Summer 2018

Design for test methods to reduce test set size

Yingdi Liu
University of Iowa

Follow this and additional works at: https://ir.uiowa.edu/etd
Part of the Electrical and Computer Engineering Commons

DESIGN FOR TEST METHODS TO REDUCE TEST SET SIZE

by

Yingdi Liu

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

August 2018

Thesis Supervisor: Professor Sudhakar M. Reddy

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

____

PH.D. THESIS

____

This is to certify that the Ph.D. thesis of

Yingdi Liu

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the August 2018 graduation.

Thesis Committee: ______________________________________
Sudhakar M. Reddy, Thesis Supervisor

____
Janusz Rajski, Thesis Advisor

____
Jon G. Kuhl

____
David R. Andersen

____
Xiaodong Wu

To My Parents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Prof. Sudhakar M. Reddy for the continuous support throughout the years of my graduate work, for his patience, encouragement, and immense knowledge. I would also like to express my heartfelt thanks to my advisor, Dr. Janusz Rajski for suggesting the topics researched in this thesis, the extraordinary patience, guidance and encouragement throughout the years of my study and related research.

I would like to thank the rest of my thesis committee members, Prof. David Andersen, Prof. Jon Kuhl and Prof. Xiaodong Wu for their insightful comments and advices on this work.

My sincere thanks also go to Chen Wang, Xijiang Lin, Elham Moghaddam, Jedrzej Solecki, Yu Huang and Nilanjan Mukherjee at Mentor, a Siemens Business, and Prof. Jerzy Tyszer at Poznan University of Technology, for their sharing and help.

Finally, I am profoundly grateful to my wife Jin Xiang, my parents and my grandparents, for their love and support throughout the years of my PhD study.

ABSTRACT

With rapid development in semiconductor technology, today's large and complex integrated circuits require a large amount of test data to achieve desired test coverage. Test cost, which is proportional to the size of the test set, can be reduced by generating a small number of highly effective test patterns. Automatic Test Pattern Generators (ATPGs) generate effective deterministic test patterns for different fault models and can achieve high test coverage. To reduce ATPG-produced test set size, design for test (DFT) methods can be used to further improve the ATPG process and apply generated test patterns in more efficient ways.

The first part of this dissertation introduces a test point insertion (TPI) technique that reduces the test pattern counts and test data volume of a design by adding additional hardware called control points. These dedicated control points are inserted at internal nodes of the design to resolve large internal conflicts during ATPG. Therefore, more faults can be detected by a single test pattern. To minimize silicon area needed to implement these control points, we propose a method that reuses some existing functional flip-flops as drivers of the control points, instead of inserting dedicated flip-flops for the control points. Experimental results on industrial designs indicate that the proposed technique can achieve significant test pattern reductions, similar to the control points using dedicated flip-flops.

The second part of this dissertation proposes a staggered ATPG scheme that produces deterministic test-per-clock-based staggered test patterns by using dedicated compactor scan chains to capture additional test responses during scan shift cycles that are used for regular scan cells to completely load each test pattern. These compactor scan chains are formed by dedicated capture-per-cycle observation test points inserted at suitable locations of the design. By leveraging this new scan infrastructure, more compacted test patterns can be generated, and more faults can also be systematically detected during the simulation process, thus reducing the overall test pattern count.

To meet the stringent test requirements for in-system test (especially for automotive test), a built-in self-test (BIST) approach, called Stellar BIST, is introduced in the last part of this dissertation. Stellar BIST employs a dedicated BIST infrastructure with additional on-system memory to store some parent test patterns (seeds). Derivative test patterns can be obtained by complementing selected bits of corresponding parent patterns through an on-chip Stellar BIST controller. A dedicated ATPG process is also proposed for generating a minimal set of test patterns that need to be stored and their effective derivative patterns that require short test application time. Furthermore, the proposed scheme can provide flexible trade-offs between stored test data volume and test application time.

PUBLIC ABSTRACT

With rapid development in semiconductor technology, today's large and complex integrated circuits require a large amount of test data to achieve desired test coverage. Test cost, which is proportional to the size of the test set, can be reduced by generating a small number of highly effective test patterns. Automatic Test Pattern Generators (ATPG) generate effective deterministic test patterns for different fault models and can achieve high test coverage. To reduce ATPG-produced test set size, design for test (DFT) methods can be used to further improve the ATPG process and apply generated test patterns in more efficient ways.

We first introduce a test point insertion (TPI) technique that reduces test pattern count and test data volume of a design by adding additional hardware called control points that are inserted at internal nodes of the design to resolve large internal conflicts of ATPG assignments. To minimize silicon area needed to implement these control points, we propose a method that reuses some existing functional flip-flops as drivers of the control points, instead of inserting dedicated flip-flops for the control points.

We also propose a staggered ATPG scheme that produces deterministic test-per-clock-based staggered test patterns by using dedicated compactor scan chains to capture test responses by leveraging the scan shift cycles. This new scan infrastructure allows us to generate more compacted test patterns to systematically detect more faults and reduces the overall test pattern count.

Finally, we introduce a built-in self-test (BIST) approach, called Stellar BIST that employs a dedicated BIST infrastructure with additional memory to store a minimal set of parent test patterns (seeds). Desired test stimuli can be obtained by complementing selected bits of corresponding parent patterns through an on-chip Stellar BIST controller.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With rapid growth in technology, Moore's law indicates that the number of transistors in an integrated circuit doubles every eighteen months to two years, while the size of the transistors themselves are actually decreasing. This may lead to more complicated design structures which would require more test patterns to test the entire design. It may also require advanced ways to apply tests.

Contemporary Automatic Test Pattern Generators (ATPG) generate test patterns for different fault models and for large complex nanometer designs to achieve high fault coverage. Working synergistically with on-chip test compression logic, ATPG can produce highly compact (small) test sets. However, due to the fast-changing semiconductor manufacturing process and the increasing complexity of digital designs, the inflated test data volume has had a significant impact on the test application time, which, in turn, has had a visible impact on the overall test cost. At the same time, the rapid development of advanced driver assistance systems (ADAS) and self-driving vehicles, with stringent requirements for high quality and long-term reliability driven by functional safety standards, demands advanced test solutions that would rely on ATPG to produce small but effective test sets.

In this thesis, we address the problem of reducing the size of ATPG-produced test sets (deterministic test patterns). The main objective of this study is to introduce methods that result in more compact test sets and to apply generated tests in more efficient ways for different test concerns.

In this chapter, we introduce the basic concept of design for test (DFT) and some of the DFT methods and fundamentals that are involved in our work.

1.1 Design for Test

Manufacturing test screens products after they are manufactured and helps eliminate products containing defects that occurred during manufacturing process. Test vectors are loaded into a chip through its primary input pins while responses are observed through its primary output pins. The tester, known as automatic test equipment (ATE), is used during the manufacturing test for the circuit under test (CUT). Test vectors and expected responses are stored in the ATE memory in advance. As shown in Figure 1.1, during the manufacturing test, test patterns are applied to the CUT and output responses are then analyzed and compared to expected fault-free responses. If the output responses do not match the fault-free responses, then the circuit is considered to be defective.

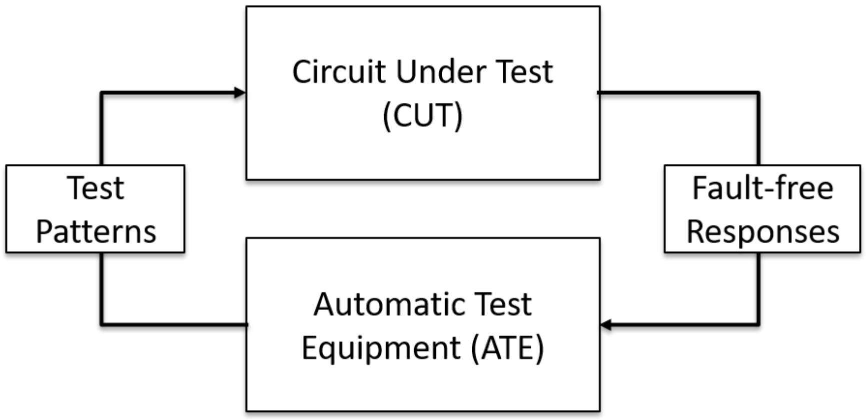


Figure 1.1 Manufacturing test

The cost of performing tests by an ATE could account for a large portion of the total manufacturing cost. Most of this cost will depend on factors, such as the test application time and the memory capacity of the ATE. In other words, complicated designs which require more test patterns and longer test application time would have a higher cost in ATE based testing. The process of design for testability (DFT) is one way to make a design easier to test. DFT techniques add additional hardware to the CUT to improve its testability.

In this sub-section, we will introduce some popular DFT methodologies and concepts including scan design, built-in self-test (BIST), and test data compression.

1.1.1 Scan Design

A CUT typically contains both combinational logic and sequential cells (such as flip-flops and latches). To test a circuit with sequential cells, test vectors should be generated over multiple cycles. This may increase the total test application time, as well as the test vector size. If a circuit contains many sequential elements, generating tests to achieve meaningful fault coverage for the entire design would be very difficult and has been found to be impractical.

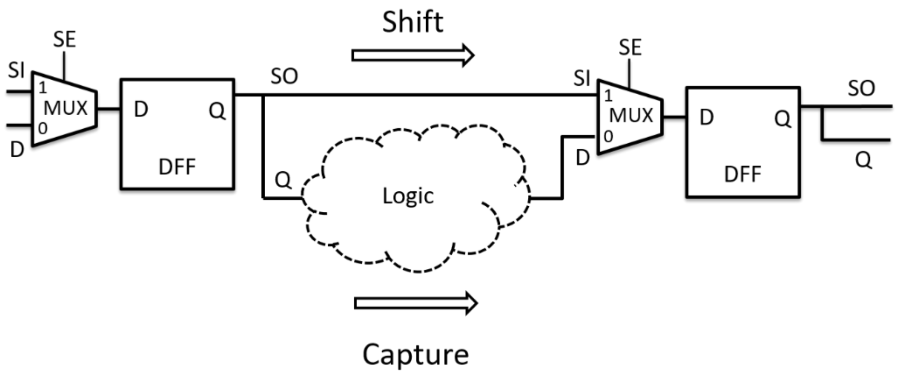


Figure 1.2 Scan flip-flops

Testing a circuit with pure combinational logic is much simpler and is the preferred method. If possible, the sequential elements should be made controllable and observable (like primary inputs and outputs), so the circuit can be tested as a design with pure combinational logic. The DFT methodology of scan design [1] was introduced to achieve this objective.

Scan design modifies original flip-flops to scan cells, by adding an additional test mode to the flip-flops. As shown in Figure 1.2, a scan-modified flip-flop has an extra data input called scan-in (SI) and a test mode control input called scan enable (SE), which selects between the flip-flop data input SI and its original data input (D). Each SI pin is directly connected to a primary input or the output of another scan flip-flop (SO), and the output is connected to an SI pin of another scan flip-flop or a primary output. Thus, during test mode, these flip-flops form one or more shift registers, also known as scan chains. During shift cycles (the scan shift is enabled), test vectors are shifted into scan chains from primary inputs to set scan flip-flops to specific values. Test responses from the combinational logic of the CUT are then captured (by disabling the scan shift) in to scan cells. Captured responses are shifted out of the scan chains to primary output pins, while loading the next test pattern.

Although the scan design technique may add extra hardware to the circuit and may require extra time for shifting data in and out, it provides a high level of fault coverage and the capability of performing defect diagnosis.

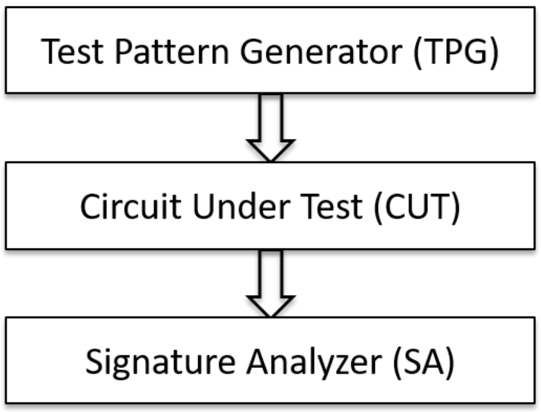


Figure 1.3 BIST scheme

1.1.2 Built-in Self-Test (BIST)

Built-In Self-Test (BIST) [2] is a way to make it possible for a circuit to test itself through additional hardware.

Figure 1.3 shows a basic BIST scheme. With no external patterns, the BIST technique uses an internal test pattern generator (TPG) as the pattern source. Linear feedback shift registers (LFSRs), which are good at generating pseudo-random patterns, yet require little area overhead, are commonly used as a TPG. The CUT responses to these random patterns generated from the TPG, are then analyzed through a signature analyzer (SA), which determines whether the circuit has passed the test. Multiple-input signature registers (MISRs) are often used to construct a signature analyzer. Responses are compacted and examined through the MISR. With the help of the BIST technique, the test can be performed in-field. Thus, the data exchanged between CUT and the tester is drastically reduced.

Although many faults can typically be detected using random patterns, BIST requires that the CUT has no bus conflicts and no unbounded X sources (unknown values that may corrupt the BIST signature have to be bounded by additional logic), and that the circuit should also be random pattern testable. Typically, it is to top off pseudo-random patterns with additional deterministic patterns to cover remaining random pattern resistant faults or use weighted random patterns [2], [3], [4] to achieve required fault coverage. However, experimental data shows that using top off tests may not help reduce the total test application time. For this reason, for in-field test, stored pattern test methods are being considered in which deterministic test patterns are stored in the memory of a system/circuit under test [5], [6].

1.1.3 Test Compression

Since the design sizes may double every 18 months, test data volume (which is proportional to the number of test patterns and the number of scan cells) also grows, and the test application time increases. Therefore, a larger test data volume would require more tester memory for storing test data and would also lead to a longer test application time. This may lead to a significant increase in test cost.

To achieve reduction of test data volume, several test compression methodologies have been introduced. An effective test compression technique, called Embedded Deterministic Test (EDT) [7] was developed by inserting additional logic into the circuit, utilizing the existing scan design, and refraining from touching any functional paths of the CUT.

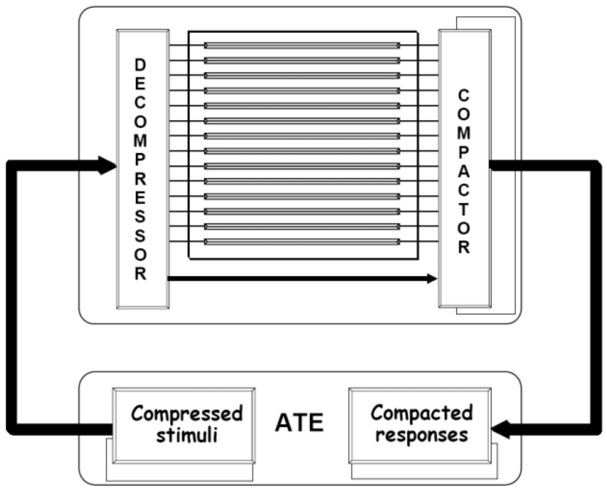


Figure 1.4 EDT architecture [7]

As shown in Figure 1.4, EDT logic contains a decompressor placed between the chip's input channels and internal scan chain inputs and a compactor placed between internal scan chain outputs and the chip's output channels. The decompressor consists of a ring generator and a phase shifter [7]. The ring generator, which is an optimized LFSR, is used to decompress the compressed data injected from input channels. The phase shifter, placed at the outputs of the ring generator, reduces linear dependencies between sequences shifted into internal scan chains. A high compression ratio between the number of internal scan chains and the number of input channels can be achieved with the help of the decompressor. This allows for a shorter length of internal scan chains by accommodating a large number of scan chains. Therefore, both the test data volume stored in the tester memory and the test application time can be reduced. To evaluate output responses, the compactor, which mainly consists of an XOR tree, is used to observe the output data with fewer outputs, while not losing the observability of errors in responses from internal scan chains of faulty circuits.

1.2 Fault Models

During circuit fabrication process, physical defects may occur throughout a device [8]. These defects may include shorts, opens, or transistor defects. To obtain tests to detect these defects, different fault models are introduced for pattern generation. In this sub-section, a brief description of several fault models which have yielded tests that have achieved great success in detecting defects in manufactured circuits is given.

1.2.1 Stuck-at Fault Model

Stuck-at fault model [9] is used to describe the faulty behavior of a line permanently tied to a logic value. A line considered to be tied to logic 1 or 0, is said to be stuck-at-1 (s-a-1) or stuck-at-0 (s-a-0). To test a stuck-at fault, test vectors need to be capable of exciting the fault as well as propagating the fault-effect to a primary output or a pseudo-primary output (a scan cell input).

Stuck-at fault model is an important fault model. All the other fault models that are used today, can be considered as conditional stuck-at faults [10].

1.2.2 Transition Fault Model

Transition delay fault model [11] is introduced to detect defects that cause increased propagation delays of circuit paths. The two types of transition faults are called the slow-to-rise fault and the slow-to-fall fault. A slow-to-rise fault occurs when a line that needs to switch from 0 to 1 requires a longer than normal time (delay). If this delay effect can be captured by any primary output or scan flip-flop, the circuit is not able to function properly at the given clock speed. Similarly, a slow-to-fall fault occurs when a line requires a longer time (delay) to switch from 1 to 0.

Based on the definition of the transition fault model, to detect a given transition fault, two patterns applied at a given clock speed are required. To test a slow-to-rise (slow-to-fall) fault at a given line, the first pattern has to set the line to the initial value 0 (1), and the second pattern needs to set the line to the final value 1 (0) and propagate the delayed transition effect to a primary output or a scan flip-flop. Two of the commonly used transition fault pattern generation methods include: launch-off-capture (LOC) and launch-off-shift (LOS), which are briefly introduced below.

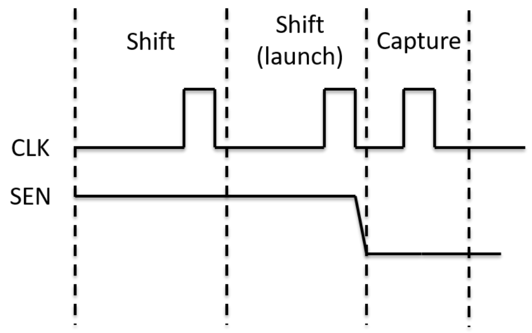


Figure 1.5 Waveform for Launch-off-Shift delay test

Launch-off-Shift (LOS) [12]: The timing waveform for a LOS test procedure is shown in Figure 1.5. The transition at faulty the line is launched in the last shift cycle of shifting the test, followed immediately by a functional clock pulse to capture the circuit outputs. The scan enable (SEN) signal is used to switch the CUT from shift mode (launch the transition) to function mode (capture the transition) with at-speed timing. Both patterns of a two-pattern LOS test are obtained by shifted in values. In LOS test to achieve at-speed test of the fault the SEN line has to switch fast.

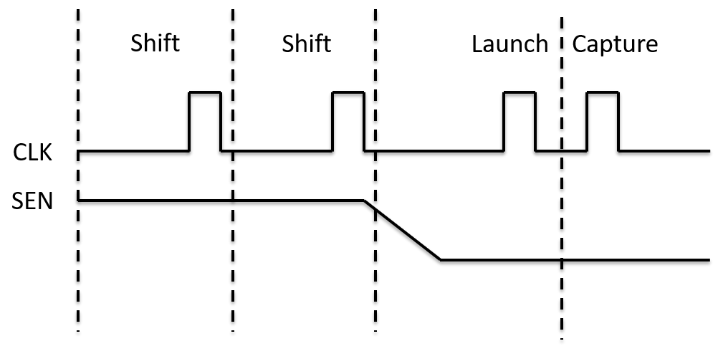


Figure 1.6 Waveform for Launch-off-Capture delay test

Launch-off-Capture (LOC) [13]: The timing waveform for a LOC test procedure is shown in Figure 1.6. A pair of functional clock pulses is used to launch and capture the transition after SEN is de-asserted. The scan enable signal does not have to switch fast for LOC and the second pattern of the two-pattern LOC test is generated from the functional response of the first pattern.

1.2.3 Path Delay Fault Model

Path delay fault model [14] is another fault model introduced for testing defects causing circuit delays to increase. Compared to the transition delay fault model, which is considered to be a local delay fault model to test the transition delay of selected gate inputs or outputs, the path delay fault model is a global delay fault model that tests the delay of an entire path.

1.2.4 Cell-aware Test

Cell-aware test [15] is a new approach targeting transistor level defects to further reduce defective parts shipped, especially for advanced manufacturing technologies, such as FinFET technology. Since tests generated using the traditional stuck-at fault model may not be sufficient to detect cell-internal defects, the cell-aware test provides a new fault modeling approach that is based on post-layout transistor-level netlist of cells (gates) with parasitic effects. As a result, ATPG can then be applied to detect the cell-internal defects that are not covered by stuck-at patterns.

1.3 Test Generation

To test a given fault, certain controllable inputs (scan cells or PIs) need to be specified to activate the fault effect and also to propagate the fault effect to an observable site (scan cells or POs) for the detection of the targeted fault. D-algorithm [16] is a commonly used algorithm for line justifications and fault propagations during test generation.

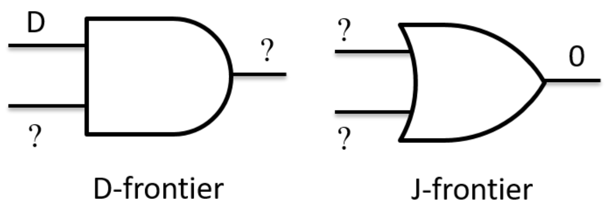


Figure 1.7 Examples of a D-frontier and a J-frontier

To test a stuck-at-v (v = 0 or 1) fault on a given line k, the value v' (the complement value of v) should be first implied (justified) on line k. The D notation [16] uses D or D' to represent the value on a circuit line in the fault-free and faulty circuit. D (D') stands for

1/0 (0/1) where the value above (below) the "/" is the value on a circuit line in the fault-free (faulty) circuit. These D-values are then propagated forward to an observed output (a scan flip-flop or a primary output). Propagation of D or D' to an output requires the implication of values on the circuit along the path through which D or D' is propagated. Two data structures are used during line justifications and D propagations. One data structure (called D-frontier) contains those gates, whose inputs have received one or more Ds, while the outputs could not yet be specified. The other data structure, called J-frontier, contains all those gates with specified outputs but undetermined inputs. Simple examples of a D-frontier AND gate and a J-frontier OR gate with unspecified pins are shown in Figure 1.7. To justify a gate in the D-frontier, proper values are assigned to one or more of the gate's inputs. If multiple choices of input assignments exist, a decision is made at the gate by selecting one choice of input assignments. For propagation, gate entries are selected from the J-frontier and necessary input assignments are made to propagate D or D'. Selecting gates from any of the data structures and making necessary assignments to related lines, may remove corresponding entries and add new gate entries.

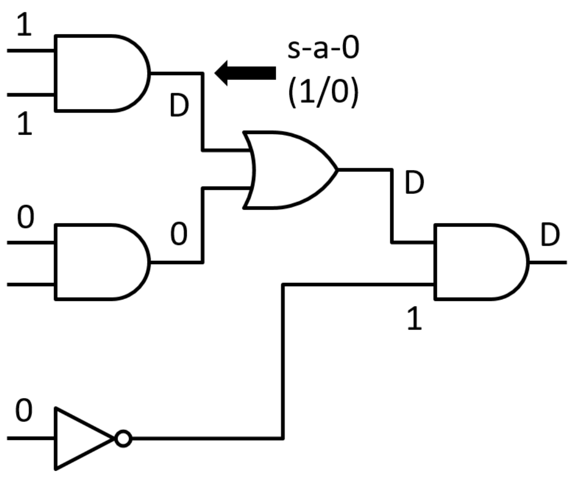


Figure 1.8 Test generation example

Test generation for a given fault using the D-algorithm is the process of clearing all entries in the J-frontier by making necessary assignments and decisions among internal lines, with at least one circuit output value being a D or D’. A test generation example for a s-a-0 fault using D-algorithm is shown in Figure 1.8.

1.4 Test Points

Additional test logic, known as test points, can be inserted into a design to improve its test quality. Test point insertion (TPI) techniques add control points (CP) and observation points (OP) to internal lines of a circuit to improve a specific objective related to testing [8].

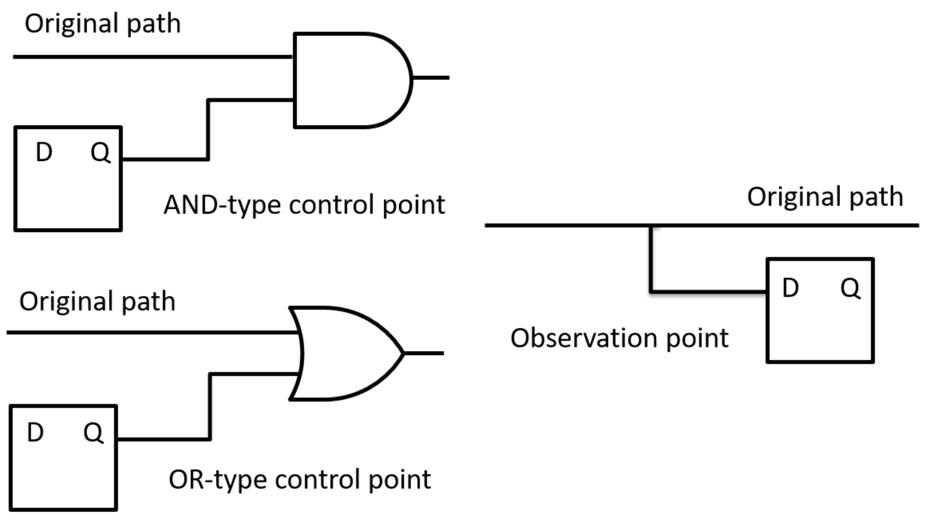


Figure 1.9 Test point examples

Control points are additional inputs which together with additional AND/OR gates set internal lines to desired values when control points are enabled. The control points are typically driven by dedicated scan flip-flops, rather than additional input pins on the CUT. AND type control points, using flip-flops driving AND gates, are used to set internal lines to the value 0, by simply setting the driver flip-flop to 0 through scan shift-in without changing any of the functional path’s values. Similarly, OR type control points, using flip-flops driving OR gates, are used to set internal lines to the value 1. Observation points are added to make internal nodes observable. This is similar to jumper cables used to observe internal nodes on PC boards. Observation points allow the values of the corresponding locations to be observed, as the states of the observation points are captured in additional scan flip-flops. Without observation points, these values can only be observed by propagating through existing paths to some scan cells or primary outputs. Examples of an AND-type control point, an OR-type control point, and an observation point are shown in Figure 1.9.

1.5 Automotive Test

In the fast-growing automotive electronics market, integrated circuits (ICs) must adhere to the most stringent requirements for high quality and long-term reliability driven by functional safety standards such as ISO 26262 and its Automotive Safety Integrity Level D (ASIL D) targets [17]. In addition to the high quality of manufacturing test, an ASIL D compliance for automotive ICs requires advanced and complementary test solutions that need to respond to the challenges posed by automotive parts and support such test requirements as:

1. the ability to run in-system tests during functional operations,

2. short test application time, due to strict limits on key-on, key-off, and especially idle times deployed for periodic (and often segmented [18]) on-line tests,

3. low test power,

4. low silicon area,

5. the ability to deal with defect sensitivities unknown at the time of IC manufacturing,

6. the potential to scale up easily.

1.6 Thesis Overview

In this thesis, we present methods for reducing ATPG-produced test set size, involving the addition of extra test logic and new ATPG processes. The rest of the thesis is organized as follows.

Motivation and the review of previous works for each technique are reported in Chapter 2. In Chapter 3, we introduce a control point insertion technique that reduces the ATPG pattern counts and test data volume with a minimized silicon area overhead by reusing functional flip-flops as drivers of control points [19]. Control points are inserted at suitable locations that resolve large internal conflicts of ATPG assignments. This helps a single test pattern to target more faults and reduce the overall pattern count. Since each control point requires a dedicated driver of an extra flip-flop, we introduce a method to replace the dedicated drivers by some existing functional flip-flops, while maintaining a similar functionality.

In Chapter 4, we introduce a new test scheme that reduces ATPG pattern counts by applying patterns in a test-per-clock manner [20], instead of applying patterns after a complete scan load as described in Section 1.1.1. Dedicated capture-per-cycle observation scan chains with observation test points are added to capture additional test results during shift cycles of scan loading. By leveraging this new scan feature, we can generate more compact test patterns (also called staggered test patterns) and systematically detect many additional faults. In this way, fewer patterns are required to achieve the same test coverage compared to any conventional test-per-scan scheme.

In Chapter 5, we introduce a deterministic two-level compression scheme named Stellar BIST to address the new challenges of in-system automotive test [21]. This method can generate effective test stimuli from a small number of stored seeds (also called parent patterns) through additional hardware, while the stored test patterns are generated and compressed through a dedicated ATPG process. With a user-specified parameter, this approach can provide significant trade-offs between on-chip memory usage and total test application time. Finally, a summary of the methods proposed in this thesis and future research are given in Chapter 6.

CHAPTER 2
MOTIVATION AND PREVIOUS WORKS

As discussed in Chapter 1, ATPG tools generate deterministic test patterns to achieve high fault coverage. To reduce test costs, one has to reduce the number of tests needed to achieve desired fault coverage defined as the percentage of targeted faults that are detected by the tests applied to the CUT. If stored pattern tests are used for in-field test, then effective ways to trade-off pattern counts and memory needed to store test patterns should be considered. In this thesis we investigate solutions to both these issues.

Methods to reduce test pattern counts can be classified into two classes. The first one is software or algorithm based that typically use heuristics to improve ATPG procedures to achieve lower test pattern counts [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38]. Typically, these methods try to maximize the number of faults detected by a generated test. The second class of procedures insert test points to reduce pattern counts [39], [40], [41], [44], [46], [47]. Postprocessing may be used to further reduce pattern counts by eliminating redundant tests in the test sets generated by the ATPGs [28], [32], [33], [34], [35]. In this thesis we investigate two methods to use test points to reduce pattern counts. We also investigate a method to generate tests such that they can be derived from a subset of stored patterns which provides an effective way to trade off stored pattern counts and test application time.

Next, we review earlier software-based methods to generate compact test sets (in Section 2.1), and methods for test point insertion (in Section 2.2), as well as methods to reduce test application time (in Section 2.3), and stored patterns for in-field test (in Section 2.4).

2.1 Software Based Methods to Reduce Pattern Counts

Two approaches have been investigated to derive compact test sets. One is to use heuristics to guide ATPGs to generate compact test sets and the other is to investigate methods to realize logic circuits that require small test sets to detect modeled faults. In Section 2.1.1, we review the methods to guide ATPGs to derive small test sets and in Section 2.1.2, we give a brief review of methods to realize logic circuits that require small test sets to detect all modeled faults.

2.1.1 Generation of Compact Test Sets

As discussed in Section 1.3, test for a given fault (also called a test cube) can be generated using the D-algorithm. The generation of test sets can use static or dynamic compactions, or a combination of both to achieve a small (compact) test set size. Static compaction methods generate effective test patterns based on a complete test set, either through the merging of multiple test cubes to form test patterns that can detect many faults, or through some postprocessing procedures by simulating the test set in a different order and dropping redundant tests that cannot help detecting any new fault. Dynamic compaction methods try to incrementally fill the unspecified bits of test cubes in an intelligent way to create effective test patterns that detect many undetected faults. In this subsection, we will review some of these test compaction methods that guide ATPGs to generate compact test sets.

A dynamic and a static compaction method are introduced and compared in [24]. The dynamic compaction procedure gradually fills the unspecified positions of a test vector (targeting a single fault) to form a test pattern that detects many undetected faults, and the static compaction method tries to merge compatible test cubes together to form a test pattern after the generation of a complete test set (targeting all the faults). This paper shows that dynamic compaction methods may yield better performance, since static compaction methods require more memory and time to target all the faults. Another method, introduced in [25], suggests a fast and effective way that gradually generates and merges test cubes targeting only a set of undetected faults at a time (instead of targeting all the faults at once).

COMPACTEST, introduced in [26], suggests several new concepts that can guide ATPG to generate a more compact test set, such as the fault ordering and the potential compatible fault list that can be utilized for test generation, a dynamic method to reduce specified positions of a test vector, and a rotating backtrace method for making ATPG decisions. Some other methods, introduced in [22], [23], [27], [28], [29], [30], [31], focus on similar concepts as using fault ordering, compatible faults, or other ATPG decision guidance to get compact test sets.

After obtaining a complete test set, some of the test patterns can still be redundant (test patterns with all the detected faults that are already covered by other test patterns). Therefore, the test set can be further compacted by dropping test patterns that cannot detect any new fault. Several postprocessing procedures are introduced in [28], [32], [33], [34], [35]. They use the fault detection profiles from the pattern simulation results to either come up with certain pattern simulation orders to drop redundant test patterns, or directly get a minimal test set based on the fault detection information.

2.1.2 Synthesis of Easily Testable Circuits

Typically, test problems are not considered until a design is completed. From a testing point of view, test aspects can be specified as some of the design criteria at the design stage, which may lead to a better test solution. It has been suggested in [36] that it is feasible to synthesize easily testable circuits that require minimal test sets. This paper suggests a realization of any arbitrary logic function using a Reed-Muller Network of *n* variables (as introduced in [37], states that any Boolean function can be implemented by this network) and shows that it only requires *n*+4 test vectors to fully test the circuit, which is also independent of the function that has been realized. Another paper in [38] also presents that a minimum test set with only 11 test vectors are required to completely test for all multiple faults of any ripple carry adder, independent of the number of cells.

Therefore, if easy-to-test criteria are specified at the design phase, it could be possible to synthesize circuits with minimal test sets that can be easily obtained.

Another way to modify a design to make it easy-to-test is using test point insertion techniques, which will be discussed in Section 2.2.

2.2 Test Point Insertion

In this section, we will discuss the motivation for test point insertion and review some test point insertion (TPI) techniques for different test purposes and present a TPI technique investigated in this thesis.

2.2.1 Motivation for Test Point Insertion

As discussed in Section 1.4, test points are extra test logic inserted into a design to gain additional controllability or observability of some internal nodes. This may help detect some hard-to-detect faults or to control/observe an internal node for some other test concerns. Since test points also require additional chip area, the most significant task for TPI is to find a limited number of test point locations that are most effective to achieve desired test goals. Due to different application purposes, different methodologies of finding optimal test points may need to be developed.

In Section 2.2.2, we review TPI techniques for testability improvement that helps detect hard-to-detect faults. In Section 2.2.3, we review TPI techniques proposed to improve test pattern counts. We give a detailed review of a TPI technique that targets the reduction of ATPG pattern counts and test data volume, which we also investigated to improve area overhead required for this method.

2.2.2 Test Point Insertion for Testability Improvement

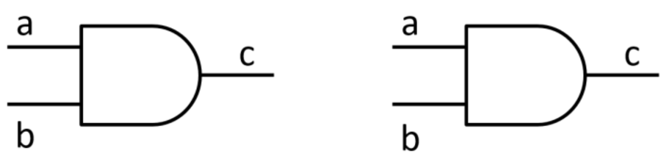
As discussed in Chapter 1, BIST technique generates pseudo-random test patterns for self-testing a design. Although BIST has certain advantages, the low detection probabilities of faults that are pseudo-random-pattern-resistant (also called hard-to-detect) continues to be a problem. To overcome this, the TPI approach is developed for improving random pattern testability. With the help of test points, random-pattern resistant faults are rendered random pattern detectable.

To detect random-pattern-resistant or hard-to-detect faults, test points are inserted at certain nodes of the design to improve testability of targeted faults. Testability can be improved by improving controllability and/or observability [8]. Controllability is used to determine the difficulty of setting an internal line to the value 1 or 0, while observability is used to determine the difficulty of driving a fault-effect from an internal line to a primary output or a pseudo-primary output (a scan flip-flop).

Several empirical methods for testability measurements have been proposed to guide test point placement [39], [40], [41], [42], [43], [44]. Circuit testability is estimated either through exact fault simulation or approximate measures. To avoid the large CPU-time that fault simulation requires, approximate measures are used. This provides a less time-consuming way to characterize controllability and observability information without applying any additional input vectors. The Sandia Controllability/Observability Analysis Program (SCOAP) [45] is one such measure and an easy-to-compute method for testability measurement. It can be used to guide ATPG and the placement of test points.

A digital design typically consists of combinational cells (such as AND gates, OR gates, inverters, etc.), sequential cells (such as flip-flops or latches), and the interconnections that exist between them. SCOAP [45] defined six metrics to calculate controllability/observability values. Associated with each node, these measures are as follows: combinational 0-controllability (CC0), combinational 1-controllability (CC1), combinational observability (CO), sequential 0-controllability (SC0), sequential 1-controllability (SC1), and sequential observability (SO). For a combinational node (a node that is the output of a combinational cell), combinational controllability (CC) is defined as the minimum effort to justify a 0 or 1 at the node. Likewise, combinational observability (CO) is defined as the minimum effort to propagate the value on that node to a primary output. Sequential metrics are defined in a similar way for a sequential node that is the output of a sequential cell.

The controllability and observability of each individual cell is calculated to determine the difficulty of setting the node to a particular value. The controllability of an output is calculated by first checking all possible input assignments that achieve the desired output value, and then taking a minimum value of the sum of the controllability values of its inputs. The observability of an input of the cell is calculated by the observability of its output, plus the sum of the controllability of other inputs that sensitize the path to the output at minimum cost [45]. The calculation for an AND gate is shown in Figure 2.1.



$$CC0_c = \min\{CC0_a, CC0_b\} \qquad CO_a = CO_c + CC1_b$$
$$CC1_c = CC1_a + CC0_b \qquad CO_b = CO_c + CC1_a$$

Figure 2.1 Controllability/observability calculation

For an entire digital design, the process of computing controllability and observability can be divided into two phases [45]. At the beginning of Phase One, controllability and observability values have not yet been assigned. Combinational controllability values of primary inputs are initialized to 1, while sequential controllability values of primary inputs are initialized to 0. All other nodes' controllability values are initialized to infinity. During Phase One, controllability values are computed forward from primary inputs and updated for each node until the numbers stabilize. Observability values are then computed during Phase Two. At the beginning of Phase Two, observability values for all primary outputs are initialized to 0, and observability values for all other nodes are initialized to infinity. During Phase Two, observability values are computed backward from each primary output. Utilizing the controllability values that were calculated during Phase One, observability values are computed until all observability numbers stabilize. Controllability and observability are completely assigned by the end of Phase Two. Large controllability values may indicate nodes that are difficult to justify to specific logic values while large observability values may indicate nodes that are difficult to observe through any primary output. This information can be used to guide the TPI process and find potential test point locations.

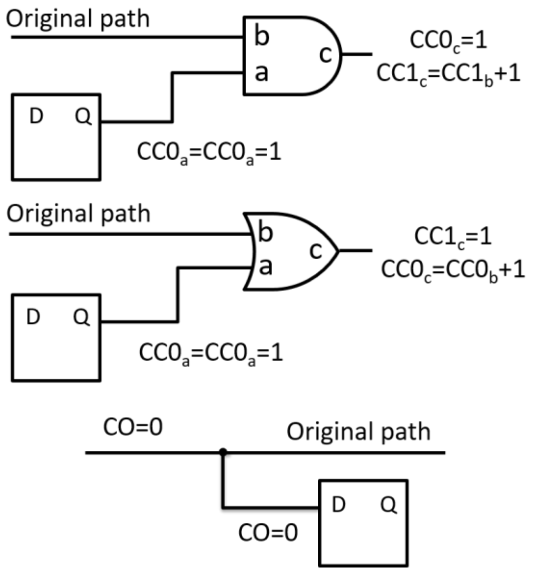


Figure 2.2 Test point insertion for testability improvement

As shown in Figure 2.2, according to the controllability and observability measures, by inserting a control point at a specific node, either 1-controllability or 0-controllability of the node can be reduced, making it easier to justify the node as well as some forward-traced nodes to certain logic values. In the meantime, according to Phase Two of the

SCOAP process, observability values of certain nodes may also be reduced after the insertion of control points. Similarly, by adding observation points, observability values of some nodes can also be reduced, making these nodes easier to be observed.

2.2.3 Test Point Insertion for Reducing Deterministic Test Patterns

Although testability-based TPI techniques are found to have an incidental decrease in pattern counts [23], [46], their performance in pattern count reduction is significantly less predictable. Several methods, introduced in [39], [40], [41], [44], [46], [47], focus on using test points for the purpose of test compaction. These methods model one or more aspects that determine the final test pattern count and improve test compaction results with properly inserted test points. In this subsection, we will discuss the approach proposed in [47] of using test points to reduce ATPG pattern counts and test data volume by modeling and resolving conflicts of internal assignments during the ATPG process, which is the approach that will be further investigated in Chapter 3.

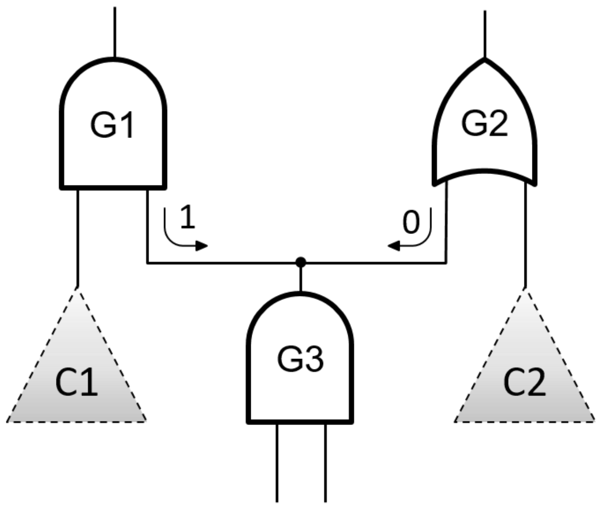


Figure 2.3 Conflict on internal lines

To detect as many faults as possible by a single pattern, these faults must become parallel targets, provided there are no internal assignment conflicts. Given a specific node in a design, two groups of faults that require opposite values at that node cannot be detected simultaneously. As shown in Figure 2.3, Gate G1 requires an input value of 1 to propagate faults in C1, whereas Gate G2 requires an input value of 0 to enable propagation of faults in C2. Clearly, the immediate conflict between backward implied values at G1 and G2 precludes simultaneous detection of faults in C1 and C2 by any test pattern. In general, incompatible assignments made by ATPG during fault excitation, forward propagation, or backward implication lead to conflicts on internal signal lines that may stop the propagation of effects of many faults. Fortunately, by inserting appropriate control points, these conflicts can be successfully resolved. Note that faults in C1 and C2 could be targeted simultaneously, if an AND type control point (for 0-controllability) was added to the input of Gate G2, or an OR type control point (to achieve 1-controllability) was inserted at the input of Gate G1.

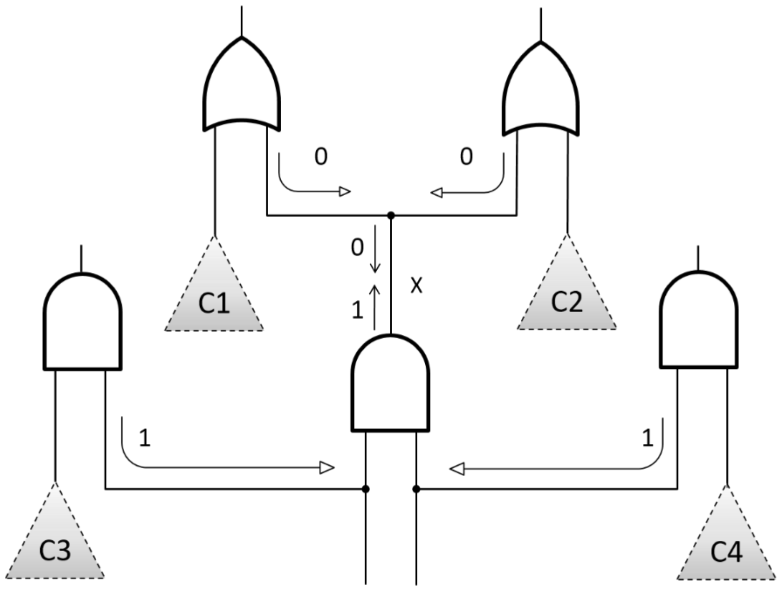


Figure 2.4 Forward propagation and backward justification

The example of Figure 2.3 suggests that fault-blocking mechanisms can be used to quantify the effect of internal signal assignments on faults detected by one test pattern. Having a line set to a certain logic value may block forward propagation of several faults to an observation point. Therefore, the opposite value on that line becomes a necessary ATPG assignment for the same group of faults. As shown in Figure 2.4, faults in C1 and C2 require stem *x* to be set to 0, whereas faults in C3 and C4 require 1s on the inputs of the respective AND gates. There is a conflict at line x between the forward-implied value of 1 and the backward-implied value of 0.

To determine conflicts such as those illustrated in Figures 2.3 and 2.4, in [47], fault-blocking information is forward-implied and backward-implied throughout the circuit. Four metrics are used to characterize the internal signal assignments during the ATPG process. Given node *x*, these metrics are defined as follows:

- B_x – the number of faults whose propagation could be halted (blocked), if node *x* was set to 0; this is equivalent to the count, in the process of backward implication, of how many times one needs to set *x* to 1 to propagate these faults through all relevant gates,
- b_x – the same as above, but to characterize the process of backward implication with respect to 0,
- F_x – the number of forward-implied 1s on node *x*, based on earlier backward implication,
- f_x – the number of forward-implied 0s on node *x*, based on earlier backward implication.

For a fan-out stem x_0 with fan-out branches denoted as x_i, the above four metrics are computed using the following formulae:

$$B_{x_0} = \sum B_{x_i} \qquad (1)$$

$$b_{x_0} = \sum b_{x_i} \qquad (2)$$

$$F_{x_k} = F_{x_0} + \sum B_{x_i} \qquad i \neq k \qquad (3)$$

$$f_{x_k} = f_{x_0} + \sum bx_i \qquad i \neq k \tag{4}$$

In addition, forward-implied F and f values can also be computed for outputs of different gates, based on F and f values assigned to their inputs. This is done through a simple structural analysis. F and f values of output node s are determined as follows [47]:

$$\begin{array}{lclc} f_s = f_k & \multirow{2}{*}{(5a)} & f_s = F_k & \multirow{2}{*}{(5b)} \\ F_s = F_k & & F_s = f_k & \\ f_s = \max\{f_k\} & \multirow{2}{*}{(6a)} & f_s = \min\{F_k\} & \multirow{2}{*}{(6b)} \\ F_s = \min\{F_k\} & & F_s = \max\{f_k\} & \\ f_s = \min\{f_k\} & \multirow{2}{*}{(7a)} & f_s = \max\{F_k\} & \multirow{2}{*}{(7b)} \\ F_s = \max\{F_k\} & & F_s = \min\{f_k\} & \end{array}$$

The above formulae correspond to buffer (5a), inverter (5b), AND (6a), NAND (6b), OR (7a) and NOR (7b) gates. Once forward-implied and backward-implied information is known, we can measure the degree of conflicts for a given node x_k as follows:

$$c_{x_k} = \min\{b_{x_k}, F_{x_k}\} \tag{8}$$

$$C_{x_k} = \min\{B_{x_k}, f_{x_k}\} \tag{9}$$

An example of computing metrics (8) and (9) is shown in Figure 2.5. For a stem x, forward-implied values on each fan-out branch are computed by the forward-implied values on x and the backward-implied values on other branches.

The test point insertion steps of [47] can be summarized as follows. Following the observations from conflict analysis, as discussed earlier in this subsection, conflict metrics are first computed. F and f values are calculated for each gate, starting from the first level. In the meantime, B and b values are also determined when a fan-out branch is encountered. After traversing the entire design, conflict values of each node are calculated by equations (8) and (9). A sorted node list by conflict values can then be collected and used to guide the test point insertion process. Test points are then inserted at nodes with the largest conflict values. After the insertion of a test point, related conflict metrics are updated, recomputing backwards and forwards from the branch with the newly-inserted test point. Since the newly-inserted test point may also affect earlier decisions, it is also necessary to check whether previously inserted test points have become less effective after the insertion of a new test point. This is done by removing an early-inserted test point and comparing the conflict metrics with the current conflict metrics. The whole process will continue, until a predefined number of test points are inserted.

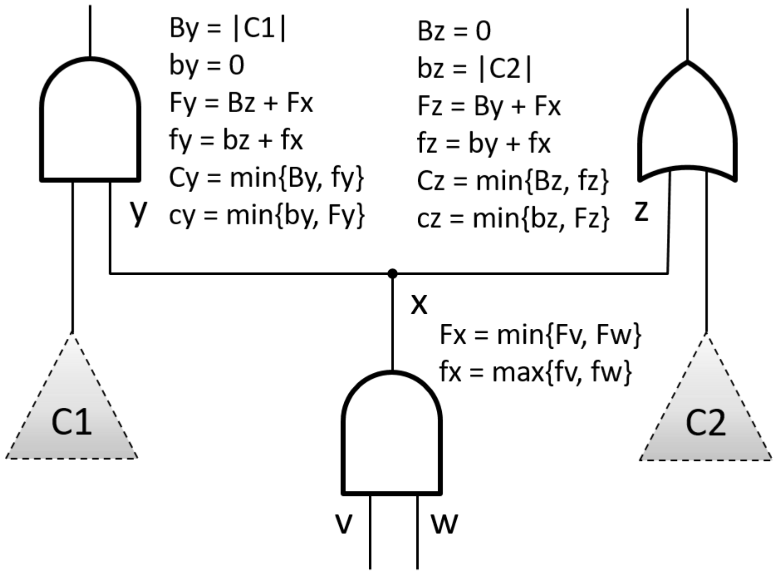


Figure 2.5 Conflict metrics

Similar to the control points used for improving random pattern testability, conflict-aware control points are implemented by using dedicated flip-flops (scan cells) driving additional AND/OR gates. To reduce the area overhead, existing functional flip-flops can be selected to replace the dedicated flip-flops. In Chapter 3, we will propose a method to insert conflict-aware control points, while minimizing the area overhead by reusing functional flip-flops as drivers.

2.3 Test-per-clock

In this section, we will discuss the difference between test-per-clock and test-per-scan and review previous works applying the test-per-clock idea for both random and deterministic patterns to reduce test application time.

The test application techniques used by most scan-based designs can be classified as either test-per-scan or test-per-clock. For a test-per-scan procedure, test patterns are applied only after all the scan registers have been completely loaded. The total test length for each test pattern is equal to the total shift length plus an additional capture cycle. However, in a test-per-clock process, test responses are captured every clock cycle while test vectors are applied. Many test schemes, introduced in [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], leverage the advantage of test-per-clock to apply test patterns in shorter time. These earlier methods include build-in logic block observers (BILBO) [54], a circular self-test path [51], [53], [55], [57], E-BIST [50], [56], and some techniques applying deterministic test patterns [52], [61], [62].

A tri-modal scan (TMS), discussed in [58], has scan cells partitioned dynamically to work in three modes as shown in Figure 2.6, acting as either mission memory elements (M mode scan cells in Figure 2.6), sources of test stimuli (S mode scan cells in Figure 2.6), or test response compactors (C mode scan cells with XOR gates in Figure 2.6). In the last two cases, scan cells form the actual scan chains. Scan chains in the stimuli mode resemble the conventional scan chains in the shift mode. However, test data is applied to the CUT every clock cycle, and these scan chains do not capture test responses. The latter functionality is assumed by scan chains in the compaction mode that accumulate test responses every clock cycle. At the same time, a single bit (per chain) of the resultant signature is always shifted-out. The remaining scan cells are kept in the mission mode. Test results propagating through the combinational part of the circuit can also reach the scan cells in the mission mode. These responses further circulate within the circuit and eventually reach the observation scan chains during the subsequent clock cycles. Since test patterns are applied every clock cycle, the scheme is time-efficient and makes it possible to complete a test within much shorter durations than done by conventional schemes.

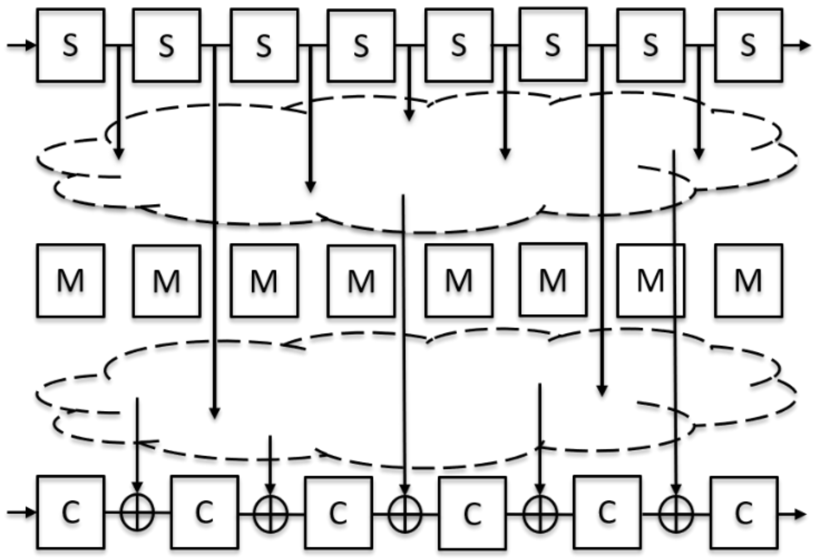


Figure 2.6 Tri-modal scan architecture [58]

Another approach is introduced in [59]. Unlike the dynamic scan structure used by TMS [58], this method employs dedicated shadow registers (observation points) with XOR gates to capture test results during the scan shift to fortuitously detect cell-aware faults with test patterns generated for stuck-at faults. The shadow flip-flops are directly associated with scan cells that are capable of observing the largest number of cell-aware faults during successive shift cycles. Test data of the original test patterns form additional intermediate test patterns during these shift cycles and test results are captured and accumulated in a test-per-clock fashion by the test response compactor formed by the shadow registers. According to the experimental results, this scheme achieves 10% – 45% less cell-aware test patterns with selected shadowed registers. This method can also be applied to fortuitously detect stuck-at faults to reduce the test pattern count. Results in [59] show that, with all scan cells shadowed, this method can achieve 30% – 90% test pattern reduction for designs with highly observable sites (scan cells that can easily capture many fault effects), and for more complicated industrial designs with limited observation of faults, this method achieves 5% – 10% test pattern reduction for stuck-at faults after all the scan cells are shadowed. In Chapter 4, we will present an approach that uses a similar idea as [59] to insert observation test points to capture fault effects during shift cycles. Different from [59], we develop a dedicated test generation procedure to produce effective test-per-clock patterns based on this test scheme.

For a scan-based Logic Built-In Self-Test (LBIST), as introduced in [60], to maximize the fault detection of intermediate random patterns (test patterns obtained during scan shift cycles), dedicated observation points are placed at the design's internal node, where fault effects of a significant group of faults can propagate through. The most suitable locations for test-per-clock-driven observation points are identified by selecting internal lines with low observability, which, as discussed in Section 2.2.2, are likely to be the most preferable propagation paths for a large group of faults. Moreover, control points can also be inserted to facilitate fault propagation towards the dedicated test-per-clock observation points. Therefore, many faults are detectable during the shift cycles, while loading LBIST pseudorandom test patterns.

The principle of test-per-clock can also be used in the context of deterministic test patterns [61], [62]. As proposed in [61], a test-per-clock scheme is designed to capture test responses for every clock cycle without mixing with other test patterns. This is done by using additional multiplexers to control the CUT inputs and internal D flip-flop inputs. The test-per-clock scan chain input sequences are generated from tests created for each individual fault. The tests are then applied to the CUT, while test responses are observed by all primary outputs and scan cells in a test-per-clock manner. Another approach introduced in [62], a technique that is somewhere between the test-per-scan and test-per-clock schemes, saves shift cycles and generates a compacted test set by reusing overlapping bits between former test responses and current test vectors.

2.4 Stored Pattern Test

In this section, we will discuss the new test schemes, known as the stored pattern test, that provide much shorter test application time and better test coverage than conventional LBIST schemes for in-system test.

2.4.1 LBIST Test Schemes

LBIST is a commonly used technology developed for board, system, and in-field test. To keep up with the demands of new technologies for a viable in-system test alternative, LBIST is more often used with on-chip test compression and also employs scan as its operational baseline. With the mass market driving safety critical systems, the concept of combining LBIST and test data compression has allowed several test schemes to rival conventional manufacturing test techniques. For devices destined for long-term deployment, high test coverage with a very short test application time has become crucial for their efficient and reliable operations. However, conventional LBIST test schemes may not be sustainable to meet these high-quality test demands.

To further improve the test quality of LBIST schemes, weighted random patterns can be used to deal with unacceptably low fault coverage numbers given a feasible pattern count, [3], [4], [63]. Alternatively, desired stimuli could be obtained by perturbing pseudorandom vectors [3], [4], [64], [67], [68]. The bit-flipping [68] and its applications [69], [70] may serve as examples. Unfortunately, these schemes were heavily dependent on target test sets and had to be substantially resynthesized every time the test pattern changed, due to logic Engineering Change Orders (ECO). There are other aspects of LBIST functionality that also need to be worked out. For example, an LBIST scheme should be less vulnerable to unknown states [71], [72], or it should produce low power test patterns in a programmable fashion. Relevant solutions [73], [74], however, still handle primarily pseudorandom test data. With these patterns, it becomes increasingly difficult to achieve the desired test quality, when targeting advanced fault models, not to mention random pattern resistant failures that need routinely test points to improve test coverage.

2.4.2 Stored Pattern Test

To overcome the bottleneck of test data bandwidth of conventional LBIST and to meet high test coverage requirements, the advent of hybrid BIST schemes are introduced by storing deterministic top-up patterns in a compressed form and utilizing the existing BIST infrastructure to obtain desired test stimuli [23], [75], [76], [77], [78], [79], [80], [81]. If BIST is reused to handle compressed test data, then underlying encoding schemes typically take advantage of test cubes' low fill rates (specified scan cells to detect a fault). Solutions in this class include LFSR coding [82], static [83], [84], [85], [86], [87], [88], [89] and dynamic [7], [90], [91] LFSR reseeding. They are comprehensively surveyed in [92] and [93]. Interestingly, all test data could ultimately be stored in an on-system memory, provided that an efficient-enough test compression scheme is deployed. In such a case, stored deterministic test patterns (also known as stored pattern test) would eventually become a legitimate alternative for in-system test.

Stored pattern test applies desired test stimuli (for certain test coverage targets) that can be derived from a minimal set of stored vectors. Typically, applied test patterns are derived from stored test patterns through bit complements, [5], [6]. In [5], the author introduces a method called Star-test that selectively flips bits of the parent patterns (seeds) to obtain the corresponding derivative test patterns for each pattern cluster. Two applications of Star-test are also introduced in [5], called Star-BIST and Star-ATPG. The Star-BIST scheme requires complex test logic that makes use of scan order, polarity between the neighboring scan cells, control points inserted between them, and a waveform generator. In this manner, the scan cells can behave like a ROM to encode several deterministic parent patterns. And derivative patterns can then be applied after flipping one or more bits of the parent patterns. Another application of Star-test, called Star-ATPG, is also introduced in [5]. The Star-ATPG generates parent patterns based on a candidate fault list obtained through a fault clustering analysis. Besides the simulation of parent patterns, derivative patterns obtained through deterministic one-bit-flipping or random-bits-flipping are simulated to detect and drop additional faults. Therefore, it speeds up the ATPG performance. Another approach, discussed in [6], suggests that stored test set can be compacted through multiple bit-complements. This method introduces a dedicated complementation vector used to transform test patterns to other effective test patterns that may not need to be stored. In this case, fewer test patterns are required to be stored and all the derivative test patterns can be obtained through bit-complements. Therefore, with properly-selected complementation vectors, it is possible to further compact the test set using this method.

Star-EDT, introduced in [94], is a fully deterministic test compression that uses both the EDT-based compression and a deterministic inversion of decompressed test patterns to achieve a reduced number of stored patterns and effective derivative patterns. Instead of flipping specific bits as described in [5] and [6], the Star-EDT determines scan-slices of tests (scan cells of the same shift time frame) to be complemented. The EDT-encodable parent patterns are selected among an ATPG-produced test set and the effective derivative patterns of each test cluster are selected among those modified test patterns, each with a single complemented scan-slice of the corresponding parent pattern.

In Chapter 5, we will introduce a method called Stellar-BIST. Different from the selection of stored test patterns among existing test sets, as discussed in [6] and [94], this approach develops an ATPG-process that directly generates desired parent patterns and their derivative patterns. And each derivative test pattern is obtained by flipping multiple scan-slices of the parent test pattern, instead of the single scan-slice complement introduced in [94].

CHAPTER 3

MINIMAL AREA TEST POINTS FOR DETERMINISTIC PATTERNS

In this chapter, we propose a method for reducing area overhead for the conflict-aware control points, by reusing functional flip-flops to replace the dedicated flip-flops that have been used as drivers of control points, while maintaining similar pattern count reduction.

The rest of this chapter is organized as follows. Section 3.1 describes the motivation for reducing the test point area. Sections 3.2 to 3.4 describe the method of reusing functional flip-flops as drivers for control points and the necessary verification steps. Experimental results using this technique on several industrial designs are presented in Section 3.5, followed by conclusions in Section 3.6.

3.1 Motivation

Conflict-aware test points, described in Chapter 2, facilitate significant reductions in deterministic test patterns. However, the additional non-functional flip-flops driving the control points inevitably introduce area overhead. Depending on the number of test points, the additional area overhead required for dedicated flip-flops could be high. If we could reuse existing functional flip-flops to replace dedicated drivers, the area overhead for test point insertion would be considerably reduced. We consider using functional flip-flops as drivers for control points in this work.

3.2 Reuse of Functional Flip-flops

As discussed in Chapter 2, to resolve a conflict with a large value of conflict *c*, an AND type control point needs to be added; whereas to resolve a conflict with a large value of conflict *C*, an OR type control point needs to be added to the corresponding node.

The basic structures of control points are shown in Figure 3.1. A single control point consists of an AND/OR gate to set the control point to the dominating value, another gate to enable it, and a dedicated flip-flop to drive the control point. Dedicated flip-flops added for test points are finally stitched to scan chains once the test point insertion is completed. All formulae referred to in this chapter are from Chapter 2.

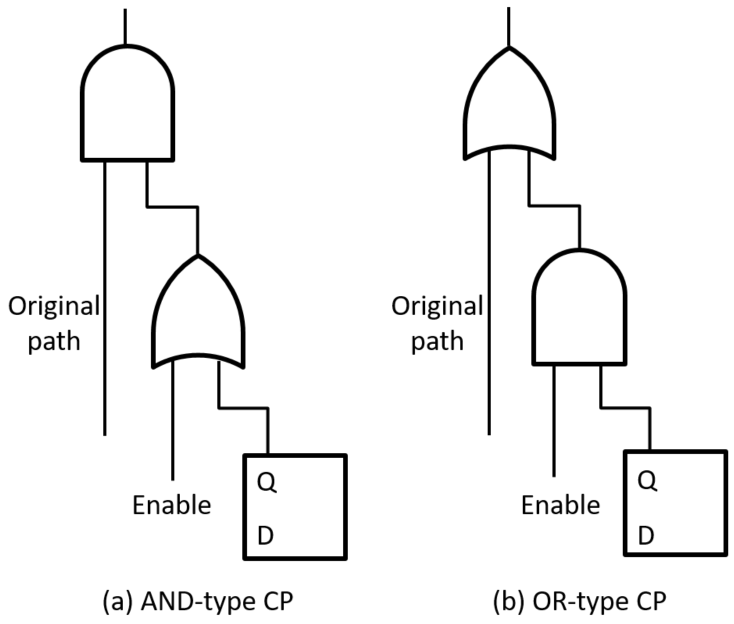


Figure 3.1 Control point templates

Figure 3.2 shows computations of conflict metrics after the insertion of control points. Note that F_R and f_R implied by the dedicated flip-flops are both 0s. The original functional path of the control point site implies F_L and f_L. Before adding a control point, lines L and S are the same net; thus, $F_S = F_L$ and $f_S = f_L$. The original conflicts can be calculated as $C_S = \min\{B_S, f_S\}$ and $c_S = \min\{b_S, F_S\}$. According to (6a) and (6b) in Section 2.2.3, after adding an AND control point to force the value of 0, F_S of the control point is reduced to 0 and c_S (the "0" conflict) is resolved. Similarly, by adding an OR control point, we can reduce f_S as well as C_S (the "1" conflict) of the control point to 0.

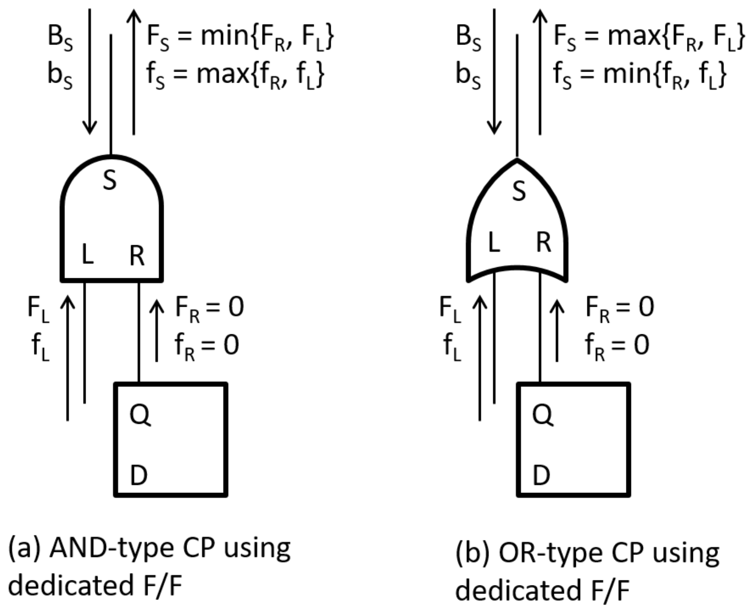


Figure 3.2 Conflict analysis for dedicated flip-flops

To minimize the area taken up by control points, functional flip-flops can be used to replace dedicated flip-flops acting as drivers of control points. As shown in Figure 3.3, control points are now connected to existing functional flip-flops, instead of adding new scan cells. This may significantly reduce the area required for each control point. However, functional flip-flops may already have fan-outs. According to formulas (1) – (4) in Section 2.2.3, forward-implied metrics (*F* and *f* values) of fan-out branches are determined by backward-justified (*B* and *b*) values occurring on other fan-out branches. Compared to dedicated flip-flop drivers where $F = 0$ and $f = 0$, the values of the same metrics for functional flip-flop drivers are determined by values of *B* and *b* coming from native fan-out branches. They are no longer equal to 0. In this case, reusing functional flip-flops as control point drivers may not reduce the conflict degree to 0. Even worse, it may introduce new conflicts at a connection interfacing a control point with original logic. Although reusing functional flip-flops may result in higher test pattern counts due to these extra conflicts, we will demonstrate that it is still possible to successfully trade-off silicon area and test pattern counts, by selecting appropriate driver candidates through a conflict analysis.

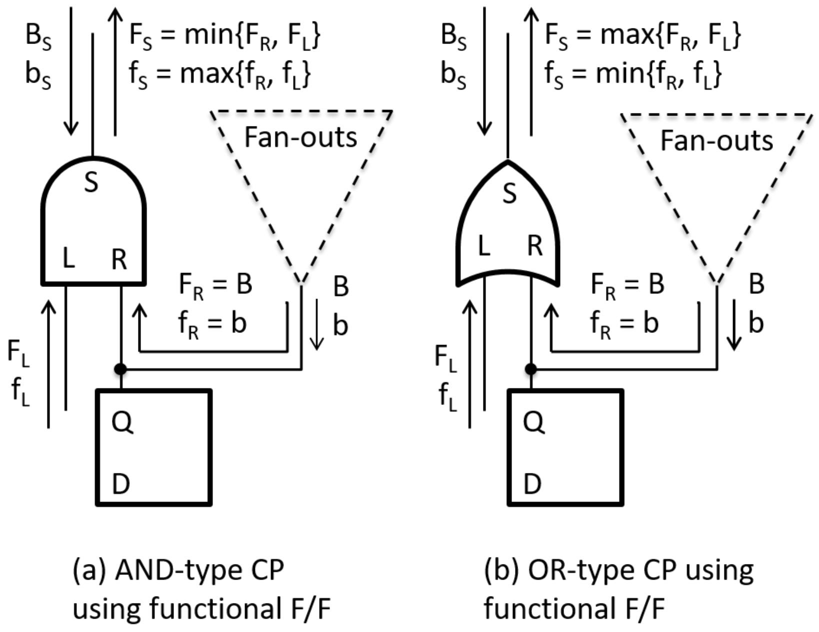


Figure 3.3 Conflict analysis for functional drivers

Figure 3.3 illustrates how conflict metrics can be computed using formulas (1) – (4). The results indicate that $F_R = B$ and $f_R = b$. Therefore, according to (6a), $F_s = \min\ \{F_L, B\}$ and $f_s = \max\ \{f_L, b\}$. From (8) and (9) in Section 2.2.3, it follows that for the AND type control point, the degree of new conflict at the control point is equal to $c_s = \min\ \{b_s, \min\ \{F_L, B\}\}$ and $C_s = \min\ \{B_s, \max\ \{f_L, b\}\}$. Conflicts due to a new flip-flop connection are $c_R = \min\ \{b_R, B\}$ and $C_R = \min\ \{B_R, b\}$. Similar results can be derived for the OR type control point. To maintain the control point's functionality, i.e., to reduce the 1/0 conflict corresponding to a control point, and to avoid introducing large conflicts, we propose to select the functional flip-flops with a minimal sum of conflicts between its own F and f values yielded by backward justifications of other fan-out branches with the control point's B and b values. In this case, we can reduce the original large conflict (conflict 0 for the AND type control point or conflict 1 for the OR type control point), as well as keep the other conflicts low. We pick a proper candidate from a set of functional flip-flops that are “logically” close to the control point bounded by neighborhood criteria. Moreover, since different circuits may have different conflict degrees, a user-defined threshold is employed to guide a searching algorithm. Candidates with a large sum of conflicts exceeding this predefined threshold are not considered, and dedicated flip-flops are used instead. By varying the threshold, it is possible to trade-off the number of functional flip-flops versus the number of dedicated flip-flops.

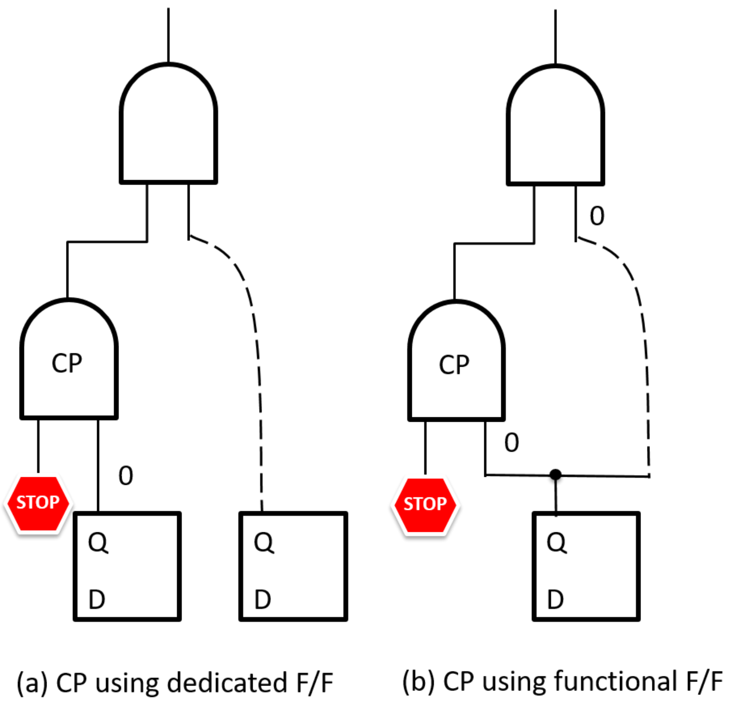


Figure 3.4 Control mode

3.3 Flip-flop Verification

When reusing a functional flip-flop as a control point driver, newly added connections may form a reconvergent fan-out with the flip-flop output branches. We need, therefore, to verify whether connecting functional flip-flops to control points compromises fault propagation. To avoid test coverage loss, flip-flops failing this verification procedure should not be employed as drivers of control points.

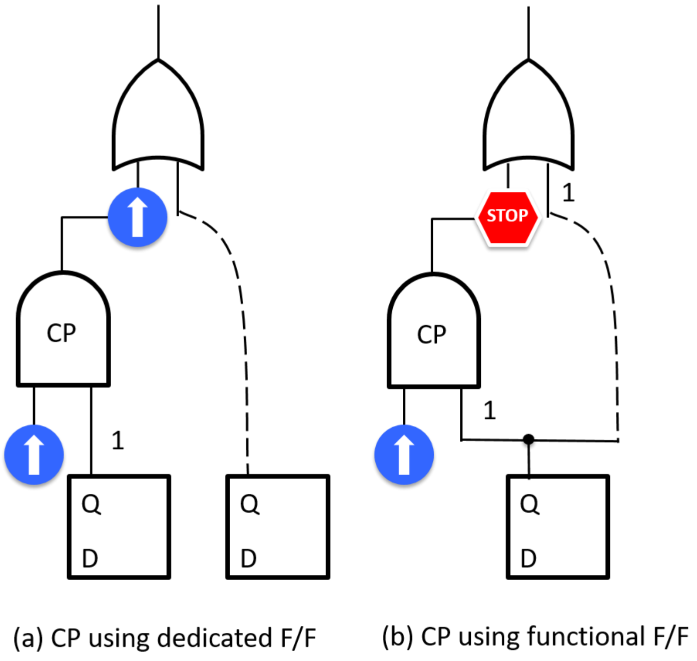


Figure 3.5 Propagation mode

During test application, control points work in two modes: a control mode to force its value, and a propagation mode to allow faults to pass. In this section, we will discuss an ATPG-based verification for these two modes.

Control points having their own dedicated flip-flops do not experience a reconvergence problem, when in the control mode. As shown in Figure 3.4(a), where an

AND type control point is used as an example, a dedicated flip-flop sets gate CP to 0 in the control mode, while a functional flip-flop may have a different assignment. Contrary to a control point driven by its dedicated flip-flop, a functional flip-flop acting as a driver may feature additional connections, as shown in Figure 3.4(b). In this case, the new connection added for enabling the control point may have different backward-justified ATPG-produced values than those of its remaining fan-out branches. As discussed in Section 3.2, incompatible assignments made to a flip-flop are quantified as conflicts, and they are already considered during the conflict analysis. This inconsistency will only lead to an increase in pattern count with no coverage loss. Therefore, it is not necessary to perform an ATPG-based verification for the reconvergence issue in the control mode.

It is not the case, however, when it comes to the propagation mode. As shown in Figure 3.5, the driver is set to the non-controlling value, so that faults from the other input can pass through the control point. Control points with a driver feeding a reconvergent fan-out may block faults, whose only propagation path goes through the control point. Consider the AND-type control point. In the propagation mode, the driver is set to 1 to let faults propagate through the control point's original path. As shown in the figure, faults whose only propagation path goes through the control point can be blocked by the value that sets the top OR gate to its dominating value. In this case, a functional flip-flop that precludes faults from further propagation cannot be considered as a driver candidate. Consequently, additional ATPG verification is required to check whether the candidate flip-flop is blocking any forward fault propagation through the corresponding control point in the propagation mode.

After running the conflict analysis, the ATPG-based verification is performed by imposing the non-controlling value on a candidate functional flip-flop. This verification checks whether the implied value may block faults whose only propagation path goes through the control point. Clearly, candidate flip-flops passing ATPG verification will not cause any coverage loss due to the reconvergence problem.

3.4 Candidate Flip-flops Search Flow

As discussed in previous sections, the selection of functional flip-flops to act as control point drivers is done within a certain (and limited) search space. For pre-layout designs, to avoid long paths from selected functional flip-flops to the control points, we select a suitable candidate among flip-flops, which are "logically" adjacent to the control point, while a user-defined threshold limits the total number of flip-flops checked for each control point.

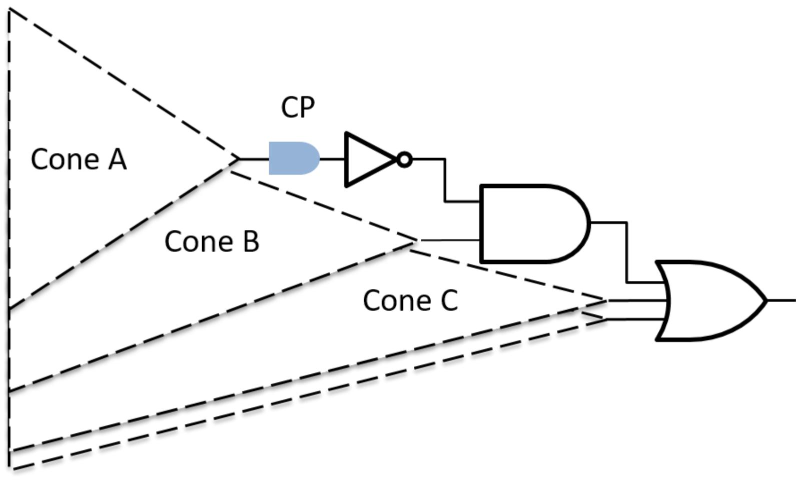


Figure 3.6 Tracing during locality analysis to find extra candidate flip-flops

Figure 3.6 illustrates a search for the most appropriate flip-flop among a limited number of functional flip-flops "near" a given control point CP. Starting with the cone of logic that drives the control point, a breadth-first search is applied to find a limited number of candidate flip-flops within the cone. If this cone does not contain enough flip-flops, we gradually enlarge the search space by tracing forward from the control point and then tracing backward from any further off-path inputs to find more flip-flops. For example (see

Figure 3.6), we first check flip-flops inside cone A. Next, we may seek more flip-flops by visiting cone B, and then cone C, until reaching the threshold.

The combined search flow to select a suitable functional flip-flop to be used as a driver of a single control point can be summarized as follows (compare Figure 3.7):

1 Start searching for a candidate flip-flop with a minimal conflict sum within a predefined conflict threshold.

2 Gradually enlarge the search space to find more flip-flops through backward tracing from any off-path input of the forward path of the control point.

3 Perform ATPG verification to check fault blocking using a non-controlling value of the control point.

4 Repeat steps 1 – 3, until a predefined number of flip-flops are reached.

5 If no candidate is found, use a dedicated flip-flop to drive the control point.

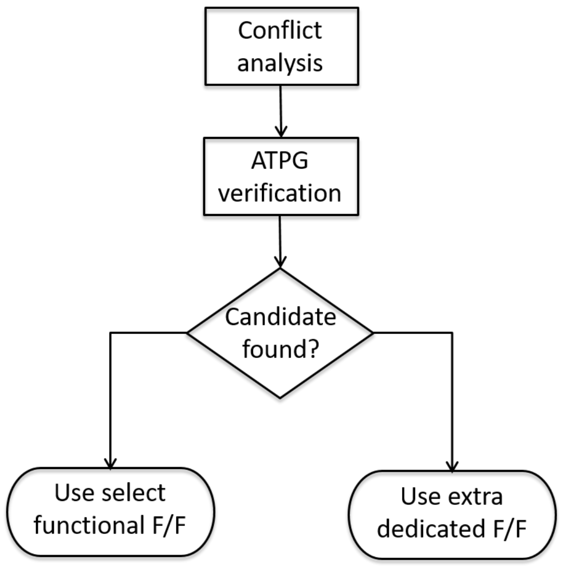


Figure 3.7 Combined search flow

3.5 Experimental Results

We have conducted experiments on 10 large industrial designs with on-chip EDT compression logic [7]. Table 3.1 includes the characteristics of the circuits, including the number of gates, the number of scan cells, the number of scan chains, the number of EDT input and output channels, as well as the baseline stuck-at test coverage obtained with no test points. For all experiments, observation points are inserted as proposed in [41]. By setting the conflict threshold to a design-specific value, we can achieve a large fraction of control points working with functional flip-flops as their drivers. Therefore, by having a small number of dedicated scan cells to drive control points, we are minimizing the silicon area occupied by test points. Functional flip-flops deployed as drivers of control points are selected within their "logical" proximity comprising 100 flip-flops, as described in Section 3.4.

Table 3.1 Circuit characteristics

	Gates	Scan cells	Scan chains	Channels	TC [%]
D1	1.19M	72K	400	4, 4	96.95
D2	103K	1K	10	1, 1	97.78
D3	218K	14K	20	1, 1	99.99
D4	2.08M	143K	400	4, 4	99.51
D5	1.32M	52K	220	6, 6	98.13
D6	1.04M	57K	400	4, 4	91.39
D7	3.34M	325K	400	4, 4	96.49
D8	2.60M	154K	1,200	12, 12	91.34
D9	1.69M	86K	400	4, 4	97.82
D10	2.31M	252K	490	10, 10	99.06

For the designs given in Table 3.1, test points are inserted using the method presented in [47] which uses dedicated flip-flops to drive control points and by using functional flip-flops as drivers of control points as proposed in this chapter. ATPG was run on the 10 designs. Although our focus is on showing how pattern count reduction depends on sharing functional flip-flops with control points, observation points are also added to each design by following the rules presented in [47]. Consequently, Table 3.2 provides stuck-at test coverage (column TC) after TPI, the total number of control points (CPs), the total number of observation points (OPs), and the reuse ratio, which is defined as the percentage of control points using functional flip-flops as drivers. For all test cases, at least a 90% reuse ratio is achieved. Therefore, for more than 90% of control points, we can minimize the area, without adding extra scan cells for test point insertion.

Table 3.2 Reuse of functional flip-flops ATPG results

	TC [%]	CPs	OPs	Reuse ratio [%]
D1	96.97	1,364	1,636	99.7
D2	98.78	300	300	90.0
D3	100.00	379	701	91.3
D4	99.55	750	750	95.0
D5	99.42	260	260	94.2
D6	92.11	600	600	97.2
D7	96.52	2,973	3,027	90.4
D8	91.66	770	770	100.0
D9	97.99	944	1,056	95.2
D10	99.70	1,500	1,500	91.3

Table 3.3 Pattern counts for dedicated (D) and functional (F) drivers (stuck-at)

	Pattern count		PC increase [%]	Pattern reduction	
	D	F		D	F
D1	1,628	1,693	4.0	1.89	1.82
D2	4,260	4,588	7.7	2.54	2.36
D3	4,102	4,425	7.9	2.68	2.48
D4	8,744	8,896	1.7	2.75	2.70
D5	24,486	24,881	1.6	1.34	1.32
D6	7,606	7,791	2.4	2.02	1.97
D7	2,590	2,619	1.1	1.36	1.34
D8	6,424	6,486	1.0	3.10	3.07
D9	9,280	9,397	1.3	1.71	1.69
D10	2,033	2,123	4.4	2.16	2.07
Average	7,115	7,289	3.3	2.16	2.08

Table 3.4 Pattern counts for dedicated (D) and functional (F) drivers (transition)

	Pattern count		PC increase [%]	Pattern reduction	
	D	F		D	F
D1	5,952	4,864	-18.3	2.47	3.03
D2	2,304	2,176	-5.6	3.62	3.83
D3	9,152	8,128	-11.2	3.32	3.74
D4	15,232	15,232	0.0	2.57	2.57
D5	8,704	9,472	8.8	3.24	2.98
D6	8,704	7,872	-9.6	2.10	2.33
D7	4,288	4,544	6.0	1.61	1.52
D8	13,362	13,924	4.2	2.47	2.37
D9	12,992	13,184	1.5	2.00	1.97
D10	832	896	7.7	6.92	6.43
Average	8,152	8,029	-1.6	3.03	3.08

Compared to the control points using dedicated flip-flops, reusing functional flip-flops reduces the total number of scan cells and minimizes the area required by the control points. As presented in Section 3.2, using existing functional flip-flops may result in additional conflicts. Hence, replacing dedicated flip-flops by functional flip-flops may increase the total pattern count. Tables 3.3 and 3.4 show comparisons that address this problem for both stuck-at faults and transition faults. Here, the pattern counts while using dedicated (functional) flip-flops are shown under D (F) in Columns 2 and 3, and the percentage increase in a pattern count, when functional flip-flops are used, is given in the next column. In the last two columns, we give the factor by which the pattern counts are reduced, relative to the pattern counts for designs not using control points. The results indicate that with an average 3.3% increase in pattern count for stuck-at faults, conflict-aware control points with functional flip-flops as their drivers can achieve a significant compression during deterministic test pattern generation. Interestingly, for transition faults, functional drivers reduce the pattern counts for D1, D2, D3, and D6, with an average 4.7% pattern count increase for other designs.

3.6 Conclusion

In this chapter, we have presented a method to insert conflict-aware control points using functional flip-flops as drivers. This approach allows the reduction of the silicon area required by control point test logic. The experimental results on industrial designs show that a 90% reuse ratio can be achieved under certain criteria. The conflict-aware test points reusing functional flip-flops can still achieve a significant reduction (2x – 3x) in pattern count, compared to baseline designs without any control points, similar to the control points using dedicated flip-flops.

CHAPTER 4
STAGGERED ATPG WITH CAPTURE-PER-CYCLE OBSERVATION TEST POINTS

The test points discussed in Chapters 2 and 3 are capable of resolving large internal conflicts to achieve the goal of reducing the overall test data volume. This TPI technique adds additional hardware to each CUT, while test patterns are still generated and applied using the same test scheme. In this chapter, we will leverage the characteristic of observation test points (as introduced in Section 2.2) and propose a new staggered test pattern generation scheme. This method produces deterministic stimuli in a test-per-clock-based process by using dedicated capture-per-cycle observation test points. We also introduce a new test generation process to adapt to the new test scheme. Therefore, the final test set size can be further reduced.

This chapter is organized as follows. Section 4.1 describes the motivation of deterministic tests-per-clock. Section 4.2 introduces a scan architecture that is used in conjunction with the proposed staggered ATPG. Section 4.3 presents the staggered test generation technique. An ATPG-oriented method and a simpler method using fault propagation is proposed in Section 4.4 for identifying the most suitable locations for capture-per-cycle observation test points. Experimental results obtained for large industrial designs are discussed in Section 4.5, and the chapter concludes with Section 4.6.

4.1 Motivation

The idea of test-per-clock, as introduced in Section 2.3, utilizes the “wasted” scan-shift-cycles to apply additional tests. These additional tests that are applied during shift cycles can only be effective when the test responses are captured properly. An easy way to achieve this is having additional observation sources to capture corresponding test results, so that additional faults can be detected. In this case, for every individual test cycle, tests can be applied to the CUT. Therefore, we may also consider generating proper tests, based on every clock cycle instead of generating tests for a complete scan.

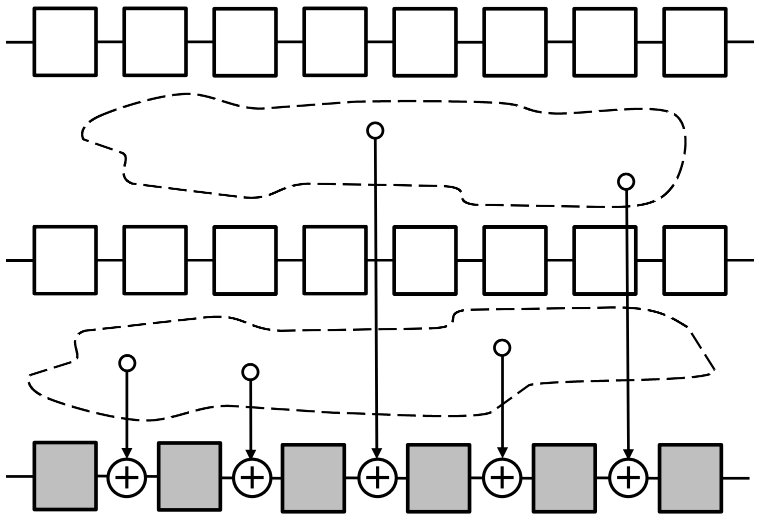


Figure 4.1 Scan architecture

4.2 Test-per-clock Scan Architecture

In a conventional test-per-scan scheme, the operational rule is to feed serial inputs of the scan chains with test stimuli and capture test responses through scan chain serial outputs. All scan cells are typically controlled by a single scan enable signal. Therefore, all scan chains are functionally indistinguishable, i.e., they all either shift data in and out or capture test results. In contrast to this paradigm, a test-per-clock architecture adapted in this work operates as shown in Figure 4.1. The majority of memory elements form conventional scan chains (white blocks in the figure), i.e., they operate either in the shift mode (with the asserted scan enable) or in the capture mode. Scan cells serving the observation points (the grey boxes in Figure 4.1) are arranged into separate scan chains operating exclusively in the compaction (capture) mode. These compaction chains accumulate test responses, using XOR gates interspaced between their successive cells. A scan cell associated with a single observation point is shown in Figure 4.2. The global test point enable (EN) signal activates observation points in the test mode, and disables them in the mission mode. Test results received from CUT (input D) are XOR-ed with data provided by another scan cell, thus incorporating shift and capture functionality within a single clock cycle. Although the capture-per-cycle observation test points may only work in the compaction mode, it is possible to modify them for the sake of scan chain integrity test, as shown in [58].

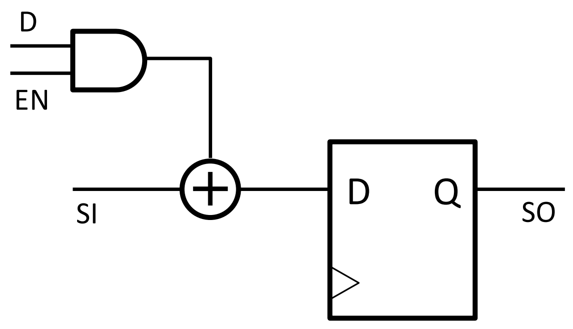


Figure 4.2 Observation point

Once a test is launched, test data are serially fed into the conventional scan chains through the scan serial inputs, while the chains' content drives the CUT. Note that after every single shift cycle there is a test pattern presented to the CUT. Test responses corresponding to these patterns are captured and accumulated by the observation test point chains every single shift cycle while regular scan cells are still operating in the shift mode (compare with Figure 4.1). As a result, it becomes possible to stagger ATPG patterns based on this functionality. The key idea of our ATPG is to deploy the capture-per-cycle observation points to record data, while regular scan cells are still being loaded. It will also generate more compact test patterns by utilizing the subsequent vectors gradually filling the regular scan chains.

4.3 Staggered ATPG

A conventional ATPG framework typically comprises the actual test pattern generation and fault simulation. First, a test cube, i.e., a group of specified scan cells, is produced to detect a single fault. A test cube is generated to activate the fault effect at the fault site, as well as to sensitize a propagation path towards an observable detection gate (scan cells or POs). Successive test cubes are kept in a buffer, where, at some point, they become subject to a cube merging process. Depending on specified bits, test cubes with no conflicting assignments are considered compatible and can be merged to form a single test pattern (Figure 4.3). ATPG continues test generation and cube merging until a certain number of patterns are created. Fault simulation usually follows to determine all faults detected by the newly formed tests. It is clear that ATPG iterates, until all faults are covered. With the capture-per-cycle observation test points allowing tests every clock cycle, the above test-per-scan-based ATPG can be modified to generate staggered test patterns that fully leverage the test-per-clock scan design of Section 4.2.

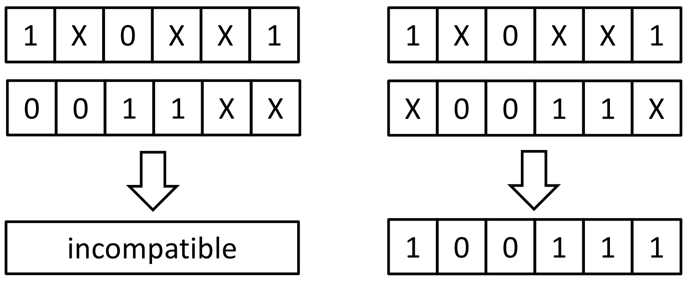


Figure 4.3 Compatible test cubes

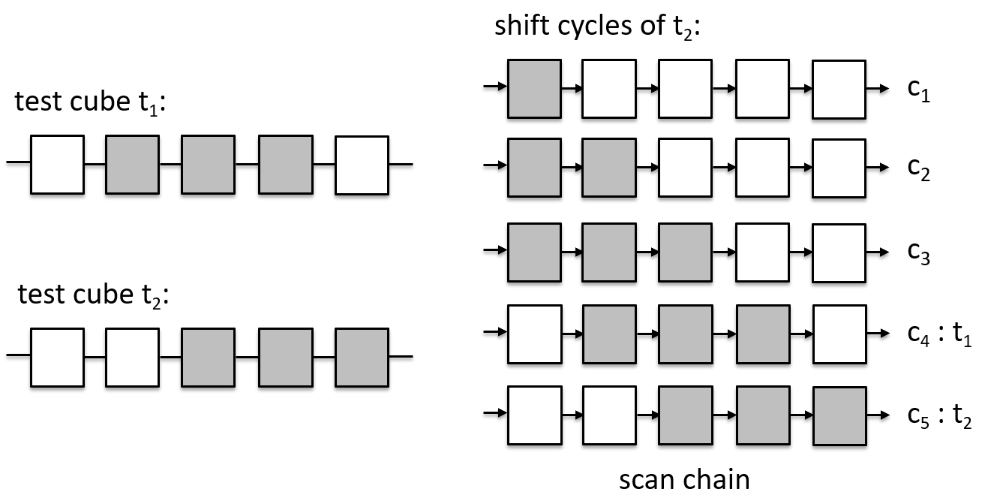


Figure 4.4 Scan specifications of test cubes during shift cycles

4.3.1 Test Cube Merging

As mentioned earlier, a test pattern can be obtained as a superposition of several compatible test cubes. With the test-per-scan paradigm in place, conflicting test cubes cannot be merged to form a single pattern. However, the test-per-clock approach provides an opportunity to merge even conflicting test cubes. Consider test cubes t_1 and t_2 shown in Figure 4.4. It appears that loading vector t_2 suffices to apply test pattern t_1 (it occurs at shift cycle c_4), provided the corresponding specified values (grey boxes) in both vectors are the same. In this case, having observation points that capture test results every clock cycle, one can generate staggered patterns based on mutually shifted test cubes, without compromising the fault coverage. Figure 4.5 is another example where one can make incompatible test cubes mergeable. This is achieved by shifting (adjusting) cubes along scan chains to form a more compact pattern, which activates all specified values of every initial test cube in a test-per-cycle manner. For example, a test pattern formed in Figure 4.5 will apply successive specified bits as follows:

- bits A during the fifth shift cycle,
- bits B during the sixth cycle,

• bits C during the seventh cycle, and

• bit D also during the seventh cycle.

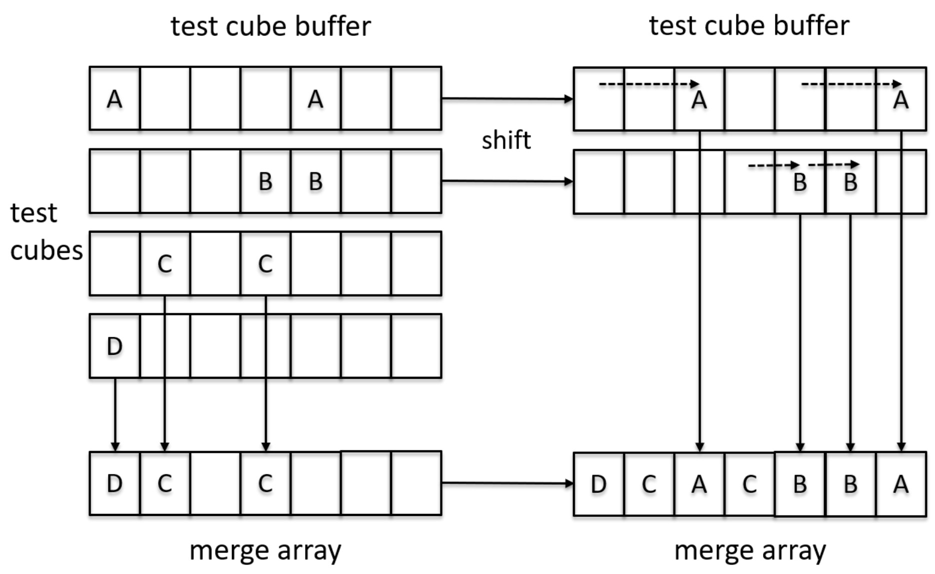


Figure 4.5 Test cube merging

If every potential fault effect can be recorded by at least one capture-per-cycle observation test point, this type of shift-and-merge process can be performed to further decrease the total number of required test patterns that need to be applied. Clearly, this approach is also well positioned to reduce the effective number of ATPG-produced test cubes. This is because many fortuitous detections are likely to be recorded by fault simulation of staggered patterns, as discussed below.

The test cube merging integrated with the staggered ATPG can be summarized as follows:

1. First, create an empty merge array M.

2. Merge test cube *t* with M, provided that it is compatible with the content of M.

3. If *t* is incompatible with M, check whether fault effects caused by *t* can be recorded by any observation point.

4. If so, inspect compatibility of *t* with M, by shifting *t* towards the scan tail end until a shifted replica of *t* becomes compatible with M (then merge *t* with M); if all shift positions have been unsuccessfully tried, then skip *t*.

5. After traversing the entire test cube buffer, generate a pattern based on the values gathered in M.

6. Reset M and repeat these steps until all relevant test cubes are examined.

Before the above procedure is carried out, it is possible to presort the test cube buffer, so that test cubes targeting faults that do not propagate to any observation points (faulty effects are exclusively captured by regular scan cells) are examined prior to the remaining test cubes. Note that these test cubes cannot be shifted along scan chains, since their capture cycles occur only at particular time frames following completion of the scan upload phase. Test cubes that propagate errors to observation points having a much more flexible capture principle (virtually every intervening scan shift cycle may record erroneous responses) can be positioned at any suitable time frame within the scan uploading sequence. As one may expect, experimental evidence indicates that this approach provides better results in terms of test cube mergeability. It is also worth noting that the test cube buffer is gradually refilled as ATPG keeps adding new test cubes for successive faults. Therefore, the procedure presented is fully compatible with the conventional ATPG flow.

4.3.2 Fault Simulation

Test cubes typically target only a small number of faults while the remaining ones are detected by simulation. After test cube merging, all bits that remain unspecified are also filled with certain values, such as, for example, pseudorandom ones. Unlike a conventional test-per-scan ATPG using a single fault simulation pass per pattern, a staggered ATPG process requires several fault-simulation runs to analyze every intermediate stimulus generated during every shift cycle, and to drop all faults detected this way. In other words, besides a fault simulation pass corresponding to the original test pattern, additional (staggered) test patterns are formed and fault simulated with cell constraints disabling regular scan cell-based observation sites. This is done by shifting the current pattern towards the scan front end, i.e., in the opposite direction of the test cube compatibility check. The total number of staggered patterns depends on the scan size. It is expected that, in addition to faults targeted by the original pattern and its merged derivatives, more faults can be detected by simulation, compared to a conventional test-per-scan ATPG. In summary, the proposed staggered ATPG flow consists of the following steps:

1. Generate test cubes for selected faults.

2. Generate a derivate test pattern by first merging the test cubes targeting faults that propagate to regular scan cells only, and then the remaining test cubes; fill all unspecified bits.

3. Fault simulate the original test pattern.

4. Fault simulate staggered patterns, by appropriately shifting the original test pattern.

5. Repeat steps 1 – 4, until no faults remain on a fault list.

Since a staggered test pattern is formed by scan cells' values of shifted test patterns plus some of the capture data of previous test patterns of corresponding scan cells. It is worth pointing out that staggered test patterns would mainly rely on their deterministic parts (shifted versions of originally generated bits) to detect faults. Although previously captured data also form parts of the staggered patterns and may detect some faults during simulation, these data are highly order-dependent, and it also has been found that additional faults detected by the captured data are not having a significant role in determining the final pattern count. Therefore, the simulation order of the test patterns is not considered as a factor to our staggered tests, and the captured data parts can be ignored (treated as unknowns) during the simulation.

4.3.3 Staggered ATPG with EDT

When on-chip test data compression (EDT) is enabled, ATPG-generated test cubes must pass an additional compression check, before forming a test pattern. This check is performed during test cube merging to determine whether the merged test can be encoded. Otherwise, the current test cube must be skipped, even though it has no conflict specifications with the merge array.

The proposed staggered ATPG can be easily adopted to work with the additional EDT compression check, by performing the check whenever a generated test cube or a shifted test cube version is being merged to the merge array. Although this may lead to a degradation in the performance of test cube merging as compared to a staggered ATPG without EDT, our staggered ATPG can provide additional flexibility to shift-and-merge a test cube in a situation where a merged cube cannot be compressed.

4.4 Observation Test Points Selection

A proper selection of capture-per-cycle observation sites is essential for successful application of the staggered test generation scheme presented in the previous section. They can be either picked to improve the cube merging efficiency, or to increase the number of faults detected by fault simulation.

A given test cube excites a fault and propagates the fault effect to at least one observation point, including regular scan cells or primary outputs. Any circuit's internal node that has been assigned values, so that the fault effect can propagate through, is called a detection gate of the corresponding test cube. As shown in Section 4.3 (see also Figure 5.5), not every test cube can be subject to the shift-and-merge procedure. Only test cubes with detection gates being observable during shift cycles can be adjusted along scan chains for compatibility checks. In other words, only test cubes activating several dedicated capture-per-cycle observation points can have multiple (flexible) locations in the merge array. Clearly, the more test cubes with detection gates acting as dedicated observation points, the better test cube merging results one may expect during the staggered test generation.

Once the merging of compatible test cubes is completed, we assume that all the remaining unspecified bits are filled with pseudorandom values. The resultant fully specified patterns are fault simulated and all detected faults are dropped. This includes faults detected by staggered test patterns as well as faults fortuitously covered by a fully specified pattern. It is worth noting that test patterns used in this process are formed by all shifted replicas of original patterns combined with captured test responses of former test patterns. Clearly, the capture-per-cycle observation test points need to be placed at proper locations to capture faults playing a significant role in determining the final pattern count of the staggered ATPG.

4.4.1 Observation Test Points Selection Using ATPG-detection

In the early stages of our experiments, we used two test point insertion techniques [41], [95], which have already proven to be successful in reducing test pattern counts and test application time. Although making those test points capture-per-cycle observation sites may allow one to detect many faults, the very same observation points are seldom sensitized by a reasonable number of test cubes. Consequently, the method that has demonstrated its superiority in a convincing manner, which seemed to offer a practical and economical way to get sufficient coverage of randomly-detectable faults, employs regular scan cells. They were found to be good observation sites for both test cube generation and fault detection. They can also serve as the capture-per-cycle observation test points for the staggered test patterns.

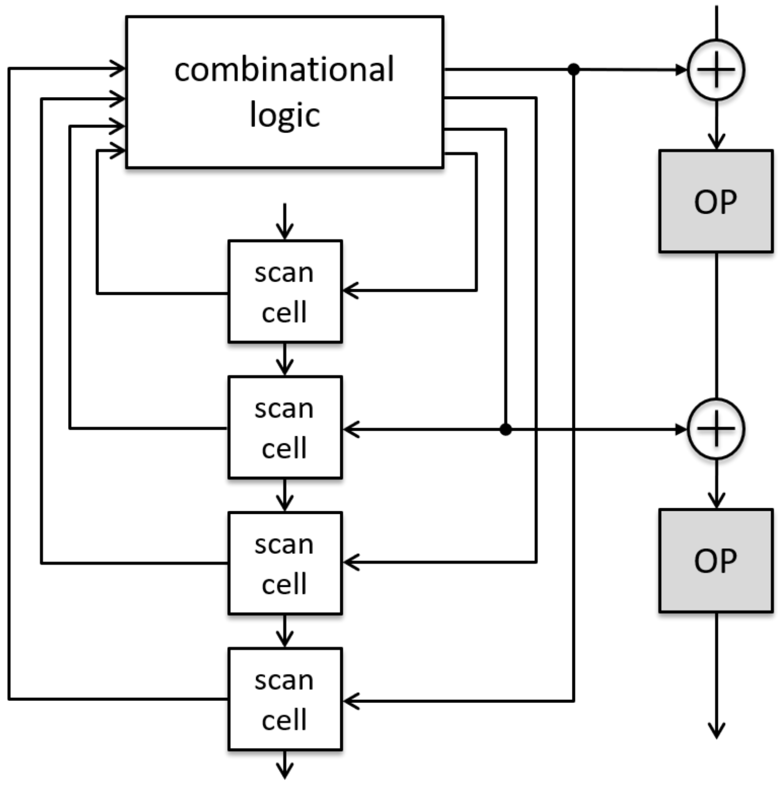


Figure 4.6 Observation points insertion

We deploy an approach similar to that of [59], by inserting observation points directly at the inputs of certain regular scan cells to capture test results that otherwise would be lost, due to the scan shift mode separating the scan cells from a combinational part of a design (see Figure 4.6). The same method avoids the risk of having observation points inserted at the internal nodes of the design's functional paths. The candidate scan cells are selected based on fault detection profiles obtained by counting how many target faults are observed (and thus detected) by every individual scan cell during a baseline (reference) ATPG run. Although the staggered ATPG may behave differently, this information can still represent the likelihood of fault detection with respect to every individual scan cell. Moreover, additional faults may propagate through sensitized paths set by the test cubes towards scan cells that are likely to be reached. Note that linking observation points with scan cells having the highest fault detection counts (targeted by test cubes), makes several test cubes "flexible". Otherwise, we cannot shift and merge test cubes, since the fault-effect targeted by the test cube cannot be captured by any observation point. Because of silicon area constraints, we only select a limited number of scan cells (approximately 1% – 2% of a design's total number of scan cells) with the largest fault detection counts. Observation test points described in Section 4.2 are inserted at these scan cell inputs.

4.4.2 Observation Test Points Selection Using Fault-propagation

Observation test points selection, based on the ATPG-detection-profile, requires a complete reference (baseline) ATPG run to determine the scan cells with good test cube detection that is based on test generation information recorded by the ATPG tool. It may not be accessed easily from the user-side. Therefore, this test points selection approach can be time-consuming and tool dependent. We propose a simpler optional approach to select locations for the observation test points using a fault-propagation analysis.

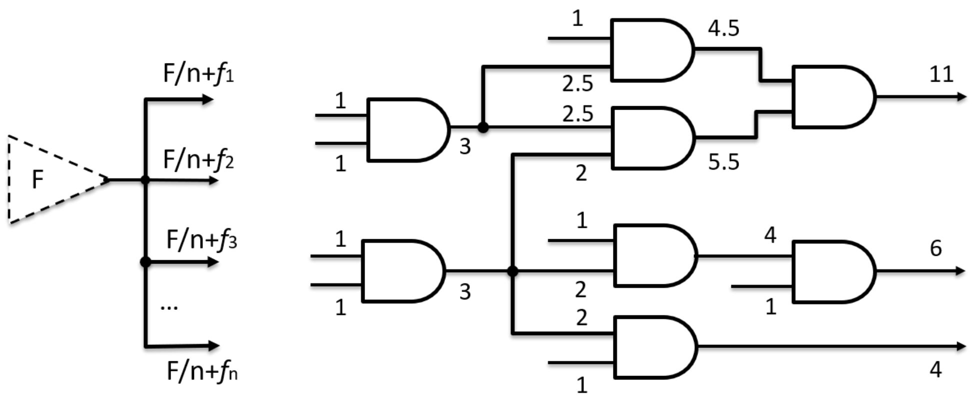


Figure 4.7 Fault propagation analysis

With the same idea of selecting suitable scan cells that can cover more test cubes, as well as detect additional faults during staggered pattern simulation, instead of learning the information from a complete ATPG run, we can perform a structural tracing based on the fault list to obtain a fault propagation profile for the scan cells. The propagation analysis is shown in Figure 4.7 and can be described as follows:

1. For each fault in the fault list, we assign a value “1” at the fault site for counting the forward propagation of the fault.

2. We divide the value evenly on every fan-out stem encountered (modeling random fault propagation decisions with equal probabilities) and propagate the fraction towards every fan-out branch. The calculation for a total number of F faults propagating through a fanout stem with n fanout branches is shown in Figure 4.7(a) and Figure 4.7(b) shows an example of the fault propagation analysis.

3. We record the fault propagation value received by each scan cell.

4. After traversing the entire design, we sort the scan cell list by the fault propagation value and select a predefined number of scan cells with highest fault propagation values as locations of our observation test points.

Note that, the fault list used for this approach is not the entire fault list of a design. We observe that for most designs, a large proportion of the faults are some easy-to-detect faults that can be detected by the simulation of a few test patterns at the beginning of an ATPG process. These faults act like “noise” to our fault propagation analysis and detecting these faults by our observation test points would not help to reduce the final pattern count. Therefore, we prune the fault list by removing those faults detected at the early-stage of an ATPG process and use the reduced fault list (also called as tail-end fault list) for our observation test points selection. Although obtaining such a tail-end fault list may still require an ATPG run, this ATPG process normally stops at an early stage and has a much shorter run time than a complete ATPG run.

4.5 Experimental Results

The performance of the staggered ATPG working with the capture-per-cycle observation test points, has been verified experimentally on 10 large industrial designs with on-chip EDT-based test compression [7]. They represent different design styles, different scan methodologies, and mirror the latest technology nodes. The basic data regarding these circuits, such as the number of gates, the fault population, the number of scan cells, the number of EDT channels, the number of chains, and the length of the longest scan chain are listed in Table 4.1. Although longer scan chains may help to get more staggered test patterns, we only choose reasonable scan chain length for each test case.

Table 4.1 Circuit characteristics

	Gates	Faults	Scan cells	Channels	Chains	Chain length
D1	1.69M	4.99M	87K	4	400	219
D2	1.19M	4.48M	72K	16	400	189
D3	2.08M	7.17M	143K	4	400	371
D5	1.04M	1.90M	57K	4	400	146
D6	1.38M	3.78M	93K	10	169	555
D7	2.53M	4.93M	206K	10	374	558
D8	1.09M	2.94M	75K	10	141	546
D9	4.04M	8.76M	199K	10	364	548
D10	3.35M	15.15M	331K	4	400	829

The experimental results for stuck-at faults are presented in Table 4.2. Columns "Pattern count" gather the key data for both examined scenarios – the number of ATPG-produced test cubes in line with the conventional scenario and its staggered counterpart. In all experiments reported in this section, the staggered ATPG results were obtained after inserting observation test points, as discussed in Section 4.4, i.e., by using the method working with the fault propagation profile, and the fault list is pruned to a 90% tail-end fault list (removing the first 90% of the faults that are detected by a few patterns in the early stage of an ATPG run). The total number of test points is kept to below 2% of the entire scan cell population (see column “Observation points” of the table). This percentage is an industry-wide accepted standard. As can be seen, the staggered ATPG yields a visible pattern count reduction in all examined cases. It varies from 1.2x to 3.0x for otherwise similar test coverage numbers provided by the conventional ATPG.

Table 4.2 Experimental results (stuck-at faults)

	Conventional ATPG		Pattern reduction (x)	Staggered ATPG with OPs		
	Coverage [%]	Pattern count		OPs	Coverage [%]	Pattern count
D1	95.89	11,523	2.98	1,600	95.89	3,870
D2	96.45	2,010	1.46	1,400	96.46	1,377
D3	97.24	7,299	1.69	2,800	97.26	4,325
D4	92.92	8,697	1.97	1,100	92.92	4,422
D5	97.68	8,235	1.58	1,800	97.68	5,228
D6	97.64	1,838	1.85	4,000	97.64	993
D7	95.64	2,127	2.05	1,500	95.65	1,036
D8	99.29	1,002	1.17	4,000	99.29	860
D9	96.54	3,129	1.86	6,000	96.54	1,682
D10	99.86	1,149	1.39	1,500	99.86	824
Average	96.92	4,701	1.80	2,570	96.92	2,462

Table 4.3 Experimental results (transition faults)

	Conventional ATPG		Pattern reduction (x)	Staggered ATPG with OPs		
	Coverage [%]	Pattern count		OPs	Coverage [%]	Pattern count
D1	91.22	38,493	5.68	1,600	91.23	6,779
D2	94.31	5,090	1.24	1,400	94.34	4,099
D3	91.87	16,473	1.49	2,800	91.95	11,042
D4	89.29	17,085	2.05	1,100	89.31	8,326
D5	94.32	21,132	1.68	1,800	94.32	12,566
D6	94.91	2,210	1.97	4,000	94.94	1,120
D7	92.64	3,368	3.61	1,500	92.74	932
D8	95.24	1,417	1.28	4,000	95.25	1,104
D9	95.78	10,528	2.17	6,000	95.79	4,859
D10	98.79	5,471	1.50	1,500	98.80	3,639
Average	93.84	12,127	2.27	2,570	93.87	5,447

In Table 4.3, we present the experimental results for transition faults for the same test cases with the same observation test points we used for stuck-at faults in Table 4.2. We may consider the scan cells' data at every two adjacent shift cycles as a pair of launch-and-capture tests for the two-cycle transition test, as introduced in Section 1.2.2. In this case, we can apply our staggered ATPG to two-cycle tests, like transition faults, by treating the scan shift data at every clock cycle as an individual LOS test. Although handling the at-speed application of transition patterns can be a difficult task, by applying the idea of staggered ATPG, we can get a good pattern count reduction (an average of 2.27x for transition faults shown in Table 4.3). And this may lead to the test generation of staggered two-cycle test patterns for stuck-open faults, which require two test patterns that do not need to be applied at-speed. It may also be possible to gain some “free” test coverage for these faults that require two-cycle test patterns, during the test application of staggered test patterns that are generated only for stuck-at faults.

It is worth noting that fault simulation used in the experiments does not account for an unlikely event of aliasing that may occur when fault effects are masked within scan chains driven by the observation test points. The likelihood of such an event is fortunately extremely small [96], since these scan chains form finite memory devices, where after several clock cycles (depending on a fault injection site) an error is shifted out. Moreover, missed faults remain ATPG targets.

4.6 Conclusion

In this chapter, we present a staggered ATPG working synergistically with the capture-per-cycle observation test points. This approach generates highly mergeable deterministic test patterns and detects many additional faults through staggered pattern fault simulation. The observation test points are inserted at the scan cell inputs based on the fault propagation profiles. This process has a minimum impact on a design, compared to other test point insertion techniques. At the same time, test responses captured every clock cycle by means of the observation points visibly improving the fault observability. Compared to the method introduced in [59], our approach develops a constrained ATPG that produces effective test-per-clock patterns and requires fewer observation points (no more than 2% of the design's total scan cell count), to achieve good test pattern reduction. Experimental results obtained for large industrial designs demonstrate – on average – a 1.8x pattern count reduction for stuck-at patterns while preserving the original fault coverage and may also be applicable to two-cycle patterns such as transition faults.

CHAPTER 5
DETERMINISTIC STELLAR BIST FOR IN-SYSTEM TEST

Both the TPI technique introduced in Chapter 3 and the staggered ATPG presented in Chapter 4 focus on reducing the test set size for ATPG-produced deterministic patterns. However, pseudorandom patterns still play a significant role for in-system test. As it is becoming increasingly difficult for conventional test solutions that handle pseudorandom patterns to achieve high test quality for advanced in-system test, these pseudorandom-pattern-based solutions can be replaced by the stored pattern test method, by obtaining desired test stimuli through stored deterministic test patterns, and thus it is important to produce a minimal but effective test set for stored pattern test solutions.

In this chapter, we propose the Stellar BIST approach. While it inherits some underlying principles of its predecessors [5], [94], including primarily a concept of deterministic parents and their derivatives, the new scheme, in vivid contrast to the earlier techniques, produces derivatives of a given encodable parent pattern, by complementing its bit slices several times rather than just once, as done in [94], during a single test pattern application. An additional mechanism skews multiple complements to enrich population of patterns. As a result, the scheme offers flexible trade-offs between test application time and test data volume on a scale that is not matched in earlier test data compression schemes. Consequently, Stellar BIST outperforms the state-of-the-art on-chip test compression solutions in terms of processing time needed to arrive with the required test cubes, the corresponding fault crediting results, and eventually compressed test patterns.

This chapter is organized as follows. In Section 5.1, we introduce the predecessor of this work and the basic principle of test clusters used in our work. The new Stellar BIST test scheme and the implementation flow are proposed in Sections 5.2 and 5.3. Experimental results are discussed in Section 5.4, and conclusions are given in Section 5.5.

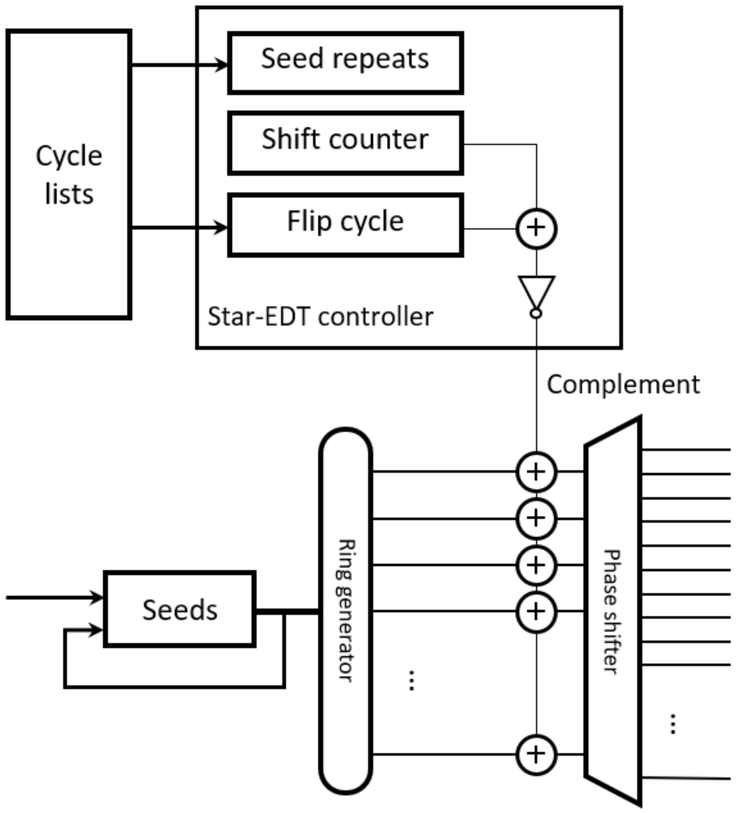


Figure 5.1 STAR-EDT architecture [94]

5.1 STAR-EDT Architecture

The proposed test solution of this chapter can be considered as an evolutionary step, that builds on the Star-EDT scheme [94] discussed in detail in this section for the sake of reference. A test set produced within this framework, is comprised of test vector clusters. Each cluster consists of a single ATPG-produced and EDT-encodable parent pattern and several children derived from the parent, by complementing all bits of a scan shift cycle. A careful selection of these cycles may produce a group of valuable stimuli. These vectors, due to a slight departure from the original parent sequence, detect many faults that otherwise would need to be targeted separately.

Figure 5.1 sketches the Star-EDT architecture. An n-bit ring generator and a phase shifter make up a continuous flow sequential decompressor (as introduced in Section 1.1.3). The ring generator outputs can be inverted by means of additional XOR gates controlled by a single complement signal. Since the phase shifter feeds every scan chain through a 3-input XOR gate [7], inverting all phase shifter inputs complements all its outputs, i.e., it causes all bits of a given cycle to flip. Whenever the content of a flip cycle register matches a scan shift counter (Figure 5.1), a Star-EDT controller asserts the complement signal for a single cycle period. This match only occurs, if the resultant test pattern can detect additional faults. Hence, parent patterns are assigned lists of effective cycles, so that the controller can apply every parent pattern repeatedly, each time complementing – once per pattern application – all bits of a single shift cycle. A seed repeats register stores the number of repetitions of the same parent pattern. Both – the flip cycle and the seed repeats registers – receive new content from a list of effective cycles kept in the cycle lists memory.

As an example, consider seed *s* that yields a parent pattern *p*. Let p_a, p_b, and p_c be derivatives of *p* obtained by complementing all scan cells of p for time frames a, b, and c, respectively. In this case, the seeds memory stores seed *s*, while the cycle lists memory contains the binary-coded values of a, b, and c. The controller starts by disabling the complement signal and loading scan chains with the original pattern *p*. Next, the value of a is loaded to the flip cycle register, seed *s* is decompressed again, and the resultant test pattern p_a is applied to a circuit. The same is repeated for b and c. The whole procedure then continues for other seeds.

5.2 Stellar BIST Test Scheme

There are two findings that our solution is based on. First, certain clusters of test vectors can detect many related faults. A cluster typically consists of a parent pattern and several children vectors derived from it through simple transformations. Consider a single test pattern that detects stuck-at-0 and stuck-at-1 faults at the outputs of a circuit as shown in Figure 5.2. This test sets all inputs of an AND super-gate to 1 and all inputs of an OR super-gate to 0 (a super-gate is a structural implication abstract representing a circuit block whose functionality is equivalent to an AND/OR gate). Now, some of the transformed patterns that detect successive stuck-at-1 and stuck-at-0 faults at the same inputs are illustrated at the bottom of the figure. Clearly, the original test vector is a good parent pattern, since it allows one to derive additional tests through systematic (every eight inputs in this example) bit flipping of its respective components.

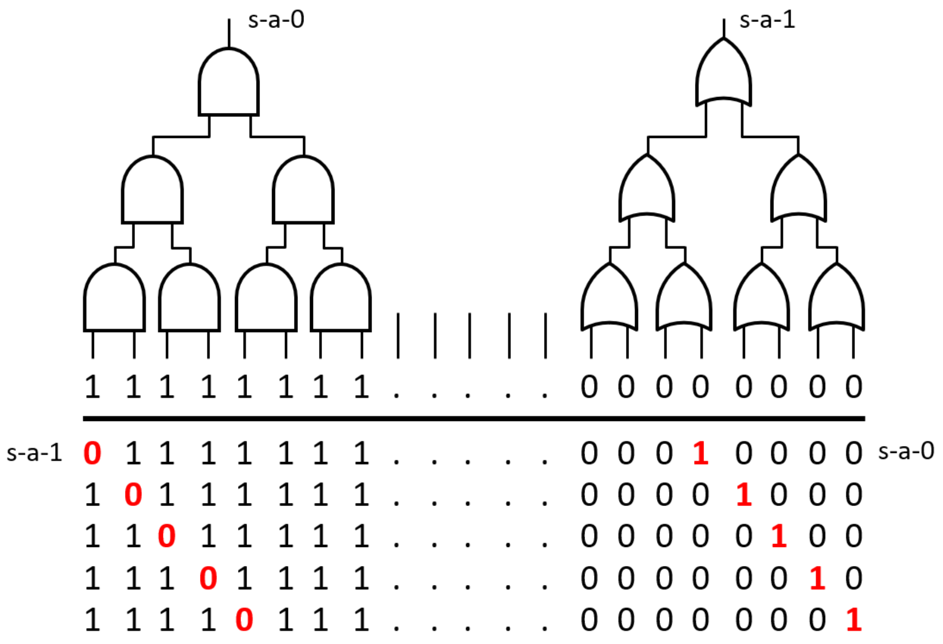


Figure 5.2 Test clusters with multiple complements

It is useful to also consider a circuit that is depicted in Figure 5.3. The first pattern detects a stuck-at-0 fault on the output of the super-gate comprising two fan-out-free regions (FFRs) made of AND gates. As can be seen, this pattern is a suitable choice, as a parent vector for both FFRs. Derivates of the parent are obtained by complementing any of its input bits. This allows the detection of all stuck-at-1 faults at the inputs of the super-gate constructed through multiple FFRs. In the meantime, other bits of this vector can serve as additional flipping choices, that potentially detect some other faults without blocking the observation of those already-activated faults.

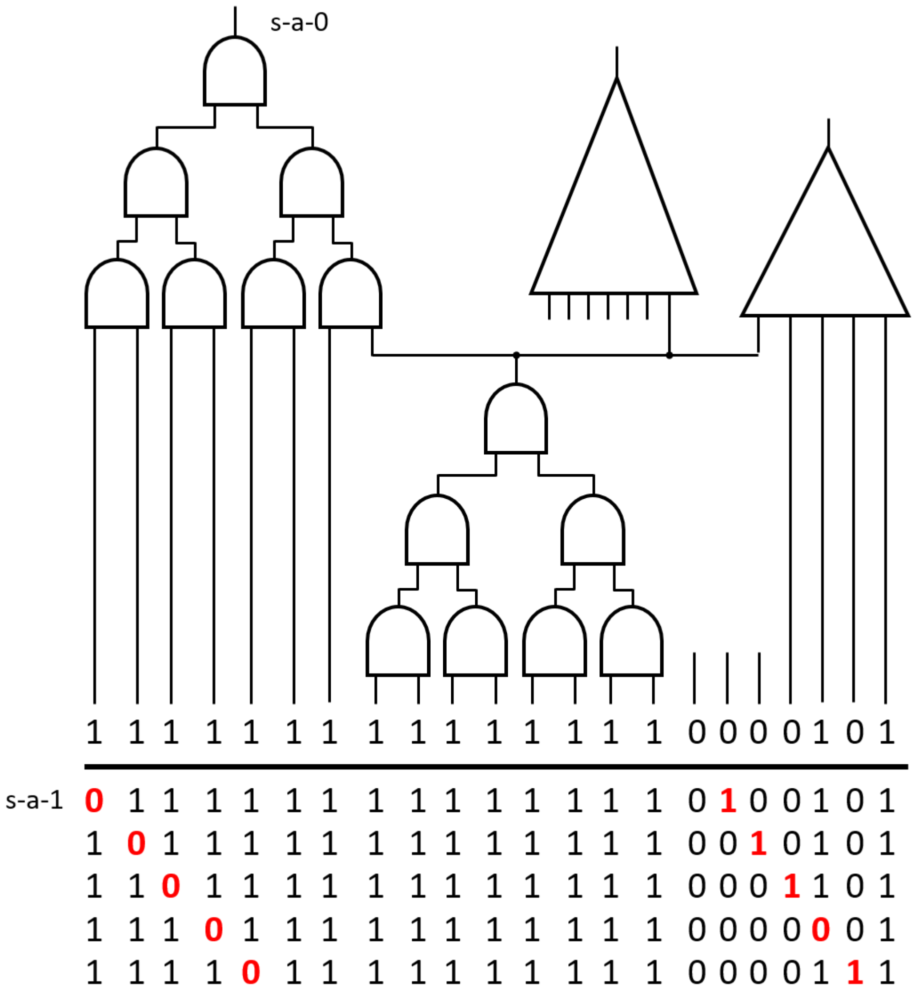


Figure 5.3 Parent pattern and its derivatives

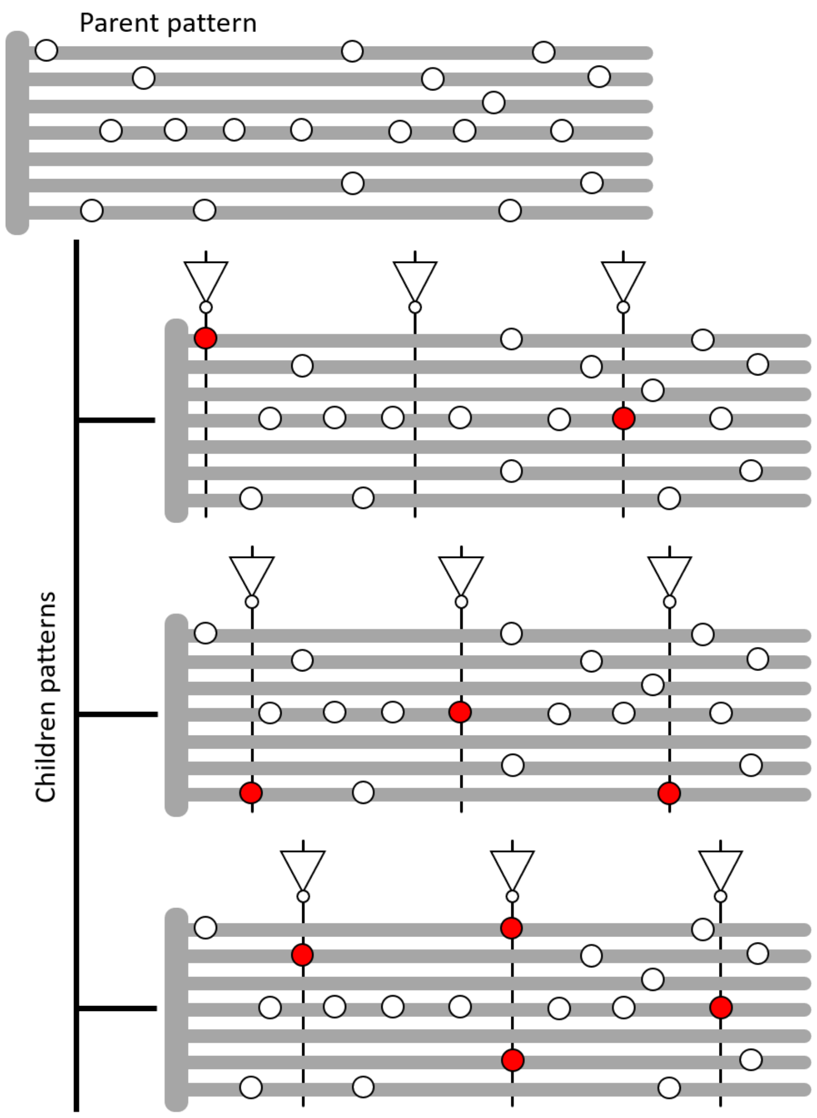


Figure 5.4 Stellar BIST test cluster

It also appears that typically a very few scan chains host specified bits of a test cube. Although this may depend on a scan chain stitching method and the resultant scan architecture, since forming scan chains is guided by a design layout as well as clock and power distribution networks, it nevertheless turns out to be the case across many industrial designs, as reported in [94].

As is the case with Star-EDT [94], test patterns employed by Stellar BIST comprise clusters of vectors. Each cluster consists of a single encodable ATPG-produced parent pattern and several children patterns derived from the parent. What contrasts the new scheme with the previous approach, is a method to create all parent's derivatives. Instead of complementing bits of just a single scan shift cycle, Stellar BIST complements bits every k shift clock cycles during a single test pattern application. This new approach has been fostered by observations, like those shown in Figures 5.2 and 5.3. Therefore, if k = 32, then the first child pattern is obtained by inverting bits belonging to slices (time frames) 0, 32, 64, and so forth. Subsequently, the second child pattern is generated by inverting bit slices 1, 33, 65, etc. The next patterns are obtained in a similar manner by having complemented bit slices beginning with shift cycles 2, 3, all the way to 31. However, some of these patterns may not be able to detect any new faults not yet covered. These test patterns are simply skipped. Given a parent pattern, the resultant cluster of patterns is illustrated in Figure 5.4. If k is chosen to be a power of 2, then it eases the design constraints, as shown in the next paragraph. It is worth observing that having specified bits in a few scan chains and complementing bits of time frames that follow the pattern of Figure 5.2, will most likely cause only target specified bits of a test cube to be affected.

Figure 5.5 is a block diagram of Stellar BIST logic. A CPU, ATE, or any other appropriate on-chip device or a device external to the chip but part of the system, may store test data used to control this circuitry. A test set is represented by a clock-between-complements (CBC) vector and test pattern clusters. This CBC vector defines the value k of a distance between successive time frames being complemented. Since the content of the CBC register is used to gate a down counter producing the actual complement signal, k is binary-coded as $2^d - 1$, where $d = \log_2 k$. For example, an 8-bit CBC register indicating that the complements are done every k = 16 shift cycles, assumes the form 00001111. Every pattern cluster, in turn, comprises an encoded parent pattern (seed) and a binary vector, indicating which derivatives of the parent test are deployed. If only derivatives 1, 4, 5, and 10 should be produced for k = 16, then the child selection vector becomes equal to 0000010000110010, i.e., with bits b_1, b_4, b_5, and b_{10} asserted (the least significant bit is on the right-hand side).

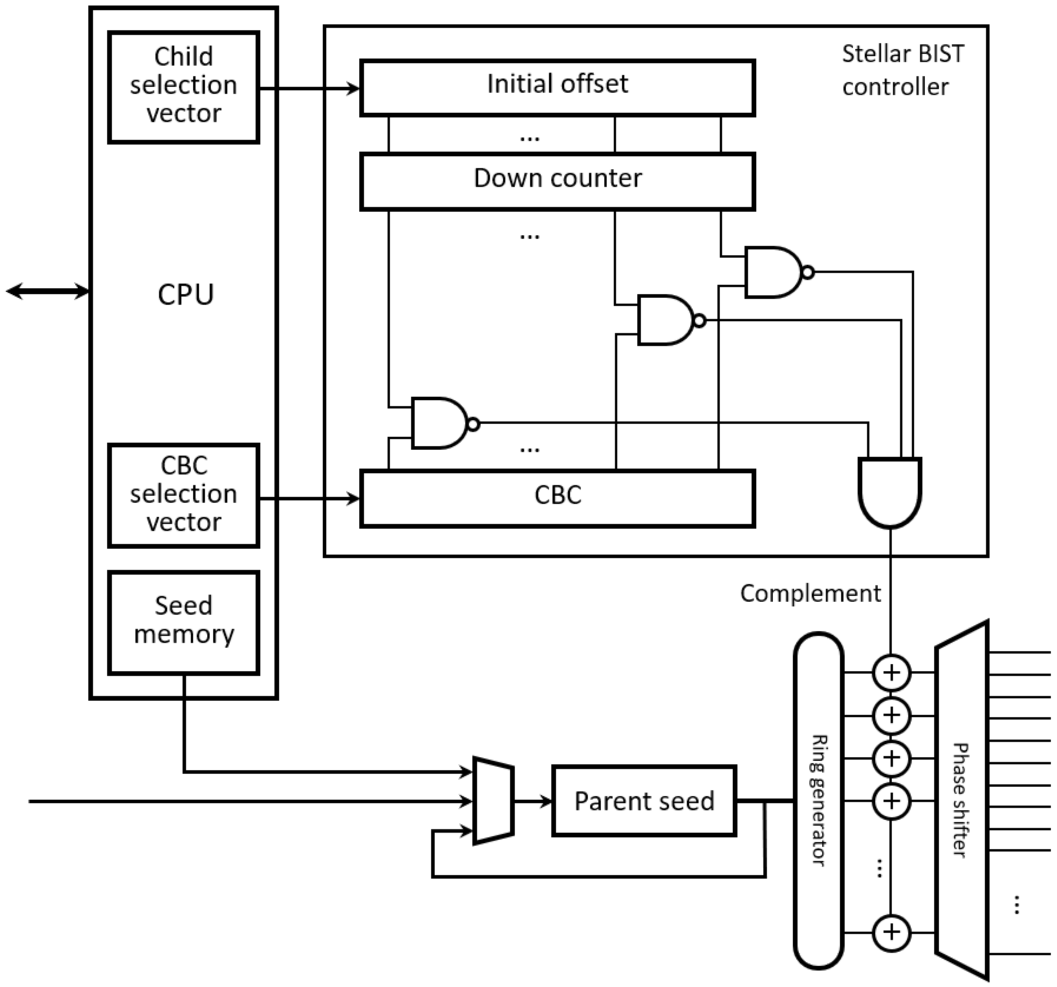


Figure 5.5 Stellar BIST architecture

The initial offset register is uploaded based on the content of the child selection vector in such a way that if bit b of this vector is asserted, b = 0, 1, ..., k - 1, then the register gets the binary-coded value of b. The initial offset is subsequently used to initialize a down counter, which works synchronously with the scan shift clock. By observing the counter and detecting the all-0 sequence on its least significant d bits (determined by the CBC value, as discussed in the previous paragraph) one can decide when to yield the complement signal. For n-bit registers, this is achieved by n NAND gates whose outputs' product is finally delivered by an n-input AND gate.

Given the main components of the Stellar BIST controller, the application of a single pattern cluster proceeds as follows. After applying the original parent pattern (with the complement signal disabled) and setting up the CBC register, whose bits indicate which part of the down counter is taken into account, the CPU attempts to load the initial offset register, which is subsequently used to initialize the down counter. Let the offset be set to 3. A very few first states of the counter will, therefore, be the following: 3, 2, 1, 0 (here the complement signal is going to be asserted), $2^n - 1$, $2^n - 2$, and so forth. Once the least significant d bits become the all-0 vector again, the next complement signal is delivered. Note that all complements are phase-shifted with respect to the first scan shift clock pulse by three cycles in this case. Once the entire child pattern is applied, the initial offset register is reloaded with a phase shift corresponding to the next valuable child pattern, and the process repeats with the parent pattern seed circulating within the parent seed register, until all the desired child patterns are generated. This is now the time to apply the next parent pattern with no complements, and then its derivatives, as described above. The former is done through a combination of reloading the parent seed register and disabling the complement wire driven by the AND gate.

5.3 Implementation Flow

The degree of test data compression attainable by Stellar BIST is a result of storing only parent pattern seeds and coordinates of the associated flip cycles. It allows the placement of all relevant test data on a chip. This makes the approach compatible with deterministic BIST. In the following paragraph, we present a basic test flow that results in the highest compression ratios, the most aggressive test coverage ramp-ups, and the most efficient usage of test memories.

A preprocessing step of our test flow consists of a circuit structural analysis. It identifies all super-gates (discussed in Section 5.1) within the design and the fan-out free regions (FFRs) that they belong to. Furthermore, SCOAP testability measures (as introduced in Section 2.2.2) are computed and recorded. A complete set of deterministic test patterns is subsequently created by running ATPG. This central step is essentially a framework that produces and verifies successive parent patterns and their derivatives iteratively, a given number of parents at a time. Typically, a single and compressible parent pattern is going to be a result of merging of ATPG-produced test cubes obtained for properly selected faults. The fault selection procedure randomly picks a fault *f* from the entire fault list. However, *f* usually does not become the direct ATPG target. Instead of *f*, a fault at the output of a super-gate that hosts *f* is selected. As a result, there is a better chance of getting the most suitable parent pattern that can subsequently be deployed to yield derivative test patterns detecting all target faults, within a super-gate or the corresponding FFR (compare Figure 5.3). Moreover, the SCOAP values recorded earlier can also be used to guide the selection process, by choosing a fault with the highest sum of controllability and observability metrics within FFR hosting *f*. Every ATPG-produced stimulus now becomes a kernel of a test cluster also comprising its children patterns, obtained due to multiple complements of the parent pattern by using a user-defined period between successive complements, as detailed in Section 5.2.

Once a given number of parent patterns are generated, the corresponding test clusters are now individually fault simulated with the fault dropping enabled. Note that this process is significantly less CPU-intensive than the previous approach of [94], since derived patterns are fairly restrained, and their number can easily be controlled by a user (setting a proper CBC value). Moreover, every cluster is further revised in such a way that only effective child patterns, i.e., those that detect some faults, are retained. As a result, every parent pattern is now assigned a binary-coded child selection vector, and the original parent patterns are recorded as seeds. The procedure presented above is repeated, until the complete test coverage target is reached.

In order to further reduce the total number of test patterns, a pattern reordering procedure is applied to all effective patterns (for both parents and children) obtained in the previous steps. These test patterns are first sorted in the descending order, with respect to the number of faults they detect, based on information captured during a simulation process). In the following steps, a fault dropping simulation repeatedly determines faults detected by successive patterns, beginning with a test pattern that features the largest fault detection count. Therefore, this phase basically implements a reverse order fault simulation that reveals faults not yet detected by the previously examined patterns. It also updates child selection vectors for those clusters whose members were removed. It is worth noting that the entire test clusters may be deleted during this phase, if none of their patterns, including the parent, detect any new faults after reordering the test vectors. The postprocessing method, as discussed in [6], can also be served as potential solutions to further compact the obtained stored pattern set.

As a result, the test application time is strictly dependent on the number of test clusters and the number of their final components. The algorithm that is presented virtually produces a minimal set of test clusters that allow one to flexibly trade-off test coverage, test application time, and the size of the on-chip test memories. The algorithm can be summarized as follows:

```
while fault list F is not empty do
        generate a given number of parent patterns
        produce all children patterns
        fault dropping simulate the current test clusters on F
        save effective patterns
        update F by removing detected faults
run pattern reordering and save effective patterns
```

5.4 Experimental Results

Stellar BIST has been verified by conducting experiments on 12 industrial designs, all of them with on-chip EDT-based test compression. They represent different design styles and scan methodologies. The basic data regarding the designs, such as the number of gates, number of scan cells, scan architecture, and the total number of stuck-at faults are listed in Table 5.1. And the results of super-gate analysis for each design are shown in Table 5.2.

Table 5.1 Circuit characteristics

	Gates	Scan cells	Chains	Chain length	Faults
D1	1.19M	72.3K	400	181	4.41M
D2	2.07M	148.0K	400	371	7.17M
D3	3.32M	326.0K	400	814	15.03M
D5	1.68M	86.0K	400	215	4.97M
D6	1.04M	57.0K	400	144	1.89M
D7	1.38M	93.2K	168	555	3.78M
D8	2.53M	206.3K	370	558	4.94M
D9	4.83M	325.9K	598	545	13.41M
D10	4.04M	199.7K	364	549	8.79M
D11	2.59M	154.0K	1,200	129	8.97M
D12	2.29M	252.4K	490	516	9.95M

Table 5.2 Super-gate structural analysis

	Gates	Super-gates	Ratio	Max size
D1	766,145	173,854	4.41	532
D2	881,012	222,974	3.95	1,528
D3	1,465,002	381,808	3.84	238
D5	885,417	227,477	3.89	683
D6	563,694	155,060	3.64	442
D7	938,775	203,219	4.62	443
D8	1,377,846	335,241	4.11	251
D9	2,828,442	749,333	3.77	371
D10	2,310,490	540,740	4.27	112
D11	1,531,011	366,255	4.18	990
D12	1,226,935	269,052	4.56	750

Table 5.3 Stored pattern (SP) reduction and test time (Time) increase (stuck-at)

	Baseline patterns @90%	CBC=8		CBC=16		CBC=32		CBC=64	
		SP (x)	Time (x)	SP (x)	Time (x)	SP (x)	Time (x)	SP (x)	Time (x)
D1	640	2.50	2.90	3.33	3.90	5.00	5.90	5.00	8.89
D2	960	3.31	1.47	5.05	2.13	5.00	2.80	5.00	4.47
D3	640	2.01	2.70	3.33	4.30	3.33	4.50	5.00	10.90
D4	2,432	2.12	2.26	3.46	3.43	4.76	4.81	4.76	5.67
D5	9,788	1.14	1.52	1.53	2.05	2.55	2.50	4.63	2.96
D6	448	1.77	2.43	2.33	4.43	3.50	6.43	3.50	9.67
D7	80	2.50	2.20	2.50	3.20	2.50	4.80	2.50	6.40
D8	384	3.00	2.83	3.00	3.16	3.00	4.83	3.00	6.33
D9	256	4.00	1.75	4.00	2.75	4.00	5.00	4.00	9.25
D10	11,178	1.38	1.29	1.92	2.51	2.04	3.49	2.15	3.51
D11	704	3.67	2.27	5.50	2.64	5.50	3.00	5.50	4.18
D12	1,792	2.15	2.92	3.12	3.49	4.67	5.12	5.60	7.50
Avg	2,442	2.46	2.21	3.26	3.17	3.82	4.43	4.22	6.64

As discussed in Section 5.2, super-gate analysis locates a certain number of gates and groups them into corresponding super-gate structures. Parent patterns are generated targeting faults at super-gate outputs prior to other fault locations and faults within each super-gate are likely to be detected by the derivative patterns of these parent patterns. In this way, each design can be treated as a simplified version that is formed by some super-gates with a reduced fault list. As shown in Table 5.2, for each design, a large number of gates (column "Gates") can be grouped into fewer super-gate structures (column "Super-gates") and column "Ratio" may reflect the fault count reduction (or the average size of a super-gate) after the super-gate transformation. The last column "Max size" includes the size of the largest super-gate for each design.

Table 5.4 Stored pattern (SP) reduction and test time (Time) increase (transition)

	Baseline patterns @85%	CBC=8		CBC=16		CBC=32		CBC=64	
		SP (x)	Time (x)	SP (x)	Time (x)	SP (x)	Time (x)	SP (x)	Time (x)
D1	4,156	1.59	2.72	2.33	4.08	3.61	5.37	4.64	7.10
D2	12,608	1.53	2.34	2.14	3.58	3.13	4.98	4.11	6.21
D3	3,967	1.50	4.23	1.82	6.25	2.38	8.90	3.26	14.01
D4	14,206	1.52	1.66	2.29	2.93	3.76	4.14	5.29	5.25
D5	24,216	1.06	1.32	1.13	1.79	1.55	2.18	2.01	2.44
D6	2,880	1.18	2.77	1.41	4.58	2.65	6.91	3.75	9.86
D7	640	2.50	2.90	3.33	4.20	3.33	4.90	5.00	10.74
D8	4,222	1.41	3.46	1.61	4.91	1.89	7.10	2.28	9.68
D9	448	3.50	1.71	3.50	2.00	3.50	2.29	7.00	7.99
D10	16,092	1.53	1.79	2.83	2.72	3.93	3.51	4.34	4.63
D11	1,280	2.50	3.25	4.00	3.60	5.00	4.05	6.67	7.55
D12	4,032	2.17	3.15	3.32	3.78	4.85	4.65	6.30	7.01
Avg	7,396	1.83	2.61	2.48	3.70	3.30	4.92	4.55	7.71

Other results presented in this section were obtained for stuck-at (Table 5.3) and transition (Table 5.4) faults. The second column of Table 5.3 lists baseline pattern counts (PC) necessary to achieve 90% stuck-at test coverage (TC) by means of a conventional EDT-based compression [7], whereas the second column of Table 5.4 reports baseline pattern counts for 85% transition faults reference coverage obtained through a launch-off-capture (LOC) approach. These two values were selected as widely-deployed test quality standards responding to the ASIL D safety requirements. The ISO 26262 demands that 90% of static permanent faults are detected even when ASIL is set to level B. On the other hand, a 90% detection threshold is needed by ASIL D for latent faults. These are targets of particular importance for in-system tests executed through the life-time of automotive ICs. Although the standard does not mandate a coverage for any particular fault model, the industry is primarily targeting 90% stuck-at fault coverage during in-system testing. In some cases, transition test patterns are also used, but the target test coverage for such patterns is much lower.

Besides the second column, both tables consist of four main vertical segments corresponding to four different values of the clock-between-complements (CBC) parameter: 8, 16, 32, and 64. For each design, we report the following statistics: a stored pattern (SP) reduction, i.e., a ratio of the number of baseline patterns and the number of parent patterns, and a test time increase (Time) determined by dividing the total number of test patterns by the number of baseline patterns. Consider, for example, design D2. It needs 290 parent patterns and effectively employs 1,408 test patterns to reach the 90% stuck-at test coverage level for CBC = 8. The conventional sequential test compression scheme needs 960 test patterns to achieve the same coverage. Therefore, the observed reduction of stored patterns is 960 / 290 ≈ 3.31x, whereas the test application time increases, in this case, 1,408 / 960 ≈ 1.47x. As can be seen, Tables 5.3 and 5.4 illustrate one of the advantages of Stellar BIST – the ability to significantly reduce the number of patterns that need to be saved as seeds, while preserving the attainable test coverage. Moreover, Table 5.5 presents the memory usage reduction for both stuck-at and transition patterns. The memory usage, given a CBC value, is calculated by adding the total number of bits required for child selection vectors to the total number of bits for EDT compressed parent patterns. For the case of D2 and CBC = 8, the total size of child selection vectors is 8 · 290 = 2,320 bits (by multiplying the value of CBC and the total number of parent patterns).

Table 5.5 Memory reduction for stuck-at (SAF) and transition (TDF) faults

	CBC=8		CBC=16		CBC=32		CBC=64	
	SAF (x)	TDF (x)	SAF (x)	TDF (x)	SAF (x)	TDF (x)	SAF (x)	TDF (x)
D1	2.49	1.58	3.32	2.31	4.95	3.57	4.90	4.54
D2	3.29	1.52	5.00	2.12	4.90	3.07	4.80	3.95
D3	2.00	1.50	3.32	1.81	3.30	2.36	4.91	3.20
D4	2.11	1.51	3.40	2.25	4.60	3.64	4.46	4.95
D5	1.12	1.04	1.50	1.10	2.43	1.48	4.22	1.84
D6	1.77	1.18	2.33	1.40	3.48	2.63	3.46	3.71
D7	2.50	2.50	2.49	3.32	2.49	3.31	2.47	4.94
D8	3.00	1.40	2.99	1.61	2.98	1.87	2.97	2.25
D9	3.99	3.50	3.99	3.49	3.98	3.48	3.96	6.92
D10	1.37	1.53	1.90	2.80	2.00	3.86	2.07	4.19
D11	3.65	2.49	5.46	3.97	5.42	4.93	5.34	6.47
D12	2.14	2.16	3.07	3.27	4.53	4.70	5.28	5.94
Avg	2.45	1.83	3.23	2.45	3.76	3.24	4.07	4.41

As can also be noticed from Table 5.5, Stellar BIST requires, on the average over all examined designs, 2.45, 3.23, 3.76, and 4.07 times less memory than the baseline scheme to reach the reference 90% test coverage for CBC = 8, 16, 32, and 64, respectively. For transition faults, the corresponding numbers are as follows: 1.83, 2.45, 3.24, and 4.41 times. It is also of interest to compare the presented results with what logic BIST deploying test points can achieve, under otherwise identical conditions. We have run experiments for designs D1, D3, D5, and stuck-at faults. The results are as follows:

• D1: 200,000 pseudorandom patterns, 83.55% test coverage, 3,000 test points (approximately 4% of all memory elements); as can be seen after applying 200K tests the target coverage is not yet reached,

• D3: 35,520 pseudorandom patterns, 90% test coverage, 6,000 test points (approximately 2% of all memory elements); LBIST needs 20.5 times more patterns to match the target test coverage,

• D5: 156,288 pseudorandom patterns, 90% test coverage, 3,000 test points (approximately 5% of all memory elements); LBIST needs 10.5 times more patterns to match the target test coverage.

How large the CBC distance is actually needed to achieve desired test coverage with the best reduction of a parent pattern count can also be retrieved from Tables 5.3 and 5.4. Consider the results for stuck-at faults and design D4. As can be easily verified, the parent pattern count reductions obtained for CBC = 8, 16, 32, 64 are 2.12, 3.46, 4.76, and 4.76, respectively. A similar trend can be observed for the remaining test cases, where increasing the value of CBC, and thus potentially increasing the number of children patterns, due to the increased number of complements, results in entering the area of diminishing returns: there are no further parent pattern count reduction gains, despite increasing the value of CBC. Interestingly, for design D9, even the value of 8, i.e., the smallest CBC examined, suffices to ensure the best reduction of stored patterns. This finding is of special interest, since by increasing the value of CBC, one may visibly increase test application time, due to more children patterns needed to secure target test coverage. It can also be easily observed in the tables. Although similar results have been obtained for transition faults, it is also evident that in this case, the parent pattern count reduction is monotonically increasing with the increasing value of CBC. This trend is, however, accompanied by a similar increase in test time. Possible trade-offs that must be considered here are better pronounced by additional experimental results presented in the following paragraph.

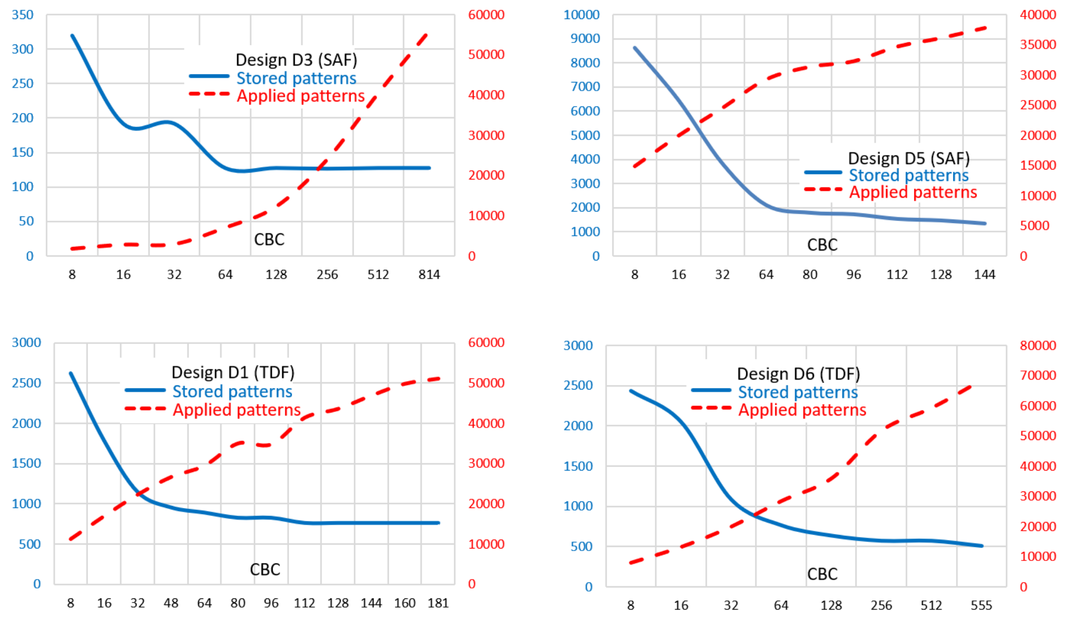


Figure 5.6 Stored patterns vs. applied patterns

Figure 5.6 illustrates how the number of patterns to store (in the form of decompressible seeds) drops, and how the number of patterns to apply (including both parents and their derivatives) rises with the increasing distance between successive complements (represented by the variable CBC). Here we use a higher granularity of CBC values than that of Tables 5.3 and 5.4. The results are for designs D3 and D5 (stuck-at faults), as well as D1 and D6 (transition faults). In each case, the solid curve represents stored patterns, whereas the dashed one represents patterns that need to be applied. All of the diagrams presented clearly demonstrate that while increasing the CBC value may lead to a reduction of parent patterns (and hence relatively small on-chip test memories), this phenomenon is associated with a visible increase in the number of test patterns that need to eventually be applied, or alternatively in the increased test time.

It is also worth noting that the results presented can be used to compare Stellar BIST with the original Star-EDT scheme [94]. The number of stored and applied patterns for Star-EDT, correspond to points where CBC (here the abscissa) matches the size of the longest scan chains. In such a case, there is just a single complement per test application. The longest scan chains for designs D3, D5, D1, and D6 (Figure 5.6) feature 814, 144, 181, and 555 cells, respectively. As can be seen, designs D3 and D5 (stuck-at faults) need 128 and 1,344 stored patterns, while applying 55,850 and 37,800 patterns, respectively. Similarly, designs D1 and D6 (transition faults) work with 768 and 512 stored patterns and apply 51,118 and 68,436 tests. It is indisputable that compared to Star-EDT, Stellar BIST provides a wide-range of options, allowing users much better and flexible trade-offs between test data volume and test application time. As discussed earlier, it is also confirmed here that increasing the value of CBC (including setting it to the scan chain length) results in diminishing returns and sub-optimal choices.

5.5 Conclusion

Since automotive integrated circuits have become one of the key drivers of innovation in test, in this chapter, we introduce a next generation test compression scheme, particularly suitable for in-system automotive test applications. It exclusively deploys compressed ATPG-produced test patterns and their on-chip-generated derivatives. Given an encodable parent pattern, the latter ones are produced by complementing its bit slices several times during a single test pattern application. An additional mechanism skews multiple complements to increase the number of valuable tests. As a result, the presented Stellar BIST outperforms earlier state-of-the-art sequential test compression techniques in many respects. It offers very flexible trade-offs between test application time and test data volume. This degree of flexibility was virtually impossible to achieve in earlier test data compression schemes, where the optimization scope was rather limited. Furthermore, it allows a faster test coverage ramp-up, requires less memory to achieve otherwise similar test coverage (which is indicative of higher test data compression and enhanced encoding efficiency), or, alternatively, it returns higher test coverage numbers for comparable memory usage. Besides its simple in-system test controller, a test memory remains the only hardware component that shapes the Stellar BIST silicon real state. Fortunately, the scheme's ability to easily trade-off test data volume and test time, can alleviate this problem in a programmable fashion.

CHAPTER 6
CONCLUSIONS

6.1 Summary

This thesis introduces three DFT methods that reduce the ATPG-produced test set size used with different test schemes.

In Chapter 3, we have proposed a TPI technique that resolves large internal conflicts to increase the number of faults detected by each test pattern, while replacing dedicated flip-flops by properly-selected existing functional flip-flops as drivers of the control points to minimize the area overhead. Experiments conducted on several industrial designs show that similar pattern count reduction can be achieved, when compared to the method of control points using dedicated drivers which has already been proven to be very effective.

In Chapter 4, we presented a staggered ATPG process that generates compacted test patterns and applies them in a test-per-clock manner, with the help of a scan architecture which contains dedicated compactor scan chains formed by capture-per-cycle observation test points. This staggered ATPG process can merge more test cubes together, by shifting certain incompatible test cubes along scan chains to make them compatible with the merge array without losing any fault detection. A limited number of capture-per-cycle observation test points are carefully selected to detect faults to reduce the overall pattern count. Experimental results show that a good pattern count reduction can be achieved using this staggered ATPG with an acceptable number of observation test points, compared to a conventional ATPG for both 1-cycle and 2-cycles test patterns.

In Chapter 5, we seek to meet the high-quality test demands required by in-system test (especially automotive test). Therefore, we have developed a Stellar BIST approach that obtains the desired test stimuli by complementing multiple bits from a small set of stored seeds. Bit-complements are performed through a simple test controller hardware.

Valuable test clusters are also generated with a high compression ratio. According to the experimental results, the Stellar BIST offers a faster test-coverage ramp-up with less memory usage, not to mention the very flexible trade-offs between test data volume and test application time.

Table 6.1 Staggered ATPG with additional conflict-aware control points

	Without control points			With control points		
	Conventional ATPG PC	Staggered ATPG PC	Pattern reduction	Conventional ATPG PC	Staggered ATPG PC	Pattern reduction
D1	2,564	2,345	1.09x	1,400	953	1.47
D2	26,375	8,626	3.06x	4,161	2,089	1.99
D3	3,204	1,878	1.71x	1,724	1,025	1.68
D4	8,263	4,508	1.83x	5,433	3,000	1.81
D5	1,914	889	2.15x	1,370	723	1.89
D6	5,200	5,111	1.02x	1,822	1,151	1.58
D7	3,206	1,820	1.76x	3,195	1,798	1.78

Furthermore, our proposed methods can also be applied in a combined way for even better performance on the reduction of test set size. First, staggered ATPG can be applied to designs which have already been inserted with conflict-aware control points discussed in Chapter 3 and can work synergistically with these control points. As shown in Table 6.1, staggered ATPG is capable of further reducing the pattern count, based on the reduced patterns after control points insertion. For D1 and D6, control points can also improve the effectiveness of staggered ATPG (from 1.02x to 1.58x pattern count reduction).

Another option is to combine the Stellar BIST approach with the staggered ATPG. This combined solution allows us to generate both parent and children patterns for Stellar BIST in a similar way as proposed by staggered ATPG, and with additional capture-per-cycle observation test points inserted (similar to that discussed in Section 4.4), all Stellar

BIST test stimuli can be applied in a test-per-clock fashion. In this case, parent patterns are generated from original or shifted test cubes, using the same fault order as discussed in Section 5.3. Extra faults can also be detected by both parent and children patterns during their shift cycles. Table 6.2 compares the total stored patterns and applied patterns of the combined solution with a small CBC value of 8 to the Stellar BIST results in Section 5.4 (5 designs using stuck-at faults for the 90% test coverage target). As shown in Table 6.2, the combined solution using a CBC of 8 requires similar or even less stored patterns, compared to the results with a CBC value of 64, while the total number of applied patterns is much less than the original Stellar BIST approach.

Table 6.2 Apply Staggered ATPG to Stellar BIST

	Original Stellar BIST				Combined solution	
	Stored patterns		Applied patterns		Stored patterns	Applied patterns
	CBC=8	CBC=64	CBC=8	CBC=64		
D1	256	128	1,856	5,688	128	569
D2	290	192	1,408	4,288	192	1,344
D3	1,145	511	5,500	13,792	384	3,116
D4	253	128	1,088	4,330	128	1,148
D5	32	32	176	512	32	160

6.2 Future Work

In this study, we have proposed several different methods to reduce ATPG-produced test set size. Since we primarily focus on traditional fault models, like stuck-at and transition fault models throughout our experiments, the application of our approaches on other fault models (especially such as cell internal faults) remains to be explored.

In Chapter 4, we introduce a staggered ATPG leveraging carefully-inserted capture-per-cycle observation test points to apply staggered test patterns during shift cycles.

Suitable ways to load out the output data have not been examined yet. And it remains to be another unexplored area of whether these staggered test patterns, along with their capture-per-cycle test responses, can be used for diagnosis purposes.

For the Stellar BIST approach presented in Chapter 5, we focus on optimizing test data associated with input stimuli. Output test data may also need to be reduced accordingly. This requires an effective test response compaction scheme deployed on the output side with the capability of eliminating/tolerating unknown states propagating from the output channels.

REFERENCES

[1] E. B. Eichelberger and T. W. Williams, “A Logic Design Structure for LSI Testability", Proc. of DAC, pp. 462-468, 1977.

[2] V. D. Agrawal, C. R. Kime and K. K. Saluja, “A Tutorial on Buil-In Self-Test, Part 1: Principles", IEEE Design and Test of Computers, Vol. 10, Issue 1, pp. 73-82, March 1993.

[3] A. Jas, C.V. Krishna, and N.A. Touba, “Weighted pseudorandom hybrid BIST,” IEEE Trans. VLSI, vol. 12, no. 12, pp. 1277-1283, Dec. 2004.

[4] R. Kapur, S. Patil, T.J. Snethen, and T.W. Williams, “A weighted random pattern test generation system,” IEEE Trans. CAD, vol. 15, no. 8, pp. 1020-1025, Aug. 1996.

[5] K.-H. Tsai, J. Rajski, and M. Marek-Sadowska, “Star test: the theory and its applications,” IEEE Trans. CAD, vol. 19, no. 9, pp. 1052-1064, Sept. 2000.

[6] I. Pomeranz and S.M. Reddy, “Static test data volume reduction using complementation or modulo-M addition,” IEEE Trans. VLSI, vol. 19, no. 6, pp. 1108-1112, June 2011.

[7] J. Rajski, J. Tyszer, M. Kassab and N. Mukherjee, “Embedded Deterministic Test", IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 23, Issue 5, pp. 776-792, May 2004.

[8] M. Abramovici, M. Breuer, and A. Friedman, “Digital Systems Testing and Testable Design”, IEEE Press, Piscataway, NJ, 1994.

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

[10] S.M. Reddy, “Test Drivers – Past, Present, and Future”, VTS 2018 Invited Keynote.

[11] J. A. Waicukauski, E. Lindbloom, B. K. Rosen and V. S. Iyengar, “Transition Fault Simulation", IEEE Design and Test of Computers, Vol. 4, Issue 2, pp. 32-38, 1987.

[12] J. Savir, “Skewed-Load Transition Test: Part I, Calculus", Proc. of ITC, pp. 705713, September 1992.

[13] J. Savir and S. Patil, “Broad-Side Delay Test", IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 13, Issue 8, pp. 1057-1064, August 1994.

[14] G.L. Smith, “Model for Delay Faults Based Upon Paths,” Proc. ITC 1985, pp.342-349.

[15] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, “Cell-aware test,” IEEE Trans. CAD, vol. 33, pp. 396-1409, 2014.

[16] J. P. Roth, “Diagnosis of automata failures: a calculus and a method,” in IBM Journal of Research & Devlopment, vol. 10, pp. 278 –291, July 1996.

[17] ISO 26262-1:2011- Road Vehicles—Functional Safety, 2011, online available: http://www.iso.org/iso/catalogue_detail?csnumber=43464.

[18] X. Fan, S.M. Reddy, S. Wang, S. Kajihara, and Y. Sato, “Genetic algorithm based approach for segmented testing”, in Proc. Int. Conf. on Dependable Systems and Networks Workshops, 2011, pp. 85-90.

[19] Y. Liu, E. Moghaddam, N. Mukherjee, S. M. Reddy J. Rajski, and J. Tyszer, “Minimal area test points for deterministic patterns”, in Proc. ITC, 2016, paper 2.4.

[20] Y. Liu, J. Rajski, S. M. Reddy, J. Solecki and J. Tyszer, “Staggered ATPG with cpature-per-cycle observation test points”, in Proc. VTS, 2018.

[21] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy, and J. Tyszer, “Deterministic Stellar BIST for in-system automotive test”, to be submitted to ITC 2018.

[22] A. Kumar, J. Rajski, S. M. Reddy, and C. Wang, "On the generation of compact test sets, " in ITC, 2013.

[23] A. Kumar, J. Rajski, S.M. Reddy, and T. Rinderknecht, “On the generation of compact deterministic test sets for BIST ready designs,” Proc. ATS, 2013, pp. 201-206.

[24] P. Goel and B. Rosales, “Test generation and dynamic compaction of test,” in Annu Test Conference, pp. 260 – 268, 1979.

[25] B. Ayari and B. Kaminska, “A new dynamic test vector compaction for automatic test pattern generation,” IEEE Trans. on CAD, vol. 13, pp. 353 –358, mar 1994.

[26] I. Pomeranz, L. Reddy, and S. Reddy, “Compactest: a method to generate compact test sets for combinational circuits,” IEEE Trans. on CAD, vol. 12, pp. 1040–1049, jul 1993.

[27] M. Konijnenburg, J. van der Linden, and A. van de Goor, “Compact test sets for industrial circuits,” pp. 358 –366, 1995.

[28] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, “Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits,” IEEE Transactions on CAD, vol. 14, pp. 1496 –1504, dec 1995.

[29] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, “Pattern generation for a deterministic bist scheme,” in ICCAD, pp. 88–94, 1995.

[30] Z. Wang and D. Walker, “Dynamic compaction for high quality delay test,” pp. 243 –248, 2008.

[31] S. Remersaro, J. Rajski, S. Reddy, and I. Pomeranz, “A scalable method for the generation of small test sets,” in Proc. of DATE, pp. 1136 –1141, 2009.

[32] L. Reddy, I. Pomeranz, and S. Reddy, “Rotco: a reverse order test compaction technique,” in Euro ASIC, pp. 189–194, 1992.

[33] D. Hochbaum, “An optimal test compression procedure for combinational circuits,” IEEE Trans. on CAD, vol. 15, no. 10, pp. 1294–1299, 1996.

[34] I. Hamzaoglu and J. Patel, “Test set compaction algorithms for combinational circuits,” in Procs. of ICCAD, pp. 283 – 289, 1998.

[35] I. Pomeranz and S. M. Reddy, “Forward-looking fault simulation for improved static compaction,” IEEE Trans. on CAD, vol. 20, pp. 1262–1265, Nov. 2006.

[36] S. Reddy, "Easily testable realizations for logic functions," IEEE Transactions on Computers, vol. 21, no. 11, pp. 1183-1188, 1972.

[37] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection", IRE Trans. Electron. Comput., vol. EC-3, pp. 6-12, Sept. 1954.

[38] W. -T Cheng, J. H. Patel, "A Minimum Test set for Multiple Fault Detection in Ripple carry Adders", IEEE Trans. Comput., vol. C-36, no. 7, pp. 891-895, July 1987.

[39] M.J. Geuzebroek, J.T. van der Linden, and A.J. van de Goor, “Test point insertion that facilitates ATPG in reducing test time and data volume,” Proc. ITC, 2002, pp. 138-147.

[40] I. Pomeranz and S.M. Reddy, “Test-point insertion to enhance test compaction for scan designs,” Proc. ICDSN, 2000, pp. 375-381.

[41] S. Romersaro, J. Rajski, T. Rinderknecht, S.M. Reddy, and I. Pomeranz, “ATPG heuristics dependent observation point insertion for enhanced compaction and data volume reduction,” Proc. DFTVS, 2008, pp. 385-393.

[42] N. Tamarapalli and J. Rajski, “Constructive multi-phase test point insertion for scan-based BIST,” Proc. ITC, 1996, pp. 649-658.

[43] S. Udar and D. Kagaris, “Minimizing observation points for fault location,” Proc. DFT, 2009, pp. 263-267.

[44] M. Yoshimura, T. Hosokawa, and M. Ohta, “A test point insertion method to reduce the number of test patterns,” Proc. ATS, 2002, pp. 298-304.

[45] L.H. Goldstein and E.L. Thigpen, “SCOAP: Sandia controllability/observability analysis program,” Proc. DAC, 1980, pp. 190-196.

[46] M.J. Geuzebroek, J.T. van der Linden, and A.J. van de Goor, “Test point insertion for compact test sets,” Proc. ITC, 2000, pp. 292-301.

[47] C. Acero, D. Feltham, F. Hapke, E. Moghaddam, N. Mukherjee, V. Neerkundar, M. Patyra, J. Rajski, J. Tyszer, J. Zawada, “Embeded deterministic test points for compact cell-aware tests,” Proc. ITC, 2015, paper 2.2.

[48] C. Su, C. R. Kime, "Computer-Aided Design of Pseudoexhaustive BIST for Semiregular Circuits", Proc. Int. Test Conf., pp. 680-687, 1990.

[49] I. Pomeranz, S. M. Reddy, "On methods to match a test pattern generator to a circuit-under-test", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 6, no. 3, pp. 432-444, Sep. 1998.

[50] P. H. Bardell and W.H. McAnney, “Simultaneous self-testing system,” US patent 4513418, Apr. 23, 1985.

[51] F. Corno, P. Prinetto, and M. Sonza Reorda, “Making the circular self-test path technique effective for real circuits,” Proc. ITC, 1994, pp. 949-957.

[52] A. Jas, K. Mohanram, and N.A. Touba, “An embedded core DFT scheme to obtain highly compressed test sets,” Proc. ATS, 1999, pp. 275-280.

[53] E. Kalligeros, X. Kavousianos, D. Bakalis, and D. Nikolos, “An efficient seeds selection method for LFSR-based test-per-clock BIST,” Proc. ISQED, 2002, pp. 261-266.

[54] B. Konemann, J. Mucha, and G. Zwiehoff, “Built-in logic block observation techniques,” Proc. ITC, 1979, pp. 37-41.

[55] A. Krasniewski and S. Pilarski, “Circular self-test path: a low cost BIST technique for VLSI circuits,” IEEE Trans. CAD, vol. 8, no.1, pp. 46-55, Jan. 1989.

[56] Y. Son, J. Chong, and G. Russell, “E-BIST: Enhanced test-per-clock BIST architecture,” IEEE Proc. Comput. Digit. Techn., vol. 149, pp. 9–15, Jan. 2002.

[57] C. Stroud, “An automated BIST approach for general sequential logic synthesis,” Proc. DAC, 1988, pp. 3-8.

[58] G. Mrugalski, J. Rajski, J. Solecki, J. Tyszer, and C. Wang, “Trimodal scan-based test paradigm,” IEEE Trans. VLSI Systems, vol. 25, no. 3, pp. 1112-1125, March 2017.

[59] F. Zhang, D. Hwong, Y. Sun, A. Garcia, S. Alhelaly, G. Shofner, L. Winemberg, and J. Dworak, "Putting wasted clock cycles to use: Enhancing fortuitous cell-aware fault detection with scan shift capture,” Proc. ITC, 2016, paper 2.3.

[60] S. Milewski, N. Mukherjee, J. Rajski, J. Solecki, J. Tyszer, and J. Zawada, “Full-scan LBIST with capture-per-cycle hybrid test points”, Proc. ITC, 2017, paper 10.3.

[61] O. Novak and J. Nosek, “Test-per-clock testing of the circuits with scan,” Proc. Int. On-Line Test Workshop, 2001, pp. 90-92.

[62] W. Rao and A. Orailoglu, “Virtual compression through test vector stitching for scan based designs,” Proc. DATE, 2003, pp. 104-109.

[63] H.-J. Wunderlich, “Multiple distributions for biased random test patterns,” IEEE Trans. CAD, vol. 9, no. 6, pp. 584-593, June 1990.

[64] A.-W. Hakmi, H.-J. Wunderlich, C.G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel, and L. Souef, “Programmable deterministic built-in self-test,” in Proc. ITC, 2007, paper 18.1.

[65] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, “A mixed mode BIST scheme based on reseeding of folding counters,” in Proc. ITC, 2000, pp. 778-784.

[66] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlich, “Two-dimensional test data compression for scan-based deterministic BIST,” in Proc. ITC, 2001, pp. 894-902.

[67] N.A. Touba and E.J. McCluskey, “Bit-fixing in pseudo-random sequences for scan BIST,” IEEE Trans. CAD, vol. 20, no. 4, pp. 545-555, April 2001.

[68] H.-J. Wunderlich and G. Kiefer, “Bit-flipping BIST,” in Proc. ICCAD, 1996, pp. 337-343.

[69] G. Kiefer, H. Vranken, E.-J. Marinissen, and H.-J. Wunderlich, “Application of deterministic logic BIST on industrial circuits,” in Proc. ITC, 2000, pp. 105-114.

[70] H.-J. Wunderlich and G. Kiefer, “Deterministic BIST with multiple scan chains,” in Proc. ITC, 1998, pp. 1057-1064.

[71] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, “Highly X-tolerant selective compaction of test responses,” in Proc. VTS, 2009, pp. 245-250.

[72] J. Rajski, J. Tyszer, G. Mrugalski, N. Mukherjee, W.-T. Cheng, and M. Kassab, “X-Press: two-stage X-tolerant compactor with programmable selector,” IEEE Trans. CAD, vol. 7, no. 1, pp. 147159, January 2008.

[73] P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, and H.J. Wunderlich, “High defect coverage with low-power test sequences in a BIST environment,” IEEE Design & Test of Computers, vol. 19, pp. 44-52, Sept.-Oct. 2002.

[74] J. Rajski, J. Tyszer, G. Mrugalski, and B. Nadeau-Dostie, “Test generator with preselected toggling for low power built-in self-test,” in Proc. VTS, 2012, pp. 1-6.

[75] A.A. Al-Yamani, S. Mitra, and E.J. McCluskey, “BIST reseeding with very few seeds,” in Proc. VTS, 2003, pp. 69–76.

[76] D. Das and N.A. Touba, “Reducing test data volume using external/LBIST hybrid test patterns,” in Proc. ITC, 2000, pp. 115-122.

[77] R. Dorsch and H.-J. Wunderlich, “Tailoring ATPG for embedded testing,” in Proc. ITC, 2001, pp. 530-537.

[78] A. Jas, C.V. Krishna, and N.A. Touba, “Hybrid BIST based on weighted pseudo-random testing: a new test resource partitioning scheme,” in Proc. VTS, 2001, pp. 2-8.

[79] C.V. Krishna and N.A. Touba, “Hybrid BIST using an incrementally guided LFSR,” in Proc. Symp. Defect and Fault Tolerance, 2003, pp. 217-224.

[80] L. Lei and K. Chakrabarty, “Hybrid BIST based on repeating sequences and cluster analysis,” in Proc. DATE, 2005, pp. 11421147.

[81] P. Wohl, J.A. Waicukauski, S. Patel, and M. Amin, “X-tolerant compression and applications of scan-ATPG patterns in a BIST architecture,” in Proc. ITC, 2003, pp. 727-736.

[82] B. Koenemann, “LFSR-coded test patterns for scan designs,” in Proc. ETC, 1991, pp. 237-242.

[83] V. Gherman, H.-J. Wunderlich, H. Vranken, F. Hapke, M. Wittke, and M. Garbers, “Efficient pattern mapping for deterministic logic BIST,” in Proc. ITC, 2004, pp. 48–56.

[84] A.-W. Hakmi, S. Holst, H.-J. Wunderlich, J. Schloffel, F. Hapke, and A. Glowatz, “Restrict encoding for mixed-mode BIST,” in Proc. VTS, 2009, pp. 179-184.

[85] S. Hellebrand, 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 Trans. Comput., vol. 44, no. 2, pp. 223-233, Feb. 1995.

[86] C.V. Krishna and N.A. Touba, “Reducing test data volume using LFSR reseeding with seed compression,” in Proc. ITC, 2002, pp. 321-330.

[87] J. Lee and N. A. Touba, “LFSR-reseeding scheme achieving lowpower dissipation during test,” IEEE Trans. CAD, vol. 26, no. 2, pp. 396-401, Feb. 2007.

[88] P. M. Rosinger, B. M. Al-Hashimi, and N. Nicolici, “Low power mixed-mode BIST based on mask pattern generation using dual LFSR re-seeding,” in Proc. ICCD, 2002, pp. 474-479.

[89] P. Wohl, J.A. Waicukauski, S. Patel, F. DaSilva, T.W. Williams, and R. Kapur, “Efficient compression of deterministic patterns into multiple PRPG seeds,” in Proc. ITC, 2005, pp. 916-925.

[90] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, and T. Onodera, “Extending OPMISR beyond 10x scan test efficiency,” IEEE Design & Test, vol. 19, no. 5, pp. 65-73, 2002.

[91] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, P. Szczerbicki, and J. Tyszer, “Deterministic clustering of incompatible test cubes for higher power-aware EDT compression”, IEEE Trans. CAD, vol. 30, no. 8, pp. 1225-1238, Aug. 2011.

[92] R. Kapur, S. Mitra, and T.W. Williams, “Historical perspective on scan compression,” IEEE Design & Test, vol. 25, no. 2, pp. 114-120, 2008.

[93] N.A. Touba, “Survey of test vector compression techniques,” IEEE Design & Test, vol. 23, pp. 294-303, 2006.

[94] G. Mrugalski, J. Rajski, J. Solecki, Ł. Rybak, and J. Tyszer, “StarEDT: Deterministic on-chip scheme using compressed test patterns,” IEEE Trans. CAD, vol. 36, No. 4, April 2017, pp. 683-693.

[95] E. Moghaddam, N. Mukherjee, J. Rajski, J. Tyszer, and J. Zawada, “Test point insertion in hybrid test compression/LBIST architectures,” Proc. ITC, 2016, paper 2.1.

[96] J. Rajski, J. Tyszer, C. Wang, and S. Reddy, “Finite memory test response compactors for embedded test applications,” IEEE Trans. CAD, vol. 24, no. 4, pp. 622-634, April 2005.