ON MULTIPLE INTERMEDIATE SIGNATURE ANALYSIS FOR BUILT-IN SELF-TEST

By

Yuejian Wu

B. A. Sc. Beijing University of Aeronautics & Astronautics, P.R. China, 1982

M. A. Sc. Beijing University of Aeronautics & Astronautics, P.R. China, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date Nov. 4, 1993

DE-6 (2/88)

Abstract

Built-In Self-Test (BIST) is becoming a widely accepted means for testing VLSI circuits. BIST usually consists of two major functions known as *on chip test pattern generation* and *test response evaluation.* There are two major difficulties regarding test response evaluation. The first is reducing the *error escape rate* or *aliasing* while still maintaining reasonably small hardware requirements. The other is accurately assessing the impact of aliasing on the overall test quality of a BIST scheme. This dissertation addresses these two difficulties by developing a group of techniques known as *multiple intermediate signature analysis.* Compared to the conventional single signature analysis, multiple intermediate signature analysis has many advantages, e.g., smaller aliasing, easier exact fault coverage computation, shorter average test time, and increased fault diagnosability.

Based on the investigation of an aliasing model, this dissertation develops a comprehensive fault coverage model for predicting the fault coverage performance with multiple intermediate signature analysis. In addition to the parameters used in the aliasing model, such as the number of intermediate signatures and the length of each signature, the proposed model also includes information on the scheduling of intermediate signatures.

In addition to the studies on the conventional multiple intermediate signature analysis, referred to as CMS schemes, this dissertation also describes two novel multiple intermediate signature analysis techniques. The first is a *fuzzy multiple intermediate signature analysis*, or simply called the FMS scheme. Unlike the CMS schemes, where each checked signature must correspond to a specific reference on a one-to-one basis for a circuit under test (CUT) to be declared good, the FMS scheme declares a CUT good if each checked signature maps to any element of the same set of references. In comparison, the FMS scheme is very simple and easy to implement. A complete theory for the aliasing performance and

hardware requirement prediction with the FMS scheme is derived.

The second novel multiple intermediate signature analysis proposed in this dissertation is *single reference multiple intermediate signature analysis*, simply referred to as the SMS scheme. Conventionally, checking n signatures requires n references. With the SMS scheme, however, regardless of the number of checked signatures, only one single reference is needed. The SMS scheme requires minimal hardware for multiple intermediate signature analysis, i.e., essentially the same amount of hardware as for conventional single signature analysis. To efficiently implement the SMS scheme, a systematic approach is developed based on the discovery of some *identical signature properties*. This implementation approach of the SMS scheme does not require any circuit modification of the CUTs. The cost for implementing the SMS scheme is a non-recurring CPU time overhead in the design phase. In return, the SMS scheme yields significantly recurring silicon area savings as well as reduced aliasing. With the algorithms provided in this dissertation, The CPU time overhead for implementing the SMS scheme is very small. For example, if the SMS scheme is used to check two 16-bit signatures, which yields 65,536 times smaller aliasing at no extra hardware cost compared to conventional single signature schemes, the total CPU time overhead required for implementing the SMS scheme is less than 4 seconds on a Sun Sparc 2 workstation for a test length of 2^{20}, independently of the size of CUTs.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to take this opportunity to thank my supervisor, Dr. André Ivanov, whose friendship, constant encouragement, support and valuable guidance I was fortunate to receive all these years. I would also like to thank my colleague Chun Zhang for providing access to her fault simulator which initiated the work reported here. My appreciation goes to Victor Wong, Carly Wong, Kaiping Li and Gang Li for the helpful discussions on various topics. I would like to thank Peter Bonek, who included my work in his VLSI design, for being my first customer even though he has not paid me a penny yet. I am indebted to many other people who helped me these past few years, e.g., Jeff Chow, William Low and Barry Tsuji, who were always there when I had English problems; Andrew Bishop, who helped me put his nice frames on my slides. I would also like to thank my examination committee, especially Dr. N. Saxena, for helping me correct the errors that used to be in Chapter 4. The work reported here would have been impossible without the financial support of the University of British Columbia's Center for Integrated Computer Systems Research, the British Columbia Advanced Systems Institute, and the Natural Sciences and Engineering Research Council of Canada. Finally, I would like to express my sincere thanks to my family. I am very grateful to my dear wife, Yan Sa, who paved the way to the successful completion of this dissertation with her love and support. I do not know if she fully realizes just how much she has contributed to this work. I am extremely grateful to my parents, Yilee Wang and Zude Wu, who always encourage and support whatever I do. Without them, nothing would have been possible. Many thanks also go to my sister, Yueyan Wu, and brother-in-law, Tong Zhou, who gave me their great support by providing the extra care for my parents that I would have provided. I would like to mention my son, Eric Fenghua Wu, who brought special joys to my life even though he was time-consuming.

Claims of Originality

The author claims originality for the following contributions of this dissertation.

- In Chapter 4, the comprehensive fault coverage model is developed. This model is based on a fault detection probability density function of a circuit under test (CUT). In addition to the possible fault coverage loss due to aliasing, this model also takes into account the impact of the test vectors applied to the CUT.

- In Chapter 5, an efficient implementation of conventional multiple intermediate signature analysis is presented. This scheme shares some hardware existing in a conventional signature analysis BIST scheme. The detailed discussions on test control and test result observation are also not published previously.

- In Chapter 6, the proposed concept of *fuzzy multiple intermediate signature analysis* is novel. Unlike conventional multiple intermediate signature analysis, the fuzzy signature analysis does not contain the strict one-to-one signature-reference correspondence. The complete theory for aliasing performance prediction with the fuzzy multiple intermediate signature analysis is given. Considerations for practical implementations of fuzzy signature analysis as well as analysis on the hardware requirements are presented.

- In Chapter 7, a novel concept referred to as *single reference multiple intermediate signature analysis* is proposed. The proposed single reference signature analysis checks multiple signatures against a single reference, thus reducing the hardware requirements for implementation to a minimum. Therefore, this scheme is also referred to as *minimal hardware multiple intermediate signature analysis.*

- In Chapter 7, the discovery of the identical signature properties is a contribution to knowledge. Based on the identical signature properties, a systematic method for implementing single reference multiple intermediate signature analysis is developed. Two algorithms are given, one for the efficient generation of large number of fault-free sequences and the other for the fast identification of the fault-free sequence that possesses the identical signature properties.

- In Chapter 7, the classification of LFSRs according to their *singularity* (a useful property for implementing single reference multiple intermediate signature scheme) is also a contribution to knowledge.

Chapter 1

Introduction

For Very Large Scale Integrated (VLSI) circuits, testing is a difficult and expensive process. With improving semiconductor technology and computer-aided design (CAD) techniques, circuits with a very large number of devices can now be fabricated on a single chip, e.g., the Intel Pentium microprcessor has 3.1 million transistors on a single chip [Barr93]. The increasing package density of VLSI chips not only makes the chips more powerful, but also causes dramatic reductions in chip production cost. On the other hand, the percentage of the chip production expenditure consumed by testing has greatly increased. The increasing package density of VLSI chips has made the problem of testing extremely difficult.

1.1 Dissertation Objective and Outline

When testing VLSI circuits, conventional test strategies have encountered a number of problems, such as the requirement for prohibitive CPU efforts for test pattern generation, low test quality due to limited accessibility to the internal circuit nodes, and the requirement of expensive Automatic Test Equipments (ATE). Built-In Self-Test (BIST) is one of the most promising solutions to these problems. BIST is the capability of a circuit to test itself without requiring external ATEs. In BIST, both the test pattern generation and test response evaluation are performed on the same chip as the circuit under test (CUT). However, when implementing a BIST, there exist several technical difficulties. Among these, two major difficulties are the so-called *error escape* or *aliasing*, and the inability to compute the exact *fault coverage* with reasonable CPU time efforts. The error escape problem is that some faults detected by the test patterns escape detection during the test response

evaluation process, thus causing some faulty circuits to be mistakenly declared good. Due to aliasing, some BIST schemes may result in poor test quality [Saxena85][Zorian86][Argwal83]. This dissertation addresses these two problems by developing a group of techniques known as *multiple intermediate signature analysis.*

The dissertation is organized as follows. The first three chapters are basically introductory chapters. Thus, readers familiar with VLSI testing may skip these chapters. The remainder of this chapter provides a general review of the art of VLSI testing. Chapter 2 is an introduction to BIST which introduces typical BIST hardware models, important BIST components, and commonly used test pattern generation and response evaluation techniques. Since this dissertation concentrates on BIST test response evaluation, a more detailed discussion on this issue and its current status is provided in Chapter 3.

The contribution to knowledge of this dissertation starts from Chapter 4. Chapters 4 and 5 discuss the basis of multiple intermediate signature analysis. Chapter 4 studies the aliasing and fault coverage performance of multiple intermediate signature analysis. More specifically, several models for aliasing and fault coverage predictions are developed. In Chapter 5, other issues associated with multiple intermediate signature analysis are discussed. The chapter begins with a discussion of the possible implementations of multiple signature analysis, in which a new scheme that shares the hardware resources of conventional BIST schemes is developed. This is followed by the discussions of test control, test result observation, and the advantages and drawbacks of multiple intermediate signature analysis compared to other data compaction techniques.

Chapter 6 is devoted to the development of a *fuzzy* multiple intermediate signature analysis technique for BIST. It discusses the basic concept of introducing fuzziness into signature analysis so as to simplify the conventional way of checking multiple intermediate signatures. It also discusses possible implementations, their hardware requirements, models for aliasing performance prediction, and comparisons with some other techniques. Experimental results obtained while using the fuzzy multiple intermediate signature scheme are

also reported in Chapter 6.

Chapter 7 presents a single reference multiple intermediate signature scheme that requires a minimal amount of hardware for multiple signature analysis. In addition to showing how the minimal hardware requirement is achieved, Chapter 7 also develops several techniques that help to efficiently implement the proposed scheme. Feasibility studies as well as experiments on benchmark circuits are reported. Finally, Chapter 8 summarizes this dissertation and discusses possible future work in these areas.

1.2 Fault Models

Being physical devices, VLSI circuits are subject to failures. A *failure* is defined to occur when the delivered service by a circuit deviates from its specified service [Abraham86]. The cause of a failure is an *error*, which is defined to be any discrepancy between the actual circuit output sequence and the specified or expected output sequence [Breuer76]. The cause of an error is said to be a *fault* [Abraham86].

The causes of a fault can be numerous. First, incorrect design or design specification of a circuit can lead to a fault in the final fabricated circuit. Secondly, a fault can occur due to manufacturing defects, such as open and poor interconnections, shorts between conductors, excess leakage current, etc. [Bardell87]. Thirdly, even if a circuit is "perfectly" manufactured, it could subsequently wear out in the field due to electromigration, hot-electron injection, spreading charge loss, electrical overload, etc. [Abraham86]. Even during storage, faults may occur in a circuit due to factors such as temperature, humidity, leakage of sealed elements, and aging [Breuer76]. Lastly, even for a perfectly "good" circuit, a fault may occur temporarily in the field due to physical or environmental causes, such as lightening, radiation, stress, vibration, heat dissipation, etc. [Breuer76][Savaria86][Johnson89].

Regardless of the causes of a fault, in order for a fault's effect to be assessed, it must be modeled in a manner that is consistent with the representation of the circuit. In general,

a fault described at a lower level can more accurately represent failure mechanisms, but involves a much greater degree of complexity [Abraham86]. For example, a fault described at the transistor level may become intractable because of the extremely large number of transistors in a VLSI chip, though it can very accurately describe the physical phenomena causing the fault. On the other hand, a fault described at a higher level, such as the gate level or functional level, can significantly reduce the complexity of treatment but, due to the loss of information, may result in some lower level failures not being considered [Abraham86]. For different requirements, many fault models have been developed. Some of them are simple while others are sophisticated. Among them, the *stuck-at* fault model [Breuer76][Bardell87] is one of the simplest and most commonly-used.

A stuck-at-0 (stuck-at-1) fault is defined to be any fault condition that causes a signal line to behave as if it were stuck at logical 0 (1). This model is a logical fault model, and is thus technology independent. Faults can occur singly or in multiples. A special class of these stuck-at faults is the *single stuck-at* fault model that assumes the existence of at most one such fault in a circuit. The single stuck-at fault model is the most commonly used model in practice. The popularity of this model is mainly due to its simplicity and ability to cover many common defects in ICs, e.g., bridging faults [Bardell87] and multiple stuck-at faults [Kubiak91]. However, there exist some defects that the stuck-at fault cannot model very well, e.g., delay faults and CMOS stuck-open faults. Nevertheless, the single stuck-at fault model is still the most popularly used, and still the model against which all other more complex fault models are compared [Bardell87].

As many other researches in this field, this dissertation assumes the single stuck-at fault model. But, as will be noted, the methodology and techniques developed in this dissertation are also applicable to other fault models.

1.3 Test Quality Measures

Testing of digital circuits consists of applying a sequence of input vectors to a circuit, observing the output sequence, and comparing it with a precomputed or expected output sequence. The presence of a given fault is said to be detected when an appropriate input vector or vectors, applied to the CUT, causes an incorrect logic output at one or more of the CUT's output lines. The input vectors are called *test patterns* or *test vectors*. Although a test procedure can be generic, its quality measures are usually associated with a specific fault model. In VLSI testing, the quality is usually measured by the ratio of the detected faults to the total number of possible faults under the assumption of a specific fault model. This ratio is termed *fault coverage*. Due to its dependence on fault models, for a same CUT and a same set of test patterns applied in the same order, the fault coverages obtained for different fault models can be significantly different.

The fault coverage that a given set of test vectors can achieve is usually computed by a process called *fault simulation*. Fault simulation consists of simulating the application of every pattern in the test set to the fault-free as well as a set of faulty circuits (each corresponding to a circuit to which a fault is injected), and comparing the simulated test response of the fault-free circuit with that from each of the faulty circuits. Fault simulation is the only way to determine the exact fault coverage [Wagner87]. In the past few years, gate-level stuck-at fault simulation for combinational circuits has been made extremely fast [Blank84][Waicukauski85][Mamari90][Keller90][Lee91]. A report from industry even declares that further speed up may not be necessary [Atken90]. For other fault models, such as delay faults, fast fault simulation techniques have also be developed [Waicukauski87b][Schulz87][Fink90][Wu92a]. For fault coverage computation, an alternative to fault simulation is the use of analytical techniques for fault coverage estimation [Wagner87][Savir84][Savir84b]. These techniques are based on the knowledge of the *detectabilities*

[1] of the faults in the CUT. Two major difficulties for using these techniques are the obstacle of obtaining the detectability profile of the CUT, and the inadequate accuracy.

1.4 Conventional Approach to IC Testing

Testing a digital circuit requires three major steps, namely the generation of a set of test vectors, the application of these vectors to a CUT, and the analysis of the collected test response from the CUT. In conventional IC testing, an external tester is employed to apply test vectors and to collect and analyze the test response.

When dealing with large circuits, the conventional approach encounters a number of difficulties. First, the computational time requirements for the test pattern generation may be prohibitive. Due to the fact that signals at internal circuit nodes are easier to control in combinational circuits than in sequential circuits, test pattern generation for combinational circuits is much easier. Many test pattern generation algorithms have been developed, e.g., [Goel81][Fujiwara83][Rajski87] [Schulz88]. To speed up the test pattern generation process, fault simulation is usually employed to determine whether a test pattern generated for one fault is also able to detect other faults. If so, these detected faults are dropped from further consideration. Unfortunately, even with the best algorithm, to generate a complete test set (the set of test vectors that covers all detectable faults in a CUT) is still prohibitively expensive for today's large circuits [Sedmak85]. Secondly, the generated test set usually cannot achieve adequate fault coverage. This is because the controllability and observability of the internal circuit nodes through I/O pins have been significantly reduced as a result of the increased complexity and density of VLSI circuits. The reduced accessibility to the internal nodes not only makes test pattern generation more difficult, but results in large number of undetectable faults, and hence poor test quality. Thirdly, external testers or so-called ATEs are not only expensive, but impose a limitation on the speed at which the

[1] The detectability of a fault is defined to be the probability that the fault is detected by a randomly chosen test vector.

CUTs can be tested. Conventionally, IC chips are tested at low speed to verify their static state functionality. However, a CUT that passed the low speed test may not be able to work properly at its operational speed due to the existence of AC faults[1] [Schulz87] [Maxwell91]. The speed at which a circuit can run at test is limited by the speed of the ATE. High speed ATEs are extremely expensive (easily millions of dollars). Moreover, all ATEs are made with existing IC technology. Thus, their speed is likely to be slower than the latest technology. Besides, the amount of test data is becoming too large to be handled efficiently by ATEs. To ease these problems, a group of new techniques known as *design-for-testability* have been proposed [William83][McCluskey85].

1.5 Design for Testability

In general, design-for-testability (DFT) is any design technique that helps make a circuit more testable. For example, one can either enhance the controllability of internal circuit nodes, or enhance their observability, or both. Many DFT techniques are now available. Some are structured or generic design techniques. Some are ad hoc. A group of well-known structured DFT techniques is *scan path design.*

As discussed earlier, testing a sequential circuit can be much more difficult than testing a combinational circuit. Scan design defines two operational modes of a circuit: *normal* mode and *test* mode. In the normal mode, the circuit performs its normal function as a sequential circuit. In the test mode, the circuit is reconfigured into a combinational circuit and a scan register. The problem of testing a sequential circuit is thus reduced to that of testing a combinational circuit and a scan register. The conversion is accomplished by connecting all the circuit's internal state memory elements into a shift register, and connecting one end of the register to an input pin while connecting the other to an output pin. With the scan register, any internal state memory element can be fully controlled by

[1] AC faults are the faults that cause timing failure of a system but may not affect the system's steady state functionality.

shifting a bit into that specific element from the shift register's input pin. The state of the memory element can easily be observed by shifting out its content to the shift register's output pin. In this way, the circuit nodes or lines that are connected to the internal memory elements can be treated as primary inputs and outputs in the test mode.

Although scan design significantly simplifies the problem of IC testing, it has several drawbacks. A possible drawback is that some faults may not be detected since the testing is performed in the test mode instead of the circuit's normal mode. In addition, in exchange for the reduced complexity in testing, scan design implies extra hardware requirements. For example, as estimated in [Nagle89], full scan design usually entails 10% - 20% additional hardware requirements. Furthermore, scan register cells sometimes require a more complex clocking system. Take the Level Sensitive Scan Design (LSSD) [Eichelberger78] technique for instance, which is now a compulsory requirement for all IBM's designs, two clock signals are required. In addition to extra hardware requirements and possible performance degradation, scan design also requires substantially longer test time because the test vectors must be serially shifted into the scan path bit by bit. However, experience has shown that the price paid by scan design is well compensated for by the significantly reduced effort in testing [McCluskey85].

It is worth mentioning another type of scan design for testing multichip assemblies even though this dissertation focuses on chip level testing. This type of scan design is known as *boundary scan*, which assigns a memory element to each I/O pin of a chip, and connects these elements into a shift register called the *boundary scan register*. When testing a multichip assembly, a long shift register is formed by connecting each chip's boundary scan register together. Through the long shift register, the I/O pins of each chip can be easily accessed. Boundary scan design successfully solves the problem of assembly interconnection testing [Hassan88][Jarwala89] without requiring the use of *bed-of-nails* equipments. Boundary scan design has now become an IEEE standard [IEEE90]

1.6 Built-In Self-Test

DFT techniques such as scan design may ease the difficulty of test pattern generation and application. However, the core problems of limited internal circuit node accessibility and the requirements of expensive test pattern generation and ATEs still remain [Sedmak85].

Another DFT design approach which can well be coupled with scan design is built-in self-test (BIST) [Sedmak85][McCluskey85]. BIST is the capability of a product (wafer, chip, multichip assembly, system) to test itself without requiring external ATEs [Sedmak85]. This simple but very promising idea attacks not only the limited accessibility problem, but also the costly test pattern generation and ATE problems. A BIST must consist of a strategy for generating test vectors, a strategy for evaluating test response, and the implementation mechanisms [McCluskey85]. In chip level BISTs, both test pattern generation and test response evaluation are performed by some simple hardware in the same chip that is under test [Gelsinger86][Katoozi92][Zorian91][Hagihara92][Kuban84]. The conventional test approach's expensive requirements for test pattern generation and ATEs are no longer necessary with BIST. Furthermore, the on-chip test pattern generation and output evaluation significantly enhance the accessibility of internal circuit nodes, thus possibly yielding better test quality. Another strength of BIST is the ability to test circuits at their operational speed, which enables the detection of many AC defects in addition to the detection of static state failures. BIST can also be used for in-field test, and for multichip assembly fault diagnosis, which requires faulty chip localization.

As pointed out in [McCluskey85], all BIST methods have some associated cost. Since BIST circuitry uses chip area, there is a decrease both in the *yield*[1] and in the reliability [McCluskey85], and there is an increase in the power consumption [Levy91]. These costs are compensated for, however, by the significant savings in testing and especially system maintenance in the product life-cycle [Agrawal93]. The extra silicon area consumed by

[1] Yield is usually defined to be the ratio of the number of good chips over that of all the chips produced.

BIST used to be considered as an overhead. Recently, however, with increasing demands for higher test quality, more and more people in the VLSI community tend to consider BIST as a necessary function of a chip. Thus, the silicon area associated with BIST is considered as a natural requirement, and not as an overhead. Of course, it is still desirable to keep the hardware requirements of BIST as low as possible for the sake of cost, yield, reliability, and power-consumption.

1.7 Test Quality Problems of BIST

BIST is a simple and powerful idea to solve the problems of VLSI testing. However, it has a distinctive technical difficulty. In BIST, due to the on-chip test response evaluation, the bit-by-bit comparison technique that is usually adopted in the conventional approaches is not normally practical any more. To evaluate the test response efficiently while consuming reasonable amount of silicon area, the test response sequence is usually compacted into a small sequence of bits, called a *signature*. At the end of a test, the signature collected from the CUT is compared with an expected signature or *reference* to determine whether the CUT is fault-free.

A major drawback of the compaction is the loss of information, which may result in some erroneous sequences being compacted into a same signature as the fault-free one, thus causing an incorrect diagnosis whereby a faulty circuit declares itself as good. This problem is well-known as *aliasing* or *error masking* [McCluskey85][Bardell87]. Due to the aliasing problem, the overall test quality of a BIST scheme depends not only on the quality of the test vectors generated, but also on the quality of the adopted data compaction technique. Many recent research efforts have been aimed at reducing the aliasing problem while still maintaining reasonably small hardware requirements, e.g., [Zorian86], [Agarwal87], [Li87], [Robinson87], [Robinson88], [Gupta90] and [Raina91]. But, as will be discussed in the subsequent chapters, most compaction techniques have the deficiency of either excessively

large aliasing or substantially high hardware requirements.

Another difficulty associated with output data compaction is the assessment of the test quality of a BIST scheme. As discussed in Section 1.3, fault coverage is an accepted quality measure of VLSI testing. Fault coverage is usually computed by fault simulation. An important technique that makes fault simulation fast is dropping a fault from further consideration once it is detected by a test vector. This technique is known as *fault dropping*. In BIST, however, fault dropping is not usually possible. This is because, due to possible aliasing, once a fault is detected there is no guarantee that it will be detected at the end of the test. Without fault dropping, fault simulation is not generally computationally feasible for large circuits. Due to the inability to accurately quantify the test quality of a BIST scheme, the test quality issue of BIST has traditionally been split into two parts. The first part is to use fault coverage before compaction to measure the quality of the test vectors generated. The second part is to characterize the possible loss of the coverage due to aliasing. Unlike fault coverage measures, which are deterministic, measures for aliasing are usually probabilistic. Although many advanced probabilistic techniques have been developed, such techniques are difficult to use confidently when dealing with a specific CUT because of the statistical uncertainty. Evidence of the uncertainty can be found in experimental reports, e.g., [Aitken89], [Xavier92], [Rajski91b] and [Debany92].

Chapter 2

Built-in Self-Testing

BIST is becoming a widely-used means for VLSI testing. Many commercial products [Kuban84][Daniels85][Gelsinger86][Gelsigner89][Hagihara92] illustrates how far the BIST concept has become a reality. Testing different types of circuits usually requires different types of BIST [Zorian91], e.g., BIST for random logic, BIST for RAMs, BIST for ROMs, etc. This dissertation focuses on the BIST schemes designed for testing random logic. However, the schemes developed in this dissertation still apply to the BIST for other types of circuits. As discussed in Chapter 1, the two major functions of BIST are the on-chip test pattern generation and output data evaluation. This chapter provides a general review of both these BIST functions. The subsequent chapter will provide a more detailed discussion on BIST output data evaluation since it is the major topic of this dissertation.

2.1 Hardware Models of BIST

BIST is the capability for a chip to test itself. In general, BIST consists of the generation and application of test vectors to the CUT, as well as the evaluation of the test response on the same chip as the CUT. In implementation, there are some commonly-required components, namely an input pattern generator, an output data compactor, a pre-calculated fault-free signature or reference, and a comparator. Fig. 2.1 shows a generic BIST scheme. After applying the test patterns to the CUT, the comparator compares the final content of the data compactor with the reference stored on-chip, and produces a pass/fail (or go/nogo) signal as a test result. Different structures are used in different instances. Depending on how the test patterns are applied and how the output data are collected, Figs. 2.2 - 2.5

give four typical BIST structures. They are the parallel-parallel (Fig. 2.2) [Konemann79], parallel-serial (Fig. 2.3) [Pomeranz92][Li87], serial-parallel (Fig. 2.4), and serial-serial (Fig. 2.5) [Lambidonis91b].

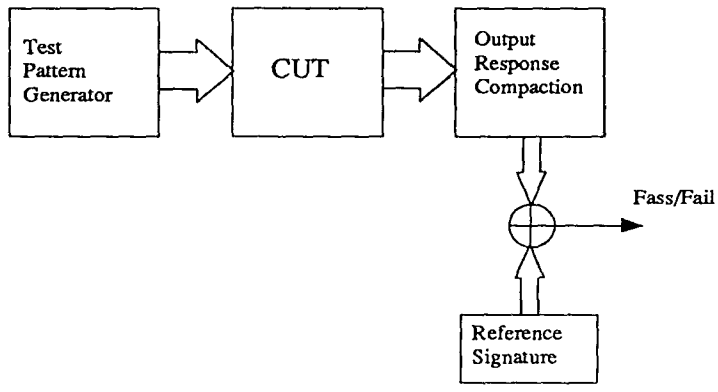


Figure 2.1: General BIST scheme.

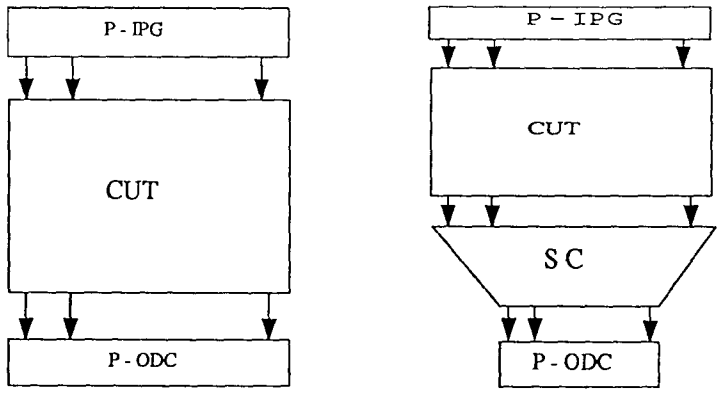


Figure 2.2: The parallel-parallel BIST model.

In Figs. 2.2 - 2.5, the P-IPG is a parallel input pattern generator; the P-ODC is a parallel output data compactor; the SC is a *space compactor* that compacts a m-bit vector into a k-bit vector, where $m > k$; the S-IPG is a serial input pattern generator; and the S-ODC is a serial output data compactor. The S-to-P and P-to-S blocks in the figures are

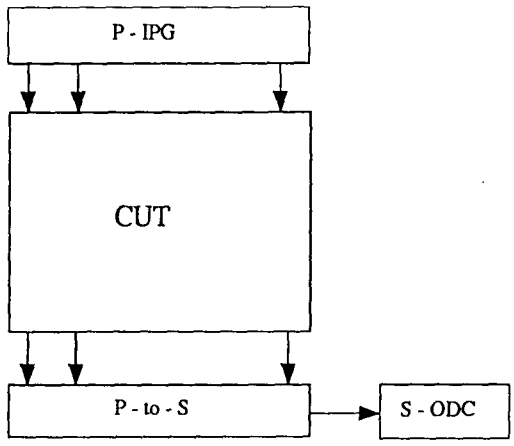


Figure 2.3: The parallel-serial BIST model.

serial-to-parallel and parallel-to-serial converters, respectively.

A P-IPG generates one multi-bit test vector for each clock cycle, and applies each vector to the CUT in parallel. A test vector generated by a S-IPG, however, consists of a series of bits applied to the CUT via a serial-to-parallel converter. The most commonly used test pattern generators are the *Linear Feedback Shift Registers* (LFSR) [Bardell87]. *Cellular Automata* (CA) [Hortensius89] can also be used as test pattern generators.

A P-ODC collects multiple bits concurrently from the CUT. A S-ODC collects bits serially. S-ODCs are often used with P-to-S converters as shown in Figs. 2.3 and 2.5, unless the CUT has only a single output line. The most commonly used P-ODC is the *Multiple Input Shift Register* (MISR) [Bardell87], which is in fact a LFSR with multiple inputs. In general, for a N-output CUT, one can use a N-stage MISR for data compaction. However, the use of the N-stage MISR can be very expensive in silicon area for large CUTs. In comparison, the use of a space compactor followed by a short MISR is more economical [Reddy88][WuM92][Zorian93]. The most commonly used S-ODCs are LFSRs [Bardell87].

Regarding the S-to-P and P-to-S converters, the scan path discussed in Chapter 1 is the most popular. When a scan chain is used, after the application of a test vector,

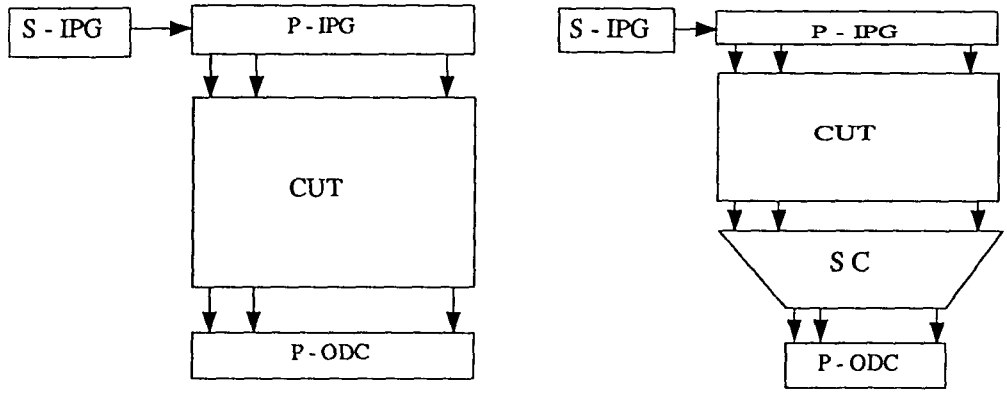


Figure 2.4: The serial-parallel BIST model.

the response at the CUT's output is shifted out of the scan chain for examination. For a scan chain P-to-S converter, each test vector usually yields multiple bits serially. Besides the scan chain, other components can also be used as the P-to-S, e.g., *Multiple Input Non-feedback Shift Registers* (MINSR) [Agarwal87], *XOR trees* [Katoozi92][Li87], and MISRs [Gelsinger86].

2.2 An Important BIST Component — LFSR

A LFSR is simply a shift register with linear feedback, i.e., feedback that is an exclusive OR (XOR) of the contents of the selected memory elements of the shift register. Fig. 2.6 shows an example LFSR with *feedback polynomial* $x^5 + x^2 + 1$.

A LFSR can have two operational modes. When used as a test pattern generator, the LFSR usually works in its *autonomous* mode, where no external input is applied to it. When such a LFSR is initialized with a non-zero *seed*, with each state transition or shift, its content is different from its previous ones. Thus, each register state can serve as a test pattern. Each LFSR has its specific *period*. After the LFSR shifts a certain number of *cycles*, its state returns to its initial state (seed). It has been shown that, for a

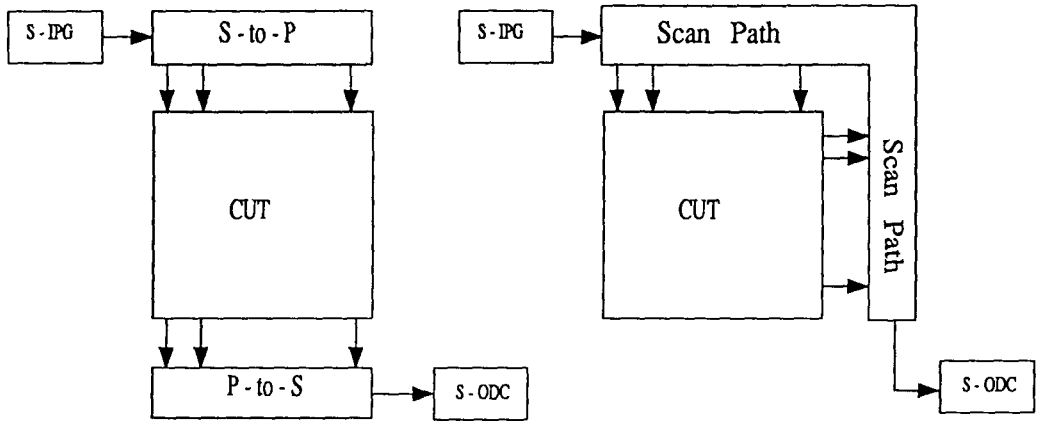


Figure 2.5: The serial-serial BIST model.

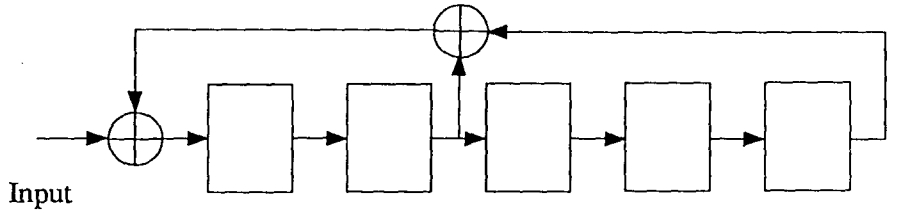


Figure 2.6: An example LFSR.

given number of stages, a LFSR reaches its maximum period if its feedback polynomial is *primitive* [Golomb82][Bardell87]. For a k-stage LFSR, this maximal period is $2^k - 1$. Thus, LFSRs with primitive feedback polynomials are also called *maximum cycle length* LFSRs. The primitive feedback polynomials for different LFSR sizes are provided in a number of publications, e.g., [Golomb82], [Lin83] and [Bardell87].

In addition to the maximum cycle length, the sequence generated from a LFSR with a primitive feedback polynomial also possesses another very useful property. In [Golomb82], it has been shown that any segment, out of the entire sequence of patterns generated, holds properties very similar to random patterns. But, unlike pure random patterns, the sequence

generated from a LFSR is predictable, thus repeatable, if its feedback polynomial and initial seed are known. Such patterns are termed *pseudorandom patterns* [Golomb82][Bardell87]. The property of pseudorandomness leads to the popular use of LFSR-generated test patterns in BIST [Bardell87][Wagner87].

Besides the autonomous mode, a LFSR can also work in an operational mode where an external input sequence is applied to it. If the input sequence is represented as a polynomial, e.g., representing the sequence 11011001 as $x^7+x^6+x^4+x^3+1$, the LFSR in this mode performs the operation of dividing the input polynomial by its feedback polynomial [Lin83][Bardell87]. The final content of the LFSR forms the remainder of the division, while the sequence generated from the last or right-most stage (see Fig. 2.6) of the LFSR represents the quotient. When used for output response compaction, LFSRs are used in this mode. In this case, the input sequence to the LFSR is the test response sequence from the CUT. After shifting the entire sequence into the LFSR, the final content of the LFSR forms the signature of the test response sequence, while the quotient bits of the division are usually discarded. Thus, given the test response sequence and the feedback polynomial, theoretically, the final signature of the sequence can be calculated by polynomial division. Ideally, different sequences produce different signatures. Then, by checking the final content of the LFSR alone, one can distinguish fault-free response sequences from faulty ones. Unfortunately, as will be discussed in later sections and chapters, it is not always the case.

The LFSR shown in Fig. 2.6 has only one input. Sometimes, one may wish to use a LFSR as a parallel output data compactor, e.g., the P-ODC in Figs. 2.2 and 2.4. In this case, extra input lines can be added by connecting one input line to each LFSR memory element via an XOR gate. Fig. 2.7 shows such a multiple input LFSR, or simply called MISR, with the same feedback polynomial as the LFSR shown in Fig. 2.6. Obviously, a MISR requires more silicon area as compared to a LFSR of the same number of stages.

Besides LFSRs, another form of pseudorandom number generator called *Cellular Automation* (CA) can also be used for test pattern generation and output data compaction

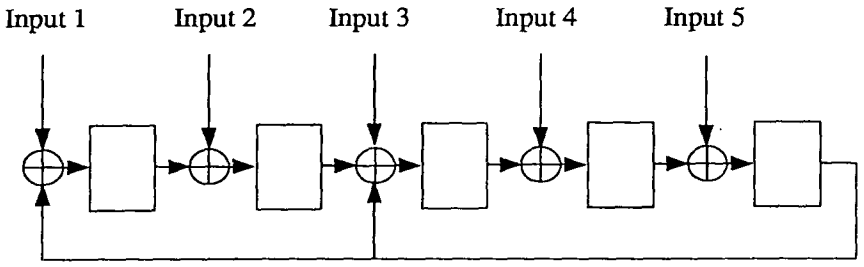


Figure 2.7: An example MISR.

[Hortensius89][Serra90]. Studies have shown that, in many case, CAs yield better test quality than LFSRs [Miller91][Zhang90]. In addition, another advantage of CAs is their regular structure, which is preferred by many VLSI CAD tools. However, concerns with using CAs are its higher hardware requirements, less understood mechanism and poorer documentation as compared to LFSRs. Thus, in practice, LFSRs remain the most popular BIST components for both test pattern generation and response compaction.

2.3 Test Pattern Generation

There are several ways to generate BIST test patterns, namely exhaustive, pseudorandom, weighted random, and embedded deterministic.

2.3.1 Exhaustive and Pseudo-exhaustive Testing

For an n-input combinational CUT, there are at most 2^n possible test vectors. *Exhaustive testing* is testing the CUT with all the possible vectors. It has been shown that exhaustive testing would detect all non-redundant combinational faults [1] in a CUT [McCluskey81], thus eliminating the needs for combinational fault models and fault simulation [McCluskey85]. Furthermore, exhaustive testing is circuit-independent since it requires no information about

[1] Combinational faults are the faults that do not introduce undefined internal states to a CUT, e.g., the stuck-at faults.

the CUT, except for the number of primary inputs. Any binary counter can serve to produce exhaustive test vectors. In practice, however, it is more efficient in silicon area to use a maximum cycle length LFSR slightly modified so that it cycles through all states, including the all-zero state [McCluskey81][Wang86][Bardell87][McLeod92]. Besides a lesser silicon area requirement compared to a binary counter of the same number of stages, a modified LFSR test pattern generator tends to detect more AC faults in addition to combinational faults due to the pseudorandomness of the vectors [Soden89]. For small n, exhaustive testing is a very good method [Gelsinger86]. However, when n is large, say $n > 25$ [Agrawal93], the long test time associated with this method may become prohibitive.

To make exhaustive testing practical for large circuits, a group of techniques known as *pseudo-exhaustive testing* or *verification testing* have been proposed [McCluskey81] [McCluskey84][Udell89][Udell92]. Pseudo-exhaustive testing is based on the observation that, in a real circuit, a single circuit output rarely depends on all of the circuit's inputs. Instead, each output typically depends on only a subset of the inputs. Each such input subset and its corresponding output or outputs form a *segment.* By exhaustively testing each segment, one can exhaustively test the entire circuit [McCluskey85]. The resultant test set, i.e., the set that includes the exhaustive test set for each segment, can be much shorter than an exhaustive test for the entire circuit [Udell89]. Sometimes, by careful arrangement, exhaustive tests for different segments can be carried out simultaneously, thus further reducing the total test time [McCluskey84]. A number of techniques have been developed for generating pseudo-exhaustive test patterns, e.g., [Barzilai83], [Wu90], [Wang86b], [Tang83], and [Akers85].

For some circuits, certain outputs may depend on a large number of inputs. In this case, techniques known as *segmentation* and *partitioning* [Udell89] may be required to divide the CUT into a number of subcircuits each with a limited number of inputs. The segmentation or partitioning of a CUT can be done by either *hardware partitioning* or *sensitized partitioning.* In hardware partitioning, multiplexers (MUX) are inserted into the

CUT for controlling access to the subcircuits [Udell89] [Bhatt86]. In sensitized partitioning [McCluskey81], however, no extra hardware is required since path sensitization is used for the control of the access to the subcircuits. A number of approaches have been developed to partition a CUT, e.g., [Bhatt86], [McCluskey81], [Shperling87] and [Roberts84].

Exhaustive and pseudo-exhaustive testing can achieve high fault coverage. Thus, they are the preferred methods whenever possible [Gelsinger86][Wu90]. However, for many CUTs, the resultant test lengths may still be too long to be feasible. Moreover, circuit partitioning usually introduces hardware and performance penalties. An alternative is *pseudo-random* testing.

2.3.2 Pseudorandom Testing

Instead of using all possible input combinations as test patterns, pseudorandom testing uses only a fraction of all the possible vectors. Pseudorandom testing is based on the observation that, for most CUTs, an effective fault coverage can be obtained if a sufficient number of pseudorandom vectors are applied [Agrawal75][McCluskey85][Bardell87]. The maximum cycle length LFSRs are very good pseudorandom vector generators.

An important problem associated with pseudorandom testing is the existence of the so-called *random-pattern resistant* faults, or simply *random hard* or *hard* faults, in some CUTs. Generally, most faults in a CUT are *random-pattern easy* faults, or simply *random easy* or *easy* faults. These faults can be detected with a relatively small number of random vectors. For the few random hard faults, however, a prohibitively large number of random vectors may be needed for their detection [Muradali89]. The difficulty in detecting such faults results from very poor controllability and/or observability of the circuit nodes where the faults reside. Thus, very few random patterns can both provoke these faults and sensitize their effects to the CUT's outputs.

Techniques have been proposed to deal with random hard faults. A first technique is to generate deterministic test patterns for the few random-pattern resistant faults, and store

them in a ROM. In testing, a few patterns would be read from the ROM, and the remaining patterns would be pseudorandomly generated [Agarwal81][Savir83]. The problem with this method is its high hardware requirements. In [Akers89][Vasudevan93][Hellebrand92], instead of storing these deterministic test patterns in a ROM, a counter or a LFSR is specially designed such that the vectors generated from the counter or LFSR cover these deterministic test patterns. Unfortunately, these methods may either result in a test length similar to that of exhaustive testing, or high silicon area requirements. A second approach to achieve high fault coverage is to modify the circuit so that none of its faults is random-pattern resistant [Eichelberger83][Savir83][Pomeranz92b]. The circuit modification usually implies insertion of extra control and/or observation points into the positions where the random-pattern resistant faults reside. This approach usually adopts analytical testability measures [Savir83][Savir84][Agrawal82] to determine where circuit modifications are necessary. Generally, this method is very effective in improving the circuit's *testability.* However, penalties of using this method includes possible hardware overhead and circuit performance degradation. Another method related to circuit modification is known as *synthesis for testability.* This method takes testability into account when synthesizing a circuit [Devadas88][Rajski90][Rajski92]. All faults in a CUT synthesized in this way are easy faults. A third method coping with random-pattern resistant faults is to modify the pseudorandom pattern generator to produce *weighted random patterns.* The success of weighted random testing relies on the fact that faults that resist detection with patterns having uniform distributions of ones and zeros will not be resistant to patterns with non-uniform distributions [McCluskey85]. Usually, to efficiently reduce test length, more than one set of *weights* are needed [Wunderlich88][Muradali89][Brglez90].

Another difficulty with pseudorandom testing is identifying the random hard faults, and estimating the required test length for a satisfactory fault coverage. A straightforward method to this problem is the use of fault simulation. For large CUTs, however, fault simulation can be expensive due to the prohibitively large number of random vectors that

may need to be simulated. Alternatively, a number of analytical techniques known as *testability analysis* have been proposed [Savir84][Seth85]. These techniques are based on the analysis of the CUT's structure to determine the probability that a given fault can be detected by a randomly selected test vector. For a *fanout-free* CUT, the analysis is easy and accurate. For CUTs with fanout, these techniques are not only expensive in terms of CPU time [McCluskey85][Wagner87], but also inaccurate [Agrawal82][Huisman88].

2.4 Test Response Evaluation

To achieve satisfactory fault coverage with pseudorandom vectors, a large number of test patterns are usually required, in turn producing a large amount of output data to evaluate. To make the output evaluation economical, data compaction or compression is usually required [McCluskey85][Bardell87]. Data compaction and compression are techniques which reduce the amount of data to evaluate, thus rendering its storage and analysis more economical. The difference between compaction and compression is that compaction implies possible error information loss, but compression does not [McCluskey85][Ivanov92]. Therefore, data compression is desirable for BIST. Unfortunately, among the numerous techniques proposed for reducing the amount of BIST output data, there exist only very few schemes that can achieve zero error information loss, e.g. [Gupta90], [Diamantaras91] and [Chakrabarty93], but none of them is practical in terms of hardware requirements. This dissertation generally deals with compaction schemes, as these are more widely-accepted and used.

In general, BIST response evaluation consists of two steps. The first is to compact the test response sequence into a small sequence of bits, called a *signature.* The second step is to compare this final signature with a pre-calculated fault-free signature or reference to determine whether the CUT is good. Theoretically, any finite state machine can be used as a compactor. However, considering the requirements of easy and economical implementation in BIST environments, the most commonly-used data compactors are LFSR-based, binary

counter-based, and CA-based.

Regarding the calculation of the fault-free signature or reference, the most commonly-used method is logic simulation. Logic simulation is a computer simulation process to calculate the output values of a fault-free CUT given a description of the CUT's structure or functionality, and its input vectors. Logic simulation is very fast and of complexity linear in the size of the simulated circuit [Tan87][Keller91]. Having obtained the fault-free output sequence by logic simulation, fault-free signatures can be easily calculated by simulating the data compactor fed by the obtained sequence [Lambidonis91b][Saluja92]. Sometimes, if actual circuits that have been "proven" fault-free, i.e., "golden circuits" are available, one can run these circuits by applying test vectors, collect their output sequences and signatures, and use the obtained signatures as the reference [LeBlanc84][Dervisoglu89]. Another method of signature calculation is an analytical technique, which represents a CUT with a Boolean function in the sum-of-products form, and calculates the sum of the contributions of each product term of this Boolean function to the final signature [Yarmolik92]. It is claimed that this later method would yield much less computational effort as compared to logic simulation. However, this has not been demonstrated.

2.4.1 LFSR-based Data Compaction

LFSR-based data compaction schemes are characterized by the concept of polynomial division described in Section 2.2. These schemes are usually referred to as *signature analysis* [Frohwerk77][Bardell87]. During signature analysis, a l-bit test output sequence is fed to a k-stage LFSR. The final state of the LFSR, or the remainder of the division, forms the signature of the sequence. In this way, a l-bit sequence is compacted into a k-bit signature, where $l >> k$. Due to the feedback of the LFSR, the final value of a k-bit signature is an attribute of the entire output sequence. Though for production testing the ability to distinguish error sequences from the error-free one suffices, diagnosis is also possible with signature analysis [McAnney87][Karpovsky91][Rajski91].

To be effective, signatures generated from error sequences must be different from the fault-free one. In this case, by checking the final signature alone, one can guarantee the identification of the good CUTs. Unfortunately, erroneous sequences mapping to only erroneous signatures is not always the case due to the information loss during the compaction process. For example, it is apparent mathematically that all the polynomials that are multiples of the feedback polynomial will be evenly divided by the feedback polynomial. Thus, all the sequences whose polynomial representation correspond to multiples of the feedback polynomial will produce the same signature since signature analysis is based on polynomial division. More specifically, denote the fault-free sequence by $s[l] = s_1\ s_2\ ...\ s_l$, and let the actual sequence generated from the CUT be $\hat{s}[l] = \hat{s}_1\ \hat{s}_2\ ...\ \hat{s}_l$. Define the error sequence in the error domain to be $e[l] = e_1\ e_2\ ...\ e_l$, where $e_i = s_i \oplus \hat{s}_i$ for $i = 1...l$. It has been proven that all the error sequences whose $e[l]$ is a multiple of the feedback polynomial will yield the same signature as the fault-free one [Ivanov92]. Thus, by checking the signature alone, there exists a possibility that one may mistakenly declare a bad CUT good. Obviously, this possibility must be kept very small to maintain high test quality.

An extreme case of the LFSRs is the *parity checker*, which corresponds to a 1-stage LFSR with feedback polynomial $x + 1$ as shown in Fig. 2.8. When initialized to zero, the final state of the parity checker is zero if the sequence fed to it has an even number of ones. Otherwise, it is one. Therefore, a parity checker guarantees the detection of any error sequence that contains an odd number of errors. However, all the sequences with even number of errors would escape detection. If all the $(2^l - 1)$ possible error sequences of length l are considered, about $\frac{2^l}{2}$ error sequences would go undetected. Since l typically ranges from $10^3 - 10^7$, in terms of test quality, parity checking may not be a very good scheme in general. However, under exhaustive testing, if a CUT is designed such that its fault-free output sequence has an odd parity, parity checking can often achieve very high stuck-at fault coverage [Carter82]. In terms of silicon area, parity checking requires minimal hardware for data compaction [Park91][Wu92b]. It has been shown that LFSRs

with feedback polynomial $(x+1)f(x)$, where $f(x)$ is any polynomial, can also detect any sequence with an odd number of errors [Carter82].

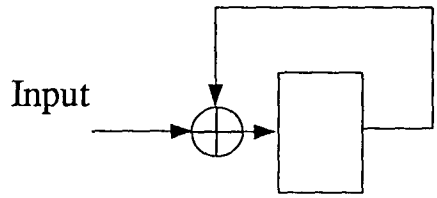


Figure 2.8: A parity checker.

Another commonly used LFSR-based data compaction scheme is the multiple input shift register (MISR) discussed in Section 2.2. It is convenient to use a N-stage MISR if the CUT has N output lines. When N is large, however, the use of a N-stage MISR is not economical [Reddy88][WuM92][Zorian93]. Alternatively, a P-to-S converter followed by a LFSR, or a space compactor followed by a short MISR would be a better choice. Another problem with a MISR is its slightly higher error information loss compared to a corresponding LFSR. This is due to an additional error information loss source known as *error cancellation* in MISRs [Bardell87]. However, error information loss due to error cancellation is usually negligibly small [Bardell87]. Thus, the aliasing performance of a MISR is very similar to that of its corresponding LFSR [Bardell87][Pradhan90][Kameda90].

2.4.2 Counter-based Data Compaction

Counter-based data compaction schemes relies on counting a certain event occurring in the output sequence with the final count forming the signature. There exist many countable events in a sequence [Fujiwara78]. Among these, the easier ones to detect include ones, transitions, and edges. All these events can be described by unified models presented in [Robinson87][Ivanov92b]. A general counter-based compaction scheme consists of an event

detector followed by a counter, as shown in Fig. 2.9. To detect different events, different detectors are needed. Fig. 2.10 shows the detectors for ones and transitions,

Figure 2.9: General counter-based compaction process.

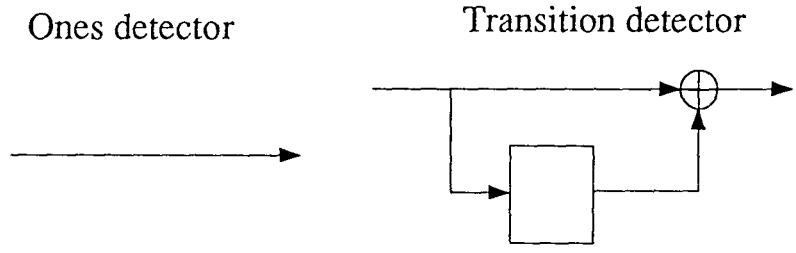


Figure 2.10: Example event detectors.

In general, the counter-based schemes yield higher aliasing than the LFSR-based schemes [Robinson87][Saxena87][Aitken88][Yih91][Pilarski92]. However, by careful arrangement of the test vectors such that the fault-free output sequence is in the form 0 0 ... 0 1 1 ... 1, the transition counting can achieve very small aliasing [Bardell87][Diamantaras91]. Unfortunately, test pattern generators that can generate test vectors in such a order and that require a reasonable amount of hardware have not been found.

Chapter 3

Aliasing and Aliasing Reduction Techniques

As discussed in the previous chapters, output data compaction implies an information loss which translates into the possibility of a wrong diagnosis whereby a faulty circuit declares itself as good. Such incorrect diagnosis is known as aliasing or error masking [McCluskey85]. To assess the overall quality of a BIST scheme, one must take into account the quality of the test vectors as well as the compaction technique. As fault coverage measures are traditionally used to assess the quality of non-BIST schemes, the use of the same measures would be highly desirable in the context of BIST. Unfortunately, as discussed in Section 1.7, the calculation of fault coverage in BIST is generally not computationally feasible for large circuits. As a result, the overall quality of a BIST scheme is usually assessed in two steps. The first is determining the fault coverage before compaction. The second is characterizing the loss of the coverage due to aliasing. Characterizing the fault coverage loss requires *aliasing measures* [Smith80][Cox88]. This chapter first discusses aliasing measures. This is followed by a survey of the advanced compaction techniques intended to reduce aliasing.

3.1 Aliasing Measures and Error Models

3.1.1 Aliasing Measures

Aliasing occurs when an error sequence applied to a data compactor yields the same signature as the fault-free one. To establish the quality of a compaction technique, let F be the set of all possible error sequences that a CUT can produce when a specific set of test vectors are applied to the CUT in a specific order. Let F_0 be the subset of F which contains all

the elements of F that map onto the fault-free signature. The cardinality of F_0 is defined as the *fault domain deception volume* DV_f, i.e., $DV_f = |F_0|$ [Cox88]. Obviously, DV_f is a measure of information loss and hence a measure of aliasing caused by the compaction. In analogy to the definition of fault coverage, $DV_f/|F|$ gives a true measure of the proportion of a CUT's possible error sequences that would alias since both F and DV_f are specific to the CUT and its faults, as well as specific to the test set and the order in which the test vectors are applied.

DV_f and $DV_f/|F|$ are deterministic aliasing measures. They can be directly combined with the fault coverage measures. For example, it is easy to show that the overall fault coverage of a BIST scheme is $1 - \frac{DV_f + \#\ of\ undetectable\ faults}{\#\ of\ possible\ faults\ in\ the\ CUT}$. However, a major difficulty with these measures is the prohibitive computational effort required to find all the elements of F and DV_f for large circuits.

Besides the fault domain, aliasing can also be studied in the error domain [Cox88] [Ivanov92]. Let E be the set of all possible error sequences of length l, where l is the length of the output sequence. Obviously, the cardinality $|E| = 2^l - 1$ since, among the 2^l possible sequences of length l, there exists one that is error-free. Denote the set of all the error sequences that would escape detection due to compaction by E_0. The *Error domain deception volume*, DV_e, is defined to be the cardinality of E_0, i.e., $DV_e = |E_0|$ [Agarwal87][Cox88]. E_0 is a subset of E. Similar to DV_f in the fault domain, DV_e provides an aliasing measure in the error domain. Unlike DV_f, however, DV_e is generally easy to find analytically for most compaction techniques [Williams87][Bardell87][Ivanov88][Ivanov91] [Ivanov92][Ivanov92b][Pilarski92]. For example, assuming the well-known *equally-likely error* model, i.e., assuming that all the 2^l possible sequences are equally-likely to appear yields $DV_e = 2^{l-k} - 1$ for a k-stage LFSR signature analyzer since the compaction process of signature analysis is a linear process that maps all the incoming sequences evenly onto each of its 2^k states [Bardell87]. For a ones counting compactor, $DV_e = \binom{l}{w} - 1$, where w is the weight or number of ones in the fault-free sequence [Bardell87].

Similar to $DV_f/|F|$, $DV_e/|E|$ provides a measure of the proportion of all the error sequences that would alias. Unlike $DV_f/|F|$, however, $DV_e/|E|$ is a probabilistic measure. Usually $|F| << |E|$ for a specific CUT. Both E and DV_e are independent of the CUT, the test vectors, and the order in which the test vectors are applied. The measure $DV_e/|E|$ is usually called the *probability of aliasing*, and denoted by P_{al}, i.e., $P_{al} = \frac{DV_e}{|E|}$ [Bardell87][1].

Considering $|E| = 2^l - 1$, for the k-stage signature analyzer for instance, the aliasing probability $P_{al} = \frac{2^{l-k}-1}{2^l-1}$. Generally since $2^l >> 1$, $P_{al} \approx 2^{-k}$. This is a very well-known result for signature analysis.

In practice, due to the prohibitive costs of obtaining F and hence DV_f, P_{al} is traditionally used, although not always explicitly, for assessing the aliasing problem of data compaction. P_{al} is a probabilistic measure of aliasing. Thus, it cannot be directly combined to the classical fault coverage measures [Cox88]. Although an experimental report [Rajski91b] has shown that the overall fault coverage of a BIST scheme can be well estimated by $(1 - P_{al})FC$ in general, where FC is the fault coverage achieved before data compaction, the aliasing measure P_{al} is still difficult to use confidently when dealing with a specific CUT tested by a specific set of test vectors applied in a specific order because of the statistical uncertainty [Lambidonis91]. Evidence of this uncertainty can be found in any experimental reports on aliasing, e.g., [Aitken89], [Xavier92], [Rajski91b] and [Debaney92].

3.1.2 Error Models

Studying aliasing in the error domain requires an assumption or model of the distribution of the error sequences. A simple error model is the equally-likely model discussed Section 3.1.1 [Bardell87]. It assumes that, in the presence of a fault, any of the $2^l - 1$ error sequences has precisely the same likelihood of occurrence. This model is easy to use. However, its use implies the denial of any circuit information. It leads to the simple result $P_{al} \approx 2^{-k}$, irrespectively of the specific CUT, the effects its faults may produce at its outputs, the

[1] Note that there are other definitions of aliasing probability. See [Williams86] and [Ivanov92] for details.

number and order of the test vectors applied to it, and even the feedback polynomial of the LFSR itself. Obviously, a model of this kind is not realistic [Williams87][Ivanov88][Aitken89] [Ivanov92]. More sophisticated error models have been developed, e.g.,

- the stationary independent error model [Williams86], which assumes that, in the presence of a fault, each sequence bit has a same probability to be in error, i.e., $Pr[e_i = 1] = p$, where $i = 1, ..., l$ and $0 \leq p \leq 1$. The probability that a bit is correct is $1 - p$, i.e., $Pr[e_i = 0] = 1 - p$ for $i = 1, ..., l$. p is the detectability of the fault. This model is a generalization of the earlier equally-likely model in that the equally-likely model is a special case of the stationary independent model with $p = 0.5$.

- the non-stationary independent error model [Ivanov88], which generalizes the stationary model in that the probability of error of the output bits may vary from bit to bit for example due to changes in the distribution of input test vectors. In this model, $Pr[e_i = 1] = p_i$ and $Pr[e_i = 0] = 1 - p_i$, where $0 \leq p_i \leq 1$ and $0 \leq i \leq l$. The above stationary model is a special case with $p_1 = p_2 = ... = p_l$.

- the asymmetric error model [Aitken89], which associates the error probability of a bit with the fault-free value of the bit. The error bits are also assumed uncorrelated, however, characterized by two possibly different conditional probabilities, depending on the value of the fault-free bit. Specifically, the probability that a bit be in error when the fault-free value of the bit is 1 is p, i.e., $Pr[e_i = 1|s_i = 1] = p$, while the probability that a bit be in error when the fault-free value of the bit is 0 is q, i.e., $Pr[e_i = 1|s_i = 0] = q$, $0 < p < 1$, $0 < q < 1$.

Similar error models for MISRs have also been developed, e.g., [Pradhan90], [Kameda93], and [Karpovsky91]. Unlike the equally-likely error model, the models discussed above incorporate certain degrees of circuit dependencies. Thus, they are considered to be more realistic than the equally-likely model, although finding precise values of the bit error probabilities, e.g., p, p_i, and q, is in itself a difficult task. Methods for these "more" realistic

models have been proposed for the calculation of "exact" aliasing probabilities for arbitrary test lengths and LFSRs, MISRs and binary counters, e.g., [Williams86], [Ivanov88], [Damiani89], [Pradhan90], [Karpovsky91], [Ivanov92b], and [Pilarski92]. Although these methods are generally of high computational complexity [Saxena92], they can accurately estimate aliasing probability P_{al}, especially for short test lengths.

Surprisingly, however, as long as the signature analyzer has an *irreducible* feedback polynomial, the aliasing probability obtained under all these "more realistic" models also converges to 2^{-k} asymptotically when test length $l \rightarrow \infty$, irrespectively of the circuit dependencies [Williams89][Damiani89b][Pradhan90][Karpovski91][Damiani91][Ivanov92] [Pilarski92b]. The only difference due to the use of different signature analyzers is that those with primitive polynomials converge to 2^{-k} faster than others [Williams86][Ivanov92]. For specific situations, however, no one is guaranteed to be superior over the others [Ahmad90]. Thus, it is suggested to use rather complex primitive feedback polynomials since it is believed that complex polynomials would prevent masking by anything but an equally complex error sequence [Bardell87][Dervisoglu89].

Due to the fact that all error models converge to the asymptotic result 2^{-k} when feedback polynomials are irreducible, in practice, one can safely assume $P_{al} \approx 2^{-k}$ without sacrificing confidence since l is usually sufficiently long in reality. Regarding counter-based compaction techniques, recent researches showed that their aliasing probabilities also converge to 2^{-k} as $l \rightarrow \infty$ for combinational faults, where k is the length of the binary counter [Ivanov92b][Pilarski92]. However, the rate at which they converge to 2^{-k} is much slower as compared to the LFSR-based techniques [Pilarski92]. Other studies also showed that the counter-based techniques generally yield poorer aliasing performance than the LFSR-based techniques [Robinson87][Saxena87][Aitken88][Yih91][Pilarski92].

3.2 Advanced Compaction Techniques

The aliasing performance of the adopted compaction technique has a significant impact on the final test quality. A k-stage LFSR signature analyzer can achieve an aliasing probability of 2^{-k}. If $k = 16$, $P_{al} \approx 2^{-16} = \frac{1}{65536} = 0.000005$. This number may seem very small, and is in fact typically accepted in practice [LeBlanc84][Kuban84][Gelsinger86]. However, for large VLSI circuits and high quality demands, it has been argued that $P_{al} \approx 2^{-16}$ is far from adequate [Agarwal83][Zorian86]. By increasing k, one can easily reduce this number. However, the impact on hardware requirements is substantial. For BIST applications, the test circuitry's hardware requirement must be taken into account seriously since higher hardware requirement implies not only higher cost, but also lower yield [McCluskey85], lower reliability [McCluskey85], and higher power consumptions [Levy91]. In the past decade, significant efforts have been made toward improving the aliasing performance of data compactors while maintaining reasonable hardware requirements, and many advanced data compaction techniques have been proposed. Some of these are briefly surveyed next.

3.2.1 Multiple Signature Analysis

Multiple LFSRs Signature Analysis

A straightforward way to check multiple signatures is to employ two or more LFSRs, each with a different feedback polynomial [Bhavsar84]. Fig. 3.11 shows a scheme employing two compactors with polynomials $f_1(x)$ and $f_2(x)$. Obviously, the error sequences aliased by the multiple LFSR compaction scheme are those that are aliased by both $f_1(x)$ and $f_2(x)$. The aliasing probability of this scheme $P_{al} = P_1 P_2$ [Bhavsar84], where P_1 and P_2 are the aliasing probabilities of the two signature analyzers. It would be desirable to choose $f_1(x)$ and $f_2(x)$ such that the error sequences aliased by $f_1(x)$ and those aliased by $f_2(x)$ be disjoint. Unfortunately, such $f_1(x)$ and $f_2(x)$ do not exist [Bhavsar84]. Assuming that the lengths of $f_1(x)$ and $f_2(x)$ are k_1 and k_2, respectively, yields $P_1 \approx 2^{-k_1}$ and $P_2 \approx 2^{-k_2}$.

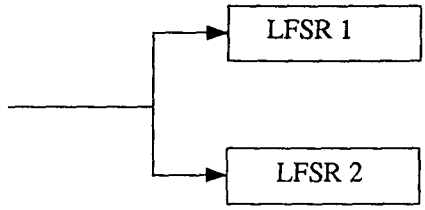


Figure 3.11: Multiple LFSRs signature analysis.

Then, $P_{al} \approx 2^{-k_1}2^{-k_2} = 2^{-(k_1+k_2)}$. Apparently, P_{al} for two LFSRs is equivalent to that of using a single $(k_1 + k_2)$-stage LFSR. In terms of aliasing performance against hardware requirements, this scheme does not achieve any improvement over simply increasing the LFSR length in the conventional single signature analysis scheme.

Alternatively to using multiple LFSRs, different types of data compaction techniques can be combined [Robinson87]. In [Robinson87][Robinson88], a scheme that simultaneously uses a LFSR and a counter was proposed. Its P_{al} is equal to the product of the aliasing probabilities of the LFSR and the ones counter. As compared to using two LFSRs, it is less attractive. This is because it requires a similar amount of silicon area, but its aliasing probability is higher on average since the aliasing probability of a ones counter is in general higher than that of a LFSR.

Multiple Test Sets Signature Analysis

Another method to check multiple signatures is to employ several test sets [Bhavsar84] [Hassan84]. This scheme requires only a single LFSR. A signature collected by the LFSR is checked after each test set has been applied to the CUT. Assume that two test sets, T_1 and T_2, are applied. Let S_1 and S_2 be the fault-free signatures obtained under T_1 and T_2, respectively. In the occurrence of a fault, the probability that it would be aliased by this scheme is the probability that its signature under T_1 matches S_1, and its signature under T_2 matches S_2. Assuming the length of the LFSR to be k yields $P_{al} \approx 2^{-k}2^{-k} = 2^{-2k}$

[Bhavsar84].

Multiple test sets signature analysis uses a single LFSR. Its extra hardware requirement compared to single signature analysis is the silicon area for storing the additional reference signatures. An extra advantage of multiple test sets signature analysis is the possibly higher fault coverage, especially for unmodeled faults, due to the use of more test vectors. However, the use of multiple test sets not only significantly increases test time, but also implies a more complex test pattern generator. In [Hassan84], instead of using multiple test sets, a single test set is used twice in different orders, thus making test generation easier.

Multiple Intermediate Signature Analysis

Multiple intermediate signature analysis is sometimes called *split* or *segmented* sequence testing [Bhavsar84][Bardell87]. The basis of this scheme is to sample intermediate signatures collected by a single LFSR in addition to checking the final one. For example, if we want to check a total of two signatures, we can check one intermediate signature after the first l_1 bits of the sequence have been shifted into the signature analyzer, and check the second signature after the entire sequence of length l has been compacted. Assume the two corresponding good signatures to be S_1 and S_2. The aliasing probability in this case is equal to the probability that an error sequence produce S_1 when the first signature is checked, and S_2 when the second one is checked. As will be shown in the next chapter, for this scheme $P_{al} \approx 2^{-2k}$.

Like the multiple test sets scheme, multiple intermediate signature analysis uses a single LFSR. Unlike the multiple test sets scheme, however, this scheme uses only one test set. Therefore, this scheme will not incur any test time increase. On the contrary, it can significantly reduce test time on average. This is because a test session can be terminated and the CUT declared faulty once an incorrect intermediate signature is found [Lee88]. In addition to reducing aliasing and test time, recent studies have demonstrated many

other advantages of this scheme, e.g., easier fault coverage computation [Lambidonis91], increased fault diagnosability [Waicukauski87], and possible zero aliasing for modeled faults [Pomeranz92]. Like all the multiple signature analysis techniques, the disadvantages of this scheme include the increased implementation and control complexities, and increased hardware requirements for storing the multiple reference signatures.

3.2.2 Output Data Modification (ODM)

ODM is a counter-based compaction scheme that takes advantage of counter-based techniques' non-uniformity of aliasing. As shown earlier, a ones counter's deception volume is $DV_e = \binom{l}{w}$, where l and w are the length and weight of a fault-free sequence. For a given l, DV_e is a function of w, which is depicted in Fig. 3.12. If the sequence weight w is

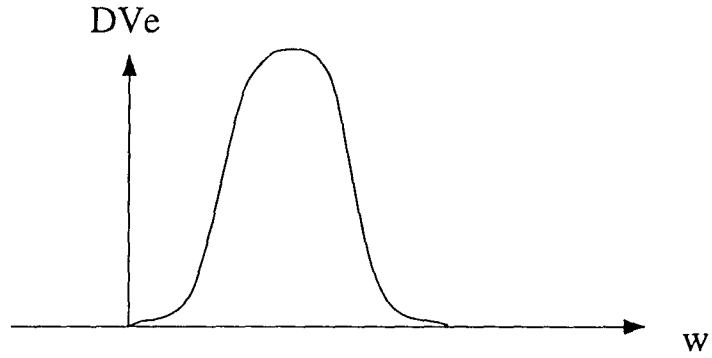


Figure 3.12: Non-uniform deception volume of counter-based schemes.

either very low or very high, the DV_e can be extremely small. The basic concept of ODM is to reduce the weight of the fault-free output sequence, and then use ones counting as a signature of the "modified" sequence. One method to do this is to generate a reference sequence such that the sequence obtained by bit-wise XOR of the reference sequence and the actual fault-free sequence yields a reduced weight [Agarwal83][Zorian86][Agarwal87]. This is illustrated in Fig. 3.13. Ideally, the reference sequence is identical to the fault-free sequence, thus making $DV_e = 0$. Unfortunately, the generation of such an ideal sequence

may generally require too much silicon area. Therefore, in general, only a proper reference sequence can be generated for the purpose of ODM. In [Agarwal87], a systematic approach for generating such reference sequences was proposed. Although it claims to reduce a ones counter's aliasing by a factor of $2^{hundreds}$ or even $2^{thousands}$, the corresponding hardware requirement is generally considerable.

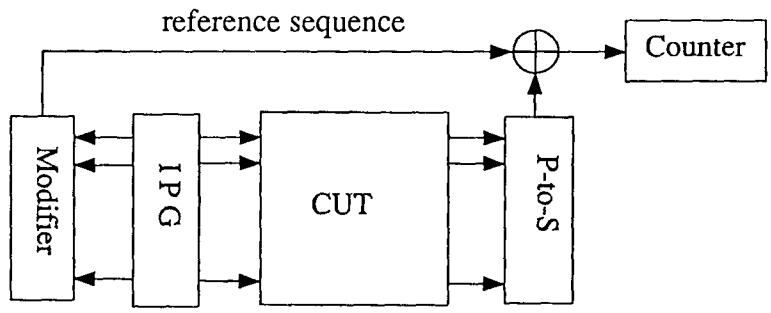


Figure 3.13: An ODM scheme.

In [Li87], a different way of implementing ODM was proposed, where the output sequence modification is carried out in the process of space compaction. However, it does not provide any general method to reduce the aliasing to meet a given requirement. In [Zorian92], the concept of ODM is applied to ROM testing. Instead of generating the reference sequence on-chip, the reference sequence is pre-calculated and stored in an extra column of the ROM under test.

3.2.3 Zero Aliasing Techniques

Zero Aliasing by Monitoring the Quotient Sequence

Signature analysis is based on the concept of polynomial division. Denote the feedback polynomial by $f(x)$, the sequence to be compacted by $R_l(x)$. Signature analysis can be represented as

$$R_l(x) = q(x)f(x) + r(x), \tag{3.1}$$

where $q(x)$ is the quotient of the division, which is also the sequence shifted out from the last stage of the LFSR; and $r(x)$ represents the remainder taken as a signature. Conventionally, the signature obtained above is compared with a predetermined fault-free signature given by

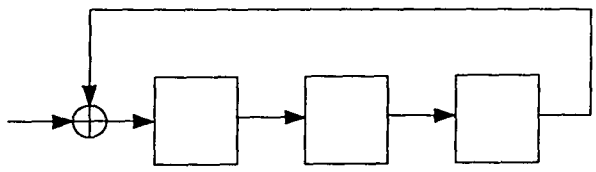
$$R_l^*(x) = q^*(x)f(x) + r^*(x), \tag{3.2}$$

where $R_l^*(x)$ represents the good circuit response.

Aliasing arises when $r(x) = r^*(x)$ but $R_l(x) \neq R_l^*(x)$. Obviously, this problem is caused by the information loss due to the discarding of the quotient sequences. Thus, if $q(x)$ and $q^*(x)$ are compared as well, i.e., $r(x)$ and $q(x)$ compared with $r^*(x)$ and $q^*(x)$, respectively, aliasing can be avoided. In general, directly storing the quotient sequences is not practical in BIST. In [Gupta90], instead of storing the quotient sequences, a new approach was proposed to select a specific $f(x)$ for a given $R_l^*(x)$ such that $q^*(x)$ is periodic. By being periodic, it is easy to use a simple periodic sequence checker to determine whether the actual quotient sequence from the LFSR is error-free. For instance, in the example shown in Fig. 3.14 where the error-free quotient sequence is all 1, the periodic sequence checker is simply a 0-detector. In the case shown in Fig. 3.14, if the signature obtained is 010, and the quotient obtained is all 1, the sequence under compaction is fault-free. Otherwise, it is faulty. Here no aliasing can occur since both $q(x)$ and $r(x)$ are monitored. Unfortunately, this data compaction scheme generally requires too much hardware for practical use. For example, it was shown in [Gupta90] that the required LFSR that corresponds to the selected $f(x)$ can have $\frac{l}{2}$-stages if the sequence to compact is of length l. Since l is usually at the order of hundreds of thousands or even millions, this makes this technique clearly not feasible.

Zero Aliasing by Transition Counting

Another zero aliasing technique was proposed in [Diamantaras91]. It is based on transition counting, whose deception volume $DV_e = \binom{l}{t} - 1$, with l and t being the length and number



Rl(x) = 101000111 r(x) = 010 q(x) = 111111

Figure 3.14: An example of zero aliasing compaction.

of transitions of the fault-free sequence [Bardell87]. If all the test vectors that produce 0s at the fault-free CUT's output are applied first, and the others are applied afterwards. This yields a fault-free output sequence in the form 00...0011...11. There is only a single transition in the sequence. Thus, the aliasing is very small, but still non-zero. In [Diamantaras91], a special 0-detector, as shown in Fig. 3.15, was proposed to monitor the output sequence. The switch shown in Fig. 3.15 is first set to the inverting terminal, and all the test vectors that would produce 0s at the CUT's output are applied. If a fault is detected by some of these vectors, the CUT would output some 1s. In this case, the 0-detector will detect the possible errors. After applying the test patterns that produce 0s at the CUT's output, the switch is set to the non-inverting terminal, and all the vectors that have the CUT output 1s in the absence of faults are applied. Similarly to the earlier case, any error, i.e., 0 in this case, in the output sequence will be caught by the 0-detector. This scheme requires little hardware on the data compaction side. However, a major problem exists with the test pattern ordering. Small hardware requirement test pattern generators that can produce test vectors in such an order have not yet been found yet. Another drawback of reducing transition count is that it also reduces the effectiveness of the test vectors for testing AC faults. This is because AC fault testing usually requires the generation of a large number of transitions at the fault-free circuits' output lines.

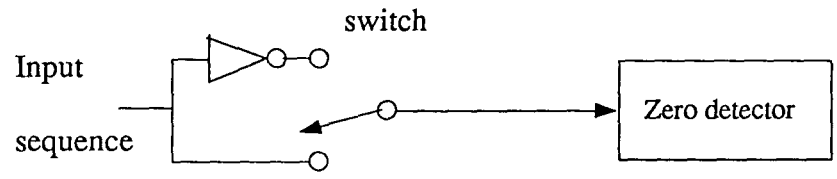


Figure 3.15: Zero aliasing transition counting.

Zero Aliasing for Modeled Faults

During signature analysis, the first error in an error sequence will always be captured by the signature analyzer LFSR [Bardell87]. However, as the compaction process carries on, the error captured in the LFSR might be cancelled by some other errors afterwards, thus causing aliasing. One technique described in [Pomeranz92] proposes to check an intermediate signature before a captured error is cancelled. Therefore, for each possible error sequence, if at least one of its errors can be detected by checking intermediate signatures, aliasing can be eliminated. To find the possible error sequences, fault simulation assuming a certain fault model must be used. Thus, unlike the zero aliasing techniques discussed earlier, this technique is only valid for modeled faults. To reduce the hardware requirements, careful scheduling of the intermediate signatures is achieved using an algorithm developed in [Pomeranz92] that minimizes the total number of required signatures. In general, the signatures are not periodically scheduled. Thus, besides the silicon area required for storing the reference signatures, the control for the signature checkings may be complex [Wu92c].

Zero Aliasing by Sequence Identification

Given a fault-free sequence, a nonlinear machine can be designed to identify the sequence. Thus, if such a machine is used for data compaction, no aliasing will occur. Such a technique was proposed in [Chakrabarty93]. However, a major problem of this method is that no upper bound on hardware requirements has been derived so far for long output sequences.

Therefore, in general, this method is not practical.

3.2.4 Modified LFSR

Recently, a modified LFSR (M-LFSR) was proposed to reduce aliasing [Raina91]. The M-LFSR has two modes. One is the regular LFSR mode. The other is the modified LFSR mode which converts the LFSR into a non-linear machine. The basic idea of this approach is to use the normal LFSR to compact the first $(l-s)$ bits of the sequence, and then use the non-linear machine to recognize the remaining s bits. If $s = 1$, i.e., if the last one sequence bit can be identified, the M-LFSR can reduce a LFSR's aliasing by half. This is because half of the 2^l possible sequences have a last bit with a opposite value to the fault-free one. By identifying the last bit alone, half of the 2^l error sequences can be detected. Regarding the remaining $2^{l-1} - 1$ error sequences, they would be detected as in regular signature analysis. Thus, the deception volume of the M-LFSR scheme is $DV_e = 2^{l-k-1} - 1$. Compared to $2^{l-k} - 1$, which is the deception volume of a k-bit LFSR, the M-LFSR yields half the aliasing. In general, if the last s bits of the sequence can be identified, the M-LFSR scheme can reduce a LFSR's aliasing by a factor of 2^s. According to [Raina91], for a r-input k-stage LFSR, the M-LFSR can achieve the following aliasing:

$$P_{M-LFSR} = \frac{1}{2^k}(1-p)^{rs}, \qquad r \leq k, \tag{3.3}$$

where p is the probability for a bit to be in error given a CUT failure, independently of the other bit errors. For the equally likely error model, $p = 0.5$, which is also assumed in the experiments in [Raina91]. Assuming $p = 0.5$ yields:

$$P_{M-LFSR} = 2^{-(k+rs)}, \qquad r \leq k. \tag{3.4}$$

The worst case of the M-LFSR is when $r = 1$, which corresponds to a single input LFSR. In this case, $P_{M-LFSR} = 2^{-(k+s)}$. The best case occurs when $r = k$, which corresponds to

a k-input MISR. In the best case, $P_{M-LFSR} = 2^{-k(s+1)}$.

In terms of hardware requirements, the M-LFSR requires $\lceil \frac{s}{3}(8 + r + k) \rceil$ 4-input NAND/NOR gates, which is equivalent to $\lceil s(8 + r + k) \rceil$ 2-input NAND/NOR gates since each 4-input gate requires the same hardware as three 2-input gates. In general, the hardware requirement of this method is high. For example, to reduce the aliasing probability from 2^{-16} to 2^{-32}, the M-LFSR requires 400 to 600 2-input gates in addition to a 16-stage LFSR.

Chapter 4

Multiple Intermediate Signature Analysis — I

As discussed in the preceding chapter, aliasing reduction is usually achieved at the cost of high hardware requirements. Furthermore, except for the zero aliasing schemes, the difficulty for test quality assessment associated with data compaction (see Section 1.7) still remains. Unfortunately, the zero aliasing schemes are not generally feasible in terms of silicon area. Among the non-zero aliasing schemes, multiple intermediate signature analysis seems to be superior over the others since recent work has shown that this scheme lends itself well to exact fault coverage computation [Lambidonis91] and is able to significantly reduce aliasing with little extra hardware cost [Wu93]. Furthermore, multiple intermediate signature analysis can also shorten average test time [Lee88] and increase fault diagnosability [Waicukauski87].

In this chapter, we will investigate the models for predicting the aliasing and fault coverage of multiple intermediate signature analysis. In the remainder of this dissertation, unless otherwise stated, multiple intermediate signature analysis is simply referred to as *multiple signature analysis.*

If a total of two signatures are checked, the sequence is divided into two *segments* of length l_1 and $l_2 = l - l_1$, and the signature analyzer is checked at the end of each segment, i.e., the first signature is checked after the first l_1 bits of the sequence have been compacted and the second signature is checked after the entire sequence has been compacted. In general, assume that a total of n signatures, S_1, S_2, ..., S_n, are checked after the first l_1, l_2, ..., l_n ($l_n = l$) bits of the sequence have been shifted into the signature analyzer, respectively. In the remainder of this dissertation, the positions l_i where the signatures are checked are

referred to as *check points*, while the subsequence between each pair of adjacent check points is referred to as a *segment.*

4.1 An Aliasing Model

In multiple signature analysis, the error sequences that escape detection are those that escape the detection at each intermediate and final signature checking. For example, assume an output sequence of length l is compacted by a k-stage LFSR. Let the sequence be divided into two *segments* of lengths l_1 and $l_2 = l - l_1$, and let the LFSR be checked at the end of each segment, i.e., at the check points l_1 and l. Assume the corresponding fault-free signatures of the two segments are S_1 and S_2. Under the equally-likely error model, the number of possible sequences that produce S_1 at l_1 is $\frac{2^{l_1}}{2^k}$, while the number of sequences that produce S_2 at l is $\frac{2^{(l-l_1)}}{2^k}$ [Bhavsar84][Bardell87], where both l_1 and $l-l_1$ are greater than k. Therefore, the deception volume in this case, denoted by $DV_e(2)$, is [Bhavsar84][Bardell87]:

$$\begin{aligned} DV_e(2) &= \frac{2^{l_1} \times 2^{(l-l_1)}}{2^{2k}} - 1 \\ &= 2^{l-2k} - 1. \end{aligned} \tag{4.5}$$

Considering the existence of the $2^l - 1$ possible error sequences yields the aliasing probability $P_{al}(2)$:

$$P_{al}(2) = \frac{2^{l-2k} - 1}{2^l - 1}. \tag{4.6}$$

Obviously, when $2^l >> 1$, $P_{al}(2) \approx 2^{-2k}$.

In general, if n signatures are checked at the arbitrary positions $l_1, l_2, ..., l_n$, with the constraint $l_i > l_j$ and $l_i - l_j > k$ for $i > j$ and $l_n = l$, it can be shown [Bhavsar84][Bardell87] that its deception volume is:

$$DV_e(n) = 2^{l-nk} - 1. \tag{4.7}$$

The corresponding aliasing probability is:

$$P_{al}(n) = \frac{2^{l-nk} - 1}{2^l - 1}. \tag{4.8}$$

Assuming $2^l >> 1$ yields:

$$P_{al}(n) \approx 2^{-nk}. \tag{4.9}$$

From Eqns. 4.7 - 4.9, the deception volume associated with multiple signature analysis, and the corresponding aliasing probability, are independent of the positions where the signatures are checked.

4.2 Fault Coverage Models

In reality, some error sequences are more likely to occur than others. For example, if two signatures are checked at the check points l_1 and l, the error sequences in the form $e[l] = 00...0e_{l_1+1}e_{l_1+2}...e_{l_n}$ would occur more often than those in the form $e[l] = e_1e_2...e_{l_1}00...0$. This is because most faults that produce errors in the first segment are random-easy faults, and thus would also have good chance producing errors during the second segment, thus generating the error sequence in the form of $e[l] = e_1e_2...e_{l_1}e_{l_1+1}...e_l$. To more precisely predict the aliasing performance, this section discusses fault coverage models that take into account the detection probability profile of a CUT.

4.2.1 Preliminaries

This section introduces definitions and briefly reviews some results presented in [Seth90] and [Rajski91b].

Basic Definitions

Definition 4.1: The *detection probability* of a fault is the probability of detecting the fault by applying a random vector to the CUT.

Detection probabilities of faults in a CUT can be represented by a probability density function $p(x)$ such that $p(x)dx$ corresponds to the fraction of testable faults with detection probability between x and $x + dx$. Since x represents a probability:

$$\int_0^1 p(x)dx = 1. \tag{4.10}$$

Fault Coverage of Random Vectors without Compaction

Since there are $p(x)dx$ faults with detection probability x, the mean coverage among these faults by a random vector is $xp(x)dx$. Suppose we apply a sequence of random vectors to the circuit. The mean coverage by the first vector is:

$$y_1 = \int_0^1 xp(x)dx. \tag{4.11}$$

The actual coverage by a random vector might be different from the mean by a random quantity. However, the variance will be small for almost all circuits [Seth90]. If all faults are assumed testable, after removing the faults detected by the first vector, the normalized number of the remaining undetected faults is:

$$\begin{aligned} UDT &= 1 - \int_0^1 xp(x)dx \\ &= \int_0^1 (1-x)p(x)dx. \end{aligned} \tag{4.12}$$

Thus, the distribution of the detection probabilities of the remaining undetected faults is $(1-x)p(x)$. Hence, the coverage of two random vectors is:

$$\begin{aligned} y_2 &= y_1 + \int_0^1 x(1-x)p(x)dx \\ &= \int_0^1 x[1+(1-x)]p(x)dx. \end{aligned} \tag{4.13}$$

Similarly, the coverage of l vectors is [Seth90]:

$$\begin{aligned} y_l &= \int_0^1 x[1 + (1-x) + (1-x)^2 + ... + (1-x)^{l-1}]p(x)dx \\ &= 1 - \int_0^1 (1-x)^l p(x)dx \\ &= 1 - I(l), \end{aligned} \tag{4.14}$$

where $I(n) = \int_0^1 (1-x)^l p(x)dx$.

Eqn. 4.14 shows the fault coverage achieved with l random vectors without data compaction. Obviously, when $l \to \infty$, $y_l \to 1$. That is, one can detect all testable faults in a CUT if a sufficiently large number of vectors are applied[1].

Fault Coverage with Single Signature Analysis

Now assume a k-bit signature is checked after applying l random vectors to a CUT. Denote the aliasing probability by ρ, i.e., $\rho \approx 2^{-k}$. Denote the probability of no aliasing by β, i.e., $\beta = 1 - \rho$. According to [Rajski91b], the fault coverage with the single signature analysis can be well estimated by $\beta \times FC = (1-\rho)FC$, where FC is the fault coverage before data compaction. Thus, from Eqn. 4.14, the fault coverage of single signature analysis can be represented as:

$$\begin{aligned} FC_1 &= \beta[1 - \int_0^1 (1-x)^l p(x)dx] \\ &= \beta(1 - I(l)). \end{aligned} \tag{4.15}$$

4.2.2 A Comprehensive Fault Coverage Model

In the following, for better understanding the fault coverage model derivation, we first label the aliasing probability of the ith signature checked at check point l_i by ρ_i, and hence the probability of no-aliasing by $\beta_i = 1 - \rho_i$, where $i = 1, ..., n$ and $l_n = l$. Then, at the end of

[1]Note that, for simplicity, Eqn. 4.14 considers only testable faults [Seth90].

this section, we remove the subscript i by defining $\rho = \rho_i = 2^{-k}$ for $i = 1, ..., n$. Similar to the analysis in Sec. 4.2.1, the fault coverage after checking the first signature is:

$$\begin{aligned} FC_1 &= \beta_1[1 - \int_0^1 (1-x)^{L_1} p(x)dx] \\ &= \beta_1[1 - I(L_1)]. \end{aligned} \tag{4.16}$$

The portion of the faults that remain undetected after checking the first signature is:

$$\begin{aligned} UDT_1 &= 1 - FC_1 \\ &= 1 - \beta_1(1 - \int_0^1 (1-x)^{L_1} p(x)dx) \\ &= \int_0^1 [1 - \beta_1 + \beta_1(1-x)^{L_1}]p(x)dx. \end{aligned} \tag{4.17}$$

Therefore, the new distribution of the detection probabilities of the remaining faults is $p_1(x) = [1 - \beta_1 + \beta_1(1-x)^{L_1}]p(x)$. The fault coverage after checking the second signature at check point l_2 is:

$$\begin{aligned} FC_2 &= FC_1 + \beta_2[UDT_1 - \int_0^1 (1-x)^{L_2} p_1(x)]dx \\ &= FC_1 + \beta_2\{UDT_1 - \int_0^1 (1-x)^{L_2}[1 - \beta_1 + \beta_1(1-x)^{L_1}]p(x)dx\} \\ &= [\beta_2 + \beta_1(1-\beta_2)] - [\beta_1(1-\beta_2)I(L_1) + \beta_2(1-\beta_1)I(L_2)] - \beta_1\beta_2 I(L_1 + L_2). \end{aligned} \tag{4.18}$$

Similarly, we can find the distribution of the detection probabilities of the remaining faults, and the fault coverage after checking the nth signature:

$$\mathrm{FC_n} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \beta_i \prod_{\mathrm{j=i+1}}^{\mathrm{n}} \rho_j - \prod_{\mathrm{i=1}}^{\mathrm{n}} \rho_i \sum_{\mathrm{j=1}}^{\mathrm{n}} [\sum_{\mathrm{m_1=1}}^{\mathrm{n-j+1}} \frac{\beta_{\mathrm{m_1}}}{\rho_{\mathrm{m_1}}} \sum_{\mathrm{m_2=m_1+1}}^{\mathrm{n-j+2}} \frac{\beta_{\mathrm{m_2}}}{\rho_{\mathrm{m_2}}} \cdots \sum_{\mathrm{m_j=m_{j-1}+1}}^{\mathrm{n}} \frac{\beta_{\mathrm{m_j}}}{\rho_{\mathrm{m_j}}} \mathrm{I}(\sum_{\mathrm{g=1}}^{\mathrm{j}} \mathrm{L_{m_g}})], \tag{4.19}$$

where $m_0 = 0$, $\sum_a^b = 0$ and $\prod_a^b = 1$ if $a > b$.

As an example, the fault coverage expression for checking three signatures is given below:

$$
\begin{aligned}
FC_3 &= [\sum_{i=1}^{3}\beta_i \prod_{j=i+1}^{3}\rho_j] - \rho_1\rho_2\rho_3[\sum_{m_1=1}^{3}\frac{\beta_{m_1}}{\rho_{m_1}}I(L_{m_1}) + \sum_{m_1=1}^{2}\frac{\beta_{m_1}}{\rho_{m_1}}\sum_{m_2=m_1+1}^{3}\frac{\beta_{m_2}}{\rho_{m_2}}I(\sum_{g=1}^{2}L_{m_g}) \\
&\quad + \sum_{m_1=1}^{1}\frac{\beta_{m_1}}{\rho_{m_1}}\sum_{m_2=m_1+1}^{2}\frac{\beta_{m_2}}{\rho_{m_2}}\sum_{m_3=m_2+1}^{3}\frac{\beta_{m_3}}{\rho_{m_3}}I(\sum_{g=1}^{3}L_{m_g})] \\
&= [\beta_1\rho_2\rho_3 + \beta_2\rho_3 + \beta_3] - \rho_1\rho_2\rho_3\{[\frac{\beta_1}{\rho_1}I(L_1) + \frac{\beta_2}{\rho_2}I(L_2) + \frac{\beta_3}{\rho_3}I(L_3)] \\
&\quad + [\frac{\beta_1\beta_2}{\rho_1\rho_2}I(L_1+L_2) + \frac{\beta_1\beta_3}{\rho_1\rho_3}I(L_1+L_3) + \frac{\beta_2\beta_3}{\rho_2\rho_3}I(L_2+L_3)] + [\frac{\beta_1\beta_2\beta_3}{\rho_1\rho_2\rho_3}I(L_1+L_2+L_3)]\} \\
&= [\beta_1\rho_2\rho_3 + \beta_2\rho_3 + \beta_3] - \{[\beta_1\rho_2\rho_3 I(L_1) + \rho_1\beta_2\rho_3 I(L_2) + \rho_1\rho_2\beta_3 I(L_3)] \\
&\quad + [\beta_1\beta_2\rho_3 I(L_1+L_2) + \beta_1\rho_2\beta_3 I(L_1+L_3) + \rho_1\beta_2\beta_3 I(L_2+L_3)] \\
&\quad + [\beta_1\beta_2\beta_3 I(L_1+L_2+L_3)]\}. \qquad (4.20)
\end{aligned}
$$

Assuming each signature to be of the same length, i.e., $\rho_i = 2^{-k}$, and hence $\beta_i = (1-2^{-k})$, for $i = 1, ..., n$, yields:

$$
\begin{aligned}
FC_n &= (1-2^{-k})\sum_{i=1}^{n}2^{-(n-i)k} \\
&\quad - \sum_{j=1}^{n}[(1-2^{-k})^j 2^{-(n-j)k}\sum_{m_1=1}^{n-1+1}\sum_{m_2=m_1+1}^{n-j+2}\cdots\sum_{m_j=m_{j-1}+1}^{n} I(\sum_{g=1}^{j}L_{m_g})]. \qquad (4.21)
\end{aligned}
$$

If $2^{-k} << 1$, we have:

$$
FC_n \approx 1 - \sum_{j=1}^{n}[2^{-(n-j)k}\sum_{m_1=1}^{n-j+1}\sum_{m_2=m_1+1}^{n-j+2}\cdots\sum_{m_j=m_{j-1}+1}^{n} I(\sum_{g=1}^{j}L_{m_g})]. \qquad (4.22)
$$

For example, when $n = 3$ and $2^{-k} << 1$, the above equation yields:

$$
FC_3 \approx 1 - \sum_{j=1}^{3}[2^{-(3-j)k}\sum_{m_1=1}^{3-j+1}\sum_{m_2=m_1+1}^{3-j+2}\cdots\sum_{m_j=m_{j-1}+1}^{3} I(\sum_{g=1}^{j}L_{m_g})]
$$

$$
\begin{aligned}
&= 1 - 2^{-2k} \sum_{m_1=1}^{3} I(L_{m_1}) - 2^{-k} \sum_{m_1=1}^{2} \sum_{m_2=m_1+1}^{3} I(L_{m_1} + L_{m_2}) \\
&\quad - \sum_{m_1=1}^{1} \sum_{m_2=m_1+1}^{2} \sum_{m_3=m_2+1}^{3} I(L_{m_1} + L_{m_2} + L_{m_3}) \\
&= 1 - 2^{-2k}[I(L_1) + I(L_2) + I(L_3)] \\
&\quad -2^{-k}[I(L_1 + L_2) + I(L_1 + L_3) + I(L_2 + L_3)] - I(L_1 + L_2 + L_3). \qquad (4.23)
\end{aligned}
$$

The comprehensive fault coverage model developed above is based on the density function of fault detection probabilities of a CUT. Like all the other aliasing analysis techniques based on detection probability profile, a major difficulty of using this model is the prohibitive computational efforts required for obtaining a precise detection probability density function for large CUTs.

4.2.3 A Simplified Fault Coverage Model

As discussed in Section 4.2, most faults detected in a segment i are also likely to be detected in some of the later segments $i+1$, $i+2$, ..., n. In [Zhang93][Zhang93b], a simplified fault coverage model was proposed which simply assumes that all faults that are detected in a segment i would also produce errors in later segments, $i+1$, $i+2$, ..., n. This simplified model is based on the aliasing probability, and the easily obtainable fault coverage curve before compaction.

Assume the aliasing probabilities at check points l_1, l_2, ..., l_n to be ρ_1, ρ_2, ..., ρ_n, respectively. Assume that, at the check points, the corresponding fault coverages before data compaction are known to be F_1, F_2, ..., F_n. The portion of faults detected in segment i is thus $F_i - F_{i-1}$, for $i = 1, ..., n$, defining $F_0 = 0$.

After checking the first signature at l_1, $F_1\rho_1$ of the faults detected in the first segment would escape detection. If all the aliased faults are assumed to be re-detected in later segments 2, 3, ..., n, after checking the second signature at l_2, the portion of faults aliased from the portion F_1 due to aliasing would be $F_1\rho_1\rho_2$. Thus, the portion of faults

that would be aliased from F_1 after checking n signatures is:

$$FCL_1 = F_1 \prod_{j=1}^{n} \rho_j. \tag{4.24}$$

Similarly, for the $(F_2 - F_1)$ faults first detected in the second segment, after checking the second signature at l_2, $(F_2 - F_1)\rho_2$ of them would be aliased. Again, assuming that these aliased faults will be re-detected in later segments 3, 4, ..., n yields the fault coverage loss from $(F_2 - F_1)$ after checking n signatures:

$$FCL_2 = (F_2 - F_1) \prod_{j=2}^{n} \rho_j. \tag{4.25}$$

In general, for the $(F_i - F_{i-1})$ faults first detected during the ith segment, the portion of the faults aliased is:

$$FCL_i = (F_i - F_{i-1}) \prod_{j=i}^{n} \rho_j \qquad i = 1, ..., n. \tag{4.26}$$

Therefore, the total fault coverage loss with n signatures is:

$$\begin{aligned} FCL &= \sum_{i=1}^{n} FCL_i \\ &= \sum_{i=1}^{n} (F_i - F_{i-1}) \prod_{j=i}^{n} \rho_j. \end{aligned} \tag{4.27}$$

Since the final fault coverage before data compaction is F_n, the fault coverage when checking n signatures is [Zhang93][Zhang93b]:

$$\begin{aligned} FC_n &= F_n - FCL \\ &= F_n - \sum_{i=1}^{n} (F_i - F_{i-1}) \prod_{j=i}^{n} \rho_j. \end{aligned} \tag{4.28}$$

Assuming $\rho_i = 2^{-k}$, for $i = 1, ..., n$, yields:

$$FC_n = F_n - \sum_{i=1}^{n} (F_i - F_{i-1}) 2^{-(n-i+1)k}. \tag{4.29}$$

Eqns. 4.28 and 4.29 predict the fault coverage with multiple signature analysis. This model is solely based on the knowledge of the fault coverage before data compaction and

the aliasing probability of the signature analyzer. It is thus much easier to use than the one developed in Sec. 4.2.2. However, this simplified model is optimistic since the model assumes that all the faults detected in segment i are re-detected in all later segments, $i+1$,$i+2$, ..., n. In practice, this assumption may not be true. Some faults may be re-detected in some of the later segments, but not necessarily in all the segments. Some faults may not be re-detected at all. However, if $L_i \leq L_{i+1}$ for $i = 1, ..., n$ as in [Lee88][Lambidonis91], the assumption can be well justified as shown in [Zhang93][Zhang93b].

Chapter 5

Multiple Intermediate Signature Analysis — II

The previous chapter studied the aliasing and fault coverage performance of multiple signature analysis. This chapter addresses other issues of multiple signature analysis, namely possible implementations, test control, test result observation, and applications.

5.1 Possible Implementations

5.1.1 Conceptual Understanding of Multiple Signature Analysis

Although multiple signature analysis can be implemented in different ways, there are some functional blocks that are commonly required. Assume that the n signatures generated by the LFSR at the check points $l_1, l_2, ..., l_n$ are $s_1, s_2, ..., s_n$, respectively. Once the check points are fixed, by logic simulation, one can easily determine the n corresponding fault-free signatures or references, $r_1, r_2, ..., r_n$. Except for the *fuzzy multiple signature* (FMS) analysis [Wu92] and the *single-reference multiple signature* (SMS) analysis [Wu93], which we will introduce in the next two chapters, when testing a CUT, the ith signature s_i is compared with a specific reference r_i at the ith check point. Thus, in conventional multiple signature (CMS) analysis, the signatures and references must correspond on a one-to-one basis for a CUT to be declared good, i.e., signature s_i must match r_i for all i. Thus, in implementation, besides the hardware required for storing the n references, an index generator is also required to build the one-to-one correspondence between signatures and references. Fig. 5.16 conceptually illustrates the CMS scheme. In Fig. 5.16, a ring counter is used to provide the index. When the test reaches the ith check point, the ith bit of

the ring counter is "1", and the signature collected in the LFSR is compared with the ith reference r_i. If they match, the ring counter advances by shifting the "1" to the $(i+1)$th bit. Otherwise, the test can be terminated, and the CUT declared bad. If signature s_i matchs r_i for all i, the CUT is declared good.

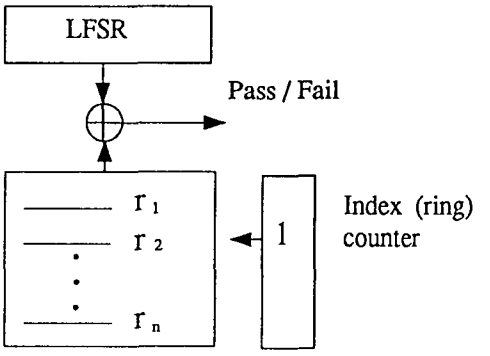


Figure 5.16: Conceptual representation of the CMS scheme.

5.1.2 Straightforward Implementations

A straightforward implementation of the CMS scheme is shown in Fig. 5.17. As shown, in addition to the ROM for storing the references, extra circuitry is required to generate the index and compare the references with the signatures. In this implementation, the index is provided by the ROM address generator which consists of a $log_2(n)$-stage binary counter and a $log_2(n) - to - n$ index decoder. The comparator can be composed of k 2-input XOR gates and a k-input NAND or NOR gate. Usually, the extra circuitry consumes more silicon area than that of the ROM itself. Furthermore, connections between these functional blocks can also take considerable silicon area.

Alternatively, instead of storing the references in a ROM and using a seperate comparator to generate the Pass/Fail signal, as shown in Fig. 5.18, one can use a combinational

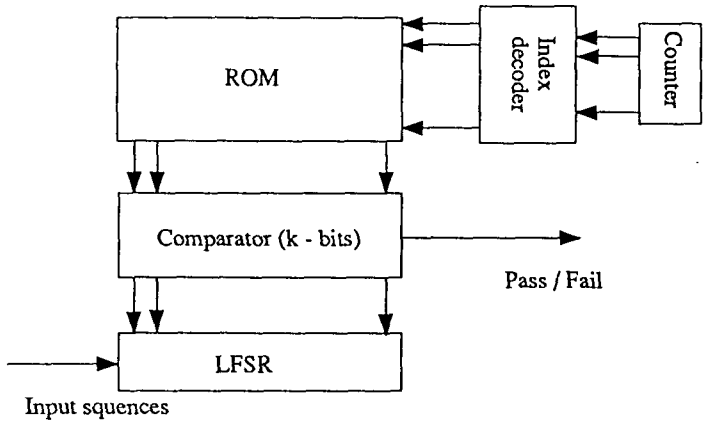


Figure 5.17: A straightforward CMS implementation with ROM.

logic (CL) for storing the references as well as for comparing them with the collected signatures. Assume the ith output line of the index decoder to be 1 at the ith check point l_i. Then, at check point l_i, if the signature collected in the LFSR matches the ith reference, the CL generates a Pass signal. Otherwise, it generates a Fail. If the CL generates Pass signals at all check points, the CUT is declared good. Otherwise, the CUT is bad.

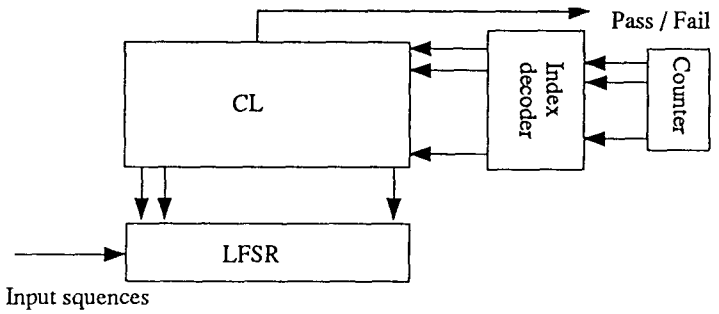


Figure 5.18: A straightforward CMS implementation with CL.

5.1.3 An Implementation by Resource Sharing

In [Wu91], a CMS implementation that shares some resources used for test pattern generation was described. Compared to the straightforward implementation discussed in the preceding section, the resource sharing CMS implementation requires less silicon area. Fig. 5.19 shows the block diagram of the CMS implementation by resource sharing.

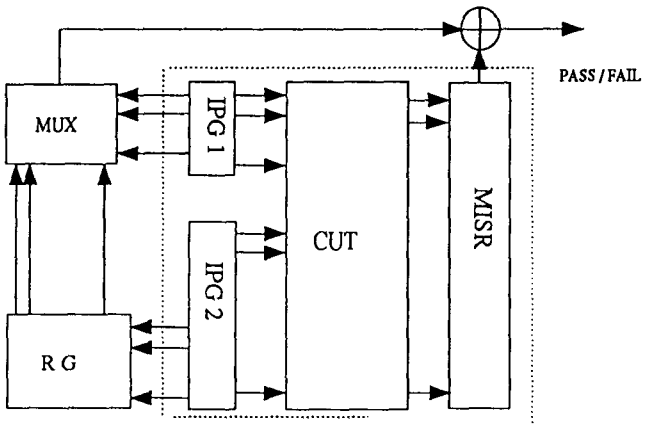


Figure 5.19: A CMS implementation with resource sharing.

In Fig. 5.19, the dotted box containing the CUT shows the standard BIST scheme, except that the pseudorandom input pattern generator (IPG) is split into two segments, IPG_1 and IPG_2, which are controlled by separate clock signals. The extra circuits required by the scheme is a reference generator (RG), and a $k - to - 1$ multiplexer (MUX), where k is the signature length. There is some local wiring overhead, but it is not significant.

The scheme works as follows. Each time a signature is checked, the RG generates a k-bit reference. Then, the MUX converts the generated reference into a k-bit sequence. Meanwhile, the XOR gate compares the signature generated in the signature analyzer, bit by bit, with the reference sequence from the MUX, and gives a Pass/Fail signal. $\mathrm{IPG_1}$ is a LFSR of length $log_2(k)$. Besides generating input vectors to the CUT, the $\mathrm{IPG_1}$ also provides the $log_2(k)$ control signals to the MUX. If the test length applied to the CUT is l, then the number of input lines to the RG is $log_2(l)$. To check k-bit signatures, the RG

has k output lines. The RG is basically a decoder which can easily be implemented with a PLA or other type of logic.

Regarding the control of the scheme, if a centralized controller can provide two clock signals to IPG_1 and IPG_2, then no extra control circuitry is required. When applying test patterns to the CUT, the two clock signals are identical. When checking a signature, the clock to IPG_2 is stopped, and k clock cycles are provided to the IPG_1. If the central controller provides only one clock signal, some extra local control circuitry is required. Fig. 5.20 shows a possible local controller required for the case $k = 4$. It works as follows. When a signature is checked, the RG outputs a logic 1 signal to the local controller, which resets the IPG_1 to a known state and cuts off the clock signal to IPG_2. After IPG_1 is reset, Gate 1 outputs a logic 0 signal to cut off the reset signal (rs). Now, the clock feeds to only IPG_1. After shifting IPG_1 k clock cycles, Gate 2 outputs a signal to reopen the clock to IPG_2. The overhead of the local controller is about the size of a 1-bit LFSR.

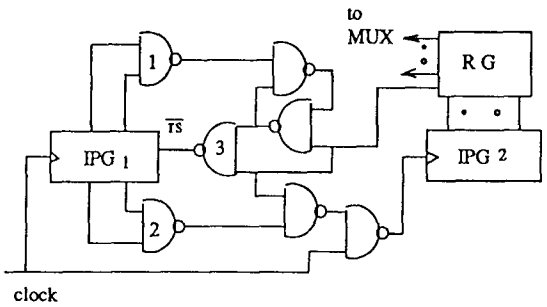


Figure 5.20: An example local control circuit.

A deficiency of this implementation is that its hardware requirement increases as the test length l increases since the size of the RG block is a function of $log_2(l)$. Thus, for large CUTs that require long test lengths, this implementation may impose substantial hardware requirements.

5.2 Test Result Observation

In BIST, after a test session terminates, the test result must be accessible externally through an I/O pin or pins. A simple way to report the test result is to produce a single bit Pass/Fail signal when the test is complete. In the single signature analysis, after the entire test output sequence has been compacted into the signature analyzer, the content of the LFSR is compared with the reference. The result of the comparison forms the Pass/Fail signal. In multiple signature analysis, however, the final Pass/Fail signal must be based on "intermediate Pass/Fail" signals produced at each of the check points. For a CUT to generate a "Pass" signal, all of the "intermediate Pass/Fail" signals must be in the "Pass" state. Otherwise, a "Fail" signal is generated. The following considers two different cases for test result observation in multiple signature analysis.

Assume that a logic "0" corresponds to an "intermediate Pass" signal, while a logic "1" corresponds to an "intermediate Fail" signal. In the first case, if there is a Pass/Fail pin available, a "zero" detector may be used, as shown in Fig. 5.21, to detect the "intermediate Pass/Fail" signals. Prior to testing a CUT, the "zero" detector is preset to "1", thus setting the Pass/Fail pin to 0. When a signature is checked, the controller temporarily sets the *chk* signal to 1. This sensitizes the "zero" detector to the intermediate Pass/Fail signal. Once a intermediate Fail signal is detected, the detector outputs and keeps a Fail signal, "1", at the Pass/Fail pin. The "zero" detector can be shared by all self-testable blocks on a chip. When shared, the detector outputs and keeps a Fail signal if any of the self-testable blocks is found faulty. This Fail signal can be used to terminate testing, thereby saving test time.

In the second case, where a specific protocol for control and observation of BIST is available [Scholz88], the test result of each self-testable block may consist of a single-bit flag stored in a status register [Ravinder89][Zorian91]. Once a faulty signature from a self-testable block is detected, the flag for that block is set. The status can be accessed through the IEEE 1149.1 Test Access Port [IEEE90][Zorian91]. In this case, the flag can be used as

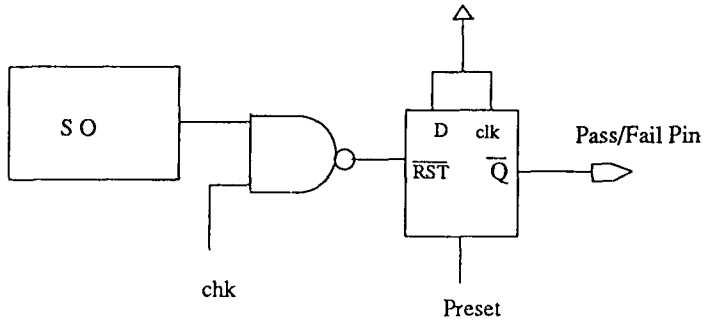


Figure 5.21: Test result observation for multiple signature analysis.

the "zero" detector required in the previous case.

5.3 Control of Multiple Signature Analysis

For the control of the conventional single signature schemes, an on-chip $log_2(l)$-bit binary counter, with l being the test length, is required to count the applied test vectors (see Fig. 5.22 a) [Breuer88][Gelsinger86]. When the final count is reached, a simple decoder detects this final count and generates a signal to stop the test and perform signature evaluation. In multiple signature analysis, however, multiple control signals are required, one for each of the check points. Therefore, the hardware requirement in this case would be in general greater than that required in the single signature schemes. However, as will be shown next, if the check points are periodically scheduled, the hardware requirement for the control of multiple signature analysis can be made as small as that required in any single signature scheme. Three possible cases depending on the scheduling of the check points are considered.

Case 1. The check points are arbitrarily scheduled, i.e., the test length between check points l_i and l_{i+1} is arbitrary. To control multiple signature analysis in this case, like in any single signature analysis scheme, an on-chip counter is required to count the applied test vectors. When a check point is reached, a decoder detects the corresponding count and outputs the *chk* signal to enable the evaluation of a signature. Unlike the single signature

scheme's controller, where the decoder detects only the final count, the decoder in this case must decode all the counts corresponding to the check points, and is thus considerably larger. Assuming the use of a single $log_2(l)$-input NAND or NOR gate for decoding the final count in the single signature case, the decoder in the multiple signature case may require up to n such gates plus a n-input gate for checking n signatures in the worst case if no logic minimization can be performed. Among the three possible control implementations considered in this section, this is the worst in terms of hardware overhead.

Case 2. The scheduling of the check points follows a periodic pattern, i.e., the test length between l_i and l_{i+1} is constant for all i. A convenient constant is 2^q, where q is an integer. To control multiple signature analysis in this case, one may simply split the binary counter required in the single signature scheme into two segments, say C_1 and C_2, as shown in Fig. 5.22b. C_1, which is q-bit long, counts the test length between two adjacent check points, i.e., 2^q. C_2 counts the number of signatures to be checked. Assume the final count to be 0 for both C_1 and C_2. Each time C_1 decrements to 0, a decoder decodes the 0 and generates the *chk* signal to enable the evaluation of a signature. If the signature is incorrect, testing can be terminated and the CUT declared faulty. Otherwise, C_2 is decremented by one, and testing continues. When both C_1 and C_2 reach 0, the test is complete. Obviously, the controller in this case requires the same amount of hardware as that for single signature analysis since the total length of C_1 and C_2 is the same as that of the counter used in single signature analysis. The total complexity of the two decoders required in the multiple signature case is also the same as that of the decoder in the single signature analysis. This is the best case in terms of hardware overhead.

Case 3. The check points are selected such that the test length between two adjacent check points, l_i and l_{i+1}, is variable but constrained to values of 2^{q_i}, where q_i is an integer. This case lies between the first two. In this case, one may still use C_1 and C_2 to count the applied test vectors. C_1 must be of length q_s, where $q_s = \min q_i$. Two decoders are required. One decodes the 0 state from C_1, while the other decodes the counts of $2^{q_i - q_s}$

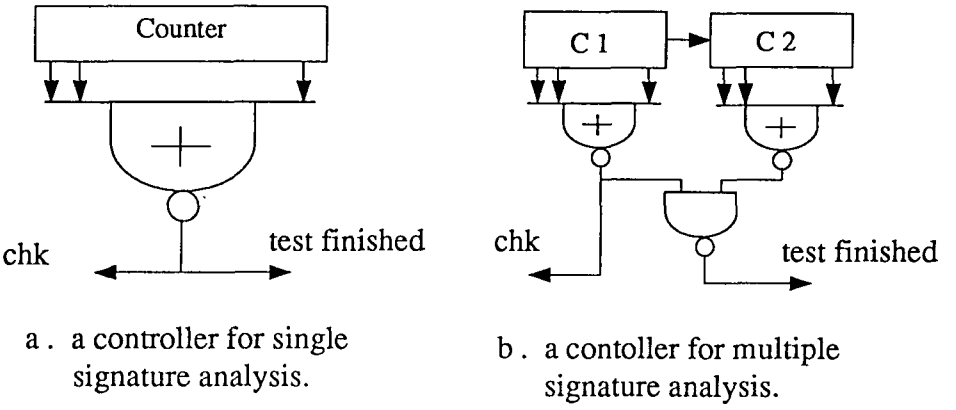


Figure 5.22: Example controllers for signature analysis.

from C_2, i = 1, .., n-1. Every time C_1 reaches 0, and C_2 reaches $2^{q_i - q_s}$, $i \in (1, ..., n-1)$, a signature is checked. The hardware overhead in this case is between the first two cases, since the decoder for C_1 detects only one count, but the decoder for C_2 has to detect n counts corresponding to the check points.

The above discussion assumes a counter for controlling the signature analysis. A LFSR in the autonomous mode can also be used to count the applied test vectors, e.g., as in the BIST controller of the Intel 80386 [Gelsinger86]. The control implementations discussed above applies equally well to the case where the conventional counter is replaced by a LFSR-type counter. The split of an LFSR into two smaller LFSR-based counters, C_1 and C_2, will not affect the randomness of the input pattern generator if concatenable polynomials are used [Bhavsar85].

5.4 Applications

As shown in the previous chapter, by careful arrangement of the check points, checking multiple signatures can yield significantly smaller aliasing than conventional single signature

analysis. Moreover, recent research has shown many other advantages of multiple signature analysis. This section provides a brief review of the possible applications of multiple signature analysis.

5.4.1 Exact BIST Fault Coverage Calculation

The calculation of exact fault coverage implies fault simulating the CUT with a specific test pattern generator and output data compactor. Without the output data compaction, fault simulation can exploit fault dropping to accelerate the process. In this case, the corresponding computational effort is proportional to the shaded area above the fault coverage curve (see Fig 5.23). However, to determine the fault coverage after compaction when only the final signature is checked, no fault dropping is possible. This is because aliasing results in the possibility that a fault detected at some point still escape detection at the end of test. Without fault dropping, fault simulation implies simulating each fault for the entire test length. In this case, the computational effort is proportional to the total shaded area above and below the fault coverage curve (see Fig. 5.24). For large CUTs, the required CPU time for such a fault simulation may become prohibitive [Waicukauski87][Lambidonis91][Zhang93].

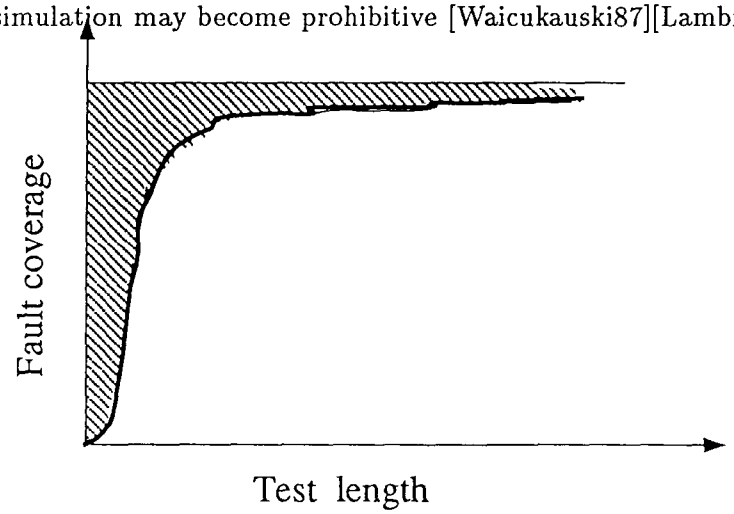


Figure 5.23: Fault simulation time to determine fault coverage before data compaction using fault dropping.

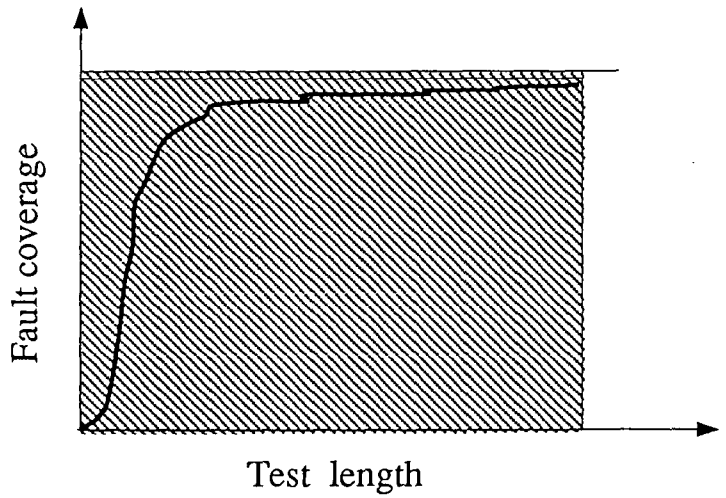


Figure 5.24: Fault simulation time to determine fault coverage after data compaction.

However, if multiple signature analysis is used, some amount of fault dropping can be used to reduce the fault simulation time. After checking an intermediate signature, all the faults that are detected by this signature can be dropped from further consideration. If a total of two signatures are checked, the required fault simulation time is illustrated by the shaded area shown in Fig. 5.25. Clearly, compared to the case shown in Fig. 5.24, this simple example shows how drastic reductions in total simulation time can be obtained using multiple signature analysis. If more intermediate signatures are checked, the computational time reduction can be more significant.

5.4.2 Test Time Reduction

Short test time is desirable in VLSI testing since it implies higher productivity and hence lower production costs. Reducing test time is another possible application of multiple signature analysis. In single signature analysis, a bad signature cannot be identified until the entire test response sequence has been compacted. In comparison, with multiple signature analysis, a test session can be terminated and the CUT declared bad as soon as an incorrect intermediate signature is found. For good CUTs, testing with multiple signature analysis

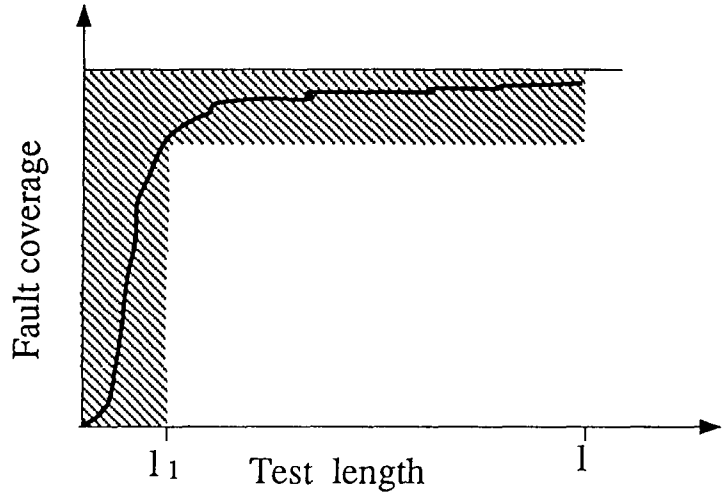


Figure 5.25: Fault simulation time with multiple signature analysis (assuming two signatures here).

takes the same time as that with a single signature scheme. Thus, on average, the test time can be significantly reduced if yield is not too high. In general, the test time reduction is dependent on the yield. The lower the yield, the shorter the test time on average. Furthermore, the reduction also depends on the scheduling of the check points, and the shape of the fault coverage curve [Robinson85][Lee88]. An algorithm that computes the optimal check point scheduling to minimize average test time was developed in [Lee88]. In general, the algorithm tends to schedule the check points at early stages of the test session. But, it was shown that even with periodic check point scheduling, average test time can be significantly shorter than using single signature schemes [Lee88].

Traditionally, BIST with output data compaction has been mainly used for *off-line testing*, e.g., manufacturing testing and in-field testing [Gelsinger86][Gelsinger89]. For off-line testing, test time affects only the production costs. Recently, researchers have also proposed to use BIST with signature analysis for concurrent testing or "on-line" testing [Saluja88][Katoozi92]. Unlike off-line testing, in concurrent testing, the time required to detect a fault once it occurs is crucial to the *dependability* of the system, which may include the *reliability, availability, safety, maintainability, performability*, depending on the application

of the system [Johnson89]. Due to the shorter average test time, segmented testing, and hence multiple signature analysis in BIST, are highly recommended for concurrent testing [Robinson85].

5.4.3 BIST Failure Diagnosis

Fault diagnosis is the process of locating the fault or faults in a faulty system. For systems at the level of circuit boards and multichip module packages, fault diagnosis is a necessary process for repair. At the VLSI chip level, fault diagnosis is often used for the analysis of failure mechanisms for fault modeling and process improvement [Waicukauski87][Aitken89b][Rajski91].

In addition to distinguishing good CUTs from bad ones, signatures obtained from CUTs also provide much information for fault diagnosis [McAnney87][Waicukauski87] [Karpovsky91][Rajski91]. A straightforward diagnostic method is to built a *fault dictionary* that contains the correspondence between each possible faulty signature and the detectable faults that would produce this signature. Upon the completion of a test session, the faulty signature obtained is used as an index to look up the fault dictionary for the location of the fault. *Diagnostic fault simulation* is the only way to build the fault dictionary. Unlike ordinary fault simulation, no fault dropping is allowed in diagnostic fault simulation [Waicukauski87]. Thus, this method is usually impractical for large CUTs.

In reality, however, only a few faults are ever actually used for diagnosing failures during the life of a product [Waicukauski87]. Thus, it is highly desirable to consider only these few faults so as to cut down the required computational efforts for fault diagnosis, and to increase the *diagnosis resolution.* Unfortunately, there is no way of knowing beforehand what these faults will be since the defects to be diagnosed are scattered throughout the design. Thus, diagnosis methods that are based on *post-test fault simulation* are proposed [Waicukauski87][Aitken89b]. These methods all collect some kind of intermediate "signatures". After completing a test session, a set of intermediate signatures are obtained. If some of them are faulty the CUT is faulty. To locate the fault, all possible modeled faults

are simulated until the first check point. Then, all the faults whose signatures obtained in the simulation differ from the corresponding signature obtained in testing are removed from further consideration. The remaining faults, i.e., the faults whose signature matches the signature obtained at the first check point in testing, are further simulated until the second check point. A similar process carries on until the final check point, or until the fault is located. This methodology has been in use at IBM for some time [Waicukauski87].

Chapter 6

Fuzzy Multiple Signature Analysis

In the CMS scheme, checking n signatures requires n references. Each time a signature is checked, it is compared to a specific reference. For a CUT to be declared good, each of the signatures must correspond to a corresponding reference. This strict one-to-one correspondence between the checked signatures and references makes the implementation of the CMS scheme relatively complex and expensive in terms of silicon area. In this chapter, we develop a *Fuzzy Multiple Signature* (FMS) analysis scheme which does not require the aforementioned one-to-one correspondence. The FMS scheme is simple to implement and requires little hardware.

6.1 Basis and Implementations

As discussed in Chapter 4, the complexity of checking multiple signatures is mainly due to the requirement of the one-to-one correspondence between the references and signatures. By removing this strict requirement one can get a much simpler data compactor. This is the basic idea of the FMS scheme. The scheme is referred to as a fuzzy multiple signature scheme because it consists of checking multiple signatures but does not impose the one-to-one correspondence between checked signatures and references. Thus a degree of fuzziness in the reference-signature relationship is introduced in the FMS scheme.

6.1.1 Basis

Like the CMS scheme, the FMS scheme checks n signatures at check points, $l_1, l_2, ..., l_n$. However, unlike the CMS scheme where a signature s_i is compared with a specific reference

r_i at check point l_i, in the FMS scheme, each signature s_i is compared with the whole set of references $\{r_1, r_2, ..., r_n\}$. A signature s_i is considered good if it matches any of the references in the reference set. Therefore, with the FMS scheme, for a CUT to be declared good, it suffices that the signature obtained from the LFSR at each check point correspond to any of the references $r_1, r_2, ..., r_n$. Fig. 6.26 conceptually illustrates the FMS scheme. The fuzziness introduced may result in a small increase of aliasing compared to the CMS scheme for given k and n. But, this can be easily compensated for by the reduced complexity of the FMS scheme compared to the CMS scheme. Otherwise, the FMS scheme possesses all the advantages that the CMS scheme has over the single signature schemes.

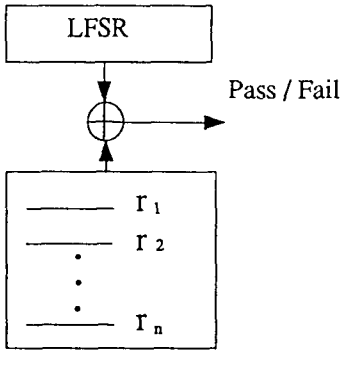


Figure 6.26: Conceptual representation of the FMS scheme.

6.1.2 Implementation

Since the one-to-one correspondence between references and signatures no longer exists, implementing the FMS scheme is very simple. As shown in Fig. 6.27, the FMS scheme consists of a Signature Observer (SO) and a LFSR. The LFSR collects signatures. The SO checks each signature and generates Pass/Fail signals. At each check point, if the signature generated by the LFSR matches any of the references, the SO outputs a Pass signal, say

logic 0. Otherwise, the SO outputs a Fail signal, say logic 1. The Fail signal can be fed to a test controller to terminate the testing and declare the CUT as faulty. If the SO outputs Pass signals at all the check points, the CUT is declared good.

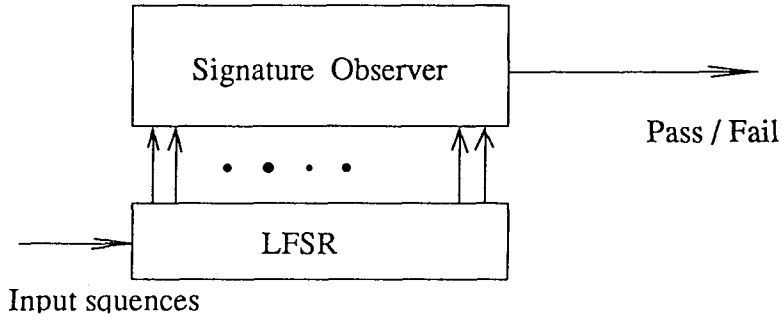


Figure 6.27: The FMS Data Compactor.

From the above discussion, the SO is a decoder with k-input, 1-output combinational circuit which outputs a 0 when its input vector belongs to the reference set, and outputs a 1 otherwise.

Example 5.1: Assume $n = k = 3$, i.e., checking three 3-bit signatures. If the references are $r_1 = 111$, $r_2 = 110$, and $r_3 = 100$, denoting the three bits of the references by b_1, b_2, and b_3, the function of the SO can be described as $\overline{Pass}/Fail = \overline{b_1b_2b_3 + b_1b_2\overline{b_3} + b_1\overline{b_2}\ \overline{b_3}} = \overline{b_1b_2 + b_1\overline{b_3}} = \overline{b_1\overline{\overline{b_2}b_3}}$. Thus, the SO can be implemented with two 2-input NAND gates as shown in Fig. 6.28. □

6.2 FMS Aliasing Performance Analysis

Assume the compaction of an l-bit random sequence into a k-bit signature, and r_1 to be the only valid reference. The total number of sequences (including the fault-free one) that map onto r_1 is 2^{l-k}. If we assume that the distinct references r_1 and r_2 are both acceptable,

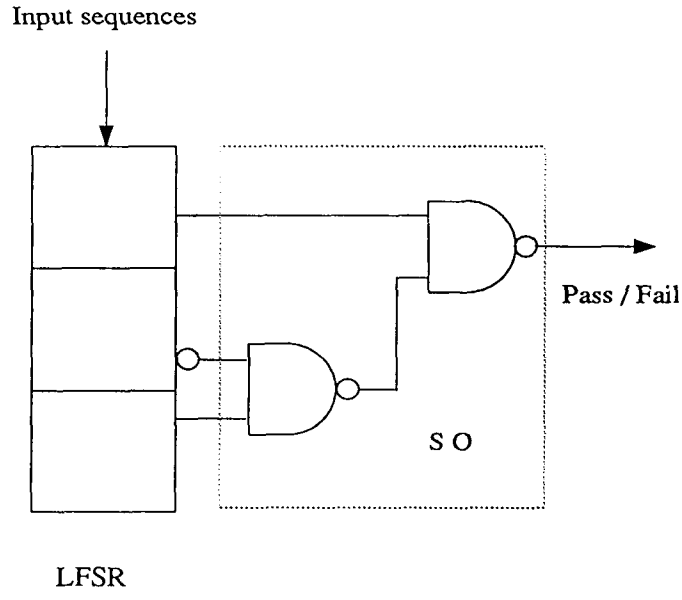


Figure 6.28: An Example of the FMS Scheme
where the references are 111, 110 and 100.

then the total number of sequences that map onto either r_1 or r_2 is $2 \times 2^{l-k}$ since the sequences that map onto one will not map onto the other. Thus, if we assume that m distinct references, $r_1, r_2, ..., r_m$ are acceptable, the total number of l-bit sequences that map onto any one of the m references is $m \times 2^{l-k}$. Excluding the fault-free sequence, the deception volume [Agarwal87] for such a case is thus $m \times 2^{l-k} - 1$. Since there are $2^l - 1$ possible error sequences, assuming all are equally likely, the aliasing probability when m distinct references are acceptable at a single check point is:

$$P_{al} = \frac{m2^{l-k} - 1}{2^l - 1}. \tag{6.30}$$

Assuming $2^l >> 1$ yields $P_{al} \approx m2^{-k}$.

The FMS scheme checks n signatures at check points, $l_1, l_2, ..., l_n$, against a set of m references, with $m \leq n$ in general (the reason that $m \leq n$ will be given later). Using the arguments presented in Chapter 4 for the aliasing probability of CMS schemes, the following aliasing probability results for the FMS scheme:

$$P_{FMS} = (\frac{m2^{l_1-k}-1}{2^{l_1}-1})(\frac{m2^{l_2-k}-1}{2^{l_2}-1})...(\frac{m2^{l_n-k}-1}{2^{l_n}-1}). \tag{6.31}$$

Assuming $2^{l_i} >> 1$ yields:

$$P_{FMS} \approx [m2^{-k}]^n, \tag{6.32}$$

where $m \leq n$ in general because there can be at most n distinct references if we check n signatures. However, some references may happen to be, or be made identical [Wu92b][Wu93] [Wu93c], thus making $m < n$. Clearly, for fixed k and n, the best case aliasing occurs for $m = 1$, for which $P_{FMS} \approx 2^{-nk}$ [Wu93]. $m = 1$ is also the best case in terms of hardware requirements for implementing the FMS scheme [Wu93]. When $m = n$, the worst case aliasing occurs, for which $P_{FMS} \approx [n2^{-k}]^n$.

The following analysis assumes the worst-case scenario (i.e., $m = n$). To study the aliasing performance of the FMS scheme, we define the *FMS scheme equivalent length*, L_e^{FMS}, as a figure of merit. For a given aliasing probability in the FMS scheme, we define L_e^{FMS} to be a continuous variable whose value corresponds to the length of a LFSR that yields the same aliasing probability in a single signature (SS) scheme. Ideally, L_e^{FMS} should be as large as possible to minimize aliasing. Since $P_{SS} \approx 2^{-k}$ and $P_{FMS} \approx [n2^{-k}]^n$, then

$$2^{-L_e^{FMS}} = [n2^{-k}]^n. \tag{6.33}$$

Solving for L_e^{FMS} yields:

$$L_e^{FMS} = n(k - log_2(n)). \tag{6.34}$$

In comparison, the equivalent length of a CMS scheme for given n and k is:

$$L_e^{CMS} = nk; \tag{6.35}$$

while that of a SS scheme is:

$$L_e^{SS} = k. \tag{6.36}$$

Example 5.2: Assuming $k = 16$ and $n = 4$, $L_e^{SS} = 16$, $L_e^{CMS} = 64$, and $L_e^{FMS} = 64 - 4 \times 2 = 56$. Thus, $P_{CMS} = 2^{-64}$, $P_{FMS} = 2^{-56}$, and $P_{SS} = 2^{-16}$. Here, the aliasing probability of the FMS scheme is 2 orders of magnitude greater than that of the CMS scheme, but still 12 orders of magnitude smaller than that of the SS scheme.□

With the CMS scheme, the equivalent length increases linearly with both k and n. With the FMS scheme, however, the equivalent length increases linearly with k but not with n. For fixed k, as n increases, L_e^{FMS} peaks and then decreases. When $n = 2^k$, $L_e^{FMS} = 0$. Fig. 6.29 shows an example of the L_e^{FMS} as a function of n for $k = 8$, $k = 9$, $k = 12$ and $k = 16$.

6.3 FMS Hardware Requirement Analysis

The SO of the FMS scheme is a k-input, 1-output combinational circuit, which can be implemented with different logic structures, e.g., a PLA, multilevel logic, or other non-PLA-type logic structures described in [Mead80][Zorian91].

If a PLA is used to implement the SO, to check n k-bit signatures, a k-input, 1-output, h-cube PLA is required, with $h \leq n$ because there can be at most n references if n signatures are checked. But, as shown in Example 5.1, the SO's function can sometimes be logically minimized, thus making $h < n$. In addition, as shown in [Wu92b][Wu93], some of

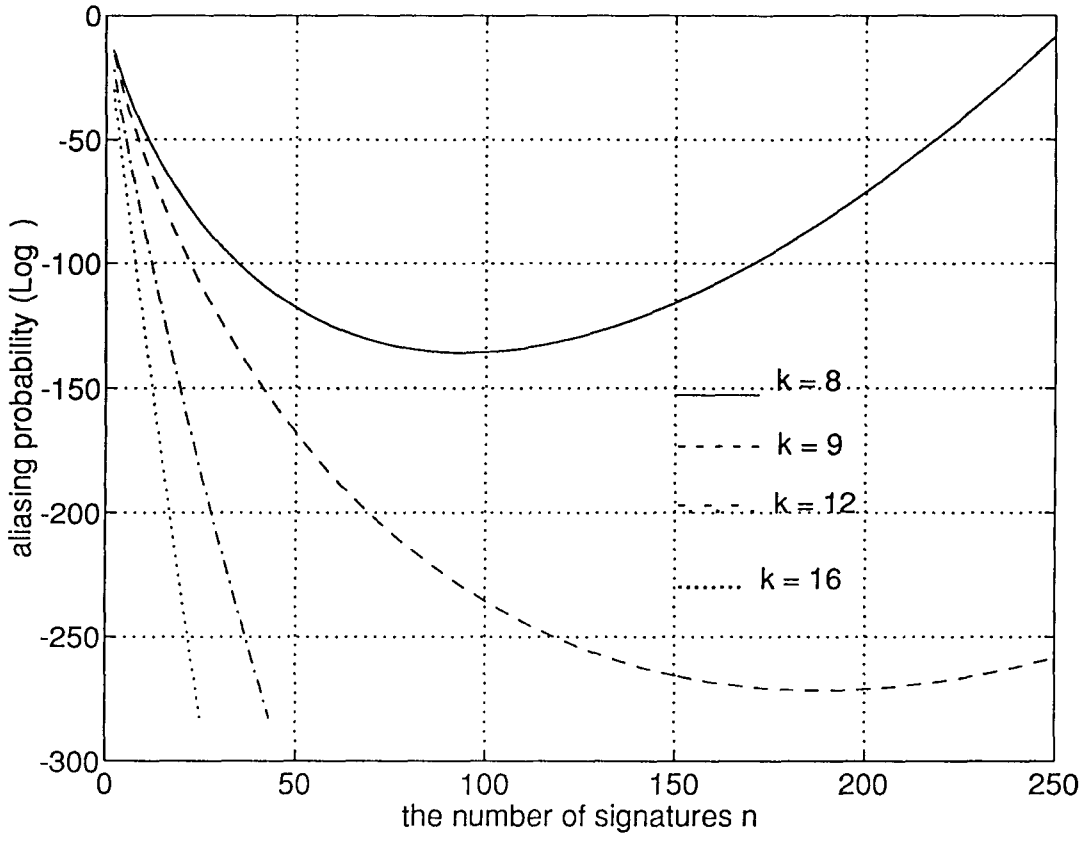


Figure 6.29: Aliasing performance of the FMS scheme.

the references may happen to be, or be made identical, which also results in $h < n$. In the worst case, i.e., $h = n$, the PLA has n cubes.

If the SO is implemented with logic gates, instead of a PLA, the hardware requirements are as follows. If there is only a single reference of length k, a k-input NAND gate is required to decode this reference from the LFSR. This k-input gate can be composed of $(k-1)$ 2-input NAND gates in a tree-structured form. (Note that this is also the hardware requirement of a SS scheme). If there exist two distinct references, assuming that they are not logically minimizable, then two k-input gates are needed to decode the two references. In addition, combining the outputs of the two gates to form the *Pass/Fail* signal of the SO requires an extra 2-input gate. Thus, a total of $2(k-1)+1 = 2k-1$ 2-input NAND gates are required since each k-input gate consists of $(k-1)$ 2-input NAND gates. In general,

for m distinct references, at most $mk - 1$ 2-input gates are needed. If $m = n$, as assumed in the worst case scenario in Section 2.1.3, the worst case hardare requirement of the SO is $nk - 1$ 2-input NAND gates.

6.4 Comparative Evaluation of the FMS Scheme

This section compares the aliasing performance and hardware requirements of the FMS scheme with those of the SS scheme, the Modified LFSR (M-LFSR) [Raina91], and the CMS scheme. Here, only the worst case of the FMS scheme is considered, i.e., it is assumed that $m = n$ and no logic minimization performed for the SO's function.

6.4.1 FMS vs. SS

To achieve an aliasing probability of 2^{-k}, the SS scheme requires a k-bit LFSR. To achieve the same aliasing probability, the FMS scheme only requires a $(k/n + log_2(n))$-bit LFSR if n signature are checked. For the following detailed area comparisons, a PLA implementation of the SO is assumed. Since each PLA-input variable corresponds to two lines in the AND plane of a PLA, and since the drivers in the PLA take an area of about 8 cubes [Gagne91], the normalized area of a k-input, s-output, n-cube PLA is $(n + 8) \times (k \times 2 + s)$ *units*. The following area estimate comparisons is based on the actual layout of a PLA and a 16-bit LFSR, using the $Cadence^{TM}$ automatic place and route tool, and a 3 μm double-metal CMOS technology. The LFSR was built with static D flip-flops, and measured approxmately 1.38×10^6 um^2. Actual layout revealed that a 12-input, 4-output, 64-cube PLA takes approxmately the same area as a 16-bit LFSR. According to the above analysis, this PLA requires an area of $(64 + 8) \times (12 \times 2 + 4) = 2016$ *units*. Therefore, we assume that a PLA of 2016 *units* corresponds to the area of a 16-bit LFSR. The comparison of the FMS and SS schemes is illustrated by the following examples.

Example 5.3: If $k = 9$ and $n = 32$, $P_{FMS} = [32 \times 2^{-9}]^{32} = 2^{-128}$. In this case, the required hardware is a 9-stage LFSR, and a 9-input, 1-output, 32-cube PLA to implement the SO. The PLA requires $(32+8) \times (9 \times 2+1) = 760$ *units* of area, which is approxmately 37.7% of the size of a 16-bit LFSR, or about the size of a 6-bit LFSR. Thus, the total area overhead for the FMS scheme to achieve $P_{FMS} = 2^{-128}$ is approximately the area of a 6+9=15 bit LFSR. In comparison, a SS scheme would require a 128-bit LFSR to achieve $P_{SS} = 2^{-128}$. □

More examples are summarized in Fig. 6.30, where the area for achieving a given aliasing probability is given in terms of equivalent LFSR sizes. As shown in Fig. 6.30, with the FMS scheme, small aliasing can be obtained against very small hardware overhead compared to what is required by SS schemes.

6.4.2 FMS vs. M-LFSR

According to [Raina91], for a r-input k-stage LFSR, and assuming the equally likely error model, the aliasing probability of the M-LFSR is:

$$P_{M-LFSR} \approx 2^{-(k+rs)}, \qquad with \;\; r \leq k \;\; and \;\; s < k. \tag{6.37}$$

Thus, the worst case of the M-LFSR is when $r = 1$, which corresponds to a single input LFSR. In this case, $P_{M-LFSR} \approx 2^{-(k+s)}$. The best case occurs when $r = k$, which corresponds to a k-input MISR. In the best case, $P_{M-LFSR} \approx 2^{-k(s+1)}$. In terms of hardware requirements, in addition to the LFSR, the M-LFSR requires $\lceil s(8 + r + k) \rceil$ 2-input NAND/NOR gates.

In comparison, for the FMS to achieve an aliasing probability of $P_{FMS} \approx 2^{-n(k-log_2(n))}$, the hardware requirement is $(nk - 1)$ 2-input NAND/NOR gates in addition to the LFSR. Tables 6.1 and 6.2 show some examples of the M-LFSR and FMS comparisons. In Tables

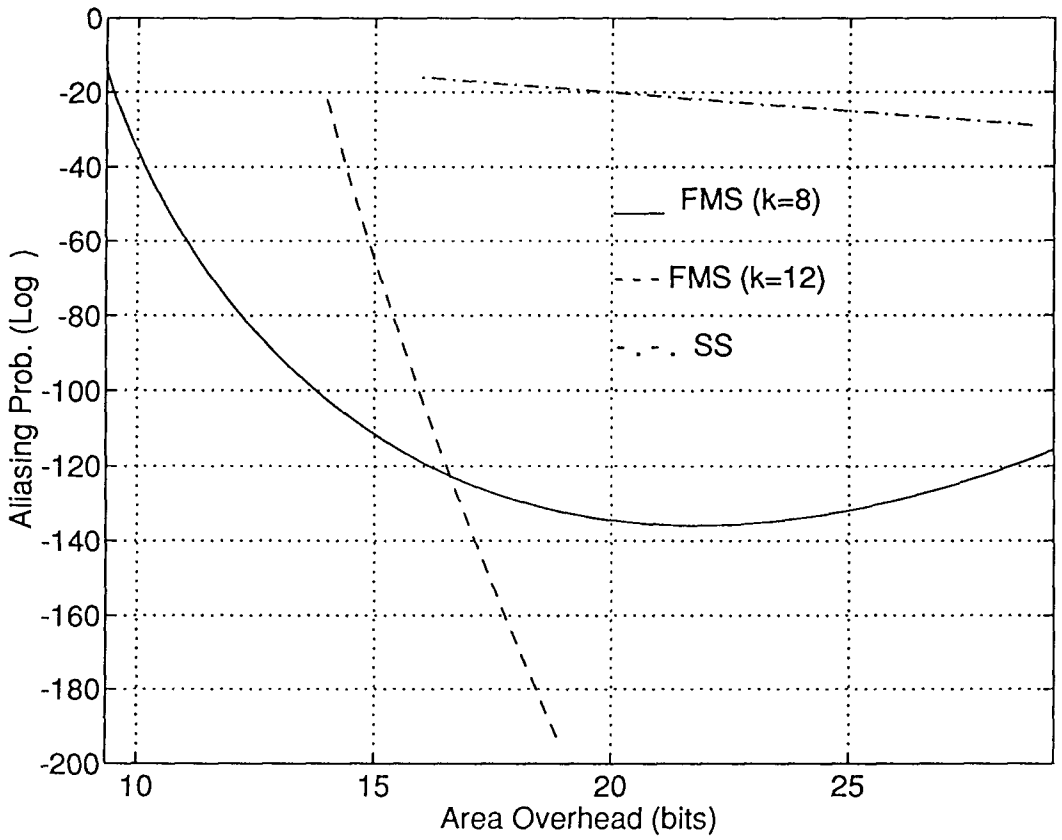


Figure 6.30: The FMS scheme vs. the SS scheme.

6.1 and 6.2, the entries in the row entitled *Hardware Req.* show the number of gates required by respective schemes in addition to the LFSR; the entries in the row entitled P_{al} indicate the aliasing probabilities. In Table 6.1, a single input 16-stage LFSR is assumed, while in Table 6.2, a 16-stage MISR is assumed. In the comparison, the s for the M-LFSR is assumed to be $k-1$, which is the best case for reducing P_{M-LFSR}. As shown in Table 6.1 and 6.2, the FMS scheme is obviously a better data compactor than the M-LFSR scheme since the FMS scheme achieves smaller aliasing with much less hardware requirement.

	M-LFSR	*FMS*	
Configuration	*k=16,r=1,s=15*	*n=2, k=16*	*n=3, k=16*
P_{al}	2^{-31}	2^{-30}	$2^{-43.2}$
Hardware Req.	*375 gates*	*31 gates*	*47 gates*

Table 6.1: The FMS scheme vs. the M-LFSR scheme when a LFSR is used.

	M-LFSR	*FMS*
Configuration	*k=16,r=16,s=15*	*n=23, k=16*
P_{al}	2^{-256}	2^{-263}
Hardware Req.	*600 gates*	*367 gates*

Table 6.2: The FMS scheme vs. the M-LFSR scheme when a MISR is used.

6.4.3 FMS vs. CMS

For given n and k, the aliasing of the FMS scheme is generally higher than that of the CMS scheme. This is due to the fuzziness introduced by removing the one-to-one reference-signature correspondence. The aliasing probability of the FMS scheme is shown to be (see Eqn. 6.2):

$$\begin{aligned} P_{FMS} &\approx (m2^{-k})^n \\ &= m^n 2^{-nk} \\ &= m^n P_{CMS}, \qquad m \leq n, \end{aligned} \tag{6.38}$$

where m is the number of distinct references, n is the number of signatures, and P_{CMS} is the aliasing probability of the CMS scheme, which is 2^{-nk} assuming n k-bit signatures.

From Eqn. 6.38, it is obvious that smaller m implies smaller P_{FMS}. When $m = 1$, the fuzziness disappears, thus $P_{FMS} = P_{CMS}$. Moreover, with $m = 1$, the hardware requirement for implementing the FMS scheme becomes minimal for multiple signatures analysis, i.e., becomes exactly the same as that for a single signature scheme [Wu93].

The parameter m can be used as a measure of fuzziness. When m increases, the

FMS scheme becomes fuzzier, thus higher aliasing. Generally, $1 \leq m \leq n$ since for checking n signatures there are at most n references. However, it is possible to make m greater than n [Ivanov93], achieved by including extra references. This can be attractive to ease the logic minimization of the SO, thus resulting in significantly less hardware requirement for implementing the FMS scheme.

6.5 Experimental Results

To study the fault coverage performance of the FMS scheme, exact fault simulation of the ISCAS'85 benchmark circuits [Brglez85] were performed with output data compaction. In the experiments, a multiple input nonfeedback shift register (MINSR) was used as a space compactor to convert the multiple bit output data from a CUT into a single bit data to the LFSR signature analyzer [Agarwal87]. In the experiments, single stuck-at faults were assumed, and the test length $l = 1024$. In the case of the SS scheme, an 8-bit primitive LFSR was used as the data compactor. For the FMS scheme, 8 signatures were checked periodically, and each was generated by the 8-bit LFSR, i.e., $k = 8$ and $n = 8$. In terms of fault simulation time reduction, the periodic check point scheduling is not the best. But the area requirement for controlling the FMS scheme with periodic check point scheduling is the same as that required for a SS scheme. The silicon area corresponding to the SO is about the size of a 2.16-bit LFSR. Thus, the total silicon area of the FMS scheme in the experiments, including the SO and an 8-stage LFSR, is approxmately equivalent to that of a 10.16-bit LFSR, as opposed to an 8-bit LFSR for the SS scheme.

Table 6.3 shows the exact fault coverage before signature analysis (the column referred as "NC"), and the coverage with the SS and the FMS schemes. From Table 6.3, in all cases, the coverage of the FMS scheme is either the same or very close to that of the no-compaction case, and higher than that of the SS scheme.

Table 6.4 shows the CPU time to perform the above fault simulations. From Table

Circuit Name	*No. Faults*	*Fault Coverage (%)*		
		NC	*FMS*	*SS*
*C*432	562	99.244	99.244	99.110
*C*499	732	98.907	98.907	97.951
*C*880	1171	95.645	95.645	95.389
*C*1355	1820	98.640	98.640	98.132
*C*1908	2015	94.652	94.615	93.697
*C*2670	2781	82.659	82.659	82.057
*C*3540	4035	93.451	93.451	93.333
*C*5315	5982	96.230	96.230	96.172
*C*7552	8408	93.012	93.010	*n/a*

test length = 1,024 random vectors

Table 6.3: Fault coverage enhancements (average of 4 trials).

6.4, the CPU time required to perform the fault simulations with the FMS scheme is always close to that of the no-compaction case, which constitutes a lower bound [Lambidonis91]. In the case of the SS scheme, however, the fault simulation time is always one order of magnitude longer. In Table 5.4, the column *Time Savings* shows the CPU time saved from using the FMS scheme rather than the SS scheme. Here, the number of test vectors simulated is small. If longer test length, larger circuits, and optimal scheduling of the check points [Lambidonis91] were considered, the fault simulation time saved from using the FMS scheme rather than a SS scheme would be even more significant.

6.6 Conclusions

In conventional multiple signature (CMS) analysis, for a CUT to be declared good, the signatures and the references must correspond on a one-to-one basis. That is, at each check point, the signature obtained must match the specific reference for that check point. This requirement of the one-to-one signature – reference correspondence makes the CMS scheme complex and expensive to implement in terms of silicon area. In this chapter, the Fuzzy Multiple Signature (FMS) scheme was developed whereby the aforementioned requirement

Circuit Name	*No. Faults*	*Fault Simulation Time (sec.)*			*Time Savings*
		NC	*FMS*	*SS*	
*C*432	562	2.25	6.96	48.90	85.77%
*C*499	732	3.70	11.32	72.64	84.42%
*C*880	1171	14.30	34.26	191.37	82.10%
*C*1355	1820	29.39	64.54	380.36	83.03%
*C*1908	2015	63.66	108.00	465.29	76.79%
*C*2670	2781	202.56	361.37	1244.56	70.91%
*C*3540	4035	268.21	420.38	1823.64	76.95%
*C*5315	5982	419.86	841.02	4992.12	83.15%
*C*7552	8408	1265.37	1865.72	*n/a*	*n/a*

test length = 1,024 random vectors

Table 6.4: Fault simulation time reductions
(average of 4 trials).

of the reference-signature correspondence is removed. The FMS scheme is very simple, thus much easier and less expensive to implement in a BIST environment. A model for predicting the FMS scheme's aliasing was developed. Compared with other data compaction schemes, e.g., the single signature (SS) schemes and the M-LFSR scheme, the FMS scheme requires less silicon area to achieve a given aliasing probability. The experimental results on fault coverage enhancement, fault simulation time savings, and silicon area overhead have shown the advantage of the FMS scheme in reducing fault simulation time in addition to reducing aliasing. By carefully scheduling the signatures as in [Pomeranz92], the FMS scheme can achieve zero aliasing for modeled faults. In this chapter, a LFSR-based data compaction was assumed. However, the FMS scheme can be applied to MISRs, CAs, or count-based data compactors.

Chapter 7

Single Reference Multiple Signature Analysis

Conventionally, checking n signatures requires n references. In the previous chapter, the FMS (fuzzy multiple signature) scheme was developed. Unlike conventional multiple signature (CMS) analysis, the FMS scheme may check n signatures against fewer references if some of the n corresponding fault-free signatures happen to be identical. The reduction of the number of references in the FMS scheme not only helps further reduce its complexity and hardware requirements for implementation, but also improves its aliasing performance since $P_{FMS} \approx m^n 2^{-nk}$, with m being the number of distinct references. It is desirable that $m = 1$, i.e., that all the references be identical. In practice, however, it is very unlikely that all the references turn out to be identical. In this chapter, we explore the possibility that $m = 1$, and develop a single-reference multiple signature (SMS) analysis scheme that deliberately makes $m = 1$ [Wu93][Wu93b]. Techniques for implementing the SMS scheme are also developed in this chapter.

7.1 Basis

If all the references required in the conventional case are identical, in effect, only one reference is necessary for checking n signatures. For implementation, the hardware requirement for checking multiple identical signatures is the same as that for checking only one, and independent of the number of checked signatures. However, checking multiple signatures provides more advantages than checking only one (see Chapter 4). Having one constant reference signature for all check points is the basis of the SMS scheme.

There exist different ways to implement the SMS scheme. Next, we present a SMS

scheme implementation approach that carefully selects (or designs) the seeds to both the input pattern generator (IPG) and the signature analyzer (SA) such that all the signatures at predetermined check points be identical if the CUT is fault-free and if the IPG and the SA are initialized with the selected seeds prior to testing. Obviously, this implementation approach requires no modification of CUTs. Compared to the CMS and SS (single signature) schemes, the cost of using this approach to implement the SMS scheme is a small non-recurring CPU time expenditure in the design phase. In return, the SMS scheme yields significant recurring silicon area savings, and increases test quality due to reduced aliasing.

In the following, for simplicity, we assume a LFSR signature analyzer for data compaction. However, as will be shown later in this chapter, the SMS scheme and the analysis can also be applied to other types of data compactors, e.g., MISRs and CAs.

7.2 Preliminaries

7.2.1 Signature Analysis

Let $\{R\}_l$ denote a sequence R of length l, and $[A]_k$ denote a k-stage LFSR in the initial state A, or seed A. Denote the signature obtained after shifting l-bits of the sequence $\{R\}_l$ into the LFSR with seed A by $S_A(l)$. The signature analysis process can be described as:

$$S_A(l) = \{R\}_l >> [A]_k, \tag{7.39}$$

where $>>$ denotes the shift operation.

Definition 7.1 [Lambidonis91b]: The contribution of a seed A to the final signature is called the *autonomous contribution*, $S_A^{auto}(l)$:

$$S_A^{auto}(l) = \{0\}_l >> [A]_k, \tag{7.40}$$

where $\{0\}_l$ is l-bit all zero sequence.

Definition 7.2 [Lambidonis91b]: The contribution of an input sequence $\{R\}_l$ to the final signature is called the *input contribution*, $S_0^{inp}(l)$:

$$S_0^{inp}(l) = \{R\}_l >> [0]_k. \tag{7.41}$$

Lemma 7.1 [1] [Lambidonis91b]: Let $[A]_k$ and $[B]_k$ denote the a k-stage LFSR with seeds A and B, respectively. Let $\{R\}_l$ represent a l-bit sequence and let $\{T\}_l$ represent another l-bit sequence. Then,

$$(\{R\}_l >> [A]_k) \oplus (\{T\}_l >> [B]_k) = (\{R\}_l \oplus \{T\}_l) >> ([A]_k \oplus [B]_k), \tag{7.42}$$

where $\oplus$ is bit-wise exclusive-or.

The following lemma is a simplified version of Theorem 1 in [Lambidonis91b].

Lemma 7.2 :

$$S_A(l) = S_0^{inp}(l) \oplus S_A^{auto}(l), \tag{7.43}$$

where $\oplus$ is bit-wise exclusive-or.

Proof :

$$\begin{aligned} S_A(l) &= \{R\}_l >> [A]_k \\ &= (\{R\}_l \oplus \{0\}_l) >> ([A]_k \oplus [0]_k) \end{aligned}$$

according to Lemma 7.1,

$$\begin{aligned} &= (\{R\}_l >> [0]_k) \oplus (\{0\}_l >> [A]_k) \\ &= S_0^{inp}(l) \oplus S_A^{auto}(l). \end{aligned} \tag{7.44}$$

Q.E.D.

[1] Lemma 7.1 is the property 1 — General Distribution Property in [Lambidonis91b].

7.2.2 Non-singular LFSRs

LFSRs are usually classified into primitive and non-primitive LFSRs (see Chapter 2). For primitive LFSRs, we also introduce a new classification: *singular* and *non-singular* LFSRs.

Definition 7.3 : A primitive LFSR of degree k is said to be *non-singular* if for any given l_i and l_j, where $|l_i - l_j| mod(2^k - 1) \neq 0$, $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j)$ is unique for each non-zero seed A. Otherwise, the LFSR is *singular.*

For example, if an LFSR is designed to implement the state transition diagram shown in Fig. 7.31a, the LFSR would be singular. This is because, when $l_i = 0$ and $l_j mod(2^k - 1) = 2$, $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j) = 010 \oplus 101 = 111$ if $A = 010$, and $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j) = 110 \oplus 001 = 111$ if $A = 110$ (see Fig. 7.31). Hence, by definition, such an LFSR is singular. However, if the LFSR is designed to implement the state transition diagram shown in Fig. 7.31b, the LFSR is non-singular.

Figure 7.31: Example state transition diagrams.

Although the design of non-singular LFSRs is still an open problem, checking for the non-singularity of a given LFSR is fairly simple. In Appendix A, an algorithm for this

purpose is presented. Applying the algorithm given in Appendix A, all the primitive feedback polynomials of degree ≤ 20 listed in [Bardell87] have been tested. All the polynomials of degree ≤ 19 have been shown to be non-singular. The polynomial of degree 20 is singular. As will be shown later, non-singular LFSRs are helpful to the implementation of the SMS scheme.

7.3 Identical Signature Properties

Assume a set of random vectors is applied to a fault-free CUT such that a fault-free sequence of length l results. Assume the checking of n k-bit signatures at the predetermined check points $l_1, l_2, ..., l_n$. [1].

Definition 7.3 : The sequence $\{R\}_l$ is said to have the *Constrained Identical Signature* property $CIS_{(n,k)}$, or CIS for simplicity, if $S_A(l_1) = S_A(l_2) = ... = S_A(l_n) = S$ and both A and S are fixed.

Definition 7.4 : The sequence $\{R\}_l$ is said to have the *Identical Signature* property $IS_{(n,k)}$, or IS for simplicity, if $S_A(l_1) = S_A(l_2) = ... = S_A(l_n) = S$ and A is fixed but S is free to be any of the 2^k possible values.

Definition 7.5 : The sequence $\{R\}_l$ is said to have the *Relaxed Identical Signature* property $RIS_{(n,k)}$, or simply RIS, if $S_A(l_1) = S_A(l_2) = ... = S_A(l_n) = S$ and both A and S are free to be any of the 2^k possible values.

Theorem 7.1 : Given a sequence and a pair of check points l_i and l_j, if $|l_i - l_j| mod(2^k - 1) \neq 0$, and if the LFSR is non-singular, there exists a seed A such that:

$$S_A(l_i) = S_A(l_j). \tag{7.45}$$

[1] Note that the scheduling of these check points is predetermined and can be arbitrary. However, different scheduling of these check points would affect hardware requirements for controlling multiple signature analysis. It has been shown in Chapter 5 that periodical check point scheduling is optimal in regards to control.

Proof : Assume the signatures obtained at check points l_i and l_j with seed $[0]_k$ to be $S_0(l_i)$ and $S_0(l_j)$, respectively. Two cases exist. One is that $S_0(l_i) = S_0(l_j)$. The second is that $S_0(l_i) \neq S_0(l_j)$. If $S_0(l_i) = S_0(l_j)$, the theorem is true with $A = 0$. So, we only need to prove $S_A(l_i) = S_A(l_j)$ when $S_0(l_i) \neq S_0(l_j)$.

Denote the $2^k - 1$ non-zero states of the LFSR by w_1, w_2, ..., w_{2^k-1}. Since

$$S_A(l_i) = S_0(l_i) \oplus S_A^{auto}(l_i),$$
$$S_A(l_j) = S_0(l_j) \oplus S_A^{auto}(l_j),$$

proving $S_A(l_i) = S_A(l_j)$ is equivalent to proving the existence of a seed A such that $S_0(l_i) \oplus S_0(l_j) = S_A^{auto}(l_i) \oplus S_A^{auto}(l_j)$. Since $S_0(l_i) \oplus S_0(l_j) \neq 0$, $S_0(l_i) \oplus S_0(l_j) \in \{w_1, w_2, ..., w_{2^k-1}\}$, and thus we only need to prove that $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j)$ generates the set $\{w_1, w_2, ..., w_{2^k-1}\}$ when $A = w_1, w_2, ..., w_{2^k-1}$, i.e., to prove that $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j)$ is distinct for each of the $2^k - 1$ possible non-zero values of A. Since $|l_i - l_j| mod(2^k - 1) \neq 0$ and the LFSR is non-singular, according to Definition 7.3, $S_A^{auto}(l_i) \oplus S_A^{auto}(l_j)$ is distinct for each non-zero seed A.

Q.E.D.

Corollary 7.1 : When checking two k-bit signatures (n=2), for any given output sequence and any predetermined check points l_1 and l_2, if the test length between the check points is not a multiple of $2^k - 1$, i.e., $|l_2 - l_1| mod(2^k - 1) \neq 0$, and if the SA LFSR is non-singular, then there always exists a seed for the SA LFSR such that the two signatures obtained at check points l_1 and l_2 are identical.

Proof : Follows directly from Theorem 7.1.

Q.E.D.

In the following, we assume that the feedback polynomial for the signature analyzer LFSR is primitive, and that the test lengths between adjacent check points are sufficiently

long for the asymptotic aliasing probability 2^{-k} to hold (see Chapter 3). If each bit of $\{R\}_l$ is assumed to be statistically independent and equally-likely to be 0 or 1, we have the following lemmas.

Lemma 7.3 : The probability that a sequence $\{R\}_l$ possesses the CIS property is 2^{-nk}.

Proof: Since each sequence bit is equally-likely to be 0 or 1, similarly to the analysis for aliasing calculation in Chapter 3, the probability for a signature $S_A(l_i)$ to be equal to a specific value, say $S_A(l_i) = [0]_k$, is 2^{-k}, where $1 \leq i \leq n$. Therefore, the probability for the n signatures to be identical and equal to a specific value, i.e., the probability for the sequence to possesses the CIS property, is 2^{-nk}.

Q.E.D.

Lemma 7.4 : The probability that a sequence $\{R\}_l$ possesses the IS property is $2^{-(n-1)k}$.

Proof: From Lemma 7.3, the probability for the sequence $\{R\}_l$ to yield n identical specific signatures is 2^{-nk}. For a k-stage LFSR, a total of 2^k possible signature values exist. Thus, if the n identical signatures can be equal to any of the 2^k values, the probability that the sequence yields n identical signatures is $2^k \times 2^{-nk} = 2^{-(n-1)k}$. According to the Definition 7.4, Lemma 7.4 is true.

Q.E.D.

Lemma 7.5 : For a non-singular LFSR, the probability that a sequence $\{R\}_l$ possesses the RIS property is $2^{-(n-2)k}$ if the test length between at least one pair of check points is not a multiple of $(2^k - 1)$. Otherwise, the probability is $2^{-(n-1)k}$.

Proof : 1). First, we prove the case where the test length between one pair of check points, say check points l_i and l_j, is not a multiple of $2^k - 1$. In this case, according to Theorem 7.1, there exists a seed A such that $S_A(l_i) = S_A(l_j)$. Assume $S_A(l_i) = S_A(l_j) = S$.

According to Lemma 7.3, once A and S are fixed, the probability for the signature at any of the $(n-2)$ remaining check points, say check point l_m, where $m \neq i$ and $m \neq j$, to be identical to S is 2^{-k}, independently of the test length between check point l_m and the others. Therefore, the probability for all n signatures to be identical is $1 \times 2^{-(n-2)k} = 2^{-(n-2)k}$ as long as $|l_i - l_j| mod(2^k - 1) \neq 0$ and the LFSR is non-singular.

2). We prove the case where the test length between each pair of check points is a multiple of $2^k - 1$. In this case,

$$S_A^{auto}(l_1) = S_A^{auto}(l_2) = ... = S_A^{auto}(l_n).$$

According to Lemma 7.2,

$$S_A(l_i) = S_A^{auto}(l_i) \oplus S_0^{inp}(l_i) \qquad for \quad i = 1, 2, ..., n,$$

where $S_0^{inp}(l_i) = S_0(l_i)$.

Therefore, if $S_0(l_1) = S_0(l_2) = ... = S_0(l_n)$, then $S_A(l_1) = S_A(l_2) = ... = S_A(l_n)$. Otherwise, the n signatures with seed A will not be identical. As a result, whether the n signatures are identical becomes independent of the value of A. According to Lemma 7.4, the probability for all n signatures to be identical is $2^{-(n-1)k}$.

Q.E.D.

For our purpose, a fault-free sequence that possesses the RIS property is adequate since the value of the reference signatures as well as the value of the seeds are not important provided that the reference signatures are identical[1]. For practical values of n and k, the probability that a sequence possesses the RIS property can be very high or very low, depending on the values of n, k, and whether the test length between check points is a

[1]The actual values of the reference signatures and the designed seeds do not matter since the initialization to any value requires the same complexity (space or time) and similarly with the reading/comparison of the signatures.

n	k	P_{al}	RIS probability	
			$2^{-(n-1)k}$	$2^{-(n-2)k}$
2	k	2^{-2k}	2^{-k}	1
3	8	2^{-24}	2^{-16}	2^{-8}
5	4	2^{-20}	2^{-16}	2^{-12}
9	2	2^{-18}	2^{-16}	2^{-14}
17	1	2^{-17}	2^{-16}	n/a[1]

Table 7.5: Example RIS probabilities.

multiple of $2^k - 1$ [Wu93b]. For example, for a non-singular LFSR, according to Corollary 7.1 and Lemma 7.5, if $n = 2$ and the test length between check points is not a multiple of $2^k - 1$, the probability for a sequence to possess the RIS property is always 1, regardless of the value of k. However, when n is large or the test length between each pair of check points is evenly divisible by $2^k - 1$, the probability for a given sequence to possess the RIS property can be very small [Wu93]. Examples are given in Table 7.5. In Table 7.5, the columns $2^{-(n-1)k}$ and $2^{-(n-2)k}$ respectively correspond to the RIS probabilities where the test length between check points is and is not evenly divisible by $2^k - 1$. The column entitled P_{al} is the corresponding aliasing probabilities for given n and k.

When the RIS probability is small, it is unlikely that the n signatures from a given sequence be identical. However, as shown in [Wu93], if there exists a large sample set of different fault-free sequences to choose from, then the probability that at least one of the sample sequences has the RIS property can be high. A choice of a large number of fault-free sequences can arise by assuming that the actual test set can be chosen from a large number of random test sets generated simply by changing the seed to the IPG.

Theorem 7.2 : Given a set of L distinct sample sequences, the probability $C_{L,n,k}$ that at least one of the sequences possesses the RIS property is:

[1] When $k = 1$, the test length between check points is always a multiple of $2^k - 1 = 1$. Therefore, the RIS probability of $2^{-(n-2)k}$ is not applicable to the case where $k = 1$.

$$C_{L,n,k} = 1 - (1-p)^L, \tag{7.46}$$

where p is the probability that a single sequence possesses the RIS property.

Proof : Assume that the probability for a sequence to possess the RIS property is p. Given a choice of two sample sequences, the probability that at least one possesses the RIS property is $p+(1-p)p$. In general, given L fault-free sequences, the probability $C_{L,n,k}$ that at least one of the sequences possesses the RIS property follows a geometric distribution, i.e.,

$$\begin{aligned} C_{L,n,k} &= p + (1-p)p + (1-p)^2 p + ... + (1-p)^{L-1} p \\ &= \sum_{i=1}^{L} (1-p)^{i-1} p \\ &= 1 - (1-p)^L. \end{aligned} \tag{7.47}$$

Q.E.D.

The probability $C_{L,n,k}$ is a measure of confidence of success in finding at least one sequence with the RIS property from a set of L possible sample sequences. Obviously, $C_{L,n,k} \rightarrow 1$ as $L \rightarrow \infty$. Thus, to find one sequence with the RIS property can be highly probable if L can be made sufficiently large.

However, generating a large number of sample sequences requires significantly more CPU time effort than generating only one such sequence since the fault-free sequence generation is usually achieved by *logic simulation* in the design phase. To successfully implement the SMS scheme, we have to develop efficient techniques for generating the large set of sample sequences as well as for testing these sequences for the RIS property. Techniques for these tasks are presented in the next section.

7.4 Fast Realization of the SMS Scheme

In this section, we present an efficient approach for realizing the SMS scheme by designing specific seeds for both the IPG and the SA that result in a fault-free CUT to yield identical signatures at predetermined check points. The design or selection of the specific IPG and SA seeds is achieved by generating and testing the sample sequences for the RIS property.

7.4.1 Efficient Sample Sequences Generation

A simple way of generating L sample sequences is to apply L different sets of random vectors to a CUT and to choose a vector set that generates an output sequence with the RIS property as the test set for the CUT. To generate L sequences, one may try L IPG LFSRs with different feedback polynomials. Unfortunately, this method implies simulating the CUT for $L \times l$ test vectors. Thus, the corresponding time complexity is $O(lL)$ (disregarding the dependency of the simulation on the CUT parameters). This may become unacceptable for large l and L. A more efficient technique is presented next.

First, consider the case where every input pattern to the CUT results in a single bit output, i.e., l patterns yields l output bits. By logic simulation, one can obtain a l-bit fault-free output sequence by shifting the IPG LFSR l clock cycles to generate l test vectors to the CUT. Denote the output sequence by b_1, b_2, ..., b_l. Shifting the IPG LFSR for one additional clock cycle to generate an extra test vector for the CUT and simulating the CUT for the extra test vector yields the additional output bit b_{l+1}. From the $(l+1)$-bit sequence, there exist two possible sequences of l consecutive bits. One is the sequence b_1, b_2, ..., b_l. The other is b_2, b_3, ..., b_{l+1}. The probability for the two sequences to be identical is extremely small (approximately 2^{-l}, see Appendix B for proof). Clearly, by shifting the IPG LFSR for a total of $l + L - 1$ additional clock cycles, i.e., simulating the CUT with a total of $l + L - 1$ test vectors, yields $l + L - 1$ output bits, $b_1, b_2, ..., b_{l+L-1}$. Such a $l + L - 1$ bit sequence yields the choice of L fault-free sequences of length l. The time complexity of

this sequence generation technique is only $O(l+L)$, which is simply a linear increase in the necessary time to generate a single l-bit fault-free output sequence.

The above method for generating sample sequences can also be applied to the cases where a scan chain is used. With a scan chain of length m, each input vector yields m output bits. In this case, the IPG LFSR requires only $\lceil (l+L-1)/m \rceil$ shifts to generate L fault-free sequences. Thus, the sequence generation time complexity is only $O(\frac{l+L}{m})$ if an m-stage scan chain is used.

The generation of the sample sequences discussed above implies that the seed of the IPG could be chosen such that any of the L sequences of length l could result from the CUT.

7.4.2 IPG and SA Seeds Selection

With the technique developed above, L fault-free sample sequences can be generated efficiently by logic simulation. It was shown that a fault-free sequence of length $l+L-1$ yields a choice of L sequences of length l. Each of the latter sequences is actually a subsequence of the $(l+L-1)$-bit sequence. In this section, given a $(l+L-1)$-bit fault-free sequence (implicitly L IPG seeds to choose from), an efficient algorithm is developed that finds the desired seeds for the IPG and the SA by testing the l-bit subsequences for the RIS property. The following example illustrates the basic idea of the algorithm.

Example: *Assume that to test a CUT requires twelve pseudorandom vectors, i.e.,* $l = 12$. *Use the 2-stage LFSR shown in Fig. 7.32 to check three signatures at the check points* $l_1 = 4$, $l_2 = 8$, *and* $l_3 = l = 12$, *i.e., we check a signature after every four bits have been shifted into the LFSR. Assume* $L = 4$ *and the* $(l+L-1) = (12+4-1) = 15$*-bit fault-free sequence* $\{R\}_{15}$ *to be:*

$$b_1\ b_2\ ..\ b_{14}\ b_{15} = 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0.$$

Starting with different seeds in the SA, the four possible transient SA LFSR state sequences when shifting in the first 12-bits of $\{R\}_{15}$, $b_1\ b_2\ \ldots\ b_{12}$, *are shown below:*

	seed	1	2	3	4	5	6	7	8	9	10	11	12
1.	(00)	(10)	(01)	(00)	**(00)**	(10)	(11)	(11)	**(11)**	(01)	(10)	(01)	**(00)**
2.	(01)	(00)	(10)	(01)	**(10)**	(01)	(10)	(01)	**(00)**	(00)	(00)	(10)	**(01)**
3.	(10)	(01)	(00)	(10)	**(11)**	(11)	(01)	(00)	**(10)**	(11)	(01)	(00)	**(10)**
4.	(11)	(11)	(11)	(11)	**(01)**	(00)	(00)	(10)	**(01)**	(10)	(11)	(11)	**(11)**

Clearly, for all four different seeds, the first 12-bits of the input sequence $\{R\}_{15}$ *do not yield identical signatures at the predetermined check points* $l_1 = 4$, $l_2 = 8$, $l_3 = 12$. *Thus, we need to consider another 12-bit subsequence of* $\{R\}_{15}$. *To do so, we can simply shift the 13th bit,* b_{13}, *into the SA LFSR. Then, we obtain another four LFSR transient state sequences of length 12 as shown under the big over-brace:*

	seed	1	2	3	4	5	6	7	8	9	10	11	12	13
1.	(00)	(10)	(01)	(00)	(00)	**(10)**	(11)	(11)	(11)	**(01)**	(10)	(01)	(00)	**(00)**
2.	(01)	(00)	(10)	(01)	(10)	**(01)**	(10)	(01)	(00)	**(00)**	(00)	(10)	(01)	**(10)**
3.	(10)	(01)	(00)	(10)	(11)	**(11)**	(01)	(00)	(10)	**(11)**	(01)	(00)	(10)	**(11)**
4.	(11)	(11)	(11)	(11)	(01)	**(00)**	(00)	(10)	(01)	**(10)**	(11)	(11)	(11)	**(01)**

Examination of the transient state sequences shows that the third transient state sequence enters the same LFSR state after every four shifts. Thus, in testing, if we initialize the SA LFSR to state (01) and apply the 12-bit fault-free sequence $b_2\ b_3\ \ldots\ b_{13}$ = *1 1 0 1 0 1 1 0 0 1 1 0 to the SA (this implies that the IPG seed must be such that the CUT yield the sequence* $b_2\ b_3\ \ldots\ b_{13}$*), we will obtain three identical signatures at the predetermined check points. The value of the expected signatures is* (11). □

For the above example, all the transient LFSR state sequences were assumed to be pre-calculated and stored. In general, for large k and l, storing these state sequences may

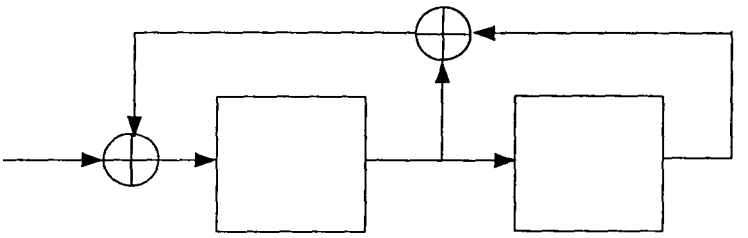


Figure 7.32: LFSR for the example.

require a large memory space. Next, we present an efficient IPG and SA seeds selection algorithm based on the idea illustrated by the example, but which eliminates the requirement of storing these transient state sequences. Assume the $(l + L - 1)$-bit fault-free sequence is $\{R\}_{(l+L-1)} = b_1, b_2, ..., b_{l+L-1}$, where b_1 is the first bit to compact. Denote the signature obtained at the jth check point with the ith seed by sig_{ij}, where $0 \leq i \leq 2^k - 1$ and $0 \leq j \leq n$, and sig_{i0} represents the corresponding seed used in testing. The pseudocode for the seeds selection algorithm follows:

RISsearch.process($\{R\}_{(l+L-1)}, L, n, k, l_1, l_2, ..., l_n$)
  $sig_{00} = 0$;
  $l_0 = 0$;
  for(i=1 to n) do
    j = i - 1;
    initialize the SA LFSR with sig_{0j};
    calculate the signature sig_{0i} by shifting the sequence bits
    $b_{l_j+1}, b_{l_j+2}, ..., b_{l_j+l_i}$ into the SA LFSR;
  endoffor
  SigCalculation.process();/*using Lemma 7.2 to compute signatures with other seeds.*/
  i = flag = 1;
  while ((i $\leq$ L) && (flag)) do
    for (m=0 to ($2^k - 1$)) do
      if(signature sig_{m1}, sig_{m2}, ..., sig_{mn} are not identical) then

```
            for(j=0 to n) do
                initialize the SA LFSR to the value of sig_mj;
                calculate a new sig_mj by shifting bit b_{i+l_j} into the SA LFSR;
            endoffor
        else
            flag = 0;
            break;
        endofif
        i = i + 1;
    endoffor
endofwhile
if(i > L) then return "No RIS sequence found ";
else return i (the position where the RIS sequence starts), sig_m1 (the identical signature
    value), and sig_m0 (the SA seed required in testing);
endofRISsearch.

SigCalculation.process()
initialize the SA LFSR with sig_10 = 1;
for(i=1 to n) do
    calculate sig_1i by shifting the SA LFSR l_i clock cycles in its autonomous mode;
endoffor
for(i=0 to n) do
    for(j=2 to 2^k - 1) do
        calculate sig_ji by shifting the SA LFSR one clock cycle in its autonomous mode;
    endoffor
endoffor
for(i=1 to n) do
    initialize the SA LFSR with sig_1i;
    for(j=1 to 2^k - 1) do
        calculate the signature sig_ji = sig_ji ⊕ sig_0i;
    endoffor
endoffor
endofSigCalculation
```

Upon obtaining the results from the **RISsearch.process**, sig_{m0} (returned from **RISsearch.process**) is the proper seed to the SA. To find the required seed for the IPG, one may simply set the IPG LFSR to the same initial state as that used for generating the sequence $\{R\}_{(l+L-1)}$ and then simulate the IPG LFSR in its autonomous mode for i (another returned result from **RISsearch.process**) clock cycles. The resulting state of the IPG LFSR forms the proper IPG seed, say $Seed^{IPG}$. When testing the CUT, the SMS scheme initializes the IPG and the SA respectively with the seeds $Seed^{IPG}$ and sig_{m0} (the initialization is conducted only once prior to testing), and then runs the IPG for l clock cycles to apply test vectors to the CUT. If the CUT is fault-free, all the signatures obtained at predetermined check points will be identical and equal to sig_{m1} (yet another returned result from **RISsearch.process**).

For many practical values of n and k, the execution of the **RISsearch.process** is very fast although its time and space complexities are high, being $O(2^k nL)$ and $O(n2^k)$, respectively. For example, when $l = 2^{20}$, the execution of **RISsearch.process** takes less than 10 seconds on a Sun Sparc 2 workstation for various n and k shown in Table 7.5 (L is a function of n and k. See next section).

7.5 Cost and Performance

This section addresses the issue of the magnitude of L, and hence the CPU time expenses required to ensure a certain confidence of finding proper IPG and SA seeds, or equivalently the confidence of finding a RIS sequence. Due to the fact that test lengths between check points that are multiples of $2^k - 1$ when $k > 1$ can easily be avoided, e.g., by changing n, k, and/or changing check point scheduling, in this section, unless otherwise emphasized, we discuss only the cases where the test length between at least one pair of check points is not a multiple of $2^k - 1$, i.e., the cases where $p = 2^{-(n-2)k}$. For the case where the test length between each pair of check points is a multiple of $2^k - 1$, i.e., where the RIS probability

$p = 2^{-(n-1)k}$, the feasibility and aliasing performance are the same as those discussed in [Wu93].

7.5.1 L vs. Aliasing Performance

Given n, k, and a desired confidence C of success in finding a RIS sequence, the required L can be obtained by solving Eqn. 7.46 with $C_{L,n,k} = C$. Example values are shown in Figs. 7.33 and 7.34.

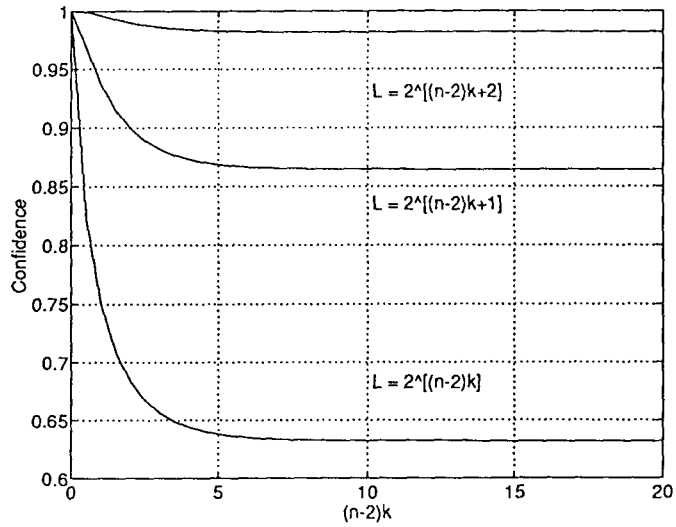


Figure 7.33: Confidence vs. (n-2)k.

As shown in Fig. 7.33, when $(n-2)k$ is small, the confidence is high. For example, when $n = 2$, and hence $(n-2)k = 0$, the confidence is always 100% for non-singular SA LFSRs. However, when $(n-2)k$ increases, i.e., when the RIS probability p decreases, the confidence decreases rapidly and saturates at a certain value, depending on the value of L. However, as shown in Fig. 7.34, for fixed p and hence fixed P_{al}, the confidence goes up very quickly with increasing L. Table 7.6 shows example saturation levels (lower bounds) on confidence for given L.

For the examples shown in Table 7.5, Table 7.7 shows the required L to ensure

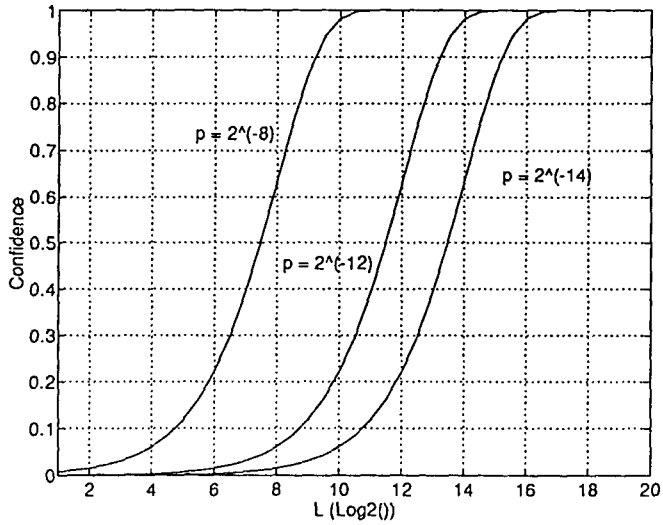


Figure 7.34: Confidence vs. L.

L	C (%)
$2^{(n-2)k}$	63.21
$2^{(n-2)k+1}$	86.47
$2^{(n-2)k+2}$	98.17
$2^{(n-2)k+3}$	99.97

Table 7.6: L vs. lower bounds on confidence, given $P_{al} \approx 2^{-nk}$.

various confidences (C). For example, to find a sequence that yields three identical 8-bit signatures, thus $P_{al} \approx 2^{-24}$, $L = 2^9$ is required to ensure a confidence of 86.47%, and $L = 2^{10}$ is required for a confidence of 98.17%. For the case where $n = 2$, the confidence is always 100% for non-singular SA LFSRs.

7.5.2 CPU Time Overhead for Implementing the SMS Scheme

Using the proposed method to implement the SMS scheme entails some computational costs (CPU time) over that required for a single signature scheme. The CPU time costs include two parts. One is from the generation of the sample sequences; the other is from the effort

n	k	P_{al}	RIS probability	Required L		
			$p = 2^{-(n-2)k}$	C=86.47%	C=98.17%	C=100%
2	k	2^{-2k}	1	n/a	n/a	1
3	8	2^{-24}	2^{-8}	2^9	2^{10}	∞
5	4	2^{-20}	2^{-12}	2^{13}	2^{14}	∞
9	2	2^{-18}	2^{-14}	2^{15}	2^{16}	∞

Table 7.7: Examples of n, k, confidence and required L.

in selecting the IPG and the SA seeds by testing the sample sequences for the RIS property.

The effort for generating the sample sequences is in fact the CPU time required for simulating the fault-free CUT to get $L-1$ extra output bits. For a given CUT, the required CPU time for obtaining $L-1$ extra output bits by logic simulation is proportional to $(L-1)$. Similarly, the CPU time for obtaining one l-bit sequence, which corresponds to the CPU time costs for implementing an SS scheme, is proportional to l. Denote the CPU time required for implementing an SS scheme by T_{SS}. Denote the CPU time for generating the $L-1$ extra output bits by T_G, and the time for testing the sample sequences for the RIS property by T_{RIS}. Then, the CPU time overhead for implementing the SMS scheme over the CPU time effort for implementing an SS scheme is:

$$\begin{aligned} \frac{\Delta T}{T_{SS}} &= \frac{T_G}{T_{SS}} + \frac{T_{RIS}}{T_{SS}} \\ &= \frac{L-1}{l} + \frac{T_{RIS}}{T_{SS}}, \end{aligned} \tag{7.48}$$

where ΔT represents the extra CPU time required for implementing the SMS scheme compared to that for implementing an SS scheme. $\frac{T_{RIS}}{T_{SS}}$ is usually very small for many practical values of n and k (see Section 7.4.2) and independent of CUT. Hence, in the following, for convenience, the CPU time overhead is given in terms of $\frac{T_G}{T_{SS}}$ (percentage) and absolute T_{RIS} (seconds on a Sun Sparc 2 workstation).

Assume an output sequence length $l = 2^{20}$, which is not too long in practice [Keller90][Gelsinger89][Hagihara92], especially when a scan chain is used. Corresponding

n	k	P_{al}	C = 98.17%			C = 100%		
			L	T_G/T_{ss}	T_{RIS} (sec.)	L	T_G/T_{ss}	T_{RIS} (sec.)
2	16	2^{-32}	n/a	n/a	n/a	1	0	< 4
3	8	2^{-24}	2^{10}	0.10%	< 6	∞	∞	∞
5	4	2^{-20}	2^{14}	1.56%	< 6	∞	∞	∞
9	2	2^{-18}	2^{16}	6.25%	< 7	∞	∞	∞

Table 7.8: Example CPU time overheads for the SMS scheme.

values of $\frac{T_G}{T_{SS}}$ and T_{RIS} are shown in Table 7.8. In Table 7.8, T_{RIS}'s are obtained on a Sun Sparc 2 workstation from the experiments reported in Section 7.6, where $l = 2^{20}$. For example, when $n = 2$ and $k = 16$, i.e., to find the proper seeds that yield two identical 16-bit signatures, no CPU time is required for generating extra output bits (follows from Corollary 7.1), and the time required for selecting the IPG and the SA seeds is less than 4 seconds; when $n = 3$ and $k = 8$, the CPU time overhead required for generating the extra output bits relative to the time for implementing an SS scheme is only 0.10%, while the CPU time spent on the IPG and SA seed selection is less than 6 seconds. Evidence reveals that such percentages of CPU time overheads are acceptable in practice. For example, for a ~50k gate CUT, the 0.10% CPU time overhead required for generating the extra output bits represents only 4.72 minutes on a Sun-4/370 (25MHz) according to the logic simulation data reported in [Hagihara92], and only 0.012 sec. on an IBM mainframe according to the data reported in [Keller91]. Since implementing the SMS scheme requires only a fault-free output sequence, simulation techniques based on functional or behavioral level circuit descriptions can be used, thus making the sample sequences generation process much faster. For example, according to [Hagihara92], functional simulation is more than five times faster than gate-level logic simulation.

The CPU time overhead can be further reduced by combining the sample sequences generation with the RIS sequence testing, i.e., generating a fault-free sequence and concurrently determining whether any of its l-bit subsequences possesses the RIS property.

n	k	P_{al}	average # of extra simulation cycles (average L -1)	average T_G/T_{ss}	T_{RIS} (sec.)
2	16	2^{-32}	0	0	< 4
3	8	2^{-24}	$2^8 - 1$	0.02%	< 6
5	4	2^{-20}	$2^{12} - 1$	0.39%	< 6
9	2	2^{-18}	$2^{14} - 1$	1.56%	< 7

Table 7.9: Example average CPU time overheads of the SMS scheme.

Once such a subsequence is found, the process can be terminated. In this case, the average CPU time overhead can be significantly reduced since the expected number of extra output bits that must be generated is much smaller than $(L - 1)$. Since the expected value of L is $2^{(n-2)k}$, which is the mean value of a random variable characterized by a geometric distribution with parameter $2^{-(n-2)k}$ [Larson82], the expected number of extra output bits that need to be generated is $2^{(n-2)k} - 1$. Table 7.9 shows example average CPU time overheads. For example, for checking three identical 8-bit signatures, the average CPU time requirement is only 0.02% of T_{SS} in addition to 6 sec. for T_{RIS}.

7.5.3 A Special Case: k = 1

A special case of the SMS scheme arises for $k = 1$, which yields $P_{al} \approx 2^{-n}$ if n 1-bit signatures are checked. In this case, the hardware requirement of the SMS scheme is only a 1-stage LFSR. Moreover, the storage of the single reference required in the general case can be avoided if the state of the 1-stage LFSR at each check point is directly used as a pass/fail signal [Wu92b]. The case where $k = 1$ can be considered as a minimal hardware requirement for BIST data compaction [Park91][Wu92b].

When $k = 1$, the test length between check points is always a multiple of $2^k - 1 = 1$. In this case, $p = 2^{-(n-1)k}$. By solving Eqn. 7.46 for L with $C_{L,n,k} = C$ and $p = 2^{-(n-1)k}$, we can find the required L similarly to the analysis in Section 7.5.1 for the

L	C (%)
$2^{(n-1)k+1}$	86.47
$2^{(n-1)k+2}$	98.17
$2^{(n-1)k+3}$	99.97

Table 7.10: L vs. lower bounds on confidence for $p = 2^{-(n-1)k}$.

cases where $p = 2^{-(n-2)k}$. Example lower bounds on the confidence for given L appear in Table 7.10. From Table 7.10, when $k = 1$ and $P_{al} \approx 2^{-n}$ is desired, $L = 2^n$ is required to ensure 86% confidence, or $L = 2^{n+1}$ for a confidence of over 98%. Theoretically, with a 1-stage LFSR, the SMS scheme can achieve arbitrarily small aliasing if L is sufficiently large. Assuming $k = 1$ and $l = 2^{20}$, achieving the typical aliasing probability of 2^{-16} [Kuban84][LeBlanc84][Gelsinger86][Dervisoglu89] requires only 6.25% CPU time overhead for 86% confidence, or 12.5% overhead for 98% confidence. On average, the CPU time overhead for achieving $P_{al} \approx 2^{-16}$ with $k = 1$ is only 3.125% in this case.

7.6 Experimental Results

To confirm the feasibility of the SMS scheme, nearly 500 experiments on the ISCAS'85 benchmark circuits [Brglez85] were conducted. The check points were periodically scheduled, thus making the hardware requirement for the control of checking multiple signatures as small as that for the control of any single signature scheme (see Chapter 5). In the experiments, the polynomials were chosen from those listed in [Bardell87]. For each given set of n and k, we performed three trials with each of the circuits. Table 7.11 shows the experimental results for $l = 2^{16}$ obtained under the condition that the test length between check points is not a multiple of $2^k - 1$.

From Table 7.11, two identical 16-bit signatures, thus $P_{al} \approx 2^{-32}$, can be found without generating any extra output bit. To find proper IPG and SA seeds that yield three identical 8-bit signatures, hence $P_{al} \approx 2^{-24}$, requires the generation of $2^{7.60} = 194$ extra bits

Circuit Name	*# of extra output bits*			
	n=9,k=2	*n=5,k=4*	*n=3,k=8*	*n=2,k=16*
$C432$	$2^{13.80}$	$2^{13.19}$	$2^{8.30}$	0
$C499$	$2^{13.28}$	$2^{10.82}$	$2^{8.68}$	0
$C880$	$2^{14.50}$	$2^{12.88}$	$2^{6.71}$	0
$C1355$	$2^{13.28}$	$2^{10.82}$	$2^{8.68}$	0
$C1908$	$2^{13.75}$	$2^{12.65}$	$2^{6.82}$	0
$C2670$	$2^{14.32}$	$2^{10.82}$	$2^{8.41}$	0
$C3540$	$2^{11.21}$	$2^{11.09}$	$2^{6.72}$	0
$C5315$	$2^{13.57}$	$2^{10.07}$	$2^{6.61}$	0
$C6288$	$2^{12.57}$	$2^{9.32}$	$2^{8.44}$	0
$C7552$	$2^{14.37}$	$2^{11.84}$	$2^{6.57}$	0
average	$2^{13.47}$	$2^{11.35}$	$2^{7.60}$	**0**
Max. T_{RIS} (sec.)	6.08	3.87	2.25	0.95

Table 7.11: Experimental results for $l = 2^{16}$
(Each entry is an average of 3 trials).

on average. If nine identical 2-bit signatures are sought, then an average of $2^{13.47} \approx 11,346$ extra output bits must be generated. In comparison with the theoretical average number of extra output bits shown in Table 7.9, the row of the entry *average* in Table 7.11 gives the average number of the extra output bits obtained in the experiments. These results are very close to the theoretical expectation. The last row of Table 7.11 shows the worst case CPU time spent on selecting the IPG and the SA seeds, i.e., the maximum T_{RIS}, obtained on a Sun Sparc 2 workstation. Table 7.12 reports more results obtained under the same experimental conditions but for test length $l = 2^{20}$.

For the case where $k = 1$, we conducted 30 experiments on the benchmark circuits. In the experiments, $n = 16$, thus $P_{al} \approx 2^{-16}$. The experimental results appear in Table 7.13.

To confirm the claim that the RIS probability $p = 2^{-(n-2)k}$ is only for the cases where the test length between check points is not a multiple of $2^k - 1$, we performed another 200 experiments for the cases where $n = 2$ and $k = 2$, and where $n = 2$ and $k = 4$. Theoretically,

Circuit Name	*# of extra output bits*			
	n=9,k=2	*n=5,k=4*	*n=3,k=8*	*n=2,k=16*
$C432$	$2^{12.35}$	$2^{12.23}$	$2^{5.04}$	0
$C499$	$2^{13.02}$	$2^{9.52}$	$2^{7.04}$	0
$C880$	$2^{14.33}$	$2^{11.01}$	$2^{8.17}$	0
$C1355$	$2^{13.02}$	$2^{9.52}$	$2^{7.04}$	0
$C1908$	$2^{13.18}$	$2^{10.82}$	$2^{9.20}$	0
$C2670$	$2^{10.73}$	$2^{11.92}$	$2^{6.62}$	0
$C3540$	$2^{14.55}$	$2^{10.73}$	$2^{7.19}$	0
$C5315$	$2^{13.97}$	$2^{11.41}$	$2^{5.76}$	0
$C6288$	$2^{14.30}$	$2^{12.38}$	$2^{7.63}$	0
$C7552$	$2^{11.30}$	$2^{11.57}$	$2^{8.33}$	0
average	$\mathbf{2^{13.08}}$	$\mathbf{2^{11.11}}$	$\mathbf{2^{7.20}}$	**0**
Max. T_{RIS} (sec.)	6.75	5.54	5.30	3.99

Table 7.12: Experimental results for $l = 2^{20}$
(Each entry is an average of 3 trials).

Circuit Name	*# of extra output bits*
$C432$	$2^{13.97}$
$C499$	$2^{12.22}$
$C880$	$2^{14.12}$
$C1355$	$2^{12.22}$
$C1908$	$2^{16.33}$
$C2670$	$2^{14.73}$
$C3540$	$2^{15.87}$
$C5315$	$2^{15.74}$
$C6288$	$2^{14.86}$
$C7552$	$2^{16.47}$
average	$\mathbf{2^{14.10}}$

Table 7.13: Experimental results for $n = 16$ and $k = 1$
(Each entry is an average of 3 trials).

in these cases, the proper IPG and SA seeds can be found without generating any extra output bit if the test length between check points is not a multiple of $2^k - 1$; otherwise, extra output bits must be generated to find the seeds. Experimental results confirmed the theoretical claims.

To confirm the claim that the RIS probability $p = 2^{-(n-2)k}$ is true only for non-singular LFSRs, another experiment was conducted where the primitive LFSR of degree 20 from [Bardell87] was used. This LFSR is singular (see Section 7.2.2). In the experiment, $n = 2$ and the test length between check points is not a multiple of $2^k - 1$. According to Corollary 7.1 and Lemma 7.5, in this case, extra output bits must be generated in order to find proper IPG and SA seeds that yield two identical signatures. Experimental result again supports the claim.

7.7 Discussions

In Lemma 7.5, it is shown that the RIS probability is $2^{-(n-1)k}$ if the test length between each pair of check points is a multiple of $2^k - 1$. Here, we first prove a lemma useful to reduce the complexities of the **RISsearch.process()** for the cases where the test length between each pair of check points is evenly divisible by $2^k - 1$.

Lemma 7.6: Given two seeds A and B, and that test length between each pair of check points is evenly divisible by $2^k - 1$, if $S_A(l_1) = S_A(l_2) = ... = S_A(l_n) = S$, then $S_B(l_1) = S_B(l_2) = ... = S_B(l_n) = S'$, or vice versa, where $A \neq B$ and $S \neq S'$.

Proof: To prove the lemma, we only need to prove that if $S_A(l_i) = S_A(l_j) = S$ then $S_B(l_i) = S_B(l_j) = S'$, or vice versa, when the test length between check points l_i and l_j is a multiple of $2^k - 1$. In this case,

$$S_A^{auto}(l_i) = S_A^{auto}(l_j),$$

$$S_B^{auto}(l_i) = S_B^{auto}(l_j).$$

According to Lemma 7.2, we have,

$$S_A(l_i) = S_0(l_i) \oplus S_A^{auto}(l_i),$$
$$S_A(l_j) = S_0(l_j) \oplus S_A^{auto}(l_j),$$

and

$$S_B(l_i) = S_0(l_i) \oplus S_B^{auto}(l_i),$$
$$S_B(l_j) = S_0(l_j) \oplus S_B^{auto}(l_j).$$

Therefore, if $S_0(l_i) \neq S_0(l_j)$, then $S_A(l_i) \neq S_A(l_j)$ and $S_B(l_i) \neq S_B(l_j)$. If $S_0(l_i) = S_0(l_j)$, then $S_A(l_i) = S_A(l_j) = S$ and $S_B(l_i) = S_B(l_j) = S'$, where $S = S_0(l_i) \oplus S_A^{auto}(l_i)$ and $S' = S_0(l_i) \oplus S_B^{auto}(l_i)$. If $A \neq B$, then $S_A(l_i) \neq S_B(l_i)$, thus $S \neq S'$.

Q.E.D.

According to Lemma 7.6, if one seed yields identical signatures at predetermined check points, all the other seeds will also yield identical signatures at the same check points, given that the test length between check points is a multiple of $2^k - 1$. The values of the signatures obtained with different seeds are different. Lemma 7.6 has two applications. The first is to reduce the time complexity of the **RISsearch.process()** when the test length between check points is evenly divisible by $2^k - 1$. To do so, we can simply set all $k = 1$ in the **RISsearch.process()**, thus reducing the **RISsearch.process()**'s time and space complexities to $O(nL)$ and $O(n)$, respectively (a detailed algorithm of the simplified **RISsearch.process()** is given in [Wu93]). However, we must point out that when the test length between all pairs of check points is evenly divisible by $2^k - 1$, the required L for ensuring a given confidence of success will be 2^k times greater than that for the cases where the test length between at least one pair of check points is not evenly divisible by $2^k - 1$. The larger L may impose significantly more CPU time for logic simulation. Thus, whenever possible, one should avoid making the test length between all pairs of check points evenly divisible by $2^k - 1$. This can be easily done by changing either k, n, l, or the check points.

The second application of Lemma 7.6 is to make the identical signatures equal to a specific value when the test length between check points is evenly divisible by $2^k - 1$. To do so, one may simply use any seed to find the RIS sequence, and then use the algorithm provided in [McAnney86] to make the signatures equal to a given value.

7.8 Extensions

7.8.1 Applications to MISR

In the above, a single input LFSR was assumed for collecting signatures. An MISR may be used for signature analysis. In this case, if the length of the MISR is not too long, the SMS scheme can be easily applied. Otherwise, one may have difficulty to apply the SMS scheme in a straightforward way due to the following two reasons. First, when the length of a MISR, i.e., k, increases, L increases exponentially if $n > 2$, thus imposing a significant CPU time requirement for sample sequences generation. Secondly, when k increases, the amount of CPU time spent on the IPG and SA seed selection will also increase exponentially because of the complexity of the **RISsearch.process()**.

To make the SMS scheme practical for long MISRs, we have to make k small. To do so, we have at least the following three choices. First, one can use a partial-length MISR [Pomeranz92] as shown in Fig. 7.35. Secondly, one can simply select a partial set of signature bits of a MISR such that this set of bits of each signature be identical at all check points.

Thirdly, when a CUT has a large number of output lines, it is much more economical in silicon area to use a space compactor followed by a short MISR compared to using a long MISR which has the same number of stages as the number of CUT output lines [Reddy88][WuM92][Zorian93]. In fact, the SMS scheme has been successfully implemented in the BIST of a VLSI Viterbi decoder with a MISR [Bonek93], where four signatures collected by a 5-stage MISR following a space compactor were checked periodically.

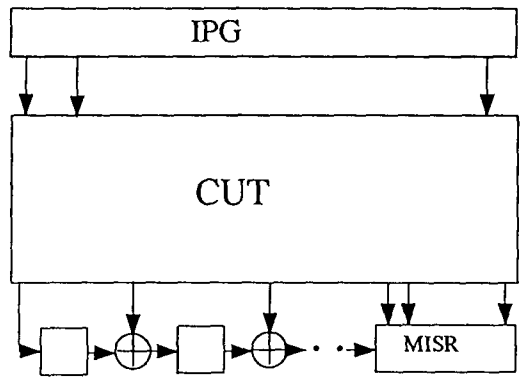


Figure 7.35: Output compaction using a partial-length MISR.

7.8.2 Extensions to the FMS

As shown in Section 7.5, the CPU time overhead of the SMS scheme is proportional to $2^{(n-2)k}$ since L is proportional to $2^{(n-2)k}$. Thus, if extremely small aliasing is required, the SMS scheme may impose a substantial CPU time overhead. One can combine the SMS scheme with the FMS scheme to achieve the required aliasing but still keep low hardware requirements. The basic idea of combining these schemes is to make only some of the reference signatures, instead of all of them, identical, thus making $m < n$, and then to use the FMS scheme for implementation.

The techniques developed in this chapter can also be applied to the FMS scheme to minimize its hardware requirements. Instead of searching for a sequence that yields identical signatures as in the SMS scheme, one can search for a sequence which yields signatures that are easily logic-minimizable when implementing the FMS scheme. Determining whether a set of signatures is optimal for logic minimization is an NP-complete problem. Thus, to successfully find a sequence that yields optimal logic-minimizable signatures, heuristic techniques should be developed for checking the "optimality".

7.8.3 Applications to Weighted Random Testing

In the development of the SMS scheme, we assumed pseudorandom testing. This gives us the freedom to design or select the seed to the IPG. For weighted random testing, however, the freedom no longer exists since the IPG seed in weighted random testing is usually predefined. Thus, in general, the SMS scheme is not applicable to weighted random testing. However, there is a special case where the SMS scheme can be applied. This is the case where $n = 2$. This is because in this case where two signatures are checked, the two references can always be made identical by designing the SA seed alone (see Theorem 7.1).

7.9 Conclusions

Conventionally, checking n signatures requires n references. In this chapter, a scheme that checks n signatures against a single reference has been proposed. The scheme is referred to as the SMS (single-reference multiple signature) scheme. Its basic idea is to make all the n references identical. By all being identical, in effect, only one reference is necessary for checking n signatures. With the SMS scheme, the hardware required for checking n signatures is reduced to a minimum, i.e., reduced to the same as that for checking a single signature. An approach for implementing the SMS scheme was developed which selects (or designs) the seeds to both the IPG (input pattern generator) and SA (signature analyzer) such that the n signatures obtained at arbitrarily predetermined check points are identical if the CUT is fault-free. Since the approach is based on a simple manipulation of the fault-free output sequence, no circuit modification is required. To efficiently implement the SMS scheme, two techniques were also developed; one for the efficient generation of sample fault-free sequences, and the other for the fast selection of required IPG and SA seeds. With these techniques, the CPU time overhead for implementing the SMS scheme is small. For example, to check two identical 16-bit signatures, thus reducing the conventional single signature scheme's aliasing by a factor of 2^{16}, the total CPU time overhead for

implementation is less than 4 seconds on a Sun Sparc 2 workstation when the test length $l = 2^{20}$, independently of the size of the CUT. For simplicity, this chapter assumed a LFSR for data compaction. However, the SMS scheme can also be applied to other types of data compactors, e.g., MISRs and CAs. E.g., the SMS scheme has been successfully implemented with a MISR in the BIST of a VLSI Viterbi decoder.

Chapter 8

Conclusions

8.1 Summary

BIST usually consists two major functions known as *on-chip test pattern generation* and *test response evaluation.* There are two major difficulties regarding test response evaluation. The first is the difficulty to reduce aliasing while still maintaining reasonably small hardware requirements. The second difficulty is accurately assessing the impact of aliasing on the overall test quality of a BIST scheme. Recent research has shown that multiple intermediate signature analysis is a promising solution to these difficulties. In Chapters 4 and 5 of this dissertation, we showed the significant impact of checking multiple intermediate signatures on the reduction of aliasing and the computational efforts for calculating exact fault coverage of BIST schemes.

Based on the investigation of an aliasing model for multiple intermediate signature analysis, Chapter 4 addressed fault coverage models for predicting the fault coverage of multiple intermediate signature analysis. More specifically, a comprehensive fault coverage model based on a detection probability density function of faults in a CUT was developed in Chpater 4. Practical considerations for implementing multiple intermediate signature analysis, such as possible resource sharing, test result observation and test control, were presented in Chapter 5.

Chapter 6 described a *fuzzy multiple intermediate signature* analysis. Unlike conventional multiple intermediate signature schemes, where each checked signature must correspond to a specific reference on a one-to-one basis for a CUT to be declared good, the fuzzy

signature analysis declares a CUT as good if each checked signature maps to any elements of the same set of references. The removal of the strict one-to-one signature-reference correspondence makes the fuzzy signature scheme very simple and easy to implement. Compared to some other data compaction schemes, the fuzzy signature scheme requires less silicon area for implementation. The aliasing performance and hardware requirements of the fuzzy signature scheme was addressed. Experiments on the benchmark circuits demonstrated the strong ability of the fuzzy signature scheme to achieve very small aliasing as well as to reduce CPU time efforts for calculating exact fault coverage.

Conventionally, checking n signatures requires n references. In Chapter 6, however, it was shown that the fuzzy signature scheme can in some cases check n signatures against fewer references. In fact, the fuzzy signature scheme benifits from fewer references since this not only improves the scheme's aliasing performance but also reduces its hardware requirements for implementation. It is desirable for the fuzzy signature scheme to check n signatures against a single reference. Chapter 7 explored this possibility. As a result, a novel signature analysis scheme known as *single-reference multiple intermediate signature analysis* was developed. The basic idea of the single reference scheme is to make all the references required in the conventional cases identical. By all being identical, in effect, only one is necessary for checking multiple signatures. Due to the use of a single reference for multiple signature analysis, in implementation, the single reference scheme is the same as any single signature scheme. (For this reason, it is also referred to as *minimal hardware multiple signature analysis*). However, compared to single signature schemes, the single reference scheme has the advantages of smaller aliasing, easier exact fault coverage computation, and shorter average test time.

A systematic method for implementing the single reference scheme was also presented in Chapter 7. This method is based on an observation that some fault-free sequences naturally yield identical signatures at fixed check points. The method proposes to carefully design the seeds to both the input pattern generator as well as the signature analyzer

such that all the signatures obtained be identical if the CUT is fault-free and if the input pattern generator and the signature analyzer are initialized with the designed seeds prior to testing. More specifically, two efficient algorithms were developed, one for generating sample fault-free sequences and the other for designing the seeds for both the test pattern generator and the signature analyzer according to the *relaxed identical signature property* of these sample sequences. Although the implementation method requires no modification of the CUT, it entails an extra CPU time overhead in the design phase in addition to the efforts for designing a conventional single signature scheme. However, as shown by theoretical analysis and experimental results, the extra CPU time overhead is generally small. For example, to check two identical 16-bit signatures, thus reducing the conventional single signature schemes' aliasing by a factor of 65536, the total CPU time overhead is only 4 seconds on a Sun Sparc 2 workstation for a test length of 2^{20}, independently of the size of CUTs. The single reference scheme has been successfully implemented in the BIST of a VLSI Viterbi decoder, where four signatures collected by a 5-stage MISR following a space compactor were checked periodically.

8.2 Future Work

In the development of the aliasing model in Chapter 4, we assumed the equal likelihood of occurrence of all the possible error sequences. As a result, the derived aliasing is independent of the scheduling of check points. In practice, however, this may not be true. The occurrence probability of each error sequence is generally a function of many factors, such as the structure of the CUT, the specific test vectors, and the specific order in which the test vectors are applied. Although it is generally impossible to consider all these factors, as a suggestion to future work, the method used in Chapter 4 for the derivation of the fault coverage models may be extended to study the aliasing of multiple intermediate signature analysis based on some other more realistic error models, e.g., those described in Chapter

3. In [Saxena92], the authors showed a simple bound on aliasing probability for the cases where the signatures were periodically scheduled and the test length between each pair of adjacent check points is smaller than $2^k - 1$. It would be interesting to obtain a simple bound for a more general case.

As BIST becomes a widely accepted means for VLSI post manufacturing test, determining how to make use of existing BIST hardware for concurrent testing is an interesting and practical topic [Saluja88][Katoozi92]. In concurrent testing, the latency between the occurrence of a fault and the detection of that fault is crucial to the system dependability. Multiple intermediate signature analysis is known to have the ability to significantly reduce the test time of faulty CUTs. However, no effort on the application of multiple intermediate signature analysis for reducing the latency in concurrent testing has been reported. Thus, a formal investigation of this issue would be of practical importance.

LFSRs are usually classified into *primitive* and *non-primitive* LFSRs. In Chapter 7, the author proposes to further classify primitive LFSRs according to their *singularity*. It was shown in Chapter 7 that *non-singular* LFSRs are helpful to the implementation of the proposed *Single Reference Multiple Intermediate Signature* analysis. For a given LFSR, an algorithm that tests for its singularity was developed in Appendix B. Applying the algorithm presented in Appendix B, many of the commonly-used primitive polynomials have been shown to be non-singular. However, the design of non-singular LFSRs is still an open problem. Any future work on the design of non-singular LFSRs and/or on developing efficient algorithm for testing LFSRs for singularity will be useful.

Bibliography

[Abraham86] Abraham, J.A., and Fuchs, W.K., "Fault and Error Models for VLSI," *Procedings of the IEEE*, Vol. 74, No.5, 1986, pp.639-654.

[Agarwal81] Agarwal, V.K. and Cerny, E., "Store and Generate Built-In-Testing Approach," *Proc. FTCS*, 1981, pp.35-40.

[Agarwal83] Agarwal, V.K., "Increasing Effectiveness of Built-In-Testing By Output Data Modification," *Proc. FTCS-14*, 1983, pp. 227-234.

[Agarwal87] Agarwal, V.K. and Zorian, Y., "An Introduction To An Output Data Modification Scheme," *in Development In Integrated Circuit Testing*, Academic Press Ltd., 1987, pp. 219-256.

[Agrawal75] Agrawal, P. and Agrawal, V.D., "Probabilistic Analysis for Random Test Generation Method for Irridundant Combinational Logic Networks," *IEEE Trans. Comput.* Vol. C-24, 1975, pp.691-695.

[Agrawal82] Agrawal, V.D., and Mercer, M.R., "Testability Measures — What Do They Tell Us ?," *Proc. Int. Test Conf.*, 1982, pp.391-396.

[Agrawal93] Agrawal, V.D., Kime, C.R., and Saluja, K.K., "A Tutorial on Built-In Self-Test — Part 1: Principles," *IEEE J. Design & Test of Computers,* March 1993, pp. 73-82.

[Ahmad90] Ahmad, A., Nanda, N.K., and Garg, K., "Are Primitive Polynomials always Best in Signature Analysis ?" *IEEE J. Design & Test,* Aug. 1990, pp.36-38.

[Aitken88] Aitken, R.C. and Agarwal, V.K., "Aliasing Probability of Non-Exhaustive Randomized Syndrome Tests," *Proc. Int. Conf. CAD*, 1988.

[Aitken89] Aitken, R.C., Xavier, D., Ivanov, A., and Agarwal, V.K., "The Role of an Asymmetric Error Model in Predicting Aliasing of Built-In Self-Test for VLSI," *Proc. Canadian Conf. Electrical. & Computer Eng.,* Sept. 1989, pp.356-361.

[Aitken89b] Aitken, R.C. and Agarwal, V.K., "A Diagnosis method using pseudorandom vectors without intermediate signatures," *Proc. ICCAD*, Nov. 1989, pp.574-577.

[Aitken90] Aitken, R.C., "Open Problems in IC Testing — An Industry Perspective," unpublished manuscipt, 1990.

[Akers85] Akers, S.B., "On the Use of Linear Sums in Exhaustive Testing," *Proc. FTCS,* 1985, pp.148-153.

[Akers89] Akers, S.B. and Jansz, W., "Test Set Embedding in a Built-In Self-Test Environment," *Proc. Int. Test Conf.*, 1989, pp.257-263.

[Archambeau84] Archambeau, E.C., and McCluskey, E.J., "Fault Coverage of Pseudo-Exhaustive Testing," *Proc. FTCS-14*, June 1984, pp.141-145.

[Bardell87] Bardell, P.H., McAnney, W.H., and Savir, J., *Built-In Test for VLSI: Pseudo-random Techniques, John Wiley & Sons*, New York, 1987.

[Barr93] Barr, C., "The Evolution of Pentium," *PC Magazine*, April 27, 1993, pp. 116-117.

[Barzilai83] Barzilai, Z., Coppersmith, D. and Rosenberg, A., "Exhaustive Bit Pattern Generation in Discontiguous Positions with Applications to VLSI Testing," *IEEE Trans. Comput.*, Vol. C-32, No. 2, Feb. 1983, pp. 190-194.

[Bhatt86] Bhatt, S.N., Chung, F.R. and Rosenberg, A.L., "Partitioning Circuits for Improved Testability," *Adv. Res. in VLSI — Proc. 4th MIT Conf.*, Cambridge, MA, 1986, pp.91-106.

[Bhavsar84] Bhavsar, D.K. and Krishnamurthy, B., "Can We Eleminate Fault Escape in Self-Testing by Polynomial Division ?" *Proc. Int. Test Conf.*, 1984, pp. 134-139.

[Bhavsar85] Bhavsar, D.K., "Concatenable Polydividers: Bit-sliced LFSR Chips for Board Self-Test," *Proc. Int. Test Conf.*, 1985, pp. 88-93.

[Blank84] Blank, T., "A Survey of Hardware Accelerator used in Computer-Aided Design," *IEEE J. Design & Test,* Aug. 1984, pp. 21-39.

[Bonek93] Bonek, P., *M.Sc. Thesis*, Dept. Elec. Eng., Univ. of British Columbia, 1993.

[Breuer76] Breuer, M.A., and Friedman, A.D., *Diagosis & Reliable Design of Digital Systems, Computer Science Press, Inc.*, 1976.

[Breuer88] Breuer, M.A., Gupta, R. and Lien, J., "Concurrent Control of Multiple BIT Structures," *Proc. Int. Test Conf.*, 1988, pp. 431-442.

[Brglez85] Brglez, F. and Fujiwara, H., "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. IEEE Int. Symp. Circuits and Systems*, 1985.

[Brglez90] Brglez, F., Gloster, C. and Kedem, G., "Built-In Self-Test with Weighted Random Pattern Hardware," *Proc. Int. Conf. Computer Design*, 1990.

[Carter82] Carter, W.C., "Signature Testing with Guaranteed Bounds for Fault Coverage," *Int. Test Conf.*, 1982, pp.75-82

[Chakrabarty93] Chakrabarty, K. and Hayes, J.P., "Aliasing-Free Error Detection (ALFRED)," *Proc. IEEE VLSI Test Symp.*, 1993, pp. 206-266.

[Cox88] Cox, H., Ivanov, A., Agarwal, V.K., and Rajski, J., "On Multiple Fault Coverage and Aliasing Probiblity Measures," *Proc. Int. Test Conf.*, 1988, pp. 314-321.

[Damiani89] Damiani, M., Olivo, P., Favalli, M., and Riccó, B., "An Analytical Model for the Aliasing Probability in Signature Analysis Testing," *IEEE Trans. CAD*, Vol. 8, No. 11, Nov. 1989, pp.1133-1144.

[Damiani89b] Damiani, M., Olivo, P., Favalli, M., Ercolani, S., and Riccó, B., "Aliasing in Signature Analysis Testing with Multiple Input Shift Registers," *Proc. European Test Conf.*, April 1989, pp. 346-353.

[Damiani91] Damiani, M., Olivo, P., and Riccó, B., "Analysis and Design of Linear Finite State Machines for Signature Analysis Testing," *IEEE Trans. on CAD.*, Vol. 40, No. 9, Sept., 1991, pp. 1034-1045.

[Daniels85] Daniels, R.G. and Bruce, W.C., "Built-In Self-Test Trends in Motorola Microprocessors," *IEEE J. Design & Test,* April 1985.

[Davadas88] Davadas, S., H.T.Ma, A.R.Newton and Sangiovanni-Vincentelli, A., "Optimal Logic Synthesis and Testability: Two Faces of the Same Coin," *Proc. Int. Test Conf.,* Sept. 1988, pp.4-12.

[Debany92] Debany, W.H., et al., "Empirical Bounds on Fault Coverage Loss Due to LFSR Aliasing," *Proc. IEEE VLSI Test Symp.*, 1992, pp. 143-148.

[Dervisoglu89] Dervisoglu, B.I., "Scan Path Achitecture for Pseudorandom Testing," *IEEE J. Design & Test*, Aug. 1989, pp. 32-48.

[Diamantaras91] Diamantaras, K.I. and Jha, N.K., "A New Transition Count Method for Testing of Logic Circuits," *IEEE Trans. CAD.,* Vol. 10, No. 3, March 1991, pp. 407-410.

[Duba88] Duba, P.A., Roy, R.K., Abraham, J.A., and Rogers, W., "Fault Simulation in a Distrituted Environment," *Proc. Design Auto. Conf.*, 1988, pp.686-691.

[Eichelberger78] Eichelberger, E.B. and Williams, T.W., "A Logic Design Structure for LSI Testability," *J. of Design Automation, Fault Tolerant Computing,* Vol. 2, No. 2, 1978, pp.165-178.

[Eichelberger83] Eichelberger, E.B. and Lindbloom, E., "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM J. Research and Development,* Vol. 27, No. 3, May 1983, pp.265-272.

[Eldred59] Eldred, R.D., "Test Routines Based on Symbolic Logical Statements," *Journal of the ACM,* 6, pp. 33-36, January, 1959.

[Fink90] Fink, F., Fuchs, K., and Schulz, M.H., "An Efficient Parallel Pattern Gate Delay Fault Simulation with Accelerated Defected Fault Size Determination Capabilities," *Proc. European Test Conf.*, 1990, pp. 171-180.

[Frohwerk77] Frohwerk, R.A., "Signature Analysis: A New Digital Field Service Method," *Hewlett Packard J.*, May 1977, pp.2-8.

[Fujiwara78] Fujiwara, H. and Kinoshita, K, "Testing Logic Circuits with Compressed Data," *Proc. FTCS-8*, 1878, pp.108-113.

[Fujiwara83] Fujiwara, H. and Shimono, T., "On the Acceleration of Test Generation Algorithms," *IEEE Trans. Comput.*, Vol. C-32, No. 12, Dec. 1983, pp. 1137-1144.

[Gagne91] Gagne, D., "Module Generation in the Cadence Design Environment — User's Manual," in preparation, UBC Elect. Eng.

[Gelsinger86] Gelsinger, P.P, "Built In Self Test of the 80386," *Proc. ICCD*, 1986, pp. 169-173.

[Gelsinger89] Gelsinger, P.P., Iyengar, S., Krauskopf, J., and Nadir, J., "Computer Aided Design and Built In Self Test on the 80486 CPU," *Proc. IEEE ICCD*, 1989

[Goel81] Goel, P., "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Comput.*, Vol. C-30, No. 3, Mar. 1981, pp.215-222.

[Golomb82] Golomb, S.W., *Shift Register Sequences*, Languna Hills, CA, Aegean Park, 1982.

[Gupta90] Gupta, S.K., Pradhan, D.K. and Reddy, S.M., "Zero Aliasing Compression," *Proc. FTCS*, 1990, pp.254-263.

[Hagihara92] Hagihara, Y., Ohkubo, C., and Okamoto, F., "Design For Testability In a 200MFLOPS Vector-Pipelined Processor (VPP)-ULSI," *Proc. 1st Asian Test Symp.*, 1992, pp.223-228.

[Hassan84] Hassan, S.Z. and McCluskey, E.J., "Increased Fault Coverage Through Multiple Signatures," *Proc. FTCS-14*, 1984, pp. 354-359.

[Hassan88] Hassan, A., Rajski, J., and Agarwal, V.K., "Testing and Diagnosis of Interconnects using Boundary Scan Achitecture," *Proc. Int. Test Conf.*, 1988, pp. 126-133.

[Hellebrand92] Hellebrand, S., Tarnick, S. and Rajski, J., "Generation of Vector Patterns Through Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *Proc. Int. Test Conf.*, 1992, pp.120-129.

[Hortensius89] Hortensius, P.D., McLeod, R.D., Pries, W., Miller, D.M., and Card, H.C., "Cellular Automata-Based Pseudorandom Number Gnerators for Built-In Self-Test," *IEEE Trans. CAD,* Vol. 8, No. 8, Aug. 1989, pp.842-859.

[Huisman88] Huisman, L.M., "The Reliability of Approximate Testability Measures," *IEEE J. Design & Test of Computers,* March 1988, pp.57-67.

[IEEE90] *IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture*, IEEE Standard Office, NJ, May 1990.

[Ivanov88] Ivanov, A. and Agarwal, V.K., "An Iterative Technique for Calculating Aliasing Probability of Linear Feedback Signature Registers," *Proc. FTCS*, 1988, pp.70-75.

[Ivanov91] Ivanov, A., Starke, C.W., Agarwal, V.K., Daehn, W., Gruetzner, M., and Williams, T.W., "Iterative Algorithms for Computing Aliasing Probabilities," *IEEE Trans. CAD*, Vol. 10, No. 2, Feb. 1991, pp.260-265.

[Ivanov92] Ivanov, A. and Pilarski, S., "Performance of Signature Analysis: a survey of bounds, exact, and heuristic algorithms," *Integration, the VLSI J.*, 13, 1992, pp.17-38.

[Ivanov92b] Ivanov, A. and Zorian, Y., "Count-Based BIST Compaction Schemes and Aliasing Probability Computation," *IEEE Trans. CAD*, Vol. 11, No. 6, June 1992, pp.768-777.

[Ivanov93] Ivanov, A. and Wu, Y., "Fuzzy Multiple Signature Analysis for BIST," US Patent Pending, UBC file No. 92-063, 1993.

[Jarwala89] Jarwala, N. and Yau, C., "A New Framework for Analyzing Test Generation and Diagnosis Algorithm for Wiring Interconnects," *Proc. Int. Test Conf.*, 1989, pp. 63-70.

[Johnson89] Johnson, B.W., *Design and Analysis of Fault Tolerant Digital Systems, Addison-Wisley Publishing Company, Inc.*, 1989.

[Kameda93] Kameda, Tiko, Pilarski, S. and Ivanov, A., "Notes on Multiple Input Signature Analysis," *IEEE Trans. Computers*, Feb. 1993, pp.228-234.

[Karpovsky91] Karpovsky, M.G., Gupta, S.K., and Pradhan, D.K., "Aliasing and Diagnosis Probability in MISR and STUMPS Using a General Error Model," *Proc. Int. Test Conf.*, 1991, pp.828-839.

[Katoozi92] Katoozi, M. et al., and Nordsieck, A.W., "Built-In Testable Error Detection and Correction," IEEE J. of Solid-State Circuits, Jan. 1992, pp. 59-66.

[Keller90] Keller, B.L. and Snethen, T.J., "Built-in Self-test Support in the IBM Engineering Design System," *IBM J. Res. & Develop.,* Vol. 34, No. 2/3, March/May 1990, 406-415.

[Keller91] Keller, B., Carlson, D., and Maloney, W., "The Compiled Logic Simulator," *IEEE J. of Design & Test,* March 1991, pp.21-34.

[Konemann79] Konemann79, Konemann, B., Mucha, J. and Zwiehoff, G., "Built-In Logic Block Observation Technique," *Proc. IEEE Test Conf.*, Oct. 1979, pp.37-41.

[Kuban84] Kuban, J.R. and Bruce, W.C., "Self-Testing the Motorola MC6804P2," *IEEE J. of Design & Test,* May, 1984, pp.33-41.

[Kubian91] Kubian, K. and Fuchs, W.K., "Multiple-Fault Simulation and Coverage of Deterministic Single-Fault Test Sets," *Proc. Int. Test Conf.*, 1991, pp. 956-962.

[Lambidonis91] Lambidonis, D., Agarwal, V.K., Ivanov, A. and Xavier, D., "Computation of Exact Fault Coverage for Compact Testing Schemes," *Proc. ISCAS*, 1991.

[Lambidonis91b] Lambidonis, D., Ivanov, A., and Agawal, V.K., "Fast Signature Computation for Linear Compactors," *Proc. Int. Test Conf.*, 1991, pp. 808-817. June 1991, pp. 1873-1876.

[Larson82] Larson, H.J., *Introduction to Probability Theory and Statistical Inference,* John Wiley & Sons, Inc., 1982.

[LeBlanc84] LeBlanc, J.J., "LOCST: A Built-In Self-Test Technique," *IEEE J. Design & Test*, Nov. 1984, pp.45-52.

[Lee88] Lee, Y.H. and Krishna, C.M., "Optimal Scheduling of Signature Analysis for VLSI Testing," *Proc. Int. Test Conf.,* 1988, pp. 443-447.

[Lee91] Lee, H.K. and Ha, D.S., "An Efficient, Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation," *Proc. Int. Test Conf.*, 1991, pp. 946-955.

[Levy91] Levy, Paul S., "Designing In Power-Down Test Circuits," *IEEE J. Design & Test of Computers*, Sept. 1991, pp. 31-35.

[Li87] Li, Y.K. and Robinson, J.P., "Space Compression Methods with Output Data Modification," *IEEE Trans. on CAD*, Vol. CAD-6, No. 2, March 1987, pp.290-294.

[Lin83] Lin, S. and Costello, D.J. Jr., *Error Control Coding,* Edglewood Cliffs, NJ: Prentice-Hall, 1983.

[Mamari90] Mamari, F. and Rajski, J., "A Method of Fault Simulation Based on Stem Regions," *IEEE Trans. CAD*, Vol. 9, No. 2, Feb. 1990, pp.212-220.

[Maxwell91] Maxwell, P.C., Aitken, R.C., Johansen, V., and Chiang, I., "The Effect of Different Test Sets on Quality Level Prediction: When is 80% Better Than 90%," *Proc. Int. Test Conf.*, 1991, pp. 358-364.

[McAnney86] McAnney, W. and Savir, J., "Built-In Checking of the Correct Self-Test Signature," *Proc. Int. Test Conf.*, 1986, pp.54-58.

[McAnney87] McAnney, W.H. and Savir, J., "There is Information in Faulty Signatures", *Proc. Int. Test Conf.*, 1987, pp.630-636.

[McCluskey81] McCluskey, E.J. and Bozorgui-Nesbat, S., "Design for Autonomous Test," *IEEE Trans. Comput.*, Vol. C-30, No. 11, Nov. 1981, pp.866-874.

[McCluskey84] McCluskey, E.J., "Verification Testing — A Pseudoexhaustive Test Technique", *IEEE Trans. Comput.*, vol. C-33, No. 6, June 1984, pp.541-546.

[McCluskey85] McCluskey, E.J., "Built-In Self-Test Techniques," *IEEE J. of Design & Test*, April 1985, pp.21-28.

[McLeod92] McLeod, G.R., "BIST Techniques for ASIC Design," *Proc. Int. Test Conf.*, 1992, pp.496-505.

[Mead80] Mead, C., and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, 1980.

[Miller89] Miller, D.M. and Zhang, S., "Aliasing in Multiple Input Data Compaction," *Proc. Canadian Conf. on Elec. and Computer Eng.*, 1989, pp.347-351.

[Miller91] Miller, D.M. and Zhang, S., "A Study of the Fault Coverage of LFSR and CA Pseudo-Random Test Pattern Generators," *private communication*

[Muradali89] Muradali, F., Agarwal, V.K. and Nadeau-Dostie, B., "A New Procedure for Weighted Random Built-In Self-Test," *Proc.*, 1989.

[Nagle89] Nagle, H.T., et al., "Design for Testability and Built-In Self Test: A Review," *IEEE Trans. Industrial Electronics,* Vol. 36, No. 2, May 1989, pp.129-140.

[Park91] Park, S., Akers, S.B., "Parity Bit Calculation and Test Signal Compaction for BIST Applications," *Proc. Int. Test Conf.,* 1991, pp. 1016-1023.

[Pilarski92] Pilarski, S. and Wiebe, K., "Counter-Based Compaction: an Analysis for BIST," *J. Electronic Testing: theory and application (JETTA)*, 3, 1992, pp.33-43.

[Pilarski92b] Pilarski, S., Kameda, T., and Ivanov, A., "Sequential Faults and Aliasing," Tech. Report, CSS-LCCR TR 92-05, Simon Fraser University.

[Pomeranz92] Pomeranz, I., Reddy, S.M. and Tangirala, R., "On Achieving Zero Aliasing for Modeled Faults," *Proc. European Design Auto. Conf.,* 1992.

[Pomeranz92b] Pomeranz, I. and Kohavi, Z., "A Limited Exponential Complexity Algorithm for Increasing the Testability of Digital Circuits by Testing-Module Insertion," *IEEE Trans. CAD,* Vol. 11, No.2, Feb. 1992, pp.247-259.

[Pradhan90] Pradhan, D.K., Gupta, S.K. and Karpovsky, M.G., "Aliasing Probability for Mulitple Input Signature Analyzer," *IEEE Trans. Compt.*, Vol. C-39, No. 4, Apr. 1990, pp.586-592.

[Raina91] Raina, R. and Marinos, P.N., "Signature Analysis with Modified Linear Feedback Shift Registers (M-LFSRs)," *Proc. FTCS*, 1991, pp.88-95.

[Rajski87] Rajski, J. and Cox, H., "A Method of Test Generation and Fault Diagnosis in Very Large Combinational Circuits," *Proc. Int. Test Conf.*, 1987, pp. 932-943.

[Rajski90] Rajski, J. and Vasudevamurthy. J., "Testability Preserving Transformations in Multi-level Synthesis," *Proc. Int. Test Conf.*, Sept. 1990, pp.256-273.

[Rajski91] Rajski, J. and Tyszer, J., "On the Diagnostic Properties of Linear Feedback Shift Register," *IEEE Trans. CAD,* Vol. No. 10, Oct. 1991, pp. 1316-1322.

[Rajki91b] Rajski, J. and Tyszer, J., "Experimental Analysis of Fault Coverage in Systems with Signature Registers," *Proc. European Test Conf.*, 1991, pp.45-48.

[Rajski92] Rajski, J., Vasudevamurthy, J. and El-Maleh, A., "Recent Advances in Logic Synthesis with Testability," *Proc. IEEE Test Symp.,* 1992, pp.254-256.

[Ravinder89] Ravinder, S.S, Yeung, P., and Tucci, P.A., "Built-In Test Methodology for a Full Customer Processor Chip," *Proc. ICCD*, 1989, pp. 571-575.

[Reddy88] Reddy, S.M., Saluja, K.K., and Karpovsky, M.G., "A Data Compression Technique for Built-In Self-Test," *IEEE Trans. Comput.*, Vol. 37, No. 9, Sept. 1988.

[Roberts84] Roberts, M.W. and Lala, P.K., "An Algorithm for the Partitioning of Logic Circuits," *IEE Proc.,* Vol. 131, Pt. E., No. 4, 1984, pp.113-118.

[Robinson85] Robinson, J.P., "Segmented Testing," *IEEE Trans. Comput.*, Vol. C-34, No. 5, May 1985, pp. 467-471.

[Robinson87] Robinson, J.P. and Saxena, N.R., "A Unified View of Test Compression Methods," *IEEE Trans. Comput.,* Vol. C-36, No. 1, Jan. 1987, pp. 94-99.

[Robinson87b] Robinson, J.P. and Saxena, N.R., "Signatures and Syndromes are Almost Orthogonal," *in Development In Integrated Circuit Testing*, Academic Press Ltd., 1987, pp.147-168 .

[Robinson88] Robinson, J.P. and Saxena, N.R., "Simultaneous Signature and Syndrome Compression," *IEEE Trans. CAD.,* May 1988, pp. 584-589.

[Saluja88] Saluja, K.K., Sharma, R., and Kime, C.R., "A Concurrent Testing Technique for Digital Circuits," *IEEE Trans. CAD*, Vol. 7, No. 12, 1988, pp.1250-1259.

[Saluja92] Saluja, K.K. and See, C., "An Efficient Signature Computation Method," *IEEE J. Design & Test*, Dec. 1992, pp. 22-26.

[Savaria86] Savaria, Y., Rumin, N.C., Hayes, J.F., and Agarwal, V.K., "Soft Error Filtering: A Solution to the Reliability Problem of Future VLSI Digital Circuits," *IEEE Proc.*, May 1986, pp. 669-683.

[Savir83] Savir, J. Ditlow, G., and Bardell, P.H., "Random Pattern Testability," *Proc. FTCS-13*, June 1983, pp.80-89.

[Savir84] Savir, J. Ditlow, G.S., and Bardell, P.H., "Random Pattern Testability," *IEEE Trans. Comput.*, Vol. C-33, No. 1, Jan. 1984, pp.79-90.

[Savir84b] Savir, J. and Bardell, P.H., "On Random Pattern Test Length," *IEEE Trans. Comput.*, Vol. C-33, No. 6, June 1984, pp. 467-474.

[Saxena84] Saxena, N.R., Master's Thsis, Dept. of Elec. and Comuter Eng., University of Iowa, May 1984.

[Saxena87] Saxena, N.R., "Effectiveness Measures for Data Compression Techniques under Unequally Likely Errors," *in Development In Integrated Circuit Testing*, Academic Press Ltd., 1987, pp. 257-278.

[Saxena92] Saxena, N.R., Franco, P., and McCluskey, E.J., "Simple Bounds on Serial Signature Analysis Aliasing for Random Testing," *IEEE Trans. Computers.*, Vol. 41, No. 5, May 1992, pp.638-644.

[Scholz88] Scholz, H. et al., "ASIC Implementations of Boundary-Scan and BIST," *Int. Custom Microelectronics Conf.*, London, UK, Nov. 1988, pp.43.0-43.9

[Schulz87] Schulz, M.H. and Brglez, F., "Accelerated Transition Fault Simulation," *Proc. DAC*, 1987, pp. 237-243.

[Schulz88] Schulz, M.H., Trishler, E., and Sarfert, T.M., "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. CAD*, Jan. 1988, pp. 125-137.

[Sedmak85] Sedmak, R.M., "Built-In Self-Test: Pass or Fail ?" *IEEE J. of Design & Test*, April 1985, pp.17-19.

[Seth85] Seth, S.C., Pan, L., and Agrawal, V.D., "PREDICT — Probabilitic Estimation of Digital Circuit Testability," *Proc. FTCS-15*, 1985, pp. 220-225.

[Seth90] Seth, S.C., Agrawal, V.D., and Farhat, H., "A Statistical Theory of Digital Circuit Testability," *IEEE Trans. Comput.*, Vol. 39, No. 4, April 1990, pp.582-586.

[Serra90] Serra, M., Slater, T., Muzio, J.C., and Miller, D.M., "The Analysis of One-Dimensional Linear Celluar Automata and Their Aliasing Properties," *IEEE Trans. CAD*, Vol. 9, No. 7, July 1990, pp.767-778.

[Shperling87] Shperling, I. and McCluskey, E.J., "Circuit Segmentation for Pseudo-Exhaustive Testing Via Simulated Annealing," *Proc. Int. Test Conf.*, 1987, pp. 58-66.

[Smith80] Smith, J.E., "Measures of Effectiveness of Fault Signature Analysis," *IEEE Trans. Comput.*, Vol C-29, June 1980, pp.510-514.

[Soden89] Soden, J.M., Treece, R.K., Taylor, M.R. and Hawkins, C.F., "CMOS IC Stuck-Open Electrical Effects and Design Considerations," *Proc. Int. Test Conf.*, 1989, PP.423- .

[Takasaki87] Takasak, S., Sasaki, T., Nomizu, N., Koike, N., and Ohmori, K., "Block-Level Hardware Logic Simulation Machine," *IEEE Trans. CAD*, Vol. CAD-6, No. 1, Jan. 1987, pp.46-54.

[Tan87] Tan, S.B., Totton, K., Baker, K., Varma, P., and Porter, R., "A Fast Signature Simulation Tool for Built-In Self-Testing Circuits," *Proc. Design Auto. Conf.*, 1987, pp.17-25.

[Tang83] Tang, D.T., and Woo, L.S., "Exhaustive Test Pattern Generation with Constant Weight Vectors," *IEEE Trans. Comput.*, Vol. C-32, No. 12, 1983, pp.1145-1150.

[Udell89] Udell, J.G.Jr. and McCluskey, E.J., "Pseudo-Exhaustive Test and Segmentation: Formal Definitions and Extended Fault Coverage Results," *Proc. FTCS,* 1989, pp. 292-298.

[Udell92] Udell, J.G. Jr., "Efficient Segmentation for Pseudo-Exhaustive BIST," *Proc. IEEE Custom IC Conf.*, 1992, pp. 13.6.1-13.6.5.

[Vasudevan93] Vasudevan, B., Ross, D.E., Gala, M. and Watson, K.L., "LFSR Based Deterministic Hardware for At-Speed BIST," *Proc. IEEE VLSI Test Symp.*, 1993, pp. 201-207.

[Wagner87] Wagner, K.D., Chin, C.K., and McCluskey, E.J., "Pseudorandom Testing," *IEEE Trans. Comput.*, Vol. C-36, No. 3, Mar. 1987, 332-343.

[Waicukauski85] Waicukauski, J.A., Eichelberger, E.B., and Forlenza, D.O., "Fault Simulation for Structured VLSI," *VLSI System Design*, Dec. 1985, pp. 20-32.

[Waicukauski87] Waicukauski, J.A., Gupta, V.P. and Patel, S.T., "Diagnosis of BIST Failures by PPSFP Simulation," *Proc. Int. Test Conf.*, 1987, pp. 480-484.

[Waicukauski87b] Waicukauski, J.A., Lindbloom, E., Rosen, B.K., and Iyengar, V.S., "Transition Fault Simulation," *IEEE J. Design & Test*, April 1987, pp. 32-38.

[Wang86] Wang, L.T. and McCluskey, E.J., "A Hybrid Design of Maximum Length Sequence Generators," *Proc. Int. Test Conf.*, 1986, pp.38-47.

[Wang86b] Wang, L.T. and McCluskey, E.J., "Condensed Linear Feedback Shift Register (LFSR) Testing — A Pseudo-Exhaustive Test Technique," *IEEE Trans. Comput.*, Vol. C-35, 1986, pp.367-370.

[William83] Williams, T.W., and Parker, K.P., "Design for Testability - A survey," *Proceedings of the IEEE*, Vol. 71, No. 1, pp.98-112, January 1983.

[Williams86] Williams, T.W., Daehn, W. Gruetzner, M., and Starke, C.W., "Comparistion of Aliasing Errors for Primitive and Non-primitive Polynomials," *Proc. Int. Test Conf.*, 1986, pp. 282-288.

[Williams87] Williams, T.W., Daehn, W., Gruetzner, M., and Starke, C.W., "Aliasing Errors in Signature Analysis Registers," *IEEE J. Design & Test*, April 1987, pp.39-45.

[Williams89] Williams, T.W. and Daehn, W., "Aliasing Errors in Multiple Input Signature Analysis Registers," *Proc. European Test Conf.*, Aril 1989, pp. 346-353.

[Wu90] Wu, E. and Autkowski, P.W., "PEST — A Tool for Implementing Pseudo-Exhaustive Self Test," *Proc. IEEE Custom IC Conf.*, 1990, pp.28.4.1-28.4.3.

[Wu91] Wu, Y. and Ivanov, A., "A Multiple Signature Compaction Scheme for BIST," *Proc. Canadian Conf. VLSI,* 1991, pp. 2.2.1 - 2.2.7. also in *Microelectronics J.*, Vol.23, No. 3, 1992, pp. 205-214.

[Wu92a] Wu, Y. and Ivanov, A., "Accelerated Path Delay Fault Simulation," *Proc. IEEE Test Symp.*, Atlantic City, April 1992, pp.1-6.

[Wu92b] Wu, Y. and Ivanov, A., "A Minimal Hardware Overhead BIST Data Compaction Scheme," *Proc. Int. Conf. ASIC,* Rochester, NY, Sept. 1992, pp. 368-371.

[Wu92c] Wu, Y. and Ivanov, A., "A Fuzzy Multiple Signature Scheme for BIST," *Proc. 1st Asian Test Symp.,* Nov. 1992, Japan, pp.247-252.

[Wu93] Wu, Y. and Ivanov, A., "Minimal Hardware Multiple Signature Analysis for BIST," *Proc. IEEE Test Symp.*, Atlantic City, April 1993, pp. 17-20.

[Wu93b] Wu, Y. and Ivanov, A., "On Achieving Minimal Hardware Multiple Signature Analysis for BIST," *Proc. 2nd Asian Test Symp.*, Nov. 1993.

[WuM92] Wu, E. and Moskowitz, M.R., "A Cost-Effective Test Data Compaction Scheme," *Proc. IEEE Custom IC Conf.*, 1992, pp. 13.7.1-13.7.3.

[Wunderlich88] Wunderlich, H.J., "Multiple Distributions for Biased Random Test Patterns," *Proc. Int. Test Conf.*, 1988, pp. 236-244.

[Xavier92] Xavier, D., Aitken, R.C., Ivanov, A., and Agarwal, V.K., "Using an Asymmetric Error Model to Study Aliasing in Signature Analysis Registers," *IEEE Trans. CAD,* Vol. 11, No. 1, Jan. 1992, pp.16-25.

[Yarmolik92] Yarmolik, V.N. and Kalosha, E.P., "A Signature Evaluation Technique," *to appear on JETTA.*

[Yih91] Yih, J. and Mazumder, P.,"Circuit Behavior Modeling and Compact Testing Performance Evaluation," *IEEE J. of Solid-State Circuits*, Vol. 26, No.1, Jan. 1991, pp.62-66.

[Zasio85] Zasio, J.J., "Non Stuck Fault Testing of CMOS VLSI," *30th IEEE Computer Society Int. Conf.*, COMPCOM Spring'85, San Franciso, CA, pp.388-391.

[Zhang90] Zhang, S. and Miller, D.M., "A Comparison of LFSR and Cellular Automata BIST," *Private communication.*

[Zhang93] Zhang, C., "On Fualt Coverage and Fault Simulation for Multiple Signature Analysis BIST Schemes," *M.Sc. Thesis*, Dept. Elec. Eng., Univ. of British Columbia, 1993.

[Zhang93] Zhang, C., Wu, Y., and Ivanov, A., "On Fault Coverage of Multiple Signature Analysis," submitted to the *Canadian Conf. on Electrical and Computer Engineering*, 1993.

[Zorian86] Zorian, Y. and Agarwal, V.K., "On Improving the Effectiveness of the standard BIST approach," *Proc. Int. Conf. Custom and Semicustom ICs*, 1986.

[Zorian90] Zorian, Y. and Agarwal, V.K., "Optimizing Error Masking in BIST by Output Data Modification," *J. Electronic Testing: Theory and Applications,* No. 1, 1990, pp.59-71.

[Zorian91] Zorian, Y., "Automated Built-In Self-Test for Embedded Macrocells," *Proc. ATE and Instrumentation,* Anaheim, CA, Jan. 1991, pp.57-62.

[Zorian92] Zorian, Y. and Ivanov, A., "An Effective BIST Scheme for ROM's," *IEEE Trans. CAD,* Vol. 41, No. 5, May 1992, pp.646-653.

[Zorian93] Zorian, Y. and Ivanov, A., "Programmable Space Compaction for BIST," to appear in *Proc. FTCS-23,*, 1993.

Appendix A

Algorithm for Testing for LFSR Singularity

```
Singularity.process(a primitive LFSR of degree k)
    create an empty pool;
    for(d = 1 to 2^(k-1)) do
        i = 1;
        sigA = sigB = 1; /*sigA and sigB are two signature variables. */
        sigB = {0}_d >> [sigB]_k;
        put (sigA ⊕ sigB) in the pool;
        while(++i < 2^k) do
            sigA = {0}_1 >> [sigA]_k;
            sigB = {0}_1 >> [sigB]_k;
            if((sigA ⊕ sigB) is in the pool) break;
            else put (sigA ⊕ sigB) in the pool;
        endofwhile
        if(i ≠ 2^k)
            print "The LFSR is Singular !";
            exit;
        endofif
        clear the pool;
    endoffor
    print "The LFSR is Non-singular !";
endofSingularity
```

The computational complexity of this algorithm is $O(2^{2k-1})$. As k increases, the required CPU time for testing an LFSR increases exponentially. However, testing for LFSR singularity is a one time cost. All the commonly-used LFSR feedback polynomials, e.g., some of the primitive polynomials listed in [Bardell87], can be tested and classified according to

singularity. A list of non-singular feedback polynomials is then provided to circuit designers for reference. For example, applying the above algorithm, all the primitive polynomials of degree ≤ 20 listed in [Bardell87] have been tested. All the polynomials of degree ≤ 19 have been shown to be non-singular while the polynomial of degree 20 has been shown to be singular.

Appendix B

The probability for two sample sequences to be identical

Assume an $l+L-1$ bit sequence of the form $b_1b_2...b_{l-1}b_l\ b_{l+1}b_{l+2}...b_{l+L-1}$. Next, we show that the probability for any two subsequences of length l to be identical is 2^{-l}. Denote a first l-bit sequence $b_1b_2...b_{l-1}b_l$ by $\{B_1\}_l$, a second l-bit sequence $b_2b_3...b_{l-1}b_lb_{l+1}$ by $\{B_2\}_l$, a third l-bit sequence $b_3...b_lb_{l+1}b_{l+2}$ by $\{B_3\}_l$, etc. Obviously, $\{B_1\}_l$ will be identical to $\{B_2\}_l$, i.e., $b_1 = b_2$, $b_2 = b_3$, ..., $b_l = b_{l+1}$, only if:

$$\{B_1\}_l = 00...0 \quad and \quad b_{l+1} = 0,$$

$$or \qquad \{B_1\}_l = 11...1 \quad and \quad b_{l+1} = 1.$$

Since

$$Pr[\{B_1\}_l = 00...0] = 2^{-l},$$

$$Pr[\{B_1\}_l = 11...1] = 2^{-l},$$

$$Pr[b_{l+1} = 0] = 2^{-1},$$

$$Pr[b_{l+1} = 1] = 2^{-1},$$

$$Pr[\{B_1\}_l = \{B_2\}_l] = 2^{-l}2^{-1} + 2^{-l}2^{-1}$$

$$= 2^{-l}.$$

Consider the sequences $\{B_1\}_l$ and $\{B_3\}_l$. The two sequences will be identical, i.e., $b_1 = b_3$, $b_2 = b_4$, ..., $b_l = b_{l+2}$, only if:

$$\{B_1\}_l = 00...00 \quad and \quad b_{l+1}b_{l+2} = 00,$$

$$or \qquad \{B_1\}_l = 11...11 \quad and \quad b_{l+1}b_{l+2} = 11,$$

$$or \qquad \{B_1\}_l = 01...01 \quad and \quad b_{l+1}b_{l+2} = 01,$$

$$or \qquad \{B_1\}_l = 10...10 \quad and \quad b_{l+1}b_{l+2} = 10.$$

The probability for $\{B_1\}_l$ to be any of the above specific sequence is 2^{-l}, and the probability for $b_{l+1}b_{l+2}$ to be any of the above value is 2^{-2}. Thus,

$$\begin{aligned} Pr[\{B_1\}_l = \{B_3\}_l] &= 2^2 2^{-l} 2^{-2} \\ &= 2^{-l} \end{aligned}$$

In general, consider the two sequences $\{B_1\}_l$ and $\{B_i\}_l$,

$$\begin{aligned} Pr[\{B_1\}_l = \{B_i\}_l] &= 2^i 2^{-l} 2^{-i} \\ &= 2^{-l} \qquad\qquad l+1 \leq i \leq l+L-1. \end{aligned}$$

Similarly, we can show that $\Pr[\{B_i\}_l = \{B_j\}_l] = 2^{-l}$, where $i \neq j$, $i \leq (L-1)$ and $j \leq (L-1)$. Therefore, the probability for any two sample sequences to be identical is 2^{-l}.