DETERMINISTIC BUILT-IN SELF TEST
FOR
DIGITAL CIRCUITS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Ahmad A. Al-Yamani

April 2004

I certify that I have read this dissertation and that, in my opinion, it is fully adequate, in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
Edward J. McCluskey (Principal Advisor)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate, in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
John T. Gill III

I certify that I have read this dissertation and that, in my opinion, it is fully adequate, in scope and quality as a dissertation for the degree of Doctor of Philosophy.

_____
Giovanni De Micheli

Approved for the University Committee on Graduate Studies.

Abstract

In built-in self-test (BIST), on-chip circuitry is added to generate test vectors or analyze output responses or both. BIST is usually performed using pseudorandom pattern generators (PRPGs). Among the advantages of pseudorandom BIST are: (1) The low cost compared to testing from automatic test equipment (ATE). (2) The speed of the test, which is much faster than when it is applied from ATE. (3) The applicability of the test while the circuit is in the field, and (4) The potential for high quality of test.

The main disadvantages of pseudorandom BIST are: (1) Testing a circuit with constraints on some signals may cause an illegal combination of values on those signals when pseudorandom patterns are used. (2) Plain pseudorandom patterns may not achieve thorough testing with a reasonable test length. Enhancement techniques are used to improve the thoroughness of pseudorandom testing. In this dissertation, a set of novel techniques are presented to address and solve the problems of pseudorandom BIST.

Many digital circuits have constraints on what combination of values can occur on a set of signal lines. Using pseudorandom BIST for such circuits may cause the circuit to be damaged or the test results to be corrupted. This dissertation presents techniques for detecting the illegal combinations of signal values and preventing them from occurring or from corrupting the test results.

BIST reseeding is used to improve fault coverage by reinitializing the PRPG to generate deterministic test patterns that target specific faults. Most of the previous work done on reseeding is based on storing the seeds in the ATE. This dissertation presents a technique for built-in reseeding. The technique requires no storage for the seeds because the seeds are encoded in circuitry on the product chip.

In reseeding, the test storage or hardware overhead are proportional to the number of seeds. This dissertation presents an algorithm for ordering the seeds in order to reduce the number of seeds needed to produce a set of deterministic test patterns. When compared to arbitrary ordering, the technique reduces seed storage by up to 80%. The dissertation also presents a technique for encoding a given seed by the number of clock cycles that the generator needs to run to reach it. This encoding requires substantially fewer bits than the bits of the seed itself. When compared with conventional reseeding, the technique reduces seed storage by up to 85%.

Acknowledgments

I would like to express my profound gratitude and appreciation to my advisor, Professor Edward J. McCluskey, for his constant guidance, support, and encouragement throughout my years at Stanford. He models many of the high quality characteristics that I aspire to emulate during my professional and personal life. Working with him has been and will continue to be a source of honor and pride for me.

I also thank Professor John Gill for being my associate advisor and being in my dissertation reading committee. I would like to also thank Professor Giovanni De Micheli for being in the examining committee and the reading committee of my dissertation. I also thank Professor Bernard Widrow for chairing the examining committee. It will always be a source of honor for me to have had the names of these world-class professors on my dissertation.

I want to thank my parents who have been a main source of success in my life. Their prayer, love and support carried me through the most difficult moments in my life. There are no words that I can use or things that I can do to thank them. My sisters and brothers have been a source of continuous encouragement for me. They always believed in me and gave me the confidence that I can go farther. I also thank the one who did the difficult part of my studies at Stanford. I thank my wife “Alaa Olwi” who always stood by my side and never gave up on me. I also thank my father-in-law and my mother-in-law who continuously supported me throughout my PhD program. I also thank “Sheikh Ali Olwi” whose prayers for me and concern about me did not stop even during the last moments in his life.

I thank Dr. Sadiq Sait for his support and encouragement throughout my graduate studies. I also acknowledge the support provided by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517. I also thank all of my friends at Stanford and at KFUPM. I especially thank my colleagues at CRC.

Dedication

I dedicate this dissertation to

my beloved parents

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

Manufacturing processes for *very large scale integration* (VLSI) circuits are far from perfect. So they do, due to different causes, introduce defects in VLSI circuits. It is very costly to rely on the customer in identifying if the shipped part is functioning properly or not. So, it is mandatory to test integrated circuits (ICs) before shipping them.

VLSI circuits are tested by applying test patterns to the circuit under test (CUT) and comparing the response of the circuit to the good circuit response, which is obtained by simulation.

Automatic test equipment (ATE) is used for testing VLSI circuits. These are special purpose computers that are designed to apply test patterns to the CUTs and compare the response of the CUTs with the correct response. Among the important features of ATE for digital test are frequency, number of pins and memory size.

Test patterns are generated using software programs called automatic test pattern generation (ATPG) tools. Since the behavior of actual defects in ICs is not yet well understood, ATPG tools target faults that imperfectly model the defects. The most popular fault model is the single-stuck fault (SSF) model. In this model, a single signal line at a time is assumed to have a fixed 0 or 1 value independent of the other signal values. Multiple-stuck fault model assumes that multiple signal lines may be stuck at some values. Bridging fault model assumes that two distinct nodes are connected to one another. Transition fault model assumes that a node does not transition from one signal value to another within the allowed time. SSF and transition fault models are the most widely used models in current ATPG tools. For both models, the ATPG tools try to generate test patterns that excite the faults and propagate them to observable outputs.

Some design for testability (DFT) techniques are used to improve the controllability (the ability to set the node at a certain value) and the observability (the ability to propagate the value of a node to an observable output) of internal nodes in digital circuits. Among the widely used DFT techniques are scan-path techniques [McCluskey 86]. In scan-path techniques, the circuit is designed to have two modes of operation, namely, a normal functional mode and a test mode. In the test mode, the

bistables (the memory elements in the circuit) are interconnected into a shift register. In test mode, it is possible to shift an arbitrary test pattern in the bistables. By going back to the functional mode for one clock pulse, the response of the circuit to the test pattern is latched into the bistables. The circuit can then be placed back in test mode to concurrently shift the response out of the chain and shift a new pattern into the chain.

The addition of on-chip circuitry to provide test vectors or to analyze output responses is called built-in self-test (BIST) [McCluskey 85] [Bardell 87]. The pattern generation in BIST is usually done using linear feedback shift registers (LFSRs) or cellular automata (CA). Both are pseudorandom pattern generators (PRPGs). By appropriately choosing the polynomial for the PRPRG, it can be assured that test patterns are not repeated within the test session.

Among the advantages cited in favor of BIST are: (1) BIST is usually lower in cost compared to external testing using ATE. (2) With BIST, it is possible to apply the test at high speed, which helps in detecting timing defects and in shortening the test time. (3) It is possible to test the circuit in the field if the circuit is built-in self-tested. (4) Pseudorandom BIST has a potential in detecting unmodeled defects.

Among the disadvantages of BIST are: (1) Pseudorandom patterns may cause illegal combinations on some signals that have constraints on the set of logic values they can have. (2) Pseudorandom patterns normally do not provide thorough test sets. (3) In order to achieve a reasonable fault coverage, BIST may require prohibitively long test lengths. (4) BIST costs additional area overhead on the circuit.

1.2 Contributions

Correct operation of digital circuits is not guaranteed if illegal combinations of logic values appear on some signal lines. For example, many digital designs contain logic that is controlled by one-out-of-n (one-hot) signals. *One-hot* signals in digital circuits are a set of signal lines of which no more than one signal should be active at a time. Examples include an n-to-1 selector implemented with n transmission gates, enabled by different signals, and a bus controlled by tri-state buffers.

Illegal values during pattern application can be caused by pseudorandom testing. When the circuit is tested using automatic test pattern generation (ATPG), the one-hot

condition can be provided as a constraint to the ATPG tool such that none of the generated patterns results in multiple-hot or zero-hot values in one-hot signals during test pattern application. However, when pseudorandom patterns are applied (externally or internally) or when the ATPG tool does not check for illegal values, some of the patterns generated may cause illegal states or illegal stimuli on the one-hot signals.

In this dissertation, new techniques are presented for detecting illegal states in digital circuits and masking their effects. Unlike previous techniques, the new techniques have no impact on the fault coverage achieved with the legal patterns of a given test set.

The new techniques do not impose restrictions on the original design. They satisfy the one-hot constraints during test pattern application so they complement the techniques that satisfy the constraints during scan-in and out. They also do not affect the fault coverage for the legal patterns, that do not cause an illegal state, in a given test set. The techniques can be directly applied for circuits with arbitrary constraints on logic values that can appear on a set of signal lines.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudorandom testing [Eichelberger 83]. The r.p.r. faults are faults with low detectability (few patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques are: (1) Modifying the circuit under test (CUT) by test point insertion [Eichelberger 83] [Touba 96a] or by redesigning the CUT, (2) *Weighted pseudorandom patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89] [Wunderlich 90] and (3) *Mixed-mode testing* where the circuit is tested in two phases. In the first phase, pseudorandom patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [Koenemann 91] [Hellebrand 95] [Touba 00]. This dissertation presents some mixed-mode techniques based on inserting deterministic patterns between the pseudorandom patterns.

Modifying the CUT is often not desirable because of performance issues or intellectual property reasons. Weighted pseudorandom sequences require multiple weight sets that are typically stored on chip. Mixed-mode testing is done in several ways; one way is to apply the deterministic patterns from an external tester or store them in an on-chip ROM. Additional circuitry is required to apply the patterns in the ROM to the circuit

under test. Instead of storing patterns, seeds can be stored on the tester or in the on-chip ROM. These seeds are transferred into the PRPG and then expanded into the scan chains. This technique does not eliminate the need for the circuitry that transfers the seeds from the ROM to the PRPG.

Another technique for mixed-mode testing uses mapping logic [Touba 95]. The strategy is to identify patterns in the pseudorandom sequence that do not detect any new faults and map them by hardware into deterministic patterns.

Reseeding refers to loading the PRPG with a seed that expands into a precomputed test pattern. In this dissertation, I present a built-in reseeding technique that combines mapping logic and reseeding [Alyamani 03a]. The technique uses a simple circuit to identify the states at which the LFSR is to be reseeded. It uses minimal additional hardware to choose the new seed. The technique utilizes the LFSR bistables for storing the seeds.

The built-in reseeding contributions in this dissertation include: (1) A reseeding technique that eliminates completely the need for external pattern storage or an on-chip ROM. It is based on encoding the seeds in hardware and using special hardware for the LFSR. (2) A hardware implementation for the given technique. (3) A reseeding algorithm that allows the user to trade off test length for hardware overhead.

This dissertation also presents a seed ordering algorithm that minimizes the number of seed loads. The algorithm is based on exploiting the algebraic properties of the LFSR [Alyamani 03b]. The previous work in [Koenemann 91], [Hellebrand 95], [Rajski 98], [Krishna 01], and many others generate one seed per pattern. The technique presented in this dissertation increases the number of patterns generated from one seed significantly, and hence reduces the seed storage or hardware required for encoding the seeds. Previous algorithms for embedding multiple patterns into a single-seed sequence [Lempel 95] [Fagot 99] have much higher computational complexity and are impractical for reasonable size circuits.

In this dissertation, I also present a seed encoding technique that encodes the seeds in a much smaller vector that corresponds to the number of cycles the PRPG runs to reach the intended seed [Alyamani 03c]. I also present an architecture that implements the encoding technique.

The built-in reseeding, seed ordering and seed encoding techniques are all applicable for single-stuck faults as well as transition faults.

1.3 Outline

This dissertation summarizes my work in built-in self test. Detailed description of each topic can be found in the appendices, which include published papers and technical reports at Stanford Center for Reliable Computing (CRC).

This dissertation is organized as follows. Chapter 2 presents techniques for testing digital circuits with constraints. The techniques provide solutions for detecting illegal states and for masking their effects. The chapter compares the new techniques with previous techniques for solving such problems. Details can be found in Appendix A.

Chapter 3 discusses built-in reseeding and compares it to previous work in the field. It also presents results of simulation experiments in built-in reseeding. Detailed discussion of built-in reseeding is provided in Appendix B.

In Chapter 4, the technique for seed ordering is presented. Previous work is discussed and the advantages of the new technique are explained. Detailed discussion and simulation setup and results can be found in Appendix C.

In Chapter 5, the seed encoding technique is presented together with the architecture modifications required to implement it. The advantages of the technique compared to the existing techniques are discussed. Detailed discussion and simulation results can be found in Appendix D.

Chapter 6 presents the seed calculation scheme for LFSRs and cellular automata. It also shows the procedure for matching a given state of the PRPG with a given pattern. Chapter 7 concludes the dissertation.

Chapter 2
Testing Digital Circuits with Constraints

This chapter presents new techniques for detecting illegal combinations of values in digital circuits and masking their effects.

Correct operation of digital circuits is not guaranteed if illegal combinations of logic values appear on some signal lines. For a tri-state bus, an illegal state occurs when more than one driver is enabled to drive the bus at the same time; this state is known as a *contention state* or a *multiple-hot state*. In case of contention, if the two drivers are writing two different values, the value of the bus is nondeterministic. This non-determinism may propagate to the output. A more severe effect is that the circuit may be damaged because pull-up and pull-down transistors are both activated. A similar non-determinism can occur if none of the tri-state buffers is enabled to drive the bus, in which case the bus will be floating (*zero-hot*).

Illegal states can appear during scan in and out because the patterns are shifted serially through the bistables. Several solutions are available in the literature for this problem. For example, bistables that may cause illegal states can be controlled with control points [Hetherington 99], removed from the scan chain, or bypassed [Raina 00]. The techniques presented in this dissertation are intended for illegal states that occur during test pattern application rather than during scan in and out.

This dissertation presents new techniques for detecting illegal states in digital circuits and masking their effects. Unlike previous techniques, these techniques have no impact on the fault coverage achieved with the legal patterns of a given test set. Although the new techniques are discussed in the context of one-hot signals, they are directly applicable with arbitrary constraints on logic values that can appear on a set of signal lines. In Sec. 2.1, an overview of the previous work is given. The new techniques are presented in Sec. 2.2 and simulation results are discussed in Sec. 2.3. Section 2.4 concludes the chapter.

2.1 Previous Work

If the one-hot signals are generated directly from the bistables, those bistables can be designed –by adding additional logic to them– to hold only one-hot values. Such

bistables are called one-hot bistables. An avoidance strategy is to impose constraints or design rules so that pass-transistor selectors are not used to implement multiplexers [Abadir 99]. Another approach is to gate the output of the one-hot signals during scan with the scan enable (SE) signal, resulting in a particular one-hot value enforced on the one-hot signals irrespective of the contents of the bistables. Only one of the signals should be OR-ed with the SE signal while all the other signals should be AND-ed with the complement of the SE signal as in Figure 2.1 [Synopsys 97]. When SE is 1 during scanning, a particular one-hot value is enforced. This technique ensures safety (one-hot property) during scan-in and scan-out operations, but multiple-hot or zero-hot values may appear on the one-hot signals if pseudorandom patterns are used. A similar scheme was used in [Levitt 95]. Figure 2.1 shows how this scheme is implemented to insure a one-hot value on E_1, E_2, E_3 and E_4 during scan-in and scan-out. A generalization of this approach is to enforce a particular one-hot value on the one-hot signals throughout the test mode of operation by using a special signal. Although this solution avoids illegal states during scan-in and scan-out operations and also when pseudorandom patterns are used to test the circuit, the fault coverage can fall drastically because the logic may not be sufficiently tested since the enforced one-hot value does not change during testing [Hetherington 99].

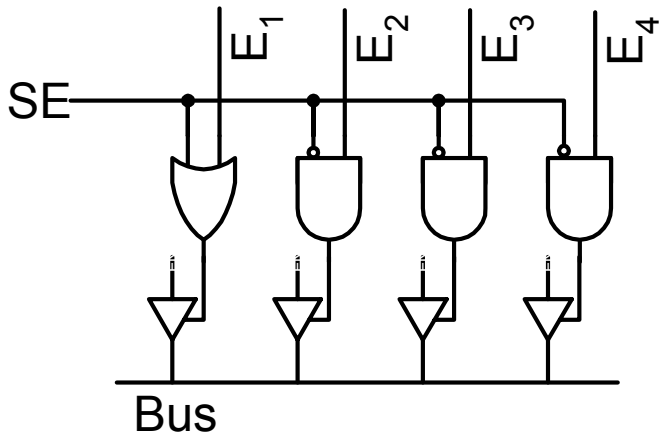


Figure 2.1 Insuring one-hot value during scan-in and out.

Another technique is to use a priority encoder for the one-hot signals. The priority encoder takes *n* arbitrary inputs and produces *n* one-hot outputs as shown in Figure 2.2 [Fleming 92] [Mitra 97]. This technique modifies the original design and adds delay overhead equivalent to several levels of logic.

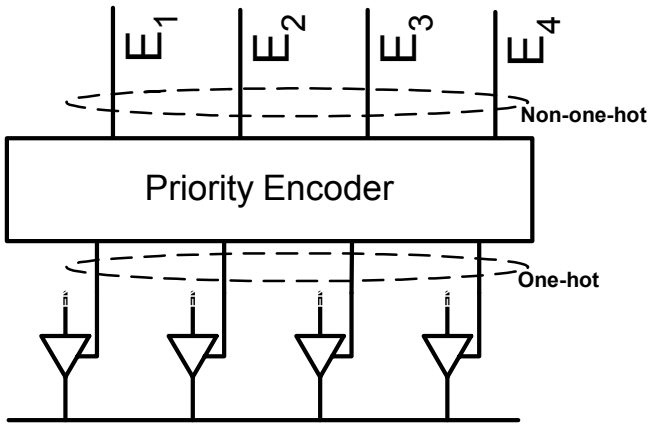


Figure 2.2 A priority encoder technique for one-hot signals.

2.2 Illegal State Detection (ISD)

The purpose of an illegal state detection (ISD) circuit is to detect whether an input pattern applied to the circuit under test (CUT) causes illegal values on a set of signals in the circuit. In this section, new techniques are presented for designing and implementing the ISD circuit. Two techniques for fixing the illegal state and taking the system back to a legal state are also discussed. The first technique is based on static fixing which requires extra hardware and adds one level of logic. The advantage of this technique is its simplicity and wide applicability. The second technique is based on skipping the patterns that cause the illegal state. This is done with no additional hardware in the circuit and with no additional delay. However, this technique requires that the test set is known in advance and is not changed later.

2.2.1 Illegal State Detection by Backtracing

The ISD function is a Boolean function that is used to detect illegal combinations of logic values. It should be expressed in terms of the primary inputs and the bistable outputs. The ISD function can be found by extracting the functions of the one-hot signals in terms of the primary inputs and the bistable outputs.

After analyzing the one-hot signals and expressing them in terms of the primary inputs and the bistables, the illegal state detection circuit is implemented so that it produces logic value 1 if the values on the current pattern causes an illegal value and it produces a 0 otherwise.

For the purpose of illustration, let us consider a circuit with full-scan. Suppose that there are 4 one-hot signal lines E_1, E_2, E_3, and E_4. As shown in Figure 2.3, $E_1 \ldots E_4$ are connected to the enable inputs of tri-state gates whose outputs are connected to a common bus. From the given combinational logic, we can find the Boolean expressions for the logic functions of $E_1 \ldots E_4$. Let the Boolean functions corresponding to E_1, E_2, E_3, and E_4 be F_1, F_2, F_3, and F_4, respectively. We form the Boolean function

$$ISD = \left(F_1 F_2' F_3' F_4' + F_1' F_2 F_3' F_4' + F_1' F_2' F_3 F_4' + F_1' F_2' F_3' F_4\right)'$$

The ISD function produces a 0 when an input combination guarantees one-hot values on the signal lines $E_1 \ldots E_4$; it produces a 1 otherwise. The ISD function can be implemented by synthesizing the ISD circuit expressed in terms of the bistables' outputs and the primary inputs. The size of the ISD circuit depends on the depth of the circuitry between the tri-state drivers and the scan bistables. We find the logic functions of the one-hot signals in terms of the primary inputs and the bistables' outputs by extracting the logic cones of such signals. The ISD circuit is then synthesized to control the fixing and the skipping logic as will be shown in Sec. 2.2.3 and Sec. 2.2.4, respectively.

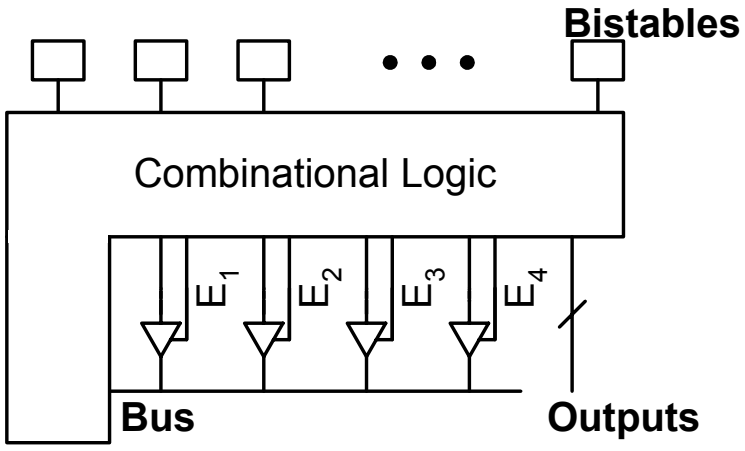


Figure 2.3 Example of tri-state busses in logic circuits.

2.2.2 Illegal State Detection Using BIST Pattern Counter

In a BIST environment, if analyzing the one-hot signals in terms of the primary inputs and bistables is not possible (e.g., for intellectual property reasons), and if the test set is known in advance, the ISD circuit may be designed to be the logical sum of the patterns that cause contention or floating values. Such patterns are found by simulation.

One way to reduce the area of the ISD circuit is to make it depend on the value of the "pattern counter" of the BIST controller. The counter value of those patterns that can cause contention can be used as the minterms for the ISD. If a set of M test patterns is applied to the circuit, the pattern counter will be of size $\lceil \log_2(M) \rceil$. This can lead to massive reduction in the complexity of the ISD circuit. From here on, the ISD circuit based on tracing will be called ISD-Trace and the ISD circuit based on the BIST pattern counter will be called ISD-Counter.

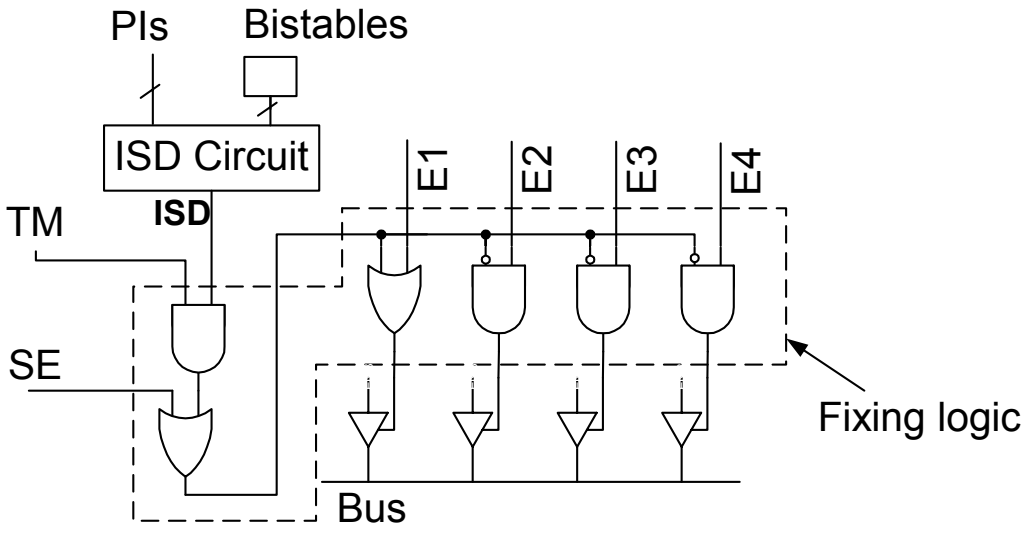


Figure 2.4 ISD-circuit and its fixing logic control tri-state enables.

2.2.3 Fixing the Illegal State

The implementation shown in Figure 2.4 is one of many possible ways to force a legal state on the one-hot signal lines in test mode. We will call the added circuitry for forcing the legal state *fixing logic*. Scan enable (*SE*) is 1 when patterns are scanned in or out of the scan chains. The test mode (*TM*) signal is 1 in the test mode, i.e. during scan in and out and also during the capture cycle. The TM signal is used to enable the test access point (TAP) controller clocking and the test points if any. The TM signal is needed for the fixing logic in order to guarantee the one-hot value during the capture cycle where SE is 0. Some circuits have a special signal to disable the capture cycle. Such a signal is used with multiple clock domains to have the flip-flops of one domain act as transmitters to flip-flops in another domain. The transmitter flip-flops need to have the capturing disabled. If the circuit has a capture-disable signal, that signal can be used to disable

capturing the combinational logic outputs of the illegal patterns without using the TM signal.

If the circuit has a BIST structure, the TM signal is provided by the BIST controller according to some BIST architectures [Dostie 00]. During normal operation, the TM and the SE inputs are both 0 and static; hence, the delay overhead introduced is very small because the output of the ISD circuit is held at 0. The above technique solves the illegal state problem for a set of signals during scan in and out and during the capture cycle because the static fixing logic is active whenever the scan enable is active or the ISD circuit is 1 in capture mode. This technique requires much less area and causes less performance overhead than the priority encoder.

In a BIST environment, the ISD circuit can be part of the BIST controller. Figure 2.5 shows an overview of the logic BIST controller in a digital circuit. It shows where the ISD circuit and the fixing logic fit in the system level view of a BIST environment.

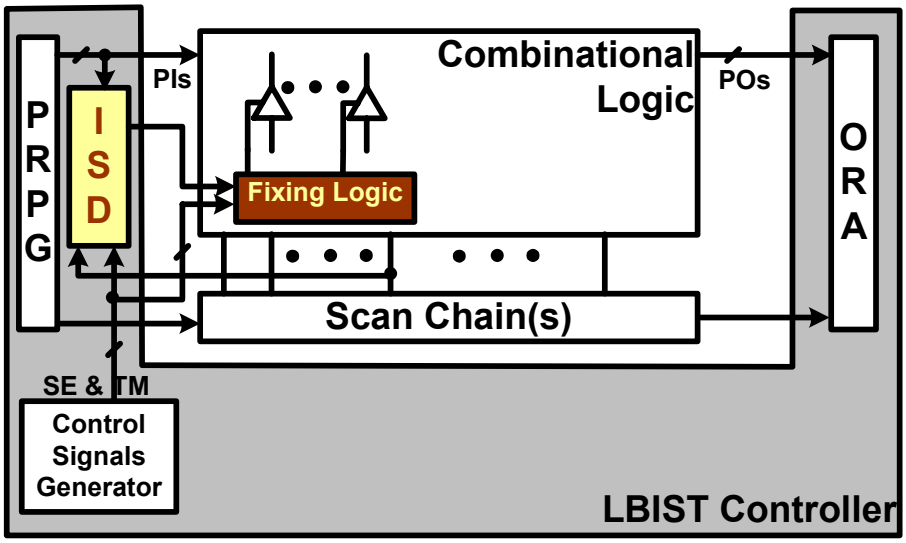


Figure 2.5 ISD circuit and fixing logic in a BIST environment.

The ISD circuit can take its inputs from the bistables in the circuit scan chain. If adding fan-outs to the scan chain bistables is not desirable, the ISD circuit can have extra bistables to store a copy of the contents of the bistables that determine the value of the ISD function. In a BIST environment, the "bit counter" is used to count the bits shifted into the scan chain. The value of the bit counter can be used to identify the bits that will go into the bistables that control the one-hot signals. These bits should be stored in the

extra bistables to determine the value of ISD. So, the ISD circuit bistables can take their inputs from the PRPG and the bit counter through simple control logic.

2.2.4 Skipping the Illegal State

A small logic circuit can be added to the BIST controller such that the patterns that cause the illegal states are not applied to the circuit. In this case, there is no need for any intrusion and no delay overhead, not even the negligible overhead, is caused by the ISD circuit.

Normally, the test patterns are scanned into the bistables with value 1 on the scan enable (SE) signal. Once the pattern is shifted in, SE is turned off for one clock cycle such that the scan chain stores the results of the combinational circuitry. Then SE is turned on again for these results to be shifted out. The logic added to the BIST controller should keep the SE signal 1 during the test application cycle in case the pattern will cause an illegal state. This way the pattern is not applied to the circuit and no corrupted output is read out.

Consider a BIST controller that uses the bit counter to count the bits shifted into the scan chain. Assume that for every scan-in and scan-out sequence, the bit counter is loaded with the count of bistables in the scan chain. It gets decremented every cycle. SE is kept 1 until the bit counter reaches zero. To avoid applying the patterns that cause an illegal state, we need to set SE to one if the value of ISD is 1 and the bit counter is 0. Figure 2.6 shows an example for the logic needed for this purpose. This logic can be part of the BIST controller. Also, in case of BIST, if multiple scan chains are used with separate SE signals, then we only need to disable capturing for the chain(s) that cause the illegal state. This way the fault coverage can be further improved.

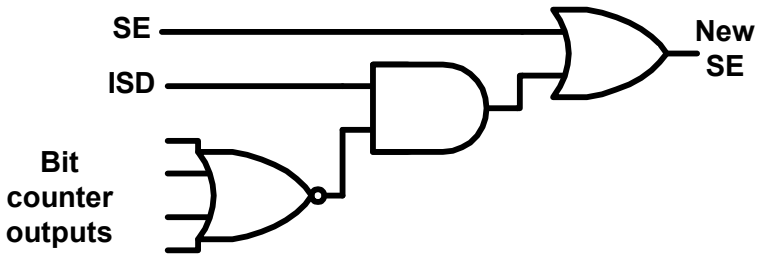


Figure 2.6 Circuitry for skipping the illegal state.

2.3 Simulation Results

We performed our experiments on I992, which is an industrial pipeline ASIC design from ITC99 benchmark suite [ITC 99]. The design has approximately 20,000 gates and four clock domains. There are no internal memories. I992 is the only circuit with tri-state drivers in the ITC benchmark suite. The other benchmark suites (e.g. ISCAS and MCNC) do not have any circuits with tri-state drivers.

I992 is an industrial circuit that has many one-hot signals. This benchmark has 6 internal busses controlled by tri-state drivers.

We traced back the tri-state enables to the bistables and primary inputs. Then we synthesized the ISD circuits for all the busses. The total area overhead of the 6 ISD circuits of I992 is only 0.892% of the total area of the circuit. This overhead is for all ISD circuits that will completely eliminate the contention problem with a full BIST solution and without any signal from the tester. Details about the overhead for every individual ISD circuit are shown in Table 2.1.

Table 2.1 I992 ISD Circuits' Area Overhead.

Busses	Bus width	Tri-state drivers	ISD circuit area	%Area overhead
Bus 1	17	2	10	0.034
Bus 2	17	2	10	0.034
Bus 3	6	3	40	0.134
Bus 4	6	5	82	0.275
Bus 5	6	7	93	0.312
Bus 6	6	3	31	0.104

If the circuit has some BIST structures but a tester is used to improve the BIST test coverage, the same skipping technique can be applied with a single bit per pattern overhead. The additional bit is used to indicate if the pseudorandom pattern applied is illegal or not. This bit can be applied from the tester and can be used to control the fixing or the skipping logic instead of the ISD circuit output.

2.4 Conclusions

Resolving the illegal states in digital circuits has existed as an obstacle for pseudorandom testing of such circuits for a long time. It has been mostly dealt with using static decoding, which sacrifices fault coverage and adds an additional level of logic that may not be needed.

The ISD technique is more generally applicable than previous techniques and it does not compromise the fault coverage. The ISD technique can be implemented without changing the given design and is hence, non-intrusive.

The ISD technique is a very low area and delay overhead technique for fixing the illegal states that can occur in pseudorandom testing. The ISD technique can be used not only during IC production test, but also during board-level or system-level tests when arbitrary test sequences are applied. It guarantees correct operation under any patterns.

The techniques presented in this paper are applicable to any circuit with constraints on the values a set of nodes can take. Furthermore, they can be combined with any technique for improving fault coverage or reducing test length in cases of pseudorandom testing.

Chapter 3
Built-In Reseeding

This chapter presents a technique for built-in reseeding and explains its architecture and reseeding algorithm. It also discusses the implementation of the technique for single-stuck faults and transition faults.

In BIST, deterministic patterns are often encoded into smaller vectors (aka seeds) that are loaded into the PRPG and then expanded into the desired patterns in the scan chains. A *seed* is an initial state for the PRPG. When the PRPG is placed in this initial state it expands into a precomputed test pattern in the scan chains after *m* cycles, where *m* is the length of the longest scan chain. *Reseeding* refers to reinitializing the PRPG with a new seed (state). It is used to improve the fault coverage with pseudorandom testing. Take as an example the LFSR used as a PRPG in Figure 3.1 for a single scan chain of 10 flip-flops. By initializing the PRPG flip-flops at the state (0111) and running the clock for 10 clock cycles the pattern (0011010111) will end up in the scan chain.

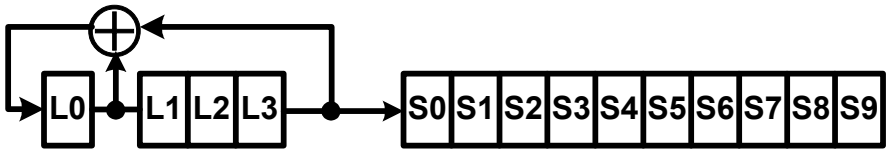


Figure 3.1 An LFSR connected to a scan chain of 10 flip-flops.

Deterministic test patterns are encoded into seeds by solving a linear system of equations, which is an algebraic representation of the linear expansion of the PRPG into the scan chains' flip-flops. There are some linear dependencies between some flip-flops of the scan chain. For example, in Figure 3.1, S5 will always have the value S9⊕S6. Due to these dependencies, solving the linear system of equations may not always be possible.

In built-in reseeding, reseeding circuitry is designed and used to load the PRPG with a new seed when needed. This is done to avoid loading the seeds from an external tester. The built-in reseeding circuit takes its inputs from the PRPG and produces outputs that change the state of the PRPG as needed.

In Sec. 3.1, an overview of the previous work is given. The built-in reseeding hardware implementation is presented in Sec. 3.2. The reseeding algorithm is explained

in Sec. 3.3 and simulation results are presented in Sec. 3.4. Section 3.5 concludes the chapter.

3.1 Previous Work

The related work discussed in this section is classified according to its relevance to this chapter.

3.1.1 Seed Calculation

Konemann presented a technique for coding test patterns into LFSRs of size S_{max}+20, where S_{max} is the maximum number of specified bits in the ATPG patterns. In current VLSI designs, S_{max} is less than 10% of the ATPG patterns. By using S_{max}+20 as the size of the LFSR, the probability that a test pattern with $S \leq S_{max}$ specified bits cannot be coded into a seed drops to 1 in a million as shown in [Koenemann 91].

Many of the bits in ATPG patterns are don't cares for the test procedure. This fact makes Koenemann's technique very useful in reducing the storage needed per pattern to S_{max}+20. S_{max} depends on the circuit under test and on the undetected faults that are targeted by the deterministic patterns. In [Koenemann 91], using the S_{max} based reseeding technique reduced the storage requirement of the test patterns from 38M bytes to 0.7M bytes.

[Zacharia 95] and [Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds together with a multiple-polynomial LFSR (MP-LFSR). The authors presented a technique that reuses part of the scan chain flip-flops in expanding the seeds.

3.1.2 Seed Storage

In [Huang 97], programmable LFSRs (PLFSRs) are used to implement multiple polynomials for the PRPG. In [Kagaris 99], a synthesis technique for counter-based test set embedding was presented. [Chakrabarty 00] presented an approach for BIST pattern generation using twisted-ring counters. The seeds are stored in a ROM. Hellebrand presented a reseeding technique based on folding counters [Hellebrand 00]. A new form of reseeding was described for high encoding efficiency in [Krishna 01].

All of the above schemes assume that seeds are either applied from an external tester or stored in an on-chip ROM. This chapter presents a technique for avoiding this requirement.

Storing seeds, instead of patterns, in a tester reduces the storage and bandwidth requirements. However, it still means that the chip has to be put on the tester. Also, although the ROM eliminates the need for the tester, there must be some circuitry to choose when to load seeds from the ROM and other circuitry to actually load them onto the PRPG.

The technique presented in this chapter embeds the seeds on the chip without requiring a ROM. Other than the circuit needed to detect when to reseed, minimal hardware is needed to load the desired seeds. The technique presented in this chapter is orthogonal to all of the above techniques and is applicable to LFSRs as well as cellular automata.

3.1.3 Hardware-Based Reseeding

[Savir 90] presented a reseeding scheme that requires duplicating the LFSR flips-flops with shadow flip-flops. The *shadow flip-flops* contain the next seed. These shadow flip-flops contents are periodically XORed with the original LFSR flip-flops contents to generate a new seed. Using this shadowing technique, the new seed is expected to be far in the sequence from the current contents of the LFSR. Kim presented a method for generating non-successive pseudorandom test patterns by cascading the LFSR with the scan chain and including a feedback from the scan-out signal into the LFSR [Kim 96]. In [Crouch 95], a self re-seeding LFSR was presented. Again the LFSR is loaded with arbitrary seeds.

The above schemes have the advantage of diversity of the sequences from which the patterns are drawn. They also have the advantage of not requiring seed storage. However, the seeds are arbitrary so they do not target specific faults. Our technique is based on deterministic seeds so 100% fault coverage can be achieved.

3.1.4 Mapping Logic

Touba and McCluskey came up with an innovative approach for applying deterministic patterns through mapping logic [Touba 95]. In their technique, pseudorandom patterns that do not detect r.p.r. faults are mapped to ATPG generated patterns through combinational logic.

Built-in reseeding is a generalization of mapping logic based on running the PRPG in autonomous mode after loading each seed to detect more faults without having to perform more mappings. In built-in reseeding, we need only the logic that detects the patterns that need to be mapped. Inserting the new values in the PRPG is done utilizing the current contents of the flip-flops of the PRPG.

3.1.5 BIST for Transition Faults

In [Pradhan 99], a new LFSR based on Galois fields (GLFSR) was presented. The experiments show that using GLFSRs, the test length is reduced for SSF and transition fault coverages of 90% and 95%. The results are for combinational circuits.

We do not rely only on pseudorandom patterns for transition faults. We perform mixed mode testing by generating both pseudorandom as well as deterministic patterns using the built-in reseeding circuitry. Also, we apply our technique on sequential circuits rather than combinational circuits.

3.2 Reseeding Circuitry Implementation

The operation of the reseeding circuit in built-in reseeding is as follows: the PRPG starts running in autonomous mode for some time according to the algorithm described in Sec. 3.3. Once it is time for reseeding, a seed is loaded into the PRPG, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the PRPG in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the PRPG.

Figure 3.2 shows the structure of an LFSR and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops as shown in the figure. By activating the select line of the i^{th} mux, the $(i+1)^{st}$ flip-flop takes Q' instead of Q from the i^{th} flip-flop.

As seen in the figure, the only modification to the LFSR compared to a standard LFSR are the muxes. The LFSR flip-flops are replaced by muxed flip-flops just as the scan chain flip-flops.

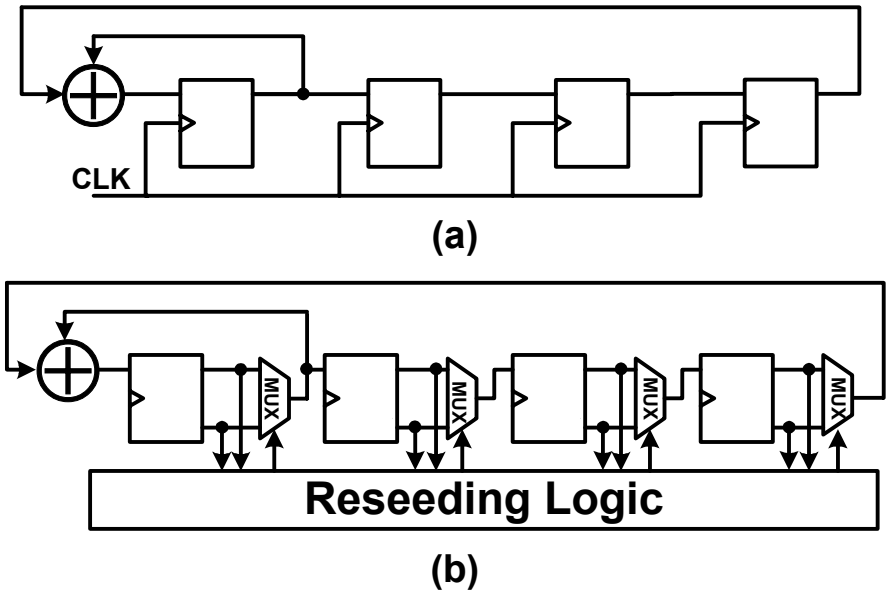


Figure 3.2 Reseeding circuit connection to the LFSR:
(a) A standard LFSR (b) LFSR with reseeding cricuit.

In the case of cellular automata, the same muxes structure can be used. The muxes should be placed right at the outputs of the flip flops before any XOR gates that are fed by the PRPG flip-flops. This way both polarities are available at the inputs of the muxes. Since XORs are linear gates, their outputs will be complemented by complementing any of the inputs, which satisfies the requirement for the above architecture to work. The connection of the reseeding logic to CA is shown in Figure 3.3.

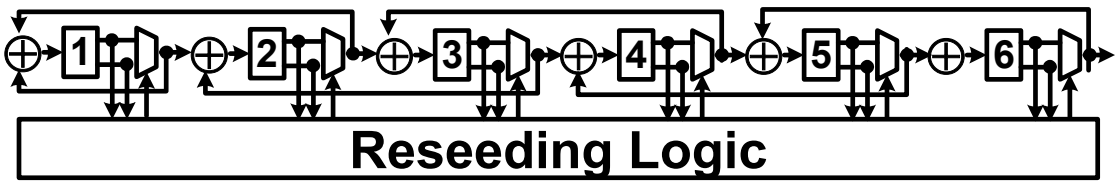


Figure 3.3 Reseeding logic connection to cellular automata.

The output of the reseeding circuit activates the select lines of some of the muxes to invert certain stages of the PRPG such that the desired seed is loaded in the next clock

cycle. The contents of the PRPG registers before reseeding are called *end of sequence* (*EOS*) contents. We also refer to them as the *final state* of the PRPG after loading the last pattern of a consecutive sequence of patterns into the scan chains.

Let us turn our attention to the reseeding circuit itself by looking at the following example. Figure 3.4 is an example using a 4-stage *self-reseeding LFSR* (LFSR with reseeding logic) with a primitive polynomial. The table in part (a) shows the full sequence of the standard LFSR. Assume that we want to reseed after the 6^{th} cycle (c6). The reseeding circuit needs to be an AND gate that takes as inputs the contents of the LFSR at c6. So in the example the input to the reseeding AND is $Q_1'Q_2Q_3'Q_4$. All the cycles that are not part of the desired sequence are considered as don't care inputs to minimize the reseeding circuit. In other words, we can combine the patterns that will activate the reseeding circuit together with all the patterns that won't occur in our desired sequence to generate the maximum possible number of don't cares.

As an example, let the seed be 0100 (c12); we can easily calculate c11 given the polynomial of the LFSR (c11 = 1001). The reason we calculate c11 and not c12 is because we want the seed to be loaded into the LFSR in the next clock cycle. To find the location where c11 is different from c6 we XOR them, XORing c6 with c11 yields 1100 which means that the output of the reseeding AND should generate a logic value 1 at the select lines of the MUXes of Q_1 and Q_2 only.

The truth table for the reseeding circuit is shown in Table 3.1, where Q_i comes from the output of the i^{th} stage and S_j goes into the select lines of the mux of j^{th} stage. The table shows that when the LFSR has the contents 0101 (c6), S_1 and S_2 will be activated to load 1001 (c11) in the LFSR. The patterns between c6 and c11 will not occur so they are don't cares. All the other patterns do not activate any muxes. The resulting circuit for the example in Figure 3.4 is a 3-input AND as shown in the figure.

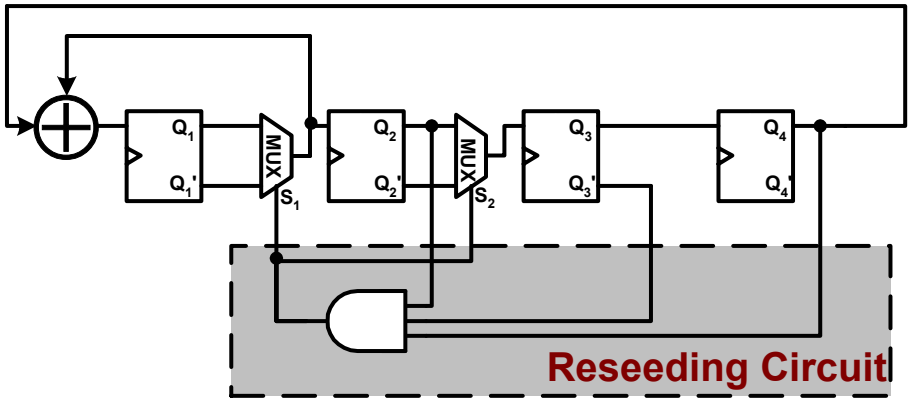
As more seeds are required, every select line of the MUXes will be a function of the end-of-sequence patterns that will activate it to complement the contents of its corresponding flip-flop. We can then optimize the circuit for that select line by combining all the patterns that will activate it together with all the patterns that won't occur in the desired sequence as don't cares. This is like minimizing a function given all

of its on-set patterns as well as all of its impossible patterns (don't care-set). Furthermore, multiple-output minimization can be done for the select lines.

Cycle	Q1	Q2	Q3	Q4	Cycle	Q1	Q2	Q3	Q4
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

End of Sequence (EOS) = c6 = 0101 Seed = 0100 = c12
Select Lines Activated = (c6) XOR (c11) = 0101 XOR 1001 = 1100
=> Select lines of Q1 and Q2 activated

(a)



(b)

Figure 3.4 Example reseeding circuit
(a) Select lines computation (b) Hardware implementation

Table 3.1 Table of Combinations for the Reseeding Circuit Example.

$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$
1 0 0 0	0 0 0 0	0 1 1 1	0 0 0 0	1 1 0 1	d d d d	0 1 0 0	0 0 0 0
1 1 0 0	0 0 0 0	1 0 1 1	0 0 0 0	0 1 1 0	d d d d	0 0 1 0	0 0 0 0
1 1 1 0	0 0 0 0	0 1 0 1	1 1 0 0	0 0 1 1	d d d d	0 0 0 1	0 0 0 0
1 1 1 1	0 0 0 0	1 0 1 0	d d d d	1 0 0 1	0 0 0 0	0 0 0 0	d d d d

3.2.1 Reseeding Circuitry for Transition Faults

There are two ways to apply transition fault test sets to circuits with scan chains. One way is to use pairs of functional clock pulses (launch on capture). Once the 1st

pattern is loaded into the scan chain, a clock pulse is applied so that the response of the combinational logic to the 1st pattern is latched into the flip-flops. Another clock pulse is then applied such that the response of the combinational logic to the 1st pattern is used as the 2nd pattern in the transition fault pair. Logic and fault simulation are used to figure out the response of the 1st pattern and accordingly find the detected faults. The other way is to load the scan chain with the two successive patterns (launch on shift). Once the first pattern is applied, the contents of the scan chain are shifted, the 2nd pattern is applied and the results are captured in the scan chain to be shifted out. The timing diagram for both techniques is shown in Figure 3.5.

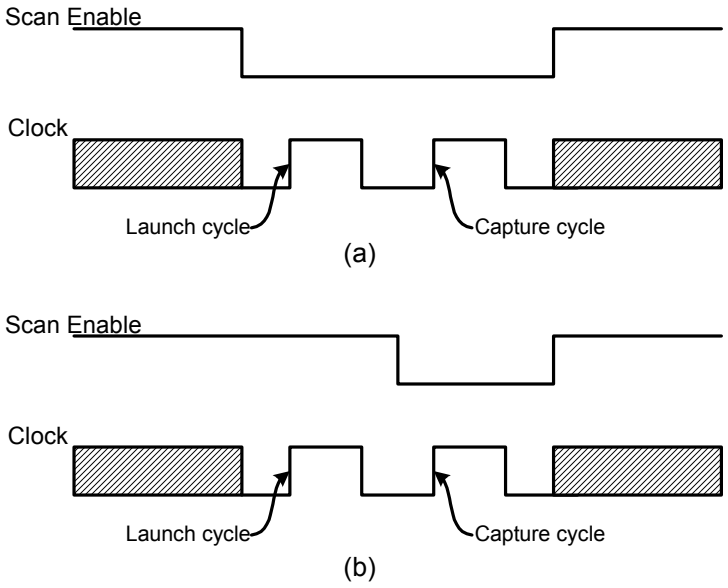


Figure 3.5 Launch on capture and launch on capture timing diagrams.

For our experiments, we used the first technique (launch on capture) for transition faults because the ATPG tool used supported only that option. However, the built-in reseeding technique is applicable with both launch on shift and launch on capture techniques. The reseeding circuit needs to load the PRPG only with the seed for the 1st pattern because the 2nd pattern is the response of the logic to the 1st pattern. The reseeding circuit is synthesized such that it changes the contents of the PRPG from the current values to the 1st pattern in the transition fault pattern pair. This means that our technique requires no extra hardware to apply it to transition faults. Although pairs of 2 vectors are

required for transition faults, only the first pattern needs to be encoded in hardware because the 2nd pattern is the circuit's response to the 1st pattern.

Figure 3.6 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller. The LBIST controller includes the additional control circuitry added for logic BIST. In some BIST architecture, the pattern counter is part of the LBIST controller and it is used to count the patterns applied to the circuit under test (CUT) [Dostie 00].

In a BIST environment, where the PRPG is known in advance and the initial seed and test length are also known, the reseeding circuit may take its inputs from the pattern counter instead of the PRPG contents. The dashed line in Figure 3.6 corresponds to the reseeding circuit taking its inputs from the pattern counter. If a set of test length *TL* is applied to the circuit, the pattern counter will be of size $\lceil \log_2(TL) \rceil$. Since the size of the pattern counter is much smaller than the PRPG, this can lead to large reduction in the complexity of the reseeding circuit because the number of inputs of the reseeding circuit is reduced. For example, a circuit that has a PRPG of 300 flip-flops and applies only 1000 test patterns will have a pattern counter of 10 flip-flops. Using the pattern counter flip-flops as inputs to the reseeding circuit reduces the number of inputs from 300 to 10. Again having both polarities available at the input gives room for further minimization of the reseeding circuit.

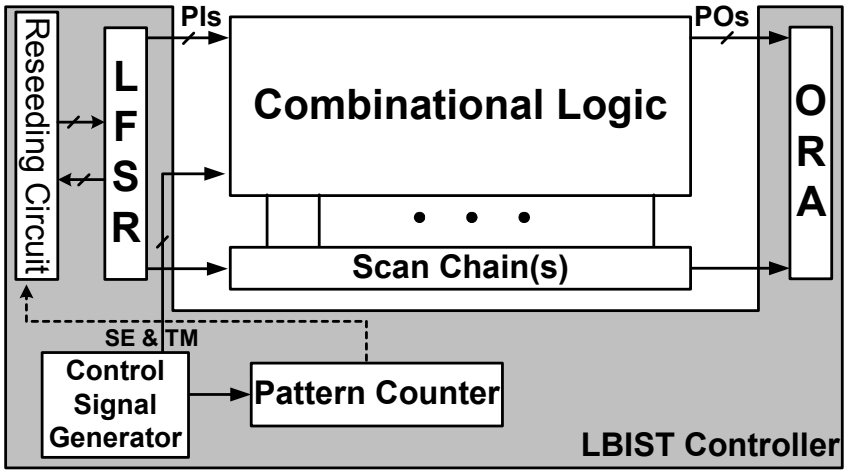


Figure 3.6 Reseeding circuit in a system view of BIST environment.

3.3 Reseeding Algorithm

The reseeding algorithm is based on the following strategies in mixed mode testing: (1) Generate ATPG patterns for faults that were not detected with pseudorandom patterns and calculate the seeds for these patterns. (2) When a seed is loaded into the PRPG, let the PRPG run in autonomous mode for some time because there is a chance that some pseudorandom patterns will detect more faults so that some of the ATPG patterns are not needed. (3) As long as the pseudorandom patterns are detecting more faults, the PRPG should continue in pseudorandom mode. When the pseudorandom patterns become ineffective, the PRPG should be loaded with a new seed. How to quantify the effectiveness of pseudorandom patterns? The answer is in the next paragraphs.

The *coverage improvement threshold* (*CIT*) is the improvement in fault coverage required by the algorithm to continue applying pseudorandom patterns. It is a parameter to quantify the effectiveness of pseudorandom patterns. As long as applying more pseudorandom patterns improves the coverage by at least CIT%, they will be considered effective. When the improvement falls below CIT%, the pseudorandom patterns are considered ineffective. We need to determine how many patterns are simulated before measuring the improvement in coverage. We define the *step size* (*SS*) as the number of patterns simulated before measuring the improvement in coverage. In our simulations, we used different values for the step size. The reseeding algorithm is shown in Appendix B.

The only user-specified parameters for this algorithm are the coverage improvement threshold (CIT) and the step size (SS). At one extreme, choosing a very small CIT means that the user prefers to stick to pseudorandom patterns as long as they have any improvement in the coverage. This in turn means the user wants low hardware overhead for the reseeding circuit. In return for that, the user is willing to have a longer test length. At the other extreme, if the user specifies a very high CIT, it means that he has enough area on the chip for the reseeding circuit while test length is considered very scarce. In other words, the number of seeds to be loaded is inversely proportional to the test length and directly proportional to the area of the self-reseeding circuit. It is up to the user to choose which of the two resources is scarcer in his design. Examples of area overhead requirements are available in Appendix B.

Reseeding based on CIT is one way to choose when to reseed the PRPG. Many other strategies can be used for selecting the reseeding cycles. One way is to choose a fixed length for running the PRPG in autonomous mode after the seed is loaded.

In case of transition faults, the only change to the algorithm is that fault simulation and ATPG should be done for transition faults instead of for SSFs. ATPG generated transition fault patterns have only the 1st pattern specified. The 2nd pattern is the response of the circuit to the 1st pattern. So, only the first patterns of the transition fault patterns' pairs should be encoded in the hardware of the reseeding circuit. For the primary inputs, the reseeding circuit is designed to detect the *EOS* and load the first vector of the pair in the next cycle and then detect the first vector and load the second vector in the next cycle.

3.4 Simulation Results

We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Appendix B. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 m technology library.

We performed some simulation experiments to compare our built-in reseeding with previous work. Most of the previous work assumes one seed per pattern [Koenemann 91] [Hellebrand 95] [Rajski 98] [Krishna 01].

We compare the number of seeds that need to be encoded if our built-in reseeding technique is used and the number of seeds that need to be stored or mapped if previous techniques are used. The number of seeds determines the area of the reseeding or mapping circuit. Since further area minimization heuristics can be applied to all techniques, it is fair to compare them in terms of the number of seeds that need to be mapped or stored for the same coverage. This comparison can be done for all possible combinations of coverage improvement threshold (CIT) and step size, see Sec. 3.3.

Built-in reseeding may require longer test length than deterministic ATPG or top-off patterns[§]. Why should we tolerate this increase in test length? (1) The area is a scarce resource in BIST. (2) The effect of increasing the test length is not as severe with BIST as

[§] Top-off patterns are deterministic test patterns applied to the circuit to improve the fault coverage of the pseudorandom patterns.

it is with external testing because BIST is much faster than external testing. (3) Increasing the test length increases the number of pseudo-random patterns that are likely to be effective in catching unmodeled defects. (4) The increase in test length we have is much smaller than that if only pseudorandom patterns were used. In fact the user has the option to minimize the test length for area overhead.

Table 3.2 shows a comparison of previous techniques and our reseeding technique for 100% single-stuck-at fault coverage. In its best case, built-in reseeding reduces the number of seeds required by an order of magnitude and increases the test length by only a factor of 1.7 compared to using one seed per pattern. Overall, the minimum reduction in the number of patterns is 7%.

Table 3.2 Comparison of Built-In Reseeding and Previous Work Encoding One Seed per Pattern.

Circuit	Number of seeds to be encoded			Test Length		
	One seed per pattern	Built-in reseeding	%Reduction	One seed per pattern	Built-in reseeding	Degradation factor
s1423	7	4	42.9	2,000	8,160	4.1
s1488	4	1	75.0	3,072	4,096	1.3
s1494	4	1	75.0	3,072	4,096	1.3
s5378	60	20	66.7	3,072	27,648	9.0
s9234	93	62	33.3	5,120	76,800	15.0
s13207	121	26	78.5	5,120	60,416	11.8
s15850	41	38	7.3	4,096	45,056	11.0
s35932	18	1	94.4	3,072	5,120	1.7
s38417	78	62	20.5	5,120	89,088	17.4
s38584	107	36	66.4	4,096	76,800	18.8

There are many cases where built-in reseeding offers a considerable improvement in the number of patterns required to test the circuit and at the same time does not cause a large increase in the test length.

Appendix B shows the area overhead of the reseeding circuits for the benchmarks we used. The reseeding circuits given were designed based on 100% SSF fault coverage. The area overhead reported is the minimum area overhead we could achieve for 100% coverage after trying multiple CITs and step sizes.

For most of the circuits, the area overhead ranged from 0.05% to 4%. In a few cases we had a large reseeding circuit.

The minimum area overhead was achieved with different values for CIT for different circuits. This is due to the fact that the number of seeds required to be loaded is highly dependent on the order in which seeds are picked. Our algorithm picks the seeds

with the objective of minimizing the area overhead so this might have different effects on the test length.

An important conclusion from the above results is that we need an algorithm to efficiently find the best order for loading the seeds such that the area of the reseeding circuit is minimized. This is investigated in the next chapter and results are available in [Alyamani 03b].

Appendix B also presents the area overhead of our built-in reseeding technique using the transition fault model. The reseeding circuits given were designed based on the maximum achievable transition fault coverage using deterministic test patterns. For most of the cases, the area overhead ranged from 0.2% to 5% as shown in the appendix.

In most cases, the minimum area overhead is achieved with different step sizes. This is again due to the fact that the number of seeds that need to be loaded is highly dependent on the order in which seeds are picked.

In general the area overhead of the built-in reseeding circuit for transition faults was close to that for SSFs. The reason is that we encode only the 1^{st} vectors of the transitions patterns pairs in the reseeding circuit.

3.5 Summary and Conclusions

This chapter presented a built-in reseeding scheme based on encoding the seeds in an on-chip reseeding circuit. 100% fault coverage can be achieved with the presented scheme without any external testing. The built-in reseeding scheme allows the designer to trade-off area overhead for test length in an optimized way.

Built-in reseeding uses special hardware for the PRPG such that the reseeding circuitry area overhead is minimized. Also, the technique is directly applicable to the transition fault model. The simulation experiments show that the average area overhead is less than 4% for 100% SSF as well as transition fault coverage.

A reseeding algorithm that minimizes the area overhead was also presented. The algorithm takes care of seed selection and reseeding cycle selection. The simulation results show that, for most of the cases, the test length does not have to be maximized when the area overhead is minimized, which is a double win for the presented technique.

Some designs may go through engineering changes in the last phase of the design. For built-in reseeding to achieve the required fault coverage, the reseeding circuitry has to be redesigned. In the cases where redesigning the built-in reseeding circuit is not possible, the original circuit may be used but that will have some impact on the resulting fault coverage and the impact is proportional to the change.

Chapter 4
Seed Ordering

This chapter presents a technique for ordering the seeds to minimize the number of seed loads needed. The number of seed loads is directly proportional to the seed storage required if the seeds are stored. It is also directly proportional to the hardware overhead needed if the seeds are encoded in hardware as explained in Chapter 3. The technique is applicable for any linear PRPG including LFSRs and cellular automata.

The technique presented utilizes the algebraic properties of the PRPG to make it enter states that will generate the desired test patterns without having to explicitly load seeds for many of those patterns.

The seed ordering algorithm increases the number of desired patterns generated from one seed significantly. The algorithm is also fault model independent. Previous algorithms for embedding multiple patterns into a single-seed sequence have much higher computational complexity and are impractical for reasonable size circuits [Lempel 95] [Fagot 99].

In Sec. 4.1, an overview of the previous work is given. The seed ordering algorithm is explained in Sec. 4.2 and simulation results are discussed in Sec. 4.3. Section 4.4 concludes the chapter.

4.1 Related Work

Seeds are calculated from test patterns by solving a system of linear equations as explained in [Koenemann 91]. For every flip-flop in the scan chains, there is a corresponding equation in terms of the flip-flops of the PRPG. Test patterns have many don't care bits. For a given test pattern, we have to satisfy all the equations that correspond to the care bits in that pattern. Since the size of the PRPG is larger than the maximum number of care bits in the test patterns, the number of equations is always smaller than the number of unknowns. These degrees of freedom result in having several solutions (PRPG flip-flop assignments) that can satisfy the constraints (equations). The ways these equations are generated and solved for a given PRPG and a given test pattern is explained in Chapter 6.

Chapter 3 presented a scheme for implementing the seeds in hardware rather than storing them in the tester or in on-chip ROM. By doing so, the chip does not need external testing and does not need a ROM with the associated decoding and loading circuitry. When seeds are implemented in hardware, finding the minimal set of seeds will reduce the hardware needed for the seeds. When seeds are stored, finding the minimal set of seeds will reduce the required storage. The technique presented in this chapter tries to exploit the algebraic properties of the PRPG and the don't care bits in the patterns to generate the maximum number of patterns from a single seed.

In [Lempel 95], an analytical method was presented for finding a single seed for random pattern resistant faults (explained in Chapter 1) based on discrete logarithms. The concept of the algorithm is to identify the position of a certain test pattern in the sequence generated by the PRPG. The complexity of the algorithm depends on the number and size of the prime factors of 2^n-1, where n is the LFSR size. In [Fagot 99], a simulation scheme for calculating initial seeds for LFSRs was presented. The scheme is based on simulating several sequences and picking the one that includes the maximum number of ATPG vectors.

In [Koenemann 00], a technique for skipping useless patterns is presented. The technique is based on having a Seed Skip Data Storage (SSDS) inside the tester. Fault simulation is performed to identify the useful (fault dropping) and useless (non fault dropping) sequences of patterns. The SSDS stores a sequence of numbers corresponding to useful and useless sequence lengths. Using additional control logic, the useless patterns are not loaded from the PRPG to the scan chain of the device under test. The SSDS reduces the storage needed for the test patterns. The control logic that skips the useless patterns reduces the test application time.

4.2 Seed Ordering Algorithm

The BIST architecture assumed for the seed ordering algorithm is shown in Figure 4.1. The algorithm is applicable with any number of scan chains and any phase shifter. The *phase shifter* is a set of XOR gates placed between the PRPG and the scan chains to prevent structural dependencies between patterns shifted into different scan chains. The only difference with multiple scan chains compared to a single chain is in seed

calculation. Chapter 6 explains phase shifters and seed calculation in details. The seeds are then either loaded from the tester or encoded in hardware on chip as explained in Chapter 3. If the PRPG runs in pseudorandom mode after loading the seeds, some of the pseudorandom patterns may detect more faults and accordingly some ATPG patterns may not be needed. This is because their corresponding faults are already covered with the pseudorandom patterns. For such a scheme to work efficiently, we should optimize the order of the seeds to considerably reduce the number of seed loads into the PRPG.

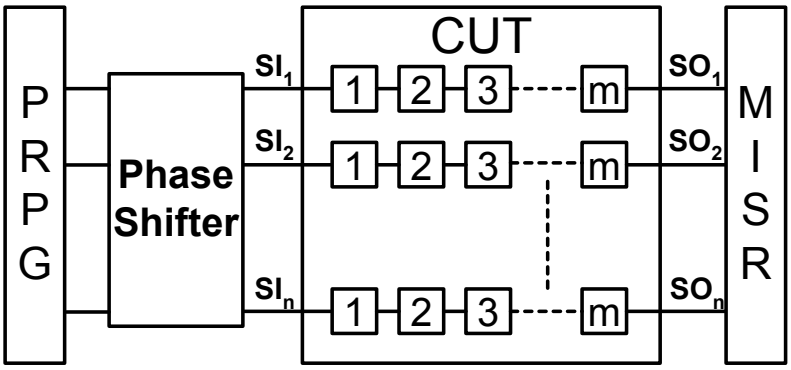


Figure 4.1 Multiple scan chains with a phase shifter.

Our seed ordering algorithm starts with pseudorandom patterns to detect the easy faults and then generates deterministic test patterns for the undetected faults. It picks the deterministic test pattern with the largest number of specified bits and encodes it into a seed. We refer to the seed as the initial state ($s(0)$) of the PRPG. By running the PRPG for m cycles, where m is the length of the longest scan chain, the seed expands into the desired test pattern in the scan chains. We refer to the state of the PRPG after the seed is expanded in the scan chains as the final state ($s(m+1)$). If the final state of the PRPG can expand into one of the remaining test patterns, then we do not need to load a seed for that pattern.

Consider a PRPG whose characteristic polynomial is

$$f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$$

Let $s(t)$ be the state of the PRPG at time t. Then $s(t+1) = s(t)H$, where H is called the transition matrix for the PRPG. By associativity of matrix multiplication, $s(t+1) = s(0)H^{t+1}$. If the length of the longest scan chain is m, and seed i is given by $s_i(0)$, then the contents of the PRPG after the scan chains are loaded is given by $s_i(m+1)$.

The ordering algorithm we present is based on looking ahead in the PRPG sequence by finding $s_i(m+1)$ for seed i and trying to find whether it generates any of the unapplied test patterns. If it can generate a test pattern j, then $s_j(0)$ (the seed for pattern j) does not need to be loaded into the PRPG. If a match is not found we can search for a match with $s_i(d(m+1))$, where $1 \leq d \leq d_{max}$. The d_{max} is a parameter that corresponds to the number of scan shifts we are willing to continue running the PRPG in pseudorandom mode before loading the next seed.

Given H, we precompute $H^{(m+1)}$, $H^{2(m+1)}$,…, $H^{d(m+1)}$, …. We keep multiplying $H^{d(m+1)}$ by the current seed $s(0)$ to get $s(d(m+1))$ until we find a match with one of the other seeds or d exceeds d_{max}. If a match is found, then that seed does not need to be stored or loaded. Computing $s(d(m+1))$ from $s(0)$ involves a single matrix multiplication.

As an example, take as PRPG a 4-stage LFSR whose polynomial is $f(x) = x^4 + x^3 + 1$. The LFSR sequence is shown in Table 4.1. Assume that the seeds are 0111, 0101, 1001 and 0001, which correspond to cycles 4, 6, 11 and 14 of the LFSR sequence shown. If d_{max} used is 3 then we will need to load S4 and S11. If d_{max} used is 5, we will only need to load S4. The example is for parallel BIST (test per clock) for simplicity. For serial BIST (test per scan), we need to include the cycles needed to fill the scan chains into account while finding the match in the ordering algorithm. Those cycles are taken care of by raising the transition matrix H to the power $m+1$ as shown earlier, where m is the length of the longest scan chain. This means that we try to match the contents of the PRPG after multiples of $m+1$.

Table 4.1 Example LFSR Sequence.

Cycle	Q1	Q2	Q3	Q4	Cycle	Q1	Q2	Q3	Q4
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

The algorithm for matching the final state of the PRPG with a seed for another pattern is explained in Chapter 6.

Given a set of seeds and a user-specified d_{max}, the best ordering is the ordering that will result in the minimum number of seeds that need to be loaded into the PRPG. To find an ordering that minimizes the seed loads, we organize the patterns in sequences. The sequence includes seeds that occur within d_{max} distance from one another and cover multiple ATPG patterns.

The seed ordering algorithm is shown in Appendix C. The output is a set of seeds that need to be loaded into the PRPG to expand into the desired patterns. After every seed is loaded, we run the PRPG for up to d_{max} scan cycles to generate all patterns that this seed can generate within d_{max}.

It is possible to find the best order of the seeds by simulating all possible permutations. However, this is prohibitively time consuming.

4.3 Simulation Results

We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Appendix C. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μ library.

In [Koenemann 91] [Hellebrand 95] [Rajski 98] [Touba 00] and [Krishna 01], a single seed per pattern is assumed. In our technique, we try to generate multiple patterns with the same seed by reordering the seeds such that we need to load the minimal number of seeds into the LFSR.

Table 4.2 Number of Seeds for Our Technique Compared to Seed per Pattern.

Circuit	Cell Area	Seed/ Pattern	Our Tech.	Red. %
s1238	2,740	30	7	76.7
s1423	4,531	7	3	57.1
s1488	3,555	4	1	75.0
s1494	3,563	4	1	75.0
s5378	14,377	35	7	80.0
s9234	25,840	89	57	36.0
s13207	44,255	74	12	83.8
s15850	48,494	38	35	7.9
s38584	115,855	55	16	70.9

Table 4.2 shows a comparison between previous techniques that assume a single seed per pattern and our seed ordering technique in terms of the number of seeds needed

for 100% single-stuck fault coverage. It is always the case that using our seed ordering technique results in a reduction in the number of seeds required. The reduction varies from 8% to 84%.

Three arbitrary orderings for the seeds were tried and pseudorandom patterns were applied after every deterministic seed to improve the coverage. Appendix C shows the reduction gained by using our ordering compared to the average number of seeds with the arbitrary orderings. The reduction varies from 5% to 80%.

Seed ordering was also combined with built-in reseeding to see the impact of seed ordering on the hardware overhead for built-in reseeding. Appendix C shows the area overhead needed for built-in reseeding [Alyamani 03a] if the arbitrary orderings are used and if our ordering technique is used. It also shows the reduction in area overhead gained by using our ordering compared to the average overhead with arbitrary orderings. The reduction varies from 7% to 84%.

4.4 Conclusions

This chapter presents a seed-ordering technique based on exploiting the algebraic properties of the PRPG.

The simulation experiments showed that the area overhead and the storage needed are reduced by up to 80% when the presented ordering technique is used compared to arbitrary ordering. The main characteristics that make the ordering technique effective are: (1) Exploiting the linearity of the PRPG and the associativity of matrix multiplication to avoid simulation. (2) Avoiding unnecessary computation to reduce the complexity of finding the best order of the seeds, and (3) Matching the future PRPG state with deterministic test patterns to avoid loading seeds for those patterns.

The technique presented provides a solution for the problem while avoiding high computational complexity as in previous analytical solutions and avoiding long simulation times as in previous simulation techniques.

Chapter 5
Seed Encoding

In this chapter, I present a technique for representing the seed by the number of clock cycles the PRPG needs to run to reach it. The purpose is to avoid loading the seeds. Instead, we load the number of clock cycles the PRPG needs to run to reach to the intended seed. Normally, seeds have to occur at m+1 clock cycles from one another according to the BIST architecture, where m is the length of the longest scan chain. The seed encoding technique presented here avoids this requirement by modifying the BIST architecture to allow variable numbers of clock pulses between seeds. The main feature of the technique is that it encodes a seed with a much smaller vector and so saves on seed storage.

In Chapter 4, a technique was presented for minimizing the number of seeds to be loaded by ordering the seeds and by exploiting the degrees of freedom in solving for the seeds. The contributions in this chapter are: (1) A seed encoding technique that encodes the seeds in a much smaller vector that corresponds to the number of cycles to reach it. The technique also exploits the don't cares in test patterns. (2) An architecture that implements the encoding technique. The technique is applicable for single-stuck and transition faults.

In Sec. 5.1, the seed encoding technique is presented. Section 5.2 presents the architecture for seed encoding. Simulation results are discussed in Sec. 5.3. Section 5.4 concludes the chapter.

5.1 Seed Encoding

The BIST architecture we assume is the same as in Figure 4.1. The seed encoding technique is applicable with any number of scan chains and any phase shifter. The way seeds are calculated is explained in Chapter 6. The seeds are then either loaded from the tester or encoded in hardware on chip as explained in Chapter 3.

If the final state of the PRPG after loading the scan chains can generate another deterministic pattern, then a seed for that pattern does not have to be loaded into the PRPG. If the final state of the PRPG does not match another seed, we can clock the PRPG a few times until we reach a state that can produce a deterministic test pattern.

Also, we can exploit the don't cares in test patterns so that we increase the chances of matching a deterministic seed with the final state of the PRPG.

If the longest scan chain is of length m and seed i is $s_i(0)$, then the contents of the PRPG after the scan chains are loaded is given by $s_i(m+1)$. PRPGs can be LFSRs or cellular automata.

The encoding algorithm we present is based on looking ahead in the sequence of the PRPG by finding $s_i(m+1)$ for seed i and trying to find whether it matches with any of the other seeds $s_j(0)$. If a match is not found, we can search for a match with $s_i(m+d)$, where $1 \leq d \leq d_{max}$. The parameter d_{max} corresponds to the number of clock cycles we are willing to continue running the PRPG before loading the next seed. Notice that this is different from seed ordering because we are searching for a match with $s_i(m+d)$ instead of $s_i(d(m+1))$. This change saves the time needed to fill the scan chains but requires modifying the BIST architecture to allow capturing after a different number of clock cycles. The architecture is presented in Sec. 5.2.

Given a set of seeds and a user-specified d_{max}, we order the seeds to minimize the number of seeds that need to be loaded as explained in Chapter 4.

5.2 Seed Encoding Architecture

A fundamental issue in applying the seed encoding technique is how to make the PRPG run for a variable number of cycles for different seeds. Normally, the PRPG runs for a number of cycles equal to the length of the longest scan chain before a capture cycle. To use our encoding technique, which represents the seed by the number of clock cycles required to reach it, we need to have the PRPG run for a variable number of cycles.

In a usual logic BIST architecture, a bit counter is used to choose when to disable the Scan Enable (SE) signal for capturing. One way to implement this is to have the bit counter loaded with the value that corresponds to the length of the longest scan chain for every pattern. The bit counter is then decremented by 1 at each clock cycle. When the bit counter reaches zero, it means that the test pattern is loaded into the scan chains, so SE is disabled for one clock cycle, and so on. The length of the scan chains is stored in a register and loaded into the bit counter with every pattern. Our technique is based on

running the PRPG for a number of cycles to reach the desired seed. To implement this, we need to load the bit counter register with different values corresponding to the number of cycles before the next capture. Unloading the scan chains starts right after the capture cycle. So for encoded seeds there are extra cycles after unloading the response to pattern *i* and before capturing the response for pattern *i*+1.

5.2.1 Running Seed Encoding from an ATE

To run our technique from a tester, we have two types of seeds; seeds that need to be loaded into the PRPG, *loaded seeds*, and seeds that can be reached by continuing to run the PRPG for additional cycles after loading the scan chains, *encoded seeds*. We use the name encoded seeds because these seeds are encoded into the number of cycles the PRPG needs to run to reach them.

Seed size: How efficient is this encoding? Why not just load all seeds? This question can be answered by a simple example, take a circuit of 10,000 flip flops that has 10 scan chains of length 1000 each. If the maximum number of care bits in the test patterns is 500 (5%), we need a PRPG of size 520 [Koenemann 91]. Since the length of the scan chain is 1000, the bit counter needs to have only 10 bits. So, by encoding the seed into the number of cycles to reach it we get a 98% (52×) reduction in seed storage. Even if we decide to run the PRPG for up to 1000 additional cycles before reaching the next desired seed, the impact on the size of the bit counter is a single bit.

Test time: In terms of test length, for the example above, loading the PRPG with a new seed takes 520 cycles. Loading the bit counter register takes 11 clock cycles. This means that we can search for a match with another seed in up to 508 cycles while saving on the test time and at the same time saving on tester storage.

We efficiently exploit the don't care bits in test patterns to increase the probability that a given LFSR state can generate a given test pattern as explained in Chapter 6.

5.2.2 Full BIST Implementation

The seed encoding technique can be applied for full on-chip BIST with 100% single-stuck fault coverage. For that, we need to have a reseeding circuit for loaded seeds and a seed encoding circuit for encoded seeds.

Loaded seeds: We use the built-in reseeding architecture presented in Chapter 3. The operation of the reseeding circuit is as follows: the PRPG starts running in pseudorandom mode according to the reseeding algorithm [Alyamani 03a]. Once it is time for reseeding, a seed is loaded into the PRPG, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the PRPG in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the PRPG.

Figure 3.2 shows the structure of an LFSR and its interaction with the reseeding circuit and Figure 3.3 shows the same for cellular automata. For our technique, we use muxed flip-flops. By activating the select line of a given mux, the logic value in the corresponding stage is inverted in the next clock cycle. The muxed flip-flops are similar to those used for scan chains. The output of the reseeding circuit activates the select lines of the muxes to invert certain stages of the PRPG such that the desired seed is loaded in the next cycle.

Encoded seeds: For the encoded seeds, we need a combinational circuit that takes as its input the value of the PRPG; the output of this circuit should be loaded into the bit counter register. Instead of reading the values of the PRPG stages as input to the seed encoding circuit, the value of the pattern counter can be used as input. The majority of the input combinations for the seed encoding circuit will load the bit counter register with the length of the scan chains. The pattern counter input combinations that correspond to seeds to be encoded will load a different value in the bit counter register. That value corresponds to the length of the scan chains plus the number of clock pulses that need to be applied before reaching the desired seed.

We can synthesize the seed encoding circuit from a table of combinations. The input combinations that correspond to normally loaded seeds should have the longest scan chain's length as the output value. The input combinations that correspond to encoded seeds should have the output as the scan chains length in addition to the number of the clock cycles needed to reach to the seed. Just as in the circuit for built-in reseeding, all input combinations that do not occur in the desired test sequence can be treated as don't cares to help minimize the seed encoding circuit.

Figure 5.1 shows where the reseeding and seed encoding circuits fit in a system level view of a circuit with an LBIST controller, which includes the additional control circuitry added for logic BIST.

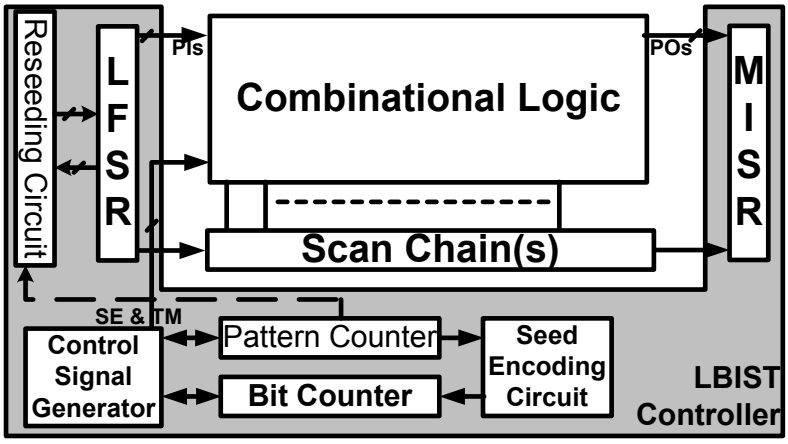


Figure 5.1 Reseeding and seed encoding circuits in a logic BIST environment.

If the CUT or the scan chain is changed, then the reseeding circuit and the seed encoding circuit must be resynthesized based on the new design and the new test patterns. However, if the seed encoding technique is applied from a tester, then changing the design results only in changes in the test patterns and accordingly the seeds that are stored. If it is preferable to apply the technique with full BIST, then it may be better to apply it after the design is stable and no more changes are applied to the circuit or the scan chain.

5.3 Simulation Results

We performed our experiments on some of the ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Appendix D. We took into account the maximum number of additional clock cycles to calculate the size of the bit counter. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μm technology library.

The experiment was designed so that pseudorandom patterns are applied first. Then, test patterns are generated for the undetected faults and the seeds are calculated from the test patterns.

In [Koenemann 91], [Hellebrand 95], [Touba 00], [Rajski 98], and [Krishna 01], a single seed per pattern is assumed. In our technique, we encode as many of the seeds as we can by the number of additional clock cycles to reach them. The remaining seeds have to be loaded from the tester or encoded on-chip. Appendix D shows the reduction in number of seeds if our technique is used compared to the case if one seed per pattern is used (as in most of the previous work). The reduction in number of seeds varied between 23% and 85%.

Since seeds have different sizes with the seed encoding technique, we also compare the actual storage in number of bits required for our seed encoding technique compared to the number of bits required for the classical one seed per pattern.

Table 5.1 shows the seed storage needed for the seed per pattern scheme and the seed storage needed for our scheme. The storage is calculated by multiplying the number of loaded seeds by the PRPG size and the number of encoded seeds by the bit counter size. The reduction gained by using our scheme varies from 25% to 86%.

Table 5.1 Seed Storage Needed by our Technique Compared to Seed per Pattern.

Circuit	Circuit Cell Area	Seed per Pattern storage	Storage for Our Technique			Storage Rduction %
			Loaded Seeds	Encoded Seeds	Total	
s953	2,286	987	189	112	301	69.50
s1196	2,722	1380	195	96	291	78.91
s1238	2,740	1455	180	72	252	82.68
s1423	4,531	850	450	0	450	47.06
s1488	3,555	534	24	54	78	85.39
s1494	3,563	510	18	66	84	83.53
s5378	14,377	2135	1586	9	1595	25.29
s9234	25,840	9360	6560	60	6620	29.27
s13207	44,255	14560	2400	160	2560	82.42
s35932	106,198	420	60	0	60	85.71

The area overhead required for our technique implemented completely on chip with reseeding and seed encoding circuits is comparable to the areas shown in [Alyamani 03b] where the overhead ranged between 0.3% and 10% and mostly less than 3%.

5.4 Conclusion

This chapter presented a seed-encoding technique based on running a variable number of clock cycles before loading the seed.

The simulation experiments showed that the seed storage needed is reduced by 25% to 86% when our encoding technique is used compared to storing a single seed per pattern. The main characteristics that make the seed encoding technique effective are: (1) Running pseudorandom patterns after loading the seeds, (2) Ordering the seeds to load the minimum number of seeds, (3) Encoding the seeds by the number of cycles needed to reach them, and (4) Exploiting the don't cares in test patterns efficiently while avoiding simulation to find a match between a given state and a deterministic test pattern.

Chapter 6
Seed Calculation

In this chapter, we show how seeds are calculated and how the unspecified bits are exploited to increase the chances of finding a match between the final state of the PRPG and a seed for one of the test patterns. The algorithm we present for seed calculation is applicable for LFSRs and cellular automata with any phase shifters.

We will start by explaining all the concepts in this chapter for LFSRs and then show how they can be applied to cellular automata.

If the current state of an LFSR is $s(t)$, then the next state can be found by the formula $s(t+1) = s(t) \times H$, where H is called the transition matrix for the LFSR. For a given seed i, the LFSR starts at the initial state $s_i(0)$. After filling the scan chains, the LFSR is at state $s_i(m+1)$, where m is the length of the longest scan chain.

Figure 6.1 shows an example of a 4-stage LFSR connected to one scan chain with 10 flip-flops. This figure will be used as an example to illustrate the equation-generation technique. We start with a simple example and then explain the technique for multiple scan chains.

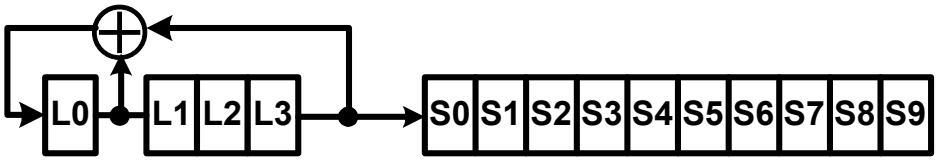


Figure 6.1 Example LFSR for seed calculation.

For every flip-flop in the scan chain, there is a corresponding equation in terms of the bits of the LFSR. Let us label the scan chain flip-flops as $S_0 \ldots S_{m-1}$, where m is the size of the scan chain. Also, let us label the stages of the LFSR as $L_0 \ldots L_{n-1}$, where n is the size of the LFSR. In the example above, the equations for the n most significant flip-flops of the scan chain are: $S_9 = L_3$, $S_8 = L_2$, $S_7 = L_1$, and $S_6 = L_0$ because after n clock cycles the bits of the seed end up in the most significant bits of the scan chain. The reader is invited to verify the remaining equations:

$S_5 = L_0 \oplus L_3$
$S_4 = L_0 \oplus L_2 \oplus L_3$
$S_3 = L_0 \oplus L_1 \oplus L_2 \oplus L_3$
$S_2 = L_1 \oplus L_2 \oplus L_3$
$S_1 = L_0 \oplus L_1 \oplus L_2$
$S_0 = L_1 \oplus L_3$

The equations above represent the values of the scan chain flip-flops at time m = 10 in terms of the values of the LFSR flip-flops at time 0.

We can represent the above equations by an $m \times n$ matrix in which the rows correspond to the scan chain flip-flops and the columns correspond to the LFSR stages. An entry (i,j) is 1 if and only if L_j appears in the equation of S_i. According to this system, the following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \\ S_9 \end{matrix}$$
$$L_0 \quad L_1 \quad L_2 \quad L_3$$

The above matrix E represents a pool of equations. It has one equation per flip-flop of the scan chain in terms of the LFSR flip-flops. Given a test pattern with unspecified bits, only the equations that correspond to the specified bits need to be satisfied to generate the seed. The equations that correspond to the specified bits are drawn from the E matrix and then a system of equations is constructed using these equations as the matrix A, the vector corresponding to the LFSR bits (the seed) as X, and the specified bits vector as B in the equation $[A][X] = [B]$.

Let us use the test pattern (0X1X0X1XXX) as an example with Figure 6.1. The specified bits are S_0, S_2, S_4 and S_6. We draw the equations for those bits from the E matrix to generate the A matrix. We get the following system:

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} L_0 \\ L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Solving the above system for L_0 … L_3, we get the seed (1010). By loading this seed in the LFSR, the pattern (0010011010) is generated in the scan chain after 10 clock cycles. This pattern satisfies the required pattern for all specified bits.

In the case of multiple scan chains, if the architecture shown in Figure 6.1 above is used, there will be undesired correlation between the bits of patterns that are shifted into the scan chains. This is because the patterns that are shifted in different scan chains will be shifted versions of one another. To avoid this correlation, the outputs of the LFSR stages must go through a phase shifter.

The purpose of the phase shifter is to avoid the structural dependency. If chains *i* and *j* are fed directly from the LFSR, the patterns they will hold will be shifted versions of one another. The phase shifter makes sure that the sequence fed into chain *j* is apart from the sequence that goes into chain *i* by at least the length of chain *i*. This means that if the output of the LFSR is looked at as a stream of bits, the sequence that goes into chain *j* is *r* bits apart from the sequence that goes into chain *i,* where *r* is the length of chain *i.* This way the structural dependency is avoided between all the flip-flops of the two chains.

The phase shifter corresponds to a linear shift by *r* stages from the previous chain, where *r* is the depth of the previous scan chain. The phase shifter for a given stage is a linear sum of some stages from the LFSR. Figure 6.2 shows 6 scan chains fed by a 4-stage LFSR through a phase shifter [Bardell 87].

The phase shifter for chain *i* can be represented by the vector $p^i = \begin{bmatrix} p_0^i & p_1^i & \cdots & p_{n-3}^i & p_{n-2}^i & p_{n-1}^i \end{bmatrix}$, where p_j^i is one if the XOR feeding scan chain *i* has the output of stage *j* of the LFSR as one of its inputs. For example, $p^2 = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}$ is the phase shifting vector for the 2nd scan chain.

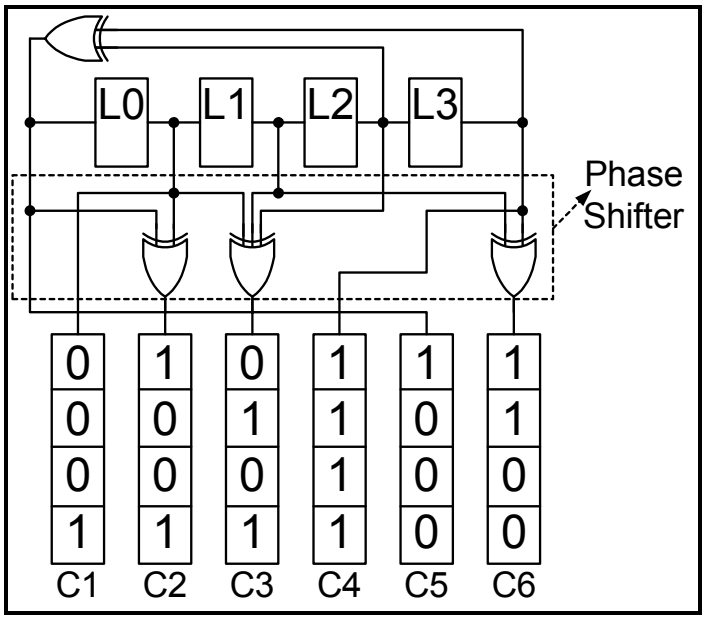


Figure 6.2 scan chains fed through a phase shifter [Bardell 87].

The algorithm that generates the equations matrix E_S for a scan chain S that is fed through a phase shifter p^S starts with the selection vector p^S. The algorithm starts by assigning the selection vector p^S to the last row of E_S. Row i of the matrix E_S is generated by multiplying the transition matrix H by the following row i+1. This algorithm can be used with any phase shifter and any number of chains. The equations matrix generation algorithm is shown in Algorithm 6.1.

1. *m: depth of the scan chains*
2. *n: size of PRPG*
3. *p^S: the phase shifter for the current scan chain*
4. *E_S: equations matrix for the current scan chain*
5. ***E_S-Generator (m, n, h, E_s)***
6. $E^{(m-1)} = p^S$
7. ***for*** *i=m-2 to 0*
8. $E^i = H \times (E^{(i+1)})^T$
9. ***endfor***
10. ***end***

Algorithm 6.1 Generating equations matrix for a scan chain fed through a phase shifter.

Let us now revisit the concept of matching a state of the LFSR with a seed for a deterministic test patterns. We are trying to match the final state of the LFSR (after loading the scan chains) with a seed that will expand into one of the ATPG patterns. The specified bits in an ATPG pattern correspond to a set of equations in terms of the LFSR stages. The equations are actually restrictions that must be satisfied so that the test pattern is generated from a given seed. The equations can be taken directly from the matrix E for the corresponding care bits of the test pattern. An LFSR state satisfies an equation for a specified bit if it is a solution for that equation. In other words, if the dot product of the equation and the LFSR state yields the specified bit needed, then the LFSR state satisfies that equation. If the final state of the LFSR $s(m+1)$ satisfies all equations for a given pattern then we know that the final state is a seed for this pattern. Accordingly, we do not have to load an extra seed for that pattern.

Based on Koenemann's recommendation in [Koenemann 91], the size of the LFSR is 20 more than the maximum number of specified bits in a test pattern. This means that the number of equations is always less than the number of unknowns and so we can have multiple seeds that can produce the same pattern. By trying to satisfy only the care bits of a given pattern, we utilize all the don't care bits in that pattern to improve the chances of using a final state as a seed for such a pattern.

One way to determine if a final state will produce a certain pattern is to simulate the LFSR sequence and check the final pattern in the scan chain. However, this is a time consuming solution. Another way for checking the same thing is to extract the vectors that correspond to the care bit equations from the E matrix and then multiply those vectors with the final state of the LFSR. If the result of this multiplication matches the care bit in the test pattern, then the final state is a valid seed for the desired test pattern.

To reduce the time required for checking the validity of a PRPG state as a seed for a test pattern, we can use a *binary decision diagram* (BDD) to implement the checking process. This way, once it is found that a certain specified bit in the ATPG pattern is not producable from the PRPG state, the checking process is aborted because it is required that all specified bits are satisfied. This reduces the check time by an average of 50%. More information about BDDs is available in [De Micheli 94].

As an example, for the LFSR shown in Figure 6.1, assume that the ATPG tool generates the following two patterns: (X0X1X10XXX, 0XXX1XXXXX). To generate the seed for the 1st pattern, we solve the following system:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} L_0 \\ L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The solution of the system is $s_0(0) = 0111$. The next step is to calculate $s_0(m+1)$, which is the value of the contents of the LFSR after the chain is loaded with the pattern. $s_0(m+1) = s_0(0) \times H^{(m+1)}$. For $m = 10$, $s_0(m+1) = 1000$. Now, we check if this state $s_0(m+1)$ can generate the second pattern. The second pattern has 2 care bits; for every one of them there is a corresponding equation in terms of the LFSR bits. All we need to do is to verify that the bits of $s_0(m+1)$ satisfy the equations for the care bits of the second pattern. This is achieved if the following condition is satisfied:

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \left(s_0(m+1)\right)^T = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Since the condition is satisfied, the final state of the LFSR after loading the 1st pattern is a valid seed for the 2nd pattern. With more than 2 seeds, the ordering algorithm explained in Chapter 4 should be used to exploit this efficiently.

Since the seed calculation algorithm is based entirely on the transition matrix H and the phase-shift vector, it is directly applicable to cellular automata. Below is an example of applying it to cellular automata.

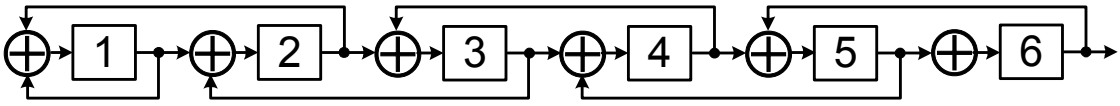


Figure 6.3 A 6-stage cellular automaton.

In the example of Figure 6.3, assume that the CA is connected to a scan chain at the output of stage 6 of the CA. Also assume that the scan chain has 9 stages S_0 … S_8.

The equation for the deepest stage of the scan chain is $S_8 = L_5$, because after m clock cycles the most significant bit of the seed ends up in the most significant bit of the scan chain. The reader is invited to verify the remaining equations:

$S_7 = L_4$
$S_6 = L_3 \oplus L_5$
$S_5 = L_2$
$S_4 = L_1 \oplus L_3$
$S_3 = L_0 \oplus L_4$
$S_2 = L_0 \oplus L_1 \oplus L_2 \oplus L_4$
$S_1 = L_2 \oplus L_5$
$S_0 = L_1 \oplus L_3 \oplus L_4$

The following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \end{matrix}$$
$$\begin{matrix} L_0 & L_1 & L_2 & L_3 & L_4 & L_5 \end{matrix}$$

The same steps in Algorithm 6.1 can be used with cellular automata and with any phase shifter to generate the matrix E. Once the matrix E is generated, the same matching procedure can be used for any state of the cellular automata and any test pattern. In fact, the algorithm is applicable to any linear machine that can be characterized with a transition matrix.

Chapter 7
Concluding Remarks

Built-in self test emerged as a low cost alternative for using automatic test equipment to store test patterns. The driving force behind this dissertation was to overcome the obstacles that made BIST an unacceptable solution in many cases.

In this dissertation, techniques were presented to deal with the pseudorandom nature of BIST and introduce deterministic alternatives in some aspects of BIST to overcome the obstacles. An important feature of this dissertation is that it did not entirely replace pseudorandom BIST with deterministic BIST. Instead, it kept the pseudorandom BIST to utilize its advantages while replacing some of its aspects with deterministic BIST to improve its quality. The following paragraphs summarize how this was accomplished.

Resolving the illegal states in digital circuits has existed as an obstacle for pseudorandom testing of such circuits for a long time. It has been mostly dealt with using ways that sacrifice the fault coverage or add additional levels of logic that may not be needed.

The illegal state detection technique presented in this dissertation for fixing the illegal states eliminates the pseudorandom nature only when needed. Hence, it does not compromise the fault coverage of the legal pseudorandom patterns. The ISD technique can be implemented without changing the given design and is hence, non-intrusive.

The ISD technique can be implemented in on-chip hardware that detects the illegal patterns and masks their effects. It can also be applied from a tester using a single additional bit per test pattern. Applying it from the tester leaves room for engineering changes to the design without having to re-synthesize the ISD circuit. Implementing it in hardware enables testing the circuit while in the field without the need for a tester.

To deal with the low fault coverage disadvantage of BIST, this dissertation presented a built-in reseeding scheme based on encoding the seeds in an on-chip reseeding circuit. 100% fault coverage can be achieved with the presented scheme without any external testing. Built-in reseeding does not eliminate pseudorandom testing because of its many advantages like detecting most of the faults with very low overhead and also detecting many unmodeled faults. In fact, built-in reseeding improved the pseudorandom nature of BIST by allowing the PRPG to jump to different states in the

pattern space to apply pseudorandom patterns that would have never been applied otherwise. Built-in reseeding is directly applicable to the transition fault model as well as the single stuck fault model.

This dissertation also presented a seed ordering technique to improve built-in self test by making it more deterministic in targeting specific faults while avoiding to load the actual seeds into the PRPG. It rather tries to choose the seeds intelligently so that the PRPG goes through a sequence of seeds that will generate the maximum number of test patterns while reducing the number of seeds.

The simulation experiments showed that the area overhead and the storage needed are reduced by up to 80% when our ordering technique is used compared to arbitrary ordering. Seed ordering provides a solution for the problem while avoiding high computational complexity like previous analytical solutions and avoiding long simulation times like previous simulation techniques.

A seed-encoding technique was also presented to avoid the disadvantage of having to load the whole scan chain to find an appropriate seed. The price in seed encoding is paid in a slight modification to the BIST architecture. The simulation experiments showed that the seed storage needed is reduced by 25% to 86% when our encoding technique is used compared to storing a single seed per pattern.

The main characteristics that make the seed ordering and seed encoding techniques effective are: (1) Running pseudorandom patterns after loading the seeds (2) Ordering the seeds to load the minimum number of seeds, (3) Encoding the seeds by the number of cycles needed to reach them, and (4) Exploiting the don't care bits in test patterns and avoiding simulation to find a match between a given seed and a deterministic test pattern.

The decision of choosing between seed ordering and seed encoding should be guided by the restrictions and characteristics of the design. Seed ordering can be implemented without any changes in the BIST architecture of the circuit. However, if slight changes in the BIST architecture can be accommodated as explained in Chapter 5, then seed encoding offers a higher degree of flexibility in designing the test procedure and so offers more reduction in the time and storage requirements of the test.

References

[Alyamani 02a] Al-Yamani A., and E. J. McCluskey, “Low-Overhead Built-In BIST Reseeding”, Test Resource Partitioning Workshop, 2002.

[Alyamani 02b] Al-Yamani, Ahmad, S. Mitra, and E.J. McCluskey, “Testing Digital Circuits with Constraints”, *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems* (*DFT'02*), pp. 195 –203, 2002.

[Alyamani 02c] Al-Yamani A., and E. J. McCluskey, “Built-In Reseeding for Built-In Self Test”, CRC Technical Report 02-1, 2002.

[Alyamani 02d] Al-Yamani, Ahmad, S. Mitra, and E.J. McCluskey, “Avoiding Illegal States in Pseudorandom Testing of Digital Circuits”, *CRC Technical Report 02-2*, 2002.

[Alyamani 03a] Al-Yamani A., and E. J. McCluskey, “Built-In Reseeding for Serial BIST”, VLSI Test Symposium, Apr., 2003.

[Alyamani 03b] Al-Yamani A., S. Mitra, and E.J. McCluskey, “BIST Reseeding with Very Few Seeds”, VLSI Test Symposium, Apr., 2003.

[Alyamani 03c] Al-Yamani A., and E.J. McCluskey, “Seed Encoding with LFSRs and Cellular Automata”, Design Automation Conference, June, 2003.

[Alyamani 03d] Al-Yamani, A., C. Tseng, J. Li, S. Mitra and E.J. McCluskey, "Pseudorandom BIST: Theory, Simulation and Tester Data," submitted to the *International Test Conference* (*ITC'03*), 2003.

[Abadir 99] Abadir, M., and R. Raina, “Design-for-test Methodology for Motorola PowerPCTM Microprocessors,” *Proc. International Test Conference*, pp. 810 –819, 1999.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, “Built-In Test for VLSI”, John Wiley, New York, 1987.

[Barnhart 01] Barnhart, C., B. Keller, B. Koenemann and R. Walther, “OPMISR: The Foundation for Compressed ATPG Vectors”, Proc. of International Test Conf., pp. 748-757, 2001.

[Benowitz 75] Benowitz, N. et al., "An advanced fault isolation system for digital logic," IEEE Trans. on Computers, vol.C-24, no.5, pp. 489-97, May 1975.

[Chakrabarty 00] Chakrabarty, K., B. Murray, and V. Iyengar, “Deterministic Built-in Test Pattern Generation for High-Performance Circuits Using Twisted-Ring Counters,” IEEE Transactions of Very Large Scale Integration Systems, Vol. 8, No. 5, pp. 633-636, Oct. 2000.

[Cheng 95] Cheng, K.-T., and C.J. Lin, “Timing-Driven Test Point Insertion for Full-Scan and Partial-Scan BIST,” Proc. of International Test Conference pp.506-514, 1995.

[Chiang 97] Chiang, C.-H., S. Gupta, “BIST TPGs for Faults in Board Level Interconnect via Boundary Scan,” *Proc. VLSI Test Symposium*, pp. 376-382, 1997.

[Crouch 95] Crouch, Alfred, and M. D. Pressly, “Self Re-seeding Linear Feedback Shift Register (LFSR) Data Processing System for Generating a Pseudo-random Test Bit Stream and Method of Operation,” US Patent 5,383,143, Jan. 1995.

[Das Gupta 81] Das Gupta, S., R. G. Walther, T. W. Williams and E. B. Eichelberger, “An Enhancement to LSSD and some Applications of LSSD in Reliability,

Availability and Serviceability," *Proc. International Symposium on Fault-Tolerant Computing*, pp. 32-34, 1981.
[De Micheli 94] De Micheli, G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
[Dostie 00] Dostie, *Design for at-speed test, diagnosis and measurement*, Kluwer publishers, Boston, MA, 2000.
[Dworak 99] Dworak, J. et al., "Modeling the probability of defect excitation for a commercial IC with implications for stuck-at fault-based ATPG strategies," Proc. 1999 International Test Conference, pp.1031-1037, 1999.
[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", IBM Journal of Research and Development, Vol. 27, No. 3, pp. 265-272, May 1983.
[Eichelberger 89] Eichelberger, E. B., E. Lindbloom, F. Motica, and J. Waicukauski, "Weigted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 1989.
[Fagot 99] Fagot, C., O. Gascuel, P. Girard and C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test," Proc. of European Test Workshop, pp. 7-14, 1999.
[Fleming 92] Fleming; L. O. and Walther; John S., "Local tristate control circuit", US Patent 5136185, Aug 1992.
[Franco 95] Franco, P., R.L. Stokes, W.D. Farwell, and E.J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Part 1: Description of Experiment," CRC TR 94-5.
[Hetherington 99] Hetherington, G., et al., "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," *Proc. International Test Conference*, pp. 358 –367, 1999.
[Hellebrand 92] Hellebrand, S., S. Tranick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," Proc. of International Test Conference, pp. 120-129, 1992.
[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Transactions on Computers, Vol. 44, No. 2, pp. 223-233, Feb. 1995.
[Hellebrand 00] Hellebrand, S., H.-G. Liang and H.-J. Wunderlich, "A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters," Proc. of International Test Conference, pp. 778-784, 2000.
[Huang 97] Huang, L.-R., J.-Y. Jou, and S.-Y. Kuo, "Gauss-Elimination-Based Generation of Multiple Seed-Polynomial Pairs for LFSR," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 9, pp. 1015-1024, Sep. 1997.
[ITC 99] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html, Benchmark Suite for ITC 1999.
[Kagaris 99] Kagaris, D., and S. Tragoudas, "On the Design of Optimal Counter-Based Schemes for Test Set Embedding," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 2, pp. 219-230, Feb. 1999.

[Kim 96] Kim, Kee, “Scan-Based Built-In Self Test (BIST) with Automatic Reseeding of Pattern Generator,” US Patent 5,574,733, Nov. 1996.
[Koenemann 91] Koenemann, B., “LFSR-Coded Test Patterns for Scan Designs,” Proc. of European Test Conference, pp. 237-242, 1991.
[Koenemann 00] Koenemann, B., “System for Test Data Storage Reduction,” US Patent 6,041,429, Mar. 2000.
[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, “Test Vector Encoding Using Partial LFSR Reseeding” Proc. of International Test Conference, pp. 885-893, 2001.
[Lempel 95] Lempel, M., S. Gupta and M. Breuer, “Test Embedding with Discrete Logarithms,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 5, pp. 554-566, May 1995.
[Levitt 95] Levitt, M., et. al., “Testability, Debuggability, and Manufacturability Features of the UltraSPARC TM –I Microprocessor,” *Proc. International Test Conference*, pp. 157-166, 1995.
[Li 99] Li, J. C.M., et.al., “ELF35 Experiment – Chip and Experiment Design,” TR 99-3, Center for Reliable Computing, Stanford University, 1999.
[Li 01] Li, J. C.M., C.W. Tseng, and E.J. McCluskey, “Testing for Resistive Opens and Stuck Opens” Proc. of International Test Conference, 2001.
[McCluskey 85] McCluskey, E.J., “Built-In Self-Test Techniques,” *IEEE Design & Test*, pp. 21-28, Apr. 1985.
[McCluskey 86] McCluskey, E.J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1986.
[McCluskey 88] McCluskey, E.J., S. Makar, S. Mourad, and K.D. Wagner, “Probability Model for Pseudo Random Test Sequence,” IEEE Tran. On Computer-Aided Design, Vol.7, No.1, pp.68-74, Jan. 1988.
[McCluskey 89] McCluskey, E.J. and F. Buelow, “IC quality and Test Transparancy,” IEEE Tran. On Industrial Electronics, Vol.36, No.2, pp.197-202, May. 1989.
[McCluskey 00] McCluskey, E.J. and C.W. Tseng, “Stuck-at Tests vs. Real Defects,” Proc. 2000 International Test Conference, pp.336-343, 2000.
[Mitra 97] Mitra, S., L. J. Avra and E. J. McCluskey, “Scan Synthesis for One-hot Signals,” *Proc. International Test Conference*, pp. 714-722, 1997.
[Muradali 90] Muradali, F., V.K. Agrawal, B. Nadeau-Dostie, “A New Procedure for Weighted Random Built-In Self Test,” Proc. of International Test Conference, pp. 660-668, 1990.
[Nigh 97] Nigh, P., W. Needham, K. Butler, P. Maxwell, and R. Aitken, "An Experimental Study Comparing the Relative Effectiveness of Functional, Scan, IDDq, and Delay-fault Testing," Proc. IEEE VLSI Test Symposium, pp. 459-464, 1997.
[Pateras 95] Pateras, S. and M. S. Schmookler, “Avoiding Unknown States when Scanning Mutually Exclusive Latches,” *Proc. International Test Conference*, pp. 311-318, 1995.
[Pradhan 99] Pradhan, D., and M. Chatterjee, “GLFSR – A New Test Pattern Generator for Built-in-Self-Test,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 2, pp. 238-247, Feb. 1999.
[Pyron 99] Pyron, C., et al., “DFT Advances in the Motorola's MPC7400, a PowerPCTM G4 Microprocessor,” *Proc. International Test Conference*, pp. 137 –146, 1999.

[Raina 00] Raina, R., et al., "DFT Advances in Motorola's Next-Generation 74xx PowerPCTM Microprocessor," *Proc. International Test Conf*., pp. 131 –140, 2000.
[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , "Test Data Decompression for Multiple Scan Designs with Boundary Scan," IEEE Transactions on Computers, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.
[Sato 00] Sato, Y., et. al., "A BIST Approach for Very Deep Sub-Micron (VDSM) Defects," Proc. 2000 International Test Conference, pp.283-291, 2000.
[Savir 90] Savir, J., and W. McAnney, "A Multiple Seed Linear Feedback Shift Register," Proc. of International Test Conference, pp. 657-659, 1990.
[Seth 84] Seth, S.C. and V.D. Agrawal, "Characterizing the LSI Yield Equation from Wafer Test Data," IEEE Tran. On Computer-Aided Design, Vol.7, No.CAD-3, Apr. 1984.
[Synopsys 97] Scan Synthesis User Guide, Synopsys (1997.08).
[Touba 95] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," Proc. of International Test Conference, pp. 674-682, 1995.
[Touba 96a] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. of VLSI Test Symposium, pp. 2-8, 1996.
[Touba 96b] Touba, N.A., and E.J. McCluskey, "Altering a Pseudo-Random Bit Sequence for Scan-Based BIST," Proc. of International Test Conference, pp. 167-175, 1996.
[Touba 00] Touba, N. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.
[Tseng 01] Tseng, C.W., and E.J. McCluskey, "Multiple Output Propagation Transition Fault ATPG" Proc. of International Test Conference, 2001.
[Venkataraman 93] Venkataraman, S., J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," Proc. of International Conference on Computer-Aided Design, Vol. 9, No. 6, pp. 572-577, 1993.
[William 81] William, T.W. and N.C. Brown, "Defect Level as a Function of Fault Coverage," IEEE Tran., Vol.C-30, pp. 987-988, Dec. 1981.
[Wunderlich 90] Wunderlich, H.-J., "Multiple Distributions for Biased Random Test Patterns," IEEE Transactions on Computer-Aided Design, Vol. 9, No. 6, pp.584-593, Jun. 1990.
[Zacharia 95] Zacharia, N., J. Rajski and J. Tyszer, "Decompression of Test Data Using Variable-Length Seed LFSRs," Proc. of VLSI Test Symposium, pp. 426-433, 1995.

Appendix A

Testing Digital Circuits with Constraints

Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 195-203, 2002.

Testing Digital Circuits with Constraints*

Ahmad A. Al-Yamani[1], Subhasish Mitra[2] and Edward J. McCluskey[1]
[1]*Center for Reliable Computing* [2]*Intel Corporation*
Stanford University, Stanford, CA *Sacramento, CA*
{alyamani, smitra, ejm@crc.stanford.edu}

Abstract

Many digital circuits have constraints on the logic values a set of signal lines can have. In this paper, we present two new techniques for detecting the illegal combinations of logic values in digital circuits and two techniques for preventing them from damaging the circuit or corrupting the test results. The hardware overhead of our technique is minimal and imposes negligible delay overhead. Simulation results show that the area overhead of our techniques is less than 1%. Unlike previous techniques, the fault coverage of the legal test patterns in a given test set is not sacrificed with our techniques. Furthermore, our techniques are applicable during IC production test, BIST, board-level tests and system-level tests.

1. Introduction

Correct operation of digital circuits is not guaranteed if illegal combinations of logic values appear on some signal lines. For example, many digital designs contain logic that is controlled by one-out-of-n (one-hot) signals. *One-hot signals* in digital circuits are a set of signal lines of which no more than one signal should be active at a time. Examples include an n-to-1 selector implemented with n transmission gates, enabled by different signals, and a bus controlled by tri-state buffers.

For a tri-state bus, an illegal state occurs when more than one driver is enabled to drive the bus at the same time; this state is known as a *contention state* or a *multiple-hot* state. In case of contention, if the two drivers are writing two different values, the value

* This work was done at Stanford-CRC and was supported by King Fahd University of Petroleum and Minerals. It was also supported by DARPA under contract No. DABT63-97-C-0024 (ROAR project). The authors would like to thank Chien-Mo Li and Chao-Wen Tseng for their help.

of the bus is non-deterministic. This non-determinism may propagate to the output. A more severe effect is that the circuit may be damaged because pull-up and pull-down transistors are both activated. A similar non-determinism can happen if none of the tri-state buffers is enabled to drive the bus in which case the bus will be *floating* (*zero-hot*).

The scan-path method is a widely used design for testability (DFT) technique [1]. The inclusion of on-chip circuitry to provide test vectors or to analyze output responses is called *built-in self-test* (*BIST*). One BIST technique is pseudo-random testing using a linear feedback shift register (LFSR) [2].

Illegal states can appear during scan in and out because the patterns are shifted serially through the bistables. Several solutions are available in the literature for this problem. For example, bistables that may cause illegal states can be controlled with control points [3], removed from the scan chain or bypassed [4]. The techniques we present are intended for illegal states that occur during test pattern application rather than during scan in and out.

Illegal values during pattern application can be caused by pseudo-random testing. When the circuit is tested using automatic test pattern generation (ATPG), the one-hot condition can be provided as a constraint to the ATPG tool (as in FastScan) such that none of the generated patterns results in multiple-hot or zero-hot values in one-hot signals during test pattern application. However, when pseudo-random patterns are applied (externally or internally) or when the ATPG tool does not check for illegal values, some of the patterns generated may cause illegal states.

In this paper, we present new techniques for detecting illegal states in digital circuits and masking their effects. Unlike previous techniques, our techniques have no impact on the fault coverage achieved with the legal patterns of a given test set. Although we will discuss our techniques in the context of one-hot signals, the techniques are directly applicable with arbitrary constraints on logic values that can appear on a set of signal lines. In Sec. 2 of this paper, we give an overview of the previous work. We present our techniques in Sec. 3 and discuss the simulation results in Sec. 4. We conclude in Sec. 5.

2. Previous work

If the one-hot signals are generated directly from the bistables, those bistables can be designed to hold only one-hot values. Such bistables are called one-hot bistables. A more general solution is to impose constraints so that pass transistor selectors are not used to implement multiplexers [5]. Another approach is to gate the output of the one-hot signals during scan with the scan enable (SE) signal, resulting in a particular one-hot value enforced on the one hot signals irrespective of the contents of the bistables. Only one of the signals should be OR-ed with the SE signal while all the other signals should be AND-ed with the complement of the SE signal [6]. When SE is 1 during scanning, a particular one-hot value is enforced. This technique ensures safety (one-hot property) during scan-in and scan-out operations, but multiple-hot or zero-hot values may appear on the one-hot signals if pseudo-random patterns are used. A similar scheme was used in [7]. Figure A.1 shows how this scheme is implemented to insure a one-hot value on E1, E2, E3 and E4 during scan-in and scan-out. A limitation of this scheme is that the SE signal has to be specially routed to avoid delays that may cause none one-hot states to occur before the fixing circuit is activated. A generalization of this approach is to enforce a particular one-hot value on the one-hot signals throughout the test mode of operation by using a special signal. Although this solution avoids illegal states during scan-in and scan-out operations and also when pseudo-random patterns are used to test the circuit, the fault coverage can fall drastically because the logic may not be sufficiently tested since the enforced one-hot value does not change during testing [3]. A variant of this scheme is described in [4].

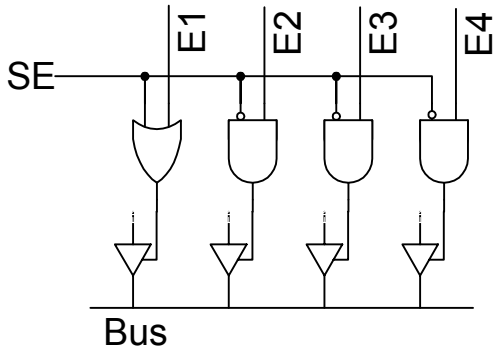


Figure A.1 Insuring a one-hot value during scan-in and out.

Another technique is to use a priority encoder for the one-hot signals. The priority encoder takes n arbitrary inputs and produces n one-hot outputs [8, 9]. This technique modifies the original design and adds delay overhead equivalent to several levels of logic.

The L3 latch based Level Sensitive Scan Design (LSSD) technique presented in [10] can also be used to take care of the one-hot signal problem. In [10], an L3 latch is used to break the path from an LSSD to a non-LSSD network. A *non-LSSD network* is a part of the circuit that contains non-LSSD latches. This way the non-LSSD logic does not interfere with the testing of the LSSD logic. This scheme does not ensure safety when pseudo-random patterns are used to test the circuit under test (CUT) during test mode because the final scanned-in value may violate the one-hot condition (because it is a random pattern).

In [11], a technique is presented where the one-hot bistable outputs are encoded and then decoded such that they only hold one-hot signals. This scheme overcomes the problem of pseudorandom pattern application. However, the scheme has limitations due to the fact that it assumes that one-hot signals are produced directly from the one-hot bistables. Moreover, the scheme requires additional bistables when the number of one-hot signals is not a power of 2. A similar scheme has been reported in [12]. The techniques described in [9] target the one-hot problem during synthesis and require changes in the logic design of the circuit.

Chiang and Gupta presented a technique for designing a test pattern generator (TPG) to test the board level interconnects via boundary scan. Their technique takes care

of the one-hot constraints and the fault coverage constraints in designing the TPG [13]. In board level interconnect testing, the TPG directly controls the nets with tri-state controllers. There is no logic between the TPG and the interconnects. This technique is not applicable if the tri-states are on-chip because of the logic between the TPG and the tri-state drivers. Most of the previous work relies on the assumption that the TPG directly controls the tri-state drivers. Our technique eliminates this assumption, so it is more widely applicable.

The technique we present in this paper does not impose restrictions on the original design. It satisfies the one-hot constraints during test pattern application so it complements the techniques that satisfy the constraints during scan-in and out. It also does not affect the fault coverage for the legal patterns, that do not cause an illegal state, in a given test set. While we describe our technique in the context of one-hot signals, the technique can be directly applied for circuits with arbitrary constraints on logic values that can appear on a set of signal lines.

3. Illegal state detection (ISD)

The purpose of an *Illegal State Detection* (*ISD*) circuit is to detect whether an input pattern applied to the circuit under test (CUT) causes illegal values on a set of signals in the circuit. In this section we present our techniques for designing and implementing the ISD circuit. We also discuss two techniques for fixing the illegal state and taking the system back to a legal state. The first technique is based on static fixing which requires extra hardware and adds logic. The advantage of this technique is its simplicity and wide applicability. The second technique is based on skipping the patterns that cause the illegal state. This is done with no additional hardware in the circuit and with no additional delay. However, this technique requires that the test set is known in advance and is not changed later.

3.1 Illegal state detection by back-tracing

The ISD function should be expressed in terms of the primary inputs and the bistable outputs. A reasonable way for finding the ISD function is by extracting the functions of the one-hot signals in terms of the primary inputs and the bistable outputs.

After analyzing the one-hot signals and expressing them in terms of the primary inputs and the bistables, the illegal state detection circuit is implemented so that it produces logic value 1 if the values on the current pattern causes an illegal value and it produces a 0 otherwise.

For the purpose of illustration, let us consider a circuit with full-scan. Suppose that there are 4 one-hot signal lines E_1, E_2, E_3, and E_4. As shown in Figure A.2, $E_1 \ldots E_4$ are connected to the enable inputs of tri-state gates whose outputs are connected to a common bus. From the given combinational logic, we can find the Boolean expressions for the logic functions of $E_1 \ldots E_4$.

Let the Boolean functions corresponding to E1, E2, E3, and E4 be F1, F2, F3, and F4, respectively. We form the Boolean function ISD = (F1F2′F3′F4′ + F1′F2F3′F4′ + F1′F2′F3F4′ + F1′F2′F3′F4)′. The ISD function produces a 0 when an input combination guarantees one-hot values on the signal lines E1…E4; it produces a 1 otherwise. We can generate the ISD function by synthesizing the ISD circuit expressed in terms of the bistables outputs and the primary inputs. The area overhead of the ISD circuit depends on the logic between the tri-state drivers and the scan bistables.

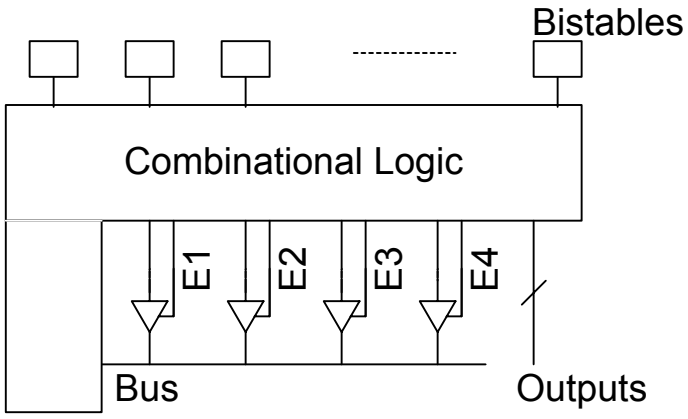


Figure A.2 Example of tri-state busses in logic circuits.

The way we perform the analysis for a one-hot signal is by extracting the logic cones of such signals. The logic cones can be extracted by tracing the gates whose outputs are connected to the one-hot signals. Then the logic cones are extracted for all inputs signals of all such gates. This recursive process continues until the inputs of the current gates are either primary inputs to the circuit or outputs of bistables in the scan chain.

Figure A.3 shows an example for a combinational circuit with one-hot signals E_1 and E_2. By applying the extraction algorithm, we obtain E_1 and E_2 in terms of the primary inputs as shown. Next, the circuits of E_1 and E_2 are pruned and simplified to extract their simplified functions.

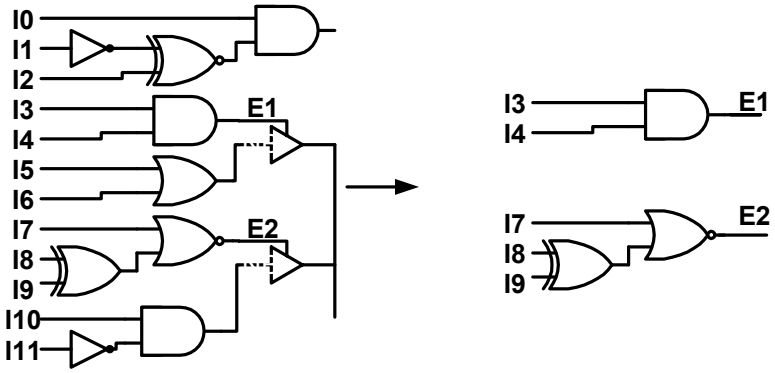


Figure A.3 Node analysis for the one-hot signals.

Figure A.4 shows the ISD circuit in a finite state machine structure of a digital circuit. If the ISD circuit relies only on the bistables that drive the one-hot signals, then there is a chance that the one-hot condition will not be satisfied after the capture cycle. This is because the shifted pattern may be such that the one-hot condition is satisfied during pattern application (launch) while the contents of the bistables after capture may violate the one-hot condition. This can cause damage to the circuit depending on when the scan enable signal is applied next. This problem never occurs in the normal mode of operation because the circuit operates such that only one signal is activated at time according to the specification.

To avoid the above problem, the ISD circuit should obtain its inputs not only from the bistables that drive the one-hot signals but also from the bistables that control those bistables. In other words, the ISD circuit should rely on the bistables that control the one-hot signals to a depth of two cycles. This way, the ISD circuit is activated one cycle before there is a problem. So, it will be activated when the response of the combinational logic may cause a non-one-hot combination at the one-hot signals.

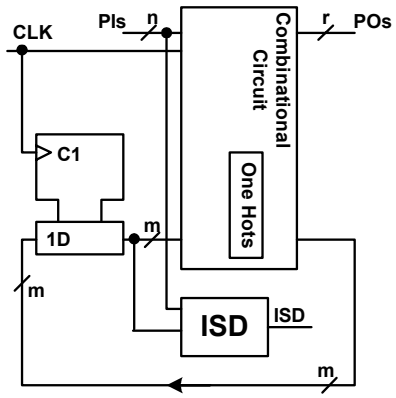


Figure A.4 ISD circuit in a finite state machine structure.

Figure A.5 shows the ISD circuit based on two clock cycles protection for the one-hot signals. The figure only shows a conceptual representation for the bistables that directly control the one-hot signals and the logic around it, and the bistables that control such bistables. The 2nd set of bistables is shown to be considered in designing the ISD circuit such that it detects the illegal state to a depth of 2 cycles.

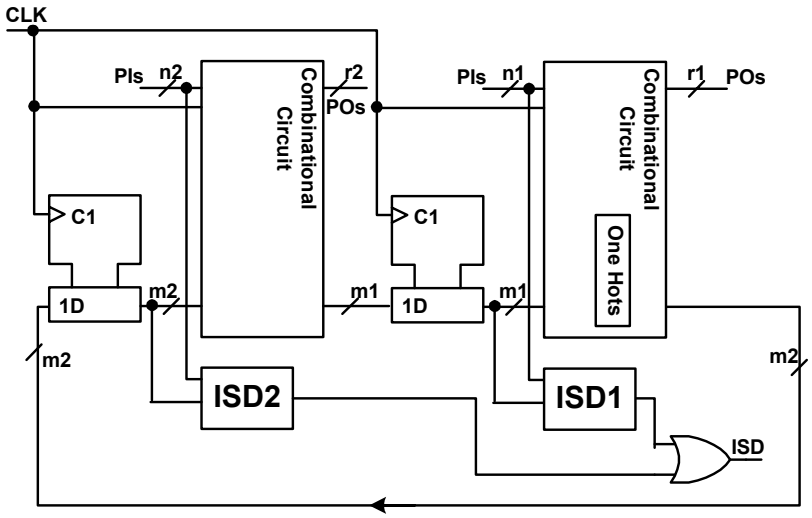


Figure A.5 Two-cycle based ISD circuit.

We implement the ISD circuit by analyzing the one-hot signals and expressing them in terms of the bistables and primary input. Although the two-cycle based ISD looks like a sequential problem, it's easy to implement it for our purpose because all we need is to continue the analysis to the bistables for two cycles instead of one.

3.2 Illegal state detection using BIST pattern counter

In a BIST environment, if analyzing the one-hot signals in terms of the primary inputs and bistables is not possible (e.g. for intellectual property reasons), and if the test set is known in advance, the ISD circuit may be designed to be the logical sum of the patterns that cause contention or floating values. Such patterns are found by simulation. This simulation step is not an extra step because it has to be performed anyhow for the signatures to be calculated.

One way to reduce the area of the ISD circuit is to make it depend on the value of the "pattern counter" of the BIST controller. The counter value of those patterns that can cause contention can be used as the minterms for the ISD. If a set of M test patterns is applied to the circuit the pattern counter will be of size $\lceil \log_2(M) \rceil$. This can lead to massive reduction in the complexity of the ISD circuit. From here on, the ISD circuit based on tracing will be called ISD-Trace and the ISD circuit based on the BIST pattern counter will be called ISD-Counter.

Even with the ISD-Counter circuit, the response of the combinational logic to a legal pattern can be illegal. If this will cause damage to the circuit, the ISD circuit should be activated to avoid the damage. A second level ISD should be designed just like what we did with ISD trace. The 2nd level ISD should be implemented by applying two cycles of logic simulation for the test patterns. The 1st cycle is to get the response to the pattern and make sure it's a legal pattern; the 2nd cycle checks if the response to the 1st pulse causes an illegal state.

3.3 Fixing the illegal state

The implementation shown in Figure A.6 is one of many possible ways to force a legal state on the one-hot signal lines in test mode. We will call the added circuitry for forcing the legal state *fixing logic*. Scan enable (*SE*) is a scan signal, which is 1 when patterns are scanned in or out of the scan chains. The test mode (*TM*) signal is 1 in the test mode, i.e. during scan in and out and also during the capture cycle. If the tester

(whether external or built-in) has a signal that disables the capture cycle, that signal can be used to disable capturing the combinational logic outputs of the illegal patterns. Otherwise, the TM signal is needed for this fixing logic in order to guarantee the one-hot value during the capture cycle where SE is 0.

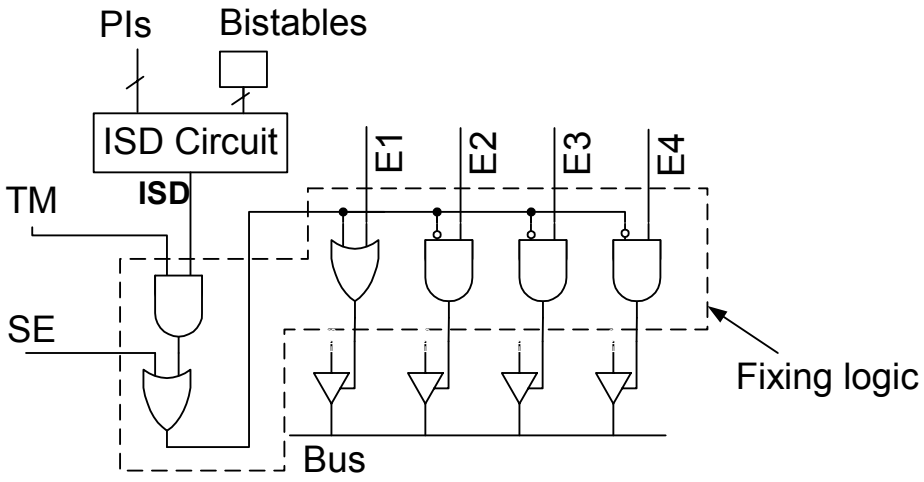


Figure A.6 ISD circuit and its fixing logic control tri-state enables.

If the circuit is BISTed, the TM signal is provided by the BIST controller according to some BIST architectures [14]. During normal operation, the TM and the SE inputs are both 0 and static; hence, the delay overhead introduced is very small because the output of the ISD circuit is blocked by the TM signal. The above technique solves the illegal state problem for a set of signals during scan in and out and during the capture cycle because the static fixing logic is active whenever the scan enable is active or the ISD circuit is 1 in test mode. This technique requires much less area and causes less performance overhead than the priority encoder.

In a BIST environment, the ISD circuit can be part of the BIST controller. Figure A.7 shows an overview of the logic BIST controller in a digital circuit. It shows where the ISD circuit and the fixing logic fit in the system level view of a BIST environment.

The ISD circuit can take its inputs from the bistables in the circuit scan chain. If adding fan-outs to the scan chain bistables is not desirable, the ISD circuit can have extra bistables to store the contents of the bistables that determine the value of ISD. In a BIST environment, the “bit counter” is used to count the bits shifted into the scan chain. The

value of the bit counter can be used to identify the bits that will go into the bistables that control the one-hot signals. These bits should be stored in the extra bistables to determine the value of ISD. So, the ISD circuit bistables can take their inputs from the pseudo random pattern generator (PRPG) and the bit counter through simple control logic.

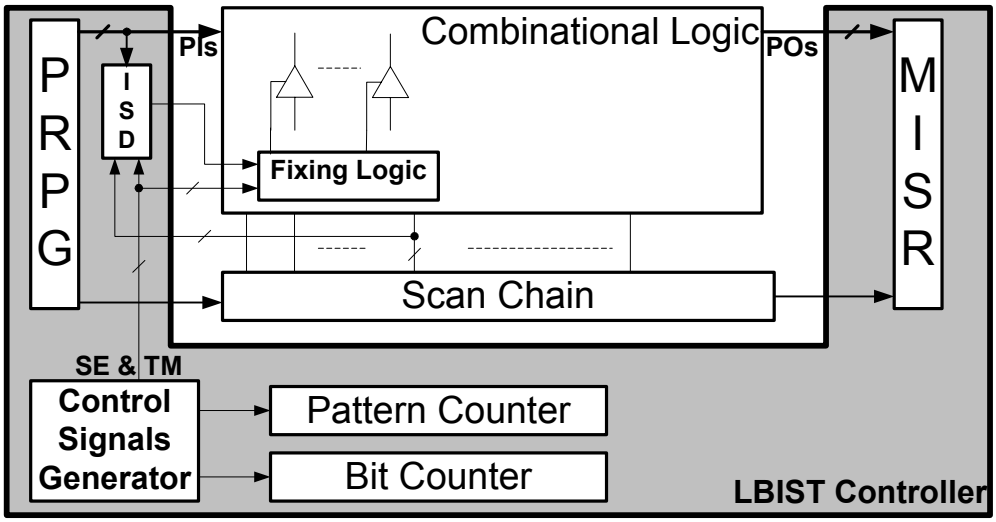


Figure A.7 ISD circuit and fixing logic in a system level view of a BIST environment.

3.4 Skipping the illegal state

A small logic circuit can be added to the BIST controller such that the patterns that cause the illegal states are not applied to the circuit. In this case, there is no need for any intrusion and no delay overhead, not even the negligible overhead, is caused by the ISD circuit.

Normally, the test patterns are scanned into the bistables with value 1 on the scan enable (SE) signal. Once the pattern is shifted in, SE is turned off for one clock cycle such that the scan chain stores the results of the combinational circuitry. Then SE is turned on again for these results to be shifted out. The logic added to the BIST controller should keep the SE signal 1 during the test application cycle in case the pattern will cause an illegal state. This way the pattern is not applied to the circuit and no corrupted output

is read out. This same pattern skipping technique can be used as a general solution to eliminate the problem of undetermined values (Xs) in pseudo-random testing.

Consider a BIST controller that uses the bit counter to count the bits shifted into the scan chain. Assume that for every scan-in and scan-out sequence, the bit counter is loaded with the count of bistables in the scan chain. It gets decremented every cycle. SE is kept 1 until the bit counter reaches zero. To avoid applying the pattern that causes an illegal state, we need to set SE to one if (1) ISD is 1 and (2) the bit counter is 0. Figure A.8 shows an example for the logic needed for this purpose. This logic can be part of the BIST controller. Also, in case of BIST, if multiple scan chains are used with separate SE signals, then we only need to disable capturing for the chain(s) that cause the illegal state. This way the coverage can be further improved.

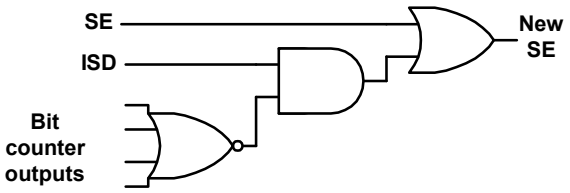


Figure A.8 Circuitry for skipping the illegal state.

If the design has a separate test clock, the test clock can be ANDed with the complement of ISD such that the pattern is clocked into the scan chain only if it's a legal pattern.

4. Simulation Results

We performed our experiments on I992, which is an industrial pipeline ASIC design from ITC99 benchmark suite [15]. The design has approximately 20,000 gates and four clock domains. There are no internal memories. I992 is the only circuit with tri-state drivers in the ITC benchmark suite. The other benchmark suites (e.g. ISCAS and MCNC) don't have any circuits with tri-state drivers.

I992 is an industrial circuit that has many one-hot signals. This benchmark has 6 internal busses controlled by tri-state drivers. Figure A.9 shows a top view of the busses controlled by tri-state drivers in the circuit.

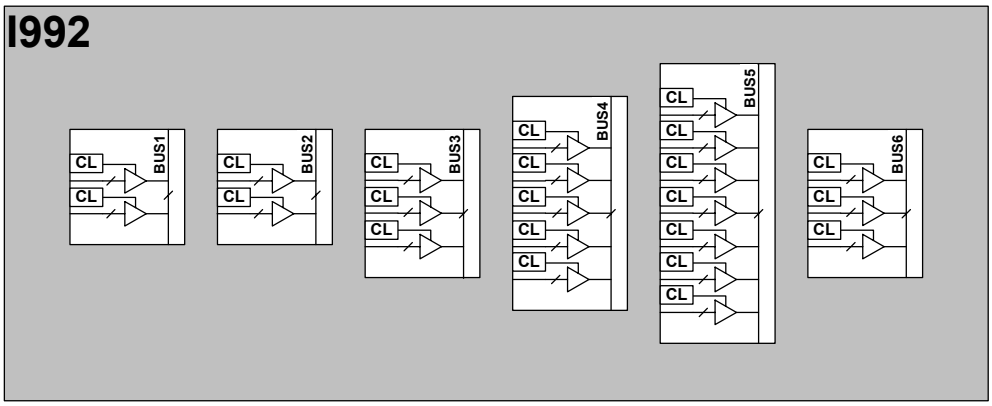


Figure A.9 Top view of I992 with its tri-state drivers.

Table A.1 shows the main features of the I992 circuit. It shows the primary inputs, primary outputs, inouts, the number of bistables, the number of busses driven by tri-state drivers and the cell area. The cell area is given in LSI cell area units where an inverter is equivalent to 1 unit. The library used for technology mapping is LSI Logic G10p library.

Table A.1 I992 Characteristics.

PIs	POs	InOuts	Bistables	Busses	Area
31	88	176	508	6	29817

We traced back the tri-state enables to the bistables and primary inputs. Then we synthesized the ISD circuits for all the busses. Table A.2 shows the details of all the tri-state busses in the circuit. It also shows the areas of the ISD circuits and their overhead relative to the area of the I992 circuit. The total area overhead of the 6 ISD circuits of I992 is only 0.892% of the total area of the circuit. This is an extremely low area overhead for our ISD circuits that will completely eliminate the contention problem in the circuit.

Table A.2 I992 ISD Circuits' Area Overhead.

Busses	Bus width	Tri-state drivers	ISD circuit area	%Area overhead
Bus 1	17	2	10	0.034
Bus 2	17	2	10	0.034
Bus 3	6	3	40	0.134
Bus 4	6	5	82	0.275
Bus 5	6	7	93	0.312
Bus 6	6	3	31	0.104

We also performed our simulations on some circuits from ISCAS 85 and ISCAS 89 benchmarks by randomly adding tri-state drivers at some of their output lines. The results are not shown here due to space limitations. They are discussed in [16]. To briefly summarize the results, although the tri-states were added at the outputs, which means that the logic cones are the largest possible, the area overhead for the ISD-Trace circuit was very low for most of the circuits (1 – 7%) area overhead. There were two cases where the ISD-Trace circuit area was large. For those cases, the ISD-Counter circuit had much lower area overhead. This means that the ISD-Counter method and the ISD-Trace method complement each other.

5. Conclusions

Resolving the illegal states in digital circuits has existed as an obstacle for pseudo-random testing of such circuits for a long time. It has been mostly dealt with using static decoding, which sacrifices the fault coverage and adds an additional level of logic that may not be needed.

The ISD technique is more generally applicable than previous techniques and it does not compromise the fault coverage. The ISD technique can be implemented without changing the given design and is hence, non-intrusive.

The ISD technique is a very low area and delay overhead technique for fixing the illegal states that can occur in pseudo-random testing. The ISD technique can be used not only during IC production test, but also during board-level or system-level tests when arbitrary test sequences are applied. Our technique guarantees correct operation under any patterns.

The techniques presented in this paper are applicable to any circuit with constraints on the values a set of nodes can take. Furthermore, they can be combined with any technique for improving fault coverage or reducing test length in cases of pseudo-random testing.

References

[1] McCluskey, E.J., *Logic Design Principle*s *with Emphasis on Testable Semicustom Circuit*s, Prentice-Hall, Englewood Cliffs, NJ, USA, 1986.

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

[3] Hetherington, G., *et al*., "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," *Proc. International Test Conference*, pp. 358 –367, 1999.

[4], Raina, R., *et al*., "DFT Advances in Motorola's Next-Generation 74xx PowerPC™ Microprocessor," *Proc. International Test Conf*., pp. 131 –140, 2000.

[5] Abadir, M., and R. Raina, "Design-for-test Methodology for Motorola PowerPC™ Microprocessors," *Proc. International Test Conference*, pp. 810 –819, 1999.

[6] Scan Synthesis User Guide, Synopsys (1997.08).

[7] Levitt, M., *et. a*l., "Testability, Debuggability, and Manufacturability Features of the UltraSPARC TM –I Microprocessor," *Proc. International Test Conference*, pp. 157-166, 1995.

[8] Fleming; L. O. and Walther; John S., "Local tristate control circuit", US Patent 5136185, Aug 1992.

[9] Mitra, S., L. J. Avra and E. J. McCluskey, "Scan Synthesis for One-hot Signals," *Proc. International Test Conference,* pp. 714-722, 1997.

[10] Das Gupta, S., R. G. Walther, T. W. Williams and E. B. Eichelberger, "An Enhancement to LSSD and some Applications of LSSD in Reliability, Availability and Serviceability," *Proc. International Symposium on Fault-Tolerant Computing*, pp. 32-34, 1981.

[11] Pateras, S. and M. S. Schmookler, "Avoiding Unknown States when Scanning Mutually Exclusive Latches*," Proc. International Test Conference,* pp. 311-318, 1995.

[12] Pyron, C., *et al*., "DFT Advances in the Motorola's MPC7400, a PowerPC™ G4 Microprocessor," *Proc. International Test Conference*, pp. 137 –146, 1999.

[13] Chiang, C.-H., S. Gupta, "BIST TPGs for Faults in Board Level Interconnect via Boundary Scan," *Proc. VLSI Test Symposium*, pp. 376-382, 1997.

[14] Dostie, *Design for at-speed test, diagnosis and measurement*, Kluwer publishers, Boston, MA, 2000.

[15] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html, Benchmark Suite for ITC 1999.

[16] Al-Yamani, Ahmad, S. Mitra, and E.J. McCluskey, "Testing Digital Circuits with Constraints", *CRC Technical Report 02-2*, 2002.

Appendix B

Built-In Reseeding for Serial BIST

Proceedings of IEEE VLSI Test Symposium, pp. 63-68, 2003.

Built-In Reseeding For Serial BIST

Ahmad A. Al-Yamani and Edward J. McCluskey
Center for Reliable Computing
Stanford University, Stanford, CA
{aaa, ejm}@crc.stanford.edu

Abstract

Reseeding is used to improve fault coverage in BIST pseudo-random testing. Most of the work done on reseeding is based on storing the seeds in an external tester. Besides its high cost, testing using automatic test equipment (ATE) makes it hard to test the circuit while in the system. In this paper, we present a technique for built-in reseeding. Our technique requires no storage for the seeds. The seeds are encoded in hardware. The seeds we use are deterministic so 100% fault coverage can be achieved. Our technique causes no performance overhead and does not change the original circuit under test. Also, the technique we present is applicable for transition faults as well as single-stuck-at faults. Built-in reseeding is based on expanding every seed to as many ATPG patterns as possible. This is different from many existing reseeding techniques that expand every seed into a single ATPG pattern. This paper presents the built-in reseeding algorithm together with a hardware synthesis algorithm and implementation.

1. Introduction

An advantage of built-in self-test (BIST) is its low cost compared to external testing using automatic test equipment (ATE). In BIST, on-chip circuitry is included to provide test vectors and to analyze output responses. One possible approach for BIST is pseudo-random testing using a linear feedback shift register (LFSR) [McCluskey 85]. Among the other advantages of BIST is its applicability while the circuit is in the system.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [Eichelberger 83]. The r.p.r. faults are faults with low detectability (few patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques can be classified as: (1) Modifying the circuit under test (CUT) by test point insertion [Eichelberger 83, Touba

96] or by redesigning the CUT, (2) *Weighted pseudo-random patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89, Wunderlich 90] and (3) *Mixed-mode testing* where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [Koenemann 91, Hellebrand 95, Touba 00]. We present a mixed mode technique based on inserting deterministic patterns between the pseudo-random patterns.

Modifying the CUT is often not desirable because of performance issues or intellectual property reasons. Weighted pseudo-random sequences require multiple weight sets that are typically stored on chip. Mixed mode testing is done in several ways; one way is to apply the deterministic patterns from an external tester or store them in an on-chip ROM. Additional circuitry is required to apply the patterns in the ROM to the circuit under test. Instead of storing patterns, seeds can be stored on the tester or in the on-chip ROM. These seeds are transferred into the LFSR and then expanded into the scan chains. This technique does not eliminate the need for the circuitry that transfers the seeds from the ROM to the LFSR.

Another technique for mixed-mode testing uses *mapping logic* [Touba 95]. The strategy is to identify patterns in the pseudorandom sequence that don't detect any new faults and map them by hardware into deterministic patterns.

In this paper, we present a technique that combines mapping logic and *reseeding* (loading the LFSR with a new seed). Our technique uses a simple circuit to identify the states at which the LFSR is to be reseeded. It uses minimal additional hardware to choose the new seed. Our technique utilizes the LFSR flip-flops for storing the seeds. The output of the circuit that detects when to reseed is used to pick the new seed by inverting the flip-flops in which the seed is different from the current contents of the LFSR.

Touba's mapping logic alters the outputs of the pseudo-random pattern generator (LFSR) to insert deterministic patterns into the test set. On the other hand, our technique alters the contents of the LFSR to insert seeds, which produce the deterministic patterns.

Our technique can achieve 100% fault coverage while eliminating the need for external testing or for a ROM to store the seeds. Furthermore, it requires fewer seeds to be encoded compared to previous work. Actually, at best, our technique causes an order of magnitude reduction in the number of seeds to be encoded compared to previous work and even in its worst case it needs fewer seeds than previous work. With a small modification, our technique can be applied for transition faults rather than single stuck faults. Even if it's more desirable to apply the deterministic patterns externally than to add the mapping logic, our technique is still applicable and it reduces the seed storage because of the pseudorandom patterns applied between the seeds. The price for this storage reduction is paid in test length.

Our contributions in this paper are summarized as follows: (1) A reseeding technique that eliminates completely the need for external pattern storage or an on-chip ROM. It's based on encoding the seeds in hardware and using special hardware for the LFSR. The technique can be used for both transition faults as well as single-stuck-at faults. (2) A hardware implementation for the given technique. (3) A reseeding algorithm based on the hardware implementation explained in Sec. 3. The algorithm allows the user to trade off test length for hardware overhead.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we explain the reseeding circuitry implementation. Section 4 discusses the reseeding algorithm. Section 5 shows the simulation results and Sec. 6 concludes the paper.

2. Related work

In serial BIST (aka test per scan), deterministic patterns are encoded into smaller vectors (aka seeds) that are loaded into the LFSR and then expanded into the desired patterns in the scan chains. The patterns are encoded into seeds by solving a linear system of equations, which is an algebraic representation of the linear expansion of the LFSR into the scan chain flip-flops. There are some linear dependencies between some flip-flops of the scan chain. Due to these dependencies, solving the linear system of equations may not always be possible.

2.1 Seed calculation and seed storage

Based on the statistical treatment of the linear dependencies in [Bardell 87], Konemann presented a technique for coding test patterns into LFSRs of size Smax+20, where Smax is the maximum number of specified bits in the ATPG patterns. By adding 20 bits to Smax as the size of the LFSR, the probability of linear dependence drops to 1 in a million [Koenemann 91].

[Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds together with a multiple polynomial LFSR. The technique reuses part of the scan chain flip-flops in expanding the seeds.

In [Krishna 01], a new form of reseeding was described for high encoding efficiency. By making use of the degrees of freedom in solving the linear system of equations the paper achieves higher encoding efficiency than static reseeding.

The technique presented in this paper encodes the seeds in hardware instead of storing them in a ROM or in the tester. Other than the circuit needed to detect when to reseed, minimal hardware is needed to load the desired seeds. Also, our technique is orthogonal to all of the above techniques; i.e., it can be combined with partial dynamic reseeding for a high encoding efficiency or with Rajski's technique that utilizes part of the scan chain.

2.2 Hardware-based reseeding

[Savir 90] presented a reseeding scheme that requires having shadow flip-flops for the LFSR flips-flops. The shadow flip-flops contain the next seed. These shadow flip-flops are loaded with the XOR of the old shadow contents and the original LFSR flip-flops contents. This way, the new seed is expected to be far in the sequence from the current contents of the LFSR. Kim presented a method for generating non-successive pseudo-random test patterns by cascading the LFSR with the scan chain and including a feedback from the scan-out signal into the LFSR [Kim 96]. In [Crouch 95], a self re-

seeding LFSR was presented. Again the LFSR is loaded with a random seed every time a pattern is repeated in part of the LFSR.

The above schemes have the advantage of diversity of the sequences from which the patterns are drawn. They also have the advantage of not requiring seed storage. However, they use seeds that don't particularly target undetected faults. Our technique is based on deterministic seeds which expand into ATPG patterns so 100% fault coverage can be achieved. We pay the price in hardware overhead.

2.3 Mapping logic

Touba and McCluskey came up with an innovative approach for applying deterministic patterns through mapping logic [Touba 95]. In their technique, random patterns that don't detect r.p.r. faults are mapped to ATPG generated cubes through combinational logic. The mapping is performed in two phases, the source patterns are identified in the first phase, and the ATPG cubes are loaded in the second phase. Several iterative minimization heuristics are applied to reduce the area overhead of the mapping logic.

Our technique is a generalization of mapping logic. It has the following advantages: (1) Mapping logic needs hardware for all patterns. In built-in reseeding the LFSR runs in autonomous mode after loading every seed detecting more r.p.r. faults without having to perform more mappings. Because of this, some patterns may not be required. (2) In mapping logic, we need logic for detecting the pattern to be mapped and more logic to perform the mapping. On the other hand, in our technique we only need the logic that detects the patterns that need to be mapped. Enforcing the new values in the LFSR is done utilizing the current contents of the flip-flops of the LFSR.

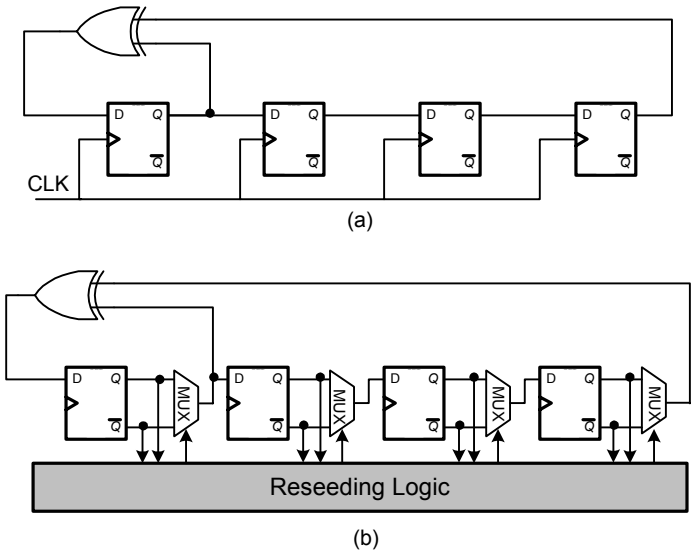


Figure B.1 Reseeding circuit connection to LFSR:
(a) A standard 4-stage LFSR (b) 4-stage LFSR with reseeding circuit.

3. Reseeding circuitry implementation

The operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for some time according to the algorithm described in Sec. 4. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the LFSR in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the LFSR.

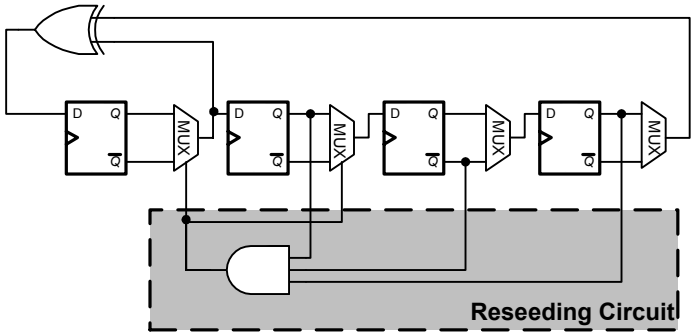
Figure B.1(b) shows the structure of the LFSR and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops as shown in the figure. These flip-flops are just like the scan chain flip-flops. By activating the select line of a given mux, the logic value in the corresponding LFSR stage is inverted.

Cycle	Q1	Q2	Q3	Q4	Cycle	Q1	Q2	Q3	Q4
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

End of Sequence (EOS) = c6 = 0101
Seed = 0100 = c12
Select Lines Activated = (c6) XOR (c11)
= 0101 XOR 1001
= 1100

=> Select lines of Q1 and Q2 activated

(a)



(b)

Figure B.2 Example reseeding circuit: (a) select lines computation (b) hardware implementation.

Definition: The *end of sequence* (*EOS*) contents are the values of the LFSR flip-flops before reseeding. The output of the reseeding circuit activates the select lines of the muxes to invert certain stages of the LFSR such that the desired seed is loaded in the next clock cycle.

As seen in the figure, the only modification to the LFSR compared to a regular LFSR is that the LFSR flip-flops are replaced by muxed flip-flops just like the scan chain.

Let's turn our attention to the reseeding circuit by looking at the following example. Figure B.2 is an example using a 4-stage *self-reseeding LFSR* (LFSR with reseeding logic) with a primitive polynomial. The table in part (a) shows the full sequence of the regular LFSR. Assume that we want to reseed after the 6^{th} cycle (c6). The reseeding circuit needs to be an AND gate that takes as inputs the contents of the LFSR at c6. So in the example the input to the reseeding AND is $\overline{Q}_1 Q_2 \overline{Q}_3 Q_4$. All the cycles that are not part of the desired sequence can be used to minimize the reseeding circuit.

As an example, let the seed be 0100 (c12); we can easily calculate c11 given the polynomial of the LFSR (c11 = 1001). The reason we calculate c11 and not c12 is because we want the seed to be loaded into the LFSR in the next clock cycle. XORing c6 with c11 yields 1100 which means that the output of the reseeding AND should activate the select lines of the MUXes of Q_1 and Q_2. The truth table for the reseeding circuit is shown in Table B.1, where Q_i comes from the output of the i^{th} stage and S_j is the select line of the mux of the j^{th} stage. The patterns between c6 and c11 will not occur so they are don't cares. All the other patterns don't activate any muxes.

The resulting circuit for the example in Figure B.2 is a 3-input AND as shown in the figure.

As more seeds are required, every select line will be a function of the end-of-sequence patterns that will activate it to complement the contents of its corresponding flip-flop. We can then optimize the circuit for that select line and multiple-output optimization can be done for all select lines.

Table B.1 Table of Combinations for the reseeding circuit example.

$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$
1 0 0 0	0 0 0 0	1 1 0 1	d d d d
1 1 0 0	0 0 0 0	0 1 1 0	d d d d
1 1 1 0	0 0 0 0	0 0 1 1	d d d d
1 1 1 1	0 0 0 0	1 0 0 1	0 0 0 0
0 1 1 1	0 0 0 0	0 1 0 0	0 0 0 0
1 0 1 1	0 0 0 0	0 0 1 0	0 0 0 0
0 1 0 1	1 1 0 0	0 0 0 1	0 0 0 0
1 0 1 0	d d d d	0 0 0 0	d d d d

There are two ways to apply transition fault test sets to circuits with scan chains. One way is to use pairs of clock pulses (launch on capture). Once the 1^{st} pattern is loaded into the scan chain, a clock pulse is applied so the response of the combinational logic to the 1^{st} pattern is latched into the flip-flops. Another clock pulse is then applied such that the response of the combinational logic to the 1^{st} pattern is used as the 2^{nd} pattern in the transition fault pair. The other way is to load the scan chain with the two successive patterns (launch on shift). Once the first pattern is applied, the contents of the scan chain

are shifted, the 2nd pattern is applied and the results are captured in the scan chain to be shifted out.

We use the first technique (launch on capture) for transition faults. The reseeding circuit only needs to load the LFSR with the 1st pattern because the 2nd pattern is the response of the logic to the 1st pattern. The reseeding circuit is synthesized such that it changes the contents of the LFSR from the current values to the 1st pattern in the transition fault pattern pair. This means that our technique requires no extra hardware to apply it to transition faults. Although pairs of 2 vectors are required for transition faults, only the first pattern needs to be encoded in hardware because the 2nd pattern is the circuit's response to the 1st pattern.

Figure B.3 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller. The pattern counter is part of the LBIST controller and it is used to count the patterns applied to the circuit under test (CUT).

Although the experiments are given for a single chain per circuit, the technique we present is directly applicable with multiple chains with any phase shifters.

In a BIST environment, where the LFSR is known in advance and the initial seed and test length are also known, the reseeding circuit may take its inputs from the pattern counter instead of the LFSR contents. The dashed line in Figure B.3 corresponds to the reseeding circuit taking its inputs from the pattern counter. If a set of test length *TL* is applied to the circuit the pattern counter will be of size $\lceil \log_2(TL) \rceil$. If the size of the pattern counter is much smaller than the LFSR, this can lead to large reduction in the complexity of the reseeding circuit because the number of inputs is reduced.

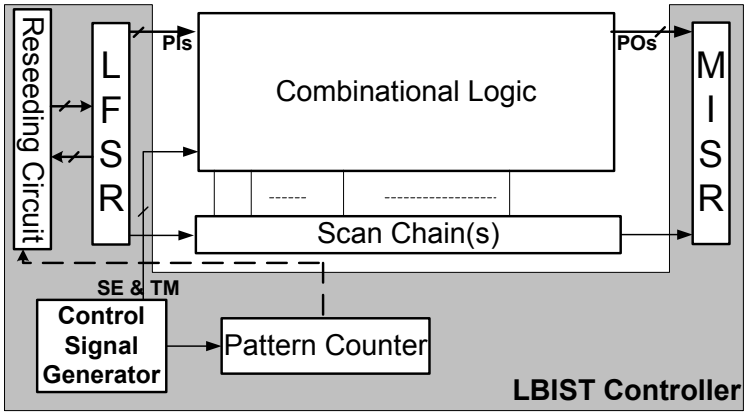


Figure B.3 Reseeding circuit in a system view of BIST environment.

4. Reseeding algorithm

The algorithm we present is based on the following strategies: (1) Generate ATPG patterns for faults that were not detected with pseudo-random patterns and calculate the seeds for these patterns, (2) When a seed is loaded into the LFSR, let the LFSR run in autonomous mode for sometime because there is a chance that some pseudo-random patterns will drop more faults so that some of the ATPG patterns are not needed. This may not be a wise idea if the seeds are applied from a tester because applying the pseudorandom patterns after every deterministic pattern takes extra time on the tester where test time may be expensive. On the other hand, if the seeds are stored or coded on-chip, as it is the case in this work, then it is definitely worth it to minimize the number of seeds that need to be loaded for the same coverage. This will directly minimize the area overhead on the chip, (3) As long as the pseudo-random patterns are detecting more faults, the LFSR should continue in pseudorandom mode. When the pseudo-random patterns become ineffective, the LFSR should be loaded with a new pattern. How to quantify the effectiveness of pseudo-random patterns? The answer is in the next paragraphs.

Definition: *coverage improvement threshold* (CIT) is the improvement in fault coverage required by the algorithm to continue applying pseudorandom patterns. It's a parameter to quantify the effectiveness of pseudorandom patterns. As long as applying

more pseudo-random patterns improves the coverage by at least CIT%, the pseudo-random patterns will be considered effective. When the improvement falls below CIT%, the pseudo-random patterns are considered ineffective. We need to determine how many patterns are simulated before measuring the improvement in coverage. **Definition:** We define the *step size* as the number of patterns simulated before measuring the improvement in coverage. In our simulations, we used different values for the step size.

The only user-specified parameters for this algorithm are the coverage improvement threshold (CIT) and the step size. At one extreme, choosing a very small CIT, means that the user prefers to stick to pseudo-random patterns as long as they have any improvement in the coverage. This in turn means the user wants low hardware overhead for the reseeding circuit. In return for that, the user is willing to have a long test length. At the other extreme, if the user specifies a very high CIT, it means that he has enough area on the chip for the reseeding circuit.

Reseeding based on CIT is one way to choose when to reseed the LFSR. Many other strategies can be used for selecting the reseeding cycles. One way is to choose a fixed length for running the LFSR in autonomous mode after the seed is loaded.

In a sense, our built-in reseeding algorithm is a generalization of mapping logic. If we choose not to continue running the LFSR in pseudorandom mode after reseeding and we choose to alter the LFSR output of the LFSR contents, then our technique becomes similar to mapping logic. We chose to expand the seeds into many patterns since this should reduce the number of seeds to be loaded and accordingly reduce the area of the reseeding circuit. We also chose to alter the contents of the LFSR to enable a different sequence of pseudorandom patterns to drop more faults.

For some circuits, all we need to catch the undetected faults is to take the LFSR to another location in the pattern space. In that case, one or two seeds are enough. For other circuits, the undetected faults require many seeds to be loaded. In that case, our technique will require many seeds and have a long test length.

In case of transition faults, the only change to the algorithm is that fault simulation and ATPG should be done for transition faults instead of SSFs.

5. Simulation results

We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table B.2. The table shows the number of primary inputs, primary outputs and flip-flops in the circuits. It also shows the size of the LFSR used. The last column shows the cell-area of the circuits in LSI library cells area units. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μ technology library.

5.1 Comparison with previous work

We performed some simulation experiments to compare our built-in reseeding with previous work. Most of the previous work assumes one seed per pattern. In mapping logic, hardware is needed to map each pattern but no pattern or seed storage is needed. From here on, "previous work" refers to all work that assumes one seed per pattern.

Table B.2 ISCAS 89 CUTs Used in The Experiments.

Circuit Name	PIs	POs	Scan Chain Size	LFSR Size	Area
s1423	17	5	74	50	4,531
s1488	8	19	6	6	3,555
s1494	8	19	6	6	3,563
s5378	35	49	179	61	14,376
s9234	36	39	211	80	25,840
s13207	62	152	638	45	44,255
s15850	77	150	534	150	48,494
s35932	35	320	1728	60	106,198
s38417	28	106	1636	200	120,180
s38584	38	304	1426	200	115,855

We compare the number of seeds that need to be encoded if our built-in reseeding technique is used and the number of seeds that need to be stored or mapped if previous techniques are used. The number of seeds determines the area of the reseeding or mapping circuit. Since further area minimization heuristics can be applied to all techniques, it's fair to compare them in terms of the number of seeds that need to be

mapped or stored for the same coverage. This comparison can be done for all possible combinations of coverage improvement threshold (CIT) and step size, see Sec. 4. Since built-in reseeding may require longer test length than previous techniques, it's fair to show the test length in the comparison. Why should we tolerate this increase in test length? (1) The area is a scarce resource in BIST. (2) The effect of increasing the test length is not as severe with BIST as it is with external testing because BIST is much faster than external testing. (3) Increasing the test length increases the number of pseudo-random patterns that are likely to be effective in catching unmodeled defects. (4) The increase in test length we have is much smaller than that if only pseudorandom patterns were used. The user can minimize the test length for area overhead.

Table B.3 shows a comparison of previous techniques and our reseeding technique. The table is for 100% single-stuck-at fault coverage, 1% CIT (coverage improvement threshold) and 1024 patterns as a step size. In its best case, built-in reseeding reduces the number of seeds required for 100% coverage by an order of magnitude and increases the test length by only a factor of 1.7 compared to using one seed per pattern.

The table shows that there are many cases where reseeding offers a considerable improvement in the number of patterns required to test the circuit and at the same time does not cause a large increase in the test length.

5.2 Area overhead and test length for SSFs

In this section we present the area overhead of our built-in reseeding technique using the SSF model. Table B.4 shows the area overhead of the reseeding circuits for the benchmarks we used. The reseeding circuits given were designed based on 100% SSF fault coverage. The area overhead given in the table is the minimum area overhead we could achieve for 100% coverage after trying multiple CITs and step sizes.

For most of the circuits, the area overhead ranged from 0.05% to 4%. In a few cases we had a large reseeding circuit.

The minimum area overhead was achieved with different values for CIT for different circuits. This is due to the fact that the number of seeds required to be loaded is highly dependent on the order in which seeds are picked. Our algorithm picks the seeds with the objective of minimizing the area overhead so this might have different effects on the test length.

An important conclusion from the above results is that we need an algorithm to efficiently find the best order for loading the seeds such that the area of the reseeding circuit is minimized. This is investigated in a different paper and results are available in [Alyamani 03].

Table B.3 Comparison of Built-In Reseeding and Previous Work Encoding One Seed per Pattern.

Circuit	Number of seeds to be encoded			Test Length		
	One seed per pattern	Built-in reseeding	%Reduction	One seed per pattern	Built-in reseeding	Degradation factor
s1423	7	4	42.9	2,000	8,160	4.1
s1488	4	1	75.0	3,072	4,096	1.3
s1494	4	1	75.0	3,072	4,096	1.3
s5378	60	20	66.7	3,072	27,648	9.0
s9234	93	62	33.3	5,120	76,800	15.0
s13207	121	26	78.5	5,120	60,416	11.8
s15850	41	38	7.3	4,096	45,056	11.0
s35932	18	1	94.4	3,072	5,120	1.7
s38417	78	62	20.5	5,120	89,088	17.4
s38584	107	36	66.4	4,096	76,800	18.8

Table B.4 Area Overhead and Test Length For Built-In Reseeding (SSFs).

Circuit	Area	Reseeding Area	% Area Overhead	TL
s1423	4,531	113.2	2.5	8,160
s1488	3,555	12	0.3	6,549
s1494	3,563	12	0.3	6,560
s5378	14,376	462	3.2	928
s9234	25,840	3566	13.8	4,544
s13207	44,255	122	0.3	1,600
s15850	48,494	1988	4.1	2,432
s35932	106,198	52	0.05	6,144
s38417	120,180	9418	7.8	6,016
s38584	115,855	1760	1.5	61,440

5.3 Area overhead and test length for transition faults

In this section we present the area overhead of our built-in reseeding technique using the transition fault model. The reseeding circuits given were designed based on the maximum achievable transition fault coverage. For most of the cases, the area overhead ranged from 0.2% to 5% as shown in Table B.5.

In most cases, the minimum area overhead is achieved with different step sizes. This is again due to the fact that the number of seeds that need to be loaded is highly dependent on the order in which seeds are picked.

In general the area overhead of the built-in reseeding circuit for transition faults was close to that for SSFs. The reason is that we only encode the 1st patterns of the transitions patterns pairs in the reseeding circuit. We measure the test length of transition fault test sets in pairs of patterns.

Table B.5 Area Overhead and Test Length For Built-In Reseeding (Transition Faults).

Circuit	Area	Reseeding Area	% Area Overhead	TL
s1423	4,531	205.2	4.5	18,432
s1488	3,555	35.2	1.0	3,072
s1494	3,563	45.2	1.3	3,072
s5378	14,376	855.2	5.9	7,680
s9234	25,840	3937.6	15.2	15,904
s13207	44,255	225.2	0.5	7,168
s15850	48,494	2513.6	5.2	7,424
s35932	106,198	174	0.2	1,376
s38417	120,180	3022.8	2.5	34,816
s38584	115,855	2668.4	2.3	33,792

6. Summary and Conclusions

We presented a built-in reseeding scheme based on encoding the seeds in an on-chip reseeding circuit. 100% fault coverage can be achieved with our technique without any external testing. Our built-in reseeding scheme allows the designer to trade-off area overhead for test length in an optimized way.

Our technique uses special hardware for the LFSR such that the reseeding circuitry area overhead is minimized. Also, the technique we presented is directly

applicable to the transition fault model. The simulation experiments show that the average area overhead is less than 4% for 100% SSF as well as transition fault coverage.

We also presented a reseeding algorithm that minimizes the area overhead. The algorithm takes care of seed selection and reseeding cycle selection. The simulation results show that, for most of the cases, the test length doesn't have to be maximized when the area overhead is minimized, which is a double win for our technique.

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517.

References

[Alyamani 03] Alyamani, A., S. Mitra and E. J. McCluskey, "BIST Reseeding with Very Few Seeds" VLSI Test Symposium, Apr. 2003.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, "Built-In Test for VLSI", John Wiley, New York, 1987.

[Crouch 95] Crouch, Alfred, and M. D. Pressly, "Self Re-seeding Linear Feedback Shift Register Data Processing System for Generating a Pseudo-random Test Bit Stream and Method of Operation," US Patent 5,383,143, Jan. 1995.

[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", IBM Journal of Research and Development, Vol. 27, No. 3, pp. 265-272, May 1983.

[Eichelberger 89] Eichelberger, E. B., E. Lindbloom, F. Motica, and J. Waicukauski, "Weigted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 89.

[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Transactions on Computers, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Kim 96] Kim, Kee, “Scan-Based Built-In Self Test (BIST) with Automatic Reseeding of Pattern Generator,” US Patent 5,574,733, Nov. 1996.

[Koenemann 91] Koenemann, B., “LFSR-Coded Test Patterns for Scan Designs,” Proc. of ETC, pp. 237-242, 1991.

[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, “Test Vector Encoding Using Partial LFSR Reseeding” Proc. of International Test Conference, pp. 885-893, 2001.

[McCluskey 85] McCluskey, E.J., “Built-In Self-Test Techniques,” IEEE Des. & Test of Comp., pp. 21-28, Apr. 85.

[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , “Test Data Decompression for Multiple Scan Designs with Boundary Scan,” IEEE Transactions on Computers, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[Savir 90] Savir, J., and W. McAnney, “A Multiple Seed Linear Feedback Shift Register,” Proc. of International Test Conference, pp. 657-659, 1990.

[Touba 95] Touba, N.A., and E.J. McCluskey, “Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST,” Proc. of International Test Conference, pp. 674-682, 1995.

[Touba 96] Touba, N.A., and E.J. McCluskey, “Test Point Insertion Based on Path Tracing,” Proc. of VLSI Test Symposium, pp. 2-8, 1996.

[Touba 00] Touba, N. and E.J. McCluskey, “Altering Bit Sequence to Contain Predetermined Patterns,” US Patent 6,061,818, May, 2000.

[Wunderlich 90] Wunderlich, H.-J., “Multiple Distributions for Biased Random Test Patterns,” IEEE Transactions on Computer-Aided Design, Vol. 9, No. 6, pp.584-593, Jun. 1990.

Appendix C

BIST Reseeding with Very Few Seeds

Proceedings of IEEE VLSI Test Symposium, pp. 69-74, 2003

BIST Reseeding With Very Few Seeds

Ahmad A. Al-Yamani[1], Subhasish Mitra[2] and Edward J. McCluskey[1]

[1]*Center for Reliable Computing*
Stanford University, Stanford , CA
{aaa, ejm}@crc.stanford.edu

[2]*Intel Corporation*
Sacramento, CA
subhasish.mitra@intel.com

Abstract

Reseeding is used to improve fault coverage of pseudo-random testing. The seed corresponds to the initial state of the LFSR before filling the scan chain. The number of deterministic seeds required is directly proportional to the tester storage or hardware overhead requirement. In this paper, we present an algorithm for seed ordering to minimize the number of seeds required to cover a set of deterministic test patterns. Our technique is applicable whether seeds are loaded from the tester or encoded on chip. Simulations show that, when compared to random ordering, the technique reduces seed storage or hardware overhead by up to 80%. The seeds we use are deterministic so 100% SSF fault coverage can be achieved. Also, the technique we present is fault-model independent.

1. Introduction

An advantage of built-in self-test (BIST) is its low cost compared to external testing using automatic test equipment (ATE). In BIST, on-chip circuitry is included to provide test vectors and to analyze output responses. One possible approach for BIST is pseudo-random testing using a linear feedback shift register (LFSR) [McCluskey 85]. Among the other advantages of BIST is its applicability while the circuit is in the field.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [Eichelberger 83].The r.p.r. faults are faults with low detectability (Few patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques can be classified as: (1) Modifying the circuit under test (CUT) by test point insertion or by redesigning the CUT [Eichelberger 83, Touba 96], (2) Weighted pseudo-random patterns, where the

random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89, Wunderlich 90] and (3) Mixed-mode testing (aka top-off) where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [Koenemann 91, Hellebrand 95, Touba 00]. The technique we present is a mixed mode technique based on inserting deterministic patterns between the pseudo-random patterns.

Modifying the CUT is often not possible due to performance restrictions or intellectual property reasons. Weighted pseudo-random sequences require multiple weight sets that are typically stored on-chip. Mixed mode testing is done in several ways; one is to apply deterministic test patterns from a tester. Another technique is to store the deterministic patterns (or the seeds) in an on-chip ROM. There needs to be additional circuitry to apply the patterns in the ROM to the circuit under test.

Another mixed-mode technique is mapping logic [Touba 95]. The strategy is to identify patterns in the original set that don't detect new faults and map them by hardware into deterministic patterns.

Reseeding refers to loading the LFSR with a seed that expands into a precomputed test pattern. We present a technique for built-in reseeding (encoding the seeds in hardware) in [Alyamani 03]. The technique combines mapping logic and reseeding, and is based on running the LFSR in pseudorandom mode after every seed reload because there is a chance more faults will be detected.

Our contributions in this paper are: a seed ordering algorithm that minimizes the number of seed loads. The algorithm is based on exploiting the algebraic properties of the LFSR. The previous work in [Koenemann 91, Hellebrand 95, Rajski 98 and Krishna 01] and many others assume one seed per pattern. Our technique increases the number of patterns generated from one seed significantly, and hence reduces the hardware required for encoding the seeds. Previous algorithms for embedding multiple patterns into a single-seed sequence [Lempel 95 and Fagot 99] have much higher computational complexity and are impractical for reasonable size circuits. Our algorithm is fault model independent.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we present the seed ordering algorithm. In Sec. 4, we present our seed calculation algorithm. In Sec. 5, we summarize our built-in reseeding scheme. Section 6 shows the simulation results and Sec. 7 concludes the paper.

2. Related Work

Konemann presented a technique for coding test patterns into PRPGs of size S_{max}+20, where S_{max} is the maximum number of specified bits in the ATPG patterns. By adding 20 to S_{max} as the size of the PRPG, the probability that test patterns with $S \leq S_{max}$ specified bits cannot be coded into seeds drops to 1 in a million [Koenemann 91]. In [Hellebrand 95], a scheme was presented for using Multiple-polynomial LFSRs (MP-LFSRs). The authors used MP-LFSRs to reduce the probability of linear dependence.

[Rajski 98] presented a reseeding technique that improves the encoding efficiency by using variable-length seeds together with an MP-LFSR. In [Krishna 01], the authors presented a scheme where the contents of the LFSR are incrementally modified instead of modifying them all at once. The technique achieves higher encoding efficiency than regular reseeding.

The above schemes assume that seeds are either applied from an external tester or stored in an on-chip ROM. They also encode a single seed per test pattern. The technique in [Alyamani 03] presents a scheme to encode the seeds in hardware compared to storing them. By doing so, the chip doesn't need external testing and doesn't need a ROM with the associated decoding and loading circuitry. When seeds are encoded in hardware, finding the minimal set of seeds will reduce the hardware needed for the seeds. The technique presented in this paper tries to exploit the algebraic properties of the LFSR and the don't care bits in the patterns to encode the maximum number of patterns per seed. The don't care bit result in additional degrees of freedom in solving the linear system of equations for seeds

In [Lempel 95], an analytical method was presented for computing a single seed for random pattern resistant circuits based on discrete logarithms. The complexity of the

algorithm depends on the number and size of the prime factors of 2^n-1, where n is the LFSR size. In [Fagot 99], a simulation scheme for calculating initial seeds for LFSRs was presented. The scheme is based on simulating several sequences and picking the one that includes the maximum number of ATPG vectors.

In [Koenemann 00], a technique for skipping useless patterns is presented. The technique is based on having a Seed Skip Data Storage (SSDS) inside the tester. Fault simulation is performed to identify the useful (fault dropping) and useless (non fault dropping) sequences of patterns. The SSDS stores a sequence of numbers corresponding to useful and useless sequence lengths. Using additional control logic, the useless patterns are not loaded from the PRPG to the scan chain of the device under test. The SSDS reduces the storage needed for the test patterns. The control logic that skips the useless patterns reduces the test application time.

Koenemann's technique is optimized for reducing test application time. On the other hand, our seed ordering technique is optimized for reducing the number of seed reloads. By reducing the number of reloads, the seed storage is reduced if seeds are stored on or off chip and the area overhead of the reseeding circuitry is reduced if seeds are encoded in hardware as in [Alyamani 03]. In our technique, we don't skip useless patterns for two reasons: (1) If full BIST is assumed, the test application time is not as big a worry as is the case with ATE. (2) Our ELF35 experiment shows the effectiveness of non fault dropping patterns in catching unmodeled faults.

3. Seed Ordering

The BIST architecture we assume is shown in Figure C.1. Our technique is applicable with any number of scan chains and any phase shifter. The results shown in Sec. 6 are for a single scan chain per circuit. However, the only difference if multiple scan chains were used is in seed calculation. The way we calculate the seeds is explained in Sec. 4. The seeds are then either loaded from the tester or encoded in hardware on chip as explained in Sec. 5. If the LFSR runs in pseudorandom mode after loading the seeds, there is a chance that some of the other seeds may not need to be loaded. This is because

their corresponding faults are already covered with the pseudorandom patterns. Relying on randomness may not exploit the benefit of this scheme maximally. For such a scheme to work optimally, we should figure out the order of the seeds that will result in the minimum number of seed loads into the LFSR.

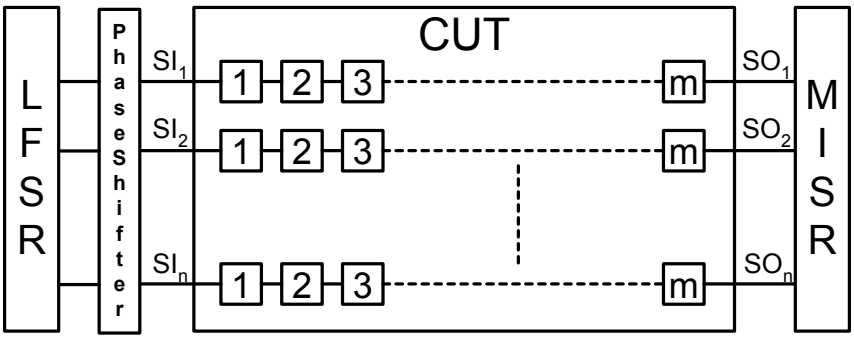


Figure C.1 Multiple scan chains with a phase shifter.

Our seed ordering algorithm starts with pseudorandom patterns to detect the easy faults and then generates deterministic test patterns fort the undetected faults. It randomly picks a deterministic test pattern and encodes it into a seed. We refer to the seed as the *initial state* (s^0) of the LFSR. By running the LFSR for *m* cycles, where *m* is the length of the scan chains, the seed expands into the desired test pattern into the scan chains. We refer to the state of the LFSR after the seed is expanded in the scan chains as the *final state* (s^{m+1}). If the final state of the LFSR can expand into one of the remaining test patterns, then we don't need to load a seed for that pattern.

Let's start with an LFSR whose characteristic polynomial is: $f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0$. Let s^t be the state of the LFSR at time *t*. $s^{t+1} = s^t H$, where *H* is called the transition matrix for the LFSR.

By associativity of matrix multiplication, $s^{t+1} = s^0 H^{t+1}$. If the length of the longest scan chains is *m*, and seed *i* is given by $s^0(i)$, then the contents of the LFSR after the scan chains are loaded is given by $s^{m+1}(i)$.

The ordering algorithm we present is based on looking ahead in the LFSR sequence by finding $s^{m+1}(i)$ for seed *i* and trying to find whether it matches any of the other seeds $s^0(j)$, where $j \neq i$. If they match, then $s^0(j)$ doesn't need to be loaded into the

LFSR. If a match is not found we can search for a match with $s^{d(m+1)}(i)$, where $1 \leq d \leq d_{max}$. The d_{max} is a parameter that corresponds to the number of scan shifts we are willing to continue running the LFSR in pseudorandom mode before loading the next seed.

1. ***P:*** *set of patterns*
2. ***S:*** *set of calculated seeds (empty initially)*
3. *Calculate the seed $s^0(1)$ of pattern 1 and add it to S*
4. ***while*** *(all patterns not covered)*
5. ***for*** *i = 1* ***to*** *dmax*
6. *Find a seed $s^0(k)$ whose final state $s^{d(m+1)}(k)$ can be an initial state for any pattern $p \in P$.*
7. *i = i + 1*
8. ***endfor***
9. ***if*** *$s^0(k)$ is found, P = P – {p}*
10. ***else****, Calculate $s^0(p)$ for p and, S = S + {$s^0(p)$}*
11. ***endwhile***

Algorithm C.1 Seed ordering algorithm.

Given H, we precompute $H^{(m+1)}$, $H^{2(m+1)}$,…, $H^{d(m+1)}$, …. We keep multiplying $H^{d(m+1)}$ by the current seed s^0 to get $s^{d(m+1)}$ until we find a match with one of the other seeds or d exceeds d_{max}. If a match is found, then that seed does not need to be stored or loaded. Computing $s^{d(m+1)}$ from s^0 involves a single matrix multiplication.

The algorithm for matching the final state of the LFSR with a seed for another pattern is explained in Sec. 4.

Table C.1 Example LFSR Sequence.

Cycle	Q1	Q2	Q3	Q4	Cycle	Q1	Q2	Q3	Q4
0	1	0	0	0	8	1	1	0	1
1	1	1	0	0	9	0	1	1	0
2	1	1	1	0	10	0	0	1	1
3	1	1	1	1	11	1	0	0	1
4	0	1	1	1	12	0	1	0	0
5	1	0	1	1	13	0	0	1	0
6	0	1	0	1	14	0	0	0	1
7	1	0	1	0	15	1	0	0	0

Given a set of seeds and a user-specified d_{max}, the best ordering is the ordering that will result in the minimum number of seeds that need to be loaded into the LFSR. To find an ordering that minimizes the seed loads, we organize the patterns in sequences. The sequence includes seeds that occur within d_{max} distance from one another and cover multiple ATPG patterns.

Algorithm C.1 is the seed ordering algorithm. The output is a set of seeds that need to be loaded into the LFSR to expand into the desired patterns. After every seed is loaded, we run the LFSR for d_{max} scan cycles to generate all patterns that this seed can generate within d_{max}. Algorithm C.1 helps in choosing which seeds to load so that we know that other patterns will be covered by just running for d_{max} cycles after every seed.

As an example, take a 4-stage LFSR whose polynomial is $f(x) = x^4 + x^3 + 1$. The LFSR sequence is shown in Table C.1. Assume that the seeds are 0111, 0101, 1001 and 0001, which correspond to cycles 4, 6, 11 and 14 of the LFSR sequence shown. If d_{max} used is 3 then we will need to load S4 and S11. If d_{max} used is 5, we will only need to load S4 only. The example is for parallel BIST (test per clock) for simplicity. For serial BIST (test per scan), we need to include the cycles needed to fill the scan chains into account while finding the match in the ordering algorithm. Those cycles are taken care of by raising the transition matrix H to the power m+1 as shown earlier, where m is the length of the scan chains. This means that we try to match the contents of the LFSR after multiples of m+1.

Algorithm C.1 reorders the seeds to reduce the seeds to be loaded given d_{max}. By reducing the number of seeds, the algorithm reduces the hardware overhead if the seeds are encoded in hardware. If the seeds are loaded from a tester, the algorithm reduces the storage needed.

It's possible to find the best order of the seeds by simulating all possible permutations. However, this is prohibitively time consuming.

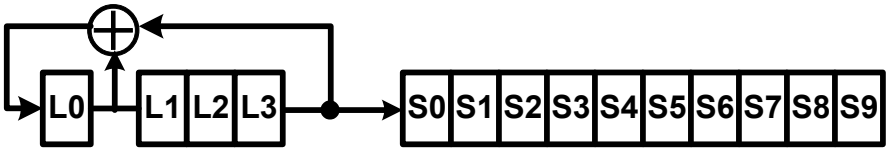


Figure C.2 4-stage LFSR connected to a chain.

4. Seed Calculation

Figure C.2 shows an example for a 4-stage LFSR connected to one scan chain with 10 flip-flops. This figure will be used as an example for illustrating the equation generation technique. We start with a simple example and then we explain the technique for the multiple scan chains.

For every flip-flop in the scan chain, there is a corresponding equation in terms of the bits of the LFSR. Let's label the scan chain flip-flops by $S_0 \rightarrow S_{m-1}$ where m is the size of the scan chain. Also, let's label the stages of the LFSR by $L_0 \rightarrow L_{n-1}$ where n is the size of the LFSR. In the example above, the equations for the n most significant flip-flops of the scan chain are: $S_9 = L_3$, $S_8 = L_2$, $S_7 = L_1$, and $S_6 = L_0$ because after n clock cycles the bits of the seed end up in the most significant bits of the scan chain. The reader is invited to verify the remaining equations:

$S_5 = L_0 \oplus L_3$ $\qquad$ $S_4 = L_0 \oplus L_2 \oplus L_3$

$S_3 = L_0 \oplus L_1 \oplus L_2 \oplus L_3$ $\qquad$ $S_2 = L_1 \oplus L_2 \oplus L_3$

$S_1 = L_0 \oplus L_1 \oplus L_2$ $\qquad$ $S_0 = L_1 \oplus L_3$

We can represent the above equations by an $m \times n$ matrix in which the rows correspond to the scan chain flip-flops and the columns correspond to the LFSR stages. An entry (i,j) is 1 if and only if L_j appears in the equation of S_i. According to this system, the following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \\ S_9 \end{matrix}$$

$$\begin{matrix} L_0 & L_1 & L_2 & L_3 \end{matrix}$$

In the case of multiple scan chains, if the architecture shown in Figure C.2 above is used, there will be undesired correlation between the bits of patterns that are shifted into the scan chains. Because of this correlation, the outputs of the LFSR stages must go through a phase shifter.

The phase shifter makes sure that the sequence fed into chain *j* is apart from the sequence that goes into chain i by at least the depth of chain *i*.

The phase shifter for a given scan chain is a linear sum of some stages from the LFSR. The phase shifter for chain *i* can be represented by:

$p^i = \begin{bmatrix} p^i_{n-1} & p^i_{n-2} & \cdots & p^i_2 & p^i_1 & p^i_0 \end{bmatrix}$. p^i_j is one if the XOR feeding scan chain *i* has the output of stage *j* of the LFSR as one of its inputs.

The algorithm that generates the equations for a scan chain *S* that is fed through a phase shifter p^s starts with the selection vector p^s. The algorithm starts by assigning the selection vector p^s to the last row of *E*. The other rows are generated bottom up by multiplying the transition matrix *H* by the following row of *E*. This algorithm can be used with any phase shifter and any number of chains.

Let's now revisit the concept of matching the final state of the LFSR with a seed for one of the remaining patterns. We are trying to match the final state of the LFSR with

a seed that will expand into one of the ATPG patterns. An ATPG pattern is represented by a set of equations in terms of the LFSR stages. The equations are actually restrictions that need to be satisfied so that the test pattern is generated from a given seed. The equations can be taken directly from the matrix *E* for the corresponding care bits of the test pattern. If the final state of the LFSR (s^{m+1}) satisfies all equations for a given pattern then we know that the final state is a seed for this pattern. Accordingly, we don't have to load an extra seed for that pattern.

In other words, there are degrees of freedom in solving for the seed. The degrees of freedom are caused by the fact that the number of equations is less than the number of unknowns. To solve such a system, we use Gaussian reduction to extract the pivots from the variables. The non-pivot variables can be assigned arbitrary values. The pivot variables are linear combinations of the non-pivot variables, so they are fixed. By assigning all the non-pivot variables in one of the remaining seeds the same values in $s^{(m+1)}$, we increase the chance of finding a match between $s^{(m+1)}$ and that seed.

As an example, for the LFSR shown in Figure C.2, assume that the ATPG tools generates the following two patterns: (X0X1X10XXX, 0XXX1XXXXX). To generate the seed for the 1st pattern, we solve the following system:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} L_0 \\ L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The solution of the system is $s^0(0) = 0111$. The next step is to calculate $s^{(m+1)}(0)$, which is the value the contents of the LFSR after the chain loaded with the pattern. $s^{(m+1)}(0) = s^0(0) \times H^{(m+1)}$. Since $m = 10$, $s^{(m+1)}(0) = 1000$. Now, we solve the following system to generate the seed for the other pattern:

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} L_0 \\ L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

In the above system, we can choose L_0 and L_1 to be the pivots. By doing so, L2 and L3 become free variables that we can assign any arbitrary value to, and L_0 and L_1 are given by:

$L_1 = L_3 \oplus 0$ $\qquad\qquad$ $L_0 = L_2 \oplus L_3 \oplus 1$

So, if we assign 10 to L_2L_3, the second seed will be $s^0(1) = 0010$. However, if we assign 00 to L_2L_3, the second seed will be $s^0(1) = 1000$, which matches $s^{(m+1)}(0)$. This means that the 2nd seed doesn't need to be loaded. We can continue running the LFSR and it will expand to put the 2nd pattern in the scan chain.

5. Built-In Reseeding

In built-in reseeding, the operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for sometime according to the reseeding algorithm [Alyamani 03]. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the LFSR in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the LFSR.

Figure C.3(b) shows the structure of the LFSR and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops as shown in the figure. By activating the select line of a given mux, the logic value in the corresponding LFSR stage is inverted.

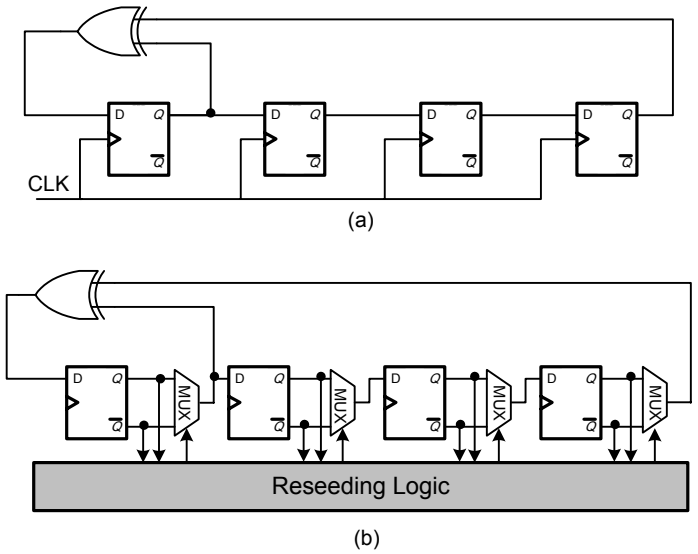


Figure C.3 Reseeding circuit connection to the LFSR:
(a) A standard 4-stage LFSR (b) 4-stage LFSR with reseeding circuit.

Figure C.4 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller, which includes the additional control circuitry added for logic BIST. Complete details about the architecture and synthesis for built-in reseeding are given in [Alyamani 03].

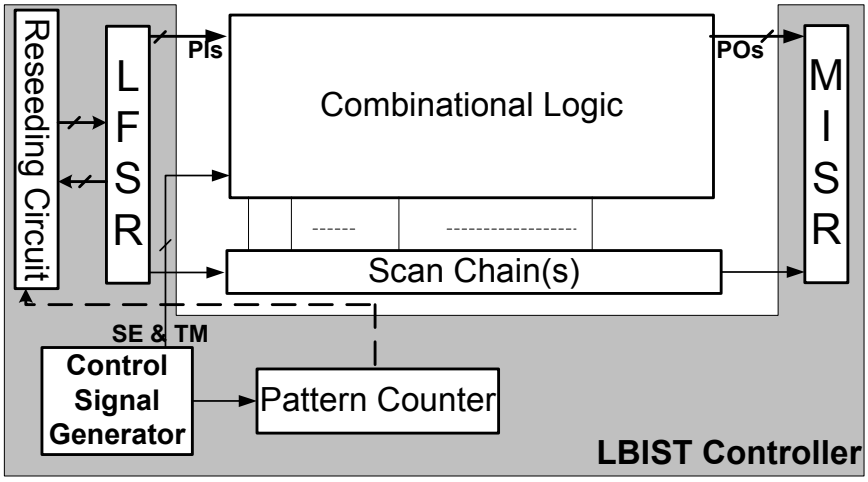


Figure C.4 Reseeding circuit in a system view of BIST environment.

6. Simulation Results

We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table C.2. The table shows the number of primary inputs, primary outputs, flip-flops in the scan chain, and in the LFSR. It also shows the cell-area of the circuits in LSI library cells area units. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μ library.

The experiment was designed such that pseudo random patterns are applied first to detect easy faults. Then, test patterns are generated for the undetected faults and the seeds are calculated for the test patterns.

In [Koenemann 91, Hellebrand 95, Rajski 98, Touba 00, and Krishna 01] a single seed per pattern is assumed. In our technique, we try to generate multiple patterns with the same seed by reordering the seeds such that we need to load the minimal number of seeds into the LFSR.

Table C.2 ISCAS Circuits Used for the Experiments.

Circuit Name	PIs	POs	Scan Chain Size	LFSR Size	Cell Area
s1238	14	14	18	15	2,740
s1423	17	5	74	50	4,531
s1488	8	19	6	6	3,555
s1494	8	19	6	6	3,563
s5378	35	49	179	61	14,377
s9234	36	39	211	80	25,840
s13207	62	152	638	45	44,255
s15850	77	150	534	150	48,494
s38584	38	304	1426	200	115,855

Table C.3 Number of Seeds for Our Technique Compared to Seed per Pattern.

Circuit	Cell Area	Seed/ Pattern	Our Tech.	Red. %
s1238	2,740	30	7	76.7
s1423	4,531	7	3	57.1
s1488	3,555	4	1	75.0
s1494	3,563	4	1	75.0
s5378	14,377	35	7	80.0
s9234	25,840	89	57	36.0
s13207	44,255	74	12	83.8
s15850	48,494	38	35	7.9
s38584	115,855	55	16	70.9

Table C.3 shows the number of test patterns generated for the undetected faults. The seed/pattern column shows the number of seeds that need to be stored or mapped if a single seed per pattern is assumed as it is the case in most of the previous work. The table shows the number of seeds that need to be stored or mapped with our technique. As shown in the table, the reduction varies from 8 to 84%.

Table C.4 shows the test time (given by the number of scan patterns applied) when regular top off is used compared to the test time when our seed ordering technique is used. The test time increases by a factor of 1.3 to 3.5 in all cases.

Table C.4 Test Time Increase for Our Technique Compared to Top Off.

Circuit	Test Time		Test Time Increase (x times)
	Top Off	Our Tech.	
s1238	414	832	2.0
s1423	487	672	1.4
s1488	228	288	1.3
s1494	228	288	1.3
s5378	1507	1920	1.3
s9234	1433	4992	3.5
s13207	1994	2688	1.3
s15850	1702	3904	2.3
s38584	1655	2624	1.6

Table C.5 shows the number of seeds that need to be loaded with three random orderings of the seeds. It also shows the reduction gained by using our ordering compared

to the average number of seeds with random ordering. The reduction varies from 5% to 80%. The three random ordering are: (1) the original order given by the ATPG tool, (2) the reverse order, and (3) taking every other seed (i.e., all odd numbered seeds followed by all even seeds).

Table C.6 shows the area overhead needed for built-in reseeding [Alyamani 03] if the random orderings are used. It also shows the reduction in area overhead gained by using our ordering compared to the average overhead with random ordering. The reduction varies from 7% to 84%.

Table C.5 Number of Seeds Needed by Random Ordering Compared to our Ordering.

Circuit	Cell Area	Number of Seeds					
		Rand1	Rand2	Rand3	Rand.Avg.	Our Technique	Reduction %
s1238	2,740	28	27	28	27.7	7	74.7
s1423	4,531	6	6	6	6.0	3	50.0
s1488	3,555	4	3	4	3.7	1	73.0
s1494	3,563	4	4	3	3.7	1	73.0
s5378	14,377	32	33	32	32.3	7	78.3
s9234	25,840	87	85	87	86.3	57	34.0
s13207	44,255	66	65	64	65.0	12	81.5
s15850	48,494	37	38	36	37.0	35	5.4
s38584	115,855	47	50	48	48.3	16	66.9

Table C.6 Area Overhead Needed by Random Ordering Compared to Our Ordering.

Circuit	Cell Area	Area Overhead %					
		Rand1	Rand2	Rand3	Rand.Avg.	Our Technique	Reduction %
s1238	2,740	39.6	37.0	42.7	39.8	9.9	75.1
s1423	4,531	5.8	4.8	4.6	5.1	2.1	58.8
s1488	3,555	2.1	2.0	1.6	1.9	0.3	84.2
s1494	3,563	2.1	2.0	1.6	1.9	0.3	84.2
s5378	14,377	13.0	16.6	15.6	15.1	3.1	79.5
s9234	25,840	14.4	13.8	14.4	14.2	13.2	7.0
s13207	44,255	2.0	1.8	1.9	1.9	0.3	84.2
s15850	48,494	4.1	3.7	4.2	4.0	3.5	12.5
s38584	115,855	1.5	1.4	1.5	1.5	1.2	20.0

7. Conclusion

We presented a seed ordering technique based on the algebraic properties of the LFSR and exploiting the degrees of freedom in solving the seed equations.

The simulation experiments showed that the area overhead and the storage needed are reduced by up to 80% when our ordering technique is used compared to random ordering. The main characteristics that make our technique effective are: (1) Exploiting the linearity of the LFSR and the associativity of matrix multiplication to avoid simulation. (2) Avoiding unnecessary computation to reduce the complexity of finding the best order of the seeds, and (3) Exploiting the degrees of freedom in solving for the seeds to match them with the current contents of the LFSR.

The big win for our technique is that it provides a solution for the problem while avoiding high computational complexity like previous analytical solutions and avoiding long simulation times like previous simulation techniques.

We also presented an algorithm for generating the system of linear equations used to calculate the seeds for with any LFSR and phase shifter.

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517.

References

[Alyamani 03] Al-Yamani, A., and E. J. McCluskey, "Built-In Reseeding for Built-In Self Test", VLSI Test Symposium, Apr. 2003.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, "Built-In Test for VLSI", John Wiley, New York, 1987.

[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", IBM Journal of Research and Development, Vol. 27, No. 3, pp. 265-272, May 1983.

[Eichelberger 89] Eichelberger, E. B., E. Lindbloom, F. Motica, and J. Waicukauski, "Weighted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 1989.

[Fagot 99] Fagot, C., O. Gascuel, P. Girard and C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test," Proc. of European Test Workshop, pp. 7-14, 1999.

[Hellebrand 95] Hellebrand, S. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Trans. on Comp., Vol.44, No.2, pp. 223-233, Feb. 95.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," Proc. of European Test Conference, pp. 237-242, 1991.

[Koenemann 00] Koenemann, B., "System for Test Data Storage Reduction," US Patent 6,041,429, 2000.

[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding" Proc. of International Test Conference, pp. 885-893, 2001.

[Lempel 95] Lempel, M., S. Gupta and M. Breuer, "Test Embedding with Discrete Logarithms," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 5, pp. 554-566, May 1995.

[McCluskey 85] McCluskey, E.J., "Built-In Self-Test Techniques," IEEE Design & Test of Computers, pp. 21-28, Apr. 1985.

[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , "Test Data Decompression for Multiple Scan Designs with Boundary Scan," IEEE Transactions on Computers, Vol. 47, No. 11, pp. 1188-1200, Nov. 98.

[Touba 95] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," Proc. of ITC, pp. 674-682, 1995.

[Touba 96] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. of VLSI Test Symposium, pp. 2-8, 1996.

[Touba 00] Touba, N. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.

[Wunderlich 90] Wunderlich, H.-J., “Multiple Distributions for Biased Random Test Patterns,” IEEE Transactions on Computer-Aided Design, Vol. 9, No. 6, pp.584-593, Jun. 1990.

Appendix D

Seed Encoding with LFSRs and Cellular Automata

Proceedings of ACM Design Automation Conference, pp. 560-565, 2003.

Seed Encoding with LFSRs and Cellular Automata

Ahmad A. Al-Yamani and Edward J. McCluskey
Center for Reliable Computing
Stanford University, Stanford, CA
{alyamani, ejm}@crc.stanford.ed

Abstract

Reseeding is used to improve fault coverage of pseudo-random testing. The seed corresponds to the initial state of the PRPG before filling the scan chain. In this paper, we present a technique for encoding a given seed by the number of clock cycles that the PRPG needs to run to reach it. This encoding requires many fewer bits than the bits of the seed itself. The cost is the time to reach the intended seed. We reduce this cost using the degrees of freedom (due to don't cares in test patterns) in solving the equations for the seeds. We show results for implementing our technique completely in on-chip hardware and for applying it from a tester. Simulations show that with low hardware overhead, the technique provides 100% single-stuck fault coverage. Also, when compared with conventional reseeding from an external tester or on-chip ROM, the technique reduces seed storage by up to 85%. We show how to apply the technique for both LFSRs and CA.

Categories and Subject Descriptors

B.8.1 [Integrated Circuits]: Reliability, Testing, and Fault Tolerance.
General Terms
Algorithms, Performance, Design, Reliability.
Keywords
VLSI Test, Built-In Self Test, Reseeding.

1. Introduction

Among the advantages of built-in self-test (BIST) are low cost compared to external testing using automatic test equipment (ATE), and applicability while the circuit is in the field. In BIST, on-chip circuitry is included to provide test vectors and to analyze

output responses. One possible approach for BIST is pseudo-random testing using a linear feedback shift register (LFSR) [McCluskey 85, Bardell 87].

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [Eichelberger 83].The r.p.r. faults are faults with low detectability (Few patterns detect them).

Several techniques have been suggested for improving BIST fault coverage. They are: (1) Modifying the circuit by test point insertion or by redesigning the circuit [Eichelberger 83, Touba 96], (2) *Weighted pseudorandom patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89, Wunderlich 90], and (3) *Mixed-mode testing* where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied that target the undetected faults [Koenemann 91, Hellebrand 95, Touba 00]. Our technique is a mixed mode technique based on encoding the seeds in terms of the number of clock cycles required for the PRPG to reach them.

Modifying the CUT is often not possible because of performance issues or intellectual property rights. Weighted pseudo-random sequences require multiple weight sets. Mixed mode testing is done in several ways; one is to apply deterministic test patterns from a tester. Another technique is to store the deterministic patterns (or the seeds) in an on-chip ROM. There needs to be additional circuitry to apply the patterns in the ROM to the circuit under test.

Another technique for mixed-mode testing is *mapping logic* [Touba 00] where non-fault dropping patterns in the original set are mapped by hardware into deterministic patterns.

Reseeding refers to loading the PRPG with a seed that expands into a precomputed test pattern. We presented a technique for *built-in reseeding* (encoding the seeds in hardware) in [Alyamani 03a]. The technique combines mapping logic and reseeding and applies pseudorandom patterns between the deterministic seeds because there is a chance more faults will be detected.

A seed is a PRPG *initial state*. When the PRPG is loaded with this initial state, it loads the scan chains with the desired pattern after *m* clock cycles, where *m* is the length of the scan chains. We call the state of the PRPG after loading the scan chains the *final state*.

In [Alyamani 03b], we presented a technique for minimizing the number of seeds to be loaded by ordering the seeds and by exploiting the degrees of freedom in solving for the seeds. In this paper, our contributions are: (1) A seed encoding technique that encodes the seeds in a much smaller vector that corresponds to the number of cycles to reach it. The technique also exploits the degrees of freedom in solving for the seed. (2) An architecture that implements the encoding technique. The technique is applicable for SSF and transition faults. (3) An improvement over the built-in reseeding and seed ordering techniques to make them valid for any linear machine i.e., for LFSRs or cellular automata with or without phase shifters.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we present the seed encoding scheme. In Sec. 4, we present the architecture for built-in seed encoding and reseeding. Section 5 shows the simulation results and Sec. 6 concludes the paper.

2. Related Work

Konemann presented a technique for coding test patterns into PRPGs of size Smax+20, where Smax is the maximum number of specified bits in the ATPG patterns. By adding 20 to Smax as the size of the PRPG, the probability that test patterns cannot be coded into seeds drops to 1 in a million [Koenemann 91].

[Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds. In [Krishna 01], the authors presented partial dynamic reseeding to incrementally modify the LFSR contents instead of modifying them all at once. This technique achieves higher encoding efficiency than static reseeding. The technique in [Alyamani 03a] encodes the seeds in hardware.

The technique presented in this paper tries to exploit the degrees of freedom in solving the linear system of equations for the seed to encode a seed by the number of additional clocks needed to reach it.

In [Lempel 95], an analytical method was presented for computing seeds for random pattern resistant circuits based on discrete logarithms. In [Fagot 99], a simulation scheme for calculating seeds for LFSRs was presented. The scheme is based on simulating several sequences that include a set of ATPG vectors.

In [Koenemann 00], a technique for skipping useless patterns is presented. The technique is based on having a Seed Skip Data Storage (SSDS) inside the tester. Fault simulation is performed to identify the useful (fault dropping) and useless (non fault dropping) sequences of patterns. Using additional control logic, the useless patterns are not loaded from the PRPG to the scan chains.

3. Seed Encoding

The BIST architecture we assume is shown in Figure D.1. Our technique is applicable with any number of scan chains and any phase shifter. The results shown in Sec. 5 are for a single scan chain per circuit. However, the only difference if multiple scan chains were used is in the seed calculation. The way we calculate the seeds is explained in the appendix. The seeds are then either loaded from the tester or encoded in hardware on chip as explained in Sec. 4.

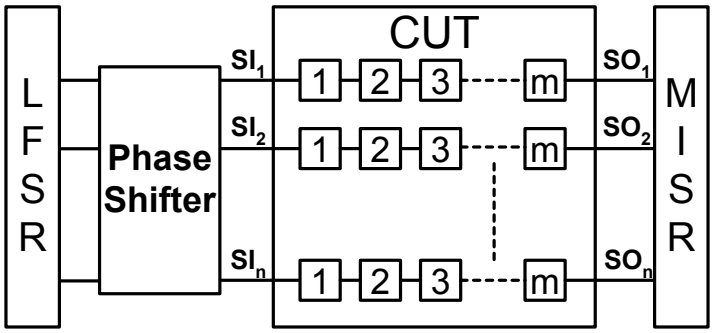


Figure D.1 Multiple scan chains with a phase shifter.

If the final state of the PRPG after loading the scan chain matches another seed, then that seed doesn't have to be loaded into the PRPG. If the final state of the PRPG

doesn't match another seed, we can clock the PRPG a few times until we reach a match with one of the seeds. Also, instead of relying completely on randomness, we can exploit the degrees of freedom in solving the equations to generate the seed so that we increase the chances of matching a seed with the final state of the PRPG.

Let the state of the PRPG at time t be given by $s(t)$, then the PRPG state at time t+1 is given by $s(t+1) = s(t)H$, where H is called the transition matrix, and $s(t+1) = s(0)H^{t+1}$. If the longest scan chain is of length m, and seed i is $s_i(0)$, then the contents of the PRPG after the scan chains are loaded is given by $s_i(m+1)$. The PRPG can be a cellular automaton or an LFSR.

The encoding algorithm we present is based on looking ahead in the sequence of the PRPG by finding $s_i(m+1)$ for seed i and trying to find whether it matches with any of the other seeds $s_j(0)$, where $j \neq i$. If they match, then $s_j(0)$ doesn't need to be loaded into the PRPG. If a match is not found, we can search for a match with $s_i(m+d)$, where $1 \leq d \leq d_{max}$. The parameter d_{max} corresponds to the number of clock cycles we are willing to continue running the PRPG before loading the next seed. Choosing $d_{max} = 1$ means that if $s_i(m+1)$ doesn't match any of the remaining seeds, we will load a new seed.

The technique explained above requires changes to the BIST architecture to allow capturing after a different number of clock cycles. The architecture is presented in Sec. 4.

Given a set of seeds and a user-specified d_{max}, we order the seeds to minimize the number of seeds that need to be loaded as explained in [Alyamani 03b].

4. Seed Encoding Architecture

A fundamental issue in applying our seed encoding technique is how to make the PRPG run for a variable number of cycles for different seeds. Normally, the PRPG, runs for a number of cycles equal to the length of the longest scan chain before a capture cycle. To use our encoding technique, which represents the seed by the number of clock cycles required to reach it, we need to have the PRPG run for a variable number of cycles.

In a usual logic BIST architecture, a *bit counter* is used to choose when to disable the Scan Enable (SE) signal for capturing. One way to implement this is to have the bit counter loaded with the value that corresponds to the length of the longest scan chain for every pattern. The bit counter is then decremented by 1 at each clock cycle. When the bit counter reaches zero, it means that the test pattern is loaded into the scan chains, so SE is disabled for one clock cycle, and so on. The length of the scan chains is stored in a register and loaded into the bit counter with every pattern. Our technique is based on running the PRPG for a number of cycles to reach the desired seed. To implement this, we need to load the bit counter register with different values corresponding to the number of cycles before the next capture. Unloading the scan chains starts right after the capture cycle. So, for encoded seeds, there are extra cycles after unloading the response to pattern i and before capturing the response for pattern i+1.

Running the technique from an ATE

To run our technique from an external tester, we have two types of seeds, seeds that need to be loaded into the scan chain, *loaded seeds*, and seeds that can be reached by continuing to run the PRPG for additional cycles after loading the scan chains, *encoded seeds*. We use the name encoded seeds because these seeds are encoded into the number of cycles the PRPG needs to run to reach them.

Seed size: How efficient is this encoding? Why not just load all seeds? This question can be answered by a simple example, take a circuit of 10,000 flip flops that has 10 scan chains of length 1000 each. If the maximum number of care bits in the test patterns is 500 (5%), we need a PRPG of size 520 [Koenemann 91]. Since the length of the scan chain is 1000, the bit counter needs to have only 10 bits. So, by encoding the seed into the number of cycles to reach it we get a 98% (52×) reduction in seed storage. Even if we decide to run the PRPG for up to 1000 additional cycles before reaching the next desired seed, then impact on the size of the bit counter is a single bit.

Test time: In terms of test length, for the example above, loading the PRPG with a new seed takes 520 cycles. Loading the bit counter register takes 11 clock cycles. This

means that we can search for a match with another seed in up to 508 cycles while saving on the test time and at the same time saving on tester storage.

If we only rely on luck in finding a match by clocking the PRPG, then we may not have a very good chance. That's why we exploit the degrees of freedom in solving the linear system of equations to force such a match as explained in the appendix.

Full BIST Implementation

Our technique can be applied for full on-chip BIST with 100% SSF fault coverage. For that, we need to have a reseeding circuit for loaded seeds and a seed encoding circuit for encoded seeds.

Loaded seeds: We use the built-in reseeding architecture presented in [Alyamani 03a]. The operation of the reseeding circuit is as follows: the PRPG starts running in autonomous mode according to the reseeding algorithm [Alyamani 03a]. Once it is time for reseeding, a seed is loaded into the PRPG, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the PRPG in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the PRPG.

Figure D.2(b) shows the structure of the PRPG and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops. By activating the select line of a given mux, the logic value in the corresponding stage is inverted. The muxed flip-flops are similar to those used for scan chains. The output of the reseeding circuit activates the select lines of the muxes to invert certain stages of the LFSR such that the desired seed is loaded in the next cycle.

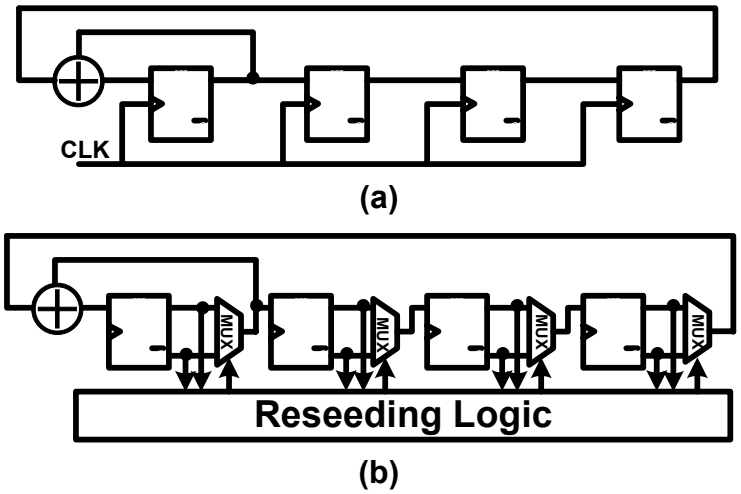


Figure D.2 Reseeding circuit connection to LFSR:
(a) A standard LFSR (b) LFSR with reseeding ckt.

As seen in the figure, the only modification to the LFSR compared to a regular LFSR are the muxes. The LFSR flip-flops are replaced by muxed flip-flops just like the scan chain. In case of cellular automata, the same muxes structure can be used. The muxes should be placed right at the outputs of the flip flops before any XOR gates that are fed by the scan chain flip-flops. This way both polarities are available at the inputs of the muxes. Since XORs are linear gates, their outputs will be complemented by complementing any of the inputs, which satisfies the requirement for the above architecture to work. The connection of the reseeding logic to CA is shown in Figure D.3.

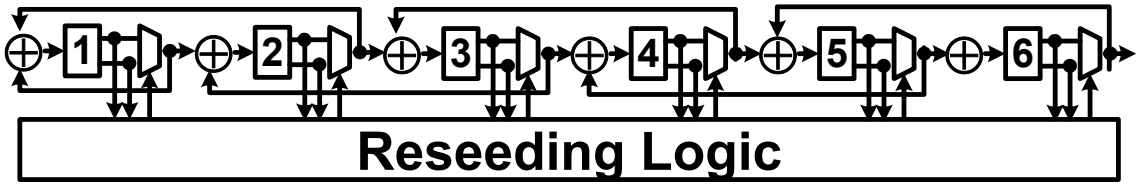


Figure D.3 Reseeding circuit connection to CA.

Encoded seeds: For the encoded seeds, we need a combinational circuit that takes as its input the value of the PRPG; the output of this circuit should be loaded into the bit counter register. In logic BIST architectures, a pattern counter is used to count the patterns applied to the circuit. Instead of reading the values of the PRPG stages as input

to the seed encoding circuit, the value of the pattern counter can be used as input. The majority of the input combinations for the seed encoding circuit will load the bit counter register with the length of the scan chains. The pattern counter input combinations that correspond to seeds to be encoded will load a different value in the bit counter register. That value corresponds to the length of the scan chains plus the number of clock pulses that need to be applied before reaching the desired seed.

We can synthesize the seed encoding circuit from a table of combinations. The input combinations that correspond to normally loaded seeds should have the scan chains length as the output value. The input combinations that correspond to encoded seeds should have the output as the scan chains length in addition to the number of the clock cycles needed to reach to the seed. Just as in the circuit for built-in reseeding, all input combinations that won't occur in the desired test sequence can be treated as don't cares to help minimize the seed encoding circuit.

Figure D.4 shows where the reseeding and seed encoding circuits fit in a system level view of a circuit with an LBIST controller, which includes the additional control circuitry added for logic BIST.

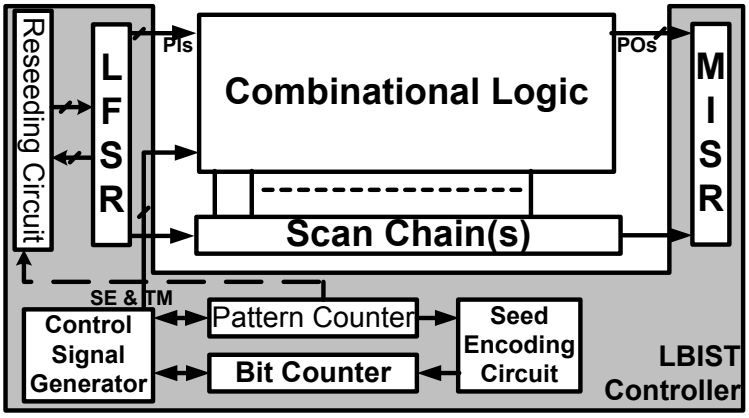


Figure D.4 : Reseeding and seed encoding circuits in a system view of BIST environment.

If the CUT or the scan chain is changed, then the reseeding circuit and the seed encoding circuit need to be re-synthesized based on the new design and the new test patterns. However, if the seed encoding technique is applied from a tester, then changing

the design only results in changes in the test patterns and accordingly the seeds that are stored. If it's preferable to apply the technique with full BIST, then it may be better to apply it after the design is stable and no more changes are applied to the circuit or the scan chain.

5. Simulation Results

In this section we present the results of some simulation experiments to evaluate our seed encoding technique. We performed our experiments on some of the ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table D.1. The table shows the number of primary inputs, primary outputs, flip-flops in the scan chain, and in the LFSR. The BC column lists the sizes of the bit counters. We took into account the maximum number of additional clock cycles to calculate the size of the bit counter. The table also shows the cell-area of the circuits in LSI library cells area units. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 μ technology library.

The experiment was designed such that pseudo random patterns are applied first. Then, test patterns are generated for the undetected faults and the seeds are calculated from the test patterns.

Table D.1 ISCAS 89 Circuits Used in Experiments.

CUT Name	PIs	POs	FFs	LFSR	BC	Cell Area
s953	16	23	29	21	7	2,286
s1196	14	14	18	15	6	2,722
s1238	14	14	18	15	6	2,740
s1423	17	5	74	50	8	4,531
s1488	8	19	6	6	6	3,555
s1494	8	19	6	6	6	3,563
s5378	35	49	179	61	9	14,377
s9234	36	39	211	80	10	25,840
s13207	62	152	638	45	10	44,255
s35932	35	320	1728	60	11	106,198

In [Koenemann 91, Hellebrand 95, Touba 00, Rajski 98, and Krishna 01] a single seed per pattern is assumed. In our technique, we encode as many of the seeds as we can by the number of additional clock cycles to reach them. The remaining seeds have to be loaded from the tester or encoded on-chip. Table D.2 shows the number of test patterns generated for the undetected faults. The seed per pattern column shows the number of seeds that must be stored if a single seed per pattern is assumed as it is the case in most of the previous work. The table shows the number of seeds that must be stored and the seeds that need to be encoded with our technique.

Table D.3 shows the seed storage needed for the seed per pattern scheme and the seed storage needed for our scheme. The storage is calculated by multiplying the number of loaded seeds by the PRPG size and the number of encoded seeds by the bit counter size. The table also shows the reduction gained by using our scheme. The reduction varies from 25% to 85%.

The area overhead required of our technique implemented completely on chip with reseeding and seed encoding circuits is comparable to the areas shown in [Alyamani 03b] where the overhead ranged between 0.3% and 10% and mostly less than 3%.

Table D.2 Number of Seeds for Our Technique Compared to One Seed per Pattern.

Circuit	Seed per Pattern	Our Technique		
		Loaded Seeds	Encoded Seeds	Total
s953	47	9	16	25
s1196	92	13	16	29
s1238	97	12	12	24
s1423	17	9	0	9
s1488	89	4	9	13
s1494	85	3	11	14
s5378	35	26	1	27
s9234	117	82	6	88
s13207	182	30	16	46
s35932	7	1	0	1

Table D.3 Seed Storage Needed by our Technique Compared to Seed per Pattern.

Circuit	Circuit Cell Area	Seed per Pattern storage	Storage for Our Technique			Storage Rduction %
			Loaded Seeds	Encoded Seeds	Total	
s953	2,286	987	189	112	301	69.50
s1196	2,722	1380	195	96	291	78.91
s1238	2,740	1455	180	72	252	82.68
s1423	4,531	850	450	0	450	47.06
s1488	3,555	534	24	54	78	85.39
s1494	3,563	510	18	66	84	83.53
s5378	14,377	2135	1586	9	1595	25.29
s9234	25,840	9360	6560	60	6620	29.27
s13207	44,255	14560	2400	160	2560	82.42
s35932	106,198	420	60	0	60	85.71

6. Conclusion

In this paper, we presented a seed encoding technique based on running a variable number of clock cycles before loading the seed.

The simulation experiments showed that the storage needed is reduced by 25%-85% when our encoding technique is used compared to storing a single seed per pattern. The main characteristics that make our technique effective are: (1) Running pseudorandom patterns after loading the seeds (2) Ordering the seeds to load the minimum number of seeds, (3) Encoding the seeds by the number of cycles needed to reach them, and (4) Exploiting the degrees of freedom in solving for the seeds to match them with the current contents of the LFSR.

References

[Alyamani 03a] Al-Yamani A., and E. J. McCluskey, "Built-In Reseeding for Serial BIST", VLSI Test Symposium, Apr., 2003.

[Alyamani 03b] Al-Yamani A., S. Mitra, and E.J. McCluskey, "BIST Reseeding with Very Few Seeds", VLSI Test Symposium, Apr., 2003.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, "Built-In Test for VLSI", John Wiley, New York, 1987.

[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", IBM Journal of Research and Development, Vol. 27, No. 3, pp. 265-272, May 1983.

[Eichelberger 89] Eichelberger, E., E. Lindbloom, F. Motica, and J. Waicukauski, "Weigted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 1989.

[Fagot 99] Fagot, C., O. Gascuel, P. Girard and C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test," Proc. of European Test Workshop, pp. 7-14, 1999.

[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," IEEE Transactions on Computers, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," Proc. of European Test Conference, pp. 237-242, 1991.

[Koenemann 00] Koenemann, B., "System for Test Data Storage Reduction," US Patent 6,041,429, Mar. 2000.

[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding" Proc. of International Test Conference, pp. 885-893, 2001.

[Lempel 95] Lempel, M., S. Gupta and M. Breuer, "Test Embedding with Discrete Logarithms," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 5, pp. 554-566, May 1995.

[McCluskey 85] McCluskey, E.J., "Built-In Self-Test Techniques," IEEE Design & Test, pp. 21-28, Apr. 1985.

[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , "Test Data Decompression for Multiple Scan Designs with Boundary Scan," IEEE Transactions on Computers, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[Touba 96] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. of VLSI Test Symposium, pp. 2-8, 1996.

[Touba 00] Touba, N. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.
[Wunderlich 90] Wunderlich, H.-J., "Multiple Distributions for Biased Random Test Patterns," IEEE Transactions on CAD, Vol. 9, No. 6, pp.584-593, Jun. 1990.

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517.

Appendix

In [Alyamani 03b], we presented seed calculation for LFSRs. In this section, we show how to apply the algorithm to cellular automata. Figure D.5 shows an example for a 6-stage Cellular Automaton.

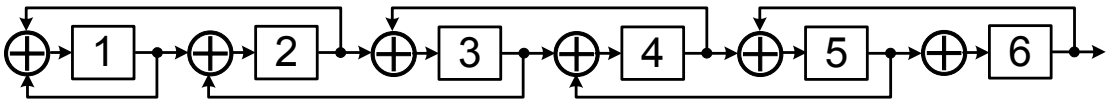


Figure D.5 A 6-stage cellular automaton.

For every flip-flop in the scan chains, there is a corresponding equation in terms of the bits of the PRPG. Let's label the flip-flops of a given scan chain by $S_0 \rightarrow S_{m-1}$ where m is the length of the scan chain. Also, let's label the stages of the PRPG by $L_0 \rightarrow L_{n-1}$ where n is the size of the PRPG. In the example above, assume that the CA is connected to a scan chain at the output of stage 6 of the CA. Also assume that the scan chain has 9 stages $S_0 - S_8$. The equation for the deepest stage of the scan chain is $S_8 = L_5$, because after m clock cycles the most significant bit of the seed ends up in the most significant bit of the scan chain. The reader is invited to verify the remaining equations:

$S_7 = L_4$ $\qquad$ $S_6 = L_3 \oplus L_5$

$S_5 = L_2$ $\qquad$ $S_4 = L_1 \oplus L_3$

$S_3 = L_0 \oplus L_4$ $\qquad$ $S_2 = L_0 \oplus L_1 \oplus L_2 \oplus L_4$

$S_1 = L_2 \oplus L_5$ $\qquad$ $S_0 = L_1 \oplus L_3 \oplus L_4$

We can represent the above equations by an $m \times n$ matrix in which the rows correspond to the scan chain flip-flops and the columns correspond to the PRPG stages. An entry (i,j) is 1 if and only if L_j appears in the equation of S_i. According to this system, the following matrix shows the equations for all the flip-flops in the scan chain of the example above:

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \end{matrix}$$

$$\begin{matrix} L_0 & L_1 & L_2 & L_3 & L_4 & L_5 \end{matrix}$$

In the case of multiple scan chains, the outputs of the PRPG stages may need to go through a phase shifter to avoid structural dependencies that cause undesired correlation between patterns in different chains.

The phase shifter for a given scan chain is a linear sum of some stages from the PRPG. The phase shifter for chain i can be represented by a vector $p^i = \begin{bmatrix} p^i_{n-1} & p^i_{n-2} & \cdots & p^i_2 & p^i_1 & p^i_0 \end{bmatrix}$. p^i_j is one if the XOR feeding scan chain i has the output of stage j of the PRPG as one of its inputs.

The algorithm that generates the equations for a scan chain S that is fed through a phase shifter p^s starts with the vector p^s. Algorithm D.1 is used to generate the equation matrix E_s.

The algorithm starts by assigning the vector p^s to the last row of E_s. The other rows are generated bottom up by multiplying the transition matrix H by the following row of E. This algorithm can be used with any number of scan chains.

The algorithm works with any PRPG and phase shifter. It depends on the transition matrix and the phase shifting vector, so it works with any linear machine.

1. ***m:*** *depth of the scan chains*
2. ***n:*** *size of PRPG*
3. ***p^s:*** *the phase shifter for the current scan chain*
4. ***E_s:*** *equations matrix for the current scan chain*
5. ***E_s-Generator (m, n, h, E_s)***
6. $E^{(m-1)} = p^s$
7. ***for*** *i=m-2 to 0*
8. $E^i = H \times (E^{(i+1)})^T$
9. ***endfor***
10. ***end***

Algorithm D.1 Generating equations matrix Es for scan chain s fed through a phase shifter.

We exploit the degrees of freedom in solving the equations to increase the probability of finding a match between s(m+1) and the remaining seeds. The degrees of freedom are caused by the fact that the number of equations is less than the number of unknowns. This is a consequence of don't care bits in test patterns and the fact that the size of the PRPG depends on the maximum number of care bits.

The methodology for utilizing the degrees of freedom to increase the chances of finding a match between the final state of the PRPG and one of the remaining seeds is explained with an example in [Alyamani 03b].