Spectral Methods for Testing of Digital Circuits

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

Nitin Yogi

Certificate of Approval:

Victor P. Nelson
Professor
Electrical and Computer Engineering

Vishwani D. Agrawal, Chair
Professor
Electrical and Computer Engineering

Adit D. Singh
Professor
Electrical and Computer Engineering

Charles E. Stroud
Professor
Electrical and Computer Engineering

George T. Flowers
Dean
Graduate School

Spectral Methods for Testing of Digital Circuits

Nitin Yogi

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
August 10, 2009

Spectral Methods for Testing of Digital Circuits

Nitin Yogi

Permission is granted to Auburn University to make copies of this dissertation at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date of Graduation

Vita

Nitin Yogi, son of Vasant Yogi and Vandana Yogi was born on December 15, 1980 in Mumbai, India. He graduated from Rizvi College of Engineering, affiliated with the Mumbai University, with a Bachelor of Engineering degree in Electronics. He joined Auburn University in Fall 2002 and pursued the Masters program in the Electrical and Computer Engineering Department under the supervision of Prof. Victor P. Nelson. During his Masters program he worked as a Graduate Research Assistant in the Department of Forestry at Auburn University and was actively involved in the development of an interactive database and matrix population modeling/simulation software package "AvesModeler". He obtained his Master of Science degree in Fall 2004. He joined the doctoral program at Auburn University in Fall 2005 under the supervision of Prof. Vishwani D. Agrawal. During the period 2004-2006 he worked as a Graduate Research Assistant in the Department of Rehabilitation and Special Education at Auburn University and was active in the development of an on-line data collection and report generation system "PODS". As a Graduate Research Assistant in the Department of Electrical and Computer Engineering he received support in parts from the Wireless Engineering Research and Education Center (WEREC). During the summer of 2007, he held an internship at Intel Corporation, Folsom, CA, working in the area of verification of DDR memory protocols. In the summer of 2008, he interned with NXP Semiconductors, The Netherlands, and worked on static timing analysis and silicon timing data.

Dissertation Abstract

Spectral Methods for Testing of Digital Circuits

Nitin Yogi

Doctor of Philosophy, August 10, 2009
(M.S., Auburn University, 2004)
(B.E., Bombay University, 2002)

131 Typed Pages

Directed by Vishwani D. Agrawal

Due to increasing design complexities of digital circuits in recent years, a growing problem in Very Large Scale Integrated (VLSI) digital circuit testing is the exponential rise in the test generation complexity and an increasing need for high quality test vectors. For Built-In Self-Test (BIST) of digital circuit, the in-built pattern generator shows increased area overhead, as larger number and more specific patterns need to be generated. In this thesis we address these issues of digital circuit testing.

We propose a novel test generation algorithm for sequential circuits using spectral methods. We generate test vectors for faults defined at Register-Transfer Level (RTL) and analyze them for spectral properties. New test vectors are generated using these properties to detect all faults of the circuit. Our proposed algorithm shows equal or improved test coverage and reduced test generation time as compared to a commercial sequential test generation tool, FlexTest, for various benchmark circuits. For an experimental processor PARWAN, FlexTest achieved a test coverage of 93.40% requiring 1403 test vectors in 26430 CPU seconds. The proposed spectral method achieved a coverage of 98.23% requiring 2327 vectors in 2442 CPU seconds. We also propose a Design-For-Testability (DFT) method at RTL which enables improved test coverage and reduced test generation time.

We define N-model tests that target faults belonging to N specified fault models of choice. We propose a method for minimizing these tests using Integer Linear Programming (ILP) without reducing the individual fault model coverage. Stuck-at, transition, and pseudo stuck-at IDDQ faults are used as illustrations. The proposed method shows a noticeable reduction in test set size as compared to conventional minimization. For ISCAS'89 benchmark circuit s1488, the initial test set consisted of 557 test vectors (with 57 I_{DDQ} vectors) (represented as 557(57)). Conventional single fault model minimization achieved 451(45) test vectors while our multiple fault model minimization achieved 175(39) test vectors. We also propose an ILP model to offer a trade-off between the total number of test vectors and the cost of test application (number of I_{DDQ} vectors in our example). For s1488, depending on the cost of application, our method offers a choice anywhere from 175(39) to 187(33) test vectors. Since solving ILP problems has an exponential time complexity, we also propose a reduced complexity ILP approximation.

We propose a method for designing a Test Pattern Generator (TPG) for BIST using spectral techniques, which replicates the efficacy of a given set of test patterns generated for a digital circuit. Spectral properties extracted from the test patterns are regenerated in hardware using a novel spectral TPG architecture. For combinational circuits, a test vector reshuffling algorithm is proposed to enhance the extraction of spectral properties. In six out of eight sequential benchmark circuits considered, our method achieved at least as much fault coverage as the ATPG vectors. For the circuit s38417, our proposed method detected 17020 faults as compared to 15472 faults detected by ATPG vectors. Our proposed BIST method detects equal or greater number of faults in six out of eight circuits than random, weighted random and an earlier published work. In case of combinational circuits, for circuit c7552, our method attained a test coverage of 99.82%, while random and weighted random attained 97.41% and 97.86% respectively for the same test vector length. We also show the benefits of reseeding of our proposed spectral TPG in terms of test compression on two combinational benchmark circuits. In the considered circuits, our proposed architecture provides a maximum test data compression exceeding 90%.

Acknowledgments

I would like to start by thanking deeply my adviser Prof. Vishwani D. Agrawal without whom none of this would have been possible. He has been a constant source of support, encouragement, guidance and rational thinking throughout my doctoral program. It has been an immense pleasure to work under Prof. Agrawal for my Ph.D. degree.

I would like to thank Prof. Victor P. Nelson, Prof. Adit D. Singh, and Prof. Charles E. Stroud for their valuable suggestions regarding my research work from time to time, for the various things I have learned from them including those from their classes and for being on my committee. I would like to thank Prof. Paul M. Swamidass for agreeing to be an outside reader for my doctoral dissertation and providing valuable comments and suggestions. I appreciate the support from the Wireless Engineering Research and Education Center (WEREC) and the encouragement I received from its director, Prof. Prathima Agrawal. I would also like to thank the professors at Auburn Univerisity with whom I took classes and learned immensely from them. I would also like to thank Mr. Les Simonton for his continued technical support and help. I would like to thank Dr. Ananta Majhi, Dr. Bram Kruseman and Dr. Stefan Eichenberger at NXP Semiconductors for their continued support, encouragement and guidance during my internship

My journey in my doctoral program would not have been easy if it weren't for my friends and colleagues at Auburn University. I would like to thank my friends in my research group, Kalyana, Sachin, Sudheer, Ashfaq, Gefu, Jins, Khushboo, Fan, Wei, Kim and Manish for their suggestions, comments, support and camaraderie. I would also like to acknowledge and thank my friends with whom I spent my good times. Finally but importantly I would like to thank my family for their continued support.

Style manual or journal used Journal of Approximation Theory (together with the style known as “aums”). Bibliograpy follows van Leunen’s *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures xi

List of Tables xiii

1 Introduction 1
- 1.1 Problem definition . . . 2
- 1.2 Contribution of this thesis . . . 2
- 1.3 Organization of the thesis . . . 4

2 Overview of Manufacturing Test 5
- 2.1 Testing of integrated circuits . . . 5
- 2.2 Fault modeling . . . 6
  - 2.2.1 Lower-level fault models . . . 6
  - 2.2.2 Higher-level fault models . . . 9
- 2.3 Test generation and Design For Test (DFT) . . . 11
- 2.4 Fault simulation . . . 13
- 2.5 Built-In Self Test (BIST) . . . 14

3 Spectral Analysis 15
- 3.1 Hadamard transform and Walsh functions . . . 16
- 3.2 Spectral analysis using Hadamard transform . . . 18
- 3.3 Information content and randomness . . . 21
- 3.4 Spectral analysis for test generation . . . 22

4 Test Vector Minimization 23
- 4.1 Linear programming for test vector minimization . . . 24
- 4.2 ILP formulation for test minimization . . . 25

5 Built-In Self Test 27
- 5.1 Prior work . . . 28
- 5.2 Test pattern generator . . . 29
  - 5.2.1 Linear Feedback Shift Register (LFSR) . . . 31
  - 5.2.2 Cellular Automata Register (CAR) . . . 33
- 5.3 Output response analyzer . . . 36

6 Spectral RTL Test Generation 38
6.1 Spectral RTL ATPG . . . 40
6.1.1 RTL spectral characterization . . . 40
6.1.2 Gate-level test generation . . . 43
6.2 Design-for-testability . . . 47
6.3 Implementation and results . . . 48
6.3.1 Results for ITC'99 and ISCAS'89 benchmark circuits . . . 48
6.3.2 Results for PARWAN processor . . . 53
6.4 Summary . . . 61

7 N-Model tests 62
7.1 Overview . . . 64
7.2 The $\underline{\mathbf{N}}$-model tests . . . 65
7.3 Two-step ILP model . . . 66
7.3.1 First ILP - minimize vectors . . . 66
7.3.2 Second ILP - minimize I_{DDQ} measurements . . . 67
7.4 Combined ILP model . . . 67
7.5 Hybrid LP-ILP method . . . 69
7.6 Results . . . 70
7.7 Summary . . . 75

8 Spectral Test Pattern Generation Hardware for BIST 77
8.1 Proposed spectral BIST method . . . 78
8.1.1 Determination of spectral components and noise . . . 79
8.1.2 Spectral BIST implementation . . . 85
8.2 Reseeding of proposed test pattern generator . . . 91
8.3 Results . . . 93
8.3.1 Results for BIST mode without reseeding . . . 93
8.3.2 Results for BIST mode using reseeding . . . 99
8.4 Summary . . . 102

9 Conclusion and Future Work 104
9.1 Conclusion . . . 104
9.2 Future Work . . . 105
9.2.1 Test data compression . . . 105
9.2.2 Spectral BIST for scan-inserted sequential circuits . . . 106

Bibliography 109

List of Figures

2.1 Behavior of stuck-at logic '0' fault at the output of a AND gate. 7

2.2 Behavior of a slow-to-fall transition delay fault at the output of a AND gate. 8

2.3 A short defect in a NAND gate causing abnormal I_{DDQ} current between V_{DD} and GND. 10

3.1 Walsh functions of order eight. 18

3.2 Graphical representation of Walsh Coefficients obtained in equation (3.7). . 21

5.1 General BIST architecture. 28

5.2 Generic N-bit internal feedback shift register. 32

5.3 Generic N-bit external feedback shift register. 32

5.4 Generic rule function for a flip-flop. 34

5.5 Rule 90 implementation of a CAR. 36

5.6 Rule 150 implementation of a CAR. 36

6.1 Spectral analysis of a test vector block. 41

6.2 Spectral coefficients for an arbitrary random signal. 44

6.3 Walsh spectral coefficients for input DataIn[5] signal of PARWAN processor. 44

6.4 Bit-steam generation by perturbing the spectra. 45

6.5 RTL-based DFT to improve observability of signals. 48

6.6 Test coverage of RTL ATPG (spectral vectors) for area optimized b11-A circuit. 52

6.7 PARWAN CPU [97]. 53

6.8 Test coverages for the original PARWAN circuit [97]. 56

6.9 Test coverages for the PARWAN circuit with DFT. 56

6.10 Test coverages for the original PARWAN circuit [97]. 60

6.11 Test coverages for the PARWAN circuit with DFT. 60

7.1 Number of passing/failing chips for four different test types applied [101]. . 62

7.2 Number of passing/failing chips for three different test types applied [87]. . 63

7.3 Defect level in parts per million deduced from data in [87]. 63

8.1 Appending of extra vectors to balance the weighting of bit-streams to 0.5. . 79

8.2 Spectral analysis of test vectors. 82

8.3 Determination of prominent spectral components. 84

8.4 Proposed spectral BIST architecture. 84

8.5 Walsh function generator of order 4 that generates 16 Walsh functions [156]. 86

8.6 Spectral component synthesizer that combines three spectral components. . 87

8.7 Randomizer XOR gate that randomly flips 25% of bits. 88

8.8 Holder circuit implemented using a multiplexer and clock derived signals. . 90

8.9 Reseeding of proposed spectral BIST TPG. 92

9.1 Scan-inserted sequential circuit. 107

List of Tables

5.1 Example rule functions for CAR. 35

6.1 Circuit description. 49

6.2 Spectral characterization of circuits by RTL vectors. 50

6.3 Comparison of RTL ATPG and Sequential gate-level ATPG results. 51

6.4 Spectral characterization of processor circuit by RTL vectors for stuck-at faults. 55

6.5 Spectral RTL ATPG for stuck-at faults for processor circuits. 55

6.6 Spectral characterization of processor circuit by RTL vectors for transition delay faults. 58

6.7 Spectral RTL ATPG for transition delay faults for processor circuits. 59

6.8 Stuck-at fault coverage of transition fault vectors. 59

7.1 Test vectors for stuck-at, I_{DDQ} and transition faults generated and minimized by FastScan. 70

7.2 Multiple fault model test optimization by ILP methods using two-step model. 72

7.3 Multiple fault model test optimization by ILP methods using combined model. 72

7.4 Multiple fault model test optimization by hybrid LP-ILP method using two-step model. 73

7.5 Multiple fault model test optimization by hybrid LP-ILP method using combined model. 74

7.6 Comparing solutions: hybrid LP-ILP lower bound, ILP optimum and hybrid LP-ILP. 74

8.1 Details of combinational circuits on which our proposed method was employed. 94

8.2 Details of implemented spectral BIST TPG. 94

8.3 Test coverage comparison of random, weighted random and proposed spectral BIST method for 64000 vectors. 95

8.4 Area overhead comparison of proposed spectral BIST and Pseudo-Random Pattern Generator (PRPG). 95

8.5 FlexTest ATPG results. 96

8.6 Experimental results on fault detection by BIST patterns. 97

8.7 Comparison of fault coverage and number of vectors with FlexTest ATPG. . 98

8.8 BIST area overhead in transistors. 98

8.9 Comparison of test data volume and test time for ATPG and different modes of operation of spectral BIST for c7552. 100

8.10 Comparison of test data volume and test time for ATPG and different modes of operation of spectral BIST for s15850 (combinational). 101

9.1 Comparison of fault coverage and number of vectors with FlexTest ATPG. . 106

Chapter 1

Introduction

Recent advances in microelectronic fabrication of Complimentary Metal Oxide Semiconductor (CMOS) technology have enabled a substantial increase in the level of integration of transistors per unit area and facilitated the reduction in chip cost. However, with these improvements, the design complexities of circuits have proportionally increased creating challenges in several areas including manufacturing test [20, 78]. Manufacturing test ensures that a digital circuit fabricated in silicon functions as expected and according to the original design [14]. One main goal of testing is to identify all chips that do not function as expected due to defects.

Manufacturing test faces several challenges due to increased design complexity. The problem of test generation for digital circuits is computationally intensive and has been theoretically and experimentally shown to be Non-Polynomially complete (NP-complete) [37, 43, 62]. Determining solutions to such problems in worst cases may require non-polynomial or exponential time with respect to the size of the problem. The test generation problem becomes more intricate for sequential circuits, as their internal memory states face the difficulty of not being easily controllable and observable. Hence, there is a need to reduce the test generation complexity.

Furthermore, the generated tests should have high quality or should cover a large proportion of modeled faults. A high fault coverage is required for the tests so that the number of test escapes, or the number of bad chips that are incorrectly considered good, is kept as small as possible, ideally zero, which is one of the main goals of manufacturing test. The number of bad chips tested as good is normally expressed as the *defect level* [14]. It is measured in parts per million (ppm). While a zero defect level is hard to guarantee, improved quality tests can provide 500 ppm, 100 ppm or even lower defect levels. Also,

there is a need to reduce the test application time which affects the testing cost. Hence, the number of test vectors required to be applied to the circuit under test (CUT) to test it, also need to be kept to a minimum. For Built-In Self-Test (BIST) environments, where additional inserted hardware tests the CUT, similar challenges are faced. Moreover a test generator is required to be designed in hardware with minimum area overhead which can provide reliable high quality tests.

1.1 Problem definition

The primary goals of this work have been:

- To develop an efficient test generation method for sequential circuits having reduced test generation complexity, high fault coverage and low test application time.
- To develop a minimization technique for tests which detect multiple fault models.
- To develop a BIST synthesis scheme for digital circuits which provides high fault coverage, has low area overhead and low test application time.

1.2 Contribution of this thesis

In this dissertation, we propose a novel test generation algorithm for sequential circuits using spectral methods and Register-Transfer-Level (RTL) information. We utilize RTL-related information to retrieve important spectral properties which help in efficiently testing the Circuit-Under-Test (CUT). Use of RTL information simplifies and reduces the test generation complexity in terms of the problem size. The use of spectral information for test generation has been shown to provide advantages in terms of improved fault coverage of the generated test vectors [18, 19, 41, 72, 158]. Using the benefits of RTL and spectral information, our proposed test generation scheme shows improved fault coverage and reduced test generation time as compared to the commercial sequential test generation tool FlexTest [91] as demonstrated in the results for various benchmark circuits. We also

propose a Design-for-Testability (DFT) method at the RTL to alleviate some of the bottlenecks in the testability of the circuit which provides an enhancement in the fault coverage with a benefit in a slight reduction of test generation time and number of test vectors.

We define N-model tests that target detection of faults belonging to N specified fault models of choice. We propose a method for minimizing these tests using Integer Linear Programming (ILP) without reducing the individual fault model coverage. Any test sequences, deterministic, random, functional, N-detect, etc., can be minimized for the given set of fault models. Stuck-at, transition, and pseudo stuck-at I_{DDQ} faults are used as illustrations. The proposed method shows a noticeable reduction in test set size as compared to conventional single fault model minimization. We also propose a novel configurable minimization model which can provide the trade-off between the number of test vectors and the cost of application of various types of tests. Although solving ILP formulations provide optimal tests, their worst-case complexity is exponential. Hence we also propose a reduced complexity ILP formulation which provides approximate solutions with reduced computational times.

We also propose a method for constructing a pattern generator for digital circuits in a Built-In Self-Test (BIST) environment using spectral properties. Given a set of test patterns generated for a digital circuit, the objective here is to regenerate the efficacy of those vectors in hardware for BIST using minimal area overhead and test vector length. We exhibit the implementation of our methodology for combinational and sequential benchmark circuits. For combinational circuits, a test vector reshuffling algorithm is proposed to enhance the spectral properties and facilitate their extraction. We compare our hardware implementation with an earlier published work for sequential circuits and also with pseudo-random and weighted random pattern generators for both combinational and sequential circuits. The proposed BIST pattern generator, while attaining the test coverage of the original test vectors, shows markedly improved test coverage for similar vector length and comparable area overhead as compared to other pattern generators.

1.3 Organization of the thesis

The dissertation is organized as follows. In Chapter 2 we provide a brief overview of the area of manufacturing testing and reinforce the motivation of our work. Chapter 3 gives an introduction to spectral analysis which forms the foundation of our proposed methods. In Chapter 4, we discuss the concepts of test vector minimization using Integer Linear Programming (ILP), which is used later in our proposed spectral RTL test generation scheme. Chapter 5 gives an introduction to the theory of Built-In Self Test (BIST) and describes its main components. In Chapter 6, we propose our spectral RTL test generation scheme for sequential circuits and describe its results. A new type of test called as "N-model test" is introduced in Chapter 7 and we propose an ILP-based formulation to minimize the number of tests. In Chapter 8, we propose our method for constructing the spectral test pattern generator for BIST environments and discuss its results. We give the conclusions of this work and scope for future advancements in Chapter 9.

Chapter 2

Overview of Manufacturing Test

In this chapter, we give a brief overview of the area of manufacturing test and describe its main concepts [14]. The fundamentals described in this chapter will be used in the following chapters to explain the new methods.

2.1 Testing of integrated circuits

After a digital circuit has been designed, it is fabricated in the form of silicon chips. The fabrication process is not perfect and due to various reasons, the manufactured circuit in silicon may develop defects which may prevent its correct functioning [82]. A manufacturing test performs the crucial task of identifying those silicon chips that do not function as expected. It involves exercising the functionality of the Circuit Under Test (CUT) by applying appropriate test signals to its inputs and observing the responses. If the responses of the CUT match the expected responses, then the CUT is considered good else it is labeled as bad. Thus, the goal of testing is to correctly identify a good chip as good and a bad chip as bad. The testing process may not be perfect and it may label certain good chips as bad and vice versa. The proportion of good chips that are incorrectly labeled as bad by the testing process is termed as "yield loss", while the portion of bad chips incorrectly labeled as good is referred to as "test escapes". Test escapes are quantified as the *defect level*, measured as the average number of bad chips that are tested good (usually measured for per million chips tested). The yield loss results in economic loss due to throwing away a proportion of good chips. Test escapes, on the other hand, result in defective parts shipped to customers and, depending on the application, have moderate to serious consequences in

terms of system failures, economic damages, etc. The testing process thus needs to ensure that both of these proportions are kept to a minimum [14].

2.2 Fault modeling

As mentioned earlier, when digital circuits are fabricated in the form of silicon chips, due to various fabrication process aberrations, some of the chips develop defects which may prevent their correct functioning. It is the goal of manufacturing testing to determine whether a chip possesses any such fault-causing defects, in a given finite time allotted for testing. Faults at the physical level in chips cannot be tested and detected directly, as there could be numerous types that can occur and many of them are often complex in nature to analyze. Hence faults need to be modeled at a higher abstraction level in order that they can be analyzed and test signals generated to detect them [14, 82]. These models are generally referred to as fault models.

Faults can be modeled at various abstraction levels starting with the lowest level like the transistor and gate level; and moving to higher levels like Register-Transfer-Level (RTL) and behavioral level. Based on these abstraction levels, the fault models can be roughly classified as Lower-level fault models and Higher-level fault models. We describe these types in further detail in the following sections.

2.2.1 Lower-level fault models

The lower-level fault models include those defined at the transistor and the gate levels. At this abstraction level, the digital design is described as an interconnection of transistors and gates, and faults can be modeled as imperfections in their respective components. Some of the commonly used and popular fault models at the transistor and gate-level are stuck-at fault model, transition delay fault model and I_{DDQ} fault model [15]. We shall be using these three fault models later in this thesis to evaluate our proposed methods and hence we shall describe them in a little more detail in the following subsections. Other types of

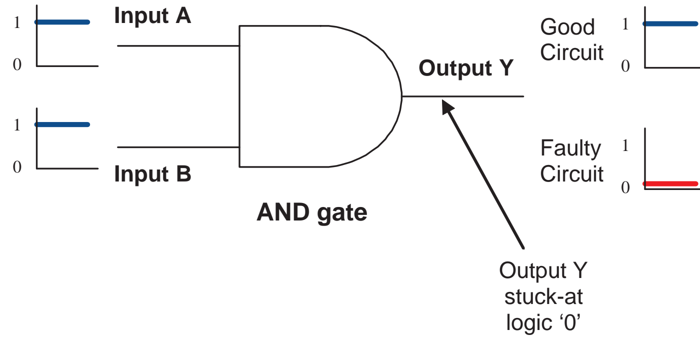


Figure 2.1: Behavior of stuck-at logic ’0’ fault at the output of a AND gate.

faults that have been defined at lower-levels of abstraction are bridging faults [116], wire stuck-open faults, parametric faults, etc [14].

Stuck-at fault model

One of the most widely used fault models for gate-level digital circuits since the earlier developments of CMOS technology has been the stuck-at fault model. The faults are modeled on signal lines or interconnects between the gates. Using the stuck-at fault model, two types of faults can be modeled for any signal line in the gate-level digital circuit. The logic value of a considered faulty signal line could be permanently stuck-at logic ’0’ or stuck-at logic ’1’. Figure 2.1 shows the behavior of a stuck-at logic ’0’ on the output of the AND gate. The two inputs ’A’ and ’B’ of the AND gate are been driven with logic ’1’. The expected good circuit behavior of the output ’Y’ of the gate is logic ’1’. However, due to the presence of the stuck-at logic ’0’ fault, in the faulty circuit the output ’Y’ will have a logic ’0’. Since two types of stuck-at faults are defined for every signal line, for a gate-level circuit with n signal lines, there exist $2n$ stuck-at faults assuming only one fault can exist at a time. For the case, where multiple faults can exist, the number of faults is equal to $3^n - 1$. Stuck-at faults model some of the physical defects that could arise in silicon manufacturing like transistors permanently in ON or OFF state, shorting of signal lines to power supply lines (V_{DD}: logic ’1’ or GND: logic ’0’), etc.

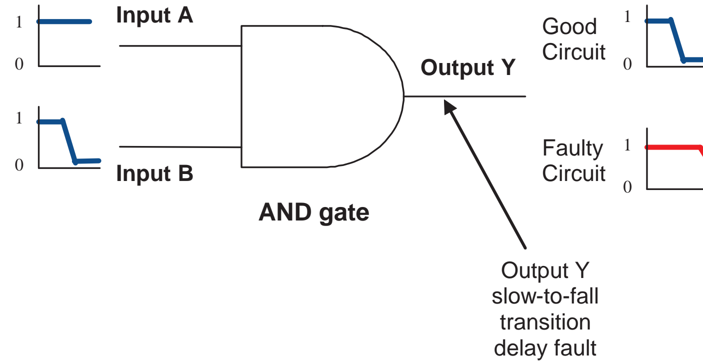


Figure 2.2: Behavior of a slow-to-fall transition delay fault at the output of a AND gate.

Transition delay fault model

With the advances in manufacturing technology and fabrication of designs which can run at increasingly faster clock frequencies, estimation and testing of timing of a circuit has gained importance. In order to model the delay defects in a digital circuit, the transition delay fault model was introduced [76]. Like the stuck-at fault model, the transition delay fault model models faults on signal lines or interconnects between the gates. As per the transition delay fault model, a faulty signal line can behave as a slow-to-rise signal or a slow-to-fall signal. For a slow-to-rise transition delay fault on a faulty signal line, the signal line behaves as a temporary stuck-at logic '0' for a time period which exceeds the maximum delay of the circuit or is generally taken to be one test cycle or clock period. A similar behavior is exhibited by a slow-to-fall transition delay fault. To detect a transition delay fault on a signal line, a two-vector pair is required to be applied to the inputs of the CUT. The first vector initializes the signal line under consideration to the required logic value. the second vector forces a transition on the signal line and propagates its effect to the primary outputs of the CUT. Figure 2.2 shows the behavior of a slow-to-fall transition delay fault on the output of a AND gate. The input 'A' of the gate is driven with a logic '1' and the input 'B' of the gate undergoes a high-to-low falling transition. Due to the falling transition on input 'B', the expected good circuit behavior of the output 'Y' is a high-to-low falling

transition. However, due to the slow-to-fall transition delay fault on the faulty output 'Y' of the AND gate, the output 'Y' behaves as a temporary stuck-at logic '1' or the falling transition of the signal line 'Y' is delayed. Like the stuck-at fault model, for a gate-level circuit with n signal lines, there exist $2n$ transition delay faults assuming only one fault can exist at a time. For the case, where multiple faults can exist, the number of faults is equal to $3^n - 1$. Transition delay faults model some of the physical defects that could arise in silicon manufacturing like gross delay defects in slow transistors, resistive shorting of signals to power lines, some cases of transistors permanently in OFF state, etc.

I_{DDQ} faults

In a CMOS gate, when the inputs of the gate are stable and not switching, then the current flowing between V_{DD} and GND is negligible, ideally equal to zero. This steady state or quiescent current is termed as I_{DDQ} current. However, in the presence of certain defects, it is observed that this current can increase by an order of magnitude as compared to the defect-free case. This observation enables detection of certain defects by measuring this current. Figure 2.3 shows an example of a short defect in a transistor in a NAND gate which causes abnormal I_{DDQ} current to flow between V_{DD} and GND. The inputs 'A' and 'B' of the gate are driven with logic '1'. In a good circuit without defects, the top two PMOS transistors will be in the OFF state and the bottom two NMOS transistors will be in the ON state. In the presence of a short defect in a PMOS transistor, it will behave as an ON transistor and cause aberrant I_{DDQ} current. By measuring the magnitude of this current the short defect of the PMOS transistor can be detected. The I_{DDQ} faults [15] model certain physical defects occurring in fabrication like shorts between signal lines, transistors permanently in ON state, etc.

2.2.2 Higher-level fault models

At higher levels of abstraction, faults can be modeled at the Register-Transfer Level (RTL) [39, 114, 132] or the behavioral level [22, 40, 102, 136]. At the RTL, the digital

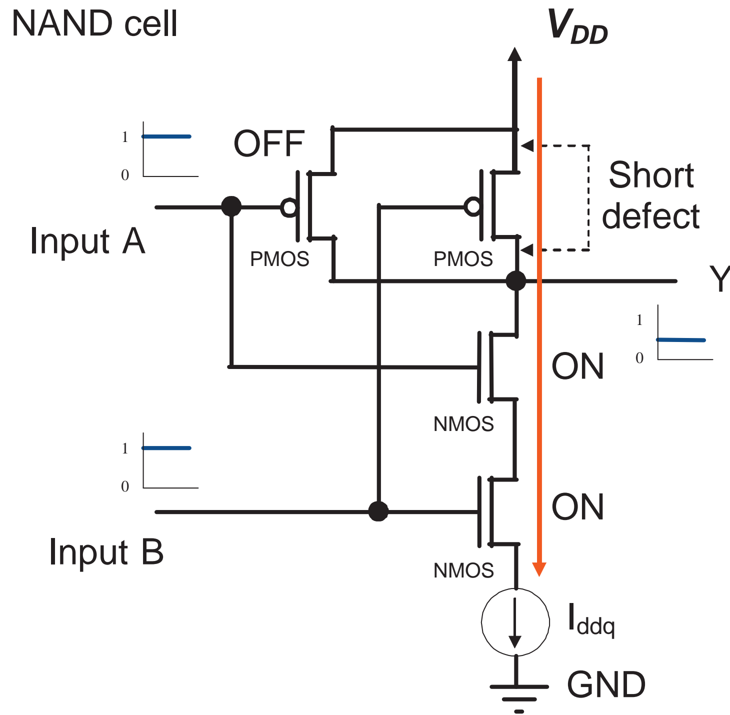


Figure 2.3: A short defect in a NAND gate causing abnormal I_{DDQ} current between V_{DD} and GND.

design is modeled as data transfers between registers and faults can be modeled in the registers and/or in the data transfers between the registers. At the behavioral level, the digital design is described in the form of an algorithm or functional description and faults can be modeled in the various operations that are defined and used in the description. One of the fault models that is defined at higher abstraction levels like the RTL and behavioral level is the RTL fault model [132].

Fault models at lower abstraction levels have higher correlation with the physical defects and hence are able to be characterized better as compared to fault models at higher levels of abstraction. However, fault models at higher abstraction levels are less complex and easier to analyze and utilize for test generation and test evaluation than those at lower abstraction levels. Hence, depending on the scenario, an appropriate fault model can be used.

2.3 Test generation and Design For Test (DFT)

Test generation is the most important step in manufacturing testing in which, given a set of faults defined using a fault model, appropriate test signals, called test vectors, are generated, which when applied to the CUT are able to detect the presence of those faults. The program which generates these test vectors is called an Automatic Test Pattern Generator (ATPG). The problem of test generation for digital circuits is computationally intensive and has been shown theoretically to be Non-Polynomial(NP)-complete [43]. Determining solutions to NP-complete problems require non-polynomial or exponential time with respect to the size of the problem. However, verifying a given prospective solution to an NP-complete problem requires only polynomial time.

Based on these characteristics, the test generation methods can be roughly classified as algorithmic methods and simulation-based methods. Algorithmic methods involve a series of well defined steps to be followed to obtain test vectors which detect a given set of faults. Since algorithmic methods take the approach of solving the test generation problem, they require non-polynomial or exponential time with respect to the number of signals in the circuit and the number of faults. Several algorithmic methods for test generation have been proposed in literature [21, 38, 49, 84] but require large computational times providing only limited fault coverage. On the other hand, simulation based methods rely on searching and simulating various test vectors based on some heuristics, which could be prospective tests to detect the given set of faults. Since simulation-based methods take the approach of verifying a prospective solution to the test generation problem, their time complexity can be much lower compared to algorithmic methods depending on the heuristics being used. Based on this concept several simulation-based methods have been proposed and developed over the years [2, 12, 80, 119, 120, 126].

The test generation problem can be differentiated into two types based on whether the Circuit Under Test (CUT) is memory-less (combinational) or possesses memory (sequential). In a memory-less or combinational circuit, all the inputs are controllable. Hence with respect

to test generation, the circuit can be easily subjected to any possible required input values. Sequential circuits on the other hand have memory states and testing of such a circuit not only depends on the values at the inputs, but also the values of the memory states. Most of the time, the memory states cannot be controlled and observed with ease, which makes the problem of test generation for sequential circuits more complex.

As a means to increase the testability of the circuits and also to reduce the Automatic Test Pattern Generation (ATPG) complexity, Design-For-Test (DFT) methods are employed. Two main parameters that determine the testability of a circuit are the controllability and observability of its signals. Controllability of a signal refers to its ability or ease to be set to a particular logic value from the primary inputs of the circuit. Observability of a signal refers to its ability or ease to be observed at one of the primary outputs of the circuit. Design-for-test (DFT) method refers to the design method of improving the controllability and observability of the signals of the given digital circuit so that the overall testability of the circuit is enhanced and tests with high fault coverage can be derived in reduced time complexity.

Several DFT schemes are employed in practice. The most popular DFT technique, widely used, is the scan chain, in which a serial shift register is formed by connecting together all the flip-flops in the sequential circuit. Any flip-flop can then be initialized to any required value by shifting in the appropriate bits. Also the value captured in any flip-flop can be observed by shifting out the bits in the scan chain. Scan-based DFT simplifies the test generation of sequential circuits to combinational test generation. However, there are downsides to using a scan chain. There is an area overhead and performance penalty associated with it, which may not be acceptable for all designs. Also, there are some issues with the generation and application of at-speed scan tests, which detect delay faults. Launch-On-Shift (LOS) and Launch-On-Capture (LOC) are two methods used for at-speed scan testing. Each method has its pros and cons. LOS has good transition delay fault coverage, but requires additional hardware for a fast scan enable signal. LOC requires

no special scan hardware but it achieves lower coverage. Since scan test vectors are non-functional tests, the problem of false paths and multi-cycle paths needs to be considered in the generation of at-speed scan tests, as the tests can cause unacceptable yield loss by failing functionally good circuits. This requires analysis of paths using static and dynamic timing analysis tools. Since to apply a test vector, all of the flip-flops need to be scanned, the number of clock cycles for testing and hence the testing time can grow very rapidly. For example if 100 test vectors need to applied to a sequential circuit consisting of 100 flip-flops, approximately 10,000 clock cycles will be required.

Although non-scan test generation has the disadvantage of high test generation complexity, it possesses certain advantages which make it an attractive approach. The disadvantages exhibited by scan-chain based test generation, like area and delay overheads and long testing times, are eliminated in this case. With sequential test generation, at-speed functional tests can be generated as they do not modify the state machine of the circuit. Hence the chances of yield loss are minimized. Different works [86, 90, 99] have attempted to show the effectiveness of at-speed functional tests over structural scan tests in detecting chip faults and hence having a better defect coverage. Better defect coverage translates to lower test escapes. Thus in our work we shall concentrate on issues and propose methods for non-scan digital circuit test generation.

2.4 Fault simulation

Fault simulation is an important part of manufacturing testing, which determines the faults detected of a given fault model by a given set of test vectors on a CUT. In fault simulation, the test vectors are simulated on the CUT in the presence of one fault at a time and the response of the CUT to the test vectors is compared with the expected correct responses. If the simulated responses differ from the expected correct responses, then the fault being simulated is considered to be detected. The process is repeated for all the faults. Along with evaluating the effectiveness of the test vectors, fault simulation also forms an integral part of the ATPG program. For a digital circuit with n signal lines, the

complexity of fault simulation is $O(n^2)$. By comparing the time complexity for solving the test generation problem with that of fault simulation, we can deduce that fault simulation based test generation methods can provide lower time complexity, as was also suggested in Section 2.3. Hence our proposed method described in this thesis will take advantage of fault simulation in test generation.

2.5 Built-In Self Test (BIST)

Built-In Self Test (BIST) is a special case of Design-For-Test (DFT) methodology in which the circuit tests itself and flags whether it is good or bad [3, 4, 89, 94, 127]. Additional hardware is inserted to generate test vectors which drive the primary inputs of the circuit, sample its primary output(s) and determine whether the circuit is good or bad by comparing the sampled output(s) with expected one(s). The use of BIST has several advantages. The need for expensive Automatic Test Equipment (ATE) is eliminated. BIST supports at-speed testing. Testing can be performed during operation as well as maintenance. BIST provides vertical testing from component level to system level.

For BIST environments there are mainly two methods used for testing; scan-based testing and non-scan based testing. Scan-based BIST utilizes the DFT structure scan chain, which was described earlier in Section 2.3, to apply the test vectors and observe the responses. Non-scan based BIST makes use only of the inputs and outputs of the CUT to test it. Scan-based BIST and non-scan based BIST have similar advantages and disadvantages as scan-chain based test generation and non-scan circuit test generation, respectively, as described earlier in Section 2.3. For non-scan based BIST, especially for sequential circuits, there is an additional challenge to detect random pattern resistant faults by generating specific sequences of test vectors in hardware, which can be intricate. As we mentioned earlier, in our work we shall concentrate on issues of non-scan based digital circuit testing. BIST methodology will be described in more details in Chapter 5 as an introduction to our proposed work.

Chapter 3
Spectral Analysis

For several years, research has being conducted on the nature and characteristics of test vectors, which will serve as good quality tests. Initially, experiments were performed using random vectors and were found to give good results [80, 143]. Later a class of random pattern resistant circuits were discovered [29], which made it difficult to use random vectors. Research then shifted towards weighted probability-based random [13, 50, 119, 143, 142] and other types of property-based test generation methods [48]. Some of these methods did work, but not satisfactorily for all circuits.

The idea of analyzing the periodicities in signals for test generation introduced the field of spectral testing. The basic idea was to look at the periodicities of the test vectors which provided high fault coverage by analyzing their information content in the frequency or the spectral domain. Several published books and articles [9, 30, 61, 134] provide introduction, general properties and applications of spectral transforms for digital signals. It is believed that good quality test vectors, which give high fault coverage, exhibit certain discernible frequency or spectrum related characteristics. By preserving these characteristics good quality high defect coverage test vectors can be generated.

Spectral methods for test generation have a long history since the development of complex VLSI circuits. In 1983, Susskind [128] showed that Walsh spectrum can be used for testing a digital circuit. Logic networks were tested for stuck faults by verifying the Walsh coefficients at the outputs. Hsiao and Seth [59] further expanded that work to compact testing where the signature formed by compaction of the output responses is chosen to be a coefficient from the Rademacher-Walsh (RW) spectrum of the function under test. More recently, Giani *et al.* [41, 42] reported spectral techniques for sequential ATPG and built-in self-test. In [41] a spectral test generation scheme for sequential circuits is proposed, where

in, starting from pseudo-random vectors the generation of new test vectors is guided by the spectral components of previously beneficial generated vectors. In [42], a spectral-based BIST scheme was proposed, in which test vectors were generated from stored prominent spectral components by executing a program on a processor. Hsiao's group at Virginia Tech have published further work on spectrum-based self test and core test [18, 19, 72]. Khan *et al.* [73, 74] have designed hardware output response compactors which use digital spectral analysis. Zhang *et al.* [158] further refined the method of extracting the spectra from a binary signal using a selfish gene algorithm. Recent work suggests that wavelet transforms can also be used for similar applications [25]. Due to the encouraging results published in earlier works, we shall use spectral methods in our work for addressing the issues of test generation.

3.1 Hadamard transform and Walsh functions

Spectral analysis of a digital signal is a decomposition process, in which the signal is represented as a linear combination of a set of orthogonal functions. These orthogonal functions are defined by their corresponding transforms. Several transforms [137] have been developed over the years which can be used for digital signals. Hadamard transform and Haar transform are two examples of those. We use Hadamard transform [133, 137, 139] in our proposed works for spectral analysis because of their ease of use and also since they have been used for testing with effective results.

The Hadamard transform decomposes a digital signal into a superposition of a set of orthogonal functions called Walsh functions. Walsh functions consist of trains of square pulses having $+1s$ and $-1s$ as the allowed states and can only change at fixed intervals of a unit time step. For an order n, there are $N = 2^n$ Walsh functions, given by the rows of the $2^n \times 2^n$ Hadamard transform matrix $H(n)$ [137], when the functions are arranged in the so-called "natural" order [133, 139].

Hadamard transform matrix can be defined in two ways [46], using a binary (base-2) representation or recursively. Using a binary (base-2) representation, the element at the jth row and k^{th} column of the Hadamard matrix is given by:

$$h(j,k) = \frac{1}{N}(-1)^{f(j,k)} \ \ where \ \ f(j,k) = \sum_{i=0}^{n-1} b_j[i]b_k[i] \tag{3.1}$$

$b_j[i]$ and $b_k[i]$ are the ith binary bits of the corresponding binary numbers b_j and b_k respectively. b_j and b_k are the binary represention of the corresponding integer values j and k given by the following relations:

$$j = b_j[n-1]2^{(n-1)} + b_j[n-2]2^{(n-2)} + ... + b_j[1]2^1 + b_j[0]2^0 \tag{3.2}$$

$$k = b_k[n-1]2^{(n-1)} + b_k[n-2]2^{(n-2)} + ... + b_k[1]2^1 + b_k[0]2^0 \tag{3.3}$$

Hadamard matrices can also be generated using the following recurrence relation:

$$H(n) = \begin{bmatrix} H(n-1) & H(n-1) \\ H(n-1) & -H(n-1) \end{bmatrix} \tag{3.4}$$

where $H(0) = 1$ and 2^n is the dimension of the nth order Hadamard matrix, $H(n)$. For example, for $n = 1$ and $n = 2$, we have:

$$H(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \ \ \text{and} \ \ H(2) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{3.5}$$

The Hadamard matrix is a symmetric matrix with each row being a unique Walsh orthogonal function, also called the basis function bit-stream. Since it consists of only +1s and −1s, it is a good choice for the signals in VLSI testing (+1 = logic 1, −1 = logic 0).

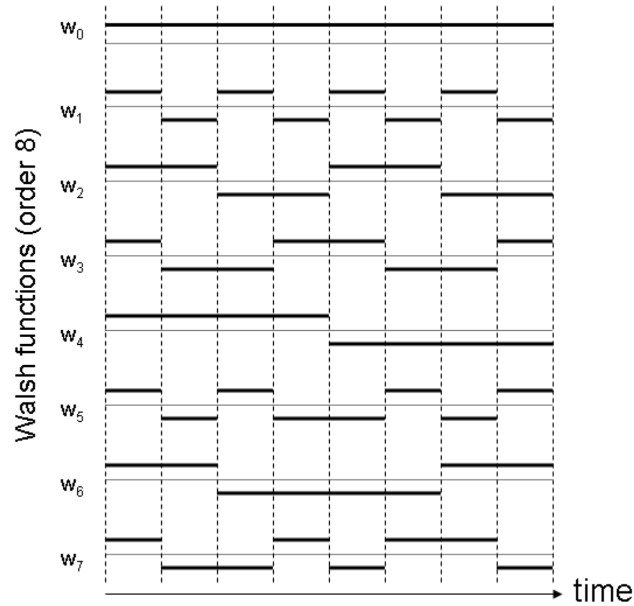


Figure 3.1: Walsh functions of order eight.

The Walsh functions include patterns with varying periodicities which are analogous to the sine and cosine functions in the analog domain. Hence, Walsh functions can be thought of as digital counterparts of analog frequencies. Figure 3.1 shows the schematic diagram of Walsh functions of order eight. Any digital bit-stream can be uniquely represented as a linear combination of the orthogonal Walsh functions. This is analogous to the analog domain where any continuous signal can be uniquely represented as a linear combination of the sine and the cosine functions. Thus, by analyzing the digital signals using Walsh functions, we are actually looking into the frequency or sequency characteristics of the digital waveforms. Frequencies refer to periodicities for analog signals, while sequencies refer to bit-flippings for digital binary waveforms [137].

3.2 Spectral analysis using Hadamard transform

Spectral analysis using Hadamard transform decomposes a digital signal or binary bit-stream into a superposition of orthogonal Walsh functions which correspond to different periodicities or sequencies (as they are sometimes referred to in the digital domain). The

goal of spectral analysis is to analyze the given binary bit-stream in the frequency or spectral domain and help in revealing its significant periodicities or spectral components. We explain below, how a binary bit-stream can be analyzed in the spectral domain and how its significant spectral components be determined.

Using Hadamard transform, any bit-stream of R bits can be represented as a linear combination of the orthogonal Walsh functions from the Hadamard matrix, $H(r)$ where $R = 2^r$. To perform the spectral analysis of a given bit-stream of R bits, 0s and 1s in the bit-stream are represented as -1s and $+1s$ respectively (as the Hadamard matrix consists of only -1s and $+1$s) and then the bit-stream is multiplied with the Hadamard matrix $H(r)$. The multiplication operation is basically a correlation of the bit-stream with each of the Walsh functions and gives a weighting value for each of the functions which we refer to as spectral coefficients. These coefficients give the degree of correlation of the original bit-stream with the different Walsh functions. A high magnitude value of the coefficient corresponds to a high correlation of the bit-stream to the corresponding Walsh function and vice-versa. By analyzing these coefficients we will be able to determine the major contributing Walsh functions in the original digital signal.

$$\begin{array}{ccc} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} & \rightarrow & \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \\ Original & & Modified \\ bit-stream & & bit-stream \end{array} \tag{3.6}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ \mathbf{6} \\ -2 \\ 2 \\ 2 \\ -2 \\ -2 \\ 2 \end{bmatrix} \tag{3.7}$$

$$Hadamard\,matrix \quad Modified \quad Spectral$$

$$H(3) \quad bit-stream \quad Coefficients$$

Equations (3.6) and (3.7) show an example of generating spectral coefficients for an 8-bit binary bit-stream. As shown in equation (3.6), 0s and 1s in the original bit-stream are represented as -1s and $+1$s respectively to generate a modified bit-stream. Equation (3.7) shows the spectral analysis of the modified bit-stream by multiplying it with a third order 8×8 Hadamard matrix, H(3). The result of the matrix multiplication yields the spectral coefficients. Figure 3.2 shows a graphical representation of the spectral coefficients obtained in equation (3.7). From the spectral coefficients obtained in equation (3.7) and displayed in Figure 3.2 it can be observed that, the Walsh function W1 has a prominent value of $+6$, while other components have lower or trivial values ranging between -2 and $+2$. From this we can deduce that, the binary bit-stream exhibits a higher correlation with and is more similar to the Walsh function W1 than the other Walsh functions. Hence we say that the Walsh function W1 is an essential or prominent spectral component of the given bit-stream and other Walsh functions are considered to be non-essential or noise-like. It is important to note here that, Hadamard transform conforms with Parseval's theorem [103], used frequently with analog signals, and that the total energy of the spectral coefficients, given by the sum of its squares, is conserved from the time domain and is constant [9].

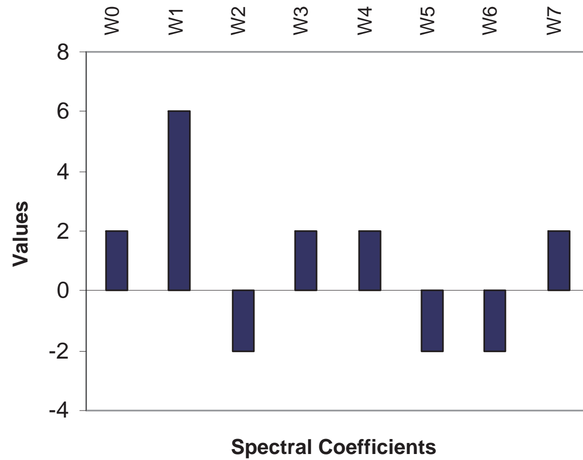


Figure 3.2: Graphical representation of Walsh Coefficients obtained in equation (3.7).

3.3 Information content and randomness

By observing the relative magnitudes of the spectral coefficients derived for a given binary bit-stream, the information content and the randomness in the bit-stream can be determined. Several works [32, 67, 157] have been published which describe determining the amount of randomness in a binary bit-stream. Kak [67] describes a method of determining a pattern and estimating the randomness in a binary bit-stream using Walsh-Fourier transform. A measure of randomness is defined as the ratio of the number of independent amplitudes in the Walsh-Fourier transform of the binary bit-stream to the length of the bit-stream itself. Yuen [157] proposes a randomness test which considers the flatness of Walsh power spectrum of the bit-stream. A truly random sequence will have an asymptotically flat Walsh power spectrum. Feldman [32] improved on Yuen's randomness test and proposed a new test which was based on an evaluation of the Walsh spectrum.

Another method which can be used to determine the information content and hence the randomness in a bit-stream is by using spectral entropy [92, 145, 146]. Shannon's entropy has been effectively used to measure the amount of information and randomness in data. The concept of Shannon's entropy can be extended for a frequency spectrum by normalizing the magnitudes of the frequency components and using the magnitudes in the

entropy formula. For a frequency spectrum with components x_n, the spectral entropy is given by:

$$E_s = -\sum p_n \log_2(p_n),\ where\ p_n = \frac{|x_n|^2}{||x||^2} \tag{3.8}$$

The spectral entropy effectively measures the flatness of a frequency spectrum. A flat spectrum is analogous to high entropy or large randomness. A peaked spectrum has low entropy and represents a relatively more deterministic signal. Spectral entropy differentiates itself from normal entropy analysis of bit streams by considering the bit-streams in relation with the fundamental orthogonal Walsh functions and the periodicity they exhibit. For example, consider a binary bit-stream which has a period of four time units and a 50% duty-cycle. Shannon's entropy analysis would calculate the probability of logic 1 and logic 0 to be equal to 0.5 and hence determine its entropy to be equal to 1. However if perform a spectral analysis of this bit-stream, then we shall obtain only a single frequency component corresponding to a period of four time units. Spectral analysis of this frequency spectrum would correctly classify the bit-stream as having low entropy. Hence spectral entropy determines the information content and randomness in a signal by considering the periodicities inherent in the signal.

3.4 Spectral analysis for test generation

The use of spectral analysis in test generation, is based on the premise that test vectors, having high fault coverage, exhibit discernible spectral characteristics. By reproducing these characteristics, effective high fault coverage test vectors can be generated. The essential spectral characteristics are unique and representative of the Circuit-Under Test (CUT) and can be obtained from various sources like Register-Transfer Level (RTL) information, verification test vectors, behavioral description, etc. We shall propose methods in chapters 6 and 8, which will use the concepts of spectral analysis as an integral part, for retrieving and analyzing essential spectral properties for performing test generation.

Chapter 4

Test Vector Minimization

Due to the recent advances in CMOS fabrication, the design complexity of digital systems is growing exponentially with time. This has not only led to a proportional increase in test generation complexity but also an increase in the test data volume or the number of test vectors that need to be applied. The number of test vectors that need to be applied to test a digital circuit directly affects the testing time for the circuit and hence the testing cost. Therefore, it has become increasingly important to reduce the number of vectors required to satisfactorily test a circuit. Test vector minimization or compaction, as it is sometimes referred to, is a process of minimizing the number of test vectors while retaining their coverage of detection of a given set of faults.

Several techniques have been published over the years for performing test vector minimization or compaction [7, 44, 51, 66, 106]. The techniques can be roughly classified as static compaction and dynamic compaction [44]. In static compaction, an already generated test vector set is reduced by removing the redundant test vectors, while maintaining the fault coverage of the original test set. Since static compaction selects test vectors from a given test set, the amount of compaction that can be performed is limited by the nature of the original test set. In dynamic compaction, additional test vectors can be generated to perform compaction, which can lead to a better compacted test set than static compaction. However, dynamic compaction requires a heuristic and/or algorithm to generate additional test vectors. Static compaction on the other hand is simpler to implement as it only needs to evaluate the quality of the given test set. Also the static compaction algorithm can be easily incorporated in a given test generation framework. For these reasons, our work on test generation uses a static compaction based approach for test vector minimization.

Several techniques have been published for performing static compaction. One of the simplest examples of static compaction is reverse order fault simulation in which after the test vectors are generated to cover the targeted faults, the test vectors are fault simulated in a reverse order, i.e. starting from the test vector which was generated last to the test vector which was generated first. After the maximum fault coverage is achieved, the remaining test vectors are discarded as redundant. The reasoning behind this technique is that, towards the end of the test generation process test vectors are generated to cover the hard to detect faults, which can also detect some easy to detect faults. A method was described in [44] where test vectors which are compatible with respect to their logic values of 0s and 1s are repeatedly merged to generate a reduced test vector set. Enhanced techniques were later proposed in [16, 115] where values in certain test vectors were modified to detect faults, which were already detected by some other test vectors. The redundant test vectors could then be removed, preserving the fault coverage. The above mentioned methods are based on heuristics and can be termed as non-exact methods where the solution is not guaranteed to be minimum. A class of techniques [27, 34, 55, 70, 85] were proposed based on Integer-Linear Programming (ILP) [122] for test vector minimization. Although these methods are a little expensive in terms of computational complexity they provide an exact solution to the test vector minimization problem. Techniques [27, 69, 71, 112, 124] have been proposed to reduce the complexity of ILP computations and obtain a sub-optimal solution. The problem of minimizing test vectors using ILP considering their diagnostic properties has been addressed in [123, 125].

4.1 Linear programming for test vector minimization

Linear programming (LP) is a technique for optimizing (minimizing or maximizing) a linear objective function subject to a given set of linear inequalities. A set of variables defined as real numbers are subject to a set of linear inequalities. The problem is then to optimize a linear objective function defined over the set of variables. Mathematically a linear programming problem is modeled using the following three components:

- Set of variables $X \in R$
- Set of linear constraint inequalities over X
- Linear objective function to optimize: $f(X)$

By solving the LP problem, values are obtained for the variables in X, such that the objective function $f(X)$ is optimum under the given set of constraints. An Integer Linear Programming (ILP) model formulation is a special case of LP, wherein the variables in X are defined as integers as opposed to real numbers. The worst-case time complexity of solving an LP problem is found to be polynomial-time (P), while that for an ILP problem is shown to be non-polynomial-time (NP-hard).

4.2 ILP formulation for test minimization

As mentioned earlier, the test vector minimization problem can be formulated as an Integer Linear Programming (ILP) problem. Consider a set of n test vectors, $T = \{t_1, t_2, ..., t_n\}$ which collectively detect a set of m faults, $F = \{f_1, f_2, ..., f_m\}$. By fault simulation the individual faults detected by each test vector are determined and say a constant C_{ij} ($i = 1 .. n$; $j = 1 .. m$) is defined for each fault-test vector pair such that $C_{ij} = 1$ if the ith test vector detects the jth fault, else $C_{ij} = 0$. A [0,1] integer variable x_i ($i = 1..n$) is defined for each test vector t_i, such that if $x_i = 1$ test vector t_i is included in the minimized test set, else it is discarded. The ILP formulation can then be modeled as:

- Set of variables $x_i \in [0, 1] \quad \forall i = 1 .. n$
- Subject to a set of linear constraints (one for each fault j):

$$\sum_{i=1}^{n} C_{ij} x_i \geq 1 \quad \forall j = 1 .. m \tag{4.1}$$

- Minimize the objective function:

$$\sum_{i=1}^{n} x_i \tag{4.2}$$

On solving the above ILP problem, [0,1] values are assigned to the variable $x_i \;\; \forall i = 1 \,..\, n$, such that the objective function $\sum_{i=1}^{n} x_i$ is minimized. A test vector t_i then forms a part of the minimized test vector set if $x_i = 1$.

The worst-case complexity for solving an ILP problem is NP-hard and hence for certain scenarios, solving by the ILP method may not be feasible. An alternative approach to solving an ILP problem is by using relaxed LP [27, 71, 112]. A relaxed LP problem is formulated in the the same way as an ILP problem described above, except that the set of variables are defined as real numbers as opposed to integers. A real-valued solution is obtained for the set of variables in polynomial time using LP and then the integer values for the variables are determined by some heuristics. In Chapter 7, we propose such a method for solving the ILP problem of test minimization using relaxed LP.

Chapter 5

Built-In Self Test

The Design-For-Test (DFT) method of Built-In Self Test (BIST) was briefly introduced in Section 2.5. As described earlier, BIST is a technique in which additional hardware is inserted to test the Circuit Under Test(CUT) and determine whether it is good or bad [3, 4, 89, 94, 127]. The general block diagram of a BIST architecture is as shown in Figure 5.1

The two main components of the BIST architecture are the test generator and the response analyzer. The test generator generates optimum test vectors to test the CUT such that a high fault coverage is achieved. One of the challenges in the design of the test generator is to keep the area overhead minimum, and at the same time maintain the ability to generate relevant test vectors with high fault coverage for testing the CUT. The response analyzer samples the primary outputs of the CUT, compares it with the expected good responses and flags whether the CUT is good or bad. The wrapper provides the facility of configuring the CUT in a test mode. Along with these components, a BIST controller may be required to control the running of the BIST sessions. An efficient BIST architecture should be designed in such a way that it has low area overhead, high fault coverage and low test application time.

We shall briefly review the prior work in the area of BIST and then discuss the two main components of the BIST architecture, which are the test pattern generator and the output response analyzer in the following sections.

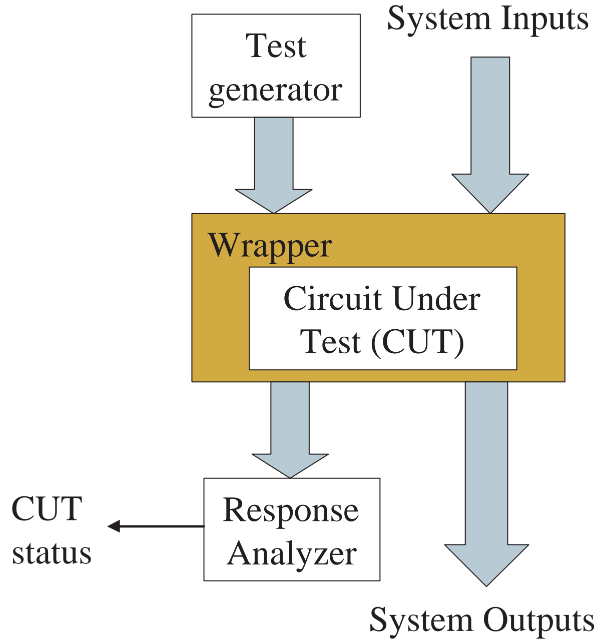


Figure 5.1: General BIST architecture.

5.1 Prior work

We shall be concentrating on non-scan based implementation of BIST where the CUT is tested using only its inputs and outputs. Several articles have been published on non-scan based BIST using random and weighted random patterns [6, 10, 95, 96, 141]. Some methods [95, 141] are based on modifying the flip-flops to increase the number of reachable states of the sequential circuit to enhance the fault coverage. In another approach [96], selected input patterns from a set of pseudo-random patterns generated by an LFSR, are held for several time units to give high fault coverages. Several methods [6, 10, 94, 140, 142] are based on weighted random patterns. Some authors [5, 65] generate weighted random patterns using counters, while others [17] use bit fixing. In [108, 110], a parameterized structure for test pattern generation using counters and comparators was proposed. Although the reported fault coverages were close to the deterministic patterns, the area overhead was high. Generally, the above mentioned methods suffer from disadvantages like high area

or delay overheads, high engineering effort and inability to obtain consistently satisfactory results for a range of circuits.

Spectral analysis of sequential test patterns show that test sequences exhibit certain periodicities. Giani *et al.* [41] proposed a spectra based test generation scheme, where new vectors were generated replicating the enhanced spectral components of earlier vectors, which detected faults. For a BIST environment, they [42] proposed an extension of that method [41] using an in-built microprocessor to generate the vectors. Chen and Hsiao [18] modified that method [42] for only the hard to detect faults. Bushnell *et al.* [26] propose a Haar wavelet based BIST method for sequential circuits which gives fault coverages close to the deterministic patterns for several circuits.

5.2 Test pattern generator

A Test Pattern Generator (TPG) is one of the important components of a BIST architecture which generates and applies test vectors to the CUT to test it. There are several methods by which test vectors can be obtained and applied to the CUT. Deterministic TPGs read and apply test vectors stored in a Read Only Memory (ROM). These test vectors could be either manually generated or obtained using a test pattern generation program. However this approach becomes infeasible as the number of test vectors grows exponentially with the design size. Other methods of test generation can be classified as concurrent test pattern generation [89] in which test vectors are calculated as they are being applied.

Several techniques have been proposed for concurrent test generation. Algorithmic TPGs generate test vectors based on some heuristics or properties that help in testing the CUT. Finite State Machines (FSMs) can be implemented to generate such test vectors. If the CUT includes or has access to a processor and a memory, then a test program can be run by the processor to generate the required test vectors [47, 54, 60, 77]. The test program can then be stored in a ROM. These test programs are generally written to exercise the functionality of the design and may require certain initializations or reconfiguration of the CUT. Another example of algorithmic generation of test vectors can be seen in BIST for

Random Access Memories (RAMs) where a sequence of test vectors are generated to perform a series of well-defined operations [135].

A class of TPGs have been proposed over the years which can be viewed as independent of the functionality or implementation of the CUT. An example of such test generators is the exhaustive TPG which generates all possible test patterns for the CUT. For an n input combinational circuit, the exhaustive TPG generates all 2^n possible test vectors, which detect all detectable gate-level stuck-at faults and bridging faults. However it will not guarantee complete coverage of delay faults and transistor-level faults. The exhaustive TPG can be implemented conveniently using an n-bit counter. However this approach becomes impractical in terms of area overhead and application time as the number of inputs n of the circuit grow. A variation of exhaustive testing is pseudo-exhaustive testing, in which the combinational circuit is partitioned into sub-circuits and each sub-circuit is tested exhaustively [88]. This approach becomes more practical as opposed to exhaustive testing as the number of inputs n of the circuit grows. This approach may not work well for circuits where exclusive partitioning of the circuit is difficult.

Another type of TPGs which are independent of the CUT are the pseudo-random test generators [8]. These TPGs are able to generate test vectors which have random-like properties and at the same time generate almost all possible combinations of test vectors. Due to these properties, the pseudo-random test generators are widely used in BIST architectures. There are mainly two types of implementations used for generating pseudo-random patterns, Linear Feed-back Shift Register (LFSR) and Cellular Automata Register (CAR) which will be explained in detail in the next subsections.

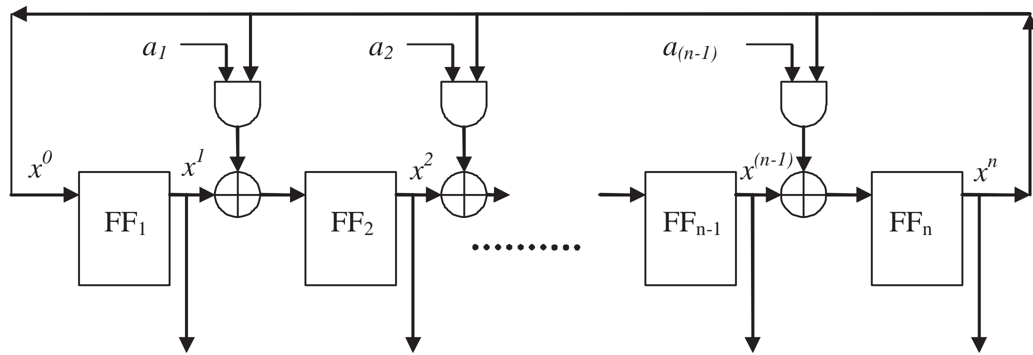
A variation of the pseudo-random test generator is the weighted pseudo-random TPG [119, 13], in which the probability of a bit being logic 0 or logic 1 is altered depending on the nature of the CUT inputs. It has been observed in several circuits that, certain inputs require probability of being logic '1' (p_1) other than 0.5 for testing the circuit. An example of such an input is a global reset signal to a sequential circuit, which needs to be deactivated with a higher probability in order to help test the circuit. Also circuits

which have an AND/NAND network or an OR/NOR network of gates require test vectors whose probabilities of logic '1' (p_1) are different from 0.5 to test the random-pattern resistant faults. Weighted pseudo-random TPGs can be implemented by combining the pseudo-random TPGs with gates like NANDs and NORs to produce the required weighting of the bits [118]. For a NAND gate, if both its inputs have the probability of being logic '1' as p_1=0.5, then the probability for its output to be logic '1' is given by $p_1(output) = 1 - p_1(input1) \times p_1(input2) = 0.75$. Similarly for a NOR gate, if both its inputs have p_1=0.5, then for its output $p_1(output) = (1 - p_1(input1)) \times (1 - p_1(input2)) = 0.25$. A combination of NAND and NOR gates can then be used to generate the required weighting of bits. A universal weight generator was proposed in [118] to generate an arbitrary weighting of pseudo-random bits.

Since the pseudo-random TPGs are widely used in a BIST architecture including our proposed work, we shall explain these in more detail. As mentioned earlier, the pseudo-random TPG can be implemented in mainly two ways: Linear Feed-back Shift Register (LFSR) and Cellular Automata Register (CAR). We explain these implementations in the following sections.

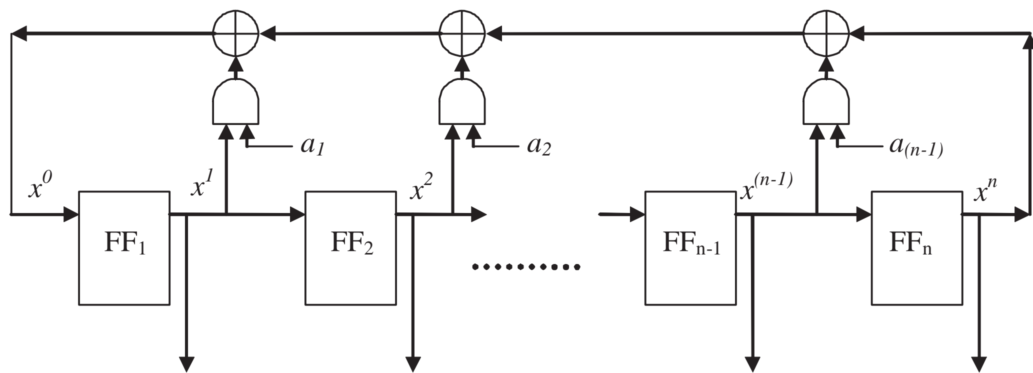
5.2.1 Linear Feedback Shift Register (LFSR)

An LFSR [28, 127] is one of the most frequently used TPGs, as compared to other TPGs, as it uses less combinational logic per flip-flop and hence is area efficient. As the name suggests, an LFSR is a shift register with a linear feedback using XOR gates. There are two types of LFSRs defined based on the type of feedback: an internal feedback LFSR and external feedback LFSR. Figures 5.2 and 5.3 show the general implementation of the internal and external feedback LFSRs, respectively. As observed in the figures, in an internal feedback LFSR, XOR feedbacks are distributively placed in the shifting path of the register, while in an external feedback LFSR, the XOR feedbacks are placed external to the shifting path. The actual implementation of an LFSR is determined by its characteristic polynomial as shown in the figures. A characteristic polynomial takes the form:



Characteristic polynomial $P(x)$: $x^n + a_{(n-1)}x^{n-1} + \ldots + a_1x^1 + 1$; *where* $a_i = [0,1]$

Figure 5.2: Generic N-bit internal feedback shift register.



Characteristic polynomial $P(x)$: $x^n + a_{(n-1)}x^{n-1} + \ldots + a_1x^1 + 1$; *where* $a_i = [0,1]$

Figure 5.3: Generic N-bit external feedback shift register.

$$P(x) = x^n + a_{(n-1)}x^{(n-1)} + ... + a_1x^1 + 1; where \ \ a_i = [0, 1] \tag{5.1}$$

The variables x^i represent the outputs of the flip-flops as indicated in the figures 5.2 and 5.3. Based on the values of the coefficients a_i, the corresponding XOR feedbacks are activated or deactivated as signified by the AND gates shown in the figures. The characteristic polynomial, by specifying the feedback connections, determines the sequence and the total number of unique vectors that can be generated by the corresponding LFSR. A characteristic polynomial for an n-bit LFSR, which is able to generate a maximal length test vector sequence of $2^n - 1$, is called a primitive polynomial. Since in a BIST environment, it is important to generate as many unique test vectors as possible, primitive polynomials play an important role in the design of LFSRs. Tables of primitive polynomials for various values of n have been published in literature [8, 105, 127] which can be used for designing a maximal-length n-bit LFSR. One of the limitations of the LFSRs is that, due to the inherent built-in shift register, the n-bit test vector sequences generated exhibit a shifting pattern and thus a correlation among the bits, which might not be effective in testing some circuits.

5.2.2 Cellular Automata Register (CAR)

A Cellular Automata Register (CAR) [57, 127] is similar to an LFSR, however it does not exhibit the shifting pattern observed in the test vectors generated by an LFSR. The construction of a CAR requires more XOR gates (anywhere between one and two) per flip-flop than an LFSR. A CAR is constructed in such a way that each flip-flop is a function of its neighboring flip-flops. More specifically, the next state value of a flip-flop FF_i in an n-bit register is a function of the current state values of the flip-flops $FF_{(i-1)}$, FF_i and $FF_{(i+1)}$. Flip-flop $FF_{(i-1)}$ is a flip-flop preceding the considered flip-flop FF_i and flip-flop $FF_{(i+1)}$ is a flip-flop succeeding the flip-flop FF_i. Based on the current state values of the three

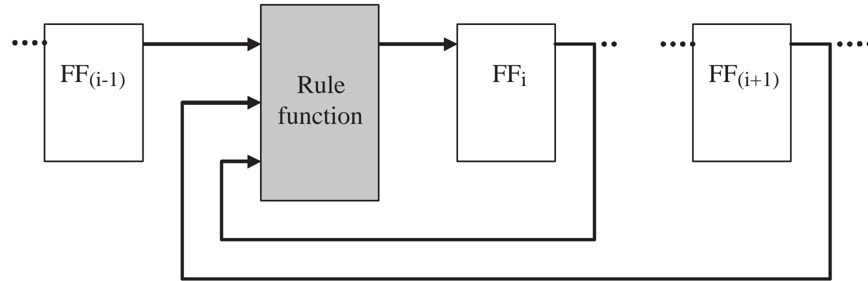


Figure 5.4: Generic rule function for a flip-flop.

flip-flops $FF_{(i-1)}$, FF_i and $FF_{(i+1)}$, a truth table for the next state values of the considered flip-flop FF_i is constructed. Since there are 3 input values, $2^{2^3} = 2^8 = 256$ possible next state value truth tables for FF_i can be constructed. Each such possible truth table is called a "Rule" and is named by the decimal equivalent of the truth table binary values. Hence in all there are 256 rules; from rule 0 to rule 255.

Figure 5.4 shows the schematic diagram of a generic rule function implementation for a considered flip-flop FF_i. The current state values of the three flip-flops drive the inputs of the next state calculation logic or Rule Function as referred to in the figure. The output of the rule function provides the next state value for the flip-flop FF_i.

Table 5.1 gives three examples of rule functions for a flip-flop in a CAR. The first three columns give the different possible values of the current state of the three flip-flops. The rule name is derived from the binary values of the truth table. The fourth column gives the corresponding bit position weight for each bit in the truth table. Columns five to seven give three example rule functions that can be used for a flip-flop. The example in column five implements a shift register as can be observed from the truth table values that ($FF_i = FF_{(i-1)}$) and is referred to as Rule 240. Columns six and seven give two other examples of rule functions and they are named Rule 90 and Rule 150, respectively.

For a stage or flip-flop in an n-bit register, any of the rules from rule 0 to 255 can be used. For the flip-flops at the boundaries of the CAR, since one of their neighboring flip-flops is missing, two options could be used for implementation. The first is to assume a

Table 5.1: Example rule functions for CAR.

Current state values			Bit position weight	Next state truth tables for FF_i		
$FF_{(i-1)}$	$FF_{(i)}$	$FF_{(i+1)}$		Rule 240	Rule 90	Rule 150
1	1	1	2^7	1	0	1
1	1	0	2^6	1	1	0
1	0	1	2^5	1	0	0
1	0	0	2^4	1	1	1
0	1	1	2^3	0	1	0
0	1	0	2^2	0	0	1
0	0	1	2^1	0	1	1
0	0	0	2^0	0	0	0
Decimal equivalent				240	90	150

null boundary condition, where the missing flip-flop is assumed to be of logic value 0. The second option is to assume a cyclic boundary condition where the end flip-flops of the CAR are assumed to be adjacent in a cyclic fashion and provide logic values to each other. Either condition can be used, however only null boundary condition is able to provide a maximal length sequence. Furthermore, null boundary condition uses fewer gates and avoids the performance penalty due to long feedback paths between the end flip-flops as compared with the cyclic boundary condition.

Among all the rules, rule 90 and rule 150 are the most widely used rules in the implementation of a CAR, as a combination of these is able to provide a maximal-length pseudo-random sequence. As can be observed from the truth tables in figure 5.4, in rule 90, the next state value of a flip-flop FF_i is an XOR operation of the current state values of its neighboring flip-flops $FF_{(i-1)}$ and $FF_{(i+1)}$. In rule 150, the next state value of a flip-flop FF_i is an XOR operation of the current state values of the flip-flops $FF_{(i-1)}$, FF_i and $FF_{(i+1)}$. Figures 5.5 and 5.6 show the implementation of the rules 90 and 150 for a considered flip-flop FF_i. Similar to a characteristic polynomial of an LFSR, for an n-bit register, the combination of rules 90 and 150 to be used for each stage can be looked up in a precomputed table. These tables have been published in the literature [56, 104, 127] which can be used conveniently to design any given n-bit CAR.

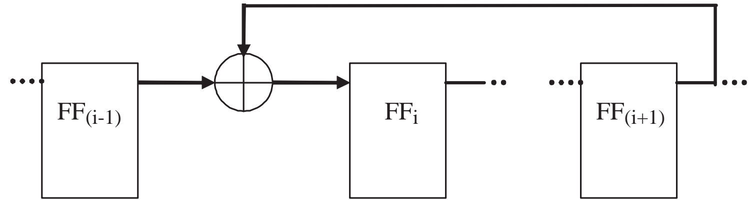


Figure 5.5: Rule 90 implementation of a CAR.

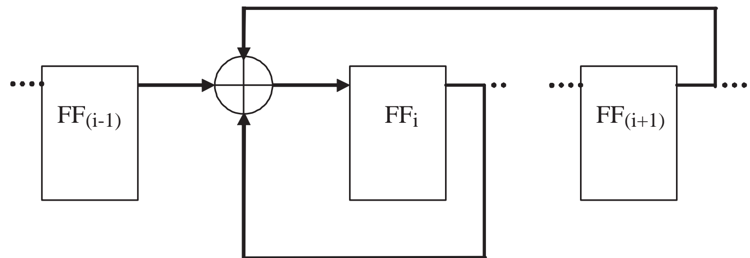


Figure 5.6: Rule 150 implementation of a CAR.

5.3 Output response analyzer

Another important responsibility of a BIST architecture, other than producing good quality test vectors, is to sample the response of the CUT to all the applied test vectors and determine whether any test failed. The Output Response Analyzer (ORA) performs an important task to this effect, by compacting the output responses of the CUT to the test vectors applied and facilitating the determination of a pass/fail indication for the CUT.

There are several methods and techniques [8, 138, 127] which can be used to implement the ORA. Concentrators use an XOR tree to compact multiple outputs of the CUT into a single output bit-stream. Although the concentrator compacts the output data spatially (for multiple outputs), some other technique needs to be used along with that to perform the compaction of output data over time. A comparator-based ORA performs a comparison of the output responses of the CUT to the expected good responses on a vector-by-vector basis. The expected good responses could be stored in a ROM. This approach is generally discouraged due to high area overhead, except for some applications where it would be

beneficial. A counter can also be used as an ORA, where a count could be maintained for the number of 0s, 1s or transitions in the output response, which can then be compared to the expected good circuit count after the testing session ends. Similarly, parity can calculated for the output response data and checked with the expected good circuit parity. A variation of the counter and the parity checker as an ORA, is the use of an accumulator which sums the values of the output responses, which are treated as a binary number. The accumulated binary value can then be compared with the good circuit value to determine the pass/fail status of the CUT. One of the widely used implementations for an ORA is the Signature Analyzer, which uses an LFSR to compact the output responses within the register. The output responses of the CUT are XORed with the inputs of the flip-flops of the LFSR. After the test sequence has been applied, the value in the LFSR is compared with the expected value to determine whether the CUT is good or bad.

Chapter 6

Spectral RTL Test Generation

In this chapter we shall propose a spectral-based test generation method for sequential circuits which uses information from the Register Transfer Level (RTL). Conventionally, test vectors are generated at the gate-level, i.e., after synthesis has been performed. Though this methodology has the advantage of being able to generate reliable, high fault coverage test vectors due to its direct use of the gate-level fault models, it suffers from certain disadvantages. For large circuits, the number of faults and the algorithm complexity make the gate-level test generation time consuming and expensive. Since gate-level test generation is performed at a later stage in the design process it is difficult to deal with testability issues, revealed during test generation, in an already verified design. Also, the gate-level Automatic Test Pattern Generator (ATPG) cannot be used for cores or circuits for which only the functional information is available. This scenario is frequently encountered in commercial environments.

Test generation performed at the RTL, eliminates the disadvantages of gate-level test generation discussed above. The amount of information to be processed at RTL is much lower than at gate-level and bottlenecks in testability can be uncovered and remedied with some ease. Also since RTL test generation has no prerequisite of a synthesized circuit, it is synthesis independent, and hence enables test generation to begin earlier in the design cycle.

Several RTL test generation methods have been proposed in the literature. Ravi and Jha [114], Ghosh and Fujita [39], Kim and Hayes [75] and Goloubeva *et al.* [45] use pre-computed test sets for RTL constructs like adders, multiplexers, etc., and derive test vectors for the whole RTL circuit. Pre-computed test sets either make some assumptions about the synthesis of the design or use a superset of the actually required test vectors. All of

them use some kind of data structures or metrics to derive the RTL test sets, which have implications of large memory and/or computation overheads. Ravi and Jha [114], and Kim and Hayes [75] use controllability and observability metrics, while Ghosh and Fujita [39] and Goloubeva *et al.* [45] use data structures called *decision diagrams*. Yi and Hayes [147], and Pomeranz and Reddy [111] have proposed a fault model which considers fault activation and propagation from the primary inputs to primary outputs. Test vectors can then be generated to cover these faults. However, these fault models have been restricted to only combinational circuits. Thaker *et al.* [129, 130, 131, 132] have shown that a set of stuck-at faults on variables in high-level synthetic operators and at the boundaries of RTL modules can be used as a statistical sample for the gate-level coverage analysis.

Several papers propose test generation schemes using spectral methods. In [41] a spectral test generation scheme for sequential circuits is proposed, where in, starting from pseudo-random vectors the generation of new test vectors is guided by the spectral components of previously beneficial generated vectors. Hsiao's group at Virginia Tech have published further work on spectrum-based self test and core test [18, 19, 72]. Zhang *et al.* [158] further refined the method of extracting the spectra from a binary signal using a selfish gene algorithm. Recent work suggests that wavelet transforms can also be used for similar applications [25].

In this chapter, we present a spectral method of generating test vectors for sequential circuits using RTL faults. The RTL faults are defined as faults on the inputs and outputs of the circuit and inputs and outputs of the flip-flops, since these faults remain invariant through synthesis. Test vectors are generated to cover these RTL faults. Since the RTL faults form only a small fraction of the total number of faults, the time complexity of this step is low. The RTL test vectors thus generated are spectrally analyzed for prominent Walsh function components and the noise level. New test vectors are then generated using these properties to cover all the gate-level faults. These test vectors, which are generated using spectral properties of RTL test vectors, are found to detect almost as many faults as any gate-level ATPG. Besides, the sequences can be compacted to about the same size as

that produced by the gate-level ATPG. In the RTL method, therefore, the use of gate-level ATPG is eliminated and only a fault simulator is used. Our proposed RTL test generation scheme is described in the following sections.

6.1 Spectral RTL ATPG

As described earlier, in our proposed RTL test generation scheme, spectral properties of test vectors, which cover RTL faults, are used to generate new test vectors which cover the gate-level faults. Our approach to RTL-based test generation consists of two principal steps:

1. RTL spectral characterization
   - Test generation for RTL faults
   - Spectral analysis
2. Gate-level test generation
   - Spectral vector generation
   - Vector sequence compaction and coverage analysis

6.1.1 RTL spectral characterization

This is the first step in our proposed methodology, in which essential spectral properties beneficial for detection of gate-level faults are determined. These spectral properties are the characteristics of the digital circuit under consideration and can be obtained in various ways, such as using functional information, gate-level information, etc. In our proposed method we retrieve these spectral properties from test vectors which detect faults at RTL.

This step of RTL spectral characterization consists of two sub-steps, viz. test generation for RTL faults and spectral analysis, which are described below.

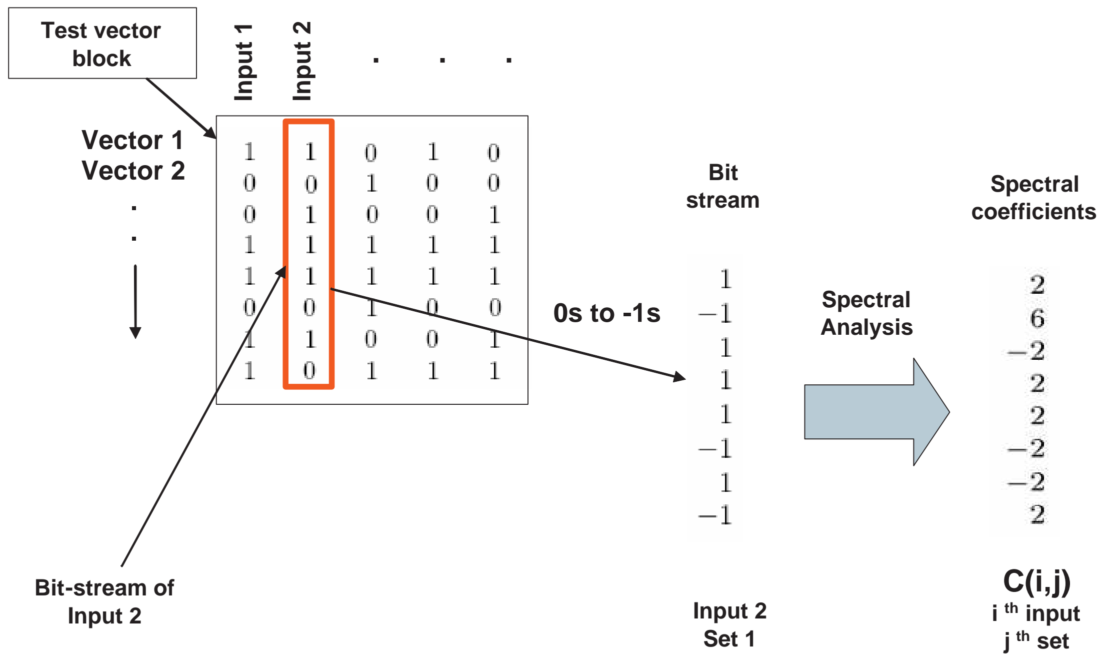


Figure 6.1: Spectral analysis of a test vector block.

Test generation for RTL faults

RTL faults are defined as faults on primary inputs and outputs of the circuit and on inputs and outputs of all flip-flops. Since these faults are also present at RTL and remain invariant through logic synthesis, we term them "RTL faults". In this step test vectors are generated to cover these faults. Any test generation scheme which works at RTL can be used to generate the test vectors. Test vectors can be generated with the circuit uninitialized or if a reset signal exists it is activated in the beginning of the test generation process to initialize the circuit.

Spectral analysis

As described in the previous step, test vectors are generated to detect the RTL faults. These vectors are analyzed using Hadamard matrix to find the major contributing Walsh components as described in Section 3.2. To analyze the test vectors, the bit-streams entering various inputs are analyzed separately. Figure 6.1 shows a schematic diagram of the process.

A test vector block, generated for detecting the RTL faults, will consist of test vectors as the rows in the block and the inputs of the circuit, to which the bits are being applied, as its columns. We perform a spectral analysis on the bits which are being applied to each of the inputs separately. As an example, shown in Figure 6.1, a spectral analysis is performed on the bits which are being applied to the input 2 of the circuit. As described in Section 3.2, a Hadamard matrix of order n is used to analyze the binary bits.

The number of bits that can be analyzed at a time, which is determined by the dimension of the Hadamard matrix (2^n), is always a power of 2. The RTL test vector sequences, however may not always be of lengths that are powers of 2. We have several options of analyzing the bit-streams. One option is to use a limited dimension Hadamard matrix and analyze pieces of the bit-stream individually. However this method has the disadvantage that, it might not be able to capture the long periodicity inherent in the bit-stream. The other option is to analyze the whole bit-stream together. To do that, we may either need to truncate the bit-stream or use a higher order Hadamard matrix than the length of the bit-stream. For practical reasons, we use a dimension which is closest to the length of the bit-stream. For example, for a bit-stream of length 135 we choose a Hadamard matrix of dimension 128 and analyze only the first 128 bits of the bit-stream. Analysis using a dimension of 256 is found to give rise to a large number of noisy components due to the uncertainty in the unspecified bits from 135 to 256 and hence presents difficulty in finding out the essential components. In cases where we choose a dimension greater than the length of the bit-stream, the bit-stream is extended with 0s up to the dimension of the matrix to indicate non-inclusion of these bits in the correlation operation.

As explained in Section 3.2 and illustrated in Figure 6.1, to perform the spectral analysis on bits of each input, 0s and 1s are represented as -1s and $+1$s respectively and then multiplied with the Hadamard matrix to obtain the spectral coefficients. The major contributing spectral coefficients for each input are determined by using a threshold value. The spectral coefficients whose magnitudes are above the threshold value are considered as essential components while others are neglected as noise.

To determine the threshold, which separates the essential components from the noise components, the power of each spectral coefficient is compared with the mean power of the total spectrum. For *white noise* we shall have all equal valued coefficients and their magnitudes will be nearly equal to the mean of the spectrum. Figure 6.2 shows the spectral coefficients obtained for an arbitrary random signal. The average of the spectrum is indicated with a dash-dotted line. As observed, the spectral coefficient values are nearly equal to the mean of the spectrum.

After squaring the coefficients, the magnitudes of the coefficients are compared with the mean of the spectrum by taking a ratio of the two. This compares the percentage of power in the coefficients to the mean noise power. If this ratio is greater than some constant K, then the coefficient, and hence the corresponding basis bit-stream, is considered to be an essential component or else is considered non-essential or noise. The constant K affects which coefficients are being considered as essential ones. A very high value of K makes the selection process very acute, selecting only few components as essential ones and vice versa for a low value of K.

As an example Figure 6.3 shows the spectral coefficients obtained for the input signal "DataIn[5]" for the experimental processor circuit PARWAN [97]. The high rising bars show high correlation with the corresponding basis bit-streams and these are considered essential components. The average power or the random noise level is indicated with a dash-dotted line. Spectral coefficients below the average power are considered to be noise components.

6.1.2 Gate-level test generation

After performing the first step of RTL spectral characterization of the test vectors, essential spectral coefficients for each input are obtained. In this second step of gate-level test generation, test vectors are generated from the extracted spectral information and they are compacted. The two sub-steps are described below.

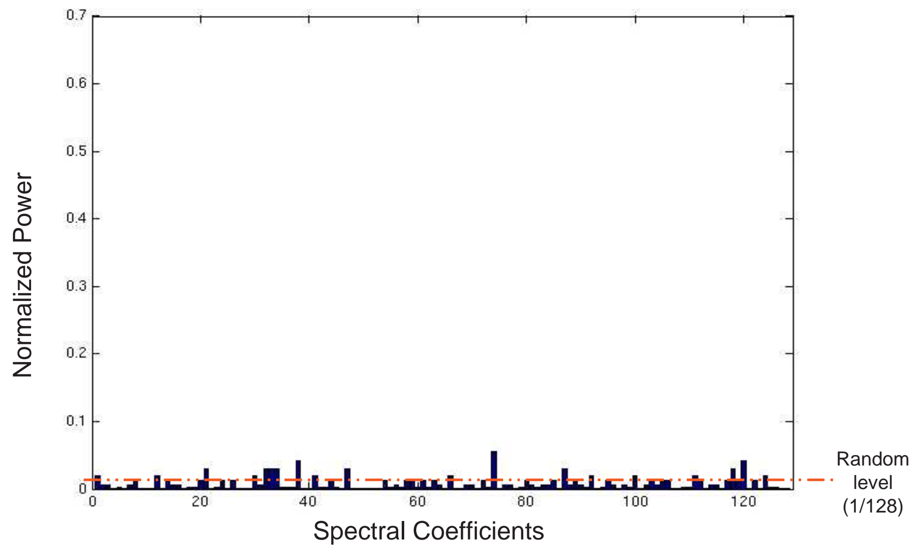


Figure 6.2: Spectral coefficients for an arbitrary random signal.

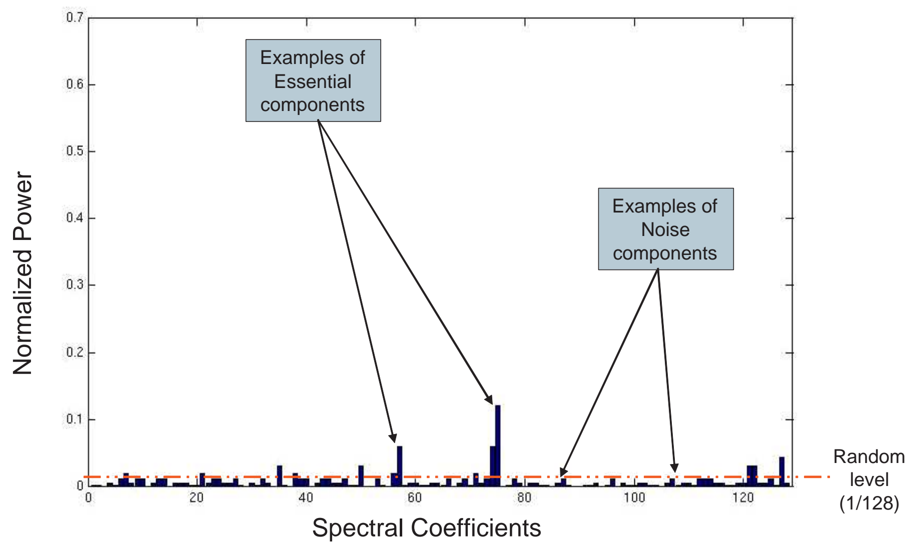


Figure 6.3: Walsh spectral coefficients for input DataIn[5] signal of PARWAN processor.

$$\text{(a) Perturbing spectra :} \begin{bmatrix} 2 \\ \mathbf{6} \\ -2 \\ 2 \\ 2 \\ -2 \\ -2 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ \mathbf{6} \\ 2 \\ -1 \\ 3 \\ -2 \\ 3 \\ -1 \end{bmatrix}$$

$$\text{(b) New bit} - \text{stream obtained from perturbed spectra :}$$
$$Sign\{[1 \;\; \mathbf{6} \;\; 2 \;\; -1 \;\; 3 \;\; -2 \;\; 3 \;\; -1] \times H(3)\}$$
$$= [1 \;\; 1 \;\; 1 \;\; -1 \;\; 1 \;\; -1 \;\; 1 \;\; -1] \rightarrow [1\; 1\; 1\; 0\; 1\; 0\; 1\; 0]$$

Figure 6.4: Bit-steam generation by perturbing the spectra.

Spectral vector generation

To generate new test vectors for gate-level faults, the essential spectral coefficients decided by the threshold in the first step are retained and others, being considered noise, can be filtered out or changed as per the selected methodology. Filtering out the noise components [41] may have the disadvantage of losing some phase information inherent in those components, which is a part of the characterization of the circuit. Hence in our approach the noise components are perturbed in a confidence range in terms of magnitude and/or in phase to generate new coefficients. The confidence levels correspond to the amount of randomness to be added.

After perturbing the spectral coefficients, test vectors are generated from the coefficients by multiplying the coefficients with the Hadamard matrix. Figure 6.4 (a) shows an example of perturbing the noise spectral coefficients. The major essential spectral coefficient of magnitude '6' is retained, while the other noise spectral coefficients are perturbed both in magnitude and/or sign randomly within a small confidence level. Figure 6.4 (b) shows the construction of the test vector from the perturbed spectral coefficients by multiplying with a 3rd order Hadamard matrix, H(3). In our method, we generate test vector sets with

different values of K. Also since we are adding noise randomly, this variation gives different characteristics to each vector set. We generate multiple sets of vectors for each value of K.

Vector sequence compaction and coverage analysis

After performing the spectral vector generation step, we obtain multiple sets or sequences of test vectors which all retain the essential spectral coefficients and use perturbed noise. The initial RTL test vectors are either generated starting from an uninitialized circuit or if a reset signal exists, the reset signal is activated at the beginning of the test sequence. Hence all the spectrally generated sets of test vectors are independent of each other and can be applied in any order.

The generated sets of test vectors will consist of several redundant sets and hence they can be compacted. We use an ILP formulation as described in Chapter 4 to formulate the problem of test vector compaction. Suppose the number of spectral components obtained for a circuit is N. We generate perturbation vectors sequences, $V_1, V_2, \cdots, V_M$, as described in the previous subsection, each of length N, such that their coverage as determined by fault simulation of the gate-level circuit either reaches some target value or simply saturates. Next, we compact the test by selecting the smallest number of these sequences without reducing the coverage.

During fault simulation, at the beginning of each vector sequence the complete fault list is restored and the circuit is set to an unknown state. Thus, the coverage obtained for a sequence remains valid irrespective to the order in which it is applied. In fault simulation, the fault simulator provides a complete list of vector sequences that detect each fault. The vector sequence V_i is assigned an integer variable, $x_i \epsilon [0,1]$ ($x_i = 1$: select the ith sequence or else discard it).

Suppose the kth fault is detected by sequences V_3, V_4, and V_{11}. Then the following ILP constraint is used to pick at least one sequence that detects this fault:

$$x_3 + x_4 + x_{11} \geq 1 \quad (6.1)$$

The number of such constraints equals the number of faults. The ILP then determines the values of variables x_i's that satisfy all constraints of the type 6.1 and provide an optimum value for the following objective function:

$$\text{Minimize} \sum_{i=1}^{i=M} x_i \tag{6.2}$$

where M is the total number of vector sequences generated. In the ILP solution, the smallest possible number of x's is assigned the value 1 and all others are assigned 0. The sequences with their x's set to 1 form the compacted test set.

6.2 Design-for-testability

Test generation for only the RTL faults has an additional advantage other than characterizing the circuit. It reveals the bottlenecks in the testability of the circuit. Analysis of hard to detect faults at the RTL gives an idea of the hard to test parts of the circuit. Hence by increasing the testability of hard to test faults, we could expect to increase the testability of the overall circuit. The testability of signals can be increased by increasing their controllability and/or observability. Several earlier works [53, 63] deal with increasing the testability by addition of control and observation points. Adding control points requires adding extra gates in the normal signal paths of the circuit, which may affect its performance. Hence we shall constrain ourselves to addition of observation points since they are less intrusive. One option is to add latches for all the observation points and connect them through a scan-out chain. This structure is often used for design debugging and can help in improving the fault coverage of tests [64]. Another option is to use an XOR tree to condense the logic values at the various observation points [24, 29, 36, 117]. Since the XOR tree incurs less hardware, we used it as our DFT methodology. Figure 6.5 shows an illustration of the RTL-based DFT scheme. Signals A, B and C are signals which exist at the RTL and are found to be hard-to-observe or unobservable during the RTL test generation step of Section 6.1.1. These signals are connected to an XOR gate chain and made

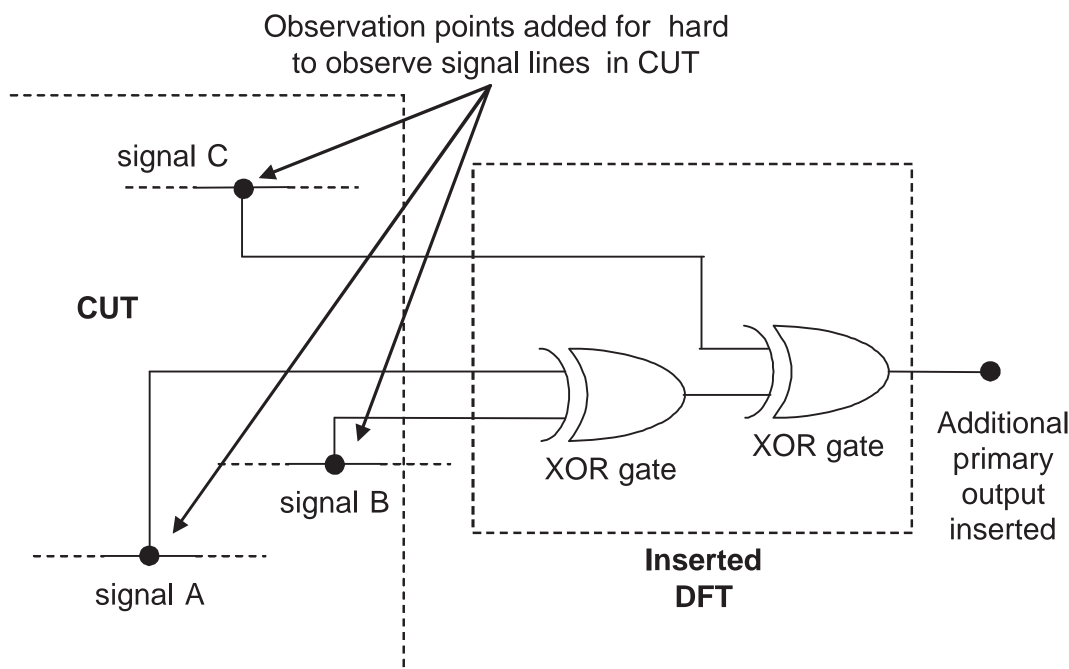


Figure 6.5: RTL-based DFT to improve observability of signals.

observable by making the output of the last XOR gate a primary output of the CUT. We implemented and evaluated our RTL DFT scheme for a simple accumulator-based processor named PARWAN [97] which will be discussed in more detail in the Section 6.3.

6.3 Implementation and results

We applied our spectral technique of RTL-ATPG to ITC'99 benchmark circuits, ISCAS'89 benchmark circuits and an experimental processor, PARWAN [97]. We describe the results of the benchmark circuits (ITC'99 and ISCAS'89) and the processor circuit PARWAN in separate sections below.

6.3.1 Results for ITC'99 and ISCAS'89 benchmark circuits

We implemented our spectral RTL-ATPG method on four ITC'99 benchmark circuits b01, b09, b11 and b14 and four ISCAS'89 benchmark circuits s1488, s5378, s9234 and

Table 6.1: Circuit description.

Circuit	**PIs**	**POs**	**FFs**
b01	2	2	5
b09	1	1	28
b11	7	6	31
b14	32	54	245
s1488	8	19	6
s5378	36	49	179
s9234	37	39	211
s35932	36	320	1728

s35932. The ITC'99 benchmark circuits are RTL/behavioral descriptions of digital designs and need to be synthesized into a gate-level netlist. Three of the ITC'99 RTL benchmark circuits (b01, b09, b11) were synthesized into gate-level netlists in two ways, by optimizing area and by optimizing delay. An area optimized netlist was used for the ITC'99 benchmark circuit b14. The characteristics of these different circuits are shown in Table 6.1. Column 1 gives the circuit name. Columns 2, 3 and 4 give the number of primary inputs (PIs), primary outputs (POs) and number of flip-flops (FFs) respectively for the circuits listed.

In the absence of an RTL/behavioral test generator, the test vectors for RTL faults were obtained using the Mentor Graphics tool FlexTest [91] which is a gate-level sequential ATPG system with a built-in fault simulator. Those RTL vectors were analyzed for their spectrum, new vector sequences were generated using the technique discussed above and finally they were compacted. The test sets were compacted using the ILP formulation as described in Section 6.1.2 by using the ILP software contained in the AMPL mathematical programming package [35]. Results were obtained on Sun Ultra 5 machines with 256MB RAM. Table 6.2 shows the characteristics of the RTL test vectors generated for the circuits. Column 1 lists the circuit name. Here b01-A and b01-D are the area and delay optimized implementations of the b01 ITC'99 benchmark. ISCAS'89 benchmarks are already at the gate-level. For s5378 and s9234, we created additional versions by adding a global reset input in the original circuits. These are denoted with an asterisk (*) in Tables 6.2 and 6.3.

Table 6.2: Spectral characterization of circuits by RTL vectors.

Circuit		RTL Characterization					
Name	Gate-level synthesis	No. of faults	No. of vectors	CPU s	No. of spectral coeffs.	RTL Cov. (%)	Gate-level Cov. (%)
b01-A	Area optimized	62	38	< 1	64	94.57	96.33
b01-D	Delay optimized	62	31	< 1	32	94.57	85.45
b09-A	Area optimized	248	109	519	128	75.22	78.18
b09-D	Delay optimized	248	193	418	256	75.22	72.69
b11-A	Area optimized	340	224	530	256	76.16	74.09
b11-D	Delay optimized	340	174	767	256	76.32	84.14
b14-A	Viper processor subset	2566	110	1684	128	63.53	47.14
s1488	ISCAS'89	104	38	1	64	83.12	64.34
s5378	ISCAS'89	1602	115	1185	128	55.36	68.82
s5378*	ISCAS'89+Reset	1962	82	444	64	69.22	65.04
s9234	ISCAS'89	1840	16	706	16	18.48	16.45
s9234*	ISCAS'89+Reset	2264	59	2495	64	49.85	43.58
s35932	ISCAS'89	14536	92	50	64	68.80	94.03

Column 3 of Table 6.2 lists the number of RTL faults, which are the faults at the primary inputs, primary outputs, and the inputs and outputs of flip-flops. For example, consider the circuit b11, which according to Table 6.1 has 7 primary inputs, 6 primary outputs and 31 flip-flops. These do not include the clock and reset inputs. Including those the primary input count is increased to 9. Each flip-flop has five terminals, three inputs (data, clock and reset) and two outputs (Q and $\overline{Q}$). Therefore, the RTL fault set consists of $2 \times (9+6+31 \times 5) = 340$ stuck-at faults as shown in column 3 of Table 6.2. Next in Table 6.2 appear the number of RTL test vectors, test generation time (CPU s) and the number of spectral components. In the absence of a true RTL ATPG program, we used FlexTest [91] to derive tests just for the selected stuck-at faults we designate as RTL faults. For the ITC'99 benchmarks this was done for two gate-level versions, one synthesized with area optimization and the other with delay optimization. The number of spectral components in the sixth column is the number of RTL vectors rounded off to the nearest power of 2.

The last two columns of Table 6.2 show the fault coverages of the RTL vectors. RTL Coverage is the coverage of just the RTL faults and gate-level coverage is the coverage of all stuck-at faults in the implementation. As expected, the gate-level coverage is lower than the normal requirement of being close to 100%. We will use our spectral technique

Table 6.3: Comparison of RTL ATPG and Sequential gate-level ATPG results.

Circuit name	No. of gate-level faults	RTL ATPG – spectral tests			Gate-level ATPG			Random inputs	
		Cov. %	No. of vectors	CPU secs	Cov. %	No. of vectors	CPU secs	No. of vectors	Cov. %
b01-A	228	99.57	128	19	99.77	75	1	640	97.78
b01-D	290	98.77	128	19	99.77	91	1	640	95.80
b09-A	882	84.68	640	730	84.56	436	384	3840	11.71
b09-D	1048	84.21	768	815	78.82	555	575	7680	6.09
b11-A	2380	88.84	768	737	84.62	468	1866	3840	45.29
b11-D	3070	89.25	1024	987	86.16	365	3076	3840	41.42
b14-A	25894	85.09	6656	5436	68.78	500	6574	12800	74.61
s1488	4184	95.65	512	103	98.42	470	131	1600	67.47
s5378	15584	76.49	2432	2088	76.79	835	4439	3840	67.10
s5378*	15944	73.59	1399	718	73.31	332	22567	2880	62.77
s9234	28976	17.36	64	721	20.14	6967	18241	160	15.44
s9234*	29400	49.47	832	2734	48.74	12365	4119	2176	33.06
s35932	103204	95.70	256	1801	95.99	744	3192	320	50.70

to enhance this coverage. The low coverage of the RTL faults, however, may indicate a testability problem, which could limit our ability to increase the coverage either by the spectral technique or by gate-level ATPG.

Table 6.3 gives a comparison of the proposed RTL ATPG method with gate-level sequential ATPG. The first two columns give the circuit name and the number of gate level single stuck-at faults. The performances of RTL ATPG–spectral tests, gate-level ATPG, and random vectors can be compared with respect to the test coverage (Cov %), number of test vectors and test generation time.

For RTL ATPG, the number of vectors is the total number vectors in the compacted test sequences. Consider the circuit b01-A. In Table 6.2, there are 38 RTL vectors that provide 64, i.e., 38 rounded to next power of 2, spectral components. Ten sequences, each of 64 vectors, were generated by the perturbation technique of Subsection 6.1.2. The fault simulator of FlexTest [91] provided a gate-level test coverage of 99.57% for the 228 stuck-at faults shown in column 2 of Table 6.3. Note that the test coverage of FlexTest is an upward adjusted coverage, accounting for faults that are found to be untestable. The ILP compaction technique of Subsection 6.1.2 selected two sequences, reducing the test length to 128 vectors. The test generation time in column 5 includes the times for RTL

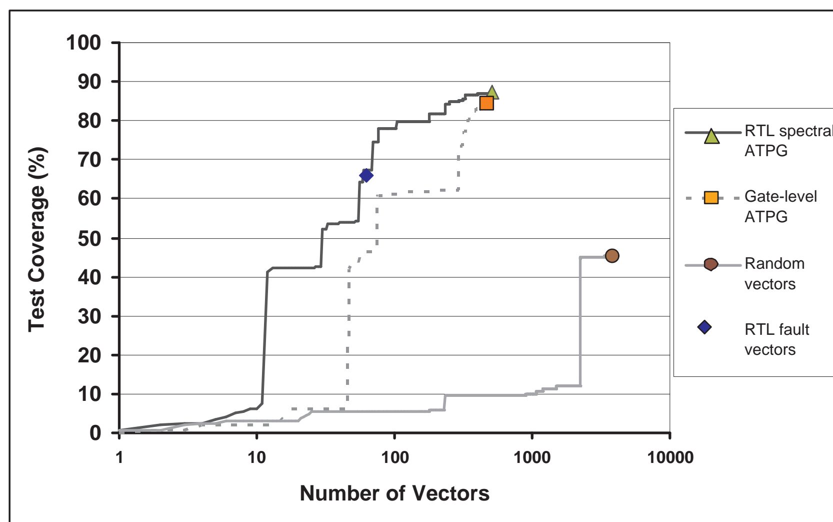


Figure 6.6: Test coverage of RTL ATPG (spectral vectors) for area optimized b11-A circuit.

characterization (from Table 6.2), perturbed spectral sequence generation, fault simulation, and ILP compaction. Of these, RTL characterization and fault simulation are the dominant components, the other two being negligible.

Next, for the circuit b01-A, we find that FlexTest generates 75 vectors for a 99.77% test coverage and 640 random vectors (same number of vectors as in ten spectral sequences of 64 vectors each) have 97.78% test coverage. As we move down in Table 6.3, circuits become larger and we observe that RTL ATPG provides about the same test coverage with slightly larger vector lengths as the gate-level ATPG, but its time increases more slowly. Moreover, the RTL faults used for circuit characterization and vector generation are implementation independent. Notably, the test coverage of random vectors tends to drop as circuits become larger.

As an example, Figure 6.6 compares the graphs of test coverage percentage against the number of vectors for ISCAS'89 benchmark circuit b11-A, for RTL spectral ATPG vectors, gate-level ATPG vectors and random vectors. The gate-level coverages of RTL vectors (generated to cover RTL faults only) are also shown by a point in each graph. For these

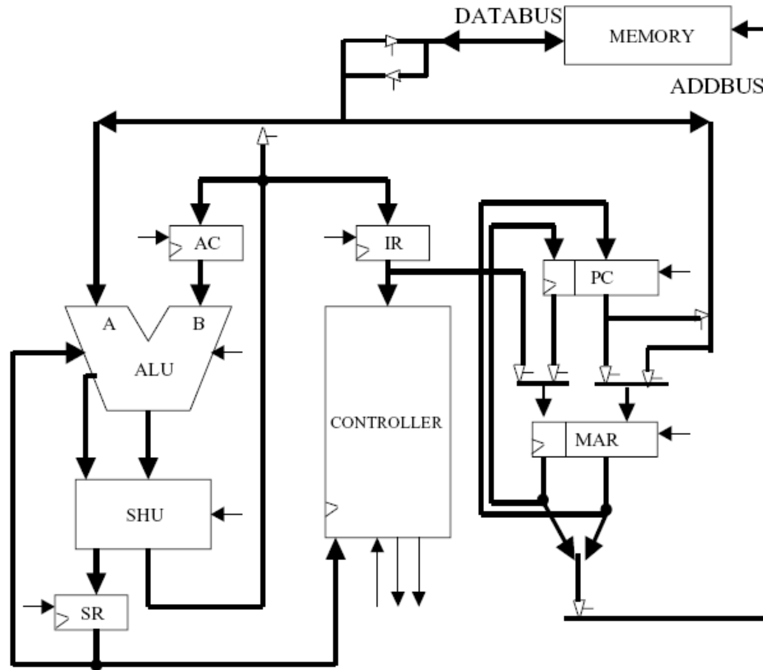


Figure 6.7: PARWAN CPU [97].

circuits, the coverages of RTL ATPG are about 2 to 4% higher, vector lengths about double and CPU times about 30 to 50% when compared to the gate-level ATPG.

6.3.2 Results for PARWAN processor

We applied our RTL-spectral based test method to a simple accumulator-based processor named PARWAN [97] to demonstrate our method's effectiveness. As shown in the schematic diagram of PARWAN processor in Figure 6.7, it includes the following components: accumulator (AC), arithmetic logic unit (ALU), shifter unit (SHU), status register (SR), instruction register (IR), program counter (PC), memory address register (MAR) and a control unit (CTRL). It has an 8-bit data-bus with a 12-bit address bus (addressing 4K memory). The circuit has around 800 gate modules and 53 flip-flops. Currently our method cannot handle bidirectional pins. Hence we had to split the bidirectional buses into separate input and output buses. Also we added a reset signal to initialize the circuit for testing.

The gate-level netlist for PARWAN processor was downloaded from an internet website [1]. As mentioned earlier, the bidirectional data-bus was separated into input and output buses and a global reset input was added. We sampled a total of 737 RTL faults, which were all the faults on the inputs-outputs of the different components (for example, ALU, SHU, etc.), inputs-outputs of the registers (for example, IR, PC, etc.) and the faults on the tri-state drivers. Vectors were generated to cover these 737 RTL faults using the commercial sequential ATPG tool FlexTest [91] and new vectors were generated using the technique described in Section 6.1. We applied our method on PARWAN processor using two types of fault models; stuck-at and transition delay. Their respective results are described in separate sections below.

Results for stuck-at fault model

In the absence of a RTL/behavioral test generator, the test vectors for RTL faults were obtained using the Mentor Graphics tool FlexTest [91] which is a gate-level sequential ATPG system with a built in fault simulator. For the stuck-at fault model, 134 vectors were generated to cover the RTL faults. These vectors were analyzed for their spectrum and new vector sequences were generated and compacted using the technique discussed in Section 6.1.2. These results were obtained on Sun Ultra 5 machines with 256MB RAM. Table 6.4 shows the characteristics of RTL test vectors. Columns (left to right) give number of RTL faults, number of vectors generated, test generation CPU seconds, number of spectral coefficients derived, RTL fault coverage and gate-level fault coverage of RTL vectors. Faults in the clock network were not included and a 50% credit was given to potentially testable faults [113]. Coverages are defined as ratios of the number of detected faults to the total number of detectable faults as reported by FlexTest. The last column gives the fault coverage of the 134 RTL vectors for collapsed gate-level faults in the entire circuit, excluding clock faults.

As discussed in Section 6.2, the RTL test generation reveals bottlenecks in the testability of the circuit. To detect the faults in tri-state bus drivers we modeled the buses with

Table 6.4: Spectral characterization of processor circuit by RTL vectors for stuck-at faults.

No. of RTL faults	No. of vectors	CPU s	No. of spectral coefficients	RTL fault coverage	Gate-level Coverage
737	134	640	128	96.30%	81.22%

Table 6.5: Spectral RTL ATPG for stuck-at faults for processor circuits.

Circuit	Collapsed gate-level faults	RTL spectral ATPG			Gate-level ATPG			Random vectors	
		Cov. %	No. of vectors	CPU s	Cov. %	No. of vectors	CPU s	Cov. %	No. of vectors
Parwan (original)	2270	98.23	2442	2327	93.40	1403	26430	80.95	2814
Parwan (with DFT)	2320	98.77	2442	1966	95.78	1619	20408	87.09	2948

memory using the high-impedance signal state. An analysis of the undetected RTL faults revealed 10 faults classified as unobservable by FlexTest. The remaining faults were either untestable or potentially tested. We selected the 10 unobservable fault sites as observation points and inserted a tree of nine XOR gates whose output was made into an added primary output. Thus, a DFT version of the processor was created. All RTL faults were now either detected or potentially detected by the same 134 vectors. However, the gate-level fault coverage of the RTL test vectors showed only a moderate increase to 83.61% from 81.22% (indicated in the last column of Table 6.4).

Table 6.5 gives the results of the RTL spectral ATPG method. Column 2 gives the number of collapsed faults in the circuits. Results for our proposed RTL spectral ATPG, Gate-level ATPG and Random vectors can be compared in terms of test coverage and number of vectors. Additionally test generation times for our proposed method and the gate-level ATPG method can be compared. As can be observed from the table, our proposed method achieves better test coverage as compared to the gate-level sequential ATPG in lower test generation time. The coverage is slightly lower than 100% because some faults were potentially testable and were given 50% credit. Since the RTL ATPG covered almost all faults in the original circuit, the benefit of DFT was small. However, the benefits of DFT were more for gate-level ATPG and random vectors. The test coverage plots of the original circuit and the circuit with DFT are shown in Figures 6.8 and 6.9.

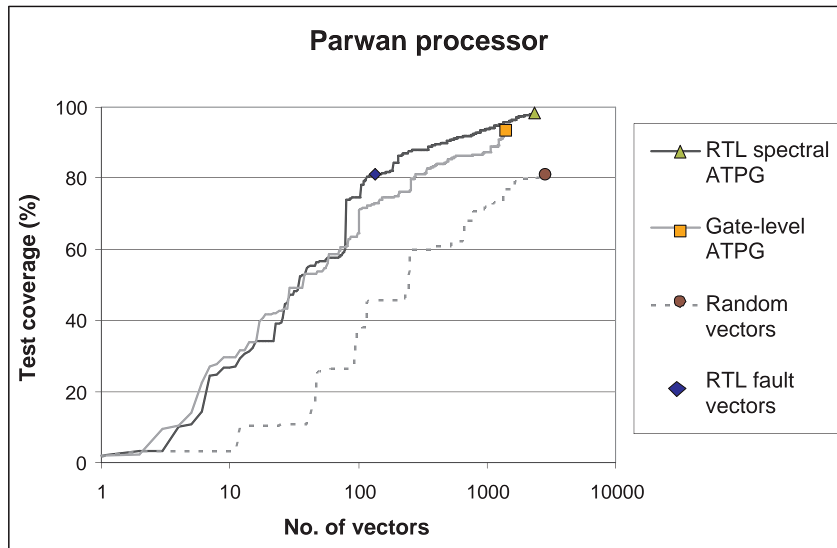


Figure 6.8: Test coverages for the original PARWAN circuit [97].

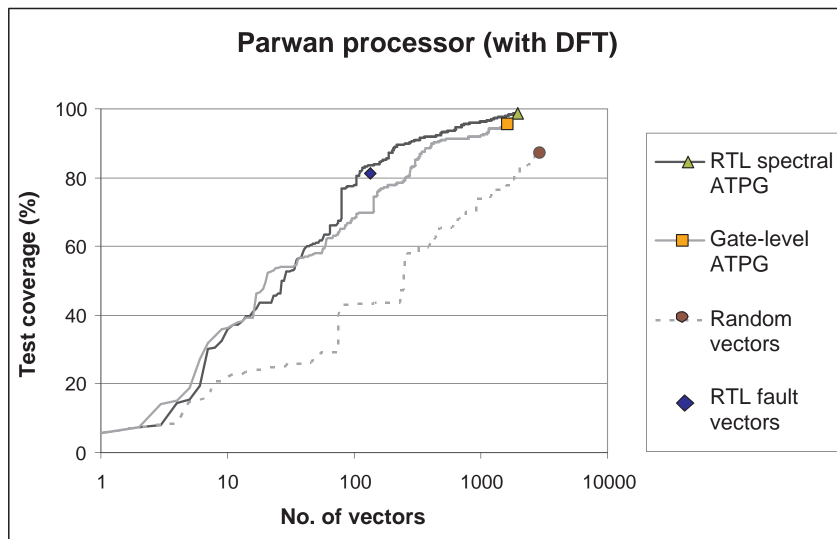


Figure 6.9: Test coverages for the PARWAN circuit with DFT.

To compare the effectiveness, we examined the gate-level faults that were left undetected by each method. For the original PARWAN, the gate-level ATPG left 129 undetected (127 unobserved and 2 uncontrolled) faults. Besides, six faults were potentially detected. When these 135 faults were simulated with 2442 spectral ATPG vectors (Table 6.5), 106 unobserved, 1 uncontrolled and 1 potentially detected faults were detected. The other 5 potentially detected faults remained potentially detected. For the original PARWAN, spectral ATPG had left 30 undetected (29 unobserved and 1 uncontrolled) faults. There were 5 potentially detected faults. These 35 faults were simulated with 1403 gate-level ATPG vectors shown in Table 6.5. Only 8 unobserved faults were detected and all 5 potentially detected faults remained potentially detected.

For the DFT version of PARWAN, gate-level ATPG left 82 undetected (78 unobserved and 4 uncontrolled) faults and there were 6 potentially detected faults. Of these, spectral ATPG vectors detected 62 unobserved and 3 uncontrolled faults. All 6 potentially detected faults remained the same way. Finally, in the reverse examination, 2442 spectral ATPG vectors had left 18 unobserved and 1 uncontrolled faults and had produced 6 potentially detected faults. The set of 1619 gate-level ATPG vectors only detected 2 of the unobserved faults. All 6 potentially detected faults remained the same way.

These comparisons between the fault detection of gate-level ATPG and our proposed spectral ATPG indicate that our proposed spectral ATPG method is able to generate better quality test vectors as compared to the gate-level ATPG method.

Results for transition delay fault model

Similar to the stuck-at RTL faults, 737 transition delay RTL faults were sampled and 160 transition delay RTL test vectors were generated using the Mentor Graphics tool FlexTest [91] . Table 6.6 shows the characteristics of transition delay RTL test vectors. Transition delay faults in the clock and reset network were not included and a 50% credit was given to potentially testable faults [113]. Columns from left to right specify the number of RTL faults sampled, number of RTL test vectors generated, CPU test generation time in

Table 6.6: Spectral characterization of processor circuit by RTL vectors for transition delay faults.

No. of RTL faults	No. of vectors	CPU s	No. of spectral coefficients	RTL fault coverage	Gate-level Coverage
737	160	3652	128	77.07%	47.84%

seconds, number of spectral coefficients used, the RTL fault coverage and the fault coverage of the RTL vectors for collapsed gate-level transition faults in the entire circuit excluding clock and reset faults. Coverages are defined as ratios of the number of detected faults to the total number of detectable faults as reported by FlexTest.

Analysis of the RTL undetected transition delay faults for PARWAN processor revealed around 24 unobservable faults. The remaining undetected transition faults were untestable or potentially testable. The circuit used tri-state drivers for driving buses and to detect these faults we modeled the high-impedance state as the previous logic state to improve its testability. It was observed that almost all of the undetected RTL faults were on the terminals of tri-state drivers. This could be because of the functional constraints of the processor and such faults could be functionally redundant delay faults. We selected the 24 unobservable fault sites as our observation points and condensed them using a 23 XOR gates tree and fed its output to an extra added output pin. All RTL transition delay faults were now either detected or potentially detected by the same 160 vectors. As a result of DFT, the test coverage of the RTL vectors increased to 65.82%.

The final results are tabulated in Table 6.7, which gives a comparison of the proposed RTL ATPG method with gate-level sequential ATPG and random vectors. Results are compared for two circuits; the original PARWAN circuit and PARWAN circuit with added DFT. The RTL ATPG and gate-level ATPG results can be compared based on test coverage of transition delay faults, number of vectors generated and the test generation CPU time. Table 6.8 gives the corresponding stuck-at fault coverages of the transition delay test vectors. As observed in the tables, we achieve better test coverage as compared to the gate-level sequential ATPG in lower test generation time for both stuck-at and transition delay faults. Also the stuck-at fault coverages obtained are close to what we achieved by performing

Table 6.7: Spectral RTL ATPG for transition delay faults for processor circuits.

Circuit	Collapsed gate-level faults	RTL spectral ATPG			Gate-level ATPG			Random vectors	
		Cov. %	No. of vectors	CPU s	Cov. %	No. of vectors	CPU s	Cov. %	No. of vectors
PARWAN (original)	3434	81.15	9120	6428	73.90	1318	43574	56.90	51200
PARWAN (with DFT)	3448	84.31	9120	6428	81.45	1444	40119	63.42	51200

Table 6.8: Stuck-at fault coverage of transition fault vectors.

Circuit	Collapsed gate-level faults	RTL spectral ATPG		Gate-level ATPG		Random vectors	
		Cov. %	No. of vectors	Cov. %	No. of vectors	Cov. %	No. of vectors
PARWAN (original)	2270	96.45	9120	91.41	1318	81.32	51200
PARWAN (with DFT)	2320	97.02	9120	95.27	1444	84.92	51200

spectral RTL-ATPG on stuck-at faults. Our proposed method however generates larger number of vectors as compared to the gate-level ATPG. In our method we employed a simple test set dropping heuristic to reduce the number of test sets. More intelligent and potent algorithms, which perform efficient test vector compaction for sequential circuits, but require higher computational times, have been proposed in literature [98, 107, 109, 58, 23]. These algorithms can be incorporated in our method to obtain much lower number of test vectors at the expense of an increase in test generation time.

Note that the highest achievable test coverage cannot be 100% as there was a small fraction of undetectable faults due to fault-induced uninitializability. These are possibly (or potentially) testable faults and were given a detection credit of 50%. We also observe an increase in test coverage when comparing the original circuit with the circuit with DFT. With added DFT, the sequential ATPG results have improved satisfactorily. Also random vectors perform better on circuit with DFT. The test coverage plots of the original circuit and the circuit with DFT are shown in Figures 6.10 and 6.11.

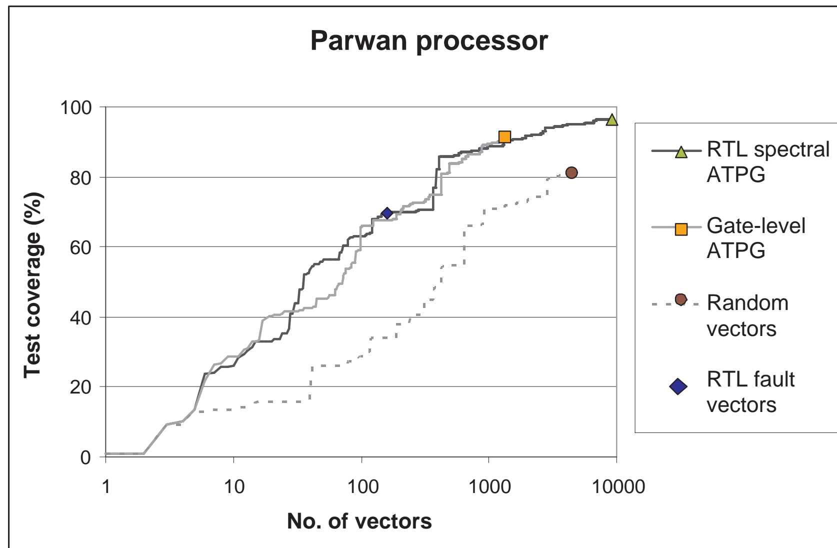


Figure 6.10: Test coverages for the original PARWAN circuit [97].

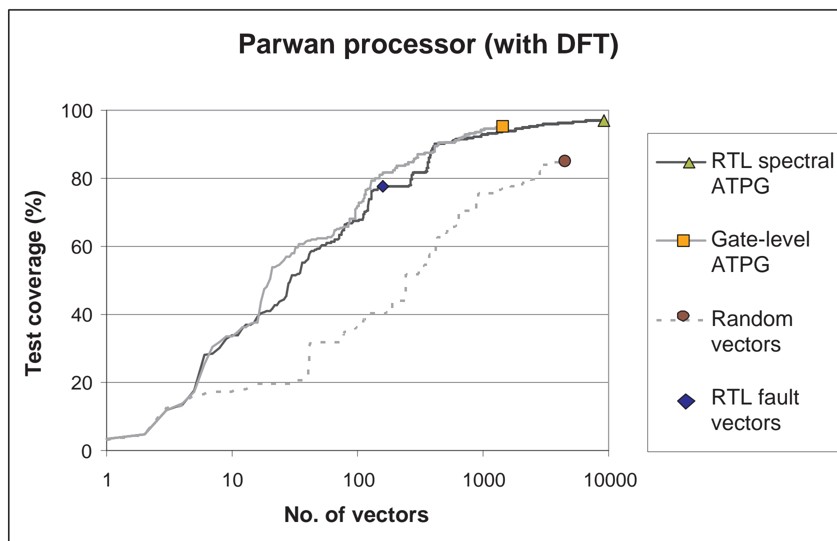


Figure 6.11: Test coverages for the PARWAN circuit with DFT.

6.4 Summary

We have presented a new method of RTL test generation using spectral techniques and demonstrated its effectiveness on benchmark circuits of ISCAS'89, ITC'99 and the experimental processor PARWAN. Test vectors generated for RTL faults are analyzed using Hadamard matrix to extract important features and new vectors are generated retaining those features. We observe improved test coverage and lower test generation time as compared to a sequential gate-level ATPG tool. Generation of different types of test sets and performing compaction on them is found to be an efficient and reliable method for test generation. Results show that as circuits become larger the RTL method may have advantages over gate-level ATPG. This reveals a promise in generation of test vectors at RTL by spectral analysis.

RTL test generation brings with it the advantages of lower memory, computation and generation time complexity. It enables the testability appraisal at RTL, and hence efforts can be made to improve testability when the design is conceptualized at higher levels of abstraction. An XOR observability tree designed at RTL improved both the stuck-at and transition delay fault coverages. Observation test points, selected after appraising the RTL ATPG results, effectively revealed the testability bottlenecks of the circuit. Further, RTL ATPG enables the testing of cores for whom only the functional information is known.

Chapter 7

N-Model tests

Several years ago the results of a Sematech study [99] were published. Four types of tests, namely, scan-based stuck-at, scan-based delay, I_{DDQ}, and functional, were examined. A general conclusion was that none of the tests could be dropped. That study has been a subject for numerous discussions [100, 101]. In [101] a study was conducted on an industrial circuit to compile pass/fail data for various test methods. The test methods used were scan-based stuck-at tests, scan-based delay tests, functional vectors and scan-based I_{DDQ} tests. Figure 7.1 shows the data presented in [101]. It gives the number of passing/failing chips for each type of test applied. As can be seen from the figure, no single test method is adequate in detecting all defects and accurately failing all defective chips. If any of the tests would have been a golden test, then a chip passing that test would be passing all other tests and failing none (assuming other tests do not cause a yield loss which is true in this case). (Two entries were left blank by the authors in [101] to hide yield data).

Another study similar to [101] is reported in [87] which summarized the pass/fail data for chips tested using three different tests: scan-based stuck-at, functional and I_{DDQ} tests. Figure 7.2 gives the corresponding pass/fail chips for the different test types [87]. Figure 7.3

		IDDQ PASS		IDDQ FAIL			
Scan-based stuck-at	**PASS**	blank	6	1463	7	**PASS**	**Scan-based delay**
		14	0	34	1	**FAIL**	
	FAIL	6	1	13	8	**PASS**	
		52	36	1251	blank	**FAIL**	
		PASS	**FAIL**	**PASS**	**FAIL**		
		Functional					

Figure 7.1: Number of passing/failing chips for four different test types applied [101].

		IDDQ			
		PASS		FAIL	
Scan-based stuck-at	PASS	22066	25	1358	36
	FAIL	19	134	122	2655
		PASS	FAIL	PASS	FAIL
		Functional			

Figure 7.2: Number of passing/failing chips for three different test types applied [87].

		IDDQ tests applied			
		YES		NO	
Scan-based stuck-at tests applied	YES	0	1132	57975	60421
	NO	860	8002	63611	164641
		YES	NO	YES	NO
		Functional tests applied			

Figure 7.3: Defect level in parts per million deduced from data in [87].

gives the defect levels in terms of defective Parts-Per-Million(ppm) for various combinations of the tests being applied with the assumption that the combined tests have 100% defect coverage. As can be seen from Figure 7.2, dropping any test has an effect on the defective ppm and hence no particular test can be dropped. However, we observe from the table that dropping stuck-at tests has the least impact on the defect level and dropping of the I_{DDQ} tests the most. The data in [87] was published in 1992 and over the years with reducing feature sizes and more complex defects arising in silicon wafers, the problem has exacerbated and the inferences from [87, 101] hold even more importance.

With advances in technology, new fault modes are continuously emerging and the gap between the age-old stuck-at fault model and "realistic defects" continues to widen. On the other hand, the need to minimize test length and test time has never been greater because of complex system-on-chip (SOC) devices. Our proposed work addresses the problem of combining tests that target several fault models into a single compact test.

Test minimization has been a widely researched area. However, most of the published methods will be difficult to apply to multiple fault models. Integer linear programming (ILP) is an effective method of test optimization. It gives global optimization and has been used for both combinational and sequential circuits [27, 148, 151] as well as for globally minimizing N-detect tests [70].

We develop ILP methods for multiple-fault models. Although applications of ILP have been reported for separately optimizing vectors for detecting stuck-at faults [27, 148, 151], N-detect stuck-at faults [69, 70], and transition faults [152], to our knowledge a simultaneous ILP optimization for multiple fault models has not been attempted before.

7.1 Overview

Different fault models address different types of realistic defects that can occur at the wafer level. For example, stuck-at faults model some types of transistor short defects and transition delay faults model certain transistor open and timing defects. Neither of the two is adequate in modeling each other's characteristics completely as was observed in one of our experiments. In our experiment, a comparison was performed between the transition delay fault coverage of stuck-at fault vectors which detected 100% of detectable stuck-at faults and stuck-at fault coverage of transition delay vectors which detected 100% of detectable transition delay faults for an ISCAS'89 benchmark circuit. In neither of the cases did we observe a coverage of 100%. A similar observation was been made in [152]. This characteristic holds true for other fault models too. This necessitates the inclusion of test vectors over different fault models to improve the defect coverage. An easy method of achieving this is to concatenate all the test sets together (as was being experimented in [152]), however, this rapidly increases the number of test vectors. Hence we need a method to find a minimized test set which caters to all the considered fault models.

In order to address this issue, we make use of the Linear Programming (LP) problem formulation. LP and its special case of Integer Linear Programming (ILP) are used

to minimize the test vectors for multiple fault models using the concepts as described in Chapter 4.

7.2 The *N*-model tests

Definition: For a set of N given fault models, $N \geq 1$, the N-model tests target detection of all faults in the superset of faults for all N fault models.

Typically, a set of fault models may include stuck-at, transition, path-delay, I_{DDQ}, bridging, coupling, etc. The tests may be a combination of functional, verification and random vectors, as well as those generated by targeting one or more fault models. The problem then is to reduce the test set size without affecting the coverage of the N specified fault models. Although an experimental verification is yet to come, we believe such tests will be more effective at uncovering real defects than those generated by the existing Automatic Test Pattern Generator (ATPG) strategies. Besides, any new fault model can be included with the basic prerequisite that we have a fault simulator for it.

The difficulty with multiple fault models is that a fault simulator can deal with only one model at a time. We therefore concatenate all tests and separately simulate the entire set for each fault model, one at a time without fault dropping. We thus obtain a fault dictionary, i.e., information about the faults detected by each test vector. Here we consider combinational and scan-inserted sequential circuits only. Hence the test sets are combinational in nature and their order of application is immaterial. Having obtained this fault dictionary, our aim is to select the least number of test vectors that cover all faults belonging to the considered fault models.

We formulate this problem as an integer linear programming model. In the following sections we introduce two ILP models for test set minimization. The first is called the “two-step ILP” model and the second the “combined ILP” model.

7.3 Two-step ILP model

Our goal is to minimize a test set for multiple fault models. Different fault models have different characteristics and procedures for testing. For example, if test vectors are generated for I_{DDQ} faults, just the application of the vectors is not enough, but an extra step of I_{DDQ} measurement needs to be carried out. Since this I_{DDQ} measurement is an expensive and time consuming test, it is not performed for all vectors, unlike the fault coverage measurement for stuck-at and transition delay test vectors. Hence the I_{DDQ} vectors and their corresponding measurements need to be given special attention. Based on this constraint, we propose a two-step ILP model. In the first step a global minimization for the total number of vectors is carried out for all the considered fault models and then in the second step the minimum number of I_{DDQ} measurements to be made is determined.

7.3.1 First ILP - minimize vectors

As described in the previous section we obtain the fault dictionary by concatenating all tests and then separately simulating the entire set for each fault model, one at a time without fault dropping. Using the fault dictionary information, for each fault (of any type), one constraint inequality is generated. Suppose, fault f_i is detected by test numbers j, m and q, then the corresponding constraint is,

$$t_j + t_m + t_q \geq 1 \tag{7.1}$$

where t_x is an integer variable for test number x that can take the value 0 (test number x is discarded) or 1 (test number x is selected). The above constraint means that at least one test is required to ensure the coverage of fault f_i. Because of this constraint, any test set solution that ILP provides will never drop any fault and the fault coverage is ensured. Notice that a test here can be a single vector (in our case for stuck-at faults and I_{DDQ} faults) or a vector pair (for transition delay faults).

A single constraint-like inequality (7.1) is generated for each fault in the fault set, which contains all faults corresponding to all fault models of interest. Our objective function for this problem is defined as follows:

$$\text{Minimize} \sum_{all\ vectors\ x} t_x \tag{7.2}$$

The ILP then finds a solution under the specified constraints (as shown in equation (7.1)) and meeting the objective function (shown in equation (7.2)). The solution consists of values assigned to variable t_x (x for all vectors) which can be either 0 (test number x is discarded) or 1 (test number x is selected). The test sets with $t_x = 1$ are minimal to guarantee detection of each modeled fault by at least one vector.

7.3.2 Second ILP - minimize I_{DDQ} measurements

Once vectors are minimized, a second ILP run is used to find the minimal set of these vectors on which I_{DDQ} should be measured. For this ILP run, the tests consist of only those selected by the first ILP run. All t_x's whose values were 0 in the first ILP run are eliminated. Constraints-like inequalities (7.1) are then specified only for the I_{DDQ} faults. The objective function is:

$$\text{Minimize} \sum_{selected\ vectors\ y} t_y \tag{7.3}$$

The non-zero t_y's in this ILP solution identify the minimal set of vectors for which I_{DDQ} needs to be measured.

7.4 Combined ILP model

Since I_{DDQ} measurement is expensive, it is necessary that their number be kept to a minimum, sometimes even at the expense of slight increase in the total number of vectors

to be applied. The "Two-step ILP" model does not give a globally minimized number of I_{DDQ} measurements as in it's first step itself it optimizes for the total number of vectors. To achieve the goal of reducing the I_{DDQ} measurements even further, we modify the ILP formulation of Section 7.3.

We represent each vector x in the initial vector set by two {0,1} integer variables, t_x and an I_{DDQ} variable i_x. The variable t_x, as before, signifies whether test number x will be selected in the final minimized test set (either as an I_{DDQ} test or a non-I_{DDQ} test). The variable i_x is used to signify whether an I_{DDQ} measurement is performed on test number x. Also this means that variable i_x needs to be a subset of variable t_x, i.e., if $i_x = 1$ then $t_x = 1$ but not necessarily vice versa.

The ILP constraints are generated as follows. If fault f_i is not an I_{DDQ} fault, then its constraint is exactly like inequality 7.1, i.e., no constraints are applied to the i variables of its tests. For an I_{DDQ} fault f_k, that is detectable by vectors u, v and w, the constraint inequalities are as follows:

$$t_u + t_v + t_w \geq 1 \tag{7.4}$$

$$i_u + i_v + i_w \geq 1 \tag{7.5}$$

$$t_u \geq i_u \tag{7.6}$$

$$t_v \geq i_v \tag{7.7}$$

$$t_w \geq i_w \tag{7.8}$$

The last three constraints mean that I_{DDQ} measurements may be conducted for some subset of all vectors with capability of detecting I_{DDQ} faults. The objective function is,

$$\text{Minimize} \sum_{all\ vectors\ x} (\ t_x + \ W \times i_x) \tag{7.9}$$

Here the weighting factor W specifies how strongly we wish to minimize the number of I_{DDQ} measurements. Setting $W = 0$ will lead to the result of Section 7.3.1, i.e., we will

minimize the vector count without any regard for I_{DDQ} measurements. A second ILP run with objective function(7.3) may give more than the smallest possible number of I_{DDQ} measurements. Choosing a larger value of W will minimize the number of I_{DDQ} measurements but may lead to slightly increased number of vectors. This trade off is evident in the results explained in Section 7.6.

Though the ILP gives an optimum solution, it's time complexity is exponential and can become a bottleneck. Our results in Section 7.6 exhibit this characteristic. Hence we need to adopt a technique which will simplify the computation. A linear programming (LP) model is often used in place of integer linear programming (ILP) model to reduce the CPU time and obtain a result that may still be quite close to the optimum. The worst-case complexity of LP is polynomial-time while that of ILP is exponential. We have devised an "hybrid LP-ILP" method, outlined in Section 7.5, which is a variation of the published recursive-LP method for test minimization [71].

7.5 Hybrid LP-ILP method

Observing the large time-complexity for some ILP runs, we developed an hybrid LP-ILP method. This method solves any ILP problem and can be effectively used for the models introduced in Sections 7.3 and 7.4. In spite of some similarities with other methods [71], the hybrid LP-ILP differs from previous work.

Once formulated as an ILP, an LP model is obtained by changing the variables to real numbers in the same [0,1] range. The LP is solved and variables below 0.1 are then rounded to 0 and variables above 0.9 are rounded to 1. The rounded variables are now treated as constants, constraints are updated and the LP solution and rounding processes are repeated until no more variables can be rounded either to a 0 or a 1. Then a much reduced ILP is solved. The results in Section 7.6 reveal the reduced time complexity of this method.

Since we round variables on both sides, i.e., to 0 and to 1, the procedure can lead to an intermediate solution that may not satisfy all remaining constraints. In such a case, the problem can be solved by the previously proposed recursive-LP algorithm [71], which

Table 7.1: Test vectors for stuck-at, I_{DDQ} and transition faults generated and minimized by FastScan.

Circuit	Type of vectors	Number of vectors		Fault coverage (%)
		Unminimized	Minimized	
(1)	(2)	(3)	(4)	(5)
c1908	stuck-at	66	46	93.93
	I_{DDQ}	37	26	98.19
	transition	113	75	91.80
	total	216	147	-
c3540	stuck-at	167	130	96.00
	I_{DDQ}	53	45	99.09
	transition	299	229	96.55
	total	519	404	-
s1238	stuck-at	178	149	95.17
	I_{DDQ}	70	58	93.04
	LOS	293	234	96.75
	LOC	296	242	95.97
	total	837	683	-
s1488	stuck-at	136	114	100.00
	I_{DDQ}	55	45	98.20
	LOS	166	126	67.92
	LOC	200	166	91.57
	total	557	451	-
s5378	stuck-at	150	145	99.30
	I_{DDQ}	71	70	85.75
	LOS	319	293	98.31
	LOC	256	242	90.05
	total	796	750	-

LOS : Launch-on-Shift; LOC : Launch-on-Capture

guarantees a solution. We believe that, whenever a solution is possible, the hybrid LP-ILP will be faster than the recursive-LP.

7.6 Results

We applied our optimization techniques to two combinational and three scan inserted ISCAS'89 sequential benchmark circuits. We used three different fault models. They were stuck-at, transition and I_{DDQ} faults. We generated initial test vectors for each fault model

using the commercial ATPG tool, Mentor Graphics FastScan [91]. For I_{DDQ} faults, the pseudo stuck-at model [87] was used because it is supported in FastScan.

Table 7.1 gives the details of the test vectors generated for the circuits. Column (1) gives the circuit name. Column (2) specifies the fault model for which test vectors in column (3) were generated. Stuck-at and I_{DDQ} vectors were single pattern tests and transition delay vectors were two-pattern tests. Each of those vector-pairs was considered a single entity in the ILP model. For combinational circuits, the transition delay tests were simply two-pattern tests applied one after another, however, for scan-inserted sequential circuits we obtained Launch-On-Shift (LOS) and Launch-On-Capture (LOC) patterns [81, 144]. Column (3) gives the number of vectors that FastScan generated for each type of test vector set and column (4) gives the number of vectors obtained after minimization without reducing the fault coverage using tools provided in FastScan. Column (5) gives the fault coverage of each test set for the corresponding fault model. The numbers in the "total" rows are the sum of the three previous rows. Vectors can be further minimized by determining the stuck-at faults detected by transition vectors and eliminating the corresponding stuck-at vectors. However this was not performed to keep the discussion generic and applicable to any set of possibly unrelated fault models.

We used the test vectors generated by FastScan (reported in column (3) of Table 7.1) as our initial test set. We simulated the combined vector set of the three fault models without fault dropping and obtained the fault dictionary information giving the complete fault detection data for all vectors. This fault dictionary was then used in the formulations of ILP as mentioned in previous sections.

For our experiments we used the Sun Sparc Ultra-10 machine with 4.0 GB of RAM shared among four CPUs. We used AMPL-CPLEX package for ILP [35]. While executing AMPL we specified a CPU time limit of 5000 seconds. The results for various circuits and their vector sets are tabulated in Tables 7.2 and 7.3. Table 7.2 gives the results of minimization using the two step ILP method described in Section 7.3 and Table 7.3 gives the results using the combined ILP method proposed in Section 7.4. For comparison,

Table 7.2: Multiple fault model test optimization by ILP methods using two-step model.

Circuit	Type of vecs.	FastScan tests: Original no. of vectors	FastScan tests: Optimized no. of vectors	Two-step model: No. of vect.	Two-step model: CPU sec.
(1)	(2)	(3)	(4)	(5)	(6)
c1908	All	216	147	79	8.97
	I_{DDQ}	37	26	24	1.50
c3540	All	519	404	225	41.35
	I_{DDQ}	53	45	39	694.41
s1238	All	837	683	203	20.59
	I_{DDQ}	70	58	51	1.78
s1488	All	557	451	175	17.11
	I_{DDQ}	55	45	40	2.37
s5378	All	796	750	320	147.75
	I_{DDQ}	71	70	84	14.21

Table 7.3: Multiple fault model test optimization by ILP methods using combined model.

Circuit	Type of vecs.	FastScan tests: Original No. of vectors	FastScan tests: Optimized CPU vectors	Combined model $W = 0.1$: No. of vect.	$W = 0.1$: CPU sec.	$W = 1$: No. of vect.	$W = 1$: CPU sec.	$W = 10$: No. of vect.	$W = 10$: CPU sec.
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
c1908	All	216	147	79	9.3	81	9.4	84	9.9
	I_{DDQ}	37	26	24		22		20	
c3540	All	519	404	225	5044.2*	226	5046.7*	247	5046.9*
	I_{DDQ}	53	45	40		41		37	
s1238	All	837	683	203	97.1	205	63.5	212	243.1
	I_{DDQ}	70	58	45		43		40	
s1488	All	557	451	175	76.0	176	254.0	187	694.9
	I_{DDQ}	55	45	39		38		33	
s5378	All	796	750	320	2313.9	326	5154.2*	353	5160.9*
	I_{DDQ}	71	70	78		73		64	

* Incomplete optimization; CPU time limit of 5000 seconds exceeded.

Table 7.4: Multiple fault model test optimization by hybrid LP-ILP method using two-step model.

Circuit	Type	FastScan tests		Two-step model	
		Original	Optimized		
	of	no. of	no. of	No. of	CPU
	vecs.	vectors	vectors	vect.	sec.
(1)	(2)	(3)	(4)	(5)	(6)
c1908	All	216	147	79	17.50
	I_{DDQ}	37	26	24	2.32
c3540	All	519	404	225	78.13
	I_{DDQ}	53	45	40	43.92
s1238	All	837	683	203	37.91
	I_{DDQ}	70	58	51	3.4
s1488	All	557	451	175	33.91
	I_{DDQ}	55	45	40	5.51
s5378	All	796	750	320	306.43
	I_{DDQ}	71	70	84	32.63

the results of single fault model minimization from columns (3) and (4) of Table 7.1 are reproduced in columns (3) and (4) of both Tables 7.2 and 7.3. Vectors for each circuit are shown in two rows. FastScan separately generated vectors for each fault model. I_{DDQ} vectors and "All" vectors (including I_{DDQ} vectors) are shown in column (3) of both tables. Column (4) shows the sum of numbers of vectors obtained by using the minimization tools of FastScan separately for each vector set with its corresponding fault model.

In Table 7.2, columns (5) and (6) show the results of the two-step ILP model of Section 7.3 in terms of the number of vectors and the CPU time. This shows a reduction in I_{DDQ} measurements but as pointed out before, further reduction is possible. In Table 7.3, Columns (5) through (10) show the results of the combined ILP model of Section 7.4, with increasing emphasis on I_{DDQ} by setting $W = 0.1, 1$ and 10, respectively, in the objective function 7.9. The trade off between the number of I_{DDQ} measurements and test length is clearly observed.

Reducing Run Times: The results of minimization by the hybrid LP-ILP method are given in Tables 7.4 and 7.5. The columns of the table have the same meaning as

Table 7.5: Multiple fault model test optimization by hybrid LP-ILP method using combined model.

Circuit	Type of vecs.	FastScan tests		Combined model					
		Original No. of vectors	Optimized CPU vectors	$W = 0.1$		$W = 1$		$W = 10$	
				No. of vect.	CPU sec.	No. of vect.	CPU sec.	No. of vect.	CPU sec.
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
c1908	All	216	147	79	17.88	81	23.74	84	23.23
	I_{DDQ}	37	26	24		23		20	
c3540	All	519	404	225	166.6	226	188.6	248	516.23
	I_{DDQ}	53	45	41		39		34	
s1238	All	837	683	203	48.36	203	61.64	215	63.12
	I_{DDQ}	70	58	46		46		40	
s1488	All	557	451	175	38.33	176	50.33	187	59.99
	I_{DDQ}	55	45	39		39		34	
s5378	All	796	750	320	528.89	326	617.14	353	793.22
	I_{DDQ}	71	70	80		72		63	

Table 7.6: Comparing solutions: hybrid LP-ILP lower bound, ILP optimum and hybrid LP-ILP.

Circuit	Weight W	minimized (vectors + $W \times I_{DDQ}$ measurements)		
		hybrid LP-ILP lower bound	ILP solution	hybrid LP-ILP solution
c1908	0.1	81.4	81.4	81.4
	1	102.25	103	104
	10	276.5	284	284
c3540	0.1	227.94	229*	229.1
	1	257.82	267*	265
	10	499.97	617*	588
s1238	0.1	207.47	207.5	207.6
	1	247.64	248	249
	10	606.04	612	615
s1488	0.1	178	178.9	178.9
	1	211.74	214	215
	10	506.89	517	527
s5378	0.1	326.76	327.8	328
	1	392.28	399*	398
	10	910.68	993*	983

* Incomplete optimization; CPU time limit of 5000 seconds exceeded.

those in Tables 7.2 and 7.3, respectively. The number of minimized vectors and the CPU time complexity can easily be compared between these tables. For the two-step model, the pure ILP method performed slightly better in terms of both the number of minimized vectors and the CPU time, as compared to the hybrid LP-ILP method. However for the combined model, the hybrid LP-ILP method achieves close to optimum solution in terms of the number of minimized vectors, with an order of magnitude reduction in CPU time as compared to the pure ILP method. In cases of circuits like c3540 and s5378 where ILP quit early because of exceeding the CPU time limit, the hybrid LP-ILP method gave better results. For some circuits like s1238 and s1488, the hybrid LP-ILP method gave a close to optimum solution, if not the optimum.

When the hybrid LP-ILP method is used, the optimality of the solution achieved can be judged by examining the value of the objective function of equation 7.9 obtained in the first run of the LP. This value (usually rounded to the next higher integer) gives a lower bound on the absolute optimum solution as would be given by the ILP. Table 7.6 gives a comparison between this lower bound number, pure ILP method and hybrid LP-ILP method for the combined model. Column (1) gives the circuit name and column (2) gives the different cases of weights in the combined model. Columns (3) through (5) give the lower bound and the results of the exact ILP and hybrid LP-ILP methods. In most of the cases we observe that the value of the result obtained by the hybrid LP-ILP method either equals or is close to the exact optimum obtained by pure LP and this fact is indicated by the closeness of the hybrid LP-ILP solution to the lower bound. The pure ILP gives non-optimum solution in some cases where it had to quit due to CPU time limit. In those cases the hybrid LP-ILP method performs better.

7.7 Summary

The results show an effective reduction in both the total number of vectors and the number of I_{DDQ} measurements required as compared to the original test set. For test set minimization with multiple fault models, the problem arises because fault simulators lack

the capability of simultaneously simulating more than one type of fault. The ILP technique allows us to combine the results of separate fault simulations. Besides, ILP gives a global optimization of the test set. Our proposed ILP models provide useful trade offs by varying the emphasis between total test length and the number of I_{DDQ} measurements. Using a hybrid LP-ILP method to solve an ILP problem significantly reduces the time complexity, while achieving a close to optimum solution.

Chapter 8

Spectral Test Pattern Generation Hardware for BIST

Built-In Self Test (BIST) was briefly introduced in Section 2.5 and later explained in a little more detail in Chapter 5. We shall be concentrating on non-scan based implementation of BIST where the CUT is tested using only its inputs and outputs. Although non-scan based BIST has several advantages over scan-based BIST, we need to consider its two main challenges. First, a sequential Automatic Test Pattern Generator (ATPG) is required for generating vectors for non-scan circuits, whose test generation complexity is high [83]. The fault coverage achieved can be low, as some faults are either not detectable in the sequential mode or no tests are found due to backtrack and CPU time limits of the ATPG program. The second problem is that generating vectors for circuits in a BIST environment can be intricate because of existence of random pattern resistant faults. Especially in the case of sequential circuits, specific test sequences may be required to detect these faults and BIST circuits to produce these test sequences can be expensive in terms of area overhead.

Our proposed spectral BIST method addresses the second issue, which is test pattern generation for BIST. The goal of this work is to replicate the characteristics of the already obtained ATPG vectors so that the new generated vectors have at least the same efficacy as the ATPG vectors. The problem can be generalized to include any pre-existing set of vectors instead of ATPG vectors. In this work we consider ATPG vectors due to their high quality in terms of fault detection. For the problem at hand, instead of reproducing the exact ATPG sequence, we synthesize BIST to generate patterns with similar spectral characteristics. These BIST patterns are not the same as the ATPG patterns but excite somewhat similar behavior in the circuit. Sequences longer than the original ATPG sequences can even produce higher coverages.

Our BIST synthesis is based on the premise that the spectrum of vectors that detect faults in a circuit reflect important characteristics of the circuit. These characteristics may include spatial and temporal correlations among the bits of primary input vectors. Along with the relevant spectra, some amount of noise or randomness is present, which corresponds to uncorrelated bits in the tests generated for some target faults. The noise-like behavior of these bits allows detection of untargeted faults that require similar, but not exactly the same test sequences.

We propose a novel BIST synthesis method using Hadamard transform and spectral techniques [154, 155]. One may use any commercial or home-grown sequential ATPG tool, FlexTest [91] in our case, to generate test vectors for the CUT using the stuck-at fault model. The aim of the proposed BIST methodology is then to design hardware with minimum area overhead which recreates the essential spectral properties of ATPG vectors and attains its original fault coverage. The ATPG vectors are analyzed for their prominent spectral components using Hadamard transform. These prominent spectral components, which are generated by a Hadamard wave generator, are mixed in appropriate proportions and randomness if necessary is inserted appropriately. The major contribution of this work is a novel approach of spectral BIST which gives high fault coverage with a reduced hardware overhead as compared to existing published works. The proposed method, although it uses Hadamard transform for spectral analysis, is adaptable for other transforms like Haar transform. We also propose a novel hardware approach for combining spectral components.

8.1 Proposed spectral BIST method

Our proposed method of BIST synthesis consists of two steps. In the first step the ATPG vectors, which are required to be imitated in hardware, are analyzed using Hadamard transform for prominent spectral components and randomness (noise-like) content. Using this information, the spectral BIST is implemented in step two. The two steps are described in the following sections.

Input 1	Input 2	Input 3	Input 4	Input 5
0	1	0	1	1
0	0	0	0	0
1	0	1	0	1
1	0	1	0	0
0	1	0	1	1
1	1	0	1	1
0	0	0	0	1
1	1	0	1	1
1	0	0	0	0
1	0	0	1	1
1	1	0	1	1
0	1	0	0	0
0	X	1	X	0
0	X	1	X	0
X	X	1	X	0
X	X	1	X	0
X	X	1	X	X
X	X	1	X	X
X	X	1	X	X
X	X	1	X	X

Original test vector set (first 12 rows)

Appended test vector set (last 8 rows)

Figure 8.1: Appending of extra vectors to balance the weighting of bit-streams to 0.5.

8.1.1 Determination of spectral components and noise

As mentioned earlier, a spectral analysis is performed on the ATPG vectors using Hadamard transform [150] to determine their prominent spectral components and noise-like content. Certain pre-processing can be performed on the ATPG vectors before or during spectral analysis which would help in enhancing the prominent spectral components and reduce the noise-like content. This is especially relevant for test vectors generated for combinational circuits, as in this case the order of application of test vectors is irrelevant and the test vectors can be reshuffled. This type of pre-processing can also be performed for test vectors for sequential circuits but with limited efficacy as it is essential to maintain the order of application of the test vectors. Hence here we shall address and discuss the pre-processing of test vectors for combinational circuits only.

Pre-processing of test vectors for combinational circuits

Since the order of the vectors for combinational circuits is immaterial, vectors can be reshuffled and inherent spectral properties can be amplified and extracted, by reducing the ambiguous noise-like content. Another technique that we found useful in extracting the spectral components unambiguously is to append the original vector set with vectors such that the weighting (proportion of logic '1' values) of individual bit-streams entering the different inputs is balanced to 0.5, i.e., equally probable logic '0' and logic '1'. Don't care bits are used when the weighting is already 0.5. Figure 8.1 shows an example of appending extra vectors to the original test vector set to make the weighting of individual bit-streams for each input balanced to 0.5. Don't care bits (Xs) are used when weighting is balanced to 0.5.

Reshuffling of test vectors is performed such that certain spectral components are amplified and noise is reduced. Bit-streams entering various inputs of the CUT are examined, separately. The 0s and 1s in a bit-stream are represented as -1s and $+1$s, respectively. The don't care bits (Xs) in the vector set are represented as 0s. The reshuffling algorithm rearranges the test vectors such that the magnitude of the selected spectral components for the bit-stream of each input is enhanced. Our proposed algorithm reshuffles the test vectors considering one input bit-stream at a time. Initially the spectral components of the first input bit-stream are determined and the prominent spectral component with the maximum magnitude is selected. Test vectors are reshuffled such that maximum bits of the considered input bit-stream match with the selected prominent spectral component. After processing the first input bit-stream, the second input bit-stream is considered and so on. The reshuffling algorithm is described as Algorithm 1.

Input:

Test Vector Set (T) with N_V rows of vectors and N_I columns of inputs;

Hadamard transform matrix(H) of dimension $2^h \times 2^h$;

Output:

Sorted Test Vector Set (T);

Parameters:

N_V : Number of test vectors in the test vector set T;

N_I : Number of inputs of the circuit being driven by the test vector set T;

h : Order of Hadamard matrix;

$hd = 2^h$: Dimension of Hadamard matrix;

Variables:

SelWalshFns : $N_V \times N_I$ matrix storing the Walsh function selected for each input

Algorithm:

Append don't care vectors to test set T such that $N_V \bmod hd = 0$

Construct a larger Hadamard matrix (H_V) of dimensions $hd \times N_V$ from the original Hadamard matrix (H) by copying it repetitively along the columns.

for i *= 1 to* N_I **do**

$S = H_V \times T(:, i)$;

$(MaxIndex, MaxValue) = max(|S|)$;

$SelWalshFns(:, i) = transpose(sign(S(MaxIndex)) \times H_V(MaxIndex, :))$;

Swap test vectors such that the number of mis-matching bits between the matrices $SelWalshFns(:, 1 : i)$ and $T(:, 1 : i)$ is minimized;

end

Algorithm 1: Reshuffling algorithm for enhancing spectral components in test vectors for combinational vectors.

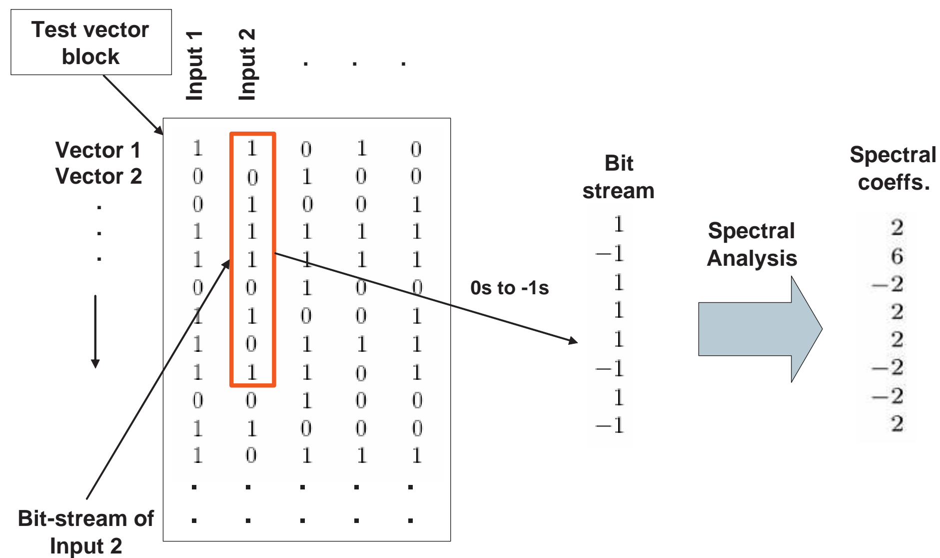


Figure 8.2: Spectral analysis of test vectors.

Spectral analysis of test vectors

In this step, the crucial process of determining the spectral components and noise is performed by spectral analysis of the test vectors. Similar to the spectral analysis procedure used in our spectral RTL-ATPG technique described in Section 6.1.1, the bit-streams entering various inputs of the CUT are analyzed separately. Figure 8.2 shows a schematic diagram of the process. An ATPG test vector block, generated to cover the stuck-at faults, will consist of test vectors as the rows in the block and the inputs of the circuit, to which the bits are being applied, as its columns. We perform a spectral analysis on the bits which are being applied to each of the inputs separately. As an example shown in Figure 8.2, an spectral analysis is performed on the bits which are being applied to the input 2 of the circuit. As described in Section 3.2, a Hadamard matrix of order n with a dimension equal to 2^n is used to analyze the binary bits. The number of bits that can be analyzed at a time is equal to the dimension 2^n of the Hadamard matrix used, 8 in case of the example illustrated. During each analysis step, the 0s and 1s in a bit-stream are represented as -1s and $+1$s, respectively. To find the spectral components for a bit-stream, it is multiplied with the Hadamard matrix. The corresponding result gives the spectral components.

The number of bits that can be analyzed at a time is restricted by the dimension 2^n of the Hadamard matrix. The number of bits that need to be analyzed for each input (equal to the number of test vectors) would generally be much higher than the dimension of the Hadamard matrix due to computational limitations. Hence the ATPG vectors are analyzed in sets, each of length $N = 2^n$, where N is an appropriate dimension chosen for the Hadamard matrix. Later we shall discuss the selection of the value for N. The analysis of ATPG vectors can be performed either on discrete sets of vectors or the sets can be overlapped. Overlapping of sets helps in sampling the vectors at closer spaced intervals. The overlapping length (O_V) has to be chosen appropriately. Analyzing vectors with low overlapping may lead to loss of information, while a high overlapping may lead to sampling the noise in the spectrum. We choose a value of O_V equal to $2^n/4$ for optimum results. The spectral analysis of a set i of length N for a primary input j of the CUT provides an original component spectrum denoted by C_{ij} and a power spectrum (which is the square of C_{ij}) denoted by P_{ij}. After the spectral analysis of all the sets, all C_i spectra and the P_i spectra are averaged for each input j, separately, to obtain the averaged component spectrum C_j and averaged power spectrum P_j respectively for all inputs j. We then perform a threshold filtering on the spectra P_j and C_j using threshold values of T_H and $\sqrt{T_H}$ respectively. The threshold value T_H, like the overlapping length O_V needs to be chosen appropriately. A high value of T_H will lead to information loss; while a low value will be ineffective in removing noise. We use a value of T_H equal to 0.5 times the average power of the resultant spectrum.

The averaged power spectrum P_j gives the prominent spectral components in the test vectors for the different inputs j of the CUT, while the averaged component spectrum C_j gives the sign or phase of those components. From the P_j spectrum we choose the top M prominent components for each input j based on their magnitude. Their sign is determined by the C_j spectrum. For our experiment, we chose a value of $M = 4$. The M prominent components along with their power magnitudes and signs are then used for BIST implementation. Figure 8.3 illustrates the process of averaging and determination of prominent spectral components.

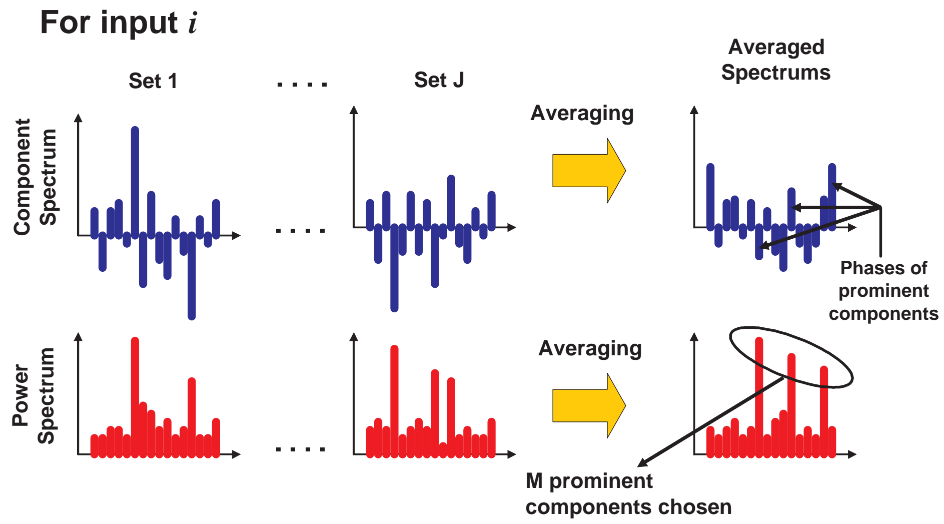


Figure 8.3: Determination of prominent spectral components.

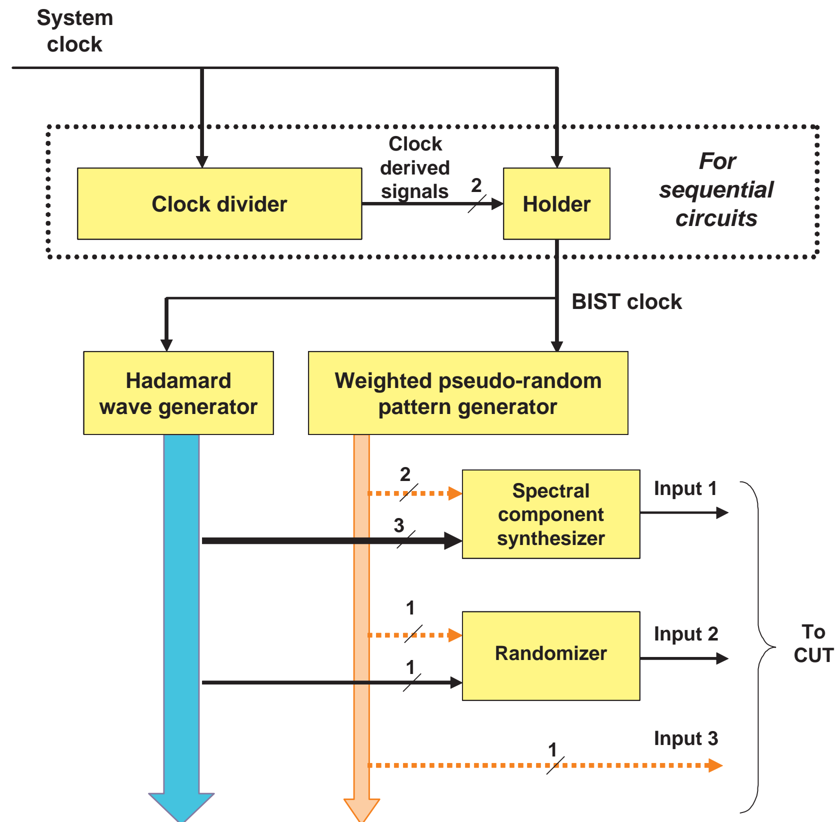


Figure 8.4: Proposed spectral BIST architecture.

8.1.2 Spectral BIST implementation

The goal of the BIST implementation is to design hardware that will generate vectors exhibiting similar component spectrum C and power spectrum P as the original ATPG vectors. This is achieved by combining the chosen M prominent components in appropriate proportions and phases. This is implemented using a spectral component synthesizer. Randomness or noise, if required, is inserted in appropriate amounts to the generated vectors. This is achieved using the randomizer circuit.

Figure 8.4 shows the proposed generic spectral BIST architecture, which consists of: Hadamard wave generator, spectral component synthesizer, Randomizer, Weighted pseudo-random bit-stream generator, Holder circuit and Clock divider circuit. The Holder circuit and the Clock divider circuit are used only in the case of sequential Circuit-Under-Test (CUT) as will be explained later. Figure 8.4 is a BIST circuit designed for a CUT with three inputs. The Hadamard wave generator generates the spectral components which are combined by the component synthesizer (generally, one per PI) using random bit-streams provided by the weighted pseudo-random bit-stream generator. The spectral component signals are shown in dark bold lines while the weighted random signals are in dotted lines. Noise, if required, is inserted in the combined spectral components by the randomizer (also one per PI) supplied with an appropriate weighted random bit-stream. In this example, the first input of the CUT has three prominent components, which are combined by the component synthesizer. The second input of the CUT has only one prominent component, hence no component synthesizer is required. A randomizer adds the required amount of noise. The third input of the CUT has no prominent components and hence a random bit-stream is directly fed from the pseudo-random bit-stream generator. Next, we describe the components of this BIST architecture.

Hadamard wave generator

The heart of the proposed BIST hardware is the Hadamard wave generator, which generates the Walsh functions or Hadamard spectral components for the spectral component

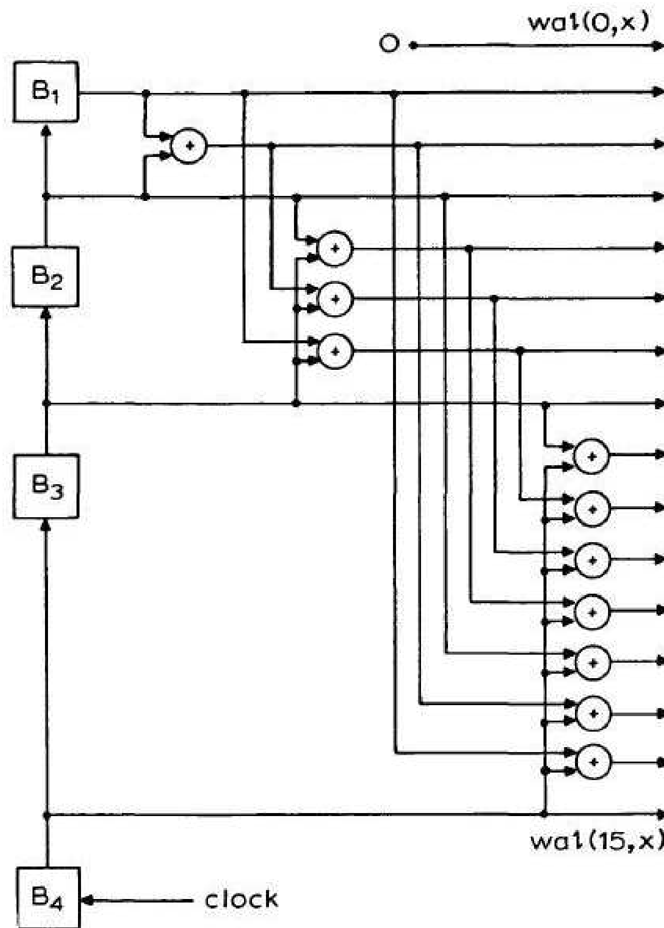


Figure 8.5: Walsh function generator of order 4 that generates 16 Walsh functions [156].

synthesizer. There is much literature on Walsh function generators [31, 11, 33, 52, 156]. Many implementations are based on an arithmetic or a Gray code counter and combinational logic. The speed of operation of such designs is restricted by the speed of the counter. Other fast implementations exist, but they require extra hardware. We selected a Walsh function generator proposed in [52], which uses an arithmetic counter and has low hardware overhead. The generator of order n, which generates 2^n Walsh functions, requires n flip-flops and $2^n - N - 1$ XOR gates. Figure 8.5 shows an example of a Walsh function generator of order 4, which is taken from [156].

The order of the Hadamard matrix n, used for spectral analysis, is also used to implement the Hadamard wave generator. Since higher order Hadamard matrices are constructed

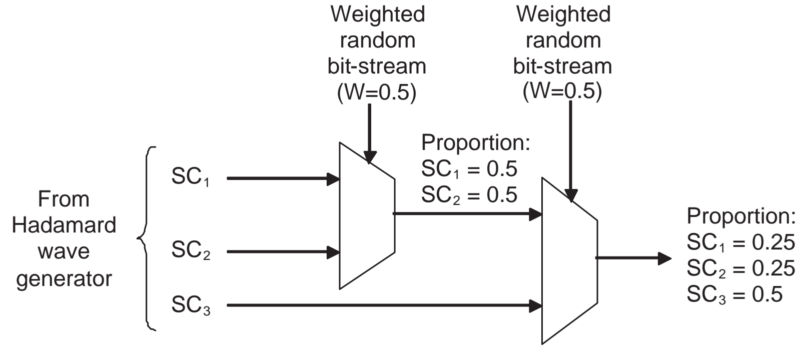


Figure 8.6: Spectral component synthesizer that combines three spectral components.

from lower order matrices [137, 139], lower order matrices form a subset of the higher order matrices. Thus higher order matrices are able to characterize a given bit-stream better than lower order matrices. However, the area overhead for implementing the Hadamard wave generator increases exponentially with the order of the Hadamard matrix as mentioned earlier. Hence we have a tradeoff between the resolution of spectral analysis and the area overhead of the implemented hardware. For our implementation, for the value of n, we chose a lower bound of 4 and an upper bound of 8. These bounds were chosen so that they would provide the optimum tradeoff mentioned above. The specific value was chosen such that the hardware overhead of the wave generator was approximately 5% of the total circuit area.

Spectral component synthesizer

As described earlier, M prominent spectral components are combined in required proportions and phases. The proportions are determined from the power magnitude of the spectral components (from spectrum P) and phases from their signs (from spectrum C), both of which are obtained from spectral analysis of Section 8.1.1. We combine the spectral components in a multiplexer called "spectral component synthesizer", as shown in Figure 8.6. The inputs to the multiplexer are the chosen spectral components, which are generated by a Hadamard wave generator, and its select line is driven by a weighted random bit-stream. The weighting of the bit-stream is determined by the proportion in which the

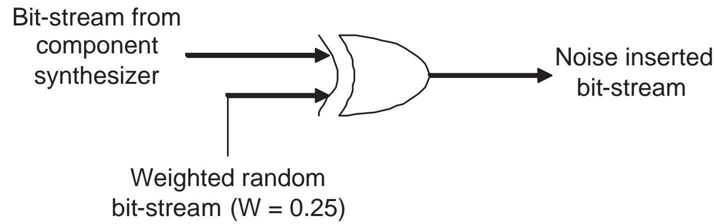


Figure 8.7: Randomizer XOR gate that randomly flips 25% of bits.

components are to be combined. The generation of a weighted random bit-stream will be discussed in the following sections. Figure 8.6 depicts an example of the proposed structure which combines three spectral components SC1, SC2 and SC3 in proportions of 0.25, 0.25 and 0.5, respectively.

Randomizer

Along with the prominent spectral components, some randomness or noise is present in the ATPG vectors which need to be inserted in the regenerated vectors. Due to the filtering effect of the threshold T_H, for some of the inputs, the number of selected prominent components (M_S) could be less than M. For inputs with $M_S > 1$, it is observed that some amount of noise is inherently inserted by the spectral component synthesizer due to its use of a weighted pseudo-random bit-stream. However for inputs with $M_S = 1$, noise needs to be inserted explicitly. For such inputs the level of noise or perturbations to the prominent spectral component is generally very low although important and spread out in the long sequences of ATPG vectors which are not picked up by the short sequence spectral analysis. To estimate the amount of noise in such sequences, we analyze their runs of 0s and 1s. We pick the top 95% of the run lengths to eliminate the noisy effects of the short run-lengths. The reciprocal of the average of the selected run-lengths then gives the average amount of randomness or perturbation to be inserted in the prominent spectral component. The perturbation is then inserted in the generated vectors using the randomizer which performs an XOR operation with a weighted random bit-stream. Figure 8.7 shows an example of flipping 25% of the bits randomly by using a randomizer XOR gate.

Weighted pseudo-random bit-stream generator

This circuit generates weighted pseudo-random bit-streams required by the component synthesizer and the randomizer. We use a 16-bit cellular automata register (CAR) to generate the pseudo-random bit-streams. The CAR is constructed using the rules as described in Section 5.2.2. Weighted bit-streams are obtained using a combination of AND and OR gates as required. The different weights to be generated are determined by the proportions in which the spectral components are to be mixed and by the amount of randomness to be added for all inputs. We quantize the required weights into $W = 2^w$ fixed weights that take the form of $i \times 2^{-w}$ for $i = 0$ to $2^w - 1$. For reported experiments, we used a value of $w = 4$. We also generated two additional weights of 2^{-6} and 2^{-8} for the randomizer.

Holder circuit

The Holder circuit implements vector holding, which involves holding input vectors constant for several clock cycles while applying the system clock to the circuit under test. As this method is not relevant for combinational circuits which have no memory states, this method is used only in the case of sequential circuit-under-test. We use vector holding to enhance our fault coverage, based upon its reported benefits [48, 93, 96, 158]. It was believed in [96] that holding proves effective due to several reasons. Holding a vector for several clock cycles helps fault effects latched in flip-flops to be observed at the primary outputs. Also if a hard-to-detect fault is activated by a random vector, then holding the vector for multiple cycles will activate the fault multiple times, thus increasing the probability of its detection. The number of clock cycles to hold the input vectors can be determined in different ways. In [96], the number of clock cycles to hold the input vectors is determined by using a deterministic sequential test pattern generator to detect the flip-flop output faults. In [48], the number of hold cycles is determined by logic simulation of the fault-free circuit with vectors having different hold cycles and determining their ability to set flip-flops to 0s and 1s and to traverse most states. It has been reported [93, 158] that "Holding a vector V_b for a testable stuck-at fault f_B in combinational circuit C_B d+1 times, where d is the sequential

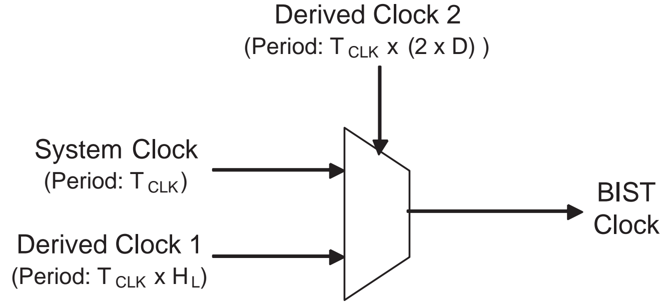


Figure 8.8: Holder circuit implemented using a multiplexer and clock derived signals.

depth of the corresponding acyclic sequential circuit C created by adding flip-flops at any wire of C_B, gives a test sequence for all sequential faults in C corresponding to f_B."

In the proposed method, we determine the number of hold cycles from an upper bound on the sequential depth of the circuit [121], which we shall denote as H_L. The value of H_L is rounded to the nearest power of 2. The upper bound of the sequential depth [121] is defined as the minimum number of cycles required to initialize a flip-flop (for both cases of 0 and 1) and propagate its value to a primary output. In our BIST scheme, vectors are generated with and without holding. First we generate D vectors without holding and then we generate D/H_L vectors, each of which is held for H_L clock cycles. Although any appropriate value can be chosen for D, an effective value for D was found to be proportional to the length of the ATPG vectors. We choose the block length D equal to *length* (ATPG Vector Length/50) rounded to the nearest power of 2. Figure 8.8 shows the implementation of the holder circuit. It consists of a multiplexer whose inputs are clock signals with periods CLK and $CLK \times H_L$. The control signal to the multiplexer is driven by a clock signal of period $CLK \times (2 \times D)$. The output of the holder circuit is the BIST clock which drives the Hadamard wave generator and the weighted pseudo-random bit-stream generator blocks. All the input signals of the holder circuit are derived from the clock and are provided by the clock divider circuit.

Clock divider circuit

The clock divider circuit provides the necessary signals to the holder circuit and again is used only in case of sequential circuit-under-test. The clock divider circuit generates two clock derived signals by dividing the clock frequency by H_L and $(2 \times D)$, respectively, which are then provided to the holder multiplexer to generate the BIST clock. The clock divider circuit is constructed from an asynchronous binary counter consisting of $\lceil \log_2(2 \times D) \rceil$ flip-flops. If the BIST environment uses a pattern counter (which is found in many cases) then the clock derived signals can be conveniently obtained from appropriate outputs of the pattern counter.

8.2 Reseeding of proposed test pattern generator

One of the main challenges for BIST has been the design of a potent Test Pattern Generator (TPG) which can generate test vectors having satisfactory fault coverage. The faults which are of concern are mainly the random pattern resistant faults which may require specific test vectors for their detection [29]. Several approaches have been proposed for covering these faults. One of the popular approaches employed is reseeding of the TPG, in which relevant values are loaded in the flip-flops of the TPG which help the TPG emulate the deterministic patterns and detect the hard-to-detect faults. This approach not only covers the random pattern resistant faults but also can provide encouraging test data compression capabilities.

In this section, we shall discuss the use of reseeding in our proposed BIST architecture. Our proposed spectral BIST TPG hardware can be viewed as two sub-modules consisting of a set of flip-flops and combinational logic. If the flip-flops are provided with a facility to be set to any required values either through a serial or parallel interface, then a required seed can be loaded in the flip-flops in order to enable the TPG to generate a required deterministic test vector. Figure 8.9 illustrates the architecture with reseeding capability. As will be shown later in the results section, the number of flip-flops in the BIST hardware is

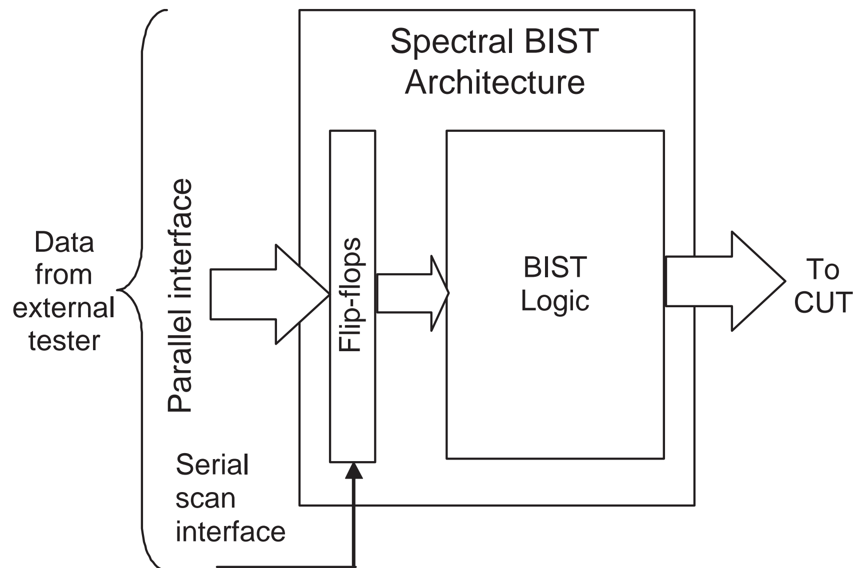


Figure 8.9: Reseeding of proposed spectral BIST TPG.

much lower than the number of inputs being driven of the CUT, which gives this architecture interesting test data decompression capabilities.

With this added reseeding capability, the proposed BIST architecture can function in two extra modes, in addition to the pure BIST mode discussed in previous sections. The two additional modes of operation are external tester mode and hybrid BIST mode. The salient parameters by which we can compare these modes of operation are fault coverage, test data volume and test application time.

In *pure BIST mode*, there is no data being supplied from the external tester and the TPG generates the test vectors for the CUT. This mode does not make use of the reseeding capability. The flip-flops are reset in the beginning of the BIST operation and are not loaded with any external values. This mode leads to lower fault coverage than the case when test vectors are applied from the external tester mode and has ideally zero test data volume. Its test application time is determined by the required fault coverage and the frequency of operation of the on-chip system clock.

In *external tester mode*, seed vectors are loaded from the external tester into the set of flip-flops (either through a serial scan interface or through a parallel interface) which are decompressed by the BIST logic and applied to the inputs of the CUT. The salient feature of this mode is high fault coverage. Test data volume is determined by the number of flip-flops to be loaded and the number of test vectors. Test application time is determined by the number of test vectors and the frequency of operation of the external tester.

In *hybrid BIST mode* the TPG generates test vectors for the CUT and also is loaded with relevant seeds in the flip-flops at frequent intervals to improve its performance. This mode is marked with high fault coverage, lower test data volume and test application time as compared to the external tester mode.

8.3 Results

We emulated our BIST method using a MATLAB program such that the generated vectors would be the same as or very close to those generated by actual hardware. These vectors include the quantization errors involved in the BIST test generation method which would not be included in the software based methods like [42]. Thus software based methods would give results which are more optimistic than hardware based methods. We implemented our method on ISCAS'85 (combinational) [155] and ISCAS'89 (sequential) [154] benchmark circuits to show the efficacy. We analyze both cases of modes, without reseeding(pure BIST mode) and with reseeding(external tester mode and hybrid BIST mode) and present their results separately in the following sections.

8.3.1 Results for BIST mode without reseeding

We analyzed the case of pure BIST mode when no reseeding is used on both combinational and sequential circuits. Their results are presented in the following sections.

Table 8.1: Details of combinational circuits on which our proposed method was employed.

Circuit	No. of inputs	No. of outputs	No. of gates	No. of faults (collapsed)
c7552	207	108	3512	7550
s15850 (combinational)	600	670	9772	11697

Table 8.2: Details of implemented spectral BIST TPG.

Circuit	Hadamard dimension	Cellular Automata Size	Total flip-flops used	No. of gates
c7552	6	24	30	856
s15850 (combinational)	7	28	35	2532

Results for combinational circuits

We implemented our proposed BIST methodology on two combinational circuits; ISCAS'85 c7552 and the combinational part of ISCAS'89 s15850 benchmark circuits. Table 8.1 gives the details of the two circuits. ATPG vectors were generated for stuck-at faults with don't care bits using the program ATALANTA [79]. These ATPG vectors were processed using spectral analysis and reshuffling algorithm as described in Section 8.1.1 to obtain the spectral information, which is then used to construct the BIST TPG hardware. To calculate the area overhead, we assumed a fixed number of transistors for each type of gate: AND - 6, OR - 6, NAND - 4, NOR - 4, XOR - 6 (pass-transistor logic design), 2 input MUX - 4 (pass-transistor logic design), D flip-flop - 22.

Table 8.2 gives the details of the implemented BIST circuit. Column 2 gives the order of Hadamard matrix used. Column 3 gives the size of the Cellular Automata Register(CAR). Columns 4 and 5 give the total number of flip-flops and gates used in the BIST hardware, respectively. The BIST TPG along with the benchmark circuit was then fault simulated using Mentor Graphics tool FlexTest [91] and test coverage was determined.

Table 8.3: Test coverage comparison of random, weighted random and proposed spectral BIST method for 64000 vectors.

Circuit	Random vectors	Weighted Random vectors	Spectral BIST
c7552	97.41%	97.86%	99.81%
s15850 (combinational)	96.81%	97.41%	98.77%

Table 8.4: Area overhead comparison of proposed spectral BIST and Pseudo-Random Pattern Generator (PRPG).

Circuit	No. of gates in circuit	Spectral BIST		PRPG	
		No. of gates	% Area overhead	No. of gates	% Area overhead
c7552	3513	976	27.78	830	23.63
s15850 (combinational)	9772	2672	27.34	2400	24.56

Table 8.3 gives the test coverage results for random vectors, weighted random vectors and our proposed method for 64000 vectors. Weights for the weighted random vectors were obtained from their corresponding ATPG vectors without quantization. Random and weighted random vectors were generated using a software random number generator. As shown in Table 8.3, our proposed spectral TPG obtains better test coverage for both considered circuits.

Table 8.4 gives the number of gates of the original circuit, the area overhead of our proposed method and the area overhead of the Pseudo-Random Pattern Generator (PRPG). For the PRPG, the area overhead comprises of the size of the LFSR (number of flip-flops) which is proportional to the number of inputs of the CUT. As can be observed, the area overhead of our proposed method is comparable to that of the pseudo-random pattern generator. The magnitudes of the area-overheads for our proposed method, however, are noticeably large. We believe that implementing our method on larger circuits would result in much smaller values for area overheads. Furthermore the algorithm for implementing

Table 8.5: FlexTest ATPG results.

Circuit	FlexTest ATPG result			
	No. of vectors	Total no. of faults	# faults detected	Fault cov. (%)
s298	153	308	273	88.64
s820	1127	850	793	93.29
s1423	3882	1515	1443	95.25
s1488	736	1486	1446	97.31
s5378	739	4603	3547	77.06
s9234	15528	6927	1588	22.92
s15850	61687	13863	7323	52.82
s38417	55110	31180	15472	49.62

our proposed method on combinational circuits can be improved, in terms of better preprocessing of test vectors and optimum selection of spectral components.

Results for sequential circuits

We implemented BIST on eight ISCAS'89 benchmark circuits. We inserted a reset signal in the circuits to initialize the flip-flops. We activate it only once before beginning the BIST session. The initial reset can be performed at a slow speed and hence the reset signal can be implemented using minimum resources. ATPG vectors with an initial reset were generated to detect stuck-at faults in the circuits using Mentor Graphics commercial tool FlexTest [91]. Table 8.5 gives the results of this ATPG. These ATPG vectors were analyzed for prominent spectral components and noise as described in Section 8.1.1. Using this information in the MATLAB program, 64,000 BIST emulated test vectors were generated and then fault simulated again using FlexTest.

Table 8.6 gives the results for the number of faults detected by our method ('Spec. BIST') for each circuit and they can be compared with results from random and weighted random vectors (both without and with holding), ATPG vectors and with the method in [26]. For random, weighted random and the proposed method we generated 64,000 vectors. For the weighted random vectors, weights for each circuit were used from their corresponding ATPG vectors without quantization. Random and weighted random vectors

Table 8.6: Experimental results on fault detection by BIST patterns.

Circuit	Total no. of Faults	Number of faults detected						
		FlexTest	Random		Weighted Random		**Spec. BIST***	Haar BIST [26]
			Without Holding	With Holding	Without Holding	With Holding		
s298	308	273	269	273	273	273	273	273
s820	850	793	414	449	744	764	777	710
s1423	1515	1443	891	1217	1449	1469	1468	1468
s1488	1486	1446	1161	1369	1443	1443	1443	1441
s5378	4603	3547	3222	3424	3288	3537	3603	3609
s9234	6927	1588	1268	1305	1293	1303	1729	1413
s15850	13863	7323	5249	6270	5847	6696	6844	5888
s38417	31180	15472	4087	4185	4803	4949	17020	4244

*Proposed spectral BIST method

were generated using a software random number generator. For implementing holding for the random and weighted random vectors, we used the same holding scheme as used in our proposed method.

As shown in Table 8.6, the proposed BIST method detects equal or greater number of faults in six out of eight circuits than the existing methods of random, weighted random and [26]. Also in five out of eight circuits, our proposed method was able to detect at least as many faults as detected by ATPG vectors using 64,000 BIST vectors. Noticeable results were obtained for s38417, where our proposed method detected 17020 faults as compared to 15472 faults detected by ATPG vectors. The effectiveness of the holding scheme is evident from the results obtained for random and weighted random vectors where most circuits benefited by the method.

Table 8.7 shows the effectiveness of our proposed BIST method over longer numbers of vectors and by comparing its results with those obtained from ATPG vectors. The last column gives the number of vectors required by our BIST method to achieve at least as much fault coverage as ATPG vectors. As observed from columns 4 and 5, the fault coverage gradually increases as more vectors are applied. Also we observe from column 6 that eventually six out of eight circuits were able to achieve at least as much fault coverage

Table 8.7: Comparison of fault coverage and number of vectors with FlexTest ATPG.

Circuit	FlexTest		Hadamard BIST		
	Fault cov. (%)	No. of vectors	Fault cov. (%) at 64K vectors	Fault cov. (%) at 128K vectors	BIST vectors for FlexTest ATPG cov.
s298	88.64	153	88.64	88.64	757
s820	93.29	1127	91.41	91.88	(!)
s1423	95.25	3882	96.90	96.90	22345
s1488	97.31	736	97.11	97.11	(!)
s5378	77.06	739	78.27	78.67	8984
s9234	22.92	15528	24.96	25.25	8835
s15850	52.82	61687	49.37	52.15	198061
s38417	49.62	55110	54.59	63.07	43240

Table 8.8: BIST area overhead in transistors.

Circuit	No. of transistors in circuit	**Hadamard BIST [this work]**				Haar BIST [26]	
		with clock divider circuit		without clock divider circuit			
		No. of transistors	% Area overhead	No. of transistors	% Area overhead	No. of transistors	% Area overhead
s298	890	908	102.02	820	92.13	834	93.71
s820	1896	1472	77.64	1340	70.68	1612	85.02
s1423	4624	1637	35.40	1483	32.07	1555	33.63
s1488	4006	1069	26.68	959	23.94	1078	26.91
s5378	12840	2342	18.24	2210	17.21	2487	19.37
s9234	23356	2700	11.56	2502	10.71	2552	10.93
s15850	43696	4908	11.23	4666	10.68	4595	10.52
s38417	108808	3606	3.31	3364	3.09	2135	1.96

as the ATPG vectors. The cells marked with (!) represent cases where our BIST vectors were not able to achieve the ATPG fault coverage although the fault coverages were close to that of ATPG vectors.

Table 8.8 shows a comparison of the area overhead in terms of number of transistors and % area overhead of our proposed method with [26]. To calculate the area overhead, we assumed a fixed number of transistors for each type of gate: AND - 6, OR - 6, NAND - 4, NOR - 4, XOR - 6 (pass-transistor logic design), 2 input MUX - 4 (pass-transistor logic design), D flip-flop - 22. In the table we report results for two cases, one where a clock divider circuit is implemented and the other where an existing pattern counter is reused eliminating the need of a clock divider. As compared to [26], the area overhead of our method with a clock divider circuit is lower in three out of eight circuits, while the overhead is lower in six out of eight circuits where the clock divider is not used.

8.3.2 Results for BIST mode using reseeding

As mentioned in Section 8.2, reseeding can be employed and the flip-flops in the TPG can be loaded with external values. The two modes of operation as discussed earlier are the external tester mode and hybrid BIST mode. We implemented reseeding on the spectral BIST architecture of two combinational circuits; ISCAS85 c7552 and the combinational part of ISCAS89 s15850 benchmark circuits. The details of the circuits and of the implemented BIST architecture were presented earlier in Tables 8.1 and 8.2. The flip-flops used in the Hadamard wave generator and the weighted pseudo-random bit-stream generator in our proposed BIST architecture (Figure 8.4), indicated in column 4 of Table 8.2, are loaded with a required seed from the external tester. Since only these two modes are able to provide 100% test coverage (fault coverage of the pure BIST mode saturates below 100% due to its deterministic nature of test generation), we compare these modes in terms of test data volume and test application time for 100% test coverage. The seeds can either be loaded through a serial scan interface or through a parallel interface as indicated in

Table 8.9: Comparison of test data volume and test time for ATPG and different modes of operation of spectral BIST for c7552.

Modes of test application		No. of vecs./ seeds	No. of inputs	Test data volume (bits)	No. of tester cycles	No. of system clock cycles	Test time (us)†
ATPG (parallel)		247	207	51129	247	0	247
ATPG (serial)		247	1	51129	51129	0	51129
Spectral BIST	ETM (parallel)	197	30	5910	197	0	197
	ETM (serial)	197	1	5910	5910	0	5910
	HBM (parallel)	33	30	990	33	8034	113
	HBM (serial)	33	1	990	990	8034	1070

† assuming tester cycle period T_{tester} = 1000ns and on-chip system clock period T_{clk} = 10ns

Figure 8.9. Several approaches have been proposed for parallel reseeding including on-chip dynamic techniques [68].

In the External Tester Mode (ETM), the TPG is used as a decompressor for one-seed-per-vector operation. In this mode the TPG is not used to generate any new vectors, but only to decompress the seeds that are being applied to its inputs. In Hybrid BIST Mode(HBM), the TPG is operated in normal mode to generate new vectors and is reseeded at specific intervals to improve the fault coverage. The specific intervals for reseeding are determined by a coverage threshold and a step-size. Whenever the test coverage improvement over a given step-size falls below the threshold, reseeding is performed. We use a test coverage improvement threshold of 0.1% for a step-size of 1000 vectors. When the number of faults detected by 1000 vectors drops to one, the TPG is run for only one clock cycle after reseeding until all the faults are detected. The next test seed to load for reseeding is obtained using an ATPG program on the remaining undetected faults in the CUT.

Table 8.9 gives the comparison of test data volume and test time for ATPG and the different modes of operation of spectral BIST for the circuit c7552. For ATPG, the test

Table 8.10: Comparison of test data volume and test time for ATPG and different modes of operation of spectral BIST for s15850 (combinational).

Modes of test application		No. of vecs./ seeds	No. of inputs	Test data volume (bits)	No. of tester cycles	No. of system clock cycles	Test time (us)†
ATPG (parallel)		530	600	318000	530	0	530
ATPG (serial)		530	1	318000	318000	0	318000
Spectral BIST	ETM (parallel)	455	35	15925	455	0	455
	ETM (serial)	455	1	15925	15925	0	15925
	HBM (parallel)	134	35	4690	134	20129	335
	HBM (serial)	134	1	4690	4690	20129	4891

† assuming tester cycle period T_{tester} = 1000ns and on-chip system clock period T_{clk} = 10ns

vectors are applied directly to the CUT (without any BIST logic) by the tester with a clock period equal to T_{tester}. The test vectors can either be applied through a parallel interface or scanned into a shift register of size equal to the number of inputs of the CUT. For BIST mode of operation, the internal on-chip system clock is used, whose period is equal to T_{clk}. In the mode ETM, the TPG functions as a decompressor for one-seed-per-vector operation and controlled by the tester clock (T_{tester}). Mode ETM (parallel) uses a parallel interface for loading the seeds in the flip-flops, while Mode ETM (serial) uses a serial scan interface. In HBM, TPG runs on the system clock (T_{clk}) during normal mode of operation, and on the tester clock (T_{tester}) during reseeding. Again, mode HBM (parallel) uses the parallel interface, while Mode HBM (serial) uses the serial scan interface.

In Table 8.9, column 1 gives the number of vectors/seeds that need to be applied for 100% test coverage. Column 2 gives the no. of inputs that are being driven by the tester. For ATPG, the tester drives the inputs of the CUT directly. For spectral BIST, the tester drives its inputs. Column 3 gives the test data volume that needs to be applied. Columns 4 and 5 give the number of clock cycles of the tester and the CUT that are required to apply

the test data. Column 6 gives test application time in micro seconds (us). In calculation of the test time, it is assumed that the period of the tester clock is $T_{tester} = 1000$ns and that of the system clock is $T_{clk} = 10$ns. We chose a much slower tester clock for several reasons. The scan circuitry is generally not optimized for high speed timing. Also using a fast scan clock increases the test power. Fast tester clocks also increase the probability of failing functionally good chips by activating non-functional critical paths. In most cases, the use of fast tester clock frequency is prohibited by the above mentioned issues, although it can be used in certain specific scenarios as required.

In Table 8.9, the results for parallel interface (ATPG (parallel), ETM (parallel) and HBM (parallel)) and serial scan interface (ATPG (scan), ETM (serial) and HBM (serial)) can be compared amongst each other. From the table, we observe that the test data volume for spectral BIST is an order of magnitude lower than direct application of ATPG vectors. Also the test application shows a marked reduction in both parallel and scan interfaces. Table 8.10, similar to Table 8.9, gives the comparison of test data volume and test application time for the circuit s15850 (combinational). A similar trend in test data volume and test application time is also observed in Table 8.10.

8.4 Summary

We present a novel hardware test generation method for BIST environments. ATPG vectors are analyzed for spectrum and random noise level. The ATPG vectors can be derived from gate, register-transfer, or function level algorithms [149, 150] for fault models like stuck-at, delay [152], or several combined fault models [153]. Our hardware patterns mimic the characteristics of ATPG vectors by controlled mixing of spectral components and noise. We propose a novel circuit for mixing spectral components called the "spectral component synthesizer". Noise is inserted using an XOR circuit. Results show fault coverages equal to or greater than those of ATPG vectors in six out of eight sequential benchmark circuits and encouraging results for combinational benchmark circuits. Our method achieved the maximum fault coverage in six out of eight sequential benchmark circuits considered. In

case of combinational circuits, our proposed method performed satisfactorily as compared to the random and weighted random test pattern generators. Area overheads are moderate compared to existing methods. We also exhibit test data compression capabilities of our proposed BIST architecture. Our proposed architecture provides a maximum test data compression exceeding 90% and a test time reduction of more than 30%.

Chapter 9

Conclusion and Future Work

In this section we summarize the contributions of this work, deduce conclusions and provide some suggestions for possible future work in this area

9.1 Conclusion

With the growing design complexities of VLSI digital circuits, it is evident that the corresponding issues of digital circuit testing would need close consideration and efforts for their abatement. The primary issues of concern are the need for reduction in time complexity of test generation, better quality of test vectors and reduction of testing cost. This thesis addresses these issues by making contributions in mainly two areas, which are automatic test pattern generation (ATPG) and Built-In-Self-Test (BIST). We proposed test generation schemes for digital circuits and a pattern generator for built-in self-test environments which utilize the concept of spectral domain signal processing. A related contribution of this thesis towards better quality test vectors is the proposal of the N-Model tests and a method for their minimization using ILP methods.

We proposed a novel method of test generation for sequential circuits using spectral techniques in Chapter 6. Spectral properties are extracted from a modest set of test vectors generated for the sampled RTL faults and new vectors are generated using those properties to cover all the faults in the circuit. Encouraging results were obtained for the various ISCAS'89 benchmark circuits considered which exhibited equal or improved test coverage and reduced test generation time as compared to the commercial sequential test generation tool FlexTest [91].

We proposed a new type of test called as "N-model test" in Chapter 7 which uses N different fault models of choice and a method for their minimization using Linear Programming (LP) methods. Our proposed method shows a noticeable reduction in test set size as compared to conventional single fault model minimization methods. Additionally our proposed ILP-based minimization model offers a trade off between the total number of test vectors and the cost of test application.

In Chapter 8, we proposed a method for designing a pattern generator for digital circuits in BIST environments using spectral techniques. Given a set of test patterns generated for a digital circuit, the objective here is to regenerate the efficacy of those vectors in hardware for BIST using minimal area overhead and test vector length. The original test vectors, which are to be imitated in hardware, are analyzed for their spectral properties and those properties are implemented in hardware to construct a TPG for BIST. We implemented our methodology on non-scan ISCAS benchmark circuits and reported their results. The proposed BIST pattern generator, while attaining the test coverage of the original test vectors in most cases, shows markedly improved test coverage for similar vector length and comparable area overhead as compared to other pattern generators. We investigated reseeding of our proposed TPG and showed interesting test compression capabilities. In the two circuits considered for reseeding, our proposed architecture provides a maximum test data compression exceeding 90% and a test time reduction of more than 30%.

9.2 Future Work

In this section we give some suggestions on future work which may be carried out in the area of spectral testing

9.2.1 Test data compression

The need for test data compression has been emphasized earlier in this thesis in Chapter 4 to reduce the test application time and hence the test cost. Earlier we had discussed exact methods to determine the minimal number of effective test vectors, mainly using

Table 9.1: Comparison of fault coverage and number of vectors with FlexTest ATPG.

Circuit	FlexTest		Hadamard BIST		
	Fault cov. (%)	No. of vectors	Fault cov. (%) at 64K vectors	Fault cov. (%) at 128K vectors	BIST vectors for FlexTest ATPG cov.
s5378	77.06	739	78.27	78.67	8984
s9234	22.92	15528	24.96	25.25	8835
s38417	49.62	55110	54.59	63.07	43240

techniques like Integer Linear Programming (ILP). Here we suggest performing test data compression by reducing the total test data size for digital circuits taking into account the spectral properties extracted from the uncompressed test set. We investigated the test data compression capabilities of our spectral BIST implementation for two combinational circuits in Section 8.3.2 and found encouraging results. We believe that this concept can be extended to all circuits including sequential (both non-scan and scan-inserted) circuits. The origin of this idea finds its roots in the results presented for our spectral BIST implementation of sequential circuits in Chapter 8. Here we duplicate some of the results in Table 9.1, which we had earlier presented in Table 8.7, where the test coverage obtained by our proposed spectral BIST implementation actually exceeded (sometimes in fewer vectors) than the test coverage of the ATPG vectors which were intended to be imitated.

Notable results to consider are those for circuits s9234 and s38417, where the test coverage of ATPG vectors was obtained in fewer vectors by our proposed spectral BIST TPG. We believe that the analysis of spectral properties that we perform in our proposed method retrieves salient characteristics of the test vector set and hints towards its test compression capabilities. By capturing only the information content present in the required salient characteristics, we believe that test data size can be reduced.

9.2.2 Spectral BIST for scan-inserted sequential circuits

In this thesis we proposed methods for implementing the spectral BIST TPG for non-scan (combinational and pure sequential) circuits. This concept can be extended to include

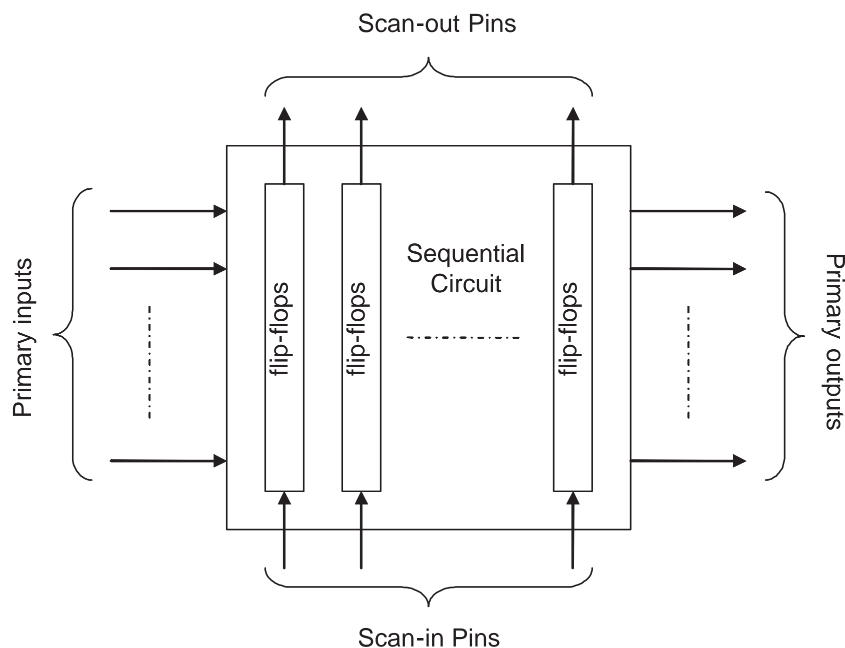


Figure 9.1: Scan-inserted sequential circuit.

scan-inserted sequential circuits, since they form the majority of the circuits that are being manufactured these days. In scan-inserted sequential circuits, a scan chain is used to connect the the flip-flops in the circuit in the form of a serial shift register. Figure 9.1 shows a generic block diagram of a scan-inserted sequential circuit. The scan chain starts with a scan-in pin and terminates with a scan-out pin. The flip-flops can be loaded with any required values by driving the scan-in pin of the scan chain with appropriate values. Similarly any values captured by the flip-flops can be scanned out from the scan-out pin. It is a popular practice to partition the flip-flops into several scan chains for several reasons like multiple clock domains, rising and falling edge flip-flops, test compression, etc.

The problem now is to design a spectral BIST test pattern generator which can drive not only the primary inputs, but also the scan-in pin(s) of the scan-chain(s). In our proposed spectral analysis method, we analyzed the bits which are being applied to each input separately and by doing this we determined the temporal correlations present in the bits. For the current problem at hand, unlike the former case, the bits which are being applied

to the scan-in pin(s) actually exhibit spatial correlations among the values loaded in the flip-flops. Hence we need to consider two types of correlations, temporal correlation for the primary inputs and spatial correlation for the scan-in pin(s). Another alternative is to insert flip-flops even for primary inputs and outputs, and connect them using a scan chain. In this scenario we will be required to consider only spatial correlations among the bits which are being applied to the scan chain(s).

Bibliography

[1] *http://aspire.ucsd.edu/ lichen/260c/parwan/.*

[2] V. D. Agrawal, K. T. Cheng, , and P. Agrawal, "A Directed Search Method for Test Generation using a Concurrent Simulator," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 2, pp. 131–138, Feb. 1989.

[3] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-In Self-Test, Part 1: Principles," *IEEE Design and Test of Computers*, vol. 10, no. 1, pp. 73 – 82, Mar. 1993.

[4] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A Tutorial on Built-In Self-Test, Part 2: Applications," *IEEE Design and Test of Computers*, vol. 10, no. 2, pp. 69 – 77, June 1993.

[5] S. B. Akers and W. Jansz, "Test Set Embedding in a Built-In Self-Test Environment," in *Proc. International Test Conf.*, Aug. 1989, pp. 257–263.

[6] M. F. Ashaibi and C. R. Kime, "Fixed-Biased Pseudorandom Built-In Self-Test for Random Pattern Resistant Circuits," in *Proc. International Test Conf.*, Oct. 1994, pp. 929–938.

[7] B. Ayari and B. Kaminska, "A New Dynamic Test Vector Compaction for Automatic Test Pattern Generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 353–358, Mar. 1994.

[8] P. Bardell, W. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo-Random Techniques*. New York: John Wiley & Sons, 1987.

[9] K. G. Beauchamp, *Applications of Walsh and Related Functions with an Introduction to Sequency Theory*. Orlando, FL: Academic Press, 1984.

[10] M. Bershteyn, "Calculation of Multiple Sets of Weights for Weighted Random Testing," in *Proc. International Test Conf.*, Oct. 1993, pp. 1031–1040.

[11] P. W. Besslich, "Walsh Function Generators for Minimum Orthogonality Error," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-15, no. 4, pp. 177–180, Nov. 1973.

[12] M. A. Breuer, "A Random and an Algorithmic Technique for Fault Detection Test Generation for Sequential Circuits," *IEEE Transactions on Computers*, vol. 20, no. 11, pp. 1364–1370, Nov. 1971.

[13] F. Brglez, C. Gloster, and G. Kedem, "Built-In Self-Test with Weighted Random-Pattern Hardware," in *Proc. IEEE Int. Conf. on Computer Design*, Sept. 1990, pp. 161–167.

[14] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000.

[15] S. Chakravarty and P. J. Thadikaran, *Introduction to I_{DDQ} Testing*. Boston: Kluwer Academic Publishers, 1987.

[16] J.-S. Chang and C.-S. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1370–1378, Nov. 1995.

[17] M. Chatterjee and D. K. Pradhan, "A Novel Pattern Generator for Near-Perfect Fault-Coverage," in *Proc. 13th IEEE VLSI Test Symp.*, Apr. 1995, pp. 417–425.

[18] X. Chen and M. S. Hsiao, "Characteristic Faults and Spectral Information for Logic BIST," in *Proceedings IEEE/ACM International Conf. on Computer-Aided Design*, Nov. 2002, pp. 294–298.

[19] X. Chen and M. S. Hsiao, "Testing Embedded Sequential Cores in Parallel Using Spectrum-Based BIST," *IEEE Trans. Computers*, vol. 55, no. 2, pp. 150–162, Feb. 2006.

[20] K.-T. Cheng, S. Dey, M. Rodgers, and K. Roy, "Test Challenges for Deep Sub-micron Technologies," in *Proc. 37th Design Automation Conf.*, June 2000, pp. 142–149.

[21] W. T. Cheng, "The BACK Algorithm for Sequential Test Generation," in *Proc. International Conf. on Computer Design*, Nov. 1988, pp. 66–69.

[22] C. H. Cho and J. R. Armstrong, "B-algorithm: A Test Generation Algorithm," in *Proc. International Test Conf.*, Oct. 1994, pp. 968–979.

[23] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "New Static Compaction Techniques of Test Sequences for Sequential Circuits," in *Europ. Design & Test Conf.*, Mar. 1997, pp. 37–43.

[24] R. P. Davidson *et al.*, "BELLMAC-32 A Self-Testable 32bit Microprocessor," in *Proc. International Test Conf.*, Oct. 1981, pp. 15–20.

[25] S. K. Devanathan and M. L. Bushnell, "Sequential Spectral ATPG Using the Wavelet Transform and Compaction," in *Proc. 19th International Conf. on VLSI Design*, Jan. 2006, pp. 407–412.

[26] S. K. Devanathan and M. L. Bushnell, "Test Pattern Generation Using Modulation by Haar Wavelets and Correlation for Sequential BIST," in *Proc. 20th International Conf. on VLSI Design*, Jan. 2007, pp. 485–491.

[27] P. Drineas and Y. Makris, "Independent Test Sequence Compaction through Integer Programming," in *Proc. International Conf. on Computer Design*, Feb. 2003, pp. 380–386.

[28] C. Dufaza and G. Gambon, "LFSR-Based Deterministic and Pseudo-Random Test Pattern Generators Structures," in *Proc. European Test Conf.*, Apr. 1991, pp. 27–34.

[29] E. B. Eichelberger and E. Lindbloom, "Random-Pattern Coverage Enhancements and Diagnosis for LSSD Logic Self-Test," *IBM J. Research and Development*, vol. 27, no. 3, pp. 265–272, May 1983.

[30] B. J. Falkowski, "Spectral Testing of Digital Circuits," *VLSI Design, An International Journal of Custom-Chip Design, Simulation and Testing*, vol. 14, no. 1, pp. 83–105, Feb. 2002.

[31] B. J. Falkowski and T. Sasao, "Unified Algorithm to Generate Walsh Functions in Four Different Orderings and Its Programmable Hardware Implementations," *IEE Proceedings Vision, Image and Signal Processing*, vol. 152, no. 6, pp. 819–826, Dec. 2005.

[32] F. A. Feldman, "Fast Spectral Tests for Measuring Nonrandomness and the DES," in *CRYPTO*, Aug. 1987, pp. 243–254.

[33] L. C. Fernandez and K. R. Rao, "Design of a Synchronous Walsh-Function Generator," *IEEE Trans. Electromagnetic Compatibility*, vol. EMC-19, no. 4, pp. 407–410, Nov. 1977.

[34] P. F. Flores, H. C. Neto, and J. P. Marques-Silva, "An Exact Solution to the Minimum Size Test Pattern Problem," *ACM Trans. Design Automation of Electronic Systems*, vol. 6, no. 4, p. 629644, Oct. 2001.

[35] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming.* South San Francisco, California: The Scientific Press, 1993.

[36] J. R. Fox, "Test-Point Condensation in the Diagnosis of Digital Circuits," *Proc. IEE*, vol. 124, no. 2, pp. 89–94, Feb. 1977.

[37] H. Fujiwara, "Computational Complexity of Controllability/Observability Problems for Combinational Circuits," in *Proc. Fault-Tolerant Computing Symp.*, June 1988, pp. 64–69.

[38] A. Ghosh, S. Devadas, and A. R. Newton, "Test Generation for Highly Sequential Circuits," in *Proc. International Conf. on Computer-Aided Design*, Nov. 1989, pp. 362–365.

[39] I. Ghosh and M. Fujita, "Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams," in *Proc. 36th Design Automation Conf.*, June 1999, pp. 43–48.

[40] S. Ghosh and T. J. Chakraborty, "On Behavior Fault Modeling for Digital Designs," *Journal of Electronic Testing: Theory and Applications*, vol. 2, no. 2, pp. 135–151, June 1991.

[41] A. Giani, S. Sheng, M. S. Hsiao, and V. D. Agrawal, "Efficient Spectral Techniques for Sequential ATPG," in *Proc. IEEE Design Automation and Test in Europe Conf. (DATE)*, Mar. 2001, pp. 204–208.

[42] A. Giani, S. Sheng, M. S. Hsiao, and V. D. Agrawal, "Novel Spectral Methods for Built-In Self-Test in a System-on-a-Chip Environment," in *Proc. 19th IEEE VLSI Test Symp*, Apr. 2001, pp. 163–168.

[43] P. Goel, "Test Generation Costs Analysis and Projections," in *Proc. 17th Design Automation Conf.*, June 1980, pp. 77–84.

[44] P. Goel and B. C. Rosales, "Test Generation and Dynamic Compaction of Tests," in *Proc. International Test Conf.*, Oct. 1979, p. 189192.

[45] O. Goloubeva, G. Jervan, Z. Peng, M. Sonza Reorda, and M. Violante, "High-level and Hierarchical Test Sequence Generation," in *Proc. of HLDVT*, Oct. 2002, pp. 169–174.

[46] R. C. Gonzalez and R. E. Woods, *Digital Image Processing.* Addison-Wesley Reading, Mass, 1987.

[47] G. Gordon and H. Nadig, "Hexadecimal Signatures Identify Trouble-spots in Microprocessor Systems," *Electronics*, pp. 89–96, Mar. 1977.

[48] R. Guo, S. M. Reddy, and I. Pomeranz, "Proptest: A Property Based Test Pattern Generator for Sequential Circuits using Test Compaction," in *Proc. 36th ACM/IEEE Design Automation Conf.*, June 1999, pp. 653–659.

[49] H.-K. T. Ma and S. Devadas and A. R. Newton, and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Circuits," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 10, pp. 1081–1093, Oct. 1988.

[50] D. S. Ha and S. M. Reddy, "On the Design of Random-Pattern Testable PLA Based on Weighted Random-Pattern Testing," *J. Electron. Testing: Theory and Applications*, vol. 3, no. 2, pp. 149–157, May 1992.

[51] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 8, pp. 957–963, Aug. 2000.

[52] H. F. Harmuth, *Transmission of Information by Orthogonal Functions.* Springer-Verlag, 1972.

[53] J. P. Hayes and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," in *Proc. Fault Tolerant Computing Symp.*, Mar. 1973, pp. 73–78.

[54] J. P. Hayes and E. J. McCluskey, "Testability Considerations in Microprocessor-Based Design," *Computer*, vol. 13, no. 3, pp. 17–26, Mar. 1980.

[55] D. S. Hochbaum, "An Optimal Test Compression Procedure for Combinational Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 15, no. 10, p. 12941299, Oct. 1996.

[56] P. Hortensius, R. McLeod, and H. Card, "Parallel Random Number Generation for VLSI Systems using Cellular Automata," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1466–1473, Oct. 1989.

[57] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, and H. C. Card, "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 8, pp. 842–859, Aug. 1989.

[58] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors," in *Proc. 15th IEEE VLSI Test Symp.*, Apr. 1997, pp. 188–195.

[59] T.-C. Hsiao and S. C. Seth, "An Analysis of the Use of Rademacher-Walsh Spectrum in Compact Testing," *IEEE Trans. Computers*, vol. 33, pp. 934–938, Oct. 1984.

[60] J. L. A. Hughes, S. Mourad, and E. J. McCluskey, "An Experimental Study Comparing 74LS181 Test Sets," *Digest of Papers Compcon*, pp. 384–387, Feb. 1985.

[61] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic.* Orlando, FL: Academic Press, 1985.

[62] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. on Computers*, vol. C-24, no. 3, pp. 242–249, Mar. 1975.

[63] V. S. Iyengar and D. Brand, "Synthesis of Pseudo-Random Pattern Testable Designs," in *Proc. International Test Conf.*, Aug. 1989, pp. 501–508.

[64] D. Josephson and B. Gottlieb, "Silicon Debug," in D. Gizopoulos, editor, *Advances in Electronic Testing: Challenges and Methodologies*, chapter 3, Springer, 2005.

[65] D. Kagaris and S. Tragoudas, "Generating Deterministic Unordered Test Patterns with Counters," in *Proc. IEEE VLSI Test Symp.*, Apr. 1996, pp. 374–379.

[66] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 14, no. 12, p. 14961504, Dec. 1995.

[67] S. C. Kak, "Classification of Random Binary Sequences Using Walsh-Fourier Analysis," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-13, no. 3, pp. 74–77, Aug. 1971.

[68] E. Kalligeros, X. Kavousianos, and D. Nikolos, "Multiphase BIST: A New Reseeding Technique for High Test-Data Compression," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 10, pp. 1429– 1446, Oct. 2004.

[69] K. R. Kantipudi, "Minimizing N-Detect Tests for Combinational Circuits," Master's thesis, Auburn University, AL, U.S.A, May 2007.

[70] K. R. Kantipudi and V. D. Agrawal, "On the Size and Generation of Minimal N-Detection Tests," in *Proc. 19th International Conf. on VLSI Design*, Jan. 2006, pp. 425–430.

[71] K. R. Kantipudi and V. D. Agrawal, "A Reduced Complexity Algorithm for Minimizing N-Detect Tests," in *Proc. 20th International Conf. on VLSI Design*, Jan. 2007, pp. 492–497.

[72] G. Kasturirangan and M. S. Hsiao, "Spectrum-Based BIST in Complex SOCs," in *Proc. 20th IEEE VLSI Test Symp.*, Apr. 2002, pp. 111–116.

[73] O. I. Khan and M. L. Bushnell, "Aliasing Analysis of Spectral Statistical Response Compaction Techniques," in *Proc. 19th International Conf. on VLSI Design*, Jan. 2006, pp. 801–806.

[74] O. I. Khan, M. L. Bushnell, S. K. Devanathan, and V. D. Agrawal, "SPARTAN: A Spectral and Information Theoretic Approach to Partial Scan," in *Proc. International Test Conf.*, Oct. 2007. Paper 21.1.

[75] H. Kim and J. P. Hayes, "High-Coverage ATPG for Datapath Circuits with Unimplemented Blocks," in *Proc. International Test Conf.*, Oct. 1998, pp. 577–586.

[76] A. Krstić and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits.* Springer, 1998.

[77] J. R. Kuban and W. C. Bruce, "Self-Testing the Motorola MC6804P2," *IEEE Design & Test*, vol. 1, no. 2, pp. 33 – 41, May 1984.

[78] S. Kundu, T. M. Mak, and R. Galivanche, "Trends in Manufacturing Test Methods and their Implications," in *Proc. International Test Conf.*, Sept. 2004, pp. 679– 687.

[79] H. K. Lee and D. S. Ha, "ATALANTA: An Efficient ATPG for Combinational Circuits," Technical Report 93-12, Dept. of Elect. Eng., Virginia Polytechnic Inst. And State Univ., Blaksburg, VA, 1993.

[80] R. Lisanke, F. Brglez, A. J. Degeus, , and D. Gregory, "Testability-Driven Random Test-Pattern Generation," *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 6, pp. 1082–1087, Nov. 1987.

[81] R. Madge, B. R. Benware, and W. R. Daasch, "Obtaining High Defect Coverage for Frequency-Dependent Defects in Complex ASICs," *IEEE Design & Test of Computers*, vol. 20, no. 5, pp. 46–53, Sept.-Oct. 2003.

[82] W. Maly, "Realistic Fault Modeling for VLSI Testing," in *Proc. 24th Design Automation Conf.*, June 1987, pp. 173–180.

[83] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski, "Complexity of Sequential ATPG," in *Proc. European Design and Test Conf.*, Mar. 1995, pp. 252–261.

[84] R. Marlett, "An Effective Test Generation System for Sequential Circuits," in *Proc. 23rd Design Automation Conf.*, June 1986, pp. 250–256.

[85] J. P. Marques-Silva, "Integer Programming Models for Optimization Problems in Test Generation," in *Proc. IEEE Asia-South Pacific Design Automation Conf.*, Feb. 1998, p. 481487.

[86] P. Maxwell, I. Hartanto, and L. Bentz, "Comparing Functional and Structural Test," in *Proc. International Test Conf.*, Oct. 2000, pp. 336–343.

[87] P. C. Maxwell and R. C. Aitken, "I_{DDQ} Testing as a Component of a Test Suite: The Need for Several Fault Coverage Metrics," *Journal of Electronic Testing: Theory and Applications*, vol. 3, no. 4, pp. 305–316, Dec. 1992.

[88] E. J. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique," *IEEE Transactions on Computers*, vol. 33, no. 6, pp. 541–546, June 1984.

[89] E. J. Mccluskey, "Built-In Self-Test Techniques," *IEEE Design & Test of Computers*, vol. 2, no. 2, pp. 21–28, Apr. 1985.

[90] E. J. McCluskey, "Stuck-Fault Tests vs. Actual Defects," in *Proc. International Test Conf.*, Oct. 2000, pp. 336–343.

[91] Mentor Graphics, *FastScan and FlexTest Reference Manual*, 2004.

[92] R. Mester and U. Franke, "Spectral Entropy-Activity Classification in Adaptive Transform Coding," *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 5, pp. 913–917, June 1992.

[93] H. B. Min and W. A. Rogers, "A Test Methodology for Finite State Machines using Partial Scan Design," *Journal of Electronic Testing: Theory and Applications*, vol. 3, no. 2, pp. 127–137, 1992.

[94] F. Muradali, V. Agarwal, and B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test," in *Proc. International Test Conf.*, Sept. 1990, pp. 660–669.

[95] F. Muradali, T. Nishida, and T. Shimizu, "A Structure and Technique for Pseudorandom-Based Testing of Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 6, no. 1, pp. 107–115, 1995.

[96] L. Nachman, K. K. Saluja, S. Upadyaya, and R. Reuse, "Random Pattern Testing for Sequential Circuits Revisited," in *Proc. Fault-Tolerant Computing Symp.*, June 1996, pp. 44–52.

[97] Z. Navabi, *Analysis and Modeling of Digital Systems.* New York: McGraw-Hill, 1993.

[98] T. M. Niermann, R. K. Roy, J. H. Patel, and J. A. Abraham, "Test compaction for sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 260–267, Feb. 1992.

[99] P. Nigh, W. Needham, K. Butler, P. Maxwell, and R. Aitken, "An Experimental Study Comparing the Relative Effectiveness of Functional, Scan, I_{DDQ} and Delay-Fault Testing," in *Proc. IEEE VLSI Test Symp.*, Apr. 1997, pp. 459–464.

[100] P. Nigh, W. Needham, K. Butler, P. Maxwell, R. Aitken, and W. Maly, "So What Is an Optimal Test Mix? A Discussion of the SEMATECH Methods Experiment," in *Proc. International Test Conf.*, Nov. 1997, pp. 1037–1038.

[101] P. Nigh, D. Vallett, A. Patel, J. Wright, F. Motika, D. Forlenza, R. Kurtulik, and W. Chong, "Failure Analysis of Timing and I_{DDQ}-only Failures from the SEMATECH Test Methods Experiment," in *Proc. International Test Conf.*, Sept. 1999, pp. 1152–1161.

[102] M. D. O'Neill, D. D. Jani, C. H. Cho, and J. R. Armstrong, "BTG: A Behavioral Test Generator," in *Proc. 9th Int. Symp. on CHDLs and their Applications*, June 1989, pp. 347–361.

[103] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and Systems.* Prentice-Hall, 1996.

[104] P. Palchaudhuri, D. Roychowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata Theory and Applications*, volume 1. Los Alamitos, California: IEEE Computer Society Press, 1997.

[105] W. W. Peterson and E. J. Weldon, *Error-Correcting Codes.* MIT Press, 1984.

[106] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 12, no. 7, p. 10401049, July 1993.

[107] I. Pomeranz and S. M. Reddy, "Dynamic Test Compaction for Synchronous Sequential Circuits using Static Compaction Techniques," in *Proc. Symposium on Fault Tolerant Computing*, June 1996, pp. 53–61.

[108] I. Pomeranz and S. M. Reddy, "Built-In Test Generation for Synchronous Sequential Circuits," in *Proc. International Conf. on Computer-Aided Design*, Nov. 1997, pp. 421–426.

[109] I. Pomeranz and S. M. Reddy, "Vector restoration based static compaction of test sequences for synchronous sequential circuits," in *Proc. International Conf. on Computer Design*, Oct. 1997, pp. 360–365.

[110] I. Pomeranz and S. M. Reddy, "Improved Built-In Test Pattern Generators Based on Comparison Units for Synchronous Sequential Circuits," in *Proc. International Conf. on Computer Design*, Oct. 1998, pp. 26–31.

[111] I. Pomeranz and S. M. Reddy, "Pattern Sensitivity: A Property to Guide Test Generation for Combinational Circuits," in *Proceedings of the 8th Asian Test Symposium*, Nov. 1999, pp. 75–80.

[112] P. Raghavan and C. D. Thompson, "Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[113] R. Raina, C. Njinda, and R. F. Molyneaux, "How Seriously Do You Take Possible-Detect Faults?," in *Proc. International Test Conf.*, Nov. 1997, pp. 819–828.

[114] S. Ravi and N. K. Jha, "Fast Test Generation for Circuits with RTL and Gate-Level Views," in *Proc. International Test Conf.*, Oct. 2001, pp. 1068–1077.

[115] L. Reddy, I. Pomeranz, and S.M.Reddy, "ROTCO: A reverse Order Test Compaction Technique," in *Proceedings of Euro ASIC*, June 1992, pp. 189–194.

[116] M. Renovell, P. Huc, and Y. Bertrand, "CMOS Bridging Fault Modeling," in *Proc. 12th VLSI Test Symposium*, Apr. 1994, pp. 392–397.

[117] E. M. Rudnick, V. Chickermane, and J. H. Patel, "Probe Point Insertion for At-Speed Test," in *Proc. IEEE VLSI Test Symp.*, Apr. 1992, pp. 223–228.

[118] J. Savir, "Module Level Weighted Random Patterns," *Journal of Electronic Testing: Theory and Applications*, vol. 10, no. 3, pp. 283–287, June 1997.

[119] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Transactions on Computers*, vol. 24, no. 7, pp. 695–700, July 1975.

[120] S. Seshu and D. N. Freeman, "The Diagnosis of Asynchronous Sequential Switching Systems," *IRE Trans. Electronic Computing*, vol. 11, pp. 459–465, Aug. 1962.

[121] L. Shen, "Genetic Algorithm Based Test Generation for Sequential Circuits," in *Proc. 8th IEEE Asian Test Symp.*, Nov. 1999, pp. 179–184.

[122] V. N. Shevchenko, *Qualitative Topics in Integer Linear Programming.* American Mathematical Society, 1997.

[123] M. A. Shukoor, "Fault Detection and Diagnostic Test Set Minimization," Master's thesis, Auburn University, AL, U.S.A, May 2009.

[124] M. A. Shukoor and V. D. Agrawal, "A Primal-Dual Solution to Minimal Test Generation Problem," in *Proc. 12th VLSI Design and Test Symp.*, July 2008, pp. 269–279.

[125] M. A. Shukoor and V. D. Agrawal, "A Two Phase Approach for Minimal Diagnostic Test Set Generation," in *Proceedings of the 14th European Test Symposium*, May 2009.

[126] T. J. Snethen, "Simulator-Oriented Fault Test Generator," in *Proc. 14th Design Automation Conf.*, Jan. 1977, pp. 88–93.

[127] C. Stroud, *A Designers Guide to Built-In Self-Test.* Kluwer Academic Publishers, 2002.

[128] A. K. Susskind, "Testing by Verifying Walsh Coefficients," *IEEE Trans. Computers*, vol. 32, no. 2, pp. 198–201, Feb. 1983.

[129] P. A. Thaker, *Register-Transfer Level Fault Modeling and Test Evaluation Technique for VLSI Circuits.* PhD thesis, George Washington University, Washington, D.C., May 2000.

[130] P. A. Thaker, V. D. Agrawal, and M. E. Zaghloul, "Validation Vector Grade (VVG): A New Coverage Metric for Validation and Test," in *Proc. 17th IEEE VLSI Test Symp.*, Apr. 1999, pp. 182–188.

[131] P. A. Thaker, V. D. Agrawal, and M. E. Zaghloul, "Register-Transfer Level Fault Modeling and Test Evaluation Technique for VLSI Circuits," in *Proc. International Test Conf.*, Oct. 2000.

[132] P. A. Thaker, V. D. Agrawal, and M. E. Zaghloul, "A Test Evaluation Technique for VLSI Circuits Using Register-Transfer Level Fault Modeling," *IEEE Trans. CAD*, vol. 22, no. 8, pp. 1104–1113, Aug. 2003.

[133] A. R. Thompson, J. M. Moran, and G. W. Swenson Jr., *Interferometry and Synthesis in Radio Astronomy.* New York: Wiley, 1986.

[134] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD.* Boston: Springer, 2001.

[135] A. J. van de Goor, *Testing Semiconductor Memories: Theory and Practice.* J. Wiley & Sons, 1991.

[136] P. C. Ward and J. R. Armstrong, "Behavioral Fault Simulation in VHDL," in *Proc. 27th. ACM/IEEE Design Automation Conf.*, June 1990, pp. 587–593.

[137] E. W. Weisstein. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com.

[138] P. Wohl, J. A. Waicukauski, and T. W. Williams, "Design of Compactors for Signature Analyzers in Built-In Self-Test," in *Proc. International Test Conf.*, Oct. 2001, pp. 54–63.

[139] S. Wolfram, *A New Kind of Science.* Champaign, IL: Wolfram Media, 2002.

[140] H. Wunderlich, "Self Test Using Unequiprobable Random Patterns," in *Proc. Fault-Tolerant Computing Symp.*, June 1988, pp. 258–263.

[141] H.-J. Wunderlich, "The Design of Random-Testable Sequential Circuits," *Proc. 19th Fault-Tolerant Computing Symp.*, pp. 110–117, June 1989.

[142] H. J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 6, pp. 584–593, June 1990.

[143] H.-J. Wunderlich, "Self Test using Unequiprobable Random Patterns," in *Proc. 17th Int. Symp. Fault-Tolerant Computing*, July 1997, pp. 258–263.

[144] L. Xijiiang, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli, "High-Frequency, At-Speed Scan Testing," *IEEE Design & Test of Computers*, vol. 20, no. 5, pp. 17–25, Sept.-Oct. 2003.

[145] W. Yang and J. Gibson, "Spectral Entropy, Equivalent Bandwidth and Minimum Coefficient Rate," in *Proceedings IEEE International Symposium on Information Theory*, June 1997, pp. 181–.

[146] W. Yang and J. D. Gibson, "Coefficient Rate and Significance Maps in Transform Coding," *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 1373–1377, Nov. 1997.

[147] J. Yi and J. P. Hayes, "A Fault Model for Function and Delay Faults," *Journal of Electronic Testing: Theory and Applications*, vol. 21, no. 6, pp. 631–649, 2005.

[148] N. Yogi and V. D. Agrawal, "High-Level Test Generation for Gate-Level Fault Coverage," in *Proc. 15th IEEE North Atlantic Test Workshop*, May 2006, pp. 65–74.

[149] N. Yogi and V. D. Agrawal, "Spectral Characterization of Functional Vectors for Gate-Level Fault Coverage Tests," in *Proc. 10th VLSI Design and Test Symp.*, Aug. 2006, pp. 407–417.

[150] N. Yogi and V. D. Agrawal, "Spectral RTL Test Generation for Gate-Level Stuck-at Faults," in *Proc. 15th IEEE Asian Test Symp.*, Nov. 2006, pp. 83–88.

[151] N. Yogi and V. D. Agrawal, "Spectral RTL Test Generation for Microprocessors," in *Proc. 20th International Conf. on VLSI Design*, Jan. 2007, pp. 473–478.

[152] N. Yogi and V. D. Agrawal, "Transition Delay Fault Testing of Microprocessors by Spectral Method," in *Proc. 39th IEEE Southeastern Symp. System Theory*, Mar. 2007, pp. 283–287.

[153] N. Yogi and V. D. Agrawal, "N-Model Tests for VLSI Circuits," in *Proc. 40th IEEE Southeastern Symp. System Theory*, Mar. 2008, pp. 242–246.

[154] N. Yogi and V. D. Agrawal, "Sequential Circuit BIST Synthesis using Spectrum and Noise from ATPG Patterns," in *Proc. 17th IEEE Asian Test Symp.*, Nov. 2008, pp. 69–74.

[155] N. Yogi and V. D. Agrawal, "BIST/Test-Decompressor Design using Combinational Test Spectrum," in *Proc. 13th VLSI Design and Test Symp.*, Aug. 2009.

[156] C. K. Yuen, "New Walsh-Function Generator," *Electronics Letters*, vol. 7, p. 605, 1971.

[157] C.-K. Yuen, "Testing Random Number Generators by Walsh Transform," *IEEE Transactions on Computers*, vol. C-26, no. 3, pp. 329–333, Apr. 1977.

[158] J. Zhang, M. L. Bushnell, and V. D. Agrawal, "On Random Pattern Generation with the Selfish Gene Algorithm for Testing Digital Sequential Circuits," in *Proc. International Test Conf.*, Oct. 2004, pp. 617–626.