/NAND, OR/NOR, or XOR/XNOR) is to find if the sequences satisfy one of the following conditions:

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

We present next the algorithm for carrying through the aforesaid optimal mergeability [103] of test data responses to design our space compression networks at the CUT output.

Algorithm 5

A. Procedure to find the candidate sequences in the original sequences for optimal merger by using AND (NAND) gates.

Step A1 : From the given input test sequence list, excluding the discarded sequences, set up the 1-covertable.

Step A2 : Find the corresponding frequency ordering of the sequences in the 1-cover table.

Step A3 : Find the one_maxgroup sequence, if it exists, in the frequency ordering. For one_maxgroup sequences to exist, $w_{N1} \geq L_s/2$; besides, the number of candidate sequences (S1) in the group should be as large as possible (in a family of sets of sequences with each set having $w_{N1} \geq L_s/2$, the set with the highest cardinality forms the one_maxgroup).

i) If one_maxgroup sequences are found in the frequency ordering, go to Step A4.

ii) Otherwise, go to Step B.

iii) To find one_maxgroup of sequences, go to Subalgorithm 5.1.

Step A4 : Based on the optimal mergeability criteria, optimally merge the candidate sequences in one_maxgroup from Step A3 by using AND (NAND) gate.

Step A5 : Store the resulting output sequence from AND (NAND) operation and discard the merged candidate sequences in the one_maxgroup.

Step A6 : Repeat Steps A1, A2, A3, A4, and A5. Find all other one_maxgroup sequences that can be merged by AND (NAND) gates at the same level until none can be found. Go to Sep B.

B. Procedure to find the candidate sequences in the original test response sequences for optimal merger by using OR (NOR) gates.

Step B1 : From the given input test sequence list, excluding the discarded sequences, set up the 0-covertable.

Step B2 : Find the corresponding frequency ordering of the sequences in the 0-cover table.

Step B3 : Find the zero_maxgroup sequences, if it exists, in the frequency ordering. For zero_maxgroup sequences to exist, $w_{N0} > L_s/2$; besides, the number of candidate sequences (S1) in the group should be as large as possible (in a family of sets of sequences with each set having $w_{N1} > L_s/2$, the set with the highest cardinality forms the zero_maxgroup).

i) If zero_maxgroup sequences are found in the frequency ordering, go to Step B4.

ii) Otherwise, go to Step C.

iii) To find zero_maxgroup of sequences, go to sub-algorithm 5.2.

Step B4 : Based on the optimal mergeability criteria, optimally merge the candidate sequences in zero_maxgroup from Step B3 by using Or (NOR) gate.

Step B5 : Store the resulting output sequence from OR (NOR) operation and discard the merged candidate sequences in the zero_maxgroup.

Step B6 : Repeat Steps B1, B2, B3, B4, and B5. Find all other zero_maxgroup sequences that can be merged by OR (NOR) gates at the same level until none can be found. Go to Sep C.

C. Optimal merger of the remaining sequences using XOR (XNOR) gates.

In this step, the remaining sequences which cannot be optimally merged by either AND (NAND) or OR (NOR) gates at the same level are optimally merged by XOR (XNOR) gates.

Store the resulting output sequence of XOR (XNOR) operation, and discard the merged remaining sequences.

D. Set up the new original input sequence list composed of the output sequences obtained from the preceding level.

Repeat Steps A through C until a single output sequence is obtained.

In finding the maxgroup of sequences from the frequency ordering, we first introduce the following notations.

- N1 : Number of matching 1s in the same bit position of all candidate sequences one_maxgroup.
- N0 : Number of matching 0s in the same bit position of all candidate sequences zero_maxgroup.
- S1 : Number of candidate sequences when N1 > $L_s/2$.
- S0 : Number of candidate sequences when N0 > $L_s/2$.
- $\alpha 1_m$: Number of 1s in the sequence α_m .
- $\alpha 0_m$: Number of 0s in the sequence α_m .

We now define the following terms.

- Good group: A group of candidate sequences where N1 (N0) > $L_s/2$ and S1 (S0) ≥ 2.
- Better group: Same conditions apply as in good group. In addition, it is the best one among the good groups in a stage.
- one_maxgroup (zero_maxgroup): A group of candidate sequences where N1 (N0) > $L_s/2$, and S1 (S0) as many as possible.

Now to find the one_maxgroup and zero_maxgroup of sequences from the frequency ordering, use sub-algorithm 5.1 and sub-algorithm 5.2 given below.

Sub-Algorithm 5.1

Finding one_maxgroup sequences from the frequency ordering obtained from 1-cover table.

Step 1 : Obtain the candidate group in which 1's number of each sequence is greater than or equal to $L_s/2$; the sequences with 1's number less than $L_s/2$ are not to be included in the candidate group.

Step 2 : Select the candidate sequences.

2.1. Start from α_0, check sequences α_0 and α_1.

2.1.1. If $\alpha 1_0 != \alpha 1_1$, check sequences α_1 and α_2.

If $\alpha 1_1 = \alpha 1_2$, go to sequences α_1 and α_3, and so on, until find $\alpha 1_1 != \alpha 1_i$, break.

Then the candidate sequence set is ($\alpha_0, \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{i-1}$).

2.1.2. If $\alpha 1_0 = \alpha 1_1$, check α_0 and α_2 and so on, until find $\alpha 1_0 != \alpha 1_d$.

Then the candidate sequence set is ($\alpha_0, \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{d-1}$).

Step 3 : From the candidate sequences ($\alpha_0, \alpha_1, \ldots, \alpha_e$), get a better group.

3.1. Start from sequence α_0. In order to obtain the 1's number in each bit position, do bitwise AND operation.

R1 = (α_0 AND α_1);

N1 = 1's number in R1;

Compare N1 with $L_s/2$:

3.1.1. If N1 $\geq L_s/2$, S1 = 2, go to Step 3.2.

3.1.2. If N1 $< L_s/2$, exclude α_1 from the group at the same stag and restart from α_0 and α_2 until find R1 = (α_0 AND α_a), where N1 $\geq L_s/2$, S1 = 2. Stop and then go to Step 3.2.

3.2. Execute R2 = (R1 AND α_2) (if from Step 3.1.2, then R2 = R1 AND α_{a+1}), count the 1's number in R2, i.e. N1' if N1' $\geq L_s/2$, then N1 = N1' and S1 = 3, go to Step 3.3.

3.2.1. If N1' $< L_s/2$, exclude α_2 (or α_{a+1}) from the group under the same stag and restart from R1 and α_3 (or α_{a+2}) until find R2 = (R1 AND α_b), where N1' $\geq L_s/2$. Stop. Then N1 = N1' and S1 = 3. Go to Step 3.3.

3.3. Execute R3 = (R2 AND α_3) (if from Step 3.2.2, then R3 = R2 AND α_{b+1}), count the 1's number in R2, i.e. N1'.

3.3.1. If N1' $\geq L_s/2$, then N1 = N1' and S1 = 4, go to Step 3.4.

3.3.2. If N1' $< L_s/2$, restart for R2 and α_4 (or α_{b+2}), till we find R3 = (R1 AND α_c), where N1' $\geq L_s/2$. Stop.

Then N1 = N1' and S1 = 4. Go to Step 3.4.

3.4. Repeat the above steps until all candidate sequences are exhausted starting with α_0, and whenever the good group in the first stag is found (Steps 3.1 through 3.3).

3.5. Execute the same algorithm, starting with α_1 and get other good groups.

3.6. Repeat until end with the sequence α_e.

3.7. From the above steps, compile a set of good groups. Select a better group from these good groups, where S1 has the maximum value.

Step 4 : From Step 3, get a better group with N1 ($\geq L_s/2$), S1, and R (the result of bitwise AND operation of all S1 sequences) among the candidate sequences ($\alpha_0, \alpha_1, \ldots, \alpha_e$). Now select the new candidate sequences between R and ($\alpha_{e+1}, \ldots, \alpha_m$) using Step 2.1.1, and get the new candidate sequences ($R, \alpha_{e+1}, \ldots, \alpha_l$), $l \leq$ m.

Step 5 : Find a better group among candidate sequences ($R, \alpha_{e+1}, \ldots, \alpha_l$), $l \leq$ m.

5.1. Execute R1 = (R AND α_{e+1}), get N1' (1's number in R1).

5.1.1. If N1' $\geq L_s/2$, N1 = N1', S1++ , go to Step 5.2.

5.1.2. If N1' $< L_s/2$, exclude α_{e+1} from the group under the same stag and restart from α_{e+2}, till we find R1 = R AND α_j (j

$\leq l$), where N1' $\geq L_s/2$. Stop. Then N1 = N1' and S1++ . Go to Step 5.2.

5.2. Execute R2 = R1 AND α_{e+2} (if from Step 5.1.2, then R2 = R1 AND α_{j+1}), count the 1's number in R2, i.e. N1'.

5.2.1. If N1' $\geq L_s/2$, then N1 = N1' and S1++ ; go to Step 5.3.

5.2.2. If N1' $< L_s/2$, restart for R1 and α_{e+3} (or α_{j+2}), till we find R2 = R1 AND α_k , where N1' $\geq L_s/2$. Stop.

Then = N1' and S1++. Go to Step 5.3.

5.3. Repeat the above steps (Steps 5.1 and 5.2) until we exhaust all candidate sequences are exhausted starting with α_{e+1}, and resulting better group is obtained, with N1 $\geq L_s/2$, S1, and R (the result of bitwise AND operation of all S1 sequences).

Step 6 : Select the new candidate sequences among R and the remaining sequences in the candidate group, as in Step 4.
Repeat Step 5.

Step 7 : Repeat Step 6, until gone through all the sequences in the candidate group and found a better group. In the end, the better group obtained is a one_maxgroup in which N1 $\geq L_s/2$, and S1 as large as possible.

Sub-Algorithm 5.2

Finding the zero_maxgroup sequences from the frequency ordering obtained from the 0-cover table.

To get a zero_maxgroup from the frequency ordering as obtained from the 0-cover table, use bitwise OR operation instead of AND operation, and replace 1 by 0 in sub-algorithm 5.1. We then get the zero_maxgroup with N0 > $L_s/2$, and S0 as large as possible.

5.2 Optimal Generalized Mergeability Under Stochastic Dependence of Multiple Line Errors

We consider the effects of error at the CUT output by introducing a new measure of probability estimate, called the *missed error probability estimate*, [103] which is based on the second-order error probability estimate $\varepsilon_2 = S_1(R_1/2L_S) + S_2(R_2/L_S)$, by Li and Robinson [41] and defined for a two-input logic function, given two input sequences (to be designated, following our usual convention, as the second-order missed error probability estimate, denoted by ∂_2) as follows:

Definition 10 The second-order missed error probability estimate for a two-input logic function, provided two input sequences of length L_s, is given as: $\partial_2 = t_1(\mu_1/2L_S) + t_2(\mu_2/L_S)$, where t_1 is the probability of single error effect not being felt at the output of the CUT, t_2 is the probability of double error effect not being felt at the output of the CUT, μ_1 is the number of single line errors missed at the output of gate i, if gate i is used and μ_2 is the number of double line errors missed at the output of gate i, if gate i is used.

The definition can be generalized to cover N sequences for their merger by N-input logic gates of types AND (NAND), OR (NOR), and XOR (XNOR) exactly the same way as was done using the Nth-order detectable error probability estimates [3, 62].

Let $\psi_1, \psi_2, \ldots, \psi_N$ be the N sequences at the output of a CUT of length L_s, having the Nth-order 1-weight and 0-weight, w_{N1} and w_{N0}, respectively. Let R be the residue, the number of bit positions in which at least two of the sequences in the bundle set have different entries, viz. the Hamming distance of multiple sequences. Let t_i denote the probability of missing i line errors, i = 1, 2, ... , N. Let A_i be the number of columns in the bundle set with vertical 1-weight equal to i. The binomial coefficient $\binom{N}{i}$ is simply denoted by the symbol C^i or C_N^i.

Realistically, one can assume that $t_1 < t_2 < \ldots < t_{N-1} < t_N$, and $t_1 + t_2 + \ldots + t_{N-1} + t_N = 1$.

Example 10 Consider the following 5 sequences of length $L_s = 12$.

$\psi_1 = 1\ 1\ 1\ 1 \quad 0\ 0 \quad 0\ 1\ 0\ 1\ 0\ 1$

$\psi_2 = 1\ 1\ 1\ 1 \quad 0\ 0 \quad 1\ 0\ 1\ 0\ 0\ 0$

$\psi_3 = 1\ 1\ 1\ 1 \quad 0\ 0 \quad 0\ 0\ 0\ 1\ 1\ 0$

$\psi_4 = 1\ 1\ 1\ 1 \quad 0\ 0 \quad 1\ 1\ 0\ 0\ 0\ 1$

$\psi_5 = 1\ 1\ 1\ 1 \quad 0\ 0 \quad 0\ 1\ 1\ 1\ 0\ 0$

$|\ w_{51}\ | \quad |\ w_{50}\ | \quad |\ \ R\ \ |$

$2\ 3\ 2\ 3\ 1\ 2$

Here the 5th-order 1-weight is $w_{51} = 4$, 5th-order 0-weight is $w_{50} = 2$, and residue R = 6, since there are 6 bit positions in which at least two of the sequences differ. The vertical 1-weights are $A_1 = 1$, $A_2 = 3$, $A_3 = 2$, $A_4 = 0$, and $A_5 = 0$.

Theorem 34 The Nth-order missed error probability estimate when N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged by using an N-input AND (NAND) gate is

$$\delta_N\,(\text{AND/NAND}) = \frac{1}{L_s}\sum_{i=0}^{N-1} A_i \; - \; \frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i} A_{N-i}\right).$$

Proof:

- The number of single line errors missed by an AND (NAND) gate is :

  $(C_N^1 - 1)\,A_{N-1} + C_N^1\,A_{N-2} + C_N^1\,A_{N-3} + \ldots + C_N^1\,A_{N-(N-1)} + C_N^1\,A_{N-N} = C_N^1\,(A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_{N-1}$

- The number of double line errors missed by an AND (NAND) gate is :

  $C_N^2 A_{N-1} + (C_N^2 - 1)\,A_{N-2} + C_N^2 A_{N-3} + \ldots + C_N^2\,A_{N-(N-1)} + C_N^2 A_{N-N} = C_N^2\,(A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_{N-2}$

- The number of (N-1) line errors missed by an AND (NAND) gate is :

  $C_N^{N-1} A_{N-1} + C_N^{N-1} A_{N-2} + C_N^{N-1} A_{N-3} + \ldots + (C_N^{N-1} - 1)\,A_{N-(N-1)} + C_N^{N-1} A_{N-N} = C_N^{N-1}\,(A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_1$

- The number of N line errors missed by an AND (NAND) gate is :

$$A_{N-1} + A_{N-2} + A_{N-3} + \ldots + A_{N-(N-1)} = C_N^N (A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_0$$

$$\delta_N (\text{AND/NAND}) = \frac{t_1}{C_N^1 L_s} [C_N^1 (A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_{N-1}] + \frac{t_2}{C_N^2 L_s} [C_N^2 (A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_{N-2}] + \ldots$$

$$\ldots + \frac{t_{N-1}}{C_N^{N-1} L_s} [C_N^{N-1} (A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_1]$$

$$+ \frac{t_N}{C_N^N L_s} [C_N^N (A_{N-1} + A_{N-2} + \ldots + A_1 + A_0) - A_0]$$

$$= \frac{1}{L_s} \sum_{i=0}^{N-1} A_i - \frac{1}{L_s} \sum_{i=1}^{N} (\frac{t_i}{C_N^i} A_{N-i}).$$

Theorem 35 The Nth-order missed error probability estimate when N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged by using an N-input OR (NOR) gate is

$$\delta_N (\text{OR/NOR}) = \frac{1}{L_s} \sum_{i=1}^{N} A_i - \frac{1}{L_s} \sum_{i=1}^{N} (\frac{t_i}{C_N^i} A_i).$$

Proof:

- The number of single line errors missed by an OR (NOR) gate is :

$$C_N^1 A_N + C_N^1 A_{N-1} + C_N^1 A_{N-2} + \ldots + C_N^1 A_2 + (C_N^1 - 1) A_1 = C_N^1 (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_1$$

- The number of double line errors missed by an OR (NOR) gate is :

$$C_N^2 A_N + C_N^2 A_{N-1} + \ldots + (C_N^2 - 1) A_2 + C_N^2 A_1$$

$$= C_N^2 (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_2$$

- The number of (N-1) line errors missed by an OR (NOR) gate is :

$$C_N^{N-1} A_N + (C_N^{N-1} - 1) A_{N-1} + \ldots + C_N^{N-1} A_2 + C_N^{N-1} A_1 = C_N^{N-1} (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_{N-1}$$

- The number of N line errors missed by an OR (NOR) gate is :

$$A_1 + A_2 + A_3 + \ldots + A_{N-1} = C_N^N (A_1 + A_2 + A_3 + \ldots + A_{N-1} + A_N) - A_N$$

$$\delta_N(\text{OR/NOR}) = \frac{t_1}{C_N^1 L_s} [C_N^1 (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_1]$$

$$+ \frac{t_2}{C_N^2 L_s} [C_N^2 (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_2] + \ldots$$

$$+ \frac{t_{N-1}}{C_N^{N-1} L_s} [C_N^{N-1} (A_1 + A_2 + \ldots + A_{N-1} + A_N) - A_{N-1}]$$

$$+ \frac{t_N}{C_N^N L_s} [C_N^N (A_1 + A_2 + A_3 + \ldots + A_{N-1} + A_N) - A_N]$$

$$= \frac{1}{L_s} \sum_{i=1}^{N} A_i - \frac{1}{L_s} \sum_{i=1}^{N} (\frac{t_i}{C_N^i} A_i).$$

Theorem 36 The Nth-order missed error probability estimate when N sequences $\psi_1, \psi_2, \ldots, \psi_N$ are merged by using an N-input XOR (XNOR) gate

- ***1) when N is odd is :***

$$\delta_N(\text{XOR/XNOR}) = t_2 + t_4 + t_6 + \ldots + t_{N-1};$$

- 2) when N is even is :

$$\delta_N(\text{XOR/XNOR}) = t_2 + t_4 + t_6 + \ldots + t_N.$$

Proof:

- Case 1) :
  - The number of single line errors missed by an XOR (XNOR) gate is : 0
  - The number of double line errors missed by an XOR (XNOR) gate is : $C_N^2 (W_{N1} + W_{N0} + R)$
  - The number of (N-1) line errors missed by an XOR (XNOR) gate is : $C_N^{N-1} (W_{N1} + W_{N0} + R)$
  - The number of N line errors missed by an XOR (XNOR) gate is : 0

$$\delta_N (\text{XOR/XNOR}) = \frac{t_1}{C_N^1 L_s} [0] + \frac{t_2}{C_N^2 L_s} [C_N^2 (W_{N1} + W_{N0} + R)] + \ldots + \frac{t_{N-1}}{C_N^{N-1} L_s} [C_N^2 (W_{N1} + W_{N0} + R)] + \frac{t_N}{C_N^N L_s} [0]$$

$$= t_2 + t_4 + t_6 + \ldots + t_{N-1} .$$

- Case 2) :
  - The number of single line errors missed by an XOR (XNOR) gate is : 0
  - The number of double line errors missed by an XOR (XNOR) gate is : $C_N^2 (W_{N1} + W_{N0} + R)$
  - The number of (N-1) line errors missed by an XOR (XNOR) gate is : 0
  - The number of N line errors missed by an XOR (XNOR) gate is : $C_N^N (W_{N1} + W_{N0} + R)$

$$\delta_N (\text{XOR/XNOR}) = \frac{t_1}{C_N^1 L_s} [0] + \frac{t_2}{C_N^2 L_s} [C_N^2 (W_{N1} + W_{N0} + R)] + \ldots + \frac{t_{N-1}}{C_N^{N-1} L_s} [0] + \frac{t_N}{C_N^N L_s} [C_N^2 (W_{N1} + W_{N0} + R)]$$

$$= t_2 + t_4 + t_6 + \ldots + t_N .$$

5.2.1 Optimal mergeability criteria and gate selection

In the previous section, we derived the Nth-order missed error probability estimates [103] for individual gates. Based on those results, we now establish the optimal generalized mergeability criteria while merging an arbitrary number of output sequences under condition of stochastic dependence of multiple line errors.

Theorem 37 For two N-input gates g_1 and g_2, g_1 is preferable to g_2 for optimal merger, if and only if, their Nth-order missed error probability estimates satisfy the condition: $\delta_N(g_1) < \delta_N(g_2)$.

Proof:
The proof of the theorem follows obviously based on the results of Theorems 34 through 36.

5.2.1.1 Implementation - Exact approach

We select the particular N-input gate for optimally merging the N sequences at the CUT output based on Theorem 37; that is, a gate g_1 is selected over a gate g_2 when their respective missed error probability estimates satisfy the condition $\delta_N(g_1) < \delta_N(g_2)$. In an exact approach, we first compute δ_N (AND/NAND) and δ_N(OR/NOR) for the N candidate sequences, compare their values, and then select the logic gate corresponding the lesser value of the estimates, that is, decide to choose either an AND (NAND) gate over an OR (NOR) gate or vice versa. But we still have to find out if our primary selection is better than the selection of an XOR (XNOR) gate, and to realize that, we next compare the missed error probability estimate $\delta_N(i)$ of this selected gate i with that of an XOR (XNOR) gate, viz. δ_N (XOR/XNOR). Ultimately we select the logic gate with the least value of the missed error probability estimate for optimal merger. Unfortunately, because of computational reasons that could be rather intensive in most situations with consequent strain on available CPU time and storage, the exact approach does not appear to be an ideal method in practice, and as such we next propose some heuristics that will considerably

reduce our computational time and storage. This latter approach we shall designate as a heuristic approach and is outlined below.

5.2.1.2 Implementation - Heuristic approach

In order to outline the basic philosophy of the heuristic approach in the selection of appropriate gates for optimal merger of an arbitrary set of response sequences at the CUT output, we need to first introduce certain relevant definitions and assumptions to establish our mathematical groundwork. The derivation given below is based on the assumption of N being odd, though the expressions will be very much similar when N is even as well.

5.2.1.2.1 Definitions and assumptions

Let $w_{N1} = A_0$, $w_{N0} = A_N$, and $A_1 + A_2 + \ldots + A_{N-2} + A_{N-1} = R$.

1) When N is odd, we have

$C_N^0 = C_N^N = 1;$

$C_N^1 = C_N^{N-1} = N;$

$C_N^2 = C_N^{N-2} = \frac{N(N-2)}{2!} = aC_N^1;$

...

$C_N^r = C_N^{N-r} = \frac{N(N-1)(N-2)...(N-r+1)}{r!} = bC_N^1;$

...

$C_N^{\frac{N-1}{2}} = C_N^{\frac{N+1}{2}} = \frac{N(N-1)(N-2)...(N-\frac{N-1}{2}+1)}{(\frac{N-1}{2})!} = qC_N^1;$

Where each of a, b,..., q is greater than 1, there being no middle term in the binomial coefficients when N is odd.

2) When N is even, we have

$$C_N^0 = C_N^N = 1;$$

$$C_N^1 = C_N^{N-1} = N;$$

$$C_N^2 = C_N^{N-2} = \frac{N(N-2)}{2!} = aC_N^1;$$

...

$$C_N^r = C_N^{N-r} = \frac{N(N-1)(N-2)...(N-r+1)}{r!} = bC_N^1;$$

...

$$C_N^{\frac{N}{2}-1} = C_N^{\frac{N}{2}+1} = \frac{N(N-1)(N-2)...[N-(\frac{N}{2}-1)+1]}{(\frac{N}{2}-1)!} = uC_N^1;$$

$$C_N^{\frac{N}{2}} = \frac{N(N-1)(N-2)...(N-\frac{N}{2}+1)}{(\frac{N}{2})!} = vC_N^1$$

(middle item in the binomial coefficients)

where again each one of a, b, ... , u, v is greater than 1.

3) Assuming N odd, we have

$t_1 < t_2 < ... < t_{N-1} < t_N$ and $t_1 + t_2 + ... + t_{N-1} + t_N = 1$.

Let $r_1 = t_{N-1} - t_1$; $r_2 = t_{N-2} - t_2$; ... ; $r_{\frac{N-1}{2}} = t_{\frac{N+1}{2}} - t_{\frac{N-1}{2}}$.

Then we get $r_1 > r_2 > ... > r_{\frac{N-1}{2}}$.

Because $t_1 + t_2 + ... + t_{N-1} + t_N = 1$, we have

$1 - t_N = t_1 + t_2 + ... + t_{N-1} = (t_{N-1} - t_1) + (2t_1 + t_2 + ... + t_{N-2})$.

Thus $t_{N-1} - t_1 = (1 - t_N) - (2t_1 + t_2 + ... + t_{N-2}) \leq (1 - t_N)$

since $2t_1 + t_2 + ... + t_{N-2} \geq 0$. Therefore $\frac{t_{N-1} - t_1}{1 - t_n} = \frac{r_1}{1 - t_n} \leq 1$.

Similarly, $\frac{t_{N-2}-t_2}{1-t_n}=\frac{r_2}{1-t_n}\le 1;$

$\frac{t_{N-3}-t_3}{1-t_n}=\frac{r_3}{1-t_n}\le 1;$

...

$\frac{t_{\frac{N+1}{2}}-t_{\frac{N-1}{2}}}{1-t_n}=\frac{r_{\frac{N-1}{2}}}{1-t_n}\le 1.$

Because a, b,..., q each is greater than 1, we then get :

$$\frac{r_2}{a(1-t_n)}\le 1\ ;\ \frac{r_3}{b(1-t_n)}\le 1\ ;\ \ldots\ ;\ \frac{r_{\frac{N-1}{2}}}{q(1-t_n)}\le 1.$$

Now we can establish the following results.

Theorem 38 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the Nth-order 1-weight be w_{N1} and Nth- order 0-weight be w_{N0}. For merger of N sequences, an N-input AND (NAND) gate will be preferable to an N-input OR (NOR) gate for minimizing the missed error probability estimate, if

$$w_{N1}\text{-}w_{N0}\ge\frac{R}{N}\ ,\quad \text{or}\quad w_{N0}\text{-}w_{N1}\le\frac{-R}{N}.$$

Proof:

We have proved that

$$\delta_N\,(\text{AND/NAND})=\frac{1}{L_s}\sum_{i=0}^{N-1}A_i\ -\frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i}A_{N-i}\right);$$

$$\delta_N\,(\text{OR/NOR})=\frac{1}{L_s}\sum_{i=1}^{N}A_i\ -\frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i}A_i\right).$$

If it can be shown that δ_N (AND/NAND) -δ_N (OR/NOR) < 0, then obviously an AND (NAND) gate is preferable to an OR (NOR) gate for optimal merger. Hence

$$\delta_N(\text{AND/NAND}) - \delta_N(\text{OR/NOR}) = \frac{1}{L_s}\sum_{i=0}^{N-1} A_i - \frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i} A_{N-i}\right) - \frac{1}{L_s}\sum_{i=1}^{N} A_i +$$

$$\frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i} A_i\right) = \frac{1}{L_s}(1 - t_N)(w_{N0} - w_{N1}) - \frac{1}{L_s}\left[\frac{1}{C_N^1}(t_{N-1} - t_1)(A_1 - A_{N-1}) + \frac{1}{C_N^2}(\right.$$

$$\left. t_{N-2} - t_2)(A_2 - A_{N-2}) + \ldots + \frac{1}{C_N^{\frac{N-1}{2}}}(t_{\frac{N+1}{2}} - t_{\frac{N-1}{2}})(A_{\frac{N-1}{2}} - A_{\frac{N+1}{2}})\right]$$

$$= \frac{1}{L_s}[(1 - t_N)(w_{N0} - w_{N1}) - Q]$$

where $Q = \frac{1}{C_N^1}(t_{N-1} - t_1)(A_1 - A_{N-1}) + \frac{1}{C_N^2}(t_{N-2} - t_2)(A_2 - A_{N-2})$

$$+ \ldots + \frac{1}{C_N^{\frac{N-1}{2}}}(t_{\frac{N+1}{2}} - t_{\frac{N-1}{2}})(A_{\frac{N-1}{2}} - A_{\frac{N+1}{2}})$$

$$= \frac{r_1}{C_N^1}(A_1 - A_{N-1}) + \frac{r_2}{aC_N^1}(A_2 - A_{N-2}) + \ldots + \frac{r_{\frac{N-1}{2}}}{qC_N^1}(A_{\frac{N-1}{2}} - A_{\frac{N+1}{2}})$$

$$= \frac{1}{C_N^1}[(r_1 A_1 + \frac{r_2}{a}A_2 + \ldots + \frac{r_{\frac{N-1}{2}}}{q}A_{\frac{N-1}{2}}) - (r_1 A_{N-1} + \frac{r_2}{a}A_{N-2} + \ldots + \frac{r_{\frac{N-1}{2}}}{q}A_{\frac{N+1}{2}})].$$

Now let $E_1 = r_1 A_1 + \frac{r_2}{a}A_2 + \ldots + \frac{r_{\frac{N-1}{2}}}{q}A_{\frac{N-1}{2}}$;

$$E_2 = r_1 A_{N-1} + \frac{r_2}{a}A_{N-2} + \ldots + \frac{r_{\frac{N-1}{2}}}{q}A_{\frac{N+1}{2}}.$$

But $E_1 \geq 0$; $E_2 \geq 0$. Then $Q = \frac{1}{C_N^1}(E_1 - E_2)$

Or, $(1 - t_N)(w_{N0} - w_{N1}) \leq Q$

Or, $(1 - t_N)(w_{N0} - w_{N1}) \leq \frac{1}{N}(E_1 - E_2)$

Or, $(w_{N0} - w_{N1}) \leq \frac{1}{N(1-t_N)}(E_1 - E_2) = F_1$.

Now F_1 has a minimum value N_1 when $E_1 = 0$. In other words,

$$E_1 = r_1 A_1 + \frac{r_2}{a} A_2 + \ldots + \frac{r_{\frac{N-1}{2}}}{q} A_{\frac{N-1}{2}} = 0$$

Since $r_1, r_2, \ldots, r_{\frac{N-1}{2}} \neq 0$, then $A_1 = A_2 = \ldots = A_{\frac{N-1}{2}} = 0$; and we get $A_{N-1} + A_{N-2} + \ldots + A_{\frac{N+1}{2}} = R$.

$$\text{Now } N_1 = \frac{1}{N(1-t_N)}(-E_2) = \frac{-1}{N(1-t_N)}\left[r_1 A_{N-1} + \frac{r_2}{a} A_{N-2} + \ldots + \frac{r_{\frac{N-1}{2}}}{q} A_{\frac{N+1}{2}} \right]$$

$$= \frac{-1}{N}\left[\frac{r_1}{1-t_N} A_{N-1} + \frac{r_2}{a(1-t_N)} A_{N-2} + \ldots + \frac{r_{\frac{N-1}{2}}}{q(1-t_N)} A_{\frac{N+1}{2}} \right].$$

Earlier it was shown that $\frac{r_1}{1-t_N}, \frac{r_2}{a(1-t_N)}, \frac{r_{\frac{N-1}{2}}}{q(1-t_N)}$ each has a maximum value of 1.

Then N_1 has the minimum value

$$N_{1\min} = \frac{-1}{N}\left[A_{N-1} + A_{N-2} + \ldots + A_{\frac{N+1}{2}} \right] = \frac{-R}{N}.$$

That is, $w_{N0} - w_{N1} \leq \frac{-R}{N}$, or $w_{N1} - w_{N0} \geq \frac{R}{N}$.

Hence we can conclude that if $w_{N1} - w_{N0} \geq \frac{R}{N}$, or $w_{N0} - w_{N1} \leq \frac{-R}{N}$, we can select an AND (NAND) gate over an OR (NOR) gate for optimal sequence merger. Similarly, it can be shown that the same result follows when N is even.

Corollary 38.1 For N sequences $\psi_1, \psi_2, \ldots, \psi_N$ at the output of a CUT of length L_s, let the Nth-order 1-weight be w_{N1} and Nth-order 0-weight be w_{N0}. For merger of N sequences, an N-input OR (NOR) gate will be preferable to an N-input AND (NAND) gate for minimizing the missed error probability estimate, if

$$w_{N0} - w_{N1} \geq \frac{R}{N}, \text{ or } w_{N1} - w_{N0} \leq \frac{-R}{N}.$$

Definition 11 A group of candidate sequences as found from the frequency ordering having the maximum value of w_{N1} or w_{N0}, with $N \geq 2$, constitutes a max_group in the present context. Again, as usual,

w_{N1} The number of matching 1s in the same bit positions of max_group sequences;

w_{N0} The number of matching 0s in the same bit positions of max_group sequences;

R The residue, that is, the Hamming distance of multiple sequences;

N The number of candidate sequences in the max_group;

t_i The probability of missing i line errors, where t_i equals $2/3\ [\ (1/3)^{N-i}\ ]$, for $2 \leq i \leq N$, and is equal to $1/3\ [\ (1/3)^{N-2}\ ]$, for $i = 1$.

We give below our next algorithm that takes advantage of these heuristics in gate selection for optimal mergeability under stochastic dependence of multiple line errors to construct the compression networks at the CUT output.

Algorithm 6

A. Choosing between AND (NAND) and XOR (XNOR) gates for optimal merger of the candidate sequences selected from the original input sequences.

Step 1 : Using the original input sequence list , excluding the discarded sequences, set up the 1-cover table.

Step 2 : Find the corresponding frequency ordering of the sequences from the 1-cover table.

Step 3 : Try to find the max_group sequences in the frequency ordering having the maximum value of w_{N1} with N greater than two.

Step 4 : If $w_{N1} - w_{N0} \geq \frac{R}{N}$, or $w_{N0} - w_{N1} \leq \frac{-R}{N}$, select an AND (NAND) gate over an OR (NOR) gate, but it still has to compared with XOR (XNOR) gate. Go to next Step 5. Otherwise, go to Step B, which means that there are no max_group sequences that can be merged by an AND (NAND) gate over an OR (NOR) gate.

Step 5 : Compute :

$$\delta_N(\text{AND/NAND}) = \frac{1}{L_s}\sum_{i=0}^{N-1} A_i - \frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i} A_{N-i}\right);$$

$$\delta_N(\text{XOR/XNOR}) = t_2 + t_4 + t_6 + \ldots .$$

Step 6 : Compare the results from the previous step , viz. Step 5.

- If $\delta_N(\text{AND/NAND}) \leq \delta_N(\text{XOR/XNOR})$, the candidate sequences will be merged by an AND (NAND) gate.
- If $\delta_N(\text{AND/NAND}) > \delta_N(\text{XOR/XNOR})$, the candidate sequences will be merged by an XOR (XNOR) gate.

Step 7 : Store the output from the preceding step, discarding the merged candidate sequences.

Step 8 : Repeat Steps 1 through 7, try to find other satisfied max_group sequences which can be merged by AND (NAND) or XOR (XNOR) gates at the same level until no such sequences can be found. Go to Step B.

B. Choosing between OR (NOR) and XOR (XNOR) gates for optimal merger of the candidate sequences selected from the original input sequences.

Step 1 : Using the original input sequence list, excluding the discarded sequences, set up the 0-cover table.

Step 2 : Find the corresponding frequency ordering of the sequences in the 0-cover table.

Step 3 : Try to find the max_group sequences from the frequency ordering. That is, try to find the sequences in the frequency ordering having the maximum value of w_{N0}, with N as large as possible.

Step 4 : If $w_{N0} - w_{N1} \geq \frac{R}{N}$, or $w_{N1} - w_{N0} \leq \frac{-R}{N}$, select an OR (NOR) gate over an AND (NAND) gate for optimal merger, but it still has to be compared with an XOR (XNOR) gate. Go to next Step 5. Otherwise, go to Step C, which means, there are no max_group sequences that can be merged by an OR (NOR) gate over an AND (NAND) gate.

Step 5 : Compute

$$\delta_N(\text{OR/NOR}) = \frac{1}{L_s}\sum_{i=1}^{N} A_i - \frac{1}{L_s}\sum_{i=1}^{N}\left(\frac{t_i}{C_N^i} A_i\right);$$

$$\delta_N(\text{XOR/XNOR}) = t_2 + t_4 + t_6 + \ldots .$$

Step 6 : Compare the results as obtained from Step 5.

- If $\delta_N(\text{OR/NOR}) \leq \delta_N(\text{XOR/XNOR})$, the candidate sequences will be merged by an OR (NOR) gate.
- If $\delta_N(\text{OR/NOR}) > \delta_N(\text{XOR/XNOR})$, the candidate sequences will be merged by an XOR (XNOR) gate.

Step 7 : Store the output from the preceding step, discarding the merged candidate sequences.

Step 8 : Repeat Steps 1 through 7, try to find other satisfied max_group sequences which can be merged by OR (NOR) or XOR (XNOR) gates at the same level until no such sequences can be found. Go to Step C.

C. Choosing amongst AND (NAND), OR (NOR) and XOR (XNOR) gates for optimal merger.

Step 1 : Compute δ_N (AND/NAND), δ_N (OR/NOR), and δ_N (XOR/XNOR), and then compare the results.

- If δ_N(AND/NAND) ≤ δ_N(OR/NOR), and δ_N(AND/NAND) ≤ δ_N(XOR/XNOR), the candidate sequences will be merged by an AND (NAND) gate.
- If δ_N(OR/NOR) ≤ δ_N(AND/NAND), and δ_N(OR/NOR) ≤ δ_N(XOR/XNOR), the candidate sequences will be merged by an OR (NOR) gate.
- Otherwise, the candidate sequences will be merged by an XOR (XNOR) gate.

Step 2 : Store the resulting output sequence, discarding the grouped sequences.

D. Set up the new original input sequence list comprised of the sequences obtained at the previous level, repeat Steps A-C, until a single output sequence is obtained.

CHAPTER 6

ZERO-ALIASING SPACE COMPACTION WITH MAXIMAL COMPACTION RATIO

In this chapter, we suggest a novel approach to designing zero-aliasing space compactors using deterministic compacted test sets as well as pseudorandom testing. The developed approach utilizes conventional switching theory concepts of edge-inclusion table and frequency ordering [101], as used in the minimization of switching functions, together with concepts of strong and weak compatibilities of response data outputs in the design, based on detectable single stuck-line faults of the MUT.

In order for a space compactor to possess zero-aliasing for a particular fault class, it is necessary that all faults that are detectable at the output of the MUT must also be detectable at the compactor output. This translates into adding logic gates at the output of the MUT in a way so that the overall circuit comprised of the MUT and compressor does not introduce any redundancy. That is, 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. For an irredundant circuit, evidently all faults are detectable at the compactor output. For pseudorandom testing, the problem of designing a zero-aliasing space compactor is similar to the case of deterministic compacted testing, that is, adding logic gates at the MUT output without introducing redundancy but only with respect to the pseudorandom test set being used. The procedure of adding logic gates is continued, for deterministic and pseudorandom testing, using gate selection criteria as developed earlier in the dissertation, for double as well as multiple mergers as long as the overall circuit comprised of the MUT and selected gates remain irredundant until either a single output is produced or as few outputs as possible is obtained.

The advantages of aliasing-free space compaction as developed herein over earlier techniques are evidently clear – zero-aliasing is achieved without any modifications of the MUT, the area overhead and signal propagation delay are relatively less compared to conventional parity tree linear compactors, and the approach used works equally well with both deterministic and pseudorandom test sets. Besides, the approach uses less computation in the design process in successive stages compared to the existing technique [14] that uses an exhaustive approach at every step to decide on the gate to be used for merger of a pair of lines of the MUT to achieve zero-aliasing, if possible. The technique utilizes a mathematically sound selection criterion of merger of a number of lines of the MUT to decide on the gate for zero-aliasing, and achieves maximum compaction ratio in the design, as is evident from extensive simulation experiments conducted on ISCAS 85 combinational and ISCAS 89 full scan sequential benchmark circuits.

The mathematical basis underlying the design procedure is given next.

6.1 Mathematical Basis

Property 5 Let A and B represent two of the outputs of a MUT. Let these MUT outputs be merged by a gate from the logic family AND/NAND, OR/NOR, XOR/XNOR gate and let the gate output be z_l. Then we can envisage the undernoted scenarios :

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

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

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

Definition 12 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3

). If the MUT outputs A, B are merged by an AND(NAND) gate, we define lines A, B to be strongly AND(NAND) compatible, written as

(AB) s-AND(NAND) compatible.

Definition 13 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above (but not Cases 1-2). If the MUT outputs A, B are merged by an AND(NAND) gate, we define lines A, B to be weakly AND(NAND) compatible, written as

(AB) w-AND(NAND) compatible.

Definition 14 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to none of the conditions as specified by Cases 1-3 above. If the MUT outputs A, B are merged by an AND(NAND) gate, we define lines A, B to be AND (NAND) incompatible, written as

(AB) AND(NAND) incompatible.

Definition 15 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the MUT outputs A, B are merged by an OR(NOR) gate, we define lines A, B to be strongly OR(NOR) compatible, written as

(AB) s-OR(NOR) compatible.

Definition 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above (but not Cases 1-2). If the MUT outputs A, B are merged by an OR(NOR) gate, we define lines A, B to be weakly OR (NOR) compatible, written as

(AB) w-OR(NOR) compatible.

Definition 17 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to none of the conditions as specified by Cases 1-3 above. If the MUT outputs A, B are merged by an OR(NOR) gate, we define lines A, B to be OR(NOR) incompatible, written as

(AB) OR(NOR) incompatible.

Definition 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the MUT outputs A, B are merged by an XOR(XNOR) gate, we define lines A, B to be strongly XOR(XNOR) compatible, written as

(AB) s-XOR(XNOR) compatible.

Definition 19 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above (but not Cases 1-2). If the MUT outputs A, B are merged by an XOR(XNOR) gate, we define lines A, B to be weakly XOR(XNOR) compatible, written as

(AB) w-XOR(XNOR) compatible.

Definition 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to none of the conditions as specified by Cases 1-3 above. If the MUT outputs A, B are merged by an XOR(XNOR) gate, we define lines A, B to be XOR(XNOR) incompatible, written as

(AB) XOR(XNOR) incompatible.

Definition 21 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to either of the three conditions as specified by Cases 1-3 above, but unknown to us. If the MUT outputs A, B are merged under these conditions by an AND(NAND), OR(NOR) or XOR(XNOR) gate, then we define lines A, B to be simply AND(NAND), OR(NOR) or XOR (XNOR) compatible, written as

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

Theorem 39 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-AND(NAND) compatible. Similarly, let the outputs B, C be s-AND(NAND) compatible, and the outputs A, C be s-AND(NAND) compatible. Then (ABC) is s-AND(NAND) compatible and all faults are detected at z_1.

Theorem 40 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $A_1, A_2, \ldots, A_m$ are s-AND(NAND) compatible. Then all faults are detected at z_1.

Theorem 41 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-OR(NOR) compatible. Similarly, let the outputs B, C be s-OR(NOR) compatible, and the outputs A, C be s-OR(NOR) compatible. Then (ABC) is s-OR(NOR) compatible and all faults are detected at z_1.

Theorem 42 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $A_1, A_2, \ldots, A_m$ are s-OR(NOR) compatible. Then all faults are detected at z_1.

Theorem 43 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-XOR(XNOR) compatible. Similarly, let the outputs B, C be s-XOR(XNOR) compatible, and the outputs A, C be s-XOR(XNOR) compatible. Then (ABC) is s-XOR(XNOR) compatible and all faults are detected at z_1.

Theorem 44 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $A_1, A_2, \ldots, A_m$ are s-XOR(XNOR) compatible. Then all faults are detected at z_1.

Theorem 45 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-AND(NAND) compatible. Similarly, let the outputs B, C be w-AND(NAND) compatible, and the outputs A, C be w-AND(NAND) compatible. Then (ABC) is w-AND(NAND) compatible and all faults may or may not be detected at z_1.

Corollary 45.1 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Case 3 above, so that the outputs $A_1, A_2, \ldots, A_m$ are w-AND(NAND) compatible. Then all faults may or may not be detected at z_1.

Theorem 46 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-OR(NOR) compatible. Similarly, let the outputs B, C be w-OR(NOR) compatible, and the outputs A, C be w-OR(NOR) compatible. Then (ABC) is w-OR(NOR) compatible and all faults may or may not be detected at z_1.

Corollary 46.1 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Case 3 above, so that the outputs $A_1, A_2, \ldots, A_m$ are w-OR(NOR) compatible. Then all faults may or may not be detected at z_1.

Theorem 47 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 pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Case 3 above, so that the outputs A, B are w-XOR(XNOR) compatible. Similarly, let the outputs B, C be w-XOR(XNOR) compatible, and the outputs A, C be w-XOR(XNOR) compatible. Then (ABC) is w-XOR(XNOR) compatible and all faults may or may not be detected at z_1.

Corollary 47.1 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 pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Case 3 above, so that the outputs $A_1, A_2, \ldots, A_m$ are w-XOR(XNOR) compatible. Then all faults may or may not be detected at z_1.

In actual situations we do not know whether the merged outputs conform to the conditions specified by Cases 1-3 as discussed, and as such we have to deal exclusively with the case of simply compatible.

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

6.2 Incompatible Pairs of MUT Outputs

The algorithm to compute all incompatible pairs of MUT output lines is given below.

Algorithm 7

//The pseudocode description of the algorithm that computes all incompatible pairs of //MUT output lines for logic AND/NAND, OR/NOR and XOR/XNOR (pairs of lines //with less than 100.00% fault coverage):

//Generating fault lists:
Generate stuck-at-0 and stuck-at-1 fault lists by applying deterministic (compacted) input test patterns and pseudorandom input test patterns to the MUT;

//Forming combinations of output lines:
Form all possible combinations $\{£_i, £_j\}$ of output lines, taken two at a time;

//Storing the pairs of lines:
Store all pairs of output lines $\{£_i, £_j\}$ in database_pairs;

//Picking up one pair of lines:
Select the first pair of output lines $\{£_i, £_j\}$;

//Generating new, modified MUTs and processing them:
while (database_pairs is not empty)

{

//Generating new MUTs:

Construct three new MUTs (MUT_AND/NAND, MUT_OR/NOR, and MUT_XOR/XNOR) by merging the selected $\{£_i, £_j\}$ lines by logic gates (AND/NAND, OR/NOR, and XOR/XNOR), respectively, and adding them to the original MUT;

Delete the primary output lines $\{£_i, £_j\}$ from the MUT;
Add a new primary output line corresponding to lines $\{£_i, £_j\}$ merged by an appropriate logic gate, generating a new MUT;

//Testing the newly generated MUTs deterministically:

Apply deterministic input test patterns (compacted input test vectors) and inject all single faults into the newly generated MUTs;

//Checking for all the detected faults:

if (fault coverage for MUT_AND/NAND < 100.00 %)

Store $\{£_i, £_j\}$ to database_AND/NAND_deterministic_testing;

if (fault coverage for MUT_OR/NOR < 100.00 %)

Store $\{£_i, £_j\}$ to database_OR/NOR_deterministic_testing;

if (fault coverage for MUT_XOR/XNOR < 100.00 %)

Store $\{£_i, £_j\}$ to database_XOR/XNOR_deterministic_testing;

//Testing the newly generated MUTs pseudorandomly:

Apply pseudorandom input test patterns (2×10^6 input test vectors generated with a certain initial seed value α) and inject all single faults into the newly generated MUTs;

//Checking for all the detected faults:

if (fault coverage for MUT_AND/NAND < 100.00 %)

Store $\{£_i, £_j\}$ to database_AND/NAND_pseudorandom_testing;

if (fault coverage for MUT_OR/NOR < 100.00 %)

Store $\{£_i, £_j\}$ to database_OR/NOR_ pseudorandom_testing;

if (fault coverage for MUT_XOR/XNOR < 100.00 %)

Store $\{£_i, £_j\}$ to database_XOR/XNOR_ pseudorandom_testing;

Delete $\{£_i, £_j\}$ in database_pairs;

Select the next pair of output lines $\{£_i, £_j\}$;

}

6.3 Maximal Compatibles (MCs) of MUT Outputs

We presently discuss the procedure of finding all the maximal compatibility classes of MUT response data outputs based on the information of the incompatible pairs utilizing the modified cut-set (MCSET) algorithm of Das [116].

Some relevant definitions in this context are given below. In order to find the modified cut-sets from a complementary graph, a technique based on the concept of an *edge-inclusion table* corresponding to the complementary graph is used.

Definition 22 Given a connected complementary graph G, a table can be readily made in the following way. Write all the edges of the graph G in a row, and below each of the edges, make an entry of the vertices that include it in a columnar form. Each of the columns, headed by a specific edge, then gives a collection of the vertices that include that particular edge. Such a table is called an *edge-inclusion table* [116]. This table is very similar to a *cover table* as used in the simplification of switching functions.

Let $y_1, y_2, y_3, \ldots, y_m$, be m vertices representing all the different output lines of an n-input m-output MUT. The edge-inclusion table from the graph G in Figure 6.1 is presented as Table 6.1.

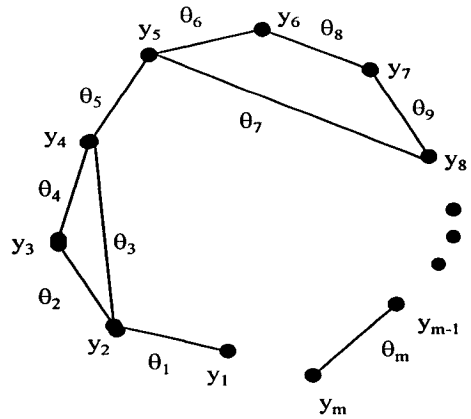


Figure 6.1 Incompatibility Graph G of m Vertices (MUT Outputs).

θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	...	θ_m
y_1	y_2	y_2	y_3	y_4	y_5	y_5	y_6	y_7	...	y_{m-1}
y_2	y_3	y_4	y_4	y_5	y_6	y_8	y_7	y_8	...	y_m

Table 6.1 Edge-Inclusion Table.

Definition 23 For any two distinct vertices y_i and y_j appearing in the *edge-inclusion table*, if the vertex y_i appears in more columns of the table than the vertex y_j, then an ordering of the vertices can be made as $y_i \geq y_j$; when both the vertices y_i and y_j appear in the same number of columns in the edge-inclusion table, the ordering can be made either as $y_i \geq y_j$ or as $y_i \leq y_j$. This kind of ordering (=) that can be established amongst different vertices in an edge-inclusion table is called the *frequency ordering* of the vertices [116].

The algorithm to compute the sets of maximal compatibles or MCs(AND/NAND), MCs(OR/NOR), and MCs(XOR/XNOR) is given next.

Algorithm 8

//The pseudocode description of the Modified Cut-Set algorithm that computes the sets
//of maximal compatibles MCs(AND/NAND), MCs(OR/NOR), and
//MCs(XOR/XNOR), given all incompatible pairs of MUT output lines for logic
//AND/NAND, OR/NOR and XOR/XNOR:

//Computing all incompatible pairs of MUT output lines:
Find all incompatible pairs of MUT output lines for logic AND/NAND, OR/NOR and XOR/XNOR (pairs of lines with less than 100.00% fault coverage), by applying deterministic (compacted) input test patterns and pseudorandom input test patterns to the MUT (Algorithm 7);

//Forming the Edge Inclusion tables:
Form the Edge Inclusion tables for

{
Incompatible pairs of MUT outputs $\{£_i, £_j\}$ for logic (AND/NAND);
Designate the table as TA_k;
Store TA_k;
and
Incompatible pairs of MUT outputs $\{£_i, £_j\}$ for logic (OR/NOR);
Designate the table as TO_k;

Store TO_k;

and

Incompatible pairs of MUT outputs $\{£_i, £_j\}$ for logic (XOR/XNOR);

Designate the table as TX_k;

Store TX_k;

}

//Determining the dimensions (sizes) of the Edge Inclusion tables:

Determine the size of the Edge Inclusion table for logic (AND/NAND) as ize_EIT_AND/NAND;

Determine the size of the Edge Inclusion table for logic (OR/NOR) as size_EIT_OR/NOR;

Determine the size of the Edge Inclusion table for logic (XOR/XNOR) as size_EIT_XOR/XNOR;

//Selecting which of the tables to be processed and its current size (dimension):

$T_k = TA_k$ (or TO_k or TX_k);

current_size_T_k = size_EIT_AND/NAND (or size_EIT_OR/NOR or size_EIT_XOR/XNOR);

//Storing the table in database:

Store T_k;

//Only one maximal compatibility class (MCC) or MC:

if (current_size_T_k == 0)

There exists only one solution, a single MC consisting of all MUT output lines;

//Finding the reduced Edge Inclusion tables and sets of redundant output lines:

while (there is a table T_k in the store and current_size_T_k ? 0)

{

//Finding the frequency ordering of the output lines:

for(i = 1; i < current_size_T_k; i++)

Find the frequency ordering of the output lines $£_i$, in the Edge Inclusion table T_k;

//Selecting one of the output lines and updating the Edge Inclusion table:

if (the number of output lines that appear in the maximum number of columns in the table $T_k > 1$)

{

Choose one of the output lines, $£_i$, that appear in the maximum number of columns in the table T_k;

Select the output line $£_i$, and delete all the r columns of the table T_k in which the line $£_i$ appears;

}

else

Select the output line $£_i$, and delete all the r columns of the table T_k in which the line $£_i$ appears;

//Forming the reduced Edge Inclusion table $T1_k$:

if (not all the columns of the table T_k are deleted)

{

Form a reduced Edge Inclusion table consisting of the remaining columns of T_k;

$T1_k = T_{k\text{-}r}$;

Store $T1_k$;

}

else

{

//No changes:

$T1_k = T_k$;

Store $T1_k$;

}

//Forming the reduced Edge Inclusion table $T2_k$:

Delete the output line $£_i$, from all the r columns of the table T_k in which it appears;

Select those output lines $£_j$, that appear alone in the r columns from which the output line $£_i$, is deleted;

Delete all the columns of the table T_k containing these output lines $£_j$;

if (not all the columns of the table T_k are deleted)

{

Form a reduced Edge Inclusion table consisting of the remaining columns in T_k;

$T2_k = T_{k\text{-}s}$;

Store $T2_k$;

}

else

{

//No changes:

$T2_k = T_k$;

Store $T2_k$;

}

Pick up one table T_k from the database;

}

//Processing all the selected sets of output lines:

Collect the different selected output lines $£_k$;

Obtain the final sets of output lines;

Sort all the different sets of output lines;

Compare the different sets for matching;

Delete all but one from the matching sets;

Compare the different sets for redundancy;

Delete the redundant sets;

Find the final sets of output line MCs;

6.4 Constructing Aliasing-Free Compactors

The algorithm to generate the aliasing-free compaction tree is given next.

Algorithm 9

//The pseudocode description of the algorithm that constructs the aliasing-free
//compaction tree:

//Defining the maximum compression ratio:
Define the maximum compression of output lines that may be achieved (max_compression);
stage = total_number_of_outputs;
current_stage = 0;
remaining_output_lines = total_number_of_outputs;

//Determining a compaction tree at every stage until a minimal number of output lines is //obtained:
while ((current_stage < stage-1) or (remaining_output_lines < max_compression))
{
//Computing all maximum compatibility (MC) classes for logic AND/NAND and //OR/NOR and XOR/XNOR:
Find all the sets of MCs(AND/NAND) *and* MCs(OR/NOR) *and* MCs(XOR/XNOR) by executing Algorithm 8;
total_number_of_MCs = number_of_MCs(AND/NAND) + number_of_MCs(OR/NOR) + number_of_MCs(XOR/XNOR);

//Making a decision regarding which of the MCs to pick up:
for (k = 0; k < total_number_of_MCs; k++)
{
//picking up one MC class:
Compare among all the MCs(AND/NAND) and MCs(OR/NOR) and MCs(XOR/XNOR);
Randomly pick up ONE MC_k with the largest number of output lines;

if (number_of_lines_in_MC_m = = number_of_lines_in_MC_n; m ? n)

{

Pick up any one arbitrarily;

}

}

//Merging selected output lines:

Merge selected output lines in MC_k by the appropriate logic gate (AND/NAND or OR/NOR or XOR/XNOR);

Add a new output line corresponding to the selected merged outputs in MC_k;

//Discarding the MCs already used:

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

remaining_output_lines = remaining_output_lines – selected_output_lines_in_ MC_k;

total_number_of_MCs = number_of_MCs(AND/NAND) + number_of_MCs(OR/NOR) + number_of_MCs(XOR/XNOR);

//Deciding on what are the other MCs that can be picked up:

while ((remaining_output_lines ? 1) and (MCs(AND/NAND) ? NULL or MCs(OR/NOR) ? NULL or MCs(XOR/XNOR) ? NULL))

{

for (p = 0; p < total_number_of_MCs; p++)

{

//picking up another MC class:

Of the remaining output lines and remaining MCs(AND/NAND) and MCs(OR/NOR) and MCs(XOR/XNOR), randomly pick up another MC_p with the largest number of lines not being merged previously;

//Merging selected output lines:

Merge the selected output lines in MC_p by the appropriate logic gate (AND/NAND or OR/NOR or XOR/NOR);

Add a new output line corresponding to the selected merged outputs in MC_p;

//Discarding the already used MCs:

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

total_number_of_MCs = number_of_MCs(AND/NAND) + number_of_MCs(OR/NOR) + number_of_MCs(XOR/XNOR);

remaining_output_lines = remaining_output_lines – selected_output_lines_in_MC_p;

Break the loop;

if (number_of_lines_in_MC_m = = number_of_lines_in_MC_n; m ? n)

{

Pick up any one arbitrarily;

Merge the selected output lines in MC_p by the appropriate logic gate (AND/NAND or OR/NOR or XOR/XNOR);

Add a new output line corresponding to the selected merged outputs in MC_p;

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

Total_number_of_MCs = number_of_MCs(AND/NAND) + number_of_MCs(OR/NOR) + number_of_MCs(XOR/XNOR);

remaining_output_lines = remaining_output_lines – selected_output_lines_in_MC_p;

Break the loop;

}

}

}

//Merging the remaining output lines that do not belong to any of the MCs already //picked up:

while ((remaining_output_lines ? 1) and ((MCs(AND/NAND) = = NULL and MCs(OR/NOR) = = NULL) and MCs(XOR/XNOR) = = NULL)))

{

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

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

Add a new output line corresponding to the selected merged outputs;

//Injecting faults into the circuit:

Inject stuck-at-0 and stuck-at-1 logic faults into the MUT + added hardware (compaction tree);

//Applying input test patterns:

Apply input test vectors (deterministic compacted test sets or pseudorandom test sets with a particular seed α) at the primary inputs of the MUT;

//Computing the fault coverage:

Compute the fault coverage at the outputs of the MUT + compaction tree;

//Checking for 100% fault coverage:

If the percentage of fault coverage is 100% then break the loop;

//Merging all the remaining lines with a two-input XOR/XNOR tree:

Else if not possible, merge the remaining output lines with XOR/XNOR, two output lines at a time;

Add a new output line corresponding to the selected merged outputs;

Inject stuck-at-0 and stuck-at-1 logic faults into the MUT + added hardware (compaction tree);

Apply input test vectors (deterministic compacted test sets or pseudorandom test sets with a particular seed α) at the primary inputs of the MUT;

Compute the fault coverage at the outputs of the MUT + compaction tree;

If the percentage of fault coverage is 100% then break the loop;

//Merging *all the remaining lines with non maximal MCs:*

Else if not possible, merge the remaining output lines with non maximal classes of MCs(AND/NAND) or MCs(OR/NOR);

Add a new output line corresponding to the selected merged outputs;

Inject stuck-at-0 and stuck-at-1 logic faults into the MUT + added hardware (compaction tree);

Apply input test vectors (deterministic compacted test sets or pseudorandom test sets with a particular seed α) at the primary inputs of the MUT;

Compute the fault coverage at the outputs of the MUT + compaction tree;

If the percentage of fault coverage is 100.00% then break the loop;

Else break the loop;

}

//Injecting faults into the circuit:

Inject stuck-at-0 and stuck-at-1 logic faults into the MUT + added hardware (compaction tree);

//Applying input test patterns at the inputs of the MUT:

Apply input test vectors (deterministic compacted test sets or pseudorandom test sets with a particular seed α) at the primary inputs of the MUT;

//Computing the number of faults detected with respect to the number of faults //injected:

Compute the fault coverage at the outputs of the MUT + compaction tree;

//Checking if all the injected faults are detected:

If the percentage of fault coverage is 100.00% then do the same for the 2^{nd} and later stages as long as a single output line or a minimal number of output lines is obtained;

//Proceeding the following next stage:

current_stage = current_stage + 1;

}

Example 11 Consider the benchmark circuit c432 shown in Figure 6.2. The circuit has thirty six primary inputs, one hundred and sixty logic gates, and seven primary outputs. For the forty seven compacted input test patterns, the fault-free output patterns and the corresponding detected stuck-at-logic faults are shown below in Table 6.2.

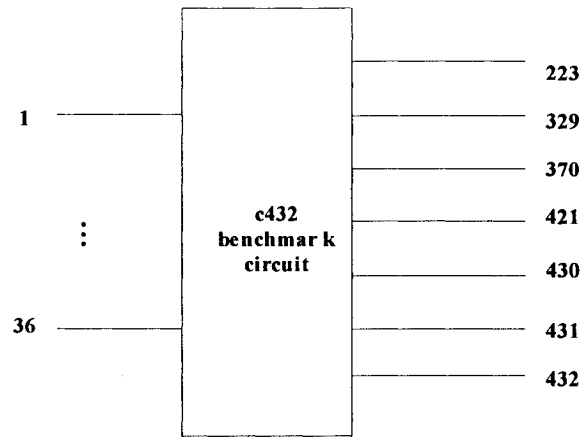


Figure 6.2 c432 Benchmark Circuit.

Five hundred and twenty stuck-at faults were injected into the MUT (c432 circuits), the (7 bits) output patterns were observed, compared to precomputed fault-free output patterns, and detected faults are recorded when faulty and fault-free output patterns mismatching occurred.

	Input Test Vectors	Outputs	No. of Faults Detected
Test #1	00011010011011010000000010101010110	1111101	26 s-at-faults detected
Test #2	00101001100101110010101111010110100	0111000	62 s-at-faults detected
Test #3	10110010010111001010110100000010010	1101110	22 s-at-faults detected
Test #4	11010110001011101100111001101000 1010	1101100	36 s-at-faults detected
Test #5	00000000000000000000000000000000000	0000000	74 s-at-faults detected
Test #6	01111001010011110111000101110101000 0	1011101	29 s-at-faults detected
Test #7	01100110001010100011001000010110 1101	1101000	19 s-at-faults detected
Test #8	01011011000000001100010000010101011 1	1111100	4 s-at-faults detected
Test #9	11010100101010000011111111010101001 1	0110010	22 s-at-faults detected
Test #10	11100001010010100111011101100001001 0	1011100	7 s-at-faults detected
Test #11	11011011001100001001101111100010101 0	0100000	12 s-at-faults detected
Test #12	11101101001101000011100000101000001 0	1111111	22 s-at-faults detected
Test #13	01111011000100001000001111000011110 1	1000000	11 s-at-faults detected
Test #14	00100111010010011100001010011111110 1	1011111	8 s-at-faults detected
Test #15	10110111000000001010010010001111110 0	1101011	18 s-at-faults detected
Test #16	00101011011111011000100011010000101 1	1111010	12 s-at-faults detected
Test #17	00100001111101000000101100101000101 1	0111101	9 s-at-faults detected
Test #18	00110110010110011110011111111110011 0	1101110	11 s-at-faults detected
Test #19	10110010101100010000001011111111010 0	1001010	5 s-at-faults detected
Test #20	01110000000111100111111110110110011 1	1010000	5 s-at-faults detected
Test #21	00011111111000111110100100010011011 1	0011110	9 s-at-faults detected
Test #22	10010101100000110010001100000010001 0	0101111	5 s-at-faults detected
Test #23	11111010111000000001100110000101101 1	1101001	19 s-at-faults detected
Test #24	10010001111010011001011100001010101 0	0011011	9 s-at-faults detected
Test #25	10111000101110011011011111111100001 1	0111001	7 s-at-faults detected
Test #26	00011011100010100011111111110001101 0	0011010	3 s-at-faults detected
Test #27	11100000100011111011000100110010111 1	1001101	4 s-at-faults detected
Test #28	11100010011101001010100101011000100 1	1011110	3 s-at-faults detected
Test #29	00110011011000111111011010110110001 0	0011110	3 s-at-faults detected
Test #30	10000010010011110101000111101000001 0	1101110	1 s-at-faults detected
Test #31	00100001001010011000010100010111111 0	1111011	2 s-at-faults detected
Test #32	00000011110110110001100101000110000 0	1011001	4 s-at-faults detected
Test #33	10101011101001011100011100100101000 1	1111001	3 s-at-faults detected
Test #34	10100001100100001101111100001110000 0	1101100	2 s-at-faults detected
Test #35	01001110110101110001000010001110000 1	1111110	1 s-at-faults detected
Test #36	11000000010011001101101000110110101 0	1101110	1 s-at-faults detected
Test #37	10101001000001011110010010101101100 1	1101101	1 s-at-faults detected
Test #38	01100000010011100100001011010011001 0	1111100	1 s-at-faults detected
Test #39	01111011001111010111010111011101011 1	1000000	5 s-at-faults detected
Test #40	00101110111111010000000101101110111 0	1011000	3 s-at-faults detected
Test #41	00100001100100101001001101000010010 1	1101000	1 s-at-faults detected
Test #42	00110010000000101111011111010010110 1	1101000	3 s-at-faults detected
Test #43	11011011011000001011100011110110101 1	1011010	2 s-at-faults detected
Test #44	10110001011100111101011100000000001	0111100	3 s-at-faults detected
Test #45	01000000101111101000010010101010000 0	1110000	1 s-at-faults detected
Test #46	01111101111100110111011110011000111 1	1000000	7 s-at-faults detected
Test #47	10110100001001111101110100101001110 1	1001101	3 s-at-faults detected

Table 6.2 Input Test Set, Fault-Free Output, and The Corresponding Detected Stuck-at-Logic Faults for the Benchmark Circuit c432.

All incompatible pairs of MUT output lines (O_i , O_j) for logic AND/NAND, OR/NOR and XOR/XNOR are computed by applying Algorithm 7 and using the deterministic input test set shown in Table 6.2. We get the following incompatible pairs [101] for logic AND/NAND, and OR/NOR as shown below in Table 6.3, Table 6.4, respectively.

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

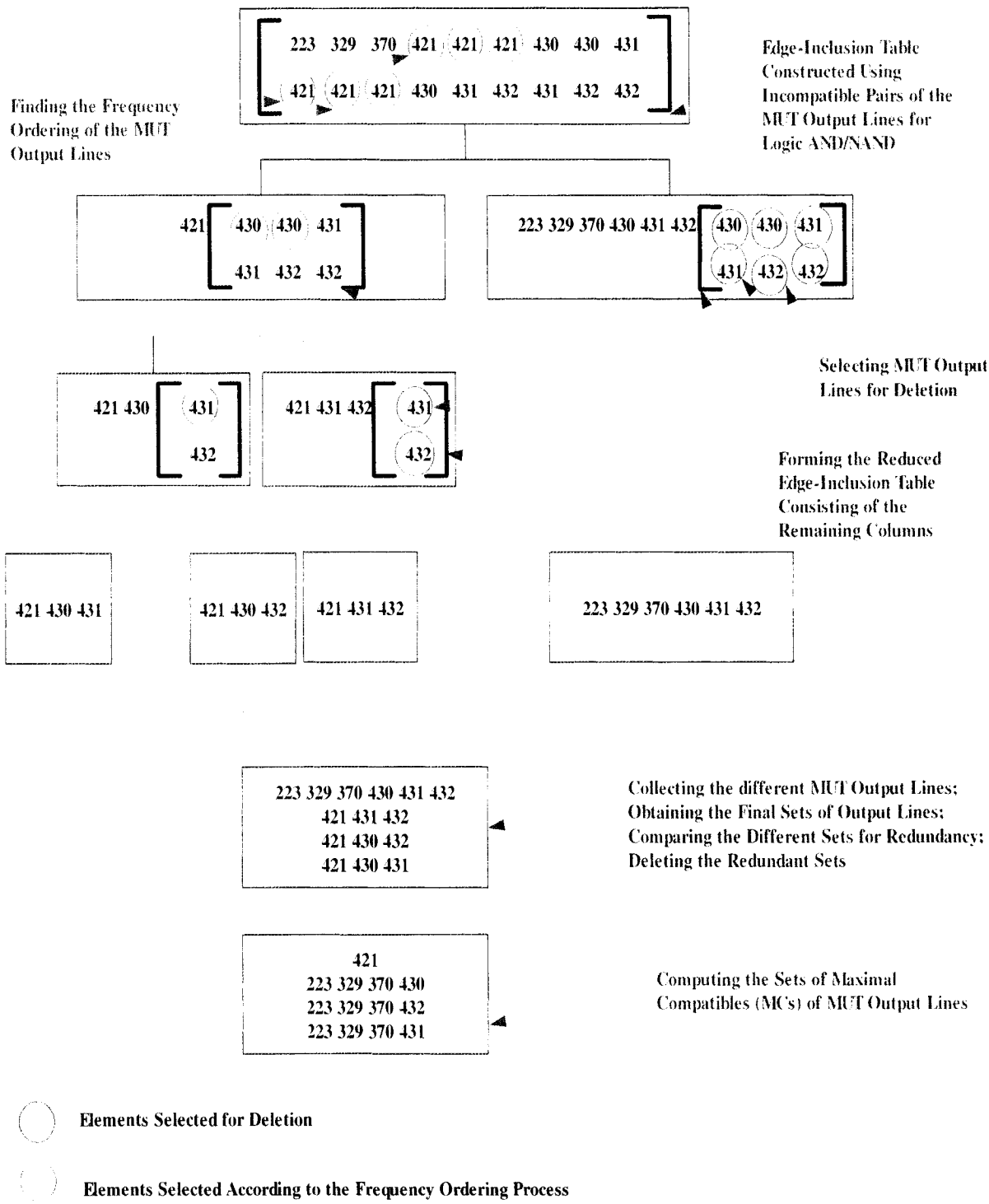
Table 6.3 Incompatible Pairs for Logic AND/NAND.

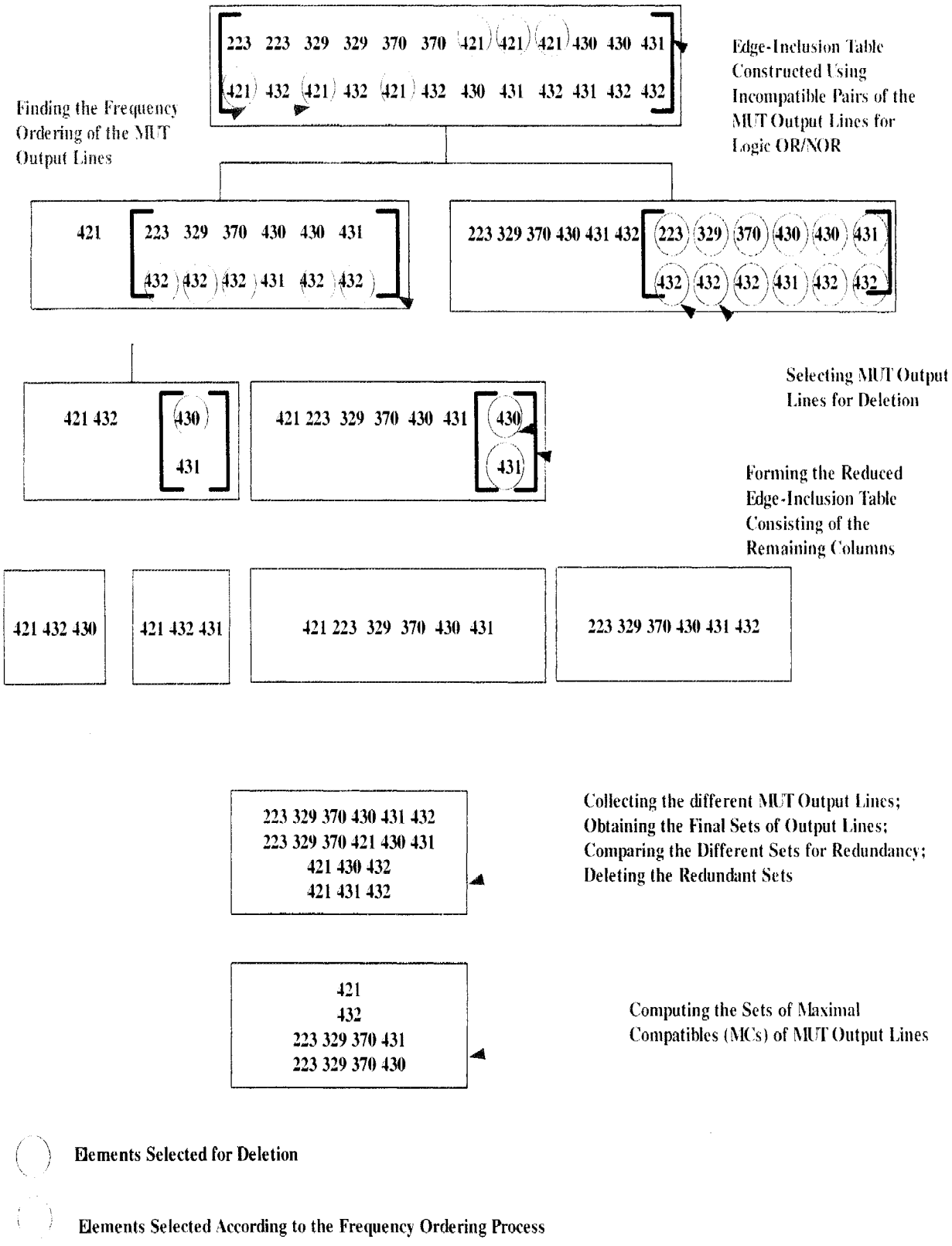
(O_i , O_j)
(223 , 421)
(223 , 432)
(329 , 421)
(329 , 432)
(370 , 421)
(370 , 432)
(421 , 430)
(421 , 431)
(421 , 432)
(430 , 431)
(430 , 432)
(431 , 432)

Table 6.4 Incompatible Pairs for Logic OR/NOR.

We also get an empty set of incompatible pairs for logic XOR/XNOR. All the output lines thus form a maximal compatibility class MC for XOR/XNOR.

The modified cut-set algorithm (Algorithm 8) to compute the sets of maximal compatibles or MCs of (AND/NAND), (OR/NOR), and (XOR/XNOR) is next applied.


223 329 370 421 421 421 430 430 431
421 421 421 430 431 432 431 432 432
Finding the Frequency Ordering of the MUT Output Lines
Edge-Inclusion Table Constructed Using Incompatible Pairs of the MUT Output Lines for Logic AND/NAND
421 430 430 431
431 432 432
223 329 370 430 431 432 430 430 431
431 432 432
Selecting MUT Output Lines for Deletion
421 430 431
432
421 431 432 431
432
Forming the Reduced Edge-Inclusion Table Consisting of the Remaining Columns
421 430 431
421 430 432
421 431 432
223 329 370 430 431 432
223 329 370 430 431 432
421 431 432
421 430 432
421 430 431
Collecting the different MUT Output Lines; Obtaining the Final Sets of Output Lines; Comparing the Different Sets for Redundancy; Deleting the Redundant Sets
421
223 329 370 430
223 329 370 432
223 329 370 431
Computing the Sets of Maximal Compatibles (MC's) of MUT Output Lines
Elements Selected for Deletion
Elements Selected According to the Frequency Ordering Process


223 223 329 329 370 370 421 421 421 430 430 431
421 432 421 432 421 432 430 431 432 431 432 432
Edge-Inclusion Table Constructed Using Incompatible Pairs of the MUT Output Lines for Logic OR/NOR
Finding the Frequency Ordering of the MUT Output Lines
421 223 329 370 430 430 431
432 432 432 431 432 432
223 329 370 430 431 432 223 329 370 430 430 431
432 432 432 431 432 432
Selecting MUT Output Lines for Deletion
421 432 430 431
421 223 329 370 430 431 430 431
Forming the Reduced Edge-Inclusion Table Consisting of the Remaining Columns
421 432 430
421 432 431
421 223 329 370 430 431
223 329 370 430 431 432
223 329 370 430 431 432
223 329 370 421 430 431
421 430 432
421 431 432
Collecting the different MUT Output Lines; Obtaining the Final Sets of Output Lines; Comparing the Different Sets for Redundancy; Deleting the Redundant Sets
421
432
223 329 370 431
223 329 370 430
Computing the Sets of Maximal Compatibles (MCs) of MUT Output Lines
Elements Selected for Deletion
Elements Selected According to the Frequency Ordering Process

We get the following maximal compatibles sets as shown in Table 6.5.

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

Table 6.5 Maximal Compatibles Sets.

The algorithm that constructs the aliasing-free compaction tree (Algorithm 9) is then used. A maximal compatibles set, consisting of more than one output line, is randomly picked up from Table 6.4 and the corresponding hardware (COMP) is added to the MUT. The stuck-at-logic faults are injected into the MUT and input test vectors are applied to the MUT+COMP. The output patterns are observed and compared to the corresponding fault-free patterns. If all injected faults are detected at the output of the MUT+COMP, then the maximal compatibles set will be retained and the same process will be repeated over until a compaction tree with a maximal compaction ratio is generated. Otherwise, another maximal compatibles set or a subset is randomly picked up again from Table 6.5 for processing.
Table 6.6 shows two different compaction trees generated with maximal compaction ratio (single output line) for c432 benchmark circuit.

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

Table 6.6 Compaction Trees with Maximal Compaction Ratio.

The two zero-aliasing compactors for the c432 benchmark circuit are shown below in Figure 6.3 and Figure 6.4.

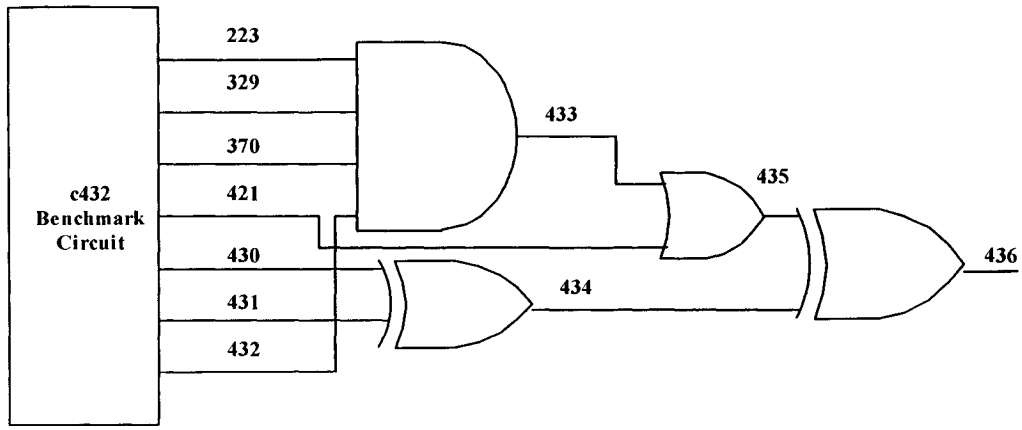


Figure 6.3 Zero-aliasing Compactor #1 Using Deterministic Testing.

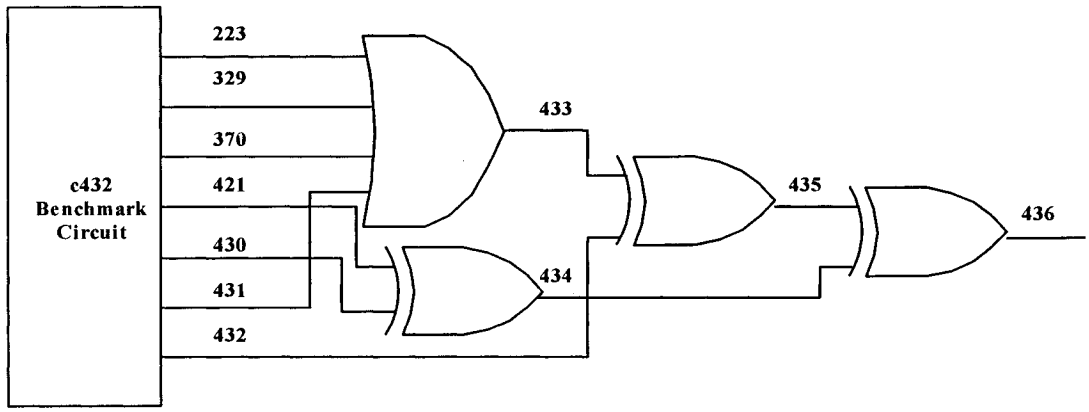


Figure 6.4 Zero-aliasing Compactor #2 Using Deterministic Testing.

Figure 6.5 and Figure 6.6 show two different zero-aliasing compactors for the c432 benchmark circuit generated using pseudorandom testing.

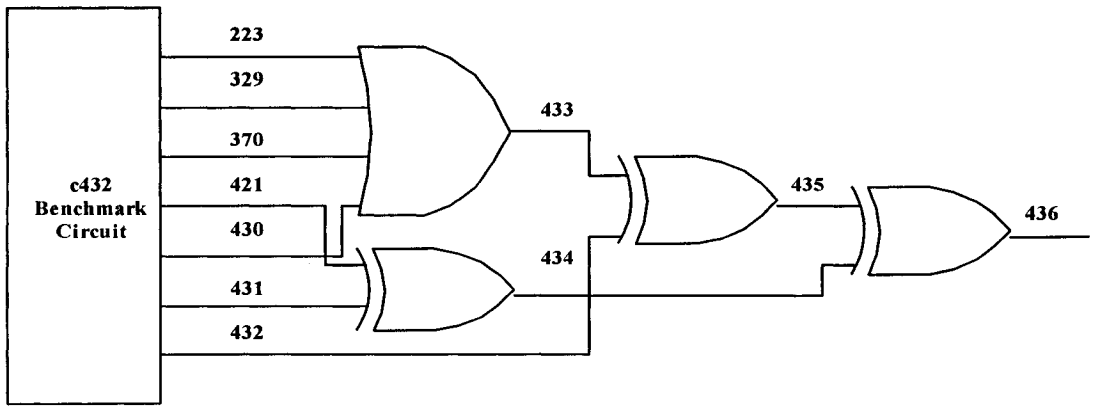


Figure 6.5 Zero-aliasing Compactor #1 Using Pseudorandom Testing.

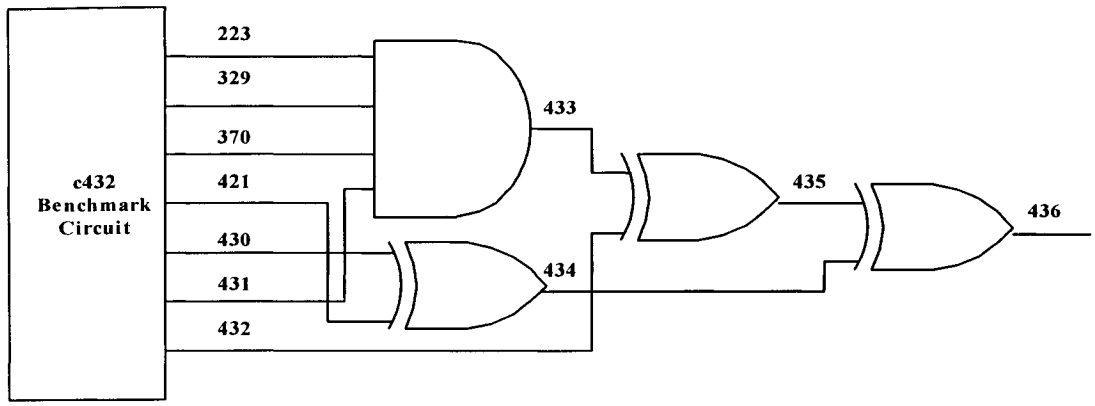


Figure 6.6 Zero-aliasing Compactor #2 Using Pseudorandom Testing.

6.5 Hardware Verification

A design architecture to verify the concept of zero-aliasing space compaction model for digital cores BIST was proposed by using a reconfigurable device. Several single-output zero-aliasing space compactors for a specific module under test [101] (MUT) were generated in software, realized in the Java language, and were downloaded and tested at runtime in a simulation environment written in JBits 2.8 by Xilinx Corporation [80].

6.5.1 JBits SDK

JBits is a development framework for Xilinx FPGAs based on the Java language. The JBits Application Programming Interface (API) provides low level access to the configuration of resources in a Xilinx FPGA [81]. Xilinx FPGAs are SRAM based and have the ability to be configured numerous times. JBits [82-84] provides a solution to support run-time reconfiguration (RTR).

RTR systems define the circuit's logic and routing just prior to, or during operation. They typically modify the circuit several times during execution of the application.

JBits provides the ability to rapidly create and modify Xilinx Virtex and XC4000 FPGA circuitry at run time by allowing direct access into the configuration bitstream.

A typical JBits program creates logic in a new bitstream, modifies the logic and interconnects in an existing bitstream, or performs an analysis of a bitstream.

The Xilinx Hardware Interface (XHWIF) is an API that provides a universal interface for communicating with different FPGA based boards. The Virtex Device Simulator (Virtex DS) works on bitstreams to emulate the FPGA device in software.

The JBits API is an ideal design environment for implementing logic circuits on mainstream FPGAs, especially when the hardware is not present or a proof-of-concept is initially required.

6.5.2 Verification scheme

The verification scheme involves introducing stuck-at-0 and stuck-at-1 faults within the MUT, and recording the corresponding outputs. The outputs are then compared to predefined fault-free signatures, in order to determine whether the injected fault can be detected or not. Consider the following multi-output combinational circuit [9] under test shown below in Figure 6.7.

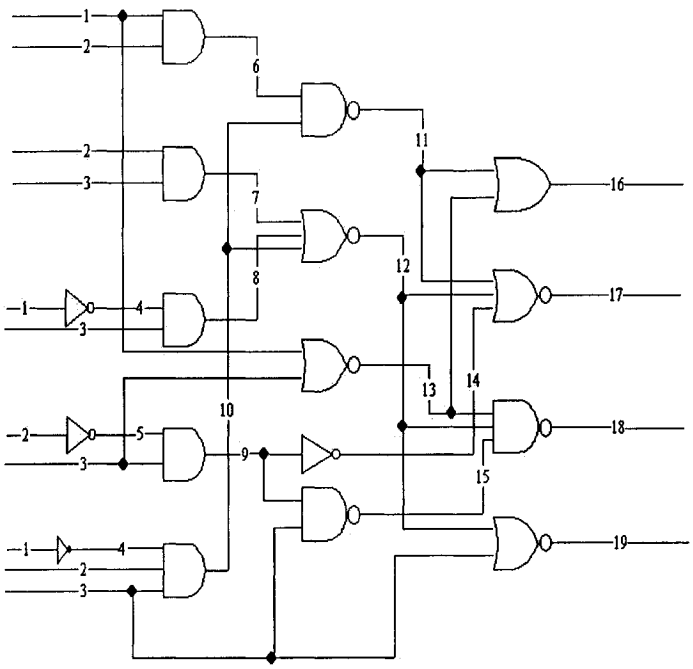


Figure 6.7 The Preliminary MUT.

6.5.3 Output compactors

Two different zero-aliasing compaction trees for the circuit under test (MUT) shown in Figure 6.7, were generated in software [101]. The single output compactors shown in Figure 6.8a and Figure 6.8b, reduce the number of system outputs from 4 to 1, which provide us with a smaller number of outputs to first store, and then compare. We will be testing both compressors, as connected to the MUT shown in figure 6.7. These compactors will prove the verification's scheme correct functionality.

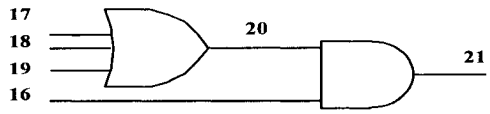


Figure 6.8a Compaction Circuit #1.

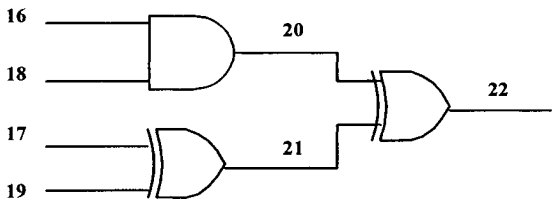


Figure 6.8b Compaction Circuit #2.

6.5.4 Hardware realization

This section will outline the actual design and realization of the verification scheme in hardware devices, such as FPGA development boards and hardware-emulating APIs (e.g. JBits). First, let us introduce the system architecture, which will be used to realize the verification scheme in hardware. Refer to figure 6.9 for the diagram at system level.

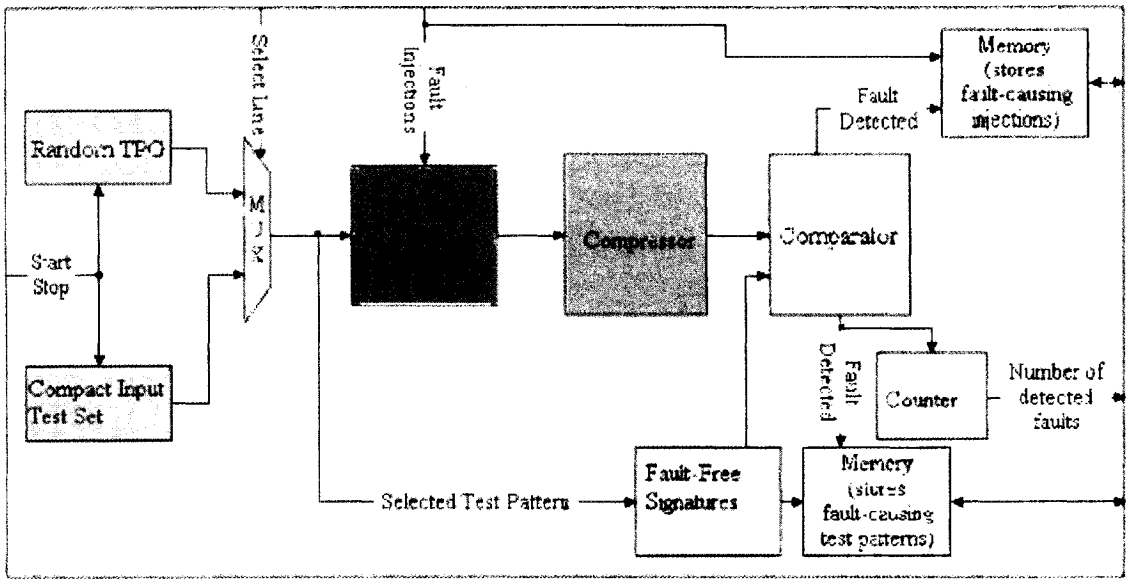


Figure 6.9 The Hardware Verification Scheme.

6.5.5 Hardware fault injection

The hardware fault injection technique is imperative in order to iteratively inject faults to every wire, and to test both the stuck-at-0 and stuck-at-1 faults. For that matter, a plan was devised that is based on figure 6.10.

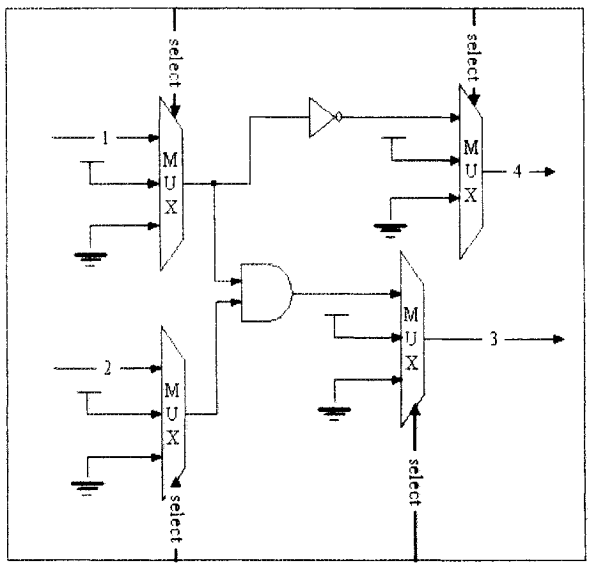


Figure 6.10 The Fault Injection Scheme in Hardware.

As we can see in the figure, every wire now has a multiplexer introduced within it, which allows us to either run the wire as is, or inject stuck-at-0 or stuck-at-1 faults. If the value of the select signals of any multiplexer are at "00", then we will run the wire as is, while if the values are at "01", then we will inject a stuck-at-1 fault indicated by the logical 1 value coming into the multiplexer, and if the values are at "10", then we will inject a stuck-at-0 fault indicated by the logical 0 value coming into the multiplexer. Finally, if the values are at "11", then we will again assume normal operation of the wire.

The WireIn signal is already present in our MUT, but the introduction of the multiplexer-based design, also adds the select signals that have to be controlled in order to inject a specific fault at a particular wire.

6.5.6 Implementation notes

The MUT of Figure 6.7 was previously designed and analyzed in order to determine both detectable and undetectable (redundant) faults.

The truth table of our implemented MUT is shown in table 6.7.

$x1$	$x2$	$x3$	$z1$	$z2$	$z3$	$z4$
0	0	0	1	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	0	0
0	1	1	1	0	1	0
1	0	0	1	0	1	0
1	0	1	1	0	1	0
1	1	0	1	0	1	0
1	1	1	1	0	1	0

Table 6.7 The MUT Truth Table.

The total number of s-at-logic faults that could be detected at the outputs of the MUT (collapsed faults) is 56;

The input test set that is needed to detect all detectable faults includes 4 input test vectors;

The percentage of fault coverage is 50%;

The number of identified redundant faults is 28;

The number of aborted faults caused by program limitations or errors is 0.

For technical reasons, we decided not insert a muliplexer (a fault injection) where there is a *fanout*. Since a wire can't physically be at the same time at high logic and low logic values, otherwise the chip would get burned. Therefore only 19 out of the 28 s-at-logic faults were injected into the MUT and detected at the outputs. But in real life, such a situation might occur.

Deterministic Compacted Input Test Patterns (MUT)				Pseudorandom Input Test Patterns (MUT)			
Test #1	111	1010	7 s-at-faults detected	Test #1	000	1000	12 s-at-faults detected
Test #2	011	1010	2 s-at-faults detected	Test #2	100	1010	4 s-at-faults detected
Test #3	000	1000	8 s-at-faults detected	Test #3	101	1010	0 s-at-faults detected
Test #4	110	1010	2 s-at-faults detected	Test #4	101	1010	0 s-at-faults detected
				Test #5	111	1010	1 s-at-faults detected
				Test #6	001	1010	1 s-at-faults detected
				Test #7	100	1010	0 s-at-faults detected
				Test #8	101	1010	0 s-at-faults detected
				Test #9	110	1010	0 s-at-faults detected
				Test #10	100	1010	0 s-at-faults detected
				Test #11	101	1010	0 s-at-faults detected
				Test #12	001	1010	0 s-at-faults detected
				Test #13	111	1010	0 s-at-faults detected
				Test #14	000	1000	0 s-at-faults detected
				Test #15	010	1000	0 s-at-faults detected
				Test #16	100	1010	0 s-at-faults detected
				Test #17	110	1010	0 s-at-faults detected
				Test #18	101	1010	0 s-at-faults detected
				Test #19	010	1000	0 s-at-faults detected
				Test #20	001	1010	0 s-at-faults detected
				Test #21	000	1000	0 s-at-faults detected
				Test #22	000	1000	0 s-at-faults detected
				Test #23	000	1000	0 s-at-faults detected
				Test #24	110	1010	0 s-at-faults detected
				Test #25	101	1010	0 s-at-faults detected
				Test #26	000	1000	0 s-at-faults detected
				Test #27	100	1010	0 s-at-faults detected
				Test #28	101	1010	0 s-at-faults detected
				Test #29	101	1010	0 s-at-faults detected
				Test #30	100	1010	0 s-at-faults detected
				Test #31	011	1010	1 s-at-faults detected
				Test #32	101	1010	0 s-at-faults detected

Table 6.8 Input Test Set, Fault-Free Output, and the Corresponding Detected Stuck-at-Logic Faults for the Preliminary MUT.

Deterministic Compacted Input Test Patterns (MUT+COMP)	Pseudorandom Input Test Patterns (MUT+COMP)
Test #1 110 1 6 s-at-faults detected Test #2 011 1 2 s-at-faults detected Test #3 111 1 1 s-at-faults detected Test #4 000 0 10 s-at-faults detected	Test #1 000 0 7 s-at-faults detected Test #2 100 1 9 s-at-faults detected Test #3 101 1 0 s-at-faults detected Test #4 101 1 0 s-at-faults detected Test #5 111 1 1 s-at-faults detected Test #6 001 1 1 s-at-faults detected Test #7 100 1 0 s-at-faults detected Test #8 101 1 0 s-at-faults detected Test #9 110 1 0 s-at-faults detected Test #10 100 1 0 s-at-faults detected Test #11 101 1 0 s-at-faults detected Test #12 001 1 0 s-at-faults detected Test #13 111 1 0 s-at-faults detected Test #14 000 0 0 s-at-faults detected Test #15 010 0 0 s-at-faults detected Test #16 100 1 0 s-at-faults detected Test #17 110 1 0 s-at-faults detected Test #18 101 1 0 s-at-faults detected Test #19 010 0 0 s-at-faults detected Test #20 001 1 0 s-at-faults detected Test #21 000 0 0 s-at-faults detected Test #22 000 0 0 s-at-faults detected Test #23 000 0 0 s-at-faults detected Test #24 110 1 0 s-at-faults detected Test #25 101 1 0 s-at-faults detected Test #26 000 0 0 s-at-faults detected Test #27 100 1 0 s-at-faults detected Test #28 101 1 0 s-at-faults detected Test #29 101 1 0 s-at-faults detected Test #30 100 1 0 s-at-faults detected Test #31 011 1 1 s-at-faults detected Test #32 101 1 0 s-at-faults detected

Table 6.9 Input Test Set, Fault-Free Output, and the Corresponding Detected Stuck-at-Logic Faults for the Preliminary MUT + COMPACTORS.

As shown in Table 6.8 and Table 6.9, the 19 stuck-at-logic injected faults are detected on the JBits implementation using both the MUT and MUT + COMPACTORS in deterministic and pseudo-random testing modes.

The hardware verification of the zero-aliasing space compaction model was realized in various different environments, while running on hardware.

CHAPTER 7

EXPERIMENTAL RESULTS

To demonstrate the feasibility of the proposed new space compaction schemes ("pairwise merger under stochastic independence of line errors" and "pairwise merger under stochastic dependence of line errors"), independent simulations were performed on various ISCAS 85 combinational and full scan ISCAS 89 sequential benchmark circuits. We have used ATALANTA [40] (fault simulation program developed at Virginia Polytechnic Institute and State University) to generate the fault-free output sequences needed to construct our space compactor circuits and to test the benchmark circuits using reduced test sets accompanied with a random testing session, FSIM [39] and HOPE [19] fault simulation programs to generate pseudorandom test sets needed to test various combinational and sequential circuits respectively, COMPACTEST [42] program to generate reduced test sets that detect most detectable single stuck-at faults for all benchmark circuits. For each circuit, we determined the number of test vectors used to construct the compaction tree, the CPU time taken to construct the compactor, the number of applied test vectors, the simulation CPU time, and the percentage of fault coverage by running ATALANTA, FSIM and HOPE programs on a SPARC ULTRA1 SUN workstation, and COMPACTEST on an IBM AIX machine. The results are listed in the figures and tables that follow. The CPU times needed to construct the compactors for all different ISCAS 85 and ISCAS 89 benchmark circuits on a SUN SPARC ULTRA1 workstation were in the range of 2.0-152.0 seconds.

For comparison purposes, we used a parity tree space compactor, composed of XOR gates, that propagates all errors that appear on an odd number of inputs and is, therefore, considered ideal for space compaction. With the parity tree space compactor as reference, we have simulated some combinational benchmark circuits to demonstrate the feasibility of the proposed scheme of constructing a compaction tree using the concept of Hamming distance and sequence weight.

Figure 7.1, Figure 7.2, and Figure 7.3 show the fault coverage for all ISCAS 85 benchmark circuits without compactors, with parity tree compactors, and with our pairwise line merger compactors, when stochastic independence of single and double line errors is assumed, and using ATALANTA, FSIM, and COMPACTEST, respectively.

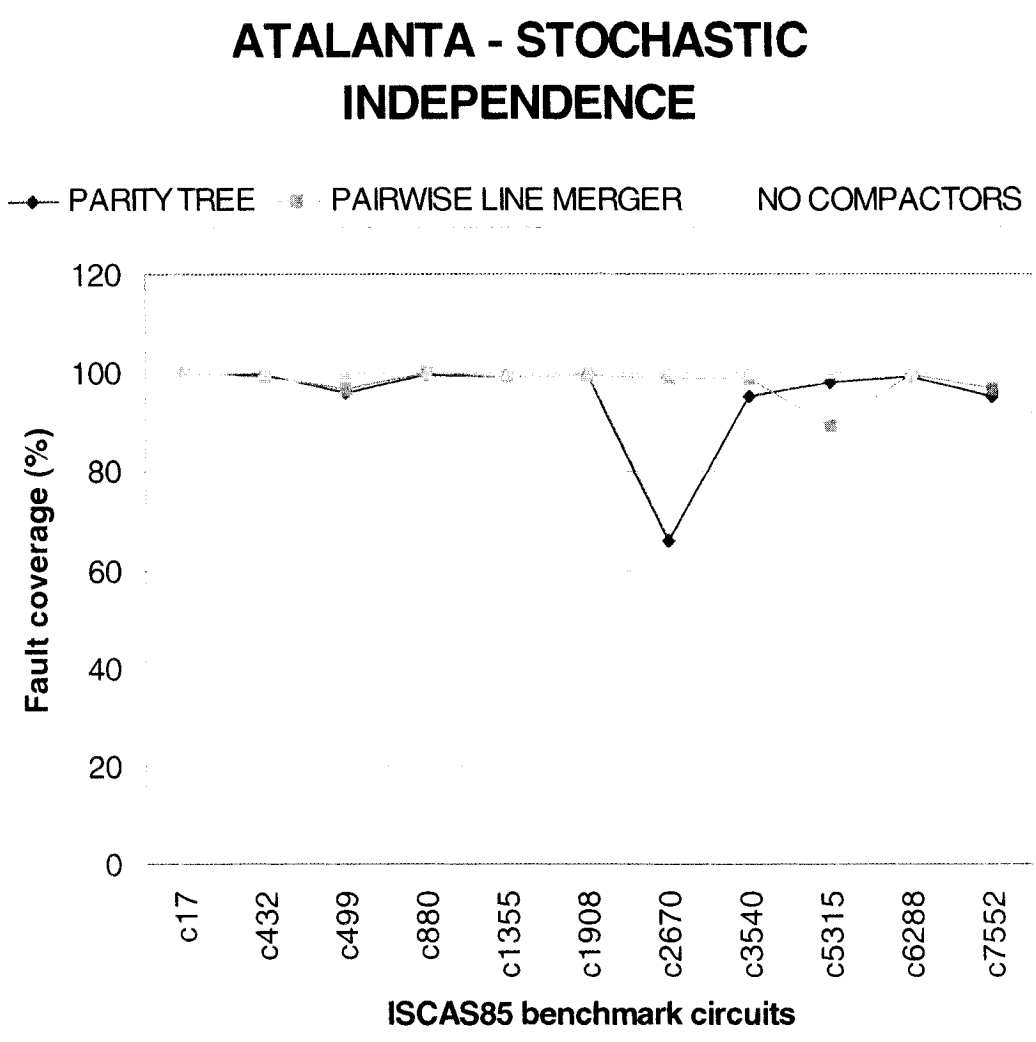


Figure 7.1 Fault Coverage Using ATALANTA – Parity Tree vs. Pairwise Line Merger Under Stochastic Independence of Line Errors.

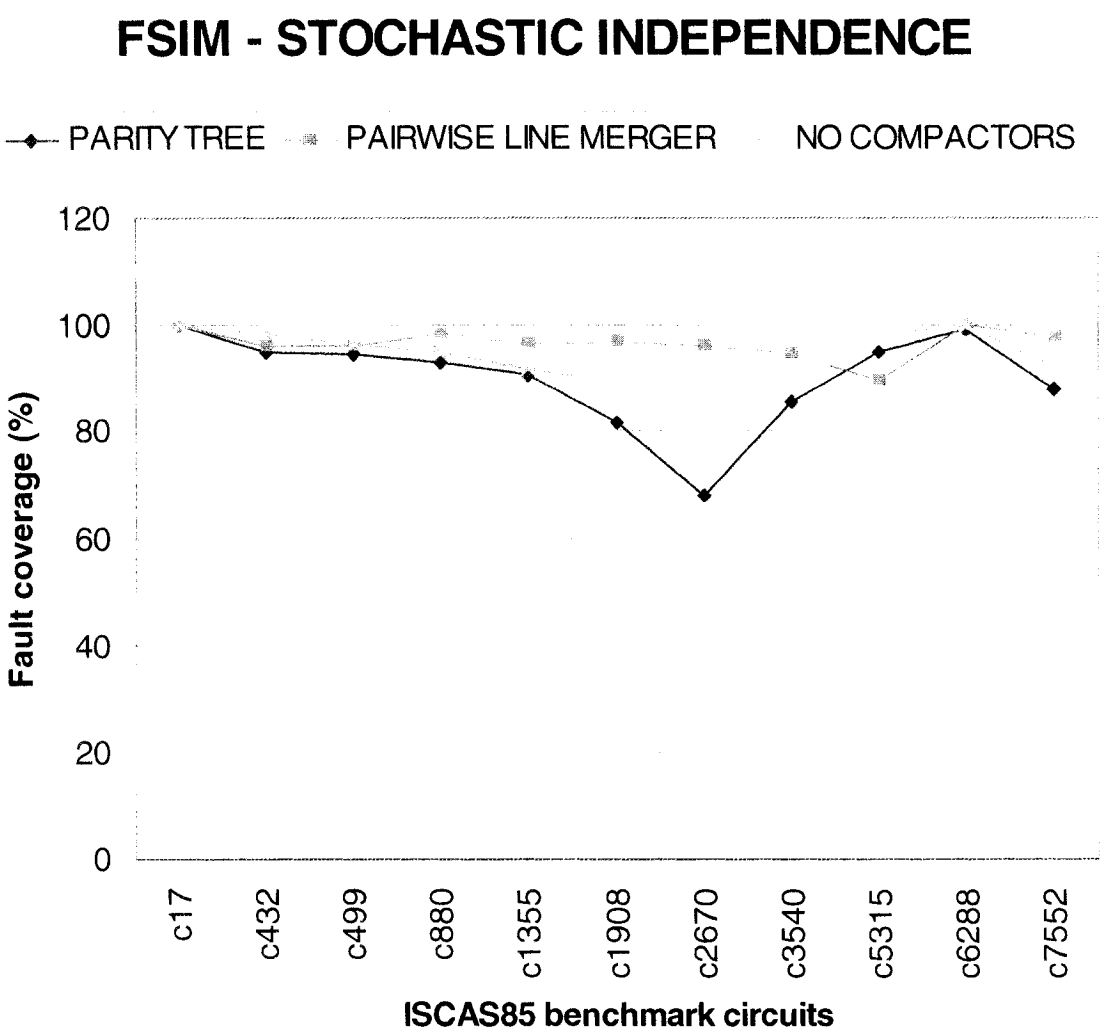


Figure 7.2 Fault Coverage Using FSIM – Parity Tree vs. Pairwise Line Merger Under Stochastic Independence of Line Errors.

Figure 7.2 shows the simulation results for all benchmark circuits using pseudo random testing (FSIM) with the same test sets (224 input test vectors) in all cases when stochastic independence of single and double line errors are assumed.

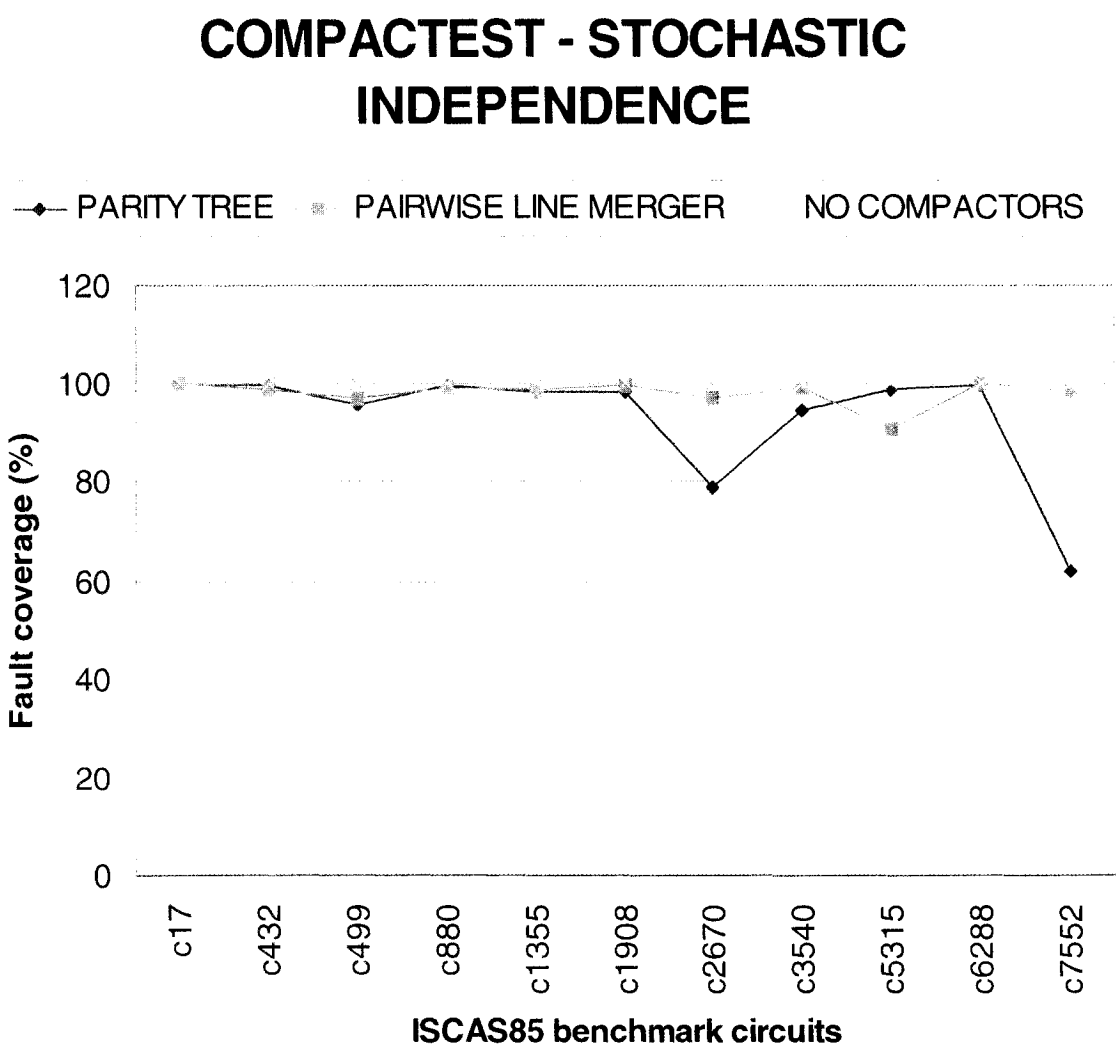


Figure 7.3 Fault Coverage Using COMPACTEST – Parity Tree vs. Pairwise Line Merger Under Stochastic Independence of Line Errors.

As shown in the above figures, we also simulated all ISCAS 85 benchmark circuits with a parity tree space compactor using ATALANTA, FSIM, and COMPACTEST, respectively.

For each circuit, we determined the simulation CPU time by running ATALANTA, and FSIM programs on a SPARC ULTRA1 SUN workstation, and COMPACTEST on an IBM AIX machine. The results are shown graphically in Figure 7.4.

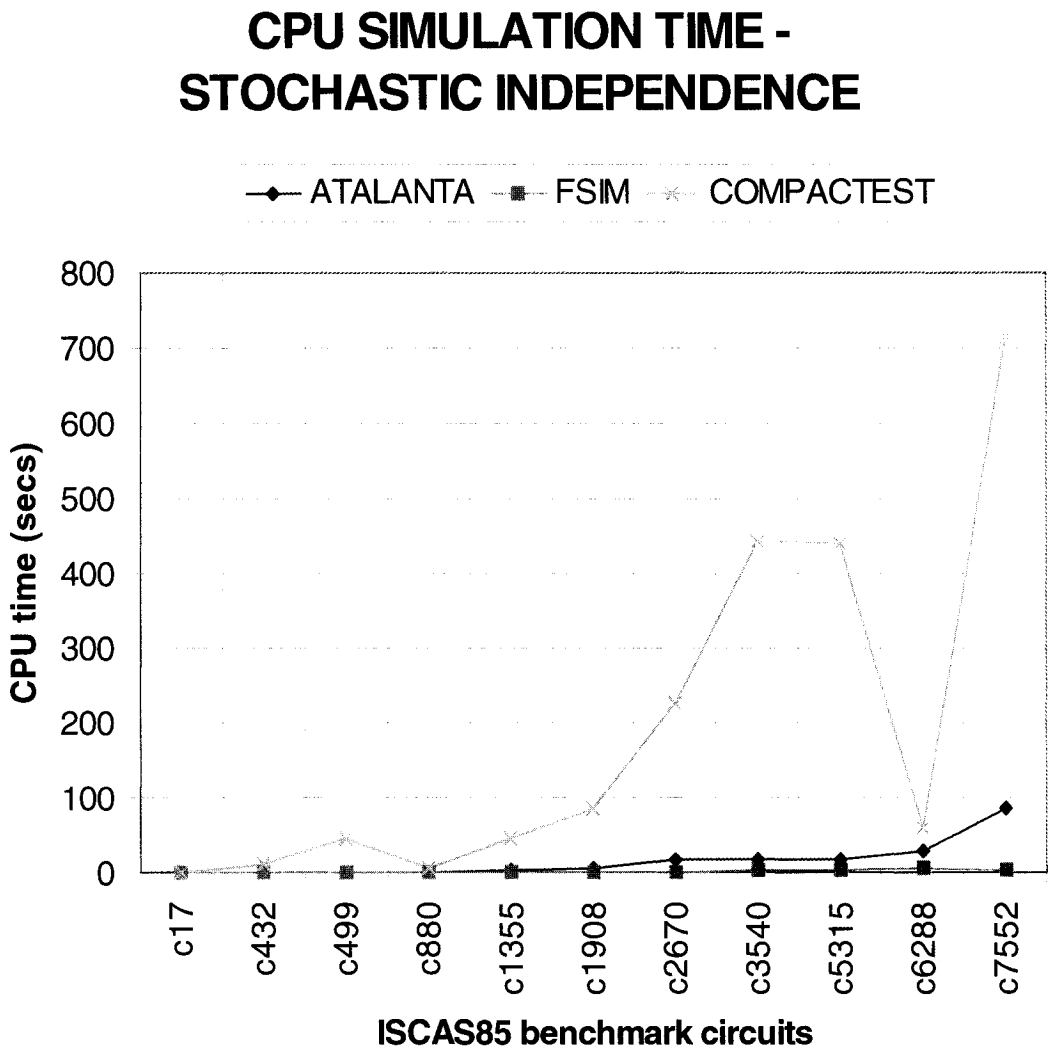


Figure 7.4 CPU Simulation Time – Pairwise Line Merger Under Stochastic Independence of Line Errors.

Figure 7.5, Figure 7.6, and Figure 7.7 show the fault coverage for all ISCAS 85 benchmark circuits without compactors, with parity tree compactors, and with our pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, FSIM, and COMPACTEST respectively, for the following probability values of single and double line errors: $((S_1 , S_2)$ equals $(0.66 , 0.33))$.

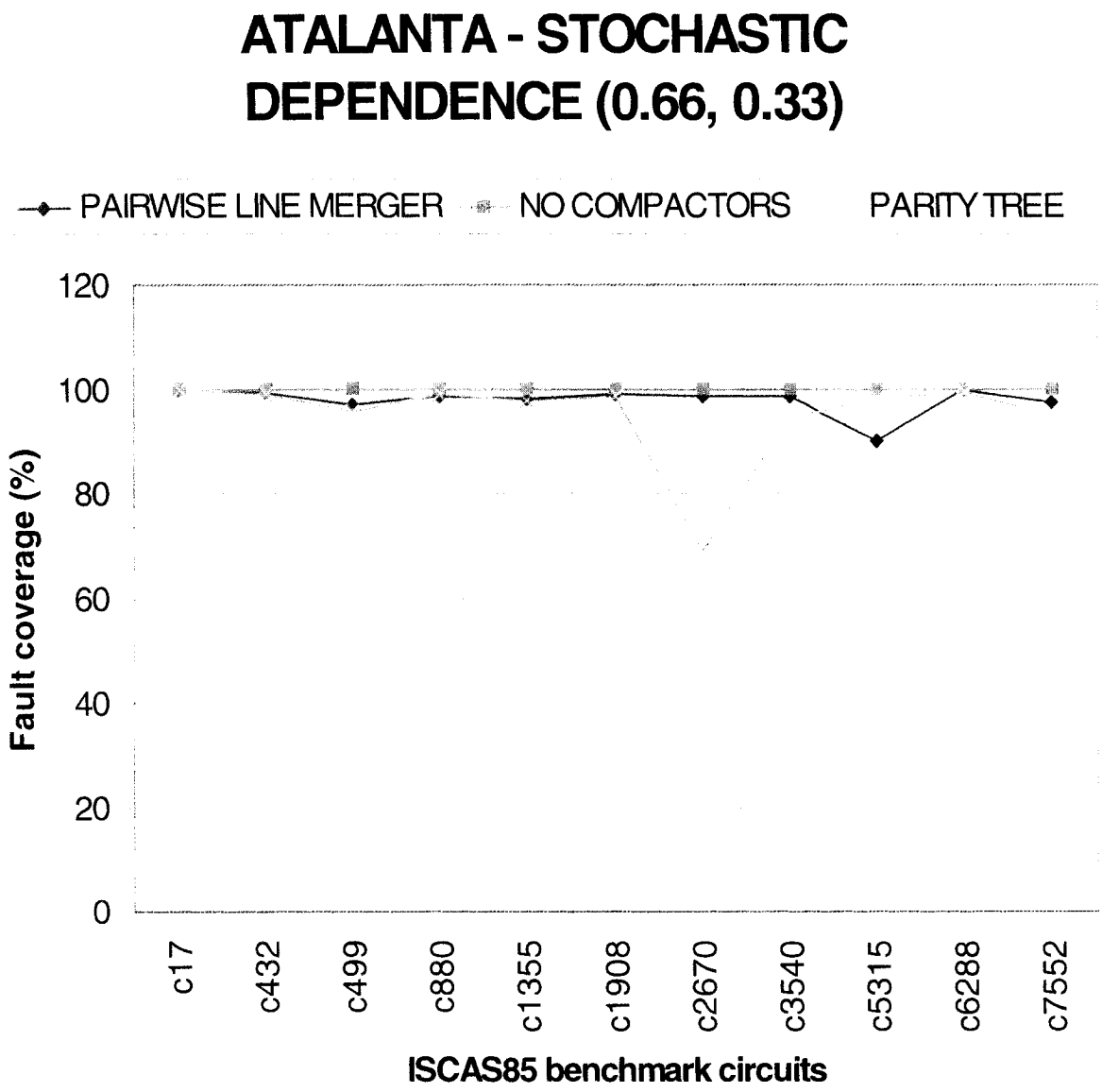


Figure 7.5 Fault Coverage Using ATALANTA – Parity Tree vs. Pairwise Line Merger Under Stochastic Dependence of Line Errors ($S_1 = 0.66$, $S_2 = 0.33$).

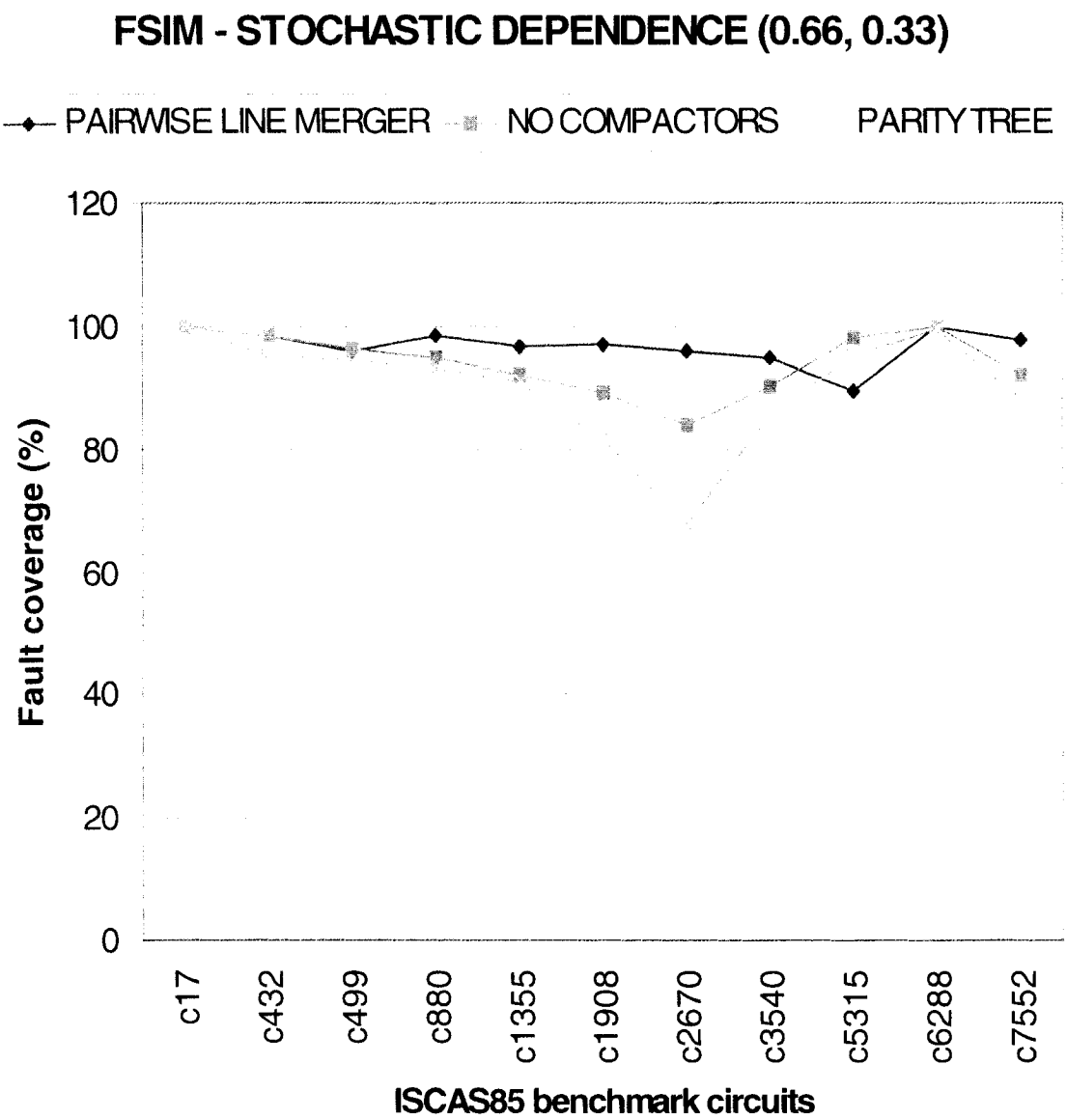


Figure 7.6 Fault Coverage Using FSIM – Parity Tree vs. Pairwise Line Merger Under Stochastic Dependence of Line Errors ($S_1 = 0.66$, $S_2 = 0.33$, and input test vectors = 224).

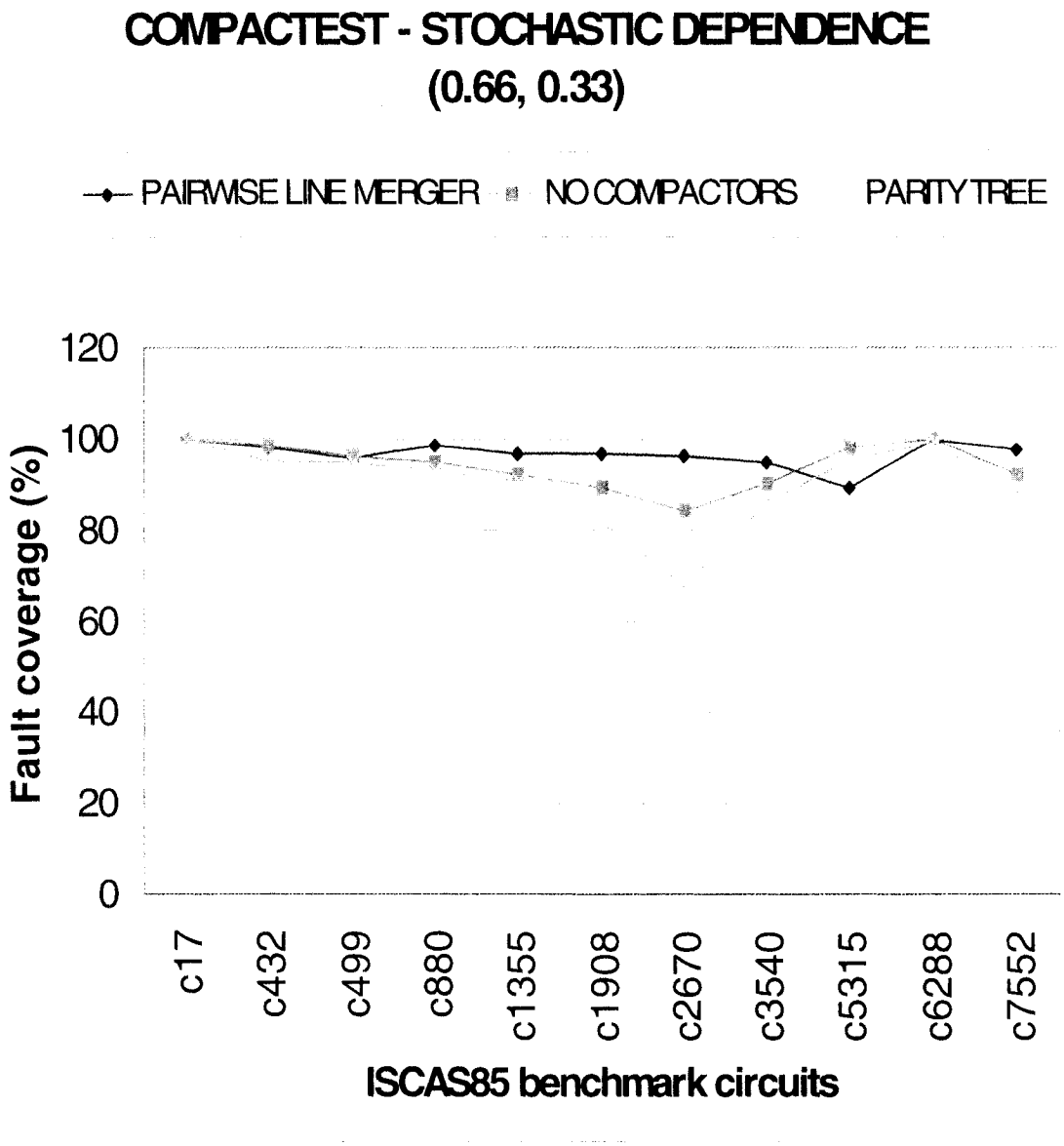


Figure 7.7 Fault Coverage Using COMPACTEST – Parity Tree vs. Pairwise Line Merger Under Stochastic Dependence of Line Errors ($S_1 = 0.66$, $S_2 = 0.33$).

We also determined the simulation CPU time was also determined, for all circuits, using ATALANTA, FSIM and COMPACTEST. The results are shown in Figure 7.8.

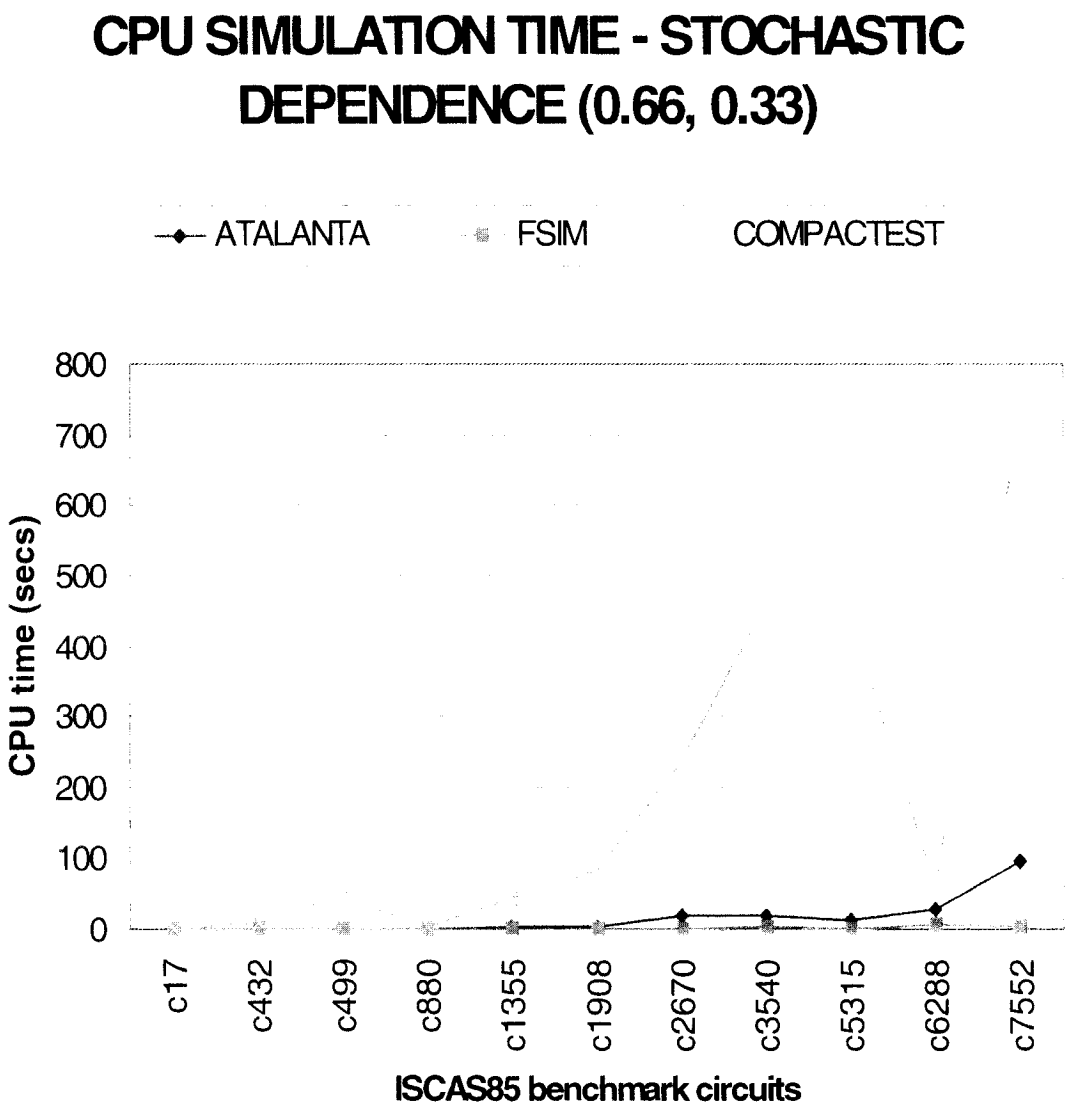


Figure 7.8 CPU Simulation Time – Pairwise Line Merger Under Stochastic Dependence of Line Errors ($S_1 = 0.66$, $S_2 = 0.33$).

We also estimated the hardware overhead of the compressor for different benchmark circuits and found it to be as small as 1-5 % in most cases and within 15% for three benchmark circuits. Table 7.1 shows the hardware overhead estimates for all ISCAS 85 benchmark circuits, also shown in Figure 7.9. To estimate the hardware overhead, we used the ratio of the weighted gate count

metric (i.e. average fanins x number of gates) of the space compressor and that of the total circuit comprised of the CUT and the space compactor.

Compactor Circuit For	Total No. of Fanin in The Compactor	Total No. of Gates in The Compactor	Average Fanin in The Compactor	Total No. of Gates in The CUT	Average Fanin in The CUT	Hardware Overhead (%)
c17	2	1	2.00	6	2.00	14.28
c432	12	6	2.00	160	2.10	3.44
c499	62	31	2.00	202	2.02	13.19
c880	50	25	2.00	383	1.90	6.42
c1355	62	31	2.00	546	1.95	5.50
c1908	48	24	2.00	880	1.70	3.10
c2670	278	139	2.00	1269	1.64	11.78
c3540	42	21	2.00	1669	1.76	1.40
c5315	244	122	2.00	2307	1.90	5.27
c6288	62	31	2.00	2416	1.99	1.27
c7552	214	107	2.00	3513	1.75	3.36

Table 7.1 Hardware Overheads for the ISCAS 85 Benchmark Circuits.

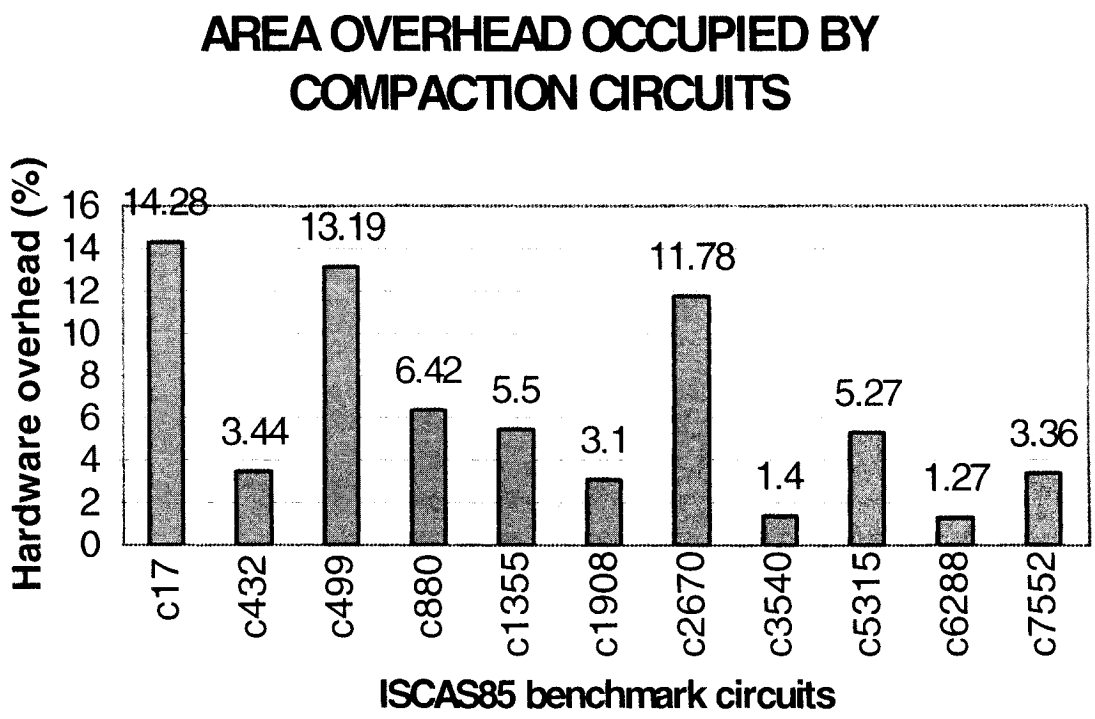


Figure 7.9 The Percentage of Area Overhead (Pairwise Merger).

Table 7.2, and Table 7.3 show the fault coverage for all ISCAS 89 full scan benchmark circuits without compactors, and with our pairwise line merger compactors, when stochastic independence of single and double line errors is assumed, and using ATALANTA, and FSIM respectively.

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit Name	Total No. of Injected Faults	No. of Applied Test Vectors (after compaction)	Fault Coverage (%) (without space compactors)	CPU Simulation Time (secs)	Fault Coverage (%) (with space compactors)
s27f	38	8	100.000	0.017	89.474*
s208f	219	31	100.000	0.067	92.237*
s298f	332	42	100.000	0.067	97.590*
s344f	386	33	100.000	0.117	97.668*
s349f	330	29	100.000	0.167	93.333
s382f	431	48	100.000	0.100	99.304*
s386f	386	59	100.000	0.217	68.912*
s400f	418	44	100.000	0.100	97.368
s420f	434	73	100.000	0.200	96.083*
s444f	460	48	100.000	0.150	94.348
s510f	578	66	100.000	0.167	96.886*
s526f	595	75	100.000	0.183	95.966*
s526f	554	68	100.000	0.200	97.473
s641f	507	65	100.000	0.267	93.294*
s713f	543	73	100.000	0.300	96.317
s820f	858	120	100.000	0.750	69.231*
s832f	856	84	100.000	1.050	49.533
s838f	861	111	100.000	1.117	86.179*
s953f	1139	100	100.000	1.017	87.884*
s1196f	1270	146	100.000	1.967	77.008*
s1238f	1286	113	100.000	2.567	74.572
s1423f	1501	87	100.000	0.917	98.801
s1488f	1500	76	100.000	2.100	56.33*
s1494f	1494	110	100.000	2.250	59.906
s5378f	4511	381	100.000	31.650	92.286
s9234f	6475	471	100.000	184.967	97.514
s13207f	**Memory, CPU time and Disk usage limits are exceeded**				
s15850	**Memory, CPU time and Disk usage limits are exceeded**				
s35932f	**Memory, CPU time and Disk usage limits are exceeded**				
s38417f	**Memory, CPU time and Disk usage limits are exceeded!**				
s38584	**Memory, CPU time and Disk usage limits are exceeded!**				

Table 7.2 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using ATALANTA (Compacted Input Test Sets) – Pairwise Line Merger and Assuming Stochastic Independence of Single and Double Line Errors.

Circuit Name	Total No. of Injected Faults	No. of Applied Test Vectors	Fault Coverage (%) (without space compactors)	CPU Simulation Time (secs)	Fault Coverage (%) (with space compactors)
s27f	38	224	100.00	0.033	89.474*
s208f	219	224	86.977	0.033	84.475*
s298f	332	224	99.675	0.033	93.976*
s344f	386	224	99.691	0.050	96.373*
s349f	330	224	100.000	0.067	91.818
s382f	431	224	96.742	0.050	93.503*
s386f	386	224	75.521	0.067	59.067*
s400f	418	224	99.043	0.067	93.062
s420f	434	224	81.163	0.050	74.885*
s444f	460	224	98.913	0.067	87.609
s510f	578	224	95.745	0.067	94.291*
s526f	595	224	86.643	0.050	78.487*
s526f	554	224	86.643	0.067	82.130
s641f	507	224	90.713	0.083	85.602*
s713f	543	224	92.081	0.083	86.188
s820f	858	224	79.882	0.067	51.632*
s832f	856	224	73.598	0.083	38.084
s838f	861	224	77.130	0.100	57.027*
s953f	1139	224	77.386	0.117	64.267*
s1196f	1270	224	78.905	0.117	53.937*
s1238f	1286	224	76.750	0.150	58.554
s1423f	1501	224	93.804	0.167	87.875
s1488f	1500	224	84.926	0.133	45.733*
s1494f	1494	224	87.282	0.167	46.118
s5378f	4511	224	85.746	0.717	64.243
s9234f	6475	224	63.413	1.700	55.552
s13207f	**Memory, CPU time and Disk usage limits are exceeded**				
s15850f	**Memory, CPU time and Disk usage limits are exceeded**				
s35932f	**Memory, CPU time and Disk usage limits are exceeded**				
s38417f	**Memory, CPU time and Disk usage limits are exceeded**				
s38584	**Memory, CPU time and Disk usage limits are exceeded**				

Table 7.3 Fault Coverage for ISCAS 89 Benchmark Circuits Using FSIM (Pseudorandom Testing) – Pairwise Line Merger and Assuming Stochastic Independence of Single and Double Line Errors.

Table 7.4, and Table 7.5 show the fault coverage for all ISCAS 89 full scan benchmark circuits without compactors, and with our pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, and FSIM respectively, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.66 , 0.33)).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit Name	Total No. of Injected Faults	No. of Applied Test Vectors (after compaction)	S1	S2	Fault Coverage (%) (without space compactors)	CPU Simulation Time (secs)	Fault Coverage (%) (with space compactors)
s27f	38	6	0.66	0.33	100.000	0.017	89.474*
s208f	219	33	0.66	0.33	100.000	0.067	92.237*
s298f	332	49	0.66	0.33	100.000	0.067	97.590*
s344f	386	30	0.66	0.33	100.000	0.133	97.668*
s349f	330	29	0.66	0.33	100.000	0.133	93.333
s382f	431	47	0.66	0.33	100.000	0.100	99.304*
s386f	386	57	0.66	0.33	100.000	0.217	68.912*
s400f	418	44	0.66	0.33	100.000	0.133	97.368
s420f	434	66	0.66	0.33	100.000	0.200	96.083*
s444f	460	52	0.66	0.33	100.000	0.150	94.348
s510f	578	66	0.66	0.33	100.000	0.150	96.886*
s526f	595	70	0.66	0.33	100.000	0.250	95.966*
s526f	554	67	0.66	0.33	100.000	0.267	97.473
s641f	507	63	0.66	0.33	100.000	0.283	93.294*
s713f	543	67	0.66	0.33	100.000	0.333	96.317
s820f	858	118	0.66	0.33	100.000	0.833	69.231*
s832f	856	86	0.66	0.33	100.000	1.117	49.533
s838f	861	111	0.66	0.33	100.000	0.933	86.179*
s953f	1139	97	0.66	0.33	100.000	0.867	87.972*
s1196f	1270	154	0.66	0.33	100.000	1.800	77.008*
s1238f	1286	115	0.66	0.33	100.000	2.850	74.495
s1423f	1501	86	0.66	0.33	100.000	1.017	98.934
s1488f	1500	83	0.66	0.33	100.000	1.983	56.333*
s1494f	1494	111	0.66	0.33	100.000	2.333	59.906
s5378f	4511	377	0.66	0.33	100.000	27.817	92.241
s9234	6475	472	0.66	0.33	100.000	173.633	97.560
s13207	Memory, CPU time and Disk usage limits are exceeded						
s15850	Memory, CPU time and Disk usage limits are exceeded						
s35932f	Memory, CPU time and Disk usage limits are exceeded						
s38417f	Memory, CPU time and Disk usage limits are exceeded						
s38584	Memory, CPU time and Disk usage limits are exceeded						

Table 7.4 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using ATALANTA (Compacted Input Test Sets) – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors.

Circuit Name	Total No. of Injected Faults	No. of Applied Test Vectors	S1	S2	Fault Coverage (%) (without space compactors)	CPU Simulation Time (secs)	Fault Coverage (%) (with space compactors)
s27f	38	224	0.66	0.33	100.00	0.033	89.474*
s208f	219	224	0.66	0.33	86.977	0.033	79.909*
s298f	332	224	0.66	0.33	99.675	0.033	93.373*
s344f	386	224	0.66	0.33	99.691	0.050	97.409*
s349f	330	224	0.66	0.33	100.000	0.067	93.333
s382f	431	224	0.66	0.33	96.742	0.067	96.752*
s386f	386	224	0.66	0.33	75.521	0.050	52.332*
s400f	418	224	0.66	0.33	99.043	0.050	94.019
s420f	434	224	0.66	0.33	81.163	0.050	66.129*
s444f	460	224	0.66	0.33	98.913	0.067	88.043
s510f	578	224	0.66	0.33	95.745	0.067	92.907*
s526f	595	224	0.66	0.33	86.643	0.067	78.151*
s526f	554	224	0.66	0.33	86.643	0.067	82.671
s641f	507	224	0.66	0.33	90.713	0.067	86.391*
s713f	543	224	0.66	0.33	92.081	0.083	86.556
s820f	858	224	0.66	0.33	79.882	0.067	49.184*
s832f	856	224	0.66	0.33	73.598	0.083	39.486
s838f	861	224	0.66	0.33	77.130	0.117	59.698*
s953f	1139	224	0.66	0.33	77.386	0.117	63.740*
s1196f	1270	224	0.66	0.33	78.905	0.133	53.307*
s1238f	1286	224	0.66	0.33	76.750	0.133	60.420
s1423f	1501	224	0.66	0.33	93.804	0.183	89.474
s1488f	1500	224	0.66	0.33	84.926	0.133	39.133*
s1494f	1494	224	0.66	0.33	87.282	0.150	47.122
s5378f	4511	224	0.66	0.33	85.746	0.717	63.977
s9234	6475	224	0.66	0.33	63.413	1.717	57.575
s13207	**Memory, CPU time and Disk usage limits are exceeded**						
s15850	**Memory, CPU time and Disk usage limits are exceeded**						
s35932f	**Memory, CPU time and Disk usage limits are exceeded**						
s38417f	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.5 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using FSIM (Pseudorandom Testing) – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors.

Table 7.6, and Table 7.7 show the fault coverage for all ISCAS 89 full scan benchmark circuits without compactors, and with our pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, and FSIM respectively, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.90 , 0.10)).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	38	6	0.90	0.10	100.000	0.000	89.474*
S208f	227	34	0.90	0.10	100.000	0.067	99.119*
S298f	346	38	0.90	0.10	100.000	0.067	98.555*
S344f	392	27	0.90	0.10	100.000	0.117	97.959*
S349f	330	27	0.90	0.10	100.000	0.133	97.576
S382f	447	40	0.90	0.10	100.000	0.117	98.210*
S386f	400	72	0.90	0.10	100.000	0.133	96.750*
S400f	418	36	0.90	0.10	100.000	0.150	97.129
S420f	440	66	0.90	0.10	100.000	0.200	98.864*
S444f	460	39	0.90	0.10	100.000	0.150	92.826
S510f	584	66	0.90	0.10	100.000	0.150	98.973*
S526f	607	64	0.90	0.10	100.000	0.167	99.012*
S526f	554	66	0.90	0.10	100.000	0.167	99.097
S641f	545	61	0.90	0.10	100.000	0.350	99.083*
S713f	543	57	0.90	0.10	100.000	0.300	97.053
S820f	868	115	0.90	0.10	100.000	0.667	72.465*
S832f	856	87	0.90	0.10	100.000	0.933	56.776
S838f	871	116	0.90	0.10	100.000	0.617	96.096*
S953f	1163	95	0.90	0.10	100.000	0.833	94.153*
S1196f	1296	148	0.90	0.10	100.000	1.183	96.682*
S1238f	1286	131	0.90	0.10	100.000	1.867	93.701
S1423f	1501	78	0.90	0.10	100.000	0.867	99.534
S1488f	1522	133	0.90	0.10	100.000	1.067	95.335*
S1494f	1494	130	0.90	0.10	100.000	1.067	94.645
S5378f	4511	341	0.90	0.10	100.000	41.867	94.214
S9234f	6475	458	0.90	0.10	100.000	168.750	98.085
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.6 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using ATALANTA – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for (S_1 = 0.90, S_2 = 0.10).

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	38	224	0.90	0.10	100.00	0.017	89.474*
S208f	227	224	0.90	0.10	86.977	0.033	85.903*
S298f	346	224	0.90	0.10	99.675	0.033	96.532*
S344f	392	224	0.90	0.10	99.691	0.050	97.194*
S349f	330	224	0.90	0.10	100.000	0.050	96.970
S382f	447	224	0.90	0.10	96.742	0.050	96.421*
S386f	400	224	0.90	0.10	75.521	0.050	82.250*
S400f	418	224	0.90	0.10	99.043	0.050	93.541
S420f	440	224	0.90	0.10	81.163	0.067	80.000*
S444f	460	224	0.90	0.10	98.913	0.067	85.870
S510f	584	224	0.90	0.10	95.745	0.067	96.747*
S526f	607	224	0.90	0.10	86.643	0.050	88.138*
S526f	554	224	0.90	0.10	86.643	0.050	81.769
S641f	545	224	0.90	0.10	90.713	0.083	90.275*
S713f	543	224	0.90	0.10	92.081	0.083	87.845
S820f	868	224	0.90	0.10	79.882	0.067	53.456*
S832f	856	224	0.90	0.10	73.598	0.083	46.379
S838f	871	224	0.90	0.10	77.130	0.100	68.083*
S953f	1163	224	0.90	0.10	77.386	0.117	71.453*
S1196f	1296	224	0.90	0.10	78.905	0.117	71.682*
S1238f	1286	224	0.90	0.10	76.750	0.133	69.440
S1423f	1501	224	0.90	0.10	93.804	0.183	92.871
S1488f	1522	224	0.90	0.10	84.926	0.133	83.311*
S1494f	1494	224	0.90	0.10	87.282	0.150	81.794
S5378f	4511	224	0.90	0.10	85.746	0.733	58.524
S9234f	6475	224	0.90	0.10	63.413	1.633	60.664
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.7 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using FSIM – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.90$, $S_2 = 0.10$).

Table 7.8, and Table 7.9 show the fault coverage for all ISCAS 89 full scan benchmark circuits without compactors, and with our pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, and FSIM respectively, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.10 , 0.90)).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	32	8	0.10	0.90	100.000	0.017	81.250*
S208f	215	35	0.10	0.90	100.000	0.100	85.116*
S298f	308	48	0.10	0.90	100.000	0.133	74.675*
S344f	342	57	0.10	0.90	100.000	0.117	96.199*
S349f	330	39	0.10	0.90	100.000	0.250	72.424
S382f	399	53	0.10	0.90	100.000	0.183	73.183*
S386f	384	44	0.10	0.90	100.000	0.317	57.812*
S400f	418	63	0.10	0.90	100.000	0.300	69.139
S420f	430	70	0.10	0.90	100.000	0.267	95.349*
S444f	460	62	0.10	0.90	100.000	0.300	74.348
S510f	564	45	0.10	0.90	100.000	0.483	63.652*
S526f	555	112	0.10	0.90	100.000	0.500	85.045*
S526f	554	92	0.10	0.90	100.000	0.433	74.007
S641f	545	50	0.10	0.90	100.000	0.067	96.881*
S713f	543	104	0.10	0.90	100.000	0.567	93.923
S820f	850	60	0.10	0.90	100.000	1.017	33.882*
S832f	856	45	0.10	0.90	100.000	1.217	31.075
S838f	857	105	0.10	0.90	100.000	1.917	77.363*
S953f	1079	148	0.10	0.90	100.000	1.667	79.240*
S1196f	1242	145	0.10	0.90	100.000	3.550	64.090*
S1238f	1286	174	0.10	0.90	100.000	2.350	65.241
S1423f	1501	209	0.10	0.90	100.000	2.350	95.803
S1488f	1486	33	0.10	0.90	100.000	4.967	12.786*
S1494f	1494	40	0.10	0.90	100.000	6.533	23.159
S5378f	4511	445	0.10	0.90	100.000	72.417	63.977
S9234f	6475	1062	0.10	0.90	100.000	264.383	92.927
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.8 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using ATALANTA – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.10$, $S_2 = 0.90$).

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	32	224	0.10	0.90	100.00	0.017	81.250*
S208f	215	224	0.10	0.90	86.977	0.050	69.767*
S298f	308	224	0.10	0.90	99.675	0.033	37.987*
S344f	342	224	0.10	0.90	99.691	0.050	77.485*
S349f	330	224	0.10	0.90	100.000	0.050	48.788
S382f	399	224	0.10	0.90	96.742	0.050	58.647*
S386f	384	224	0.10	0.90	75.521	0.050	39.583*
S400f	418	224	0.10	0.90	99.043	0.067	45.455
S420f	430	224	0.10	0.90	81.163	0.083	58.605*
S444f	460	224	0.10	0.90	98.913	0.067	33.478
S510f	564	224	0.10	0.90	95.745	0.067	59.574*
S526f	555	224	0.10	0.90	86.643	0.083	41.982*
S526f	554	224	0.10	0.90	86.643	0.083	29.242
S641f	545	224	0.10	0.90	90.713	0.083	89.725*
S713f	543	224	0.10	0.90	92.081	0.100	54.696
S820f	850	224	0.10	0.90	79.882	0.067	24.000*
S832f	856	224	0.10	0.90	73.598	0.083	24.650
S838f	857	224	0.10	0.90	77.130	0.117	43.757*
S953f	1079	224	0.10	0.90	77.386	0.133	33.828*
S1196f	1242	224	0.10	0.90	78.905	0.150	29.308*
S1238f	1286	224	0.10	0.90	76.750	0.150	36.159
S1423f	1501	224	0.10	0.90	93.804	0.233	37.575
S1488f	1486	224	0.10	0.90	84.926	0.150	11.036*
S1494f	1494	224	0.10	0.90	87.282	0.183	12.115
S5378f	4511	224	0.10	0.90	85.746	0.983	0.200
S9234f	6475	224	0.10	0.90	63.413	2.333	8.309
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.9 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using FSIM – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.10$, $S_2 = 0.90$).

Table 7.10, and Table 7.11 show the fault coverage for all ISCAS 89 full scan benchmark circuits without compactors, and with our pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, and FSIM respectively, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.33 , 0.66)).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	34	8	0.33	0.66	100.000	0.000	85.294*
S208f	215	35	0.33	0.66	100.000	0.083	85.116*
S298f	308	60	0.33	0.66	100.000	0.100	82.468*
S344f	342	55	0.33	0.66	100.000	0.100	96.199*
S349f	330	41	0.33	0.66	100.000	0.133	78.788
S382f	399	42	0.33	0.66	100.000	0.233	62.155*
S386f	384	47	0.33	0.66	100.000	0.300	57.812*
S400f	418	65	0.33	0.66	100.000	0.217	84.689
S420f	430	70	0.33	0.66	100.000	0.250	95.349*
S444f	460	65	0.33	0.66	100.000	0.333	74.565
S510f	564	48	0.33	0.66	100.000	0.467	63.475*
S526f	555	86	0.33	0.66	100.000	0.583	71.892*
S526f	554	98	0.33	0.66	100.000	0.400	78.700
S641f	467	84	0.33	0.66	100.000	0.417	82.655*
S713f	543	105	0.33	0.66	100.000	0.767	82.689
S820f	850	70	0.33	0.66	100.000	0.783	36.706*
S832f	856	45	0.33	0.66	100.000	1.183	30.491
S838f	857	101	0.33	0.66	100.000	1.767	78.763*
S953f	1079	134	0.33	0.66	100.000	2.000	81.186*
S1196f	1242	123	0.33	0.66	100.000	4.200	56.924*
S1238f	1286	141	0.33	0.66	100.000	2.833	55.832
S1423f	1501	208	0.33	0.66	100.000	3.567	89.141
S1488f	1486	33	0.33	0.66	100.000	3.000	28.600*
S1494f	1494	36	0.33	0.66	100.000	7.867	19.679
S5378f	4511	443	0.33	0.66	100.000	88.383	68.233
S9234f	6475	559	0.33	0.66	100.000	304.450	63.429
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.10 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using ATALANTA – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.33$, $S_2 = 0.66$).

Circuit name	Total No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27f	34	224	0.33	0.66	100.00	0.033	85.294*
S208f	215	224	0.33	0.66	86.977	0.033	72.558*
S298f	308	224	0.33	0.66	99.675	0.050	50.000*
S344f	342	224	0.33	0.66	99.691	0.050	40.936*
S349f	330	224	0.33	0.66	100.000	0.050	63.333
S382f	399	224	0.33	0.66	96.742	0.067	36.591*
S386f	384	224	0.33	0.66	75.521	0.050	42.708*
S400f	418	224	0.33	0.66	99.043	0.067	47.129
S420f	430	224	0.33	0.66	81.163	0.050	60.233*
S444f	460	224	0.33	0.66	98.913	0.083	44.565
S510f	564	224	0.33	0.66	95.745	0.067	59.574*
S526f	555	224	0.33	0.66	86.643	0.067	32.432*
S526f	554	224	0.33	0.66	86.643	0.067	44.585
S641f	467	224	0.33	0.66	90.713	0.083	54.176*
S713f	543	224	0.33	0.66	92.081	0.117	40.147
S820f	850	224	0.33	0.66	79.882	0.067	28.353*
S832f	856	224	0.33	0.66	73.598	0.083	26.986
S838f	857	224	0.33	0.66	77.130	0.117	40.140*
S953f	1079	224	0.33	0.66	77.386	0.150	31.789*
S1196f	1242	224	0.33	0.66	78.905	0.133	29.630*
S1238f	1286	224	0.33	0.66	76.750	0.150	29.082
S1423f	1501	224	0.33	0.66	93.804	0.233	28.381
S1488f	1486	224	0.33	0.66	84.926	0.167	13.392*
S1494f	1494	224	0.33	0.66	87.282	0.200	17.470
S5378f	4511	224	0.33	0.66	85.746	1.033	7.182
S9234f	6475	224	0.33	0.66	63.413	2.367	3.907
S13207	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	**Memory, CPU time and Disk usage limits are exceeded**						
S35932f	**Memory, CPU time and Disk usage limits are exceeded**						
S38417f	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.11 Fault Coverage for ISCAS 89 Full Scan Benchmark Circuits Using FSIM – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.33$, $S_2 = 0.66$).

Table 7.12 shows the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic independence of single and double line errors is assumed, and using HOPE with compacted input test sets (5000 test patterns).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors (after compaction)	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	5000	100.000	------	------
S208	11	2	115	215	5000	34.419	0.317	34.419*
S298	3	6	136	310	5000	62.013	0.667	18.710*
S344	9	11	184	348	5000	90.351	0.467	85.632*
S349	9	11	185	356	5000	90.000	0.483	77.528*
S382	3	6	182	401	5000	12.281	0.883	10.474*
S386	7	7	172	386	5000	51.562	0.350	46.373*
S400	3	6	186	426	5000	12.028	0.933	9.859*
S420	19	2	231	430	5000	26.512	0.833	26.512*
S444	3	6	205	476	5000	11.181	1.250	0.840*
S510	19	7	236	564	5000	0.000	1.517	0.000*
S526	3	6	217	555	5000	8.649	1.350	3.423*
S641	35	24	433	481	5000	74.304	0.600	56.549*
S713	35	23	447	593	5000	72.117	0.717	56.998*
S820	18	19	312	852	5000	33.294	1.033	21.009*
S832	18	19	310	872	5000	32.299	1.050	20.528*
S838	25	2	457	857	5000	21.587	2.350	21.587*
S953	16	23	440	1079	5000	8.341	5.733	0.185*
S1196	14	14	561	1244	5000	76.651	1.183	56.109*
S1238	14	14	540	1357	5000	71.661	1.233	53.943*
S1423	17	5	748	1517	5000	35.380	3.983	25.313*
S1488	8	19	667	1490	5000	58.210	2.317	23.221*
S1494	8	19	661	1510	5000	57.171	1.950	25.166*
S5378	35	49	3050	4613	5000	51.119	16.567	30.111*
S9234	36	39	5844	3974	5000	10.386	53.167	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**				
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**				
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**				
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded!**				
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded!**				

Table 7.12 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with compacted input test sets (5000 test patterns) – Pairwise Line Merger and Assuming Stochastic Independence of Single and Double Line Errors.

Table 7.13 shows the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic independence of single and double line errors is assumed, and using HOPE with random testing (2000000 test patterns).

P.S. A (*) indicates that faults been injected into the CUT + COMPACTOR.

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	2000000	100.000	------	------
S208	11	2	115	215	2000000	45.581	92.467	45.581*
S298	3	6	136	310	2000000	85.390	128.733	57.097*
S344	9	11	184	348	2000000	96.199	147.900	93.103*
S349	9	11	185	356	2000000	95.714	138.483	87.640*
S382	3	6	182	401	2000000	13.283	327.633	10.474*
S386	7	7	172	386	2000000	71.094	44.650	66.580*
S400	3	6	186	426	2000000	13.443	347.717	9.859*
S420	19	2	231	430	2000000	31.628	267.750	31.628*
S444	3	6	205	476	2000000	16.245	436.583	4.412*
S510	19	7	236	564	2000000	0.000	581.050	0.000*
S526	3	6	217	555	2000000	11.892	515.650	3.423*
S641	35	24	433	481	2000000	86.081	155.200	75.676*
S713	35	23	447	593	2000000	81.583	209.933	64.587*
S820	18	19	312	852	2000000	50.118	320.500	39.906*
S832	18	19	310	872	2000000	49.080	326.167	39.220*
S838	25	2	457	857	2000000	24.387	888.733	24.387*
S953	16	23	440	1079	2000000	8.341	2302.000	0.185*
S1196	14	14	561	1244	2000000	98.953	121.833	91.801*
S1238	14	14	540	1357	2000000	93.948	153.383	85.041*
S1423	17	5	748	1517	2000000	66.535	808.383	64.074*
S1488	8	19	667	1490	2000000	76.918	674.450	32.282*
S1494	8	19	661	1510	2000000	76.096	627.933	38.477*
S5378	35	49	3050	4613	2000000	63.611	3745.833	53.761*
S9234	36	39	5844	3974	2000000	11.199	20651.333	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**				
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**				
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**				
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**				
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**				

Table 7.13 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Random Testing (Initial Random Number Generator Seed = 999) – Pairwise Line Merger and Assuming Stochastic Independence of Single and Double Line Errors.

Table 7.14, and Table 7.15 show the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using HOPE with deterministic and random testing, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.66 , 0.33)).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors (after compaction)	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	5000	0.66	0.33	100.000	------	------
S208	11	2	115	215	5000	0.66	0.33	34.419	0.300	34.419*
S298	3	6	136	310	5000	0.66	0.33	62.013	0.650	18.710*
S344	9	11	184	348	5000	0.66	0.33	90.351	0.467	85.632*
S349	9	11	185	356	5000	0.66	0.33	90.000	0.483	77.528*
S382	3	6	182	401	5000	0.66	0.33	12.281	0.867	10.474*
S386	7	7	172	386	5000	0.66	0.33	51.562	0.333	46.373*
S400	3	6	186	426	5000	0.66	0.33	12.028	0.950	9.859*
S420	19	2	231	430	5000	0.66	0.33	26.512	0.817	26.512*
S444	3	6	205	476	5000	0.66	0.33	11.181	1.267	0.840*
S510	19	7	236	564	5000	0.66	0.33	0.000	1.533	0.000*
S526	3	6	217	555	5000	0.66	0.33	8.649	1.400	3.423*
S641	35	24	433	467	5000	0.66	0.33	74.304	1.317	10.921*
S713	35	23	447	593	5000	0.66	0.33	72.117	0.700	56.998*
S820	18	19	312	852	5000	0.66	0.33	33.294	1.033	21.009*
S832	18	19	310	872	5000	0.66	0.33	32.299	1.050	20.528*
S838	25	2	457	857	5000	0.66	0.33	21.587	2.417	21.587*
S953	16	23	440	1079	5000	0.66	0.33	8.341	5.817	0.185*
S1196	14	14	561	1250	5000	0.66	0.33	76.651	0.933	68.240*
S1238	14	14	540	1355	5000	0.66	0.33	71.661	1.600	45.830*
S1423	17	5	748	1517	5000	0.66	0.33	35.380	3.800	25.313*
S1488	8	19	667	1490	5000	0.66	0.33	58.210	2.217	23.221*
S1494	8	19	661	1510	5000	0.66	0.33	57.171	2.150	25.166*
S5378	35	49	3050	4613	5000	0.66	0.33	51.119	16.467	30.111*
S9234	36	39	5844	3974	5000	0.66	0.33	10.386	53.000	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.14 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Deterministic Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.66$, $S_2 = 0.33$).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	2000000	0.66	0.33	100.000	------	------
S208	11	2	115	215	2000000	0.66	0.33	45.581	92.417	45.581*
S298	3	6	136	310	2000000	0.66	0.33	85.390	128.850	57.097*
S344	9	11	184	348	2000000	0.66	0.33	96.199	134.600	93.103*
S349	9	11	185	356	2000000	0.66	0.33	95.714	138.567	87.640*
S382	3	6	182	401	2000000	0.66	0.33	13.283	327.617	10.474*
S386	7	7	172	386	2000000	0.66	0.33	71.094	44.700	66.580*
S400	3	6	186	426	2000000	0.66	0.33	13.443	347.900	9.859*
S420	19	2	231	430	2000000	0.66	0.33	31.628	267.917	31.628*
S444	3	6	205	476	2000000	0.66	0.33	16.245	441.483	4.412*
S510	19	7	236	564	2000000	0.66	0.33	0.000	581.900	0.000*
S526	3	6	217	555	2000000	0.66	0.33	11.892	521.283	3.423*
S641	35	24	433	467	2000000	0.66	0.33	86.081	170.417	71.520*
S713	35	23	447	593	2000000	0.66	0.33	81.583	204.467	64.587*
S820	18	19	312	852	2000000	0.66	0.33	50.118	322.733	39.906*
S832	18	19	310	872	2000000	0.66	0.33	49.080	334.400	39.220*
S838	25	2	457	857	2000000	0.66	0.33	24.387	969.800	24.387*
S953	16	23	440	1079	2000000	0.66	0.33	8.341	2412.117	0.185*
S1196	14	14	561	1250	2000000	0.66	0.33	98.953	90.983	97.040*
S1238	14	14	540	1355	2000000	0.66	0.33	93.948	189.867	83.026*
S1423	17	5	748	1517	2000000	0.66	0.33	66.535	801.933	64.074*
S1488	8	19	667	1490	2000000	0.66	0.33	76.918	646.667	32.282*
S1494	8	19	661	1510	2000000	0.66	0.33	76.096	633.150	38.477*
S5378	35	49	3050	4613	2000000	0.66	0.33	63.611	3876.500	53.761*
S9234	36	39	5844	3974	2000000	0.66	0.33	11.199	21425.61	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.15 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Random Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.66$, $S_2 = 0.33$).

Table 7.16, and Table 7.17 show the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using HOPE with deterministic and random testing, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.33 , 0.66)).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	5000	0.33	0.66	100.000	------	------
S208	11	2	115	215	5000	0.33	0.66	34.419	0.317	34.419*
S298	3	6	136	310	5000	0.33	0.66	62.013	0.667	18.710*
S344	9	11	184	342	5000	0.33	0.66	90.351	0.483	75.146*
S349	9	11	185	350	5000	0.33	0.66	90.000	1.300	1.714*
S382	3	6	182	401	5000	0.33	0.66	12.281	0.867	10.474*
S386	7	7	172	384	5000	0.33	0.66	51.562	0.333	45.833*
S400	3	6	186	426	5000	0.33	0.66	12.028	0.950	9.859*
S420	19	2	231	430	5000	0.33	0.66	26.512	0.817	26.512*
S444	3	6	205	476	5000	0.33	0.66	11.181	1.267	0.840*
S510	19	7	236	564	5000	0.33	0.66	0.000	1.733	0.000*
S526	3	6	217	555	5000	0.33	0.66	8.649	1.317	3.423*
S641	35	24	433	467	5000	0.33	0.66	74.304	1.350	10.921*
S713	35	23	447	581	5000	0.33	0.66	72.117	1.300	25.129*
S820	18	19	312	850	5000	0.33	0.66	33.294	1.033	20.824*
S832	18	19	310	870	5000	0.33	0.66	32.299	1.050	20.345*
S838	25	2	457	857	5000	0.33	0.66	21.587	2.400	21.587*
S953	16	23	440	1079	5000	0.33	0.66	8.341	6.117	0.185*
S1196	14	14	561	1244	5000	0.33	0.66	76.651	1.217	56.109*
S1238	14	14	540	1355	5000	0.33	0.66	71.661	1.450	45.830*
S1423	17	5	748	1515	5000	0.33	0.66	35.380	3.867	24.752*
S1488	8	19	667	1486	5000	0.33	0.66	58.210	2.100	20.727*
S1494	8	19	661	1506	5000	0.33	0.66	57.171	2.717	27.158*
S5378	35	49	3050	4603	5000	0.33	0.66	51.119	15.383	34.086*
S9234	36	39	5844	3957	5000	0.33	0.66	10.386	52.230	0.000*
S13207	62	152	7951	Memory, CPU time and Disk usage limits are exceeded						
S15850	77	150	9772	Memory, CPU time and Disk usage limits are exceeded						
S35932	35	320	16065	Memory, CPU time and Disk usage limits are exceeded						
S38417	28	106	19253	Memory, CPU time and Disk usage limits are exceeded						
S38584	38	304	22179	Memory, CPU time and Disk usage limits are exceeded						

Table 7.16 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Deterministic Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.33$, $S_2 = 0.66$).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	2000000	0.33	0.66	100.000	------	------
S208	11	2	115	215	2000000	0.33	0.66	45.581	92.917	45.581*
S298	3	6	136	310	2000000	0.33	0.66	85.390	128.983	57.097*
S344	9	11	184	342	2000000	0.33	0.66	96.199	121.117	80.409*
S349	9	11	185	350	2000000	0.33	0.66	95.714	508.800	1.714*
S382	3	6	182	401	2000000	0.33	0.66	13.283	327.550	10.474*
S386	7	7	172	384	2000000	0.33	0.66	71.094	47.017	64.323*
S400	3	6	186	426	2000000	0.33	0.66	13.443	347.850	9.859*
S420	19	2	231	430	2000000	0.33	0.66	31.628	267.233	31.628*
S444	3	6	205	476	2000000	0.33	0.66	16.245	435.783	4.412*
S510	19	7	236	564	2000000	0.33	0.66	0.000	581.783	0.000*
S526	3	6	217	555	2000000	0.33	0.66	11.892	500.517	3.423*
S641	35	24	433	467	2000000	0.33	0.66	86.081	169.550	71.520*
S713	35	23	447	581	2000000	0.33	0.66	81.583	245.067	54.905*
S820	18	19	312	850	2000000	0.33	0.66	50.118	320.550	39.765*
S832	18	19	310	870	2000000	0.33	0.66	49.080	335.467	38.966*
S838	25	2	457	857	2000000	0.33	0.66	24.387	906.083	24.387*
S953	16	23	440	1079	2000000	0.33	0.66	8.341	2559.800	0.185*
S1196	14	14	561	1244	2000000	0.33	0.66	98.953	110.750	91.801*
S1238	14	14	540	1355	2000000	0.33	0.66	93.948	166.583	83.026*
S1423	17	5	748	1515	2000000	0.33	0.66	66.535	761.033	62.376*
S1488	8	19	667	1486	2000000	0.33	0.66	76.918	677.400	34.657*
S1494	8	19	661	1506	2000000	0.33	0.66	76.096	703.000	39.575*
S5378	35	49	3050	4603	2000000	0.33	0.66	63.611	3262.717	50.858*
S9234	36	39	5844	3957	2000000	0.33	0.66	11.199	20831.154	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.17 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Random Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.33$, $S_2 = 0.66$).

Table 7.18, and Table 7.19 show the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using HOPE with deterministic and random testing, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.90 , 0.10)).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	5000	0.90	0.10	100.000	------	------
S208	11	2	115	217	5000	0.90	0.10	34.419	0.300	34.419*
S298	3	6	136	310	5000	0.90	0.10	62.013	0.700	18.710*
S344	9	11	184	352	5000	0.90	0.10	90.351	0.450	86.932*
S349	9	11	185	360	5000	0.90	0.10	90.000	0.450	80.556*
S382	3	6	182	401	5000	0.90	0.10	12.281	0.883	10.474*
S386	7	7	172	390	5000	0.90	0.10	51.562	0.333	48.205*
S400	3	6	186	426	5000	0.90	0.10	12.028	0.933	9.859*
S420	19	2	231	432	5000	0.90	0.10	26.512	0.817	26.512*
S444	3	6	205	476	5000	0.90	0.10	11.181	1.267	0.840*
S510	19	7	236	564	5000	0.90	0.10	0.000	1.517	0.000*
S526	3	6	217	555	5000	0.90	0.10	8.649	1.300	3.423*
S641	35	24	433	493	5000	0.90	0.10	74.304	0.467	73.428*
S713	35	23	447	605	5000	0.90	0.10	72.117	0.633	61.322*
S820	18	19	312	856	5000	0.90	0.10	33.294	1.033	22.897*
S832	18	19	310	876	5000	0.90	0.10	32.299	1.033	25.457*
S838	25	2	457	859	5000	0.90	0.10	21.587	2.433	21.587*
S953	16	23	440	1085	5000	0.90	0.10	8.341	6.350	0.000*
S1196	14	14	561	1250	5000	0.90	0.10	76.651	0.983	68.240*
S1238	14	14	540	1363	5000	0.90	0.10	71.661	1.033	65.371*
S1423	17	5	748	1521	5000	0.90	0.10	35.380	3.133	33.202*
S1488	8	19	667	1506	5000	0.90	0.10	58.210	1.667	40.571*
S1494	8	19	661	1526	5000	0.90	0.10	57.171	1.617	41.088*
S5378	35	49	3050	4625	5000	0.90	0.10	51.119	20.283	31.110*
S9234	36	39	5844	3978	5000	0.90	0.10	10.386	55.717	0.185*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.18 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Deterministic Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.90$, $S_2 = 0.10$).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	2000000	0.90	0.10	100.000	------	------
S208	11	2	115	217	2000000	0.90	0.10	45.581	92.217	45.581*
S298	3	6	136	310	2000000	0.90	0.10	85.390	129.817	57.097*
S344	9	11	184	352	2000000	0.90	0.10	96.199	136.750	93.750*
S349	9	11	185	360	2000000	0.90	0.10	95.714	141.117	88.333*
S382	3	6	182	401	2000000	0.90	0.10	13.283	327.467	10.474*
S386	7	7	172	390	2000000	0.90	0.10	71.094	41.917	69.744*
S400	3	6	186	426	2000000	0.90	0.10	13.443	347.217	9.859*
S420	19	2	231	432	2000000	0.90	0.10	31.628	266.450	31.628*
S444	3	6	205	476	2000000	0.90	0.10	16.245	441.367	4.412*
S510	19	7	236	564	2000000	0.90	0.10	0.000	582.500	0.000*
S526	3	6	217	555	2000000	0.90	0.10	11.892	500.417	3.423*
S641	35	24	433	493	2000000	0.90	0.10	86.081	130.667	86.081*
S713	35	23	447	605	2000000	0.90	0.10	81.583	201.467	66.942*
S820	18	19	312	856	2000000	0.90	0.10	50.118	318.500	41.005*
S832	18	19	310	876	2000000	0.90	0.10	49.080	318.317	43.721*
S838	25	2	457	859	2000000	0.90	0.10	24.387	923.650	24.387*
S953	16	23	440	1085	2000000	0.90	0.10	8.341	2372.983	0.000*
S1196	14	14	561	1250	2000000	0.90	0.10	98.953	91.033	97.040*
S1238	14	14	540	1363	2000000	0.90	0.10	93.948	120.267	91.123*
S1423	17	5	748	1521	2000000	0.90	0.10	66.535	811.883	66.272*
S1488	8	19	667	1506	2000000	0.90	0.10	76.918	558.567	57.769*
S1494	8	19	661	1526	2000000	0.90	0.10	76.096	518.983	57.667*
S5378	35	49	3050	4625	2000000	0.90	0.10	63.611	3984.267	53.989*
S9234	36	39	5844	3978	2000000	0.90	0.10	11.199	23457.19	0.185*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.19 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Random Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.90$, $S_2 = 0.10$).

Table 7.20, and Table 7.21 show the fault coverage for all ISCAS 89 benchmark circuits without compactors, and with pairwise line merger compactors, when stochastic dependence of single and double line errors is assumed, and using HOPE with deterministic and random testing, for the following probability values of single and double line errors: ((S_1 , S_2) equals (0.10 , 0.90)).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	5000	0.10	0.90	100.000	------	------
S208	11	2	115	215	5000	0.10	0.90	34.419	0.300	34.419*
S298	3	6	136	310	5000	0.10	0.90	62.013	0.650	18.710*
S344	9	11	184	342	5000	0.10	0.90	90.351	0.500	74.854*
S349	9	11	185	350	5000	0.10	0.90	90.000	1.300	2.000*
S382	3	6	182	401	5000	0.10	0.90	12.281	0.883	10.474*
S386	7	7	172	384	5000	0.10	0.90	51.562	0.333	45.833*
S400	3	6	186	426	5000	0.10	0.90	12.028	0.933	9.859*
S420	19	2	231	430	5000	0.10	0.90	26.512	0.817	26.512*
S444	3	6	205	476	5000	0.10	0.90	11.181	1.250	0.840*
S510	19	7	236	564	5000	0.10	0.90	0.000	1.550	0.000*
S526	3	6	217	555	5000	0.10	0.90	8.649	1.317	3.423*
S641	35	24	433	467	5000	0.10	0.90	74.304	1.517	0.642*
S713	35	23	447	581	5000	0.10	0.90	72.117	1.850	1.033*
S820	18	19	312	850	5000	0.10	0.90	33.294	1.033	20.824*
S832	18	19	310	870	5000	0.10	0.90	32.299	1.033	20.345*
S838	25	2	457	857	5000	0.10	0.90	21.587	2.517	21.587*
S953	16	23	440	1079	5000	0.10	0.90	8.341	7.033	0.185*
S1196	14	14	561	1242	5000	0.10	0.90	76.651	1.433	48.953*
S1238	14	14	540	1355	5000	0.10	0.90	71.661	1.417	45.830*
S1423	17	5	748	1515	5000	0.10	0.90	35.380	3.950	24.752*
S1488	8	19	667	1506	5000	0.10	0.90	58.210	1.633	40.571*
S1494	8	19	661	1506	5000	0.10	0.90	57.171	2.050	23.838*
S5378	35	49	3050	4603	5000	0.10	0.90	51.119	17.617	28.649*
S9234	36	39	5844	3954	5000	0.10	0.90	10.386	53.172	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.20 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Deterministic Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.10$, $S_2 = 0.90$).

Circuit name	No. of input lines in CUT	No. of output lines in CUT	No. of gates in CUT	No. of injected faults	No. of applied test vectors	S1	S2	Fault coverage (%) (without space compactors)	CPU simulation time (secs)	Fault coverage (%) (with space compactors)
S27	4	1	17	------	2000000	0.10	0.90	100.000	------	------
S208	11	2	115	215	2000000	0.10	0.90	45.581	92.517	45.581*
S298	3	6	136	310	2000000	0.10	0.90	85.390	141.750	57.097*
S344	9	11	184	342	2000000	0.10	0.90	96.199	122.900	80.409*
S349	9	11	185	350	2000000	0.10	0.90	95.714	507.350	2.000*
S382	3	6	182	401	2000000	0.10	0.90	13.283	327.567	10.474*
S386	7	7	172	384	2000000	0.10	0.90	71.094	47.217	64.323*
S400	3	6	186	426	2000000	0.10	0.90	13.443	347.233	9.859*
S420	19	2	231	430	2000000	0.10	0.90	31.628	267.350	31.628*
S444	3	6	205	476	2000000	0.10	0.90	16.245	435.750	4.412*
S510	19	7	236	564	2000000	0.10	0.90	0.000	582.183	0.000*
S526	3	6	217	555	2000000	0.10	0.90	11.892	500.767	3.423*
S641	35	24	433	467	2000000	0.10	0.90	86.081	591.183	4.711*
S713	35	23	447	581	2000000	0.10	0.90	81.583	719.433	1.033*
S820	18	19	312	850	2000000	0.10	0.90	50.118	339.333	39.765*
S832	18	19	310	870	2000000	0.10	0.90	49.080	330.717	38.966*
S838	25	2	457	857	2000000	0.10	0.90	24.387	924.983	24.387*
S953	16	23	440	1079	2000000	0.10	0.90	8.341	2680.050	0.185*
S1196	14	14	561	1242	2000000	0.10	0.90	98.953	135.517	89.211*
S1238	14	14	540	1355	2000000	0.10	0.90	93.948	191.717	83.026*
S1423	17	5	748	1515	2000000	0.10	0.90	66.535	850.483	62.376*
S1488	8	19	667	1506	2000000	0.10	0.90	76.918	517.767	57.769*
S1494	8	19	661	1506	2000000	0.10	0.90	76.096	702.117	37.052*
S5378	35	49	3050	4603	2000000	0.10	0.90	63.611	3348.217	51.944*
S9234	36	39	5844	3954	2000000	0.10	0.90	11.199	23257.14	0.000*
S13207	62	152	7951	**Memory, CPU time and Disk usage limits are exceeded**						
S15850	77	150	9772	**Memory, CPU time and Disk usage limits are exceeded**						
S35932	35	320	16065	**Memory, CPU time and Disk usage limits are exceeded**						
S38417	28	106	19253	**Memory, CPU time and Disk usage limits are exceeded**						
S38584	38	304	22179	**Memory, CPU time and Disk usage limits are exceeded**						

Table 7.21 Fault Coverage for ISCAS 89 Benchmark Circuits Using HOPE with Random Testing – Pairwise Line Merger and Assuming Stochastic Dependence of Single and Double Line Errors for ($S_1 = 0.10$, $S_2 = 0.90$).

In all cases, our space compactor is comparable in all respects with the parity tree space compactor. For some circuits, we obtained *better fault coverage* with a *reduction in the CPU time* using our space compactor than what we have when a parity tree compactor is used.

Simulation results are encouraging in several respects:

- design simplicity;
- low overhead for the designed compactor which might make it suitable for built-in self-testing in VLSI design even though it does not guarantee 100 % fault coverage; and,
- even though we used reduced input test sets provided by ATALANTA and COMPACTEST to simulate various ISCAS 85 and ISCAS 89 benchmark circuits, knowing that these test sets are not the minimal test sets needed to ensure a 100 % fault coverage, experimental results indicate the designed space compactors are comparable in all respects with the parity tree space compactor which propagates all errors that appear on an odd number of inputs and which is usually considered ideal for ad hoc space compaction; and,
- combined with efficient input test patterns to synthesize BIST circuits, our method provides high percentage of fault coverage with a small CPU time.

As discussed before, to demonstrate the feasibility of the proposed space compaction schemes independent simulations were conducted on various ISCAS 85 combinational benchmark circuits. We used ATALANTA [40] to generate the fault-free output sequences required to construct our space compactor circuits and to test the benchmark circuits using reduced or compact test sets accompanied with a random testing session with FSIM fault simulation program to generate pseudorandom test sets, and COMPACTEST [42] program to generate reduced test sets that detect most detectable single stuck-line faults for all of the benchmark circuits. For each circuit, we determined the number of test vectors used to construct the compaction tree, the CPU time taken to construct the compactor, the number of applied test vectors, the simulation CPU time, and the percentage fault coverage by running ATALANTA and FSIM programs on a SUN SPARC 5 workstation, and COMPACTEST on an IBM aix machine, under the condition of generalized mergeability. For comparison purposes, we used a parity tree space compactor composed of XOR gates, that propagates errors on an odd number of inputs and is usually considered ideal for space compression. In the following tables we provide some experimental results in the generalized mergeability case on ISCAS 85 benchmark circuits using FSIM, ATALANTA, and COMPACTEST. As evident, in all cases, our space compactors compare very favorably with the parity tree space compactor in terms of fault coverage and reduced CPU time. The hardware overhead of the compactors was found to be within acceptable limits (in general, within 1-4% and up to 15% in certain cases).

As a result of the conducted simulation and for some combinational circuits, we obtained several space compressors (trees). In such situation, we selected the "best tree", which is the tree that gave the highest fault coverage; when two or more trees had identical fault coverage, we selected the one that gave the smallest CPU time.

For each ISCAS 85 combinational circuit, several variables were determined and included in tabular format. These variables are: the number of test vectors used to construct the best compaction tree, the CPU time taken to construct the best compaction tree, the number of applied test vectors corresponding to the best tree obtained, the simulation CPU time and the percentage of fault coverage by running ATALANTA, FSIM and COMPACTEST programs on a SPARC 5 SUN workstation.

All the simulation results obtained from FSIM, ATALANTA and COMPACTEST are classified in charts and tables, and are presented below. We also estimated the hardware overhead for all the circuits.

Figure 7.10, Figure 7.11, and Figure 7.12 show the fault coverage for all ISCAS 85 benchmark circuits with our multiple line merger compactors, when stochastic independence of single and double line errors is assumed, and using ATALANTA, FSIM, and COMPACTEST, respectively.

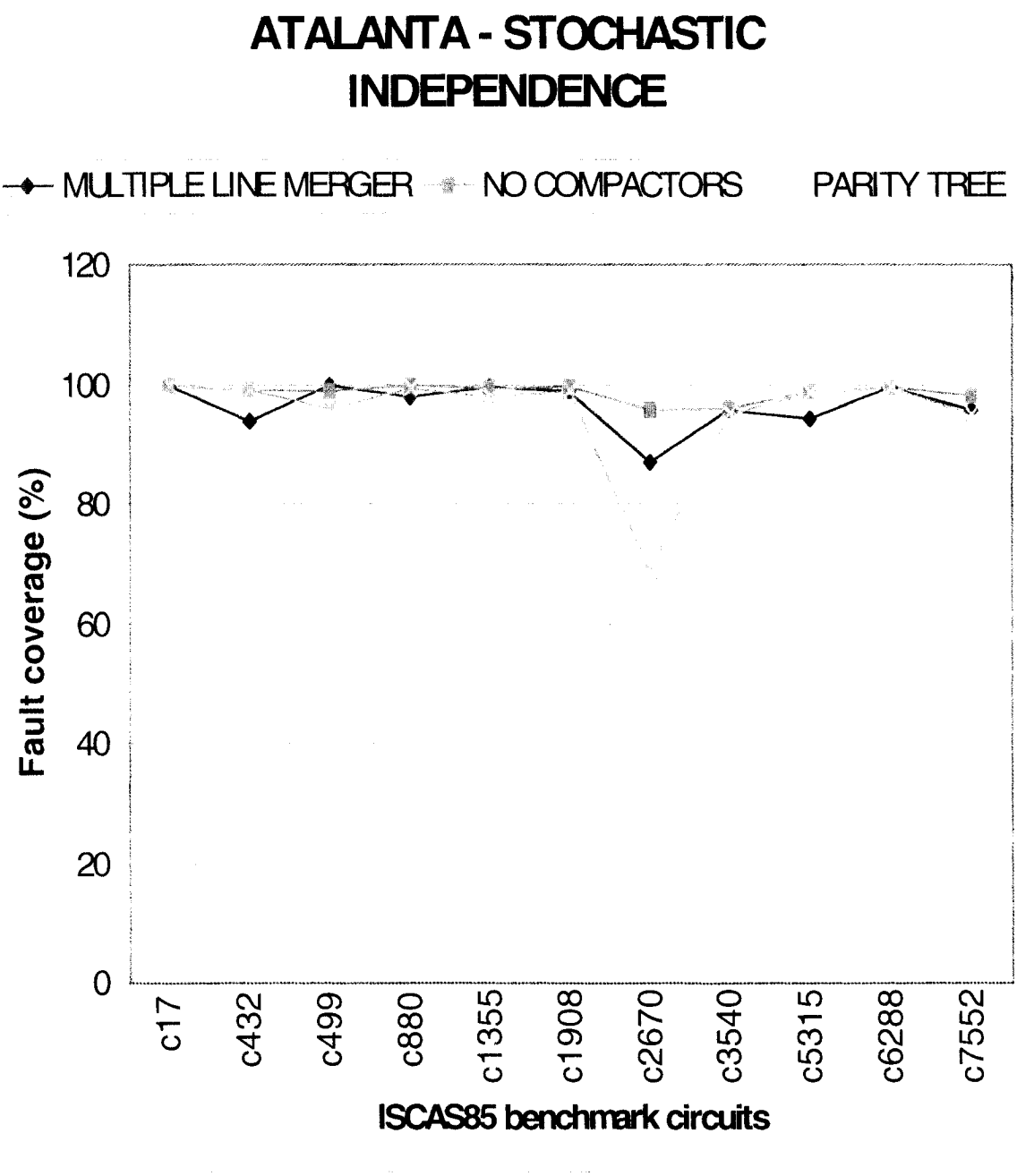


Figure 7.10 Fault Coverage Using ATALANTA – Parity Tree vs. Multiple Line Merger Under Stochastic Independence of Line Errors.

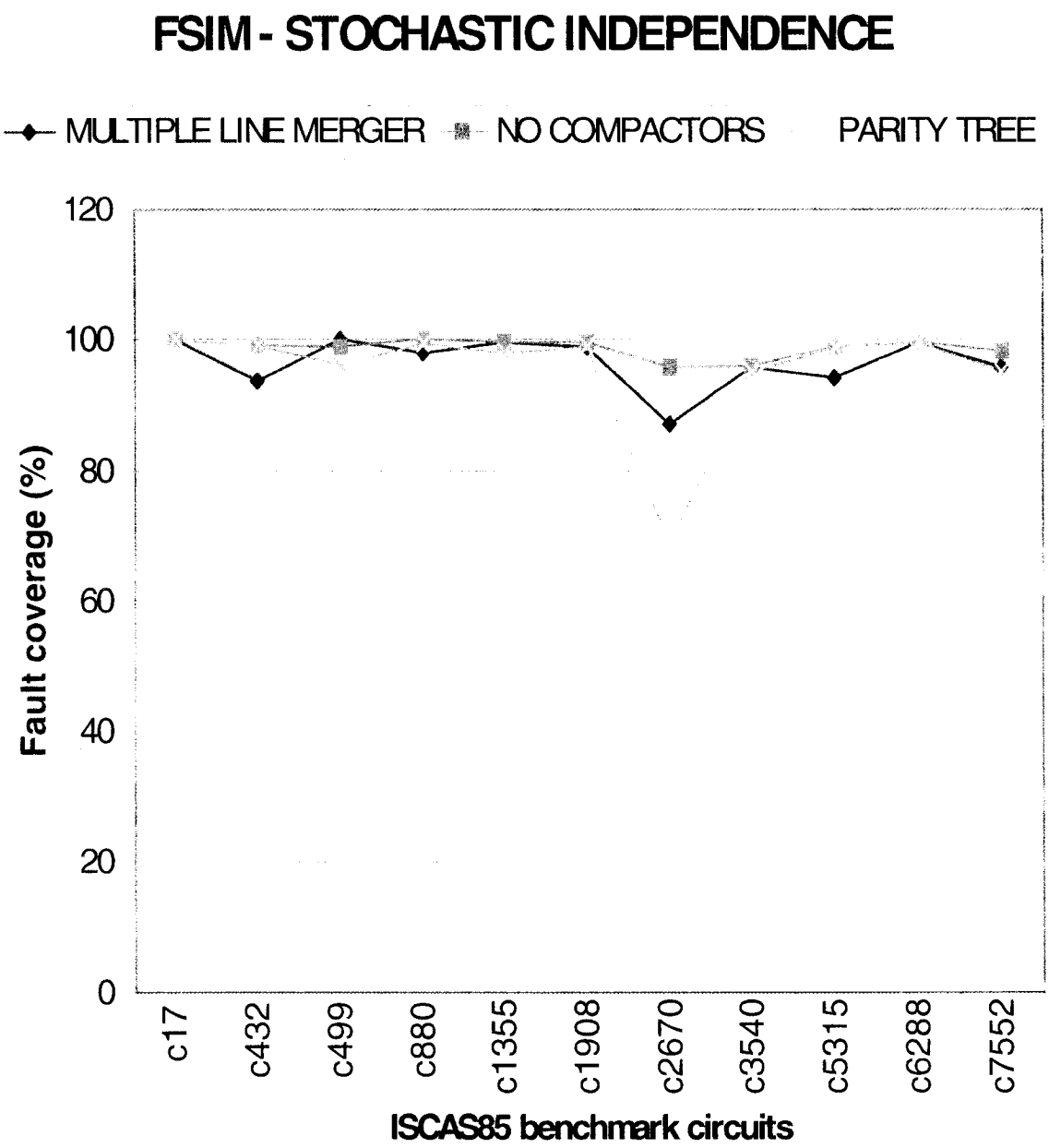


Figure 7.11 Fault Coverage Using FSIM (Input Test Vectors = 224) – Parity Tree vs. Multiple Line Merger Under Stochastic Independence of Line Errors.

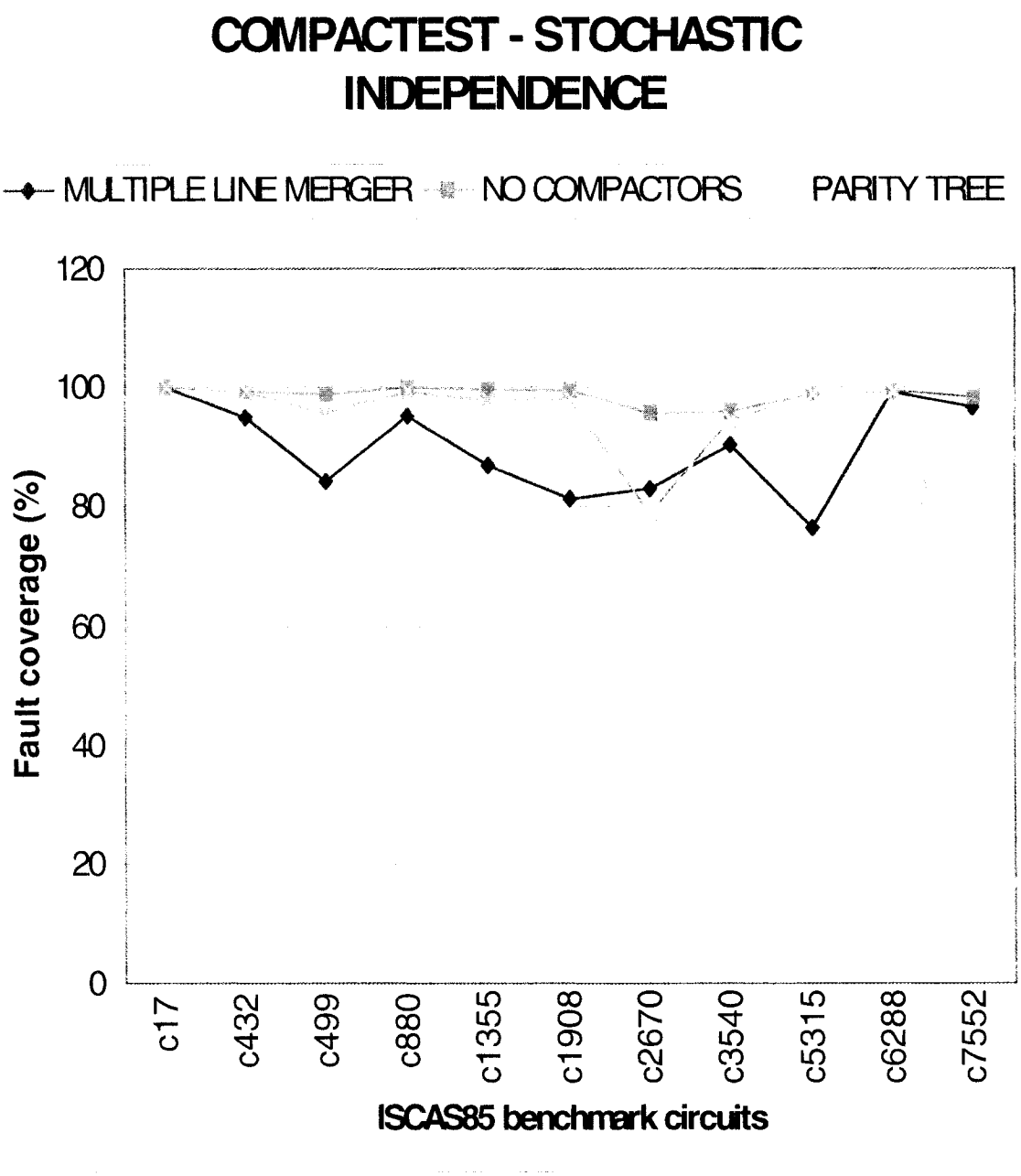


Figure 7.12 Fault Coverage Using COMPACTEST – Parity Tree vs. Multiple Line Merger Under Stochastic Independence of Line Errors.

Table 7.22, Table 7.23, and Table 7.24 show the number of applied test vectors, CPU simulation time, and the fault coverage for the ISCAS 85 benchmark circuits with our multiple line merger compactors, when stochastic dependence of single and double line errors is assumed, and using ATALANTA, FSIM, and COMPACTEST, respectively.

Circuit Name	No of Test Vectors Used to Construct The Compactor	CPU Time Taken to Construct The Compactor (secs)	No. of Applied Test Vectors	CPU Simulation Time (secs)	Fault Coverage (%)
c17	7	0.12	32	0.15	100.00
c432	46	0.25	224	0.25	94.38
c499	54	10.10	224	0.23	94.51
c880	56	7.53	224	0.30	94.62
c1355	86	25.73	224	0.45	89.98
c1908	115	21.00	224	0.70	84.02
c2670	108	2503.87	224	0.87	79.03
c3450	144	22.65	224	1.78	84.99
c5315	117	3790.17	224	1.50	84.96
c6288	31	4.03	224	5.00	99.47
c7552	217	4560.80	224	2.58	87.91

Table 7.22 Simulation Results Using FSIM and Assuming Stochastic Dependence of Multiple Line Errors.

Circuit Name	No of Test Vectors Used to Construct The Compactor	CPU Time Taken to Construct The Compactor (secs)	No of Applied Test Vectors	CPU Simulation Time (secs)	Fault Coverage (%)
c17	7	0.12	6	0.03	100.00
c432	46	0.25	67	0.78	98.50
c499	54	10.10	53	1.27	96.10
c880	56	7.53	70	1.65	99.07
c1355	86	25.73	85	2.33	98.04
c1908	115	21.00	145	4.58	98.91
c2670	108	2503.87	122	42.63	87.80
c3450	144	22.65	169	19.08	95.12
c5315	117	3790.17	148	26.43	89.91
c6288	31	4.03	67	36.35	99.56
c7552	217	4560.80	161	127.82	94.66

Table 7.23 Simulation Results Using ATALANTA and Assuming Stochastic Dependence of Multiple Line Errors.

Circuit Name	No of Test Vectors Used to Construct The Compactor	CPU Time Taken to Construct the Compactor (secs)	No of Applied Test Vectors	CPU Simulation Time (secs)	Fault Coverage (%)
c17	4	0.10	6	0.01	100.00
c432	44	0.25	65	9.86	98.28
c499	63	3108.00	87	32.45	98.81
c880	30	3.43	57	7.40	99.05
c1355	96	29.75	109	37.23	99.24
c1908	137	29.35	168	134.66	98.83
c2670	68	1016.03	117	354.129	89.36
c3450	110	12.80	162	624.97	94.49
c5315	55	9626.33	139	321.87	96.51
c6288	16	2.12	70	112.62	99.56
c7552	85	789.53	129	1313.04	93.09

Table 7.24 Simulation Results Using COMPCTEST and Assuming Stochastic Dependence of Multiple Line Errors.

The hardware area overhead estimates for all ISCAS 85 benchmark circuits are shown in Figure 7.13. The area overhead is found to be as small as 0.01 - 4 % in most cases.

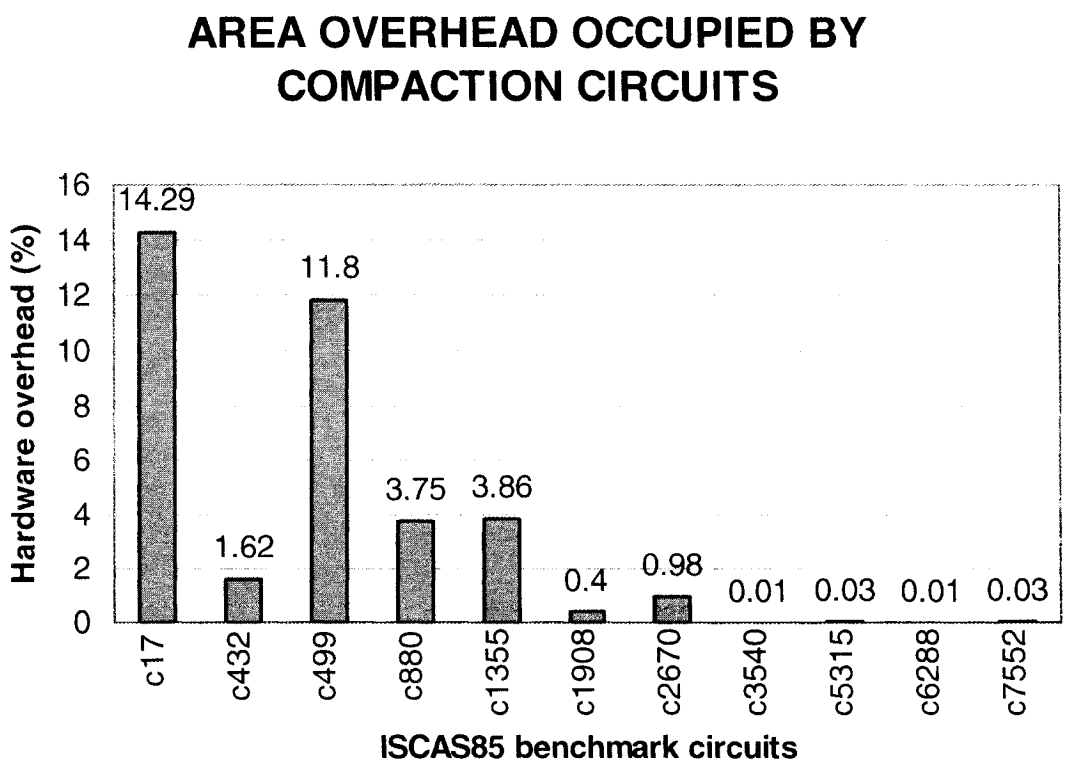


Figure 7.13 Estimates of the Hardware Overhead (Multiple Merger).

It is worth noting here that for each grouping, S_i which is the probability of the ith line error occurrence is equal to $2/3[\ 1/3^{(i-1)}]$ for $1 \leq i \leq (N-1)$, and it is equal to $1/3[\ 1/3^{(i-2)}]$ for $i = N$.
With the parity tree space compactor as reference, the novelty of the proposed schemes of constructing compaction trees based on extensive simulation runs on ISCAS 85 combinational benchmark circuits. Many of these simulation results could be found reported in [53]. The heuristic based approach to generate the compression trees also works satisfactorily, as is obvious from the results of simulation.

We have developed in chapter 5 our optimal generalized mergeability criteria, first assuming stochastic independence of multiple line errors, and then stochastic dependence of multiple line errors. Using the developed gate selection criteria, the space compressors were then designed for the ISCAS 85 combinational benchmark circuits. In order to investigate the feasibility of the proposed space compression methods, independent simulations were conducted on various ISCAS 85 combinational benchmark circuits using ATALANTA [40] to generate the fault-free output sequences needed to construct our compactor circuits and to test the benchmark circuits using reduced test sets accompanied with a random test session, FSIM fault simulation program [39] to generate pseudorandom test sets, and COMPACTEST [42] program to generate the reduced test sets that detect *most* detectable single stuck-line faults for all benchmark circuits.

For comparison purposes, we used parity tree space compactors as our *benchmark*, composed of XOR gates only, that propagates all errors appearing on an odd number of inputs, and is hence considered ideal for ad hoc space compaction.

For each benchmark circuit, several variables were determined and presented in tabular format. These include the number of test vectors used to construct the compaction tree, CPU time required to construct the compaction tree, number of applied test vectors corresponding to the obtained trees, simulation CPU time, and percentage fault coverage, by running ATALANTA and FSIM programs on a SUN SPARC ULTRA1 workstation, and COMPACTEST on an IBM AIX machine. In addition, we also estimated the hardware overhead for the space compactors.

The simulation results assuming stochastic independence of multiple line errors are shown in Figure 7.14, Figure 7.15, and Figure 7.16. In comparison with the results without space compactors, the simulation results with space compactors show that all the values, viz. CPU simulation time, fault coverage, are changed to varying degrees due to inclusion of the compressors. Figure 7.14, Figure 7.15, and Figure 7.16 also show that under stochastic independence of line errors, ATALANTA provides the highest fault coverage, ranging from 87.368% to 100.00%. The CPU simulation time provided by FSIM is the best, as shown in Figure 7.17.

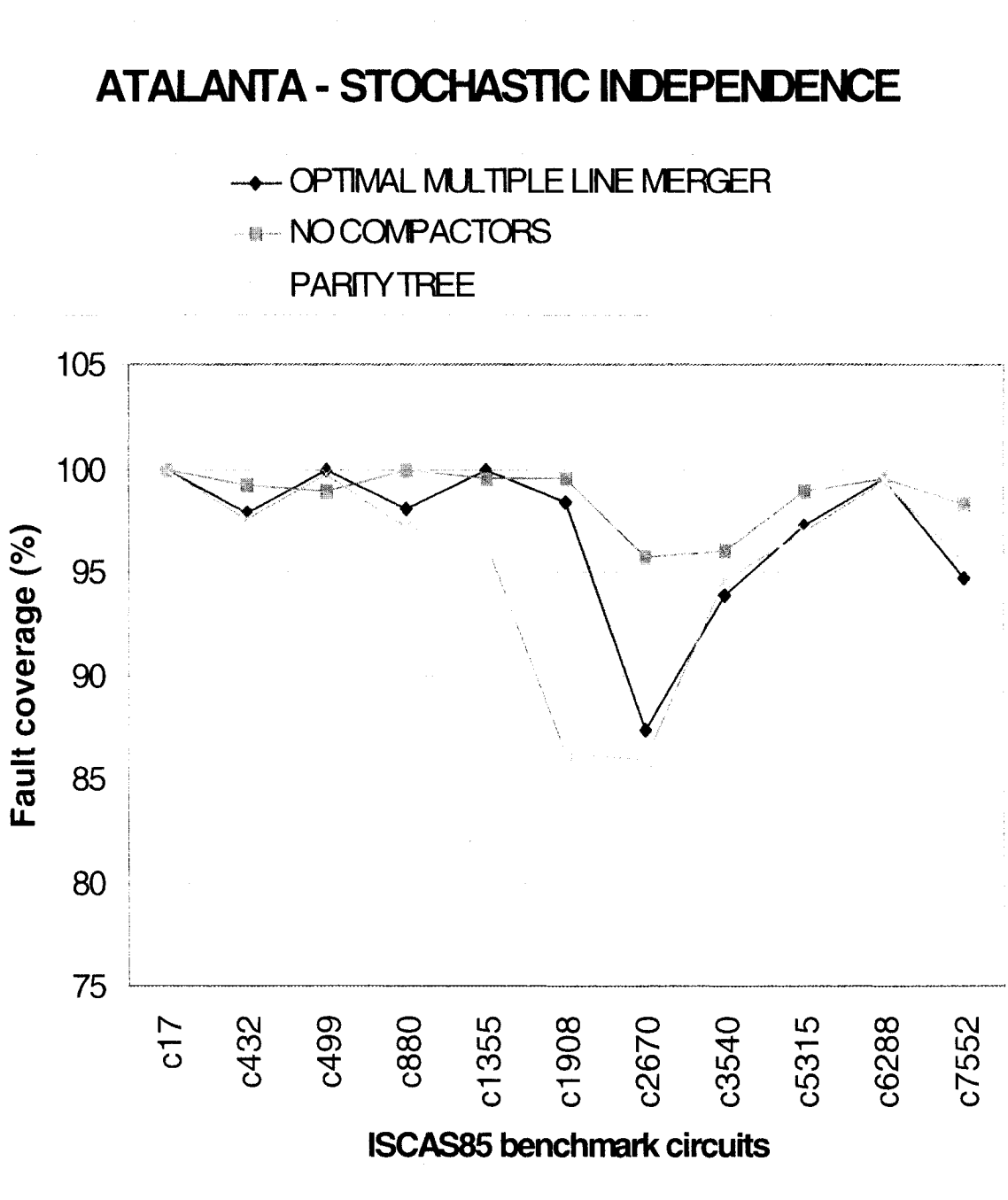


Figure 7.14 Fault Coverage Using ATALANTA – Parity Tree vs. Optimal Multiple Line Merger Under Stochastic Independence of Line Errors.

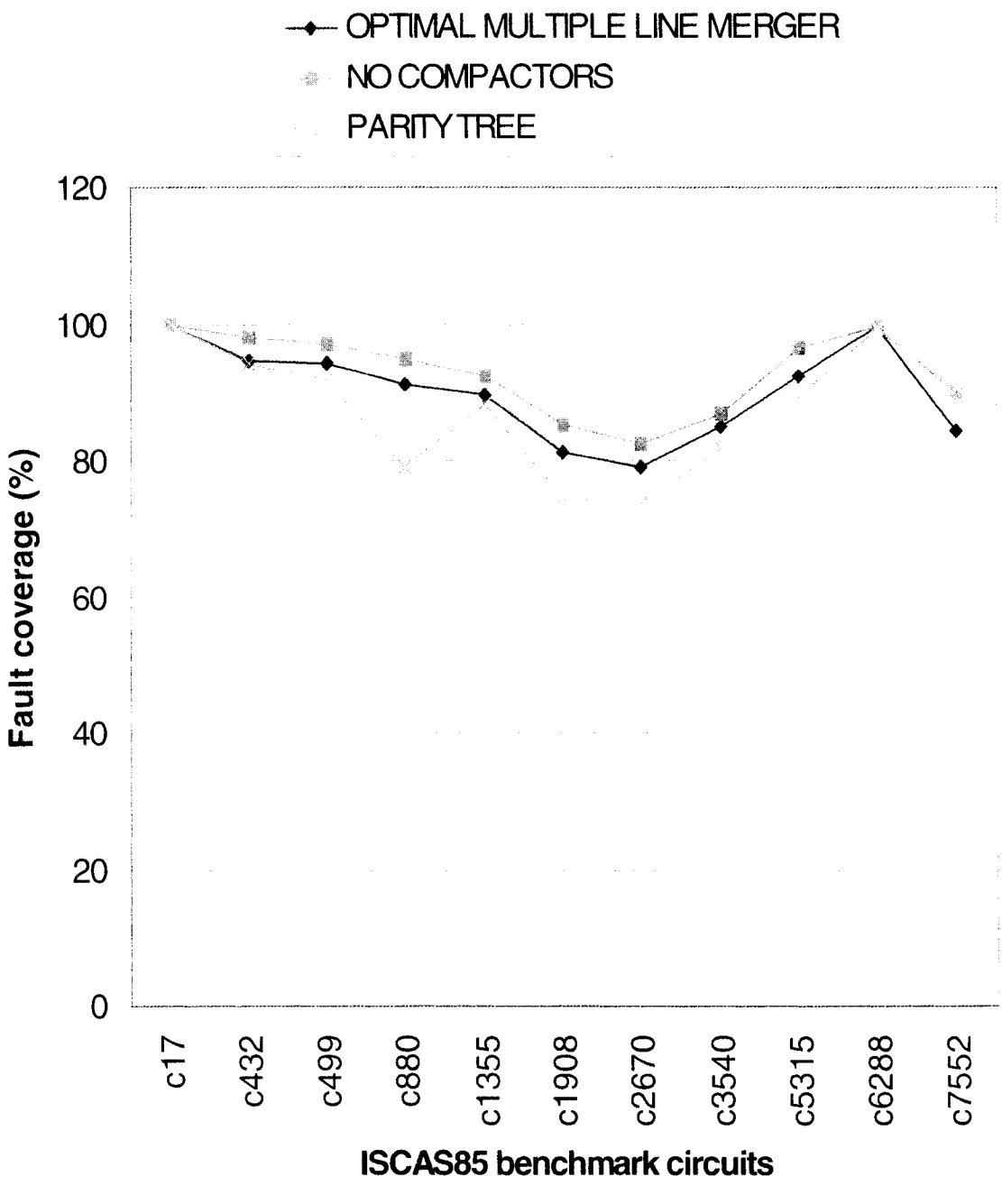


Figure 7.15 Fault Coverage Using FSIM – Parity Tree vs. Optimal Multiple Line Merger Under Stochastic Independence of Line Errors.

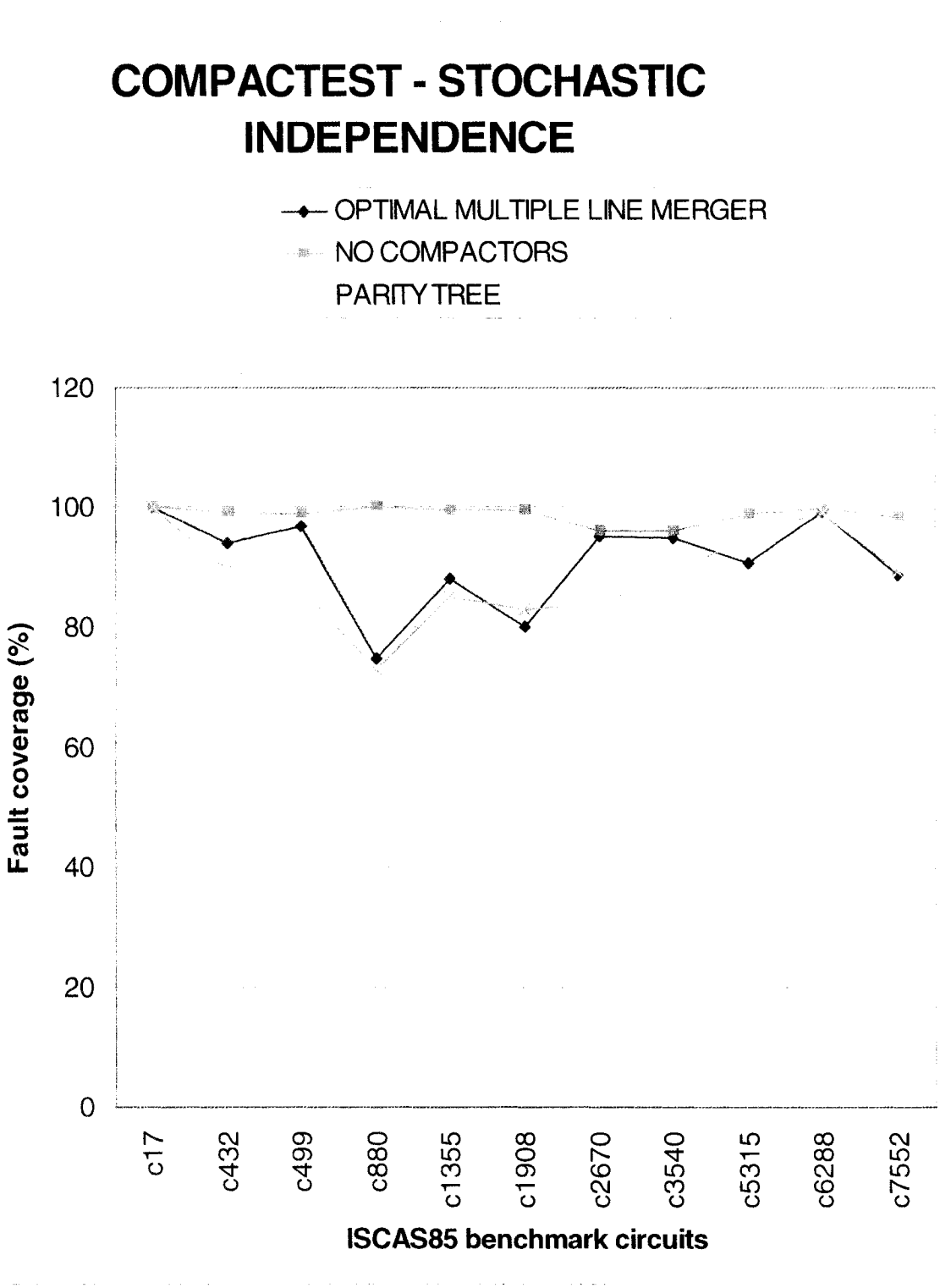


Figure 7.16 Fault Coverage Using COMPACTEST – Parity Tree vs. Optimal Multiple Line Merger Under Stochastic Independence of Line Errors.

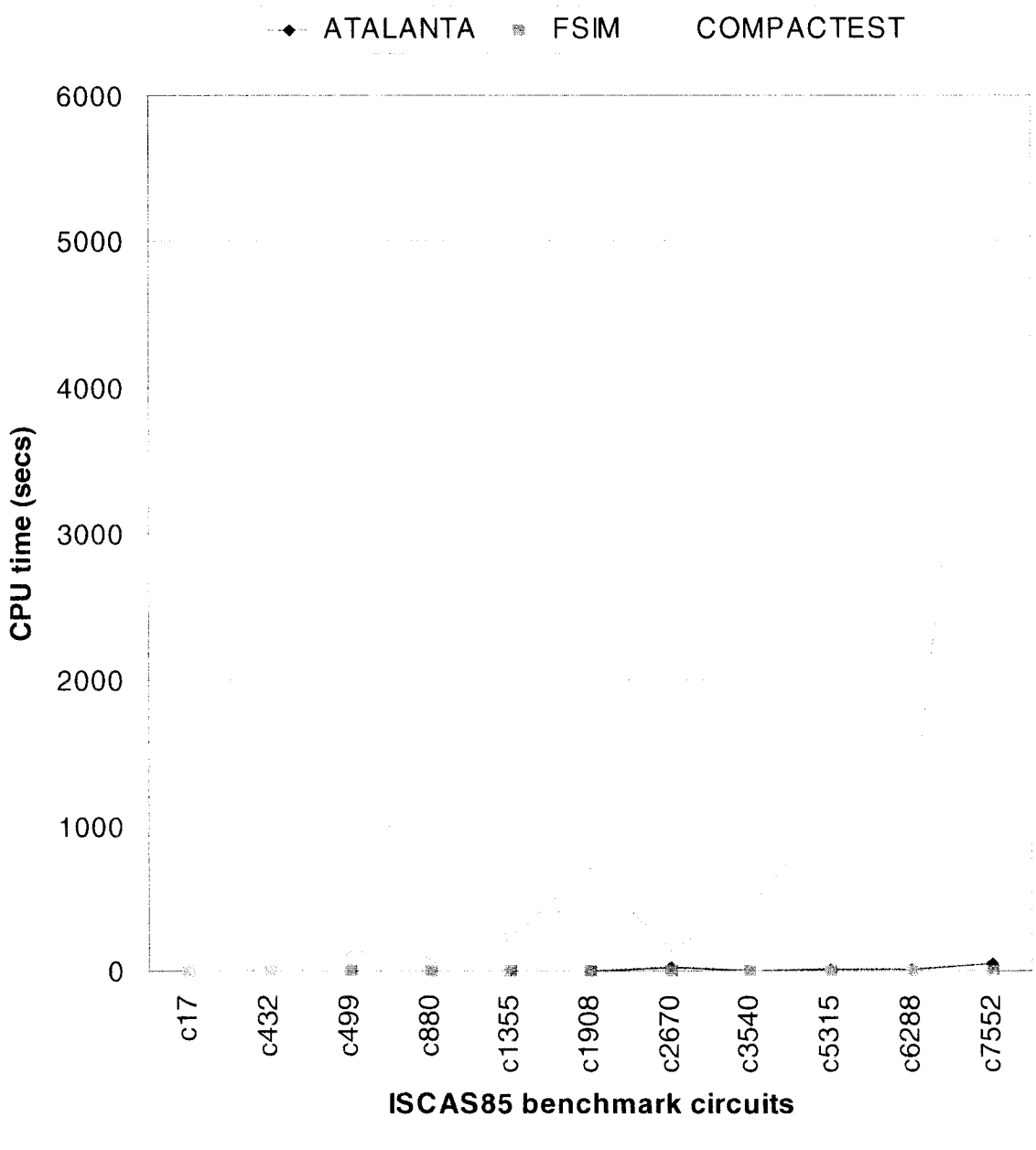


Figure 7.17 CPU Simulation Time (Optimal Multiple Merger).

For comparison purpose, we simulated the benchmark circuits using the *exact approach* as mentioned earlier, in chapter 5, by ATALANTA, FSIM, and COMPACTEST. The results are given in Table 7.25, Table 7.26, and Table 7.27, respectively.

It is worth noting that for each max_group, t_i is the probability of missing i line errors, which equals 2/3 [(1/3)$^{N-i}$], for $2 \leq i \leq N$, and 1/3 [(1/3)$^{N-2}$], for $i = 1$. In the case of stochastic dependence of multiple line errors, we see that ATALANTA provides the best fault coverage results.

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree (secs)	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	5	0.267	10	0.000	100.000
c432	49	2.483	109	0.367	92.748
c499	54	32.667	171	0.350	98.681
c880	50	22.483	229	0.850	99.257
c1355	85	34.533	191	0.967	98.602
c1908	118	22.533	346	2.850	99.308
c2670	108	779.717	297	8.050	90.532
c3540	149	19.017	476	8.433	95.858
c5315	122	591.333	522	10.700	95.475
c6288	28	32.017	221	35.483	99.561
c7552	208	415.250	735	43.600	95.865

Table 7.25 Simulation Results of the ISCAS 85 Benchmark Circuits Using ATALANTA and Assuming Stochastic Dependence of Multiple Line Errors (Exact Approach).

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree (secs)	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	5	0.267	160	0.017	100.000
c432	49	2.483	224	0.083	78.817
c499	54	32.667	224	0.083	84.828
c880	50	22.483	224	0.117	67.622
c1355	85	34.533	224	0.167	67.789
c1908	118	22.533	224	0.267	65.194
c2670	108	779.717	224	0.333	62.819
c3540	149	19.017	224	0.733	69.166
c5315	122	591.333	224	0.633	72.569
c6288	28	32.017	224	2.800	96.320
c7552	208	415.250	224	1.167	66.711

Table 7.26 Simulation Results of the ISCAS 85 Benchmark Circuits Using FSIM and Assuming Stochastic Dependence of Multiple Line Errors (Exact Approach).

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree (secs)	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	4	0.200	6	0.010	100.000
c432	44	2.650	47	80.420	76.679
c499	63	34.950	12	305.250	71.098
c880	30	20.983	106	3.610	100.000
c1355	96	32.633	126	75.880	97.738
c1908	137	22.733	208	102.440	97.714
c2670	68	807.083	204	128.650	94.941
c3540	110	18.300	233	476.740	94.697
c5315	55	547.133	300	259.800	96.038
c6288	16	32.150	72	92.570	99.564
c7552	85	333.250	501	1166.040	94.812

Table 7.27 Simulation Results of the ISCAS 85 Benchmark Circuits Using COMPACTEST and Assuming Stochastic Dependence of Multiple Line Errors (Exact Approach).

We developed the *heuristic approach* to reduce the computational time and storage for constructing the space compactors in the case of stochastic dependence of multiple line errors. The simulation results of ISCAS 85 benchmark circuits using ATALANTA, FSIM, and COMPACTEST are shown in Table 7.28, Table 7.29, and Table 7.30, respectively.

Again, it is worth noting that for each max_group, t_i is the probability of missing i line errors, which equals $2/3\ [\ (1/3)^{N-i}\ ]$, for $2 \leq i \leq N$, and $1/3\ [\ (1/3)^{N-2}\ ]$, for $i = 1$.

As expected, the CPU time for constructing the space compactors is reduced while using the *heuristic approach* compared to the exact approach, particularly for circuits with a large number of primary outputs. For example, to construct the space compactor trees for c2670 benchmark circuit, the CPU time consumed in using the heuristic approach is about 37% less for ATALANTA and FSIM, and about 33% less for COMPACTEST than while using the exact approach. The results on fault coverage using ATALATNA and COMPACTEST by the heuristic approach show some reduction, though most are still above 80%. We also found that the fault coverage while using FSIM by the heuristic approach is higher than the coverage by the exact approach. The results demonstrate that the proposed heuristic approach not only reduces the time to construct the space compactors, but also ensures relatively high fault coverage. Moreover, ATALANTA provides the best fault coverage results, while FSIM provides the best results in terms of CPU simulation time.

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	5	0.450	10	0.000	100.000
c432	49	1.867	120	0.583	77.376
c499	54	10.417	75	0.200	100.00
c880	50	18.417	183	0.667	96.308
c1355	85	11.050	128	0.567	98.160
c1908	118	8.233	248	1.917	95.587
c2670	108	493.517	308	9.117	90.000
c3450	149	10.300	328	7.550	94.315
c5315	122	524.017	478	14.567	94.002
c6288	28	21.517	133	28.867	97.999
c7552	208	294.083	329	59.317	92.846

Table 7.28 Simulation Results of the ISCAS 85 Benchmark Circuits Using ATALANTA and Assuming Stochastic Dependence of Multiple Line Errors (Heuristic Approach).

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	5	0.450	64	0.033	100.000
c432	49	1.867	224	0.067	73.004
c499	54	10.417	224	0.050	92.763
c880	50	18.417	224	0.083	87.447
c1355	85	11.050	224	0.117	86.675
c1908	118	8.233	224	0.183	77.193
c2670	108	493.517	224	0.250	79.636
c3450	149	10.300	224	0.650	73.703
c5315	122	524.017	224	0.517	86.622
c6288	28	21.517	224	2.283	97.586
c7552	208	294.083	224	0.967	80.578

Table 7.29 Simulation Results of the ISCAS 85 Benchmark Circuits Using FSIM and Assuming Stochastic Dependence of Multiple Line Errors (Heuristic Approach).

Circuit name	No of test vectors used to construct the compaction tree	CPU time taken to construct the compaction tree	No of applied test vectors	CPU simulation time (secs)	Fault coverage (%)
c17	4	0.250	6	0.000	100.000
c432	44	2.133	53	17.170	96.030
c499	63	34.717	12	306.100	71.098
c880	30	15.683	17	118.810	76.294
c1355	96	29.950	56	282.870	89.370
c1908	137	12.417	76	473.790	89.677
c2670	68	544.100	236	134.910	93.686
c3540	110	12.033	157	351.700	95.531
c5315	55	486.800	159	230.310	96.480
c6288	16	23.333	41	129.540	99.563
c7552	85	307.250	351	1655.800	94.767

Table 7.30 Simulation Results of the ISCAS 85 Benchmark Circuits Using COMPACTEST and Assuming Stochastic Dependence of Multiple Line Errors (Heuristic Approach).

We have used the probability of missing N line errors, t_N, to be 2/3, the value being decreased as the number of lines is reduced. That means, $t_N > t_{N-1} > \ldots > t_2 > t_1$, with $t_1 + t_2 + \ldots + t_N = 1$.

Here we provide our simulation results with certain distinct values of t_N, like t_N = 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0, using ATALANTA, as shown in Table 7.31.

We still assume $t_N > t_{N-1} > \ldots > t_2 > t_1$ and $t_1 + t_2 + \ldots + t_N = 1$.

From the simulation results, we see that change in fault coverage results take place while t_N changes from 0.5 to 0.9, which shows that our assumption is reasonable and in accordance with real world situations.

Circuit name	No of test vectors used to construct the compaction tree	Fault coverage (%) (t_n = 0.5)	Fault coverage (%) (t_n = 0.6 or 0.7 or 0.8 or 0.9)	Fault coverage (%) (t_n = 2/3)	Fault coverage (%) (t_n = 1.0)
c17	5	100.000	100.000	100.000	100.00
c432	49	78.137	77.567	77.376	78.327
c499	54	100.000	100.000	100.000	98.082
c880	50	98.404	96.730	96.308	97.766
c1355	85	97.208	96.447	98.160	97.736
c1908	118	96.917	96.438	95.587	96.863
c2670	108	90.000	90.000	90.000	89.927
c3540	149	94.606	94.140	94.315	82.484
c5315	122	94.413	94.357	94.002	67.984
c6288	28	98.012	98.567	97.999	96.841
c7552	208	93.311	92.965	92.846	50.027

Table 7.31 Simulation Results of the ISCAS 85 Benchmark Circuits Using ATALANTA and Assuming Different Missed Error Probability Values, viz. t_n = 0.5, 0.6, 0.7, 0.8, 0.9, 2/3, and 1.0, Respectively, for n Line Errors.

Table 7.32 shows the hardware overhead estimates of the ISCAS 85 benchmark circuits for ATALANTA/FSIM, assuming stochastic independence of multiple line errors. It can be seen that the hardware overhead range of the compressors for all the benchmark circuits (besides circuit c17) is from 0.70% to 7.27% for ATALANTA/FSIM, and is 14.29% for the circuit c17.

Compact or circuit for	Total no of fanin in the compactor	Total no of gates in the compactor	Average fanin in the compactor	Total no of gates in the CUT	Average fainin in the CUT	Hardware overhead (%)
c17	2	1	2.00	6	2.00	14.29
c432	9	3	3.00	160	2.10	2.61
c499	32	1	32.00	202	2.02	7.27
c880	30	5	6.00	383	1.90	3.96
c1355	32	1	32.00	546	1.95	2.92
c1908	25	1	25.00	880	1.70	1.64
c2670	148	9	16.44	1269	1.64	6.64
c3540	25	4	6.25	1669	1.76	0.84
c5315	131	9	14.56	2307	1.90	2.90
c6288	34	3	11.33	2416	1.99	0.70
c7552	112	5	22.40	3515	1.75	1.79

Table 7.32 Estimates of the Hardware Overhead for ATALANTA/FSIM Assuming Stochastic Independence of Multiple Line Errors.

For comparison purposes, we used parity tree space compactors composed of XOR gates as our benchmark. The simulation experience with parity tree as space compactors using ATALANTA, FSIM, and COMPACTEST are next discussed.

We first assume that the space compactors are designed with 2-input XOR gates, and then presume them to be comprised of 10-input XOR gates. To have an overall view, we compared the simulation results by ATALANTA for the ISCAS 85 benchmark circuits in different cases, which include parity trees comprised of 2-input and 10-input XOR gates, stochastic independence and dependence of multiple line errors, and using the exact as well as heuristic approaches, as shown in Table 7.33 and Table 7.34. From a comparison of the results, we found that the fault coverage results as obtained assuming stochastic independence and stochastic dependence of line errors are almost the same in the parity tree case. The difference is rather small and acceptable. Though the XOR gate is conceptually neat and simple, but practically it has to be implemented by using other gates, and so the hardware overhead of parity tree compactors (using both 2-input and 10-input XOR gates) is expected to be higher as shown in Table 7.35.

Circuit name	CPU simulation times (secs) *A**	Fault coverage (%) *A**	CPU simulation times (secs) *F**	Fault coverage (%) *F**	CPU simulation times (secs) *C**	Fault coverage (%) *C**
c17	0.000	100.000	0.017	100.000	0.000	100.000
c432	0.300	99.254	0.050	94.403	5.690	99.257
c499	0.433	96.098	0.067	94.512	34.290	96.098
c880	0.533	99.294	0.083	93.952	5.890	99.297
c1355	1.017	98.044	0.117	89.731	42.230	98.044
c1908	1.550	98.910	0.217	80.228	81.209	98.810
c2670	30.867	69.124	0.283	67.603	2399.969	61.820
c3450	7.750	95.101	0.567	85.764	445.769	94.847
c5315	5.867	98.659	0.517	95.138	103.209	98.678
c6288	12.150	99.526	1.750	99.449	94.760	99.526
c7552	64.933	94.977	0.967	87.816	1516.990	95.224

Table 7.33 Simulation Results of the ISCAS 85 Benchmark Circuits Using ATALANTA, FSIM and COMPACTEST with Parity Tree as a Space Compactor (2 Inputs of Gate XOR).

Circuit name	CPU simulation times (secs) *A**	Fault coverage (%) *A**	CPU simulation times (secs) *F**	Fault coverage (%) *F**	CPU simulation times (secs) *C**	Fault coverage (%) *C**
c17	0.017	100.000	0.017	100.000	0.020	100.000
c432	0.250	97.529	0.067	94.297	29.650	89.544
c499	0.200	99.740	0.067	91.927	69.590	86.589
c880	0.583	97.158	0.083	79.158	170.900	73.053
c1355	0.767	96.465	0.100	88.258	266.230	85.038
c1908	4.067	86.222	0.217	74.775	655.550	82.936
c2670	22.650	85.976	0.267	73.894	1039.799	83.783
c3450	8.050	94.616	0.517	81.985	1047.050	88.417
c5315	13.433	96.934	0.450	89.428	1467.510	95.305
c6288	16.450	99.445	1.700	99.329	585.090	99.032
c7552	54.017	95.539	0.783	90.589	5851.779	89.106

Table 7.34 Simulation Results of the ISCAS 85 Benchmark Circuits Using ATALANTA, FSIM and COMPACTEST with Parity Tree as a Space Compactor (10 Inputs of Gate XOR).

Compactor circuit for	Hardware overhead (2-input XOR) %	Hardware overhead (10-input XOR) %
c17	14.29	14.29
c432	3.44	2.04
c499	13.19	8.11
c880	6.42	3.83
c1355	5.50	3.27
c1908	3.10	1.84
c2670	11.78	6.97
c3540	1.40	0.84
c5315	5.27	3.05
c6288	1.27	0.742
c7552	3.36	1.93

Table 7.35 Estimates of the Hardware Overhead with Parity Tree as a Space Compactor (2-input XOR Gate and 10-input XOR Gate).

To have an overall view, we compared the ATALANTA simulation results for ISCAS 85 benchmark circuits in all the proposed cases, including parity tree (using 2-input XOR gate and 10-input XOR gate), stochastic independence and stochastic dependence of multiple line errors (exhaustive approach and heuristic approach).

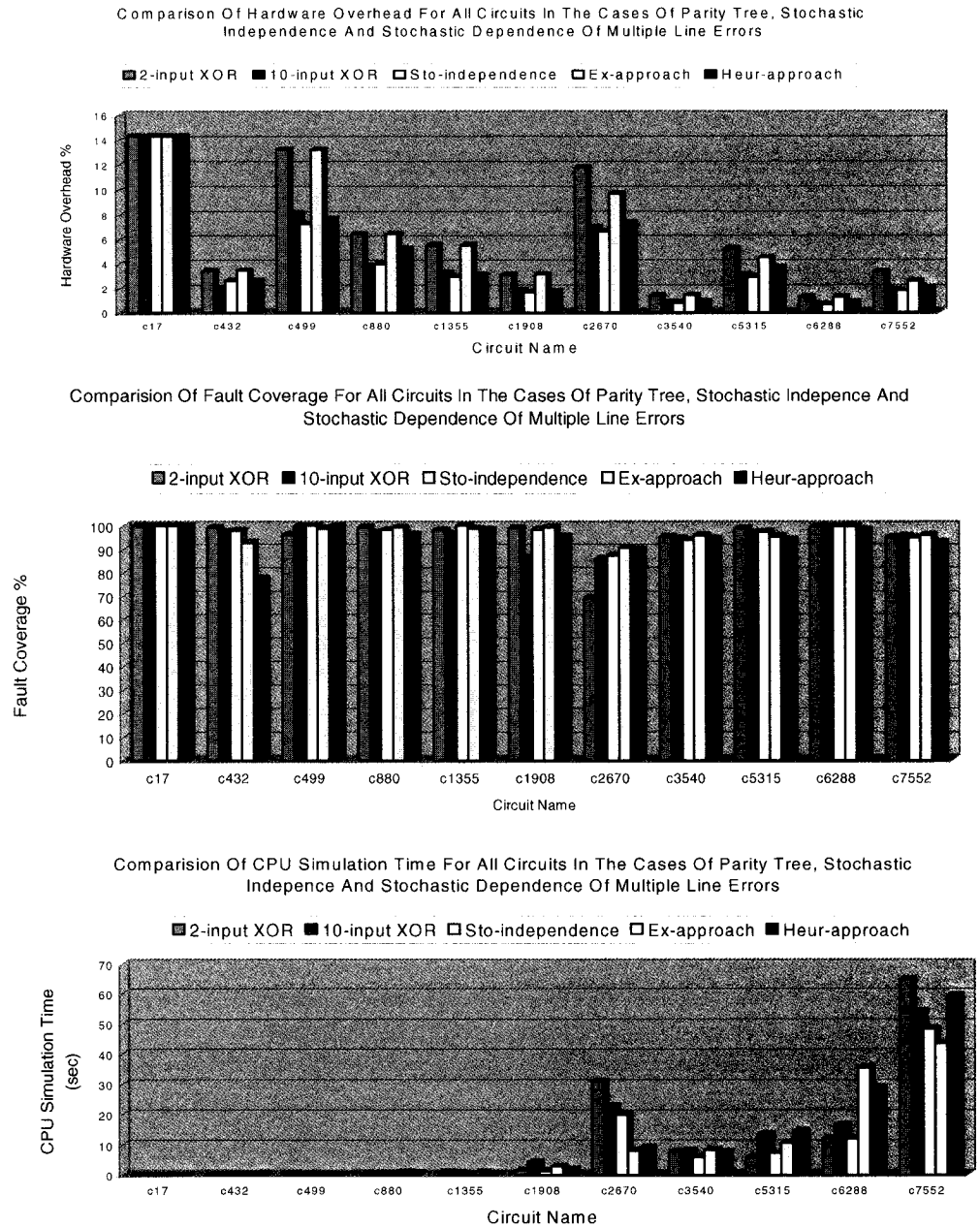


Figure 7.18 Comparison of the Fault Coverage, Hardware Overhead and CPU Simulation Time for ISCAS 85 Benchmark Circuits in the Cases of Parity Tree (2-input XOR Gate and 10-input XOR Gate), Stochastic Independence of Multiple Line Errors and Stochastic Dependence of Multiple Line Errors (Including Exhaustive Approach and Heuristic Approach) Using ATALANTA.

From the above comparison results (see Figure 7.18), we found the fault coverage results obtained by stochastic independence and stochastic dependence approaches are almost as same as those results in parity tree case. The difference is rather small and acceptable compared with parity tree. XOR gate is conceptually neat and simple, but practically it has to be implemented by using other gates, so the hardware overhead of parity tree (both using 2-input XOR and 10-input XOR) is much higher than that of our heuristic approach in practice. Therefore, the heuristic approach not only provides the satisfied fault coverage but also provides low hardware overhead and acceptable CPU simulation time for most of benchmark circuits.

We have drawn the space compaction circuits for C432 benchmark circuit in order that we can get an idea about how our space compactor looks like corresponding to stochastic independence and dependence of line errors (according to the formula in Chapter 5). There are 36 primary inputs, 160 gates and 7 primary outputs in C432 benchmark circuit. Figure 7.19 illustrates the ATALANTA /FSIM space compactor logical circuit assuming stochastic independence of multiple line errors.

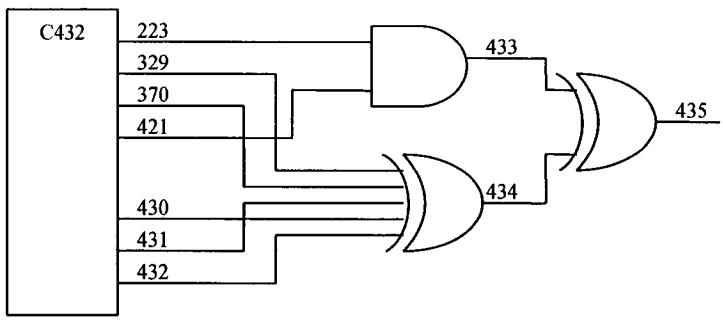


Figure 7.19 Space Compactor Circuit for c432 Assuming Stochastic independence of Multiple Line Errors.

In Figure 7.20, it illustrates the ATALANTA/FSIM space compactor logical circuit assuming stochastic dependence of multiple line errors by exhaustive approach.

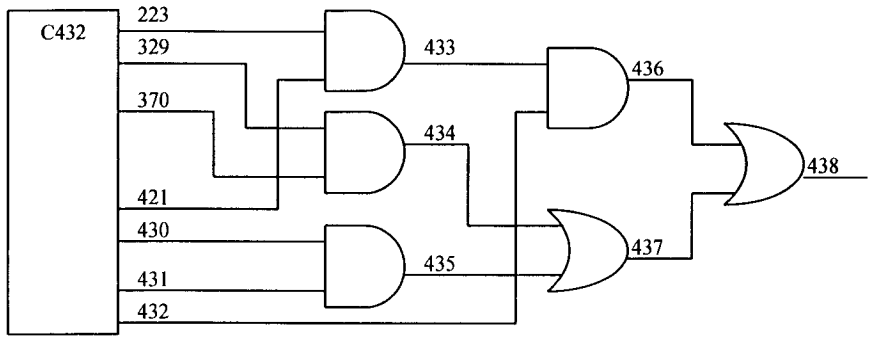


Figure 7.20 Space Compactor Circuit for c432 Assuming Stochastic Dependence of Multiple Line Errors (Exhaustive Approach).

In Figure 7.21, it illustrates the ATALANTA /FSIM space compactor logical circuit assuming stochastic dependence of multiple line errors (when t_n =2/3) by heuristic approach.

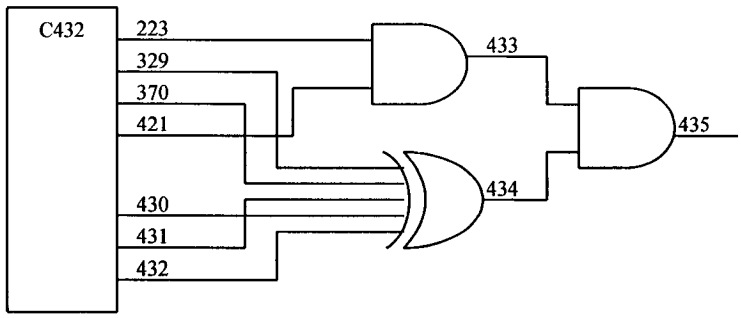


Figure 7.21 Space Compactor Circuit for c432 Assuming Stochastic Dependence of Multiple Line Errors (Heuristic Approach, t_n =2/3).

From the simulation experiments, it is obvious that in all cases, our space compactor is comparable in all respects with the parity tree space compactor. For some circuits, we obtained better fault coverage with reduction in CPU simulation time and hardware overhead using our space compactor than what we have when a parity tree space compactor is used.

In chapter 6, we have developed an approach to designing zero-aliasing space compactors using deterministic compacted test sets as well as pseudorandom testing. Conventional switching theory concepts of edge-inclusion table, frequency ordering, and concepts of strong and weak compatibilities of response data outputs, are utilized in the design, based on detectable single stuck-line faults of the MUT. To demonstrate the feasibility of the proposed zero-aliasing space compaction schemes, independent simulations were conducted on various ISCAS 85 combinational and ISCAS 89 benchmark circuits. A heuristic approach has been adopted and implemented so that we can get the results within an acceptable CPU time. We used ATALANTA [40], FSIM, and HOPE as test generation tools to generate the compacted and pseudorandom input test sets needed to test the benchmark circuits using reduced or compact test sets accompanied with a random testing session. For each circuit, we determined the number of test vectors used to construct the compaction tree, the simulation CPU time, the number of applied test vectors, and the percentage fault coverage. All the simulation results are classified in charts and tables, and are presented below.

Table 7.36, Table 7.37, and Table 7.38 show respectively, the number of maximal of compatibility classes, fault coverage, and hardware overhead for all ISCAS 85 benchmark circuits with our multiple line merger zero-aliasing compactors, using deterministic compacted test sets. Descriptions of the abbreviations used in this section can be found in APPENDDIX A.
P.S. A (*) indicates that sub maximal MCs were used in constructing the compactor.

Circuit name	PIC	POC	NGC	NAIP	NOIP	NXIP	NMCA	NMCO	NMCX
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	7	11	0	18	54	1
c1355	41	32	546	57	26	21	3614	1012	208
c1908	33	25	880	25	24	0	8	2	1
c2670	233	140	1269	204	70	9	2658	962	4
c3540	50	22	1669	36	27	1	56	24	2
c5315	178	123	2485	489	283	3	21406	14654	418
c6288	32	32	2448	3	2	1	4	4	2
c7552	207	108	3719	392	251	2	11432	6480	2

Table 7.36 Compatibility Classes for ISCAS 85 Benchmark Circuits Computed at the First Compaction Stage Using Deterministic Compacted Testing.

Circuit name	PIC	POC	NGC	NIF	TV	POCP	NL	CR	CPU	FC[%]
C17	5	2	6	22	7	1	1	1/2	0.017	100.00
c432	36	7	160	520	70	1	3	1/36	0.300	100.00
c499	41	32	202	278	55	1	2	1/32	0.050	100.00
c880	60	26	383	942	49	1	1	1/26	483.917	99.469
c880	60	26	383	942	140	1	3	1/26	1.883	100.00*
c1355	41	32	546	1558	100	1	2	1/32	1.117	100.00
c1908	33	25	880	1870	149	1	1	1/25	1.567	99.733
c1908	33	25	880	1870	215	1	4	1/25	10.067	100.00*
c2670	233	140	1269	2620	258	1	4	1/140	382.583	100.00
c3540	50	22	1669	3291	237	1	4	1/22	11.750	100.00
c5315	178	123	2485	5126	213	1	5	1/123	108.167	100.00*
c6288	32	32	2448	7710	36	1	2	1/32	6.767	100.00
c7552	207	108	3719	6946	150	1	6	1/108	982.450	100.00*

Table 7.37 Fault Coverage for ISCAS 85 Benchmark Circuits Using Deterministic Compacted Testing.

Circuit Name	NFP	NGP	AVFP	NGC	AVFC	NGCP	AO[%]
c17	2	1	2.00	6	2.00	7	14.29
c432	10	4	2.50	160	2.10	164	2.90
c499	34	3	11.33	202	2.02	205	8.21
c880	26	1	26.00	383	1.90	384	3.56
c880	29	4	7.25	383	1.90	387	3.94*
c1355	34	3	11.33	546	1.95	549	3.17
c1908	25	1	25.00	880	1.70	881	1.66
c1908	32	8	4.00	880	1.70	888	2.12*
c2670	151	12	12.58	1269	1.64	1281	7.18
c3450	36	15	2.40	1669	1.76	1684	1.21
c5315	133	11	12.09	2307	1.90	2318	3.02*
c6288	33	2	16.50	2416	1.99	2418	0.68
c7552	119	12	9.91	3513	1.75	3525	1.93*

Table 7.38 Estimates of the Hardware Overhead for ISCAS 85 Benchmark Circuits Using Deterministic Compacted Testing.

Table 7.39, Table 7.40, and Table 7.41 show respectively, the number of maximal of compatibility classes, fault coverage, and hardware overhead for all ISCAS 85 benchmark circuits with our multiple line merger zero-aliasing compactors, using pseudorandom testing.

Descriptions of the abbreviations used in this section can be found in APPENDDIX A.

P.S. A (*) indicates that sub maximal MCs were used in constructing the compactor.

Circuit name	PIC	POC	NGC	NAIP	NOIP	NXIP	NMCA	NMCO	NMCX
C17	5	2	6	0	1	0	1	2	1
c432	36	7	160	9	12	0	4	4	1
c499	41	32	202	0	0	0	1	1	1
c880	60	26	383	25	30	2	104	216	4
c1355	41	32	546	0	0	0	1	1	1
c1908	33	25	880	21	24	0	7	2	1
c2670	233	140	1269	340	206	8	1406	782	2
c3540	50	22	1669	36	27	1	56	24	2
c5315	178	123	2485	380	283	3	19228	9536	316
c6288	32	32	2448	3	2	1	4	4	2
c7552	207	108	3719	378	243	1	10228	4216	2

Table 7.39 Compatibility Classes for ISCAS 85 Benchmark Circuits Computed at the First Compaction Stage Using Pseudorandom Testing.

Circuit name	PIC	POC	NGC	NIF	TV	POCP	NL	CR	CPU	FC[%]
c17	5	2	6	22	128	1	1	½	0.033	100.00
c432	36	7	160	520	3296	1	3	1/36	0.083	100.00
c499	41	32	202	278	2624	1	1	1/32	0.083	100.00
c880	60	26	383	942	10000000	1	2	1/26	117.983	99.894
c880	60	26	383	942	113600	1	2	1/26	1.500	100.00*
c1355	41	32	546	1566	4640	1	1	1/32	0.467	100.00
c1908	33	25	880	1870	10000000	1	1	1/25	217.983	99.893
c1908	33	25	880	1870	15168	1	2	1/25	0.667	100.00*
c2670	233	140	1269	2206	398304	1	4	1/140	18.517	100.00
c3540	50	22	1669	3291	382560	1	3	1/22	19.950	100.00
c5315	178	123	2485	5952	5952	1	1	1/123	1.033	100.00
c6288	32	32	2448	7710	1666656	1	4	1/32	212.717	100.00
c7552	207	108	3719	7346	1828032	1	6	1/108	264.367	100.00*

Table 7.40 Fault Coverage for ISCAS 85 Benchmark Circuits Using Pseudorandom Testing.

Circuit name	NFP	NGP	AVFP	NGC	AVFC	NGCP	AO[%]
c17	2	1	2.00	6	2.00	7	14.29
c432	10	4	2.50	160	2.10	164	2.90
c499	32	1	32.00	202	2.02	203	7.80
c880	28	3	9.33	383	1.90	386	3.81
c880	29	4	7.25	383	1.90	387	3.94*
c1355	32	1	32.00	546	1.95	547	3.00
c1908	25	1	25.00	880	1.70	881	1.66
c1908	26	2	13.00	880	1.70	882	1.73*
c2670	150	11	13.63	1269	1.64	1280	7.14
c3450	26	5	5.20	1669	1.76	1674	0.88
c5315	123	1	123.00	2307	1.90	2308	2.80
c6288	40	8	5.00	2416	1.99	2424	0.83
c7552	134	27	4.96	3513	1.75	3540	2.16*

Table 7.41 Estimates of the Hardware Overhead for ISCAS 85 Benchmark Circuits Using Pseudorandom Testing.

Figure 7.22, Figure 7.23, and Figure 7.24 show respectively, the number applied input test patterns, compaction ratio, and the area overhead of compaction networks for all ISCAS 85 benchmark circuits with our multiple line merger zero-aliasing compactors, using deterministic compacted test sets.

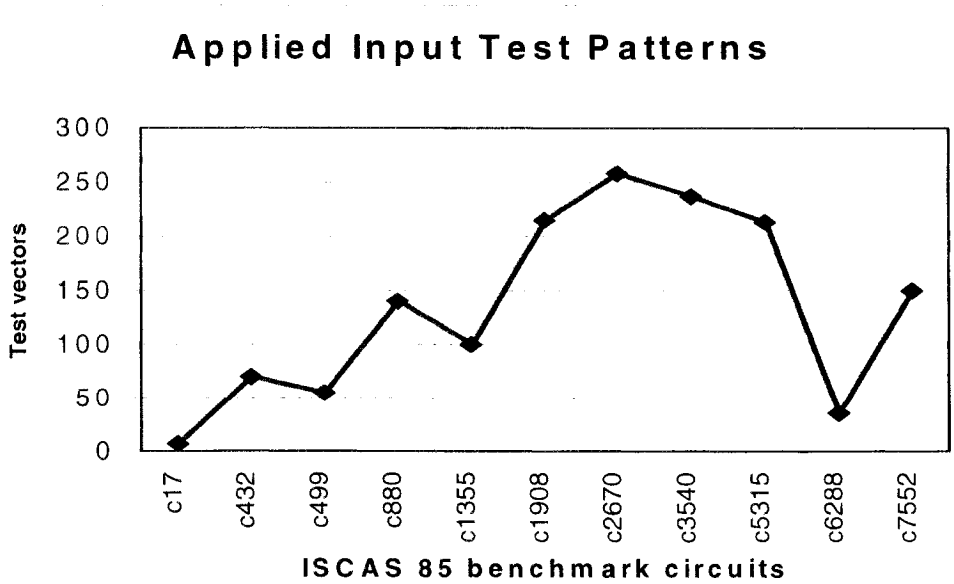


Figure 7.22 Input Test Patterns for ISCAS 85 Benchmark Circuits Using Deterministic Compacted Testing.

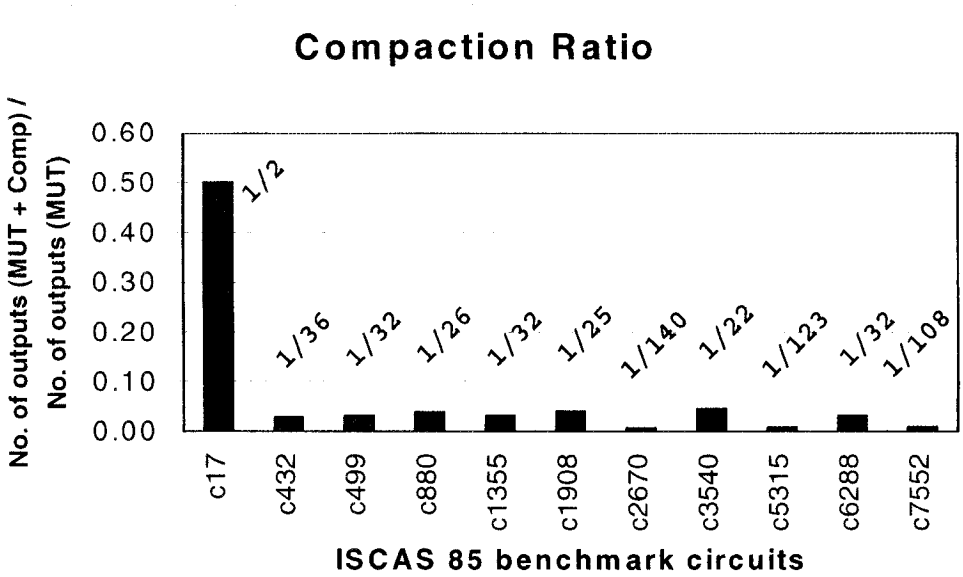


Figure 7.23 Compaction Ratio for ISCAS 85 Benchmark Circuits Using Deterministic Compacted Testing.

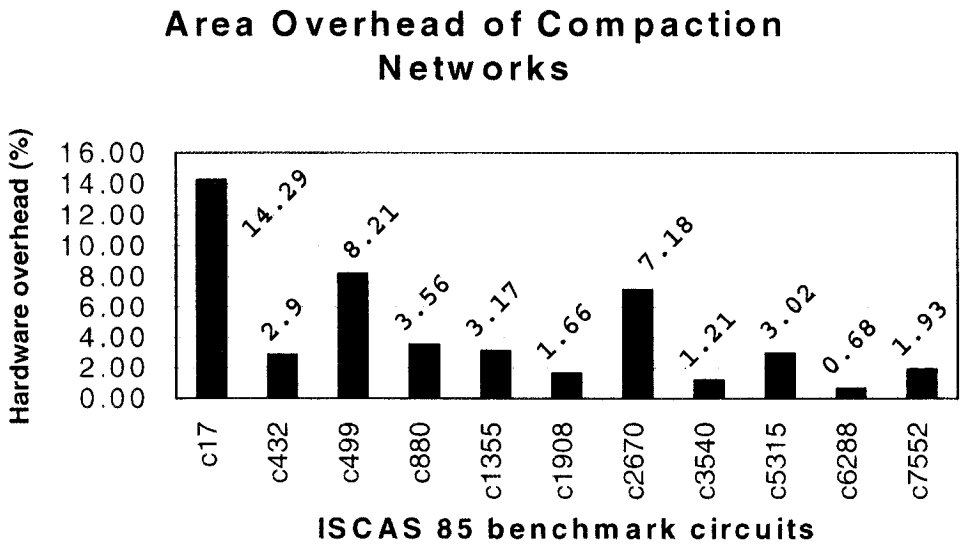


Figure 7.24 Area Overhead of Compaction Networks for ISCAS 85 Benchmark Circuits Using Deterministic Compacted Testing.

Figure 7.25, Figure 7.26, and Figure 7.27 show respectively, the number applied input test patterns, compaction ratio, and the area overhead of compaction networks for all ISCAS 85 benchmark circuits with our multiple line merger zero-aliasing compactors, using pseudorandom testing.

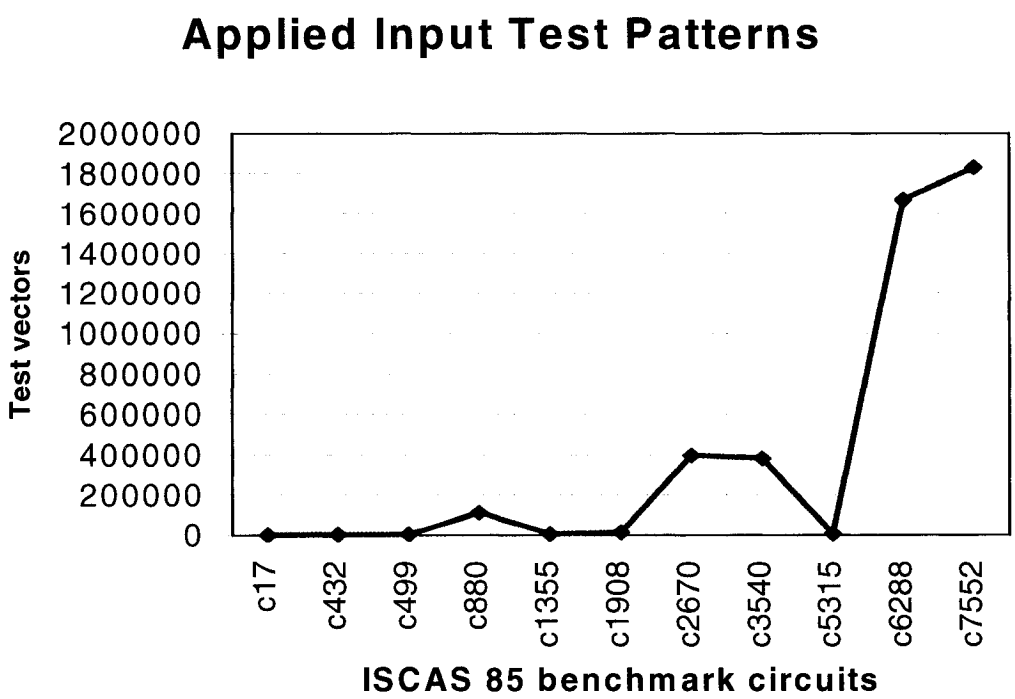


Figure 7.25 Input Test Patterns for ISCAS 85 Benchmark Circuits Using Pseudorandom Testing.

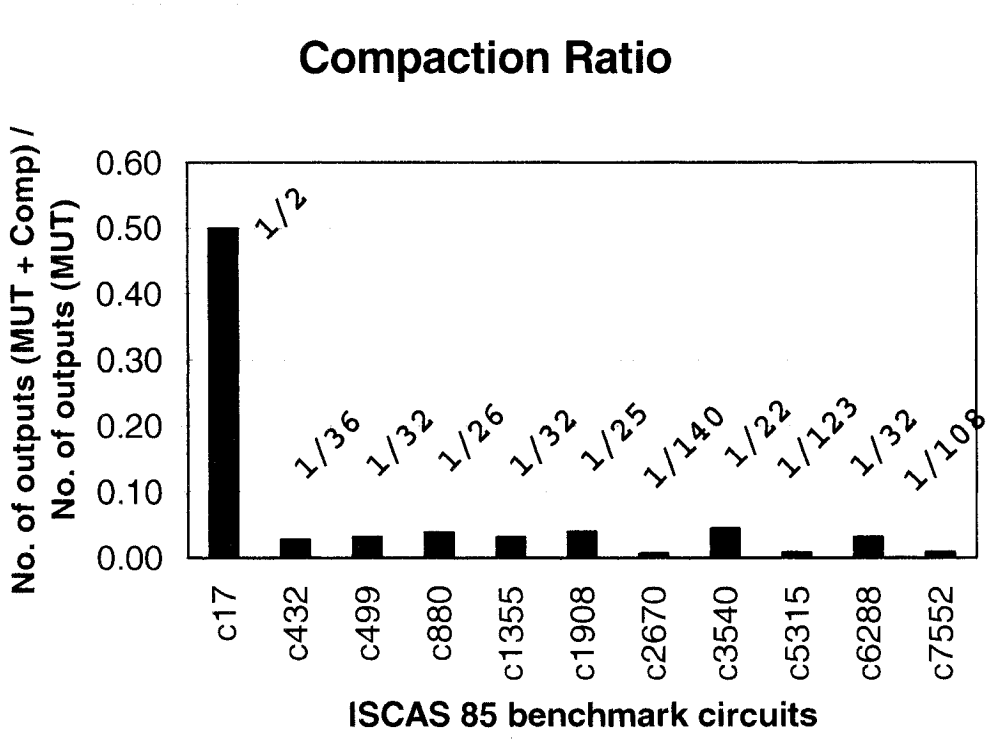


Figure 7.26 Compaction Ratio for ISCAS 85 Benchmark Circuits Using Pseudorandom Testing.

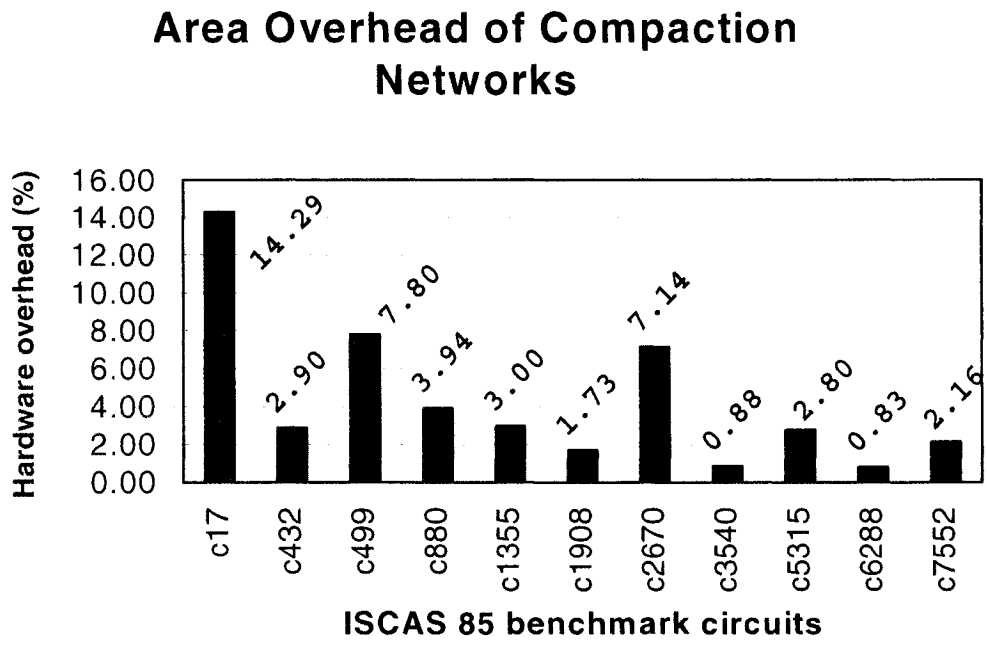


Figure 7.27 Area Overhead of Compaction Networks for ISCAS 85 Benchmark Circuits Using Pseudorandom Testing.

Table 7.42, Table 7.43, and Table 7.44 show respectively, the number of maximal of compatibility classes, fault coverage, and hardware overhead for all ISCAS 89 benchmark circuits with our multiple line merger zero-aliasing compactors, using deterministic compacted test sets. Descriptions of the abbreviations used in this section can be found in APPENDDIX A.
P.S. A (*) indicates that sub maximal MCs were used in constructing the compactor.

Circuit name	PIC	POC	NGC	NAIP	NOIP	NXIP	NMCA	NMCO	NMCX
S27	4	1	17	0	0	0	1	1	1
S208	11	2	115	1	0	0	2	1	1
S298	3	6	136	15	10	2	6	4	4
S713	35	23	447	101	165	1	70	46	2
S838	35	2	457	1	0	0	2	1	1
S953	16	23	440	175	133	134	11	512	182
S1196	14	14	561	91	53	3	14	27	8
S1238	14	14	540	91	53	3	14	27	8
S1488	8	19	667	171	110	3	19	49	8
S1494	8	19	661	171	110	3	19	49	8
S5378	35	49	2993	433	1037	72	170	112	161
S9234	36	39	5844	263	497	208	67	76	71

Table 7.42 Compatibility Classes for ISCAS 89 Benchmark Circuits Computed at the First Compaction Stage Using Deterministic Compacted Testing.

Circuit name	PIC	POC	NGC	NIF	TV	POCP	NL	CR	CPU	FC[%]
S27	4	1	17	32	44	1	0	1/1	0.033	100.00
S208	11	2	115	91	670	1	1	1/2	0.167	100.00
S298	3	6	136	184	571	2	1	2/6	0.167	100.00
S713	35	23	447	921	459	1	1	1/23	0.633	100.00
S838	35	2	457	187	269	1	1	1/2	0.233	100.00
S953	16	23	440	81	248	2	2	2/23	0.300	100.00*
S1196	14	14	561	1026	964	1	3	1/14	1.167	100.00
S1238	14	14	540	1037	964	1	2	1/14	1.150	100.00
S1488	8	19	667	775	786	1	2	1/19	0.617	100.00*
S1494	8	19	661	805	786	2	2	2/19	0.683	100.00*
S5378	35	49	2993	2180	951	3	3	3/49	6.650	100.00*
S9234	36	39	5844	357	50	3	2	3/39	0.817	100.00*

Table 7.43 Fault Coverage for ISCAS 89 Benchmark Circuits Using Deterministic Compacted Testing.

Circuit name	NFP	NGP	AVFP	NGC	AVFC	NGCP	AO[%]
S27	0	0	-	17	1.80	17	-
S208	2	1	2.00	115	1.72	116	1.00
S298	6	2	3.00	136	2.05	138	2.12
S713	23	1	23.00	447	1.50	448	3.42
S838	2	1	2.00	457	1.72	458	0.25
S953	25	4	6.25	440	1.88	444	2.99
S1196	17	4	4.25	561	1.91	565	1.57
S1238	15	2	7.50	540	2.05	542	1.35
S1488	21	3	7.00	667	2.12	670	1.47
S1494	21	4	5.25	661	2.15	665	1.46
S5378	52	5	10.40	2993	1.52	2998	1.14
S9234	43	6	7.16	5844	1.42	5850	0.51

Table 7.44 Estimates of the Hardware Overhead for ISCAS 89 Benchmark Circuits Using Deterministic Compacted Testing.

Table 7.45, Table 7.46, and Table 7.47 show respectively, the number of maximal of compatibility classes, fault coverage, and hardware overhead for all ISCAS 89 benchmark circuits with our multiple line merger zero-aliasing compactors, using pseudorandom testing.
Descriptions of the abbreviations used in this section can be found in APPENDDIX A.
P.S. A (*) indicates that sub maximal MCs were used in constructing the compactor.

Circuit name	PIC	POC	NGC	NAIP	NOIP	NXIP	NMCA	NMCO	NMCX
S27	4	1	17	0	0	0	1	1	1
S208	11	2	115	1	0	0	2	1	1
S298	3	6	136	15	11	1	6	4	2
S713	35	23	447	41	92	0	89	39	1
S838	35	2	457	1	0	0	2	1	1
S953	16	23	440	175	133	134	11	512	182
S1196	14	14	561	68	20	0	24	47	1
S1238	14	14	540	74	25	1	22	53	2
S1488	8	19	667	171	134	3	19	27	4
S1494	8	19	661	171	134	3	19	27	4
S5378	35	49	2993	2753	1929	541	3121	4258	253
S9234	36	39	5844	347	605	279	93	163	79

Table 7.45 Compatibility Classes for ISCAS 89 Benchmark Circuits Computed at the First Compaction Stage Using Pseudorandom Testing.

Circuit name	PIC	POC	NGC	NIF	TV	POCP	NL	CR	CPU	FC[%]
S27	4	1	17	32	85	1	0	1/1	0.033	100.00
S208	1	2	115	110	813803	1	1	1/2	55.633	100.00
S298	3	6	136	222	2000000	1	2	1/6	135.733	92.342
S298	3	6	136	222	1095619	1	3	1/6	95.667	100.00*
S713	35	23	447	474	824562	1	2	1/23	181.967	100.00*
S838	35	2	457	218	1185848	1	1	1/2	176.367	100.00
S953	16	23	440	81	248	2	2	2/23	0.067	100.00*
S1196	14	14	561	1238	1407989	1	3	1/14	531.850	100.00
S1238	14	14	540	1284	1407989	1	2	1/14	621.750	100.00*
S1488	8	19	667	1198	1954247	1	2	1/19	879.850	100.00*
S1494	8	19	661	1222	1954247	2	2	2/19	712.500	100.00*
S5378	35	49	2993	2929	1847028	2	3	2/49	3138.183	100.00*
S9234	36	39	5844	352	2000000	2	4	2/39	29440.617	100.00*

Table 7.46 Fault Coverage for ISCAS 89 Benchmark Circuits Using Pseudorandom Testing.

Circuit name	NFP	NGP	AVFP	NGC	AVFC	NGCP	AO[%]
S27	0	0	-	17	1.80	17	-
S208	2	1	2.00	115	1.72	116	1.00
S298	8	3	2.66	136	2.05	139	2.80
S298	9	4	2.25	136	2.05	140	3.13
S713	25	3	8.33	447	1.50	450	3.70
S838	2	1	2.00	457	1.72	458	0.25
S953	25	4	6.25	440	1.88	444	2.99
S1196	18	5	3.60	561	1.91	566	1.66
S1238	15	2	7.50	540	2.05	542	1.35
S1488	21	3	7.00	667	2.12	670	1.47
S1494	20	3	6.66	661	2.15	664	1.39
S5378	53	6	8.83	2993	1.52	2998	1.16
S9234	45	7	6.42	5844	1.42	5851	0.54

Table 7.47 Estimates of the Hardware Overhead for ISCAS 89 Benchmark Circuits Using Pseudorandom Testing.

Figure 7.28, Figure 7.29, and Figure 7.30 show respectively, the number applied input test patterns, compaction ratio, and the area overhead of compaction networks for all ISCAS 89 benchmark circuits with our multiple line merger zero-aliasing compactors, using deterministic compacted test sets.

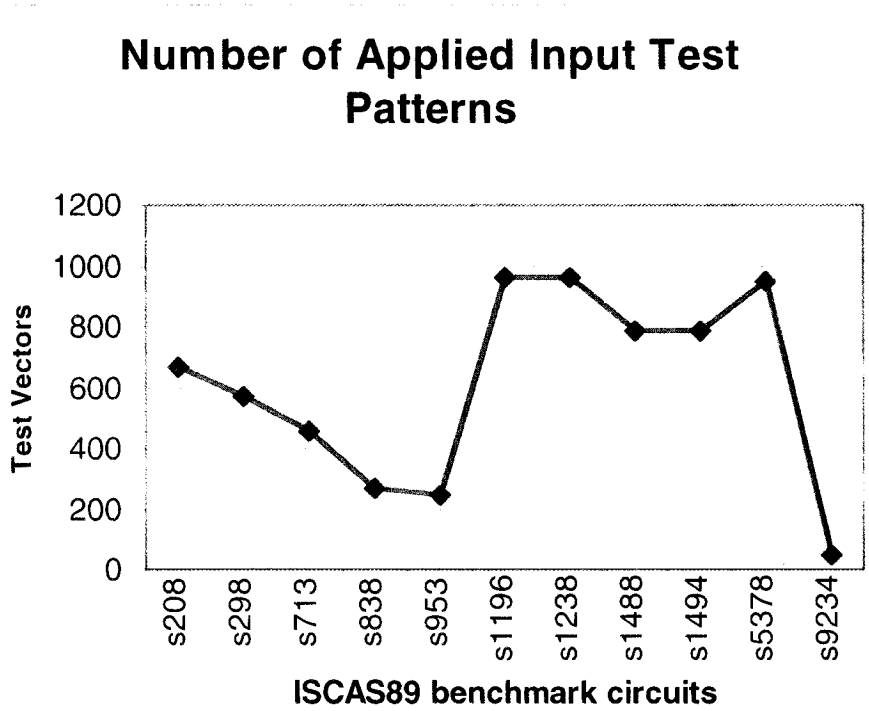


Figure 7.28 Input Test Patterns for ISCAS 89 Benchmark Circuits Using Deterministic Compacted Testing.

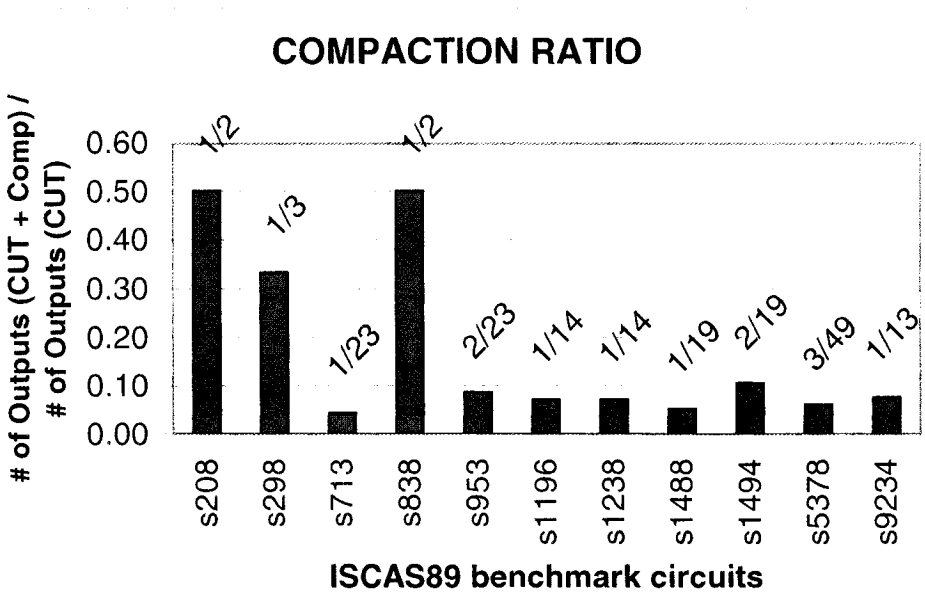


Figure 7.29 Compaction Ratio for ISCAS 89 Benchmark Circuits Using Deterministic Compacted Testing.

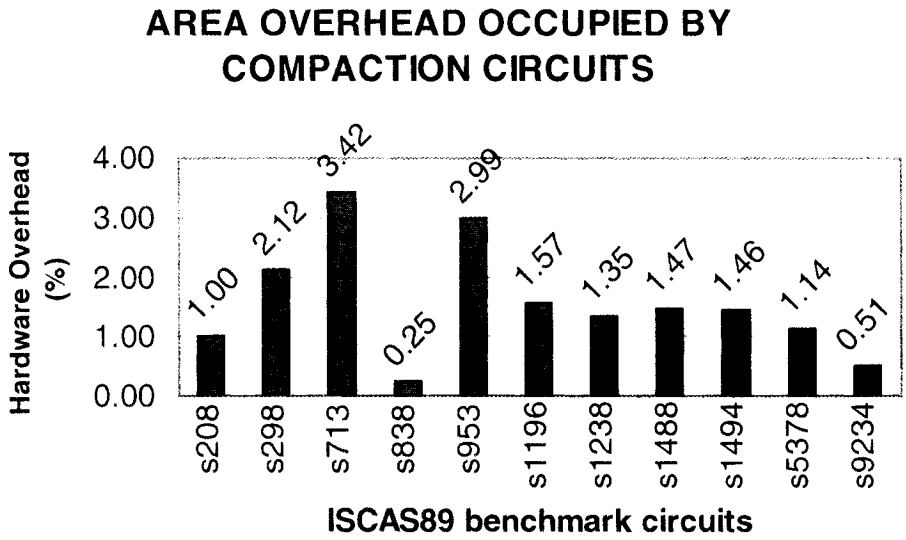


Figure 7.30 Area Overhead of Compaction Networks for ISCAS 89 Benchmark Circuits Using Deterministic Compacted Testing.

Figure 7.31, Figure 7.32, and Figure 7.33 show respectively, the number applied input test patterns, compaction ratio, and the area overhead of compaction networks for all ISCAS 89 benchmark circuits with our multiple line merger zero-aliasing compactors, using pseudorandom testing.

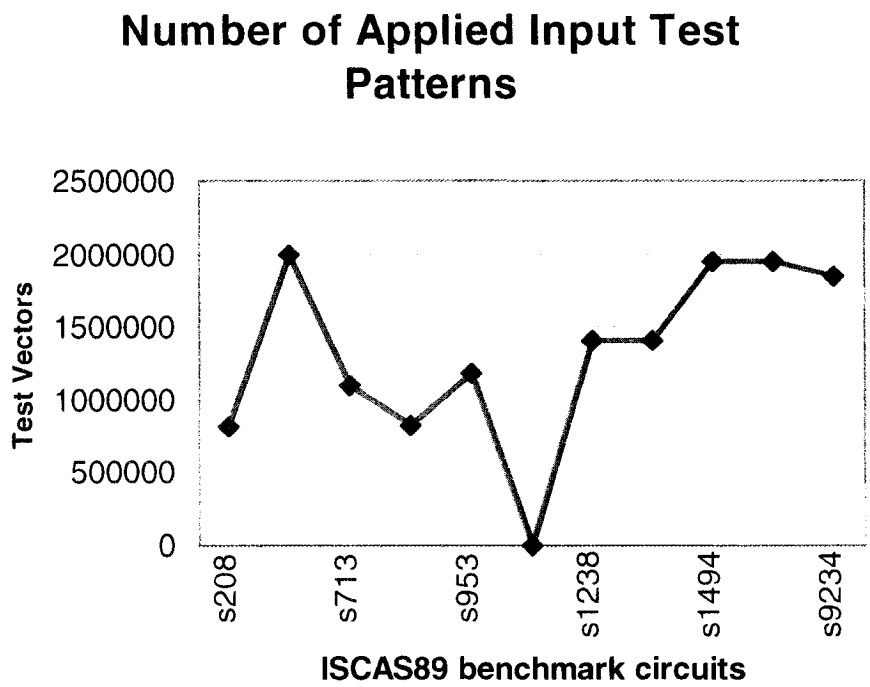


Figure 7.31 Input Test Patterns for ISCAS 89 Benchmark Circuits Using Pseudorandom Testing.

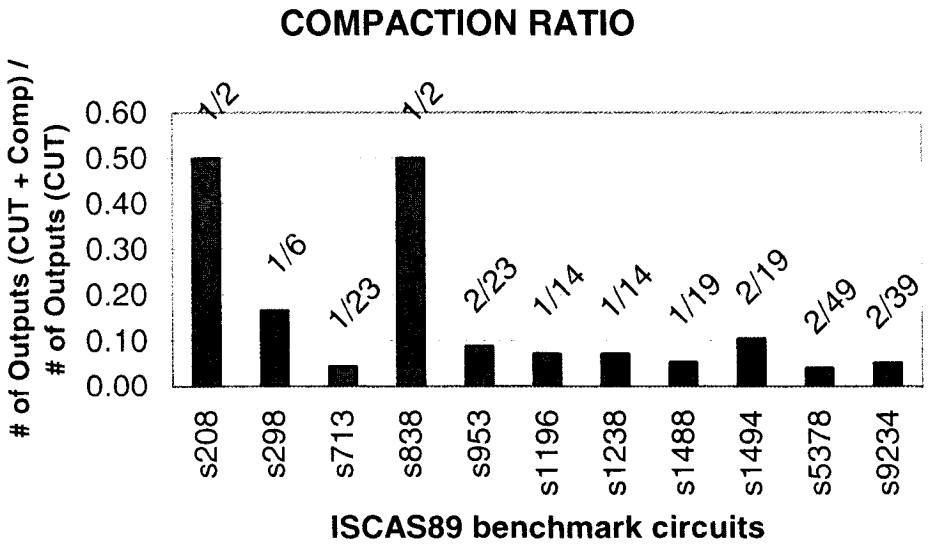


Figure 7.32 Compaction Ratio for ISCAS 89 Benchmark Circuits Using Pseudorandom Testing.

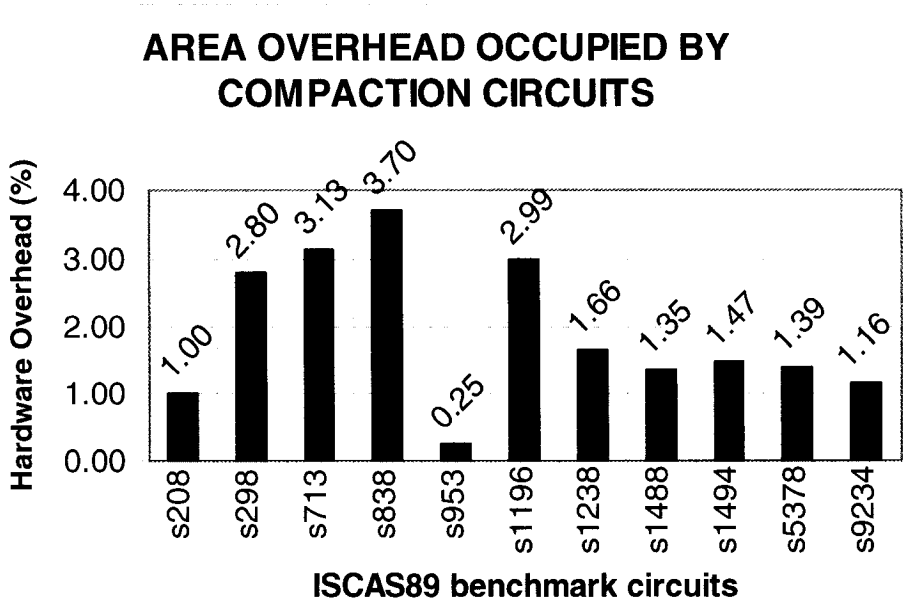


Figure 7.33 Area Overhead of Compaction Networks for ISCAS 89 Benchmark Circuits Using Pseudorandom Testing.

CONCLUSIONS

The design of space-efficient BIST support hardware is of great importance in the design and fabrication of ICs. The present dissertation reports compression techniques of test data outputs for digital IP embedded cores, which facilitate the design of such kinds of space-efficient support hardware. The proposed techniques use AND (NAND), OR (NOR), and XOR (XNOR) gates as appropriate to construct an output compaction tree that compresses the m outputs of the module under test (MUT) to ideally a single output line.

The suggested techniques take advantage of some well known concepts of conventional switching theory, particularly those of cover table and frequency ordering as commonly utilized in the minimization of switching functions, besides knowledge of Hamming distance, sequence weights, and derived sequences in the selection of specific gates for merger of an arbitrary number of output bit streams from the MUT. 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 both stochastic independence and dependence of single and multiple line errors. When we do not assume the stochastic independence case for single and multiple line errors (i.e. errors in the different lines occur independently of each other), we find that error occurrence probabilities play a distinct role in the selection of gates for merger in most cases. In this latter case (stochastic dependence), 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.

It should be recalled that the effectiveness of the proposed approaches is critically dependent on the probabilities of error occurrence in different lines of the MUT, and this dependence may be affected by the circuit structure, partitioning, etc., that is, by the number of inputs, outputs, internal lines, and types of gates they are designed of, and the way they are partitioned. Any given circuit structure has well defined features which might change based on evolving design practices with technology trends. If the ISCAS 85 and ISCAS 89 benchmarks circuits were constructed based on some recent technology like FPGA for instance, the corresponding space compactors that need be designed based on such modified circuits structure might as well be different.

Besides, in actual situations, the probability values for error occurrence or nonoccurence in particular circuits have to be experimentally determined, that is, these are *a posteriori* probabilities rather than *a priori* probabilities. If the circuit structure changes, these probability values change and obviously the corresponding compression networks that have to be designed based on optimal mergeability criteria change as well. Since the empirical formula used for computing the detectable or missed error probability estimates in our gate selection process uses exact values of these *a posteriori* probabilities of error occurrence, unless the formula is modified, whenever only intervals on the probability values are given rather than *exact* values, they cannot be used as such in the gate selection process, except only to provide two extremes of selections consistent with the probability intervals. Since the major thrust of this paper is to devise efficient methods of synthesizing compaction networks which provide improved fault coverage for fixed complexity, thereby realizing a trade off in terms of coverage and complexity (storage) than conventional techniques, the complexity issues were not addressed in depth in the current study. From the analytical viewpoint, the major issue involves the computation of the detectable or missed error probability estimates, which is rather simple in the case of two-line mergers, compared to that in the cases of generalized mergeability [7] and optimal generalized mergeability [103], where the computation is really intensive.

In the case of generalized mergeability, a heuristic approach has been adopted and implemented in order to get results within an acceptable CPU time. The heuristic approach is very useful in this case since a closed-form expression of the Nth order detectable error probability estimates can be computationally intensive. A heuristic approach has been adopted and implemented in the optimal generalized mergeability case as well, for computational reasons. Since finding the least value of the Nth order missed error probability estimate can be rather intensive in most situations with consequent strain on available CPU time and storage, heuristics will considerably reduce computational time and memory consumption under such conditions.

We did not initially emphasize designing zero-aliasing compactors [105] but rather endeavoured to reinforce the connection between the input test sets and their length and their reduction into recommended algorithms in the construction of the compaction tree. Loss of information may not be completely avoided when the size of all output responses is reduced. Therefore, depending on the amount of information loss, the corresponding space compactor designs will be affected.

The approaches developed subsequently to designing zero-aliasing space compactors [101] are based on sequence characterization, switching theory concepts such as those of edge-inclusion table and frequency ordering, and additionally utilizing concepts of strong and weak compatibilities of response data outputs. They are novel in the sense that zero-aliasing is achieved without modification of the MUT, while maximal compaction is achieved in most cases in reasonable time utilizing some simple heuristics.

In our design experiments for non-zero-aliasing compactors we used the reduced test sets provided by ATALANTA, COMPACTEST and HOPE to simulate all the ISCAS 85 combinational and ISCAS 89 full scan sequential benchmark circuits. Even though these reduced input test sets are *not* the minimal test sets needed to ensure a 100 % fault coverage, experimental results indicate that the designed space compactors are comparable in all respects with the parity tree space compactor which we used as our benchmark, and which propagates all errors that appear on an odd number of inputs and is usually considered ideal for ad hoc space compaction.

Some of the advantages inherent in the proposed methods of space compression are their simplicity, the resulting low area overhead with a high or full fault coverage for single stuck-line faults, and low CPU simulation time, which might make them suitable in a VLSI design environment as BIST support hardware. Design algorithms are proposed in the dissertation, and the simplicity and ease of their implementations are demonstrated with numerous examples. The techniques, illustrated with elaborate details through designing space compactors for all ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits with FSIM, ATALANTA, HOPE, and COMPACTEST programs, confirm the usefulness of the suggested approaches under conditions of stochastic independence and dependence of single and multiple line errors.

Finally, testing can be combined with efficient input test patterns to synthesize BIST circuits that provide more than 99 % fault coverage with small CPU simulation time and low area overhead or even 100% fault coverage for the case of zero-alising compaction. With the increase of the computational resources, in the future, our heuristic space compaction design algorithms can be improved to increase fault coverage and lower the CPU time.

Future Research

For future research, the following directions may be suggested :

- The speed and performance of the design algorithms could be substantially improved through choice of better and more efficient heuristics.

- Another possible direction could be to use better computational resources such as parallel computation.

- Developing better algorithms to realize optimal mergeability criteria for multiple line mergers for the module under test (MUT) is another possible direction.

- The compactor logic and compaction ratio depend very much on the nature and length of the test patterns being applied at the input of the MUT, and as such a precise characterization of their relationship could be promising in the search to develop suitable algorithms to design space-efficient support hardware in digital core test.

- Finally, identifying if the relationship among response data outputs of the MUT satisfies strong compatibility might significantly bring down computational requirements in the algorithm to design space compactors, a situation which could not be tackled in the research reported in the subject dissertation. This might prove a really fruitful area of future research.

BIBLIOGRAPHY

[1] B. B. Bhattacharya, A. Dmitriev, M. Goessel and K. Chakrabarty, "Synthesis of single-output space compactors with application to scan-based IP cores ", *Proc. Asia South Pacific Design Automation Conference,* pp. 496-501, 2001.

[2] A. Morosov, K. Chakrabarty, B. B. Bhattacharya and M. Goessel, "Design of parameterizable error-propagating space compactors for response observation", *Proc. IEEE VLSI Test Symposium*, pp. 48-53, 2001.

[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 Symposium,* pp. 213-219, 2000.

[5] Erik Jan Marinissen, R. Kapur, and Y. Zorian, "On Using IEEE P1500 SECT for Test Plug-n-Play", *IEEE International Test Conference*, Atlantic City, NJ, October 1-5, 2000.

[6] R. Rajsuman, *System-on-a-Chip: Design and Test*. ArtechHouse, Boston 2000.

[7] 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.

[8] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing Embedded-Core Based System Chips", *Proc. of International Test Conference*, pp. 130-143, 1998.

[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", *Proc. IEEE Instrum . Meas. Tech. Conf.*, pp. 198-203, vol. 1, 2001.

[10] Y. Zorian, S. Dey, and M. J.Rodgers, "Test of Future System-on-Chips", *Proc. of the 2000 Int. Conf. on Computer-Aided Design*, pp. 392-398, November 2000.

[11] A. Chandra and K. Chakrabarty, "Test Resource Partitioning and Reduced Pin-Count Testing Based on Test Data Compression", *Proc. Design Automation and Test in Europe (DATE) Conference,* pp. 598-603, Paris, 2002.

[12] B. Pouya and N. A. Touba, "Synthesis of zero-aliasing elementary-tree space compactors", *Proc. of IEEE VLSI Test Symposium*, pp. 70-77, 1998.

[13] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test-Data Compression and Decompression Architectures Based on Golomb Codes", *IEEE Transactions on CAD/ICAS*, vol. 20, pp. 355-368, March 2001.

[14] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test", *Proc. IEEE International Test Conference*, pages 911-920, 2000.

[15] K. Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming", *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 10, October 2000.

[16] P. Chauhan, E. M. Clarke, Y. Lu, and D. Wang, "Verifying IP-Core based System-On-Chip Designs", *IEEE Intl. ASIC/SOC Conference*, 1999.

[17] S. R. Das, M. H. Assaf, and A. Nayak, "Selecting outputs for merger in space compression using concepts of hamming distance and sequence weights", *IASTED International Conference on Modelling, Simulation and Optimization,* Gold Coast, Australia, May 6-9, 1996.

[18] H. K. Lee and D. S. Ha, "New techniques for improving parallel fault simulation in synchronous sequential circuits", Proc. International Conference on Computer-Aided Design, pp. 10-17, October 1993.

[19] H. K. Lee and D. S. Ha, "HOPE : An efficient parallel fault simulator for synchronous sequential circuits", Proc. Design Automation Conference, pp. 336-340, June 1992.

[20] K. Chakrabarty and J. P. Hayes, "Cumulative balance testing of logic circuits", *IEEE Transactions on VLSI Systems,* pp. 72-83, March 1995.

[21] Y. Zorian, and E. J. Marinissen, "System Chip Test: How Will It Impact Your Design?", pp. 136-141, *Proceedings of the 37th Conference on Design Automation*, Los Angeles, CA, June 5-9, 2000.

[22] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A tutorial on built-in self-test (part 2) ", *IEEE Design and test of Computers*, vol. 10, pp. 69-77, June 1993.

[23] V. D. Agrawal et al. , "Built-in self-test for digital circuits", *AT&T Technical Journal*, vol. 73, pp. 30-39, March/April 1994.

[24] V. K. Agarwal, "Increasing effectiveness of built-in testing by output data modification", *Proc. FTCS-13*, pp. 227-234, June 1983.

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

[26] K. Chakrabarty and J. P. Hayes, "Efficient test response compression for multiple-output circuits", *Proc. 1994 Int. Test Conference*, pp. 501-510, 1994.

[27] K. Chakrabarty. *Design and analysis of efficient response compaction schemes. Ph. D. thesis*, University of Michigan, 1995.

[28] K.-T. Cheng, V. D. Agrawal. Unified methods for VLSI simulation and test generation. *Kluwer Academic Publishers*, Massachusetts, 1989.

[29] R. G. Daniels and W. B. Bruce, "Built-in self-test trends in Motorola microprocessors", *IEEE Design and Test of Computers*, vol. 2, pp. 64-71, April 1985.

[30] 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.

[31] S. Devadas, A. Ghosh, and K. Keutzer. Logic synthesis. *McGraw-Hill Series on Computer Engineering*, New York, 1994.

[32] R. A. Frohwerk, "Signature analysis: A new digital field service method", *Hewlett-Packard Journal*, vol. 28, pp. 2-8, September 1977.

[33] M. C. Hansen and J. P. Hayes, "High-level test generation using physically-induced faults", *Proc. 1995 VLSI Test Symp.* pp. 20-28, 1995.

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

[35] J. P. Hayes, "Transition count testing of combinational logic circuits", *IEEE Transactions on Computers*, vol. C-25, pp. 613-620, June 1976.

[36] T. C. Hsiao and S. C. Seth, "An analysis of the use of Rademacher-Walsh spectrum in compact testing", *IEEE Transactions on Computers*, vol. 33, pp. 934-937, October 1984.

[37] 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.

[38] M. Karpovsky and P. Nagvajara, "Optimal time and space compression of test responses for VLSI devices", *Proc. 1987 Int. Test Conference*, pp. 523-529, 1987.

[39] H. K. Lee and D. S. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation", *Proc. 1991 Int. Conference*, pp. 946-955, 1991.

[40] H. K. Lee and D. S. Ha. On the Generation of Test Patterns for Combinational Circuits. *Technical Report No. 12-93*, Dept. of Electrical Eng. Virginia Polytechnic Institute and State University.

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

[42] I. Pomeranz, L. N. Reddy and S. M. Reddy, "Compactest: A method to generate compact test sets for combinational circuits", *Proc. 1991 International Test Conference*. pp. 194-203, 1991.

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

[44] S. M. Reddy, K. K. Saluja and M. G. Karpovsky, "A data compression technique for built-in self-test", *IEEE Transactions on Computers*, vol. C-37, pp. 1151-1156, September 1988.

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

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

[47] N. R. Saxena and J. P. Robinson, "Syndrome and transition count are uncorrelated", *IEEE Transactions on Information Theory*, vol. 34, pp. 64-69, January 1988.

[48] N. R. Saxena and J. P. Robinson, "A unified view of test response compression methods", *IEEE Transactions on Computers*, vol. C-36, pp. 94-99, January 1987.

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

[50] T. W. Williams, K. P. Parker, "Testing logic networks and design for testability", *Computer*, vol. 21, pp. 9-21, October 1979.

[51] T. W. Williams, W. Daehn, M. Gruetzner, and C. W. Starke, "Aliasing errors in signature analysis registers", *IEEE Design and Test of Computers*, vol. 4, pp. 39-45, April 1987.

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

[53] S. R. Das, E. M. Petriu, T. Barakat, M. H. Assaf and A. R. Nayak , "Space compaction under generalized mergeability", IEEE Transactions on Instrumentation and Measurement, vol. 47, pp. 1283-1293, October 1998.

[54] M. Karpovsky and P. Nagvajara, "Optimal robust compression of test responses", IEEE Transactions on Computers, vol. C-39, pp. 138-141, January 1990.

[55] E. J. Marinissen and Y. Zorian., "Challenges in Testing Core-Based System ICs", *IEEE Communications Magazine*, 37(6) pages: 104–109, June 1999.

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

[57] S. M. Reddy, K. K.aluja, and M. G. Karpovsky, "Data compression technique for test responses",IEEE Transactions on Computers, vol. C-37, pp. 1151-1156, September 1988.

[58] J. Savir, "Reducing the MISR size", IEEE Transactions on Computers, vol. C-45, pp. 930-938, August 1996.

[59] 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.

[60] M. H. Assaf, *Space Compactor Design for Built-In Self-Testing of VLSI Circuits from Compact Test Sets using Sequence Characterization and Failure Probabilities.* M.A.Sc. Thesis, Department of Electrical Engineering, University of Ottawa, Ottawa, Ontario, August 1996.

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

[62] J. Y. Liang. *Response Data Compaction in BIST under Generalized Mergeability Based on Switching Theory Formulation and utilizing a New Measure of Failure Probability.* M.A.Sc. thesis, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, September 2000.

[63] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman, 1979.

[64] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks ", *IEEE Trans. Comput* ., vol. C-24, pp. 668-670, June 1975.

[65] A. Gupta, *Groundwork of Mathematical Probability and Statistics.* Calcutta, India: Academic Publishers, 1988 (Third Edition).

[66] A. K. Choudhury and S. R. Das, "Some studies on connected cover term matrices of switching functions ", *Int* . *J. Contr.*, vol. 2, pp. 441-501, November 1965.

[67] S. R. Das and N. S. Khabra, "Clause-column table approach for generating all the prime implicants of switching functions", *IEEE Trans. Comput.*, vol. C-21, pp. 1239-1246, November 1972.

[68] W. B. Jone and S. R. Das, "Multiple-output parity bit signature for exhaustive testing", J. of Electron. Testing : Theory and Applications, vol. 1, pp. 175-178, March 1990.

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

[70] N. S. Khabra and S. R. Das, "Multiform partial symmetry and parity functions", *IEEE Trans. Comput* ., vol. C-22, p. 804, August 1973.

[71] Semiconductor Industry Association Roadmap 1997, http://notes.sematech.org/ntrs/PublNTRS.nsf.

[72] Inventra core library, Mentor Graphics, http://www.mentorg.com/inventra/

[73] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores", *Proc. IEEE Int. Test Conf.*, pages 284-293, Washington, DC, October 1998.

[74] P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-Based System Chips", *Proc. IEEE Int. Test Conf.*, pages 294-302, Washington, DC, October 1998.

[75] L. Whetsel, "Addressable Test Ports: An Approach to Testing Embedded Cores", *Proc. IEEE Int. Test Conf.*, pages 1055-1064, Atlantic City, NJ, September 1999.

[76] IEEE P1500 Web Site. http://grouper.ieee.org/groups/1500/.

[77] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, "Towards a Standard for Embedded Core Test: An Example", *Proc. IEEE Int. Test Conf.*, pages 616-627, Atlantic City, NJ, September 1999.

[78] 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. On Circuits and Systems,* Lansing, Michigan, August 2000.

[79] S. R. Das, A. R. Nayak, M. H. Assaf, and W. B. Jone, "Realizing ultimate compression with acceptable fault coverage degradation to reduce MISR size in BIST applications by nonexhaustive test patterns", *1997 International Symp. on Circuit and Systems*, pages 2717-2720, June 1997.

[80] JBits SDK, http://www.xilinx.com/products/jbits/

[81] Xilinx, Inc., The Programmable Logic Data Book. 1999.

[82] P. Sundararajan and S. A. Guccione, "XVPI : A Portable Hardware / Software Interface for Virtex", in J. Schewel, et al., editors, Reconfigurable Technology : FPGAs for Computing and Applications II, *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 4212, pp. 90-95, November 2000.

[83] D. Levi and S. A. Guccione, "Genetic FPGA : A Java-Based Tool for Evolving Stable Circuits", in J. Schewel, et al., editors, Reconfigurable Technology : FPGAs for Computing and Applications, *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 3844, pp. 114-121, September 1999.

[84] P. Bellows and B. Hutchings, "JHDL - An HDL for Reconfigurable Systems", *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1998.

[85] S. A. Guccione and D. Levi, "XBI : A Java-Based Interface to FPGA Hardware", in J. Schewel, et al., editors, Configurable Computing : Technology and Applications, *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 3526, pp. 97-102, November 1998.

[86] Yoneda, T. and Fujiwara, H., "A DFT Method for Core-based Systems-on-a-chip Based on Consecutive Testability", *Asian Test Symposium, Proceedings 10th*, pp. 193-198, Kyoto, Japan, 2001.

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

[88] Sinanoglu, O. and Orailoglu, A., "Efficient Construction of Aliasing-free Compaction Circuitry", *Micro IEEE*, pp. 82-92, Vol. 22, Oct. 2002.

[89] Sinanoglu, O. and Orailoglu, A., "Compaction Schemes with Minimum Test Application Time", Asian Test Symposium, Proceedings 10th, pp. 199-204, Kyoto, Japan 2001.

[90] Sogomonyan, E.S. Morosov, A. Gossel, M. Singh, A. and Rzeha, J., "Early Error Detection in Systems-on-chip for Fault-tolerance and At-speed Debugging", *VLSI Test Symposium, 19th IEEE Proceedings on. VTS*, pp. 184-189, Marina Del Rey, CA , USA, 2001.

[91] Ravi, S. and Jha, N.K., "Synthesis of System-on-a-chip for Testability", *VLSI Design, Fourteenth International Conference*, pp. 149-156, Bangalore , India, 2001.

[92] Al Zahir, S. El-Maleh, A. and Khan, E., "An Efficient Test Vector Compression Technique Based on Geometric Shapes [System-on-a-chip] ", *Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference*, pp. 1561-1564, vol.3, Malta, 2001.

[93] Maroufi, W. Benabdenbi, M. and Marzouki, M., "Solving the I/O Bandwidth Problem in System on a Chip Testing", *Proceedings 13th Symposium on Integrated Circuits and Systems Design*, pp. 9-14, Manaus , Brazil, 2000.

[94] Chiusano, S. Prijetto, P. and Wunderlich, H.-J., "Non-intrusive BIST for Systems-on-a-chip", *Proceedings International Test Conference*, pp. 644-651, Atlantic City, NJ , USA, 2000.

[95] Jervan, G. Peng, Z. and Ubar, R., "Test Cost Minimization for Hybrid BIST", *Proceedings. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 283-291, Yamanashi , Japan, 2000.

[96] Jas, A. Mohanram, K. and Touba, N.A., "An embedded core DFT scheme to obtain highly compressed test sets", Proceedings. *Eighth Asian Test Symposium, 1999. (ATS '99)*, pp. 275-280, Shanghai, China, 1999.

[97] Chandra, A. and Chakrabarty, K., "Reduction of SoC Test Data Volume, Scan Power and Testing Time Using Alternating Run-length Codes", *Proceedings. 39th Design Automation Conference*, pp. 673-678, 2002.

[98] Xiaoqing Wen and Hsin-Po Wang, "A Flexible Logic BIST Scheme and its Application to SoC Designs", *Proceedings. 10th, Asian Test Symposium*, pp. 463- ,Kyoto , Japan, 2001.

[99] Benso, A. Chiusano, S. Di Carlo, S. Prinetto, P. Ricciato, F. Spadari, M. and Zorian, Y., "HD2BIST: a hierarchical framework for BIST scheduling, data patterns delivering and diagnosis in SoCs", *Proceedings. International Test Conference*, pp. 892-901, Atlantic City, NJ, USA, 2000.

[100] Sinanoglu, O. and Orailoglu, A., "Space and Time Compaction Schemes for Embedded Cores", *Proceedings. International Test Conference*, pp. 521-529, Baltimore, MD, USA, 2001.

[101] M. H. Assaf, S. R. Das, E. M. Petriu and S. Mukherjee, "Design of Aliasing-Free Space Compressor in BIST with Maximal Compaction Ratio using Concepts of Strong and Weak Compatibilities of Response Data Outputs and Generalized Sequence Mergeability", *International Workshop on Distributed Computing*, Calcutta, India, Dec. 28-31, 2002.

[102] K. Chakrabarty, V. Iyengar and A. Chandra, *Test Resource Partitioning for System-on-a-Chip*, Kluwer Academic Publishers, Norwell, MA, 2002.

[103] S. R. Das, M. H. Assaf, J. Liang, E. M. Petriu and W. B. Jone, "A New Appraisal of Response Compaction Efficiency in Space Based on Detectable Single Stuck Line Faults Subjected to Complete or Near Complete Set of Tests Using Nth Order Missed Error Probability Estimates", *IASTED International Conference on Modeling, Identification and Control*, pp. 913-918, Innsbruck, Austria, Feb. 19-22, 2001.

[104] K. Chakrabarty, ed., *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation*, Kluwer Academic Publishers, Norwell, MA, 2002.

[105] Chakrabarty, K. Murray, B.T. and Hayes, J.P.,"Optimal Zero-aliasing Space Compaction of Test Responses", IEEE Transactions on Computers, pp. 1171-1187, Vol. 47, Issue 11, Nov. 1998.

[106] Tarnick, S., " Bounding Error Masking in Linear Output Space Compression Schemes", *Proceedings of the Third Asian Test Symposium*, pp. 27-32, Nara , Japan, Nov. 1994.

[107] Zorian, Y. and Ivanov, A.," Programmable Space Compaction for BIST", *FTCS-23. Digest of Papers. The Twenty-Third International Symposium on Fault-Tolerant Computing*, pp. 340-349, Toulouse, France, Jun. 1993.

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

[109] Chakrabarty, K., "Zero-aliasing Space Compaction Using Linear Compactors with Bounded Overhead", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 452-457, Vol. 17, May 1998.

[110] Chakrabarty, K. and Hayes, J.P., "Zero-aliasing Space Compaction of Test Responses using Multiple Parity Signatures", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 309-313, Jun. 1998.

[111] Seuring, M. Gossel, M. and Sogomonyan, E., "A Structural Approach for Space Compaction for Concurrent Checking and BIST", *Proceedings. 16th IEEE VLSI Test Symposium*, pp. 354-361, Monterey, CA, USA, Apr. 1998.

[112] Chakrabarty, K. and Hayes, J.P., "Test Response Compaction Using Multiplexed Parity Trees", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1399-1408, Vol. 15, Issue 11, Nov. 1996.

[113] Bhattacharya, B.B. Dmitriev, A. and Goessel, M., "Zero-aliasing Space Compression Using a Single Periodic Output and its Application to Testing of Embedded Cores", *Thirteenth International Conference on VLSI Design*, pp. 382-387, Calcutta , India, 2000.

[114] Jin Ding Moloney, D. and Xiaojun Wang, "Aliasing-free Space and Time Compactions with Limited Overhead", *Proceedings. IEEE 2000 First International Symposium on Quality Electronic Design*, pp. 355-360, San Jose, CA , USA, Mar. 2000.

[115] Bhattacharya, B.B. Dmitriev, A. Gossel, M. and Chakrabarty, K., "Synthesis of Single-output Space Compactors with Application to Scan-based IP Cores", *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference*, pp. 496-501, Yokohama , Japan, 2001.

[116] S. R. Das, "On a New Approach for Finding all the Modified Cut-sets in an Incompatibility Graph", *IEEE Trans. Comput.* C-22, pp. 187-193, Feb. 1973.

[117] Tsiatouhas, Y. and Haniotakis, T., "A Zero aliasing Built-in self Test Technique for Delay Fault Testing", *International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 95-100, Albuquerque, NM , USA, Nov. 1999.

LIST OF PAPERS BY THE AUTHOR ON SOME OF WHICH PORTION OF THE PRESENT DISSERTATION IS BASED

A. PAPERS PUBLISHED IN REFEREED JOURNALS

A1. S. R. Das, C. V. Ramamoorthy, 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*, **IEEE Transactions on Instrumentation and Measurement**, Vol. 52, No. 3, June 2003.

A2. S. R. Das, M. Suderma, 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*, **IEEE Transactions on Instrumentation and Measurement**, Vol. 52, No. 1, February 2003.

A3. 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 Transactions on Instrumentation and Measurement**, Vol. 50, No. 6, December 2001, pp. 1725-1747 - **Recipient** of **IEEE Donald Fink Prize Paper Award** for **2003**.

A4. S. R. Das, T. F. Barakat, E. M. Petriu, M. H. Assaf and K. Chakrabarty : *Space compression revisited,* **IEEE Transactions on Instrumentation and Measurement**, Vol. 49, No. 3, June 2000, pp. 690-705 – **Nominated for IEEE Transactions on Instrumentation and Measurement Prize Paper Award**.

A5. S. R. Das, M. Assaf and A. R. Nayak : *On the design of space compressor for VLSI circuits in BIST using nonexhaustive test sets*, **Transactions of the Society for Design and Process Science**, Vol. 2, No. 4, December 1998, pp. 1-12.

A6. S. R. Das, E. M. Petriu,T. F. Barakat, M. H. Assaf and A. R. Nayak : *Space compression under generalized mergeability*, **IEEE Transactions on Instrumentation and Measurement**, Vo. 47, No. 5, October 1998, pp. 1283-1293.

B. PAPERS SUBMITTED AND ON REVISION FOR PUBLICATION IN REFEREED JOURNALS

B1. S. R. Das, M. H. Assaf, E. M. Petriu and W. B. Jone : *Fault simulation and response compaction in full scan circuits using HOPE*, **IEEE Transactions on Instrumentation and Measurement** (on consideration).

B2. S. R. Das, T. Barakat, A. R. Nayak, M. H. Assaf and E. M. Petriu : *Generalized mergeability in space compressor design in built-in self-test of VLSI circuits*, **IEEE Transactions on Instrumentation and Measurement (provisionally accepted** – currently on revision).

B3. S. R. Das, C. V. Ramamoorthy, M. H. Assaf, E. M. Petriu and W. B. Jone : *Space compactor design for built-in self-testing of VLSI circuits from compact test sets using sequence characterization and failure probabilities*, **IEEE Transactions on VLSI Systems (provisionally accepted** – currently on revision).

C. PAPERS PUBLISHED IN REFEREED CONFERENCES, SYMPOSIA, WORKSHOPS, AND SEMINARS PROCEEDINGS

C1. S. R. Das, M. H. Assaf and E. M. Petriu : *Response compaction in space for full scan circuits using compact test sets based on sequence characterization*, **6th World Conference on Integrated Design and Process Technology**, Pasadena, CA, U.S.A., June 23-28, 2002 (**Conference Proceedings**, Paper 40.3, pp. 1-8).

C2. S. R. Das, M. H. Assaf, E. M. Petriu and W. B. Jone : *Fault simulation and response compaction in full scan circuits using HOPE*, Presented at the **19th IEEE Instrumentation and Measurement Technology Conference**, Anchorage, AK, U.S.A., May 21-23, 2002 (**Conference Proceedings**, Vol. 1, pp. 607-612).

C3. S. R. Das, M. H. Assaf and E. M. Petriu : *Response compaction in full scan circuits using compact test sets*, Presented at the **IASTED International Conference on Modeling, Identification and Control**, Innsbruck, Austria, February 18-21, 2002 (**Conference Proceedings**, pp. 77-82).

C4. 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*, Presented at the **18th IEEE Instrumentation and Measurement Technology Conference**, Budapest, Hungary, May 21-23, 2001 (**Conference Proceedings**, Vol. 1, pp. 198-203).

C5. S. R. Das, M. H. Assaf, J. Liang, E. M. Petriu and W. B. Jone : *A new appraisal of response compaction efficiency in space based on detectable single stuck line faults subjected to complete or near complete set of tests using Nth order missed error probability estimates*, Presented at the **IASTED International Conference on Modeling, Identification and Control**, Innsbruck, Austria, February 19-22, 2001 (**Conference Proceedings**, pp. 913-918).

C6. S. R. Das, M. Sudarma, J. Liang, 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 compact test sets*, Presented at the **43rd Midwest Symposium on Circuits and Systems**, East Lansing, MI, U.S.A., August 24-26, 2000 (**Symposium Proceedings**, pp. 1-4).

C7. S. R. Das, T. F. Barakat, E. M. Petriu, M. H. Assaf and K. Chakrabarty : *Space compression revisited*, Presented at the **16th IEEE Instrumentation and Measurement Technology Conference**, Venice, Italy, May 24-26, 1999 (**Conference Proceedings**, Vol. 2, pp. 849-854).

C8. S. R. Das, T. F. Barakat, E. M. Petriu, M. H. Assaf and W. B. Jone : *Space compression in mlti-output VLSI circuits using concept of generalized output mergeability – analysis and implementation results*, Presented at the **18th IASTED International Conference on Modeling, Identification and Control**, Innsbruck, Austria, February 15-18, 1999 (**Conference Proceedings**, pp. 541-544).

C9. 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*, Presented at the **Third World Conference on Integrated Design and Process Technology**, Berlin, Germany, July 6-9, 1998 (**Conference Proceedings**, Vol. 6, pp. 92-99).

C10. S. R. Das, E. M. Petriu, T. Barakat, M. H. Assaf and A. R. Nayak : *Space compression under generalized mergeability*, Presented at the **15th IEEE Instrumentation and Measurement Technology Conference**, St. Paul, MN, U.S.A., May 18-21, 1998 (**Conference Proceedings**, Vol. One, pp. 519-524).

C11. S. R. Das, E. M. Petriu, T. Barakat, M. H. Assaf and A. R. Nayak : *Generalized detectable error probability estimate and output data compression in integrated circuit design under different multiplicities of error*, Presented at the **IASTED International Conference on Modeling, Simulation and Optimization**, Singapore, August 11-13, 1997 (**Conference Proceedings**, pp. 195-198).

C12. S. R. Das, A. R. Nayak, M. H. Assaf and W. B. Jone : *Realizing ultimate compression with acceptable fault coverage degradation toreduce MISR size in BIST applications by nonexhaustive test patterns*, Presented at the **IEEE International Symposium on Circuits and Systems (ISCAS)**, Hong Kong, June 9-12, 1997 (**Symposium Proceedings**, Vol. IV, pp. 2717-2720).

C13. 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*, Presented at the **14th IEEE Instrumentation and Measurement Technology Conference**, Ottawa, Ontario, May 19-21, 1997 (**Conference Proceedings**, Vol. 2, pp. 1448-1453).

C14. S. R. Das, M. H. Assaf, A. R. Nayak and W. B. Jone : *Fault tolerance in systems design in VLSI using data compression under constraints of failure probabilities*, Presented at the **IEEE Workshop on Emergent Technologies and Virtual Systems for Instrumentation and Measurement**, Niagara Falls, Ontario, May 15-16, 1997 (**Workshop Proceedings**, pp. 56-58).

C15. S. R. Das, M. Assaf and A. Nayak : *On the design of space compressor for VLSI circuits in BIST using nonexhaustive test sets*, Presented at the **Second World Conference on Integrated Design and Process Technology**, Austin, TX , U.S.A., December 1-4, 1996 (**Conference Proceedings**, Vol. 1, pp. 129-136).

C16. S. R. Das, M. Assaf and A. R. Nayak : *Selecting outputs for merger in space compression using concepts of Hamming distance and sequence weights*, Presented at the **IASTED International Conference on Modeling, Simulation and Optimization**, Gold Coast, Australia, May 6-9, 1996 (**Conference Proceedings**, pp. 6.1-6.8).

D. PAPERS SUBMITTED AND ON REVISION FOR PUBLICATION IN CONFERENCES, SYMPOSIA, WORKSHOPS, AND SEMINARS PROCEEDINGS

D1. S. R. Das, M. H. Assaf, E. M. Petriu and W. B. Jone : *Revisiting response compaction in space for full scan circuits with nonexhaustive test sets using concept of sequence characterization*, **20th IEEE Instrumentation and Measurement Technology Conference**, Vail, CO, U.S.A., May 20-22, 2003 (submitted).

D3. M. H. Assaf, R. Abielmona, P. Abolghasem, S. R. Das, E. M. Petriu and V. Groza: *JBits implementation and design verification in space compressor design of digital circuits*, IASTED International Conference on Modeling, Identification and Control, Innsbruck, Austria, February 10-13, 2003 (submitted).

D3. M. H. Assaf, S. R. Das, E. M. Petriu and S. Mukherjee : *Design of aliasing free space compressor in BIST with maximal compaction ratio using concepts of strong and weak compatibilities of response data outputs and generalized sequence mergeability*, **International Workshop on Distributed Computing**, Calcutta, India, December 28-31, 2002 (accepted).

APPENDIX A

LIST OF MATHEMATICAL SYMBOLS

$\{\Psi_1, \Psi_2\}$	Output Sequence Pair
$\{\Psi'_i, \Psi'_j\}$	Derived Sequence Pair
H_s:	Hamming Distance
W_1:	The 1-weight
W_0:	The 0-weight
L_s:	Output Sequences Length
E_E:	Maximum Expectation of Error
$E_s(g)$:	Number of Single Line Errors Detected at the Output of a Gate g
$E_d(g)$:	Number of Double Line Errors Detected at the Output of a Gate g
$E_{max}(g)$:	Maximum Detectable Error for Two-input Logic Functions
ε:	Second-order Detectable Error Probability Estimate
S_1:	Probability of Single Error Effect Felt at the Output of the CUT
S_2:	Probability of Double Error Effect Felt at the Output of the CUT
R_1:	Number of Single Line Errors at the Output of Gate *i*, if Gate *i* is Used
R_2:	Number of Double Line Errors at the Output of Gate *i*, if Gate *i* is Used
ε(A/N):	Detectable Error Probability Estimate When an AND/NAND gate is Used
ε(O/N):	Detectable Error Probability Estimate When an OR/NOR Gate is Used
ε(X/X):	Detectable Error Probability Estimate When an XOR/XNOR Gate is Used
W'_{N1}:	Nth order 1-weight
W'_{N0}:	Nth order 0-weight
R:	Residue or Hamming Distance of Multiple Sequences
$(R_i)_{nj}$:	Number of Errors Corresponding to the ith Order Residue Position of $_{nj}$ 0's
E_N:	Nth Order Detectable Error Probability Estimate
E_N(A/N):	Nth order detectable error probability estimate When an AND/NAND Gate is Used
E_N(O/N):	Nth order detectable error probability estimate When an OR/NOR Gate is Used

$E_N(X/X)$:	Nth order detectable error probability estimate When an XOR/XNOR Gate is Used
A_i:	Number of Columns in the Bundle Set with Vertical 1-weight Equal to i
${}^{N}C_i$:	Binomial Coefficient
∂_2:	Second-order Missed Error Probability Estimate
δ_N :	Nth-order Missed Error Probability Estimate
δ_N (A/N):	Nth-order Missed Error Probability Estimate When an N-input AND/NAND gate is Used
δ_N (O/N):	Nth-order Missed Error Probability Estimate When an N-input OR/NOR gate is Used
δ_N (X/X):	Nth-order Missed Error Probability Estimate When an N-input XOR/XNOR gate is Used
t_i	Probability of Missing i Line Errors
FF:	Fault-free Outputs
F:	Faulty Outputs
θ:	Faults Detected at the MUT Outputs
β:	Total Number of Detectable Faults at the MUT Outputs
τ:	Compact Set of Deterministic Input Test Vectors
(AB) s-AND(NAND) compatible:	Lines A and B are Strongly AND(NAND) Compatible
(AB) w-AND(NAND) compatible:	Lines A and B are weakly AND(NAND) Compatible
(AB) AND(NAND) incompatible:	Lines A and B are AND(NAND) Incompatible
(AB) AND(NAND) compatible:	Lines A and B are AND(NAND) Compatible
(AB) s-OR(NOR) compatible:	Lines A and B are Strongly OR(NOR) Compatible
(AB) w-OR(NOR) compatible:	Lines A and B are weakly OR(NOR) Compatible
(AB) OR(NOR) incompatible:	Lines A and B are OR(NOR) Incompatible
(AB) OR(NOR) compatible:	Lines A and B are OR(NOR) Compatible
(AB) s-XOR(XNOR) compatible:	Lines A and B are Strongly XOR(XNOR) Compatible
(AB) w-XOR(XNOR) compatible:	Lines A and B are weakly XOR(XNOR) Compatible
(AB) XOR(XNOR) incompatible:	Lines A and B are XOR(XNOR) Incompatible
(AB) XOR(XNOR) compatible:	Lines A and B are XOR(XNOR) Compatible
MCs(AND/NAND):	Maximal Compatibles for Logic AND/NAND
MCs(OR/NOR) :	Maximal Compatibles for Logic OR/NOR
MCs(XOR/XNOR):	Maximal Compatibles for Logic XOR/XNOR

APPENDIX B

MAXIMAL COMPATIBITY CLASSES FOR THE C880 BENCHMARK CIRCUIT

A. C880 PRIMARY OUTPUTS

Output lines
Ω_0 = 388
Ω_1 = 389
Ω_2 = 390
Ω_3 = 391
Ω_4 = 418
Ω_5 = 419
Ω_6 = 420
Ω_7 = 421
Ω_8 = 422
Ω_9 = 423
Ω_{10} = 446
Ω_{11} = 447
Ω_{12} = 448
Ω_{13} = 449
Ω_{14} = 450
Ω_{15} = 767
Ω_{16} = 768
Ω_{17} = 850
Ω_{18} = 863
Ω_{19} = 864
Ω_{20} = 865
Ω_{21} = 866
Ω_{22} = 874
Ω_{23} = 878
Ω_{24} = 879
Ω_{25} = 880

B. AND/NAND INCOMPATIBLE PAIRS USING DETERMINISTIC TESTING

AND incompatible pairs (Deterministic testing)	
Ω_0	Ω_6
Ω_1	Ω_6
Ω_1	Ω_7
Ω_4	Ω_7
Ω_4	Ω_{13}
Ω_{12}	Ω_{13}
Ω_{21}	Ω_{23}

C. MAXIMAL COMPATIBILTY CLASSES FOR AND/NAND AT THE FIRST STAGE

AND MCs list at the first stage	
C_1	(Ω_2, Ω_3, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_2	(Ω_2, Ω_3, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_3	(Ω_2, Ω_3, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_4	(Ω_2, Ω_3, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_5	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_6	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_7	(Ω_0, Ω_2, Ω_3, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_8	(Ω_0, Ω_2, Ω_3, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_9	(Ω_0, Ω_2, Ω_3, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_{10}	(Ω_0, Ω_2, Ω_3, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_{11}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_{12}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_{13}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_{14}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_{15}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_{16}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})
C_{17}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})
C_{18}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{24}, Ω_{25})

D. OR/NOR INCOMPATIBLE PAIRS USING DETERMINISTIC TESTING

OR incompatible pairs (Deterministic testing)	
Ω_0	Ω_6
Ω_1	Ω_6
Ω_1	Ω_7
Ω_6	Ω_7
Ω_{17}	Ω_{19}
Ω_{17}	Ω_{20}
Ω_{19}	Ω_{20}
Ω_{21}	Ω_{23}
Ω_{22}	Ω_{24}
Ω_{22}	Ω_{25}
Ω_{24}	Ω_{25}

E. MAXIMAL COMPATIBILTY CLASSES FOR OR/NOR AT THE FIRST STAGE

OR MCs list at the first stage	
C_1	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{25})
C_2	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{24})
C_3	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{22}, Ω_{23})
C_4	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{25})
C_5	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{24})
C_6	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{22})
C_7	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{25})
C_8	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{24})
C_9	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{22}, Ω_{23})
C_{10}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{25})
C_{11}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{24})
C_{12}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{22})
C_{13}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{23}, Ω_{25})
C_{14}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{23}, Ω_{24})
C_{15}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{22}, Ω_{23})
C_{16}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{25})
C_{17}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{24})
C_{18}	(Ω_0, Ω_2, Ω_3, Ω_4, Ω_5, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{22})
C_{19}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{25})
C_{20}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{24})
C_{21}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{22}, Ω_{23})
C_{22}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{25})
C_{23}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{24})
C_{24}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{22})
C_{25}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{25})

OR MCs list at the first stage	
C_{26}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{24})
C_{27}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{22}, Ω_{23})
C_{28}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{25})
C_{29}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{24})
C_{30}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{22})
C_{31}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{23}, Ω_{25})
C_{32}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{23}, Ω_{24})
C_{33}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{22}, Ω_{23})
C_{34}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{25})
C_{35}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{24})
C_{36}	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{22})
C_{37}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{25})
C_{38}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{23}, Ω_{24})
C_{39}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{22}, Ω_{23})
C_{40}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{25})
C_{41}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{24})
C_{42}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{20}, Ω_{21}, Ω_{22})
C_{43}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{25})
C_{44}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{23}, Ω_{24})
C_{45}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{22}, Ω_{23})
C_{46}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{25})
C_{47}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{24})
C_{48}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{18}, Ω_{19}, Ω_{21}, Ω_{22})
C_{49}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{23}, Ω_{25})

OR MCs list at the first stage	
C_{50}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{23}, Ω_{24})
C_{51}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{22}, Ω_{23})
C_{52}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{25})
C_{53}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{24})
C_{54}	(Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_8, Ω_9 Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{21}, Ω_{22})

F. XOR/XNOR INCOMPATIBLE PAIRS USING DETERMINISTIC TESTING

XOR incompatible pairs (Deterministic testing)	
NULL	NULL

G. MAXIMAL COMPATIBILTY CLASSES FOR XOR/XNOR AT THE FIRST STAGE

XOR MCs list at the first stage	
C_1	(Ω_0, Ω_1, Ω_2, Ω_3, Ω_4, Ω_5, Ω_6, Ω_7, Ω_8, Ω_9, Ω_{10}, Ω_{11}, Ω_{12}, Ω_{13}, Ω_{14}, Ω_{15}, Ω_{16}, Ω_{17}, Ω_{18}, Ω_{19}, Ω_{20}, Ω_{21}, Ω_{22}, Ω_{23}, Ω_{24}, Ω_{25})