Fall 2013

Transition Faults and Transition Path Delay Faults: Test Generation, Path Selection, and Built-In Generation of Functional Broadside Tests

Bo Yao
Purdue University

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Bo Yao

Entitled

Transition Faults and Transition Path Delay Faults: Test Generation, Path Selection, and Built-In Generation of Functional Broadside Tests

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

IRITH POMERANZ
Chair

ANAND RAGHUNATHAN

RAYMOND A. DECARLO

YUNG-HSIANG LU

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): IRITH POMERANZ

Approved by: M. R. Melloch 11-21-2013
Head of the Graduate Program Date

TRANSITION FAULTS AND TRANSITION PATH DELAY FAULTS: TEST GENERATION, PATH SELECTION, AND BUILT-IN GENERATION OF FUNCTIONAL BROADSIDE TESTS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Bo Yao

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2013

Purdue University

West Lafayette, Indiana

This dissertation is dedicated to my parents for their unconditional love and support.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my advisor, Professor Irith Pomeranz, for her inspiration, encouragement and guidance throughout my PhD study. Without her support and help, the work cannot be accomplished. I also would like to thank Professor Raymond Decarlo, Professor Anand Raghunathan and Professor Yung-Hsiang Lu for serving on my Advisory Committee. Particularly, I would like to thank Professor Sudhakar M. Reddy from University of Iowa, Dr. Arani Sinha, Dr. Srikanth Venkataraman, Dr. Enamul Amyeen and Mr. Carlston Lim from Intel for their contributions to this work. In addition, I would like to thank Semiconductor Research Corporation (SRC) for providing the grants to support the work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Yao, Bo. Ph.D., Purdue University, December 2013. Transition Faults and Transition Path Delay Faults: Test Generation, Path Selection, and Built-In Generation of Functional Broadside Tests. Major Professor: Irith Pomeranz.

As the clock frequency and complexity of digital integrated circuits increase rapidly, delay testing is indispensable to guarantee the correct timing behavior of the circuits. In this dissertation, we describe methods developed for three aspects of delay testing in scan-based circuits: test generation, path selection and built-in test generation.

We first describe a deterministic broadside test generation procedure for a path delay fault model named the transition path delay fault model, which captures both large and small delay defects. Under this fault model, a path delay fault is detected only if all the individual transition faults along the path are detected by the same test. To reduce the complexity of test generation, sub-procedures with low complexity are applied before a complete branch-and-bound procedure. Next, we describe a method based on static timing analysis to select critical paths for test generation. Logic conditions that are necessary for detecting a path delay fault are considered to refine the accuracy of static timing analysis, using input necessary assignments. Input necessary assignments are input values that must be assigned to detect a fault. The method calculates more accurate path delays, selects paths that are critical during test application, and identifies undetectable path delay faults. These two methods are applicable to off-line test generation. For large circuits with high complexity and frequency, built-in test generation is a cost-effective method for delay testing. For a circuit that is embedded in a larger design, we developed a method for built-in generation of functional broadside tests to avoid excessive power dissipation during test application and the overtesting of delay faults, taking the

functional constraints on the primary input sequences of the circuit into consideration. Functional broadside tests are scan-based two-pattern tests for delay faults that create functional operation conditions during test application. To avoid the potential fault coverage loss due to the exclusive use of functional broadside tests, we also developed an optional DFT method based on state holding to improve fault coverage. High delay fault coverage can be achieved by the developed method for benchmark circuits using simple hardware.

1. INTRODUCTION

The correct operation of a digital integrated circuit requires not only correct functional behavior but also correct operation at the desired clock frequency. As the manufacturing technology allows smaller feature size and the complexity of circuit increases, imperfection and random variations in process parameters are more likely to cause propagation delays to exceed the clock period. To guarantee the correctness of the circuit, it is necessary to perform delay testing.

1.1. Delay Fault Models

Defects that cause the faulty timing behavior of a circuit are modeled by delay faults. Two types of delay fault models are commonly used: the transition fault model [1] and the path delay fault model [2]-[4].

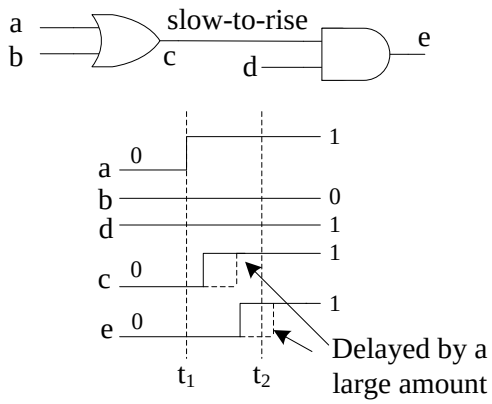

Fig. 1.1 Example of a transition fault

The transition fault model captures delay defects that cause a slow-to-rise transition or a slow-to-fall transition at a specific line in the circuit. Under this fault model, it is assumed that the extra delay caused by a transition fault on a line is large enough so that the delay of every path passing through this line exceeds the clock period. Fig. 1.1 shows an example of a slow-to-rise transition fault at line c in a 3-input circuit. Input b and d have constant values. The value of input a changes from 0 to 1 at time point t_1, i.e. a rising transition occurs at a, and the transition propagates through the circuit. If the circuit is fault free, the value of output e is 1 at the required time point t_2, where t_2-t_1 is the clock period. However, due to the slow-to-rise transition fault at line c, the value of output e remains 0 at t_2. Therefore, the circuit cannot operate correctly.

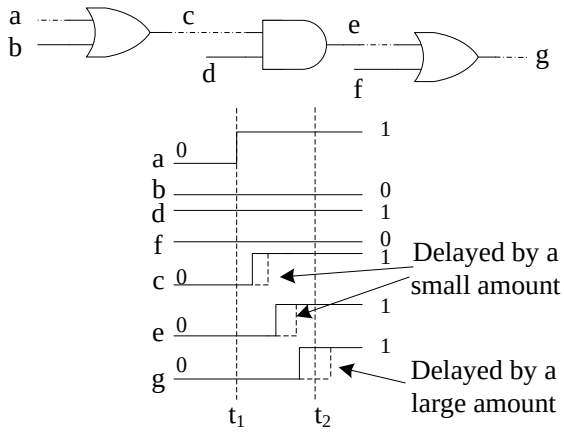

Fig. 1.2 Example of a path delay fault

Different from the transition fault model which only captures single large delay at a specific line, the path delay fault model captures small extra delays whose cumulative effect along a path from inputs to outputs may result in faulty behavior of the circuit, although each small extra delay by itself may not fail the circuit. Fig. 1.2 shows an example of a path delay fault associated with path a-c-e-g and a rising transition at its source a in a 4-input circuit. Input b, d and f have constant values. A rising transition occurs at input a at time point t_1, and the transition propagates along path a-c-e-g. If the

circuit is fault free, the value of output g is 1 at the required time point t_2, where t_2-t_1 is the clock period. However, due to the path delay fault along path a-c-e-g, the value of output g remains 0 at t_2. As a result, the circuit cannot operate correctly.

1.2. Tests for Delay Faults

Both a transition fault and a path delay fault are detected by a two-pattern test <p_1, p_2>. For a transition fault, the first pattern p_1 assigns the initial transition value at the faulty line. The second pattern p_2 assigns the final transition value at the faulty line and propagates the fault effect to the outputs. To detect the transition fault in Fig. 1.1, a test <001, 101> is applied to "abd" as shown in Fig. 1.3. The value of a line under p_1(p_2) is shown on the left(right) of the arrow. The value shown on the left of the slash is the expected value under p_2 if the circuit is fault free, and the value on the right of the slash is the faulty value under p_2 if a fault exists. The transition fault can be detected if a 0 instead of a 1 is observed at output e at the required time point. For a path delay fault, p_1 and p_2 create a transition at the source of the target path, and p_2 propagates it along the path. To detect the path delay fault in Fig. 1.2, a test <0010, 1010> is applied to "abdf" as shown in Fig. 1.4. The path delay fault can be detected if a 0 instead of a 1 is observed at output g at the required time point.

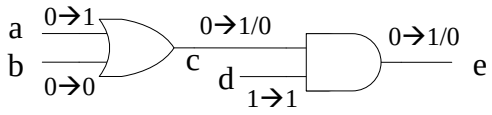

Fig. 1.3 A test for the slow-to-rise transition fault at c

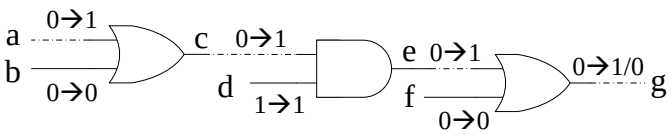

Fig. 1.4 A test for the path delay fault associated with path a-c-e-g

Based on the propagation conditions used for the detection of path delay faults, tests for path delay faults can be categorized as robust and non-robust [5]-[7]. A robust test guarantees the detection of a path delay fault regardless of the delays in the rest of the circuit. For example, the test shown in Fig. 1.4 is a robust test for the path delay fault. A non-robust test requires that the desired transition is created at the source of the target path and p_2 statically sensitizes the path to enable the propagation of the transition along the path. A non-robust test detects a path delay fault if none of the off-path input signals arrive late. Otherwise, the test may be invalid. A non-robust test <0011, 1010> for the path delay fault in Fig. 1.2 is applied to “abdf” as shown in Fig. 1.5. Different from the robust test in Fig. 1.4, a falling transition occurs at off-path input f. If the transition at f does not arrive late, the path delay fault is detected if a 0 instead of a 1 is observed at g at the required time point. Otherwise, the value of g will always be 1 at the required time point even if the delay of path a-c-e-g exceeds the clock period. In this case, the test is not valid for the path delay fault.

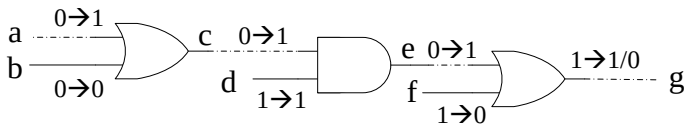

Fig. 1.5 A non-robust test for the path delay fault associated with path a-c-e-g

Non-robust tests can be further categorized as strong non-robust and weak non-robust [7]. Under a strong non-robust test, there is a transition that matches the transition at the source of the path on every line along the path, and every off-path input has a non-controlling value for the gate it drives under p_2. Under a weak non-robust test, it is only required that every off-path input has a non-controlling value for the gate it drives under p_2. A robust test for a path delay fault can detect all the transition faults along the path. However, a non-robust test for a path delay fault does not necessarily detect the transition faults along the path [8]. Fig. 1.6 shows a non-robust test for the path delay fault

associated with path b-d-f-h and a rising transition at its source b. The rising transition fault at b is then targeted under the non-robust test, as shown in Fig. 1.7. Under the test, a 0 is observed at output h in both the faulty and fault free circuit. Therefore, the rising transition fault at b cannot be detected by the non-robust test.

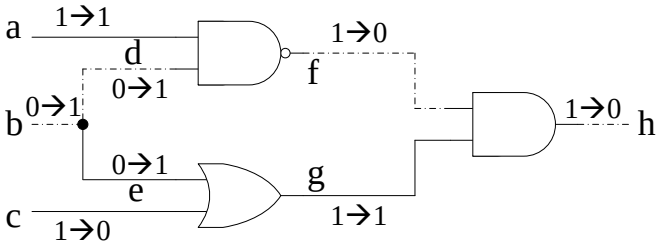

Fig. 1.6 A non-robust test for the path delay fault associated with path b-d-f-h

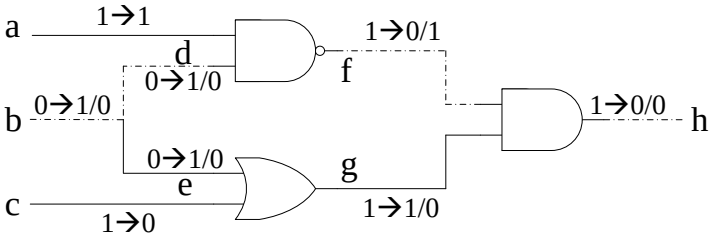

Fig. 1.7 A transition fault under a non-robust test for a path delay fault

1.3. Scan-Based Tests for Delay Faults

Sequential circuits contain sequential elements (or storage elements) such as latches and flip-flops. In order to improve the testability of a sequential circuit, scan structure is inserted by replacing the sequential elements with scannable sequential elements (scan cells) and then stitching the scan cells into scan chains [9]. Fig. 1.8 shows a D flip-flop based sequential circuit before and after scan insertion. The scan cell in Fig. 1.8 can be implemented by inserting a multiplexer in front of the data input of a regular flip-flop. When the scan enable signal SE is 0, the scan cell works as a regular flip-flop. When SE is 1, the scan chain works as a shift register, allowing arbitrary values to be shifted in and

applied to the flip-flops. The values of the flip-flops can be shifted out as well. As a result, the scan cells are considered as the inputs and outputs of the circuit. The controllability and observability of the circuit are therefore improved. The inputs of the original circuit are called primary inputs, and the outputs of the original circuit are called primary outputs.

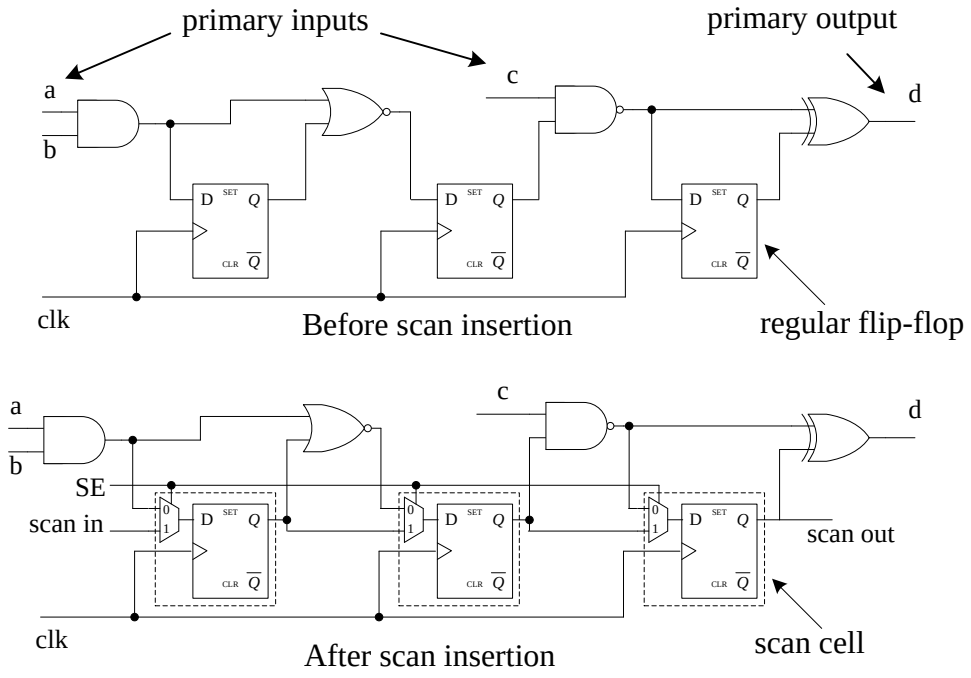

Fig. 1.8 A circuit before and after scan insertion

For a scan-based circuit, each pattern p_i under the two-pattern test $<p_1, p_2>$ has the form $<s_i, v_i>$, where s_i denotes the values of the state variables (or the scan cells), and v_i denotes the values applied to the primary inputs. The test $<p_1, p_2>$ can be rewritten as $<s_1, v_1, s_2, v_2>$. s_1 is usually shifted into scan chains. According to the way by which s_2 is obtained, scan-based tests for delay faults can be categorized into three types: enhanced scan tests [10], skewed-load tests [11] and broadside tests [12].

Under an enhanced scan test, s_1 and s_2 are independent. Both s_1 and s_2 are shifted into scan chains simultaneously. Among the three types of scan-based tests for delay faults, enhanced scan tests can achieve the highest fault coverage. However, special scan cells that can hold two bits of state values are required for applying the tests.

Under a skewed-load test, s_2 is obtained by a single shift of s_1. The timing waveform is shown in Fig. 1.9. During the application of a skewed-load test, SE is first set to 1 so that s_1 can be shifted into scan chains. v_1 is applied to the primary inputs once s_1 is completely loaded and the circuit is under $<s_1, v_1>$. By triggering the launch clock edge, s_1 is shifted by one bit and s_2 is obtained. v_2 is applied concurrently and the circuit operates under $< s_2, v_2>$. SE is then set to 0, and the capture clock edge is triggered one clock cycle after the launch clock edge to capture the response of the circuit to $<s_2, v_2>$ into scan chains. SE is then set back to 1 so that the captured response can be shifted out. The response unloaded from scan chains and the response observed at the primary outputs are compared with the expected values. A fault is detected if a mismatch is identified. The clock for shifting is usually slower than that for capture. It can be observed from Fig. 1.9 that under a skewed-load test, SE must be changed between the launch and capture clock edges.

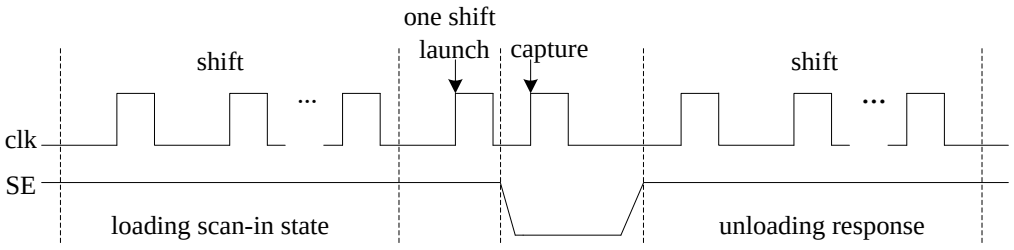

Fig. 1.9 Timing waveform for a skewed-load test

Under a broadside test, s_2 is determined by the response of the circuit to $<s_1, v_1>$. The timing waveform is shown in Fig. 1.10. During the application of a broadside test, SE is first set to 1 so that s_1 can be shifted into scan chains. v_1 is applied to the primary

inputs once s_1 is completely loaded and the circuit is under $<s_1, v_1>$. SE is set to 0 and then the launch clock edge is triggered. The response of the circuit to $<s_1, v_1>$ is captured into scan chains and s_2 is therefore obtained. v_2 is applied concurrently and the circuit operates under $<s_2, v_2>$. The response of the circuit to $<s_2, v_2>$ is captured into scan chains when the capture clock edge is triggered one clock cycle after the launch clock edge. SE is then set back to 1 so that the captured response can be shifted out. The unloaded response and the response observed at the primary outputs are compared with the expected values. A fault is detected if a mismatch is identified. It can be observed from Fig. 1.10 that under a broadside test, SE needs to be changed between the last shift and the launch clock edge. Since shifting clock is usually slower than capture clock, SE has a larger amount of time to change under a broadside test than a skewed-load test.

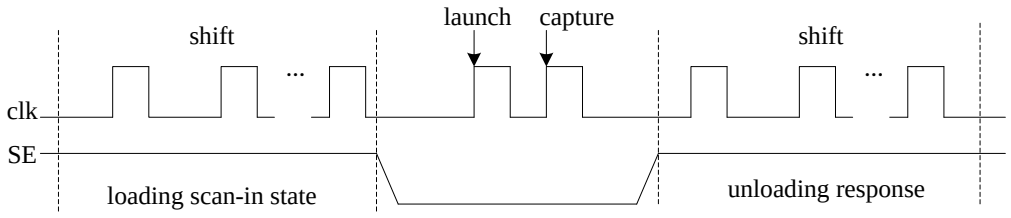

Fig. 1.10 Timing waveform for a broadside test

To guarantee that the circuit can operate at its designed speed, at-speed testing [13], which requires the launch and capture clock edges to be triggered at the designed clock rate, can be performed. This implies that SE must be changed very fast within a single designed clock period under a skewed-load test. Since it is expensive to implement such a high-speed SE, skewed-load tests are not always considered in practice even though they usually achieve higher fault coverage than broadside tests. In this dissertation, we only consider broadside tests for delay faults in scan-based circuits.

1.4. Contributions

This dissertation describes methods developed for three aspects of delay testing in scan-based circuits: deterministic test generation for a new path delay fault model, path selection, and built-in generation of functional broadside tests.

To address the issue that the transition fault model only captures single large delay and the path delay fault model only captures distributed small extra delays along a path, a new fault model named transition path delay fault model was proposed in [14]. Under this fault model, a path delay fault is detected only if all the individual transition faults along the path are detected by the same test. Therefore, both small and large delay defects can be captured. We developed a deterministic broadside test generation procedure for transition path delay faults. To reduce the complexity of test generation, the procedure consists of five sub-procedures: a deterministic test generation procedure for transition faults, a preprocessing procedure that identifies undetectable transition path delay faults without performing test generation, a fault simulation procedure that identifies transition path delay faults that can be detected by the tests for transition faults, a heuristic procedure similar to dynamic test compaction for transition faults that generates tests without backtracking on decisions made for previously detected faults, and a complete branch-and-bound procedure. Experimental results show that for most of the transition path delay faults in benchmark circuits, either a test is found or the fault is identified as undetectable.

Under the path delay fault model, it is not practical to target all the paths in a circuit for test generation since the number of paths can be exponential in the number of lines throughout the circuit. As a result, a subset of critical paths is usually selected for test generation. Various path selection methods can be used for path selection. One common method is to select the critical paths identified by static timing analysis [15]. Static timing analysis computes the path delays and identifies paths with the largest delays as critical paths. However, static timing analysis, by itself, can be inaccurate as it does not take into consideration logic conditions that are necessary for detecting path delay faults. We developed a path selection method that takes these conditions into account during static timing analysis. The logic conditions are captured as what are called input necessary

assignments [16]. By providing the static timing analysis process with the input necessary assignments for a selected path, the static timing analysis process can estimate more accurate path delays that are closer to those obtained during test application. It can also identify additional paths whose delays are at least as high as those of the selected paths. Feeding back the input necessary assignments to the static timing analysis process enhances the correlation between static timing analysis and actual timing of tests on silicon. The result of the method is a set of potentially detectable path delay faults associated with critical paths based on more accurate estimates of the path delays that can be exhibited by a test set, compared with the set that would be obtained by static timing analysis alone.

Both the deterministic test generation method and path selection method are applicable to off-line test generation, where tests are generated before being applied via an external tester. For large circuits with high clock frequency and complexity, it can be expensive to perform delay testing especially at-speed testing via an external tester, since a large amount of memory is required in the tester for storing the tests and the tests need to be applied at a high speed. For such circuits, built-in test generation is a cost-effective method for delay testing as it reduces test data volume by generating tests on-chip, and facilitates at-speed test application by avoiding the delivery of tests from an external tester. Many built-in test generation techniques allow arbitrary states to be scanned in during the application of the two-pattern tests, which may bring the circuit into non-functional operation conditions. As a result, excessive power dissipation during test application and the overtesting of delay faults may occur [17]-[20]. These issues can be addressed by using functional broadside tests that create functional operation conditions during test application [21].

We developed a method for built-in generation of functional broadside tests for a circuit that is embedded in a larger design, taking functional constraints on its primary input sequences into account. The constraints are captured by functional input sequences of the design. Specifically, the peak switching activity in the circuit under the functional input sequences is used to bound the switching activity during on-chip test generation. The exclusive use of functional broadside tests may cause fault coverage loss, i.e. faults

that can be detected by unrestricted broadside tests may not be detected by functional broadside tests. Such undetected faults may affect the reliability of the circuit in long-term. To address this issue, we also developed an optional DFT method based on state holding to improve fault coverage. By keeping the values of some state variables from changing at certain clock cycles during on-chip test generation, unreachable states can be introduced to detect faults that cannot be detected by functional broadside tests. Experimental results show that using simple hardware, the developed method can achieve high transition fault coverage for benchmark circuits.

1.5. Organization

The dissertation is organized as follows. In chapter 2, the deterministic broadside test generation method for transition path delay faults is described. Chapter 3 describes the path selection method based on static timing analysis with input necessary assignments considered. Chapter 4 describes the built-in generation method for functional broadside tests considering primary input constraints. Chapter 5 concludes the dissertation and discusses future work.

2. DETERMINISTIC BROADSIDE TEST GENERATION FOR TRANSITION PATH DELAY FAULTS

In this chapter, a deterministic broadside test generation procedure for transition path delay faults is described. The entire procedure consists of five sub-procedures: test generation for transition faults, preprocessing procedure, fault simulation, dynamic compaction heuristic procedure and branch-and-bound procedure. Experimental results show that most of the transition path delay faults associated with long paths in benchmark circuits can be detected or identified as undetectable.

2.1. Introduction

Delay faults are used to model defects that affect the timing behavior of a circuit. Two commonly used delay fault models, the transition fault model and the path delay fault model, have been introduced in chapter 1. In addition to these two fault models, the gate delay fault model [22] captures defects that cause small or large rising and falling transition delays from the input to the output of a logic gate. Variations of these three models include the double transition fault model [23] and the segment fault model [24][25].

Since distributed small extra delays caused by process variations can lead to the malfunction of a circuit only when they are accumulated along a path, it is important to apply tests for path delay faults. Robust tests are the highest quality tests for path delay faults. However, for most path delay faults, robust tests do not exist. Therefore, non-robust tests must be used. A non-robust test for a path delay fault may not detect the existence of a transition fault on the path, as shown in Fig. 1.6 and Fig. 1.7. As a result, the following was noted in [14]. Suppose that the accumulation of small extra delays along a subpath that ends at an internal line g is sufficient to cause the circuit to fail. A

non-robust test for a path delay fault that includes the subpath may not detect this situation since the test does not detect the transition fault on g. To address this issue, a different path delay fault model named the transition path delay fault model was proposed in [14]. Under this new path delay fault model, a path delay fault is detected if an only if all the individual transition faults along the path are detected by the same test. This guarantees that the cumulative propagation delay in the situation mentioned above can be detected by detecting a transition fault at the end of the subpath. Therefore, both small and large delay defects are detected by the test.

Test generation for a transition path delay fault requires the generation of a test that detects all the individual transition faults along the path. There is similarity between this requirement and the way dynamic compaction procedures produce compact test sets [26]-[28]. These procedures also try to generate a test that detects a subset of faults. However, in a dynamic compaction procedure, if test generation for a target fault fails when the current test is partially specified based on faults targeted earlier, the fault will be dropped and another fault will be selected to expand the subset of faults detected by the current test. The dropped faults will be considered later under different tests. In the test generation procedure for a transition path delay fault, all the transition faults along the path must be detected by the same test. Thus, there is no flexibility in deciding on the subset of faults that will be detected by the test. This implies that if test generation for a target transition fault fails when the current test is partially specified, it is necessary to backtrack on decisions made based on faults considered earlier until either a test that detects all the transition faults is found, or the path delay fault is shown to be undetectable. As a result, a complete test generation procedure for transition path delay faults can have a high computational complexity. To reduce this complexity, we use several sub-procedures. (1) A deterministic test generation procedure for transition faults is used to generate tests for transition faults and identify undetectable transition faults. The undetectable transition faults are used for identifying undetectable transition path delay faults. (2) A preprocessing procedure is used to identify undetectable transition path delay faults without performing test generation for them. (3) A fault simulation procedure is used to identify transition path delay faults that can be detected by the tests

generated in (1) for transition faults. (4) A heuristic procedure similar to dynamic test compaction is used to generate tests for transition path delay faults without backtracking on decisions made for transition faults detected earlier. (5) A complete branch-and-bound procedure is used to process the remaining undetected transition path delay faults.

2.2. The Transition Path Delay Fault Model

It has been shown in Fig. 1.6 and Fig. 1.7 that a non-robust test for a path delay fault may not detect a transition fault on the path. This occurs in a very common situation where different paths with opposite inversion polarities reconverge at a gate along the path. A non-robust test only requires the off-path input lines to have non-controlling values under the second pattern of a test in the fault free circuit. Therefore, the fault effects of the transition fault, which propagate along different paths with opposite inversion polarities, may counteract each other when the paths reconverge. As a result, the transition fault is not detected. Robust tests prohibit the reconvergence of fault effects in this case and remain valid for transition faults along the paths. However, they only exist for a small number of path delay faults. Therefore, it is common that a test can detect a path delay fault but cannot detect a transition fault along the path.

This issue is important for the following reason. Consider a path $p=g_1\text{-}g_2\text{-}g_3\text{-}g_4\text{-}g_5$ and a non-robust test t for the path delay fault associated with p. Suppose that the cumulative small extra delays along subpath $g_1\text{-}g_2\text{-}g_3$ are sufficient to cause the circuit to fail. This can be detected by t if it is a test for the transition fault at g_3. However, since t may not detect transition faults along p, the fault effect may not be captured.

The issue can be resolved if a path delay fault is detected by detecting all the transition faults along the path. This is the requirement for detecting a transition path delay fault in [14]. The relevant transition faults are defined in [14] as follows. Let us consider a transition path delay fault associated with a path $p=g_1\text{-}g_2\text{-}\ldots\text{-}g_k$ and a transition $v_1 \rightarrow v'_1$ on g_1. When the $v_1 \rightarrow v'_1$ transition is propagated from g_1 along the path, let the transition on g_i be $v_i \rightarrow v'_i$. We have $v_i=v_1$ if the number of inverters between g_1 and g_i is even, and $v_i=v'_1$ if the number of inverters between g_1 and g_i is odd. To detect the fault

associated with p and $v_1 \rightarrow v'_1$, it is required in [14] that the $v_i \rightarrow v'_i$ transition fault on line g_i should be detected by the same test, for $1 \leq i \leq k$.

As discussed in [14], tests for transition path delay faults are strong non-robust tests for the standard path delay faults. A strong non-robust test creates a transition on each line along the path and assigns non-controlling value to every off-path input under the second pattern. Test sets for transition path delay faults detect all or almost all the detectable transition faults. In addition, they detect all or almost all the standard path delay faults that can be detected by strong non-robust tests. Therefore, the transition path delay fault model can be an alternative to the path delay fault model.

A simulation based test generation procedure was proposed in [14]. Given a target transition path delay fault, the procedure tries to generate a test by combining tests for transition faults along the path. Compared with a deterministic test generation procedure, this procedure has lower computational complexity. However, it does not guarantee that every detectable transition path delay fault can be detected, and it cannot tell whether a transition path delay fault is detectable or not. For this, a deterministic test generation procedure is needed.

2.3. Test Generation Procedure

In this section, a deterministic broadside test generation procedure for transition path delay faults is described. For convenience, we use f_p to symbolize a transition path delay fault associated with path p, and use $TR(f_p)=\{tr_1(f_p), tr_2(f_p), \ldots, tr_k(f_p)\}$ to symbolize the set of transition faults along path p, where k is the length of the path and $tr_i(f_p)$ $(1 \leq i \leq k)$ is a single transition fault along the path.

2.3.1. Test Generation for Transition Faults

The first sub-procedure is a deterministic test generation procedure for transition faults. It achieves two goals. (1) Tests for transition faults sometimes detect transition path delay faults. We will simulate the tests under transition path delay faults as described later. (2) The procedure identifies undetectable transition faults. This information will be used in the following sub-procedure.

2.3.2. Preprocessing Procedure

The preprocessing procedure identifies as many undetectable transition path delay faults as possible without performing test generation. A transition path delay fault f_p is detected if and only if all the transition faults in $TR(f_p)$ are detected. If a transition fault $tr_i(f_p) \in TR(f_p)$ has been identified as undetectable, f_p is undetectable and no further processing of f_p is needed.

We also consider conflicts between the necessary assignments [29] of the transition faults in $TR(f_p)$. The necessary assignments for a fault are the assignments that must be made in order to find a test for the fault. For the v→v' transition fault on line g, we use g=v under the first pattern and g=v' under the second pattern as necessary assignments. In addition, we use the simple forward and backward implications of these assignments as necessary assignments. We find necessary assignments for each transition fault in $TR(f_p)$. If a conflict exists between the necessary assignments of the faults in $TR(f_p)$, we identify f_p as undetectable. Otherwise, we keep the necessary assignments on input lines for use in later sub-procedures.

Fig. 2.1 shows an example. We consider the path c-d-e with the 0→1 transition at its source. The transition path delay fault associated with this path consists of three transition faults: a 0→1 transition fault on c, a 1→0 transition fault on d, and a 0→1 transition fault on e. To detect the 0→1 transition fault on e, e should be 0 under the first pattern and 1 under the second pattern. Under a broadside test, e=0 under the first pattern implies c=0 under the second pattern. To detect the 0→1 transition fault on c, c should be 0 under the first pattern and 1 under the second pattern. A conflict occurs on c under the second pattern. Therefore, the transition path delay fault is undetectable. To identify such conflicts, we use implications to find all the necessary assignments of the 0→1 transition fault on e and the 0→1 transition fault on c. For the 0→1 transition fault on e, we have e=0 under the first pattern and c=0, e=1 under the second pattern. For the 0→1 transition fault on c, we have c=0, e=1 under the first pattern and c=1, d=0 under the second pattern. The necessary assignments of these two transition faults are compared. Since conflicts are identified on e under the first pattern and on c under the second pattern, the two

transition faults cannot be detected by the same test, i.e. the transition path delay fault is undetectable.

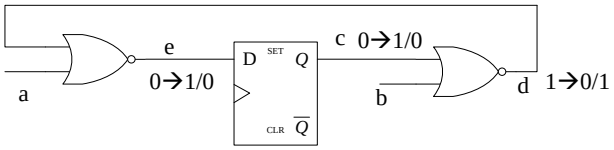

Fig. 2.1 Example of finding necessary assignments

2.3.3. Fault Simulation

After identifying undetectable transition path delay faults in the preprocessing procedure, we identify the transition path delay faults that can be detected by tests for transition faults. For this, we perform fault simulation of transition path delay faults under the test set computed in section 2.3.1.

2.3.4. Dynamic Compaction Heuristic Procedure

The dynamic compaction heuristic procedure is applied next to all the transition path delay faults that were not identified as undetectable or detected by the transition fault test set. For a transition path delay fault f_p, the dynamic compaction heuristic procedure attempts to generate a test that detects all the transition faults in $TR(f_p)$. The transition faults in $TR(f_p)$ are targeted one after the other by using unspecified bits remaining after the detection of the transition faults targeted earlier. We refer to this procedure as heuristic since it cannot guarantee that all the faults in $TR(f_p)$ will be detected by the same test.

The heuristic procedure is applied several times to each transition path delay fault f_p. Every time the procedure is applied, the transition faults in $TR(f_p)$ are considered in a different order. The order is such that faults, which are more difficult to detect, are considered earlier. As additional test generation attempts are made, new faults are identified as difficult to detect and the order changes. This is implemented as follows. We associate a parameter called “number of failures” with every transition fault $tr_i(f_p)$ in

$TR(f_p)$. The parameter records the number of times test generation for $tr_i(f_p)$ fails. The initial value of the number of failures for each transition fault is 0. Every time when test generation for $tr_i(f_p)$ fails, we increase the parameter by 1. The transition fault with the higher number of failures, i.e. the transition fault for which test generation is more difficult, should be targeted earlier so that there will be more unspecified bits to use for generating the test.

Using the terminology from [26], we refer to the first transition fault which is used to generate a test as a primary target fault and denote it by $tr_{prim}(f_p)$. We refer to any transition fault targeted after the primary target fault as a secondary target fault and denote it by $tr_{sec}(f_p)$. To distinguish between the secondary target faults, we denote the i^{th} fault by $tr_{sec(i)}(f_p)$. A parameter named "detect status" is associated with every transition fault $tr_i(f_p)$ in $TR(f_p)$. If a transition fault $tr_i(f_p)$ is detected by the current test, its detect status is "detected". Otherwise, its detect status is "undetected". A label named "used" is used to mark a fault $tr_{prim}(f_p)$ whose detection is followed by the failure of the test generation for $tr_{sec(1)}(f_p)$. Since $tr_{sec(1)}(f_p)$ can be detected individually, the failure should be caused by the detection of $tr_{prim}(f_p)$. A transition fault in $TR(f_p)$ which has been marked as "used" will not be selected as a primary target fault again because its detection will cause the test generation for some secondary target fault to fail.

The heuristic procedure proceeds as shown in Fig. 2.2. We randomly select a fault from the undetected unused transition faults that have the highest number of failures as $tr_{prim}(f_p)$. We attempt to generate a test for $tr_{prim}(f_p)$. If $tr_{prim}(f_p)$ is not detected, we stop attempting to generate a test for f_p. Otherwise, we select the transition fault $tr_i(f_p)$ which has the highest number of failures as a secondary target fault and attempt to expand the test by using the remaining unspecified bits. If a choice exists, we randomly select one of the faults. If the test generation for a secondary target fault $tr_{sec(i)}(f_p)$ fails, we increase the number of failures of $tr_{sec(i)}(f_p)$ by 1 and check whether $tr_{sec(i)}(f_p)$ is the first selected secondary target fault $tr_{sec(1)}(f_p)$. If so, we mark the current primary target fault $tr_{prim}(f_p)$ as used, discard the current test and start the procedure again. If $tr_{sec(i)}(f_p)$ is detected, we first check whether f_p can be detected by the current test. If so, a test for f_p is found. Otherwise, we continue to consider other secondary target faults. The procedure runs

until f_p is detected or the run time exceeds a predetermined limit. To accelerate the procedure, we apply the necessary assignments on input lines stored in the preprocessing procedure for f_p before the procedure starts.

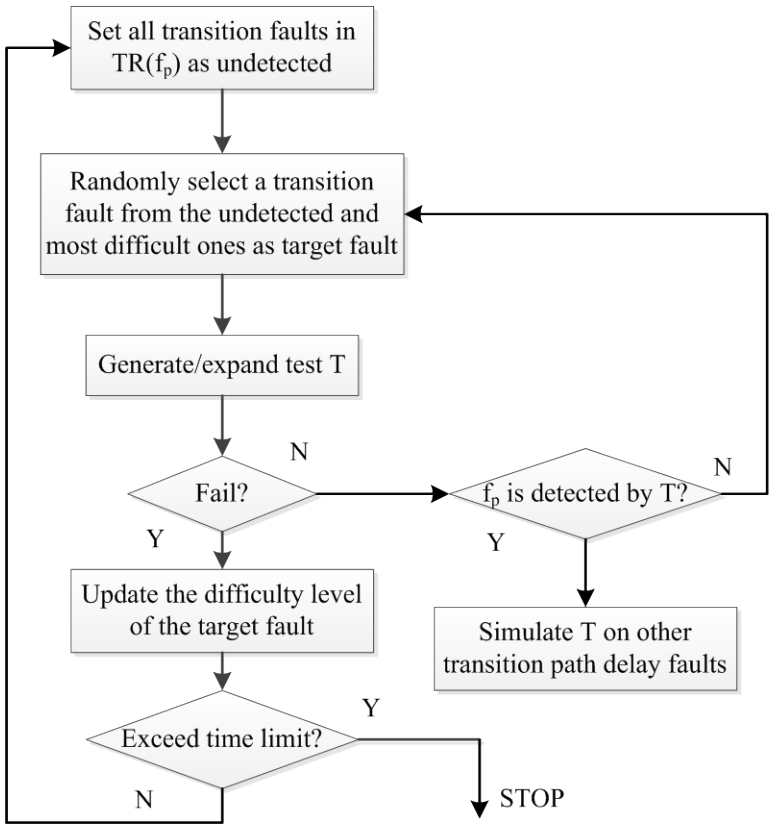

Fig. 2.2 Dynamic compaction heuristic procedure

2.3.5. Branch-and-Bound Procedure

The dynamic compaction heuristic procedure does not backtrack on decisions made based on transition faults in $TR(f_p)$ that were targeted earlier. Once the detection of a transition fault $tr_i(f_p)$ in $TR(f_p)$ fails, it discards the current test and starts to generate a new one. The branch-and-bound procedure described in this section is a complete deterministic procedure that backtracks on previously made decisions if the detection of a transition fault in $TR(f_p)$ fails.

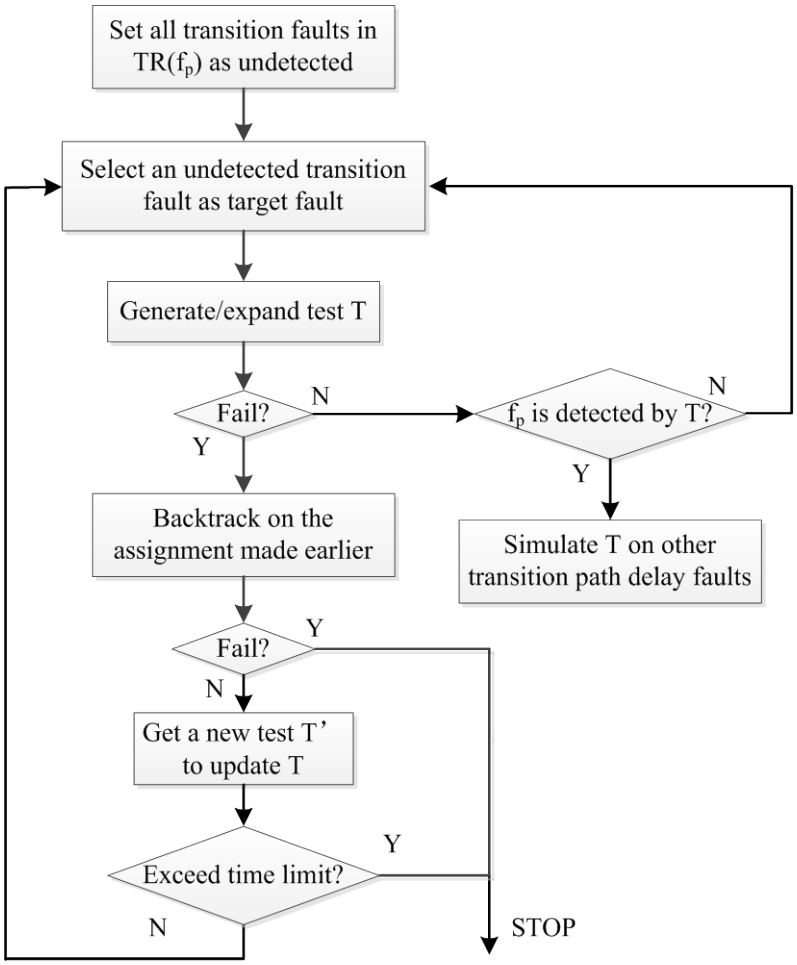

Fig. 2.3 Branch-and-bound procedure

The branch-and-bound procedure proceeds as shown in Fig. 2.3. At the beginning of the procedure for a transition path delay fault f_p, we apply the necessary assignments on input lines stored for f_p. We then select the transition fault that has the highest number of failures from the heuristic procedure to start generating the test. Next, we select one undetected transition fault as a secondary target fault to expand the test. If test generation for a transition fault fails, we backtrack on the assignments made earlier and get a new partially specified test where the last decision for which other options exist is reversed. Before continuing test generation, we check the validity of the new test by checking whether all the undetected transition faults under the new test are potentially detectable.

We first imply all the specified bits of the test, and then check whether any conflict exists between the line values and the necessary assignments of each undetected transition fault. If no conflict is found, we select an undetected transition fault to expand the test. Otherwise we keep backtracking. This branch-and-bound procedure runs until one of the following situations occurs. (1) A test is found and f_p is detected. (2) All the previously made decisions have been backtracked on and f_p is undetectable. (3) The run time limit for the branch-and-bound procedure is reached and f_p is aborted. (4) Since a test generator for transition faults is used in the branch-and-bound procedure, if the backtracking limit for transition faults is reached during test generation, f_p is aborted.

2.4. Experimental Results

The deterministic broadside test generation procedure described in section 2.3 was implemented in C++ on top of an existing test generation procedure for transition faults. Experiments were conducted on ISCAS89 benchmark circuits using a Sun Microsystems workstation which has two 450MHz CPUs, a 1024MB memory and a Solaris operating system. The run time limit for test generation for each transition path delay fault is 1 minute in the dynamic compaction heuristic procedure, and 2 minutes in the branch-and-bound procedure. The backtracking limit during test generation for transition faults is 128.

We enumerated all the paths for smaller circuits to generate the transition path delay fault list. The results for these circuits are shown in Table 2.1. For circuits with larger numbers of paths, we considered faults from the longest paths to the shorter ones until at least 1000 detected faults were found. The results for these circuits are shown in Table 2.2. In both Table 2.1 and Table 2.2, the first column identifies the circuit by name. The second to the sixth column show the number of transition path delay faults in the fault list, the number of detected faults, the number of undetectable faults, the number of aborted faults, and the total run time.

In considering the numbers of detected faults, it should be noted that they are similar to the numbers of conventional path delay faults that can be detected by strong non-robust tests. We verified that the numbers are identical for several of the circuits in Table

2.1. Therefore, there is no or little loss in fault coverage due to the use of transition path delay faults instead of conventional path delay faults [14].

Table 2.1 Results of test generation (enumerate all paths)

Circuit	No. of faults	No. of Det.	No. of Undet.	No. of Abr.	Run time
s27	56	25	31	0	00:00:00
s298	462	127	335	0	00:00:02
s344	710	259	451	0	00:00:23
s349	730	259	471	0	00:00:22
s382	800	165	635	0	00:00:04
s386	414	153	261	0	00:00:08
s444	1070	166	904	0	00:00:11
s510	738	197	541	0	00:00:22
s526	820	147	673	0	00:00:06
s641	3488	1121	2367	0	00:07:13
s713	43624	1090	42460	74	03:18:45
s820	984	369	615	0	00:00:52
s832	1012	369	643	0	00:01:22
s953	2312	961	1351	0	00:03:34
s1196	6196	3402	2793	1	00:49:15
s1238	7118	3363	3752	3	00:51:37
s1488	1924	722	1202	0	00:06:51
s1494	1952	723	1229	0	00:06:58

Table 2.2 Results of test generation (at least 1000 det. faults)

Circuit	No. of faults	No. of Det.	No. of Undet.	No. of Abr.	Run time
s1423	42782	1055	39746	1981	77:31:46
s5378	1948	1282	393	273	12:46:29
s9234	263916	1027	262757	132	08:57:07
s13207	735800	1244	734296	260	30:08:05
s35932	254400	1008	253300	92	07:21:07
s38417	70928	1227	64121	5580	225:56:15
s38584	1211890	1071	1210085	734	64:13:52

Table 2.3 Number of detected faults for sub-procedures (enumerate all paths)

Circuit	**Prep. Proc.**	**FSim Proc.**	**Heur. Proc.**	**Bran. Proc.**
s27	25	19	6	0
s298	163	104	22	1
s344	340	153	86	20
s349	340	158	82	19
s382	213	125	39	1
s386	231	138	13	2
s444	262	129	35	2
s510	377	170	27	0
s526	203	135	12	0
s641	1509	289	810	22
s713	1483	254	664	172
s820	580	316	50	3
s832	588	316	49	4
s953	1310	624	327	10
s1196	4535	1032	2216	154
s1238	4510	1153	2117	93
s1488	1495	588	127	7
s1494	1500	617	98	8

Table 2.4 Number of detected faults for sub-procedures (at least 1000 det. faults)

Circuit	**Prep. Proc.**	**FSim Proc.**	**Heur. Proc.**	**Bran. Proc.**
s1423	6063	0	106	949
s5378	1634	101	207	974
s9234	2418	6	273	748
s13207	6271	1	60	1183
s35932	2464	272	709	27
s38417	8871	1	183	1043
s38584	7637	220	238	613

Table 2.5 Run time comparison of sub-procedures (enumerate all paths)

Circuit	TG for Tran.	Prep. Proc.	FSim Proc.	Heur. Proc.	Bran. Proc.
s27	0:00	0:00	0:00	0:00	00:00:00
s298	0:01	0:00	0:00	0:01	00:00:00
s344	0:00	0:01	0:00	0:07	00:00:15
s349	0:01	0:00	0:00	0:06	00:00:15
s382	0:00	0:01	0:00	0:01	00:00:02
s386	0:01	0:01	0:00	0:02	00:00:04
s444	0:01	0:01	0:00	0:02	00:00:07
s510	0:02	0:01	0:00	0:06	00:00:13
s526	0:01	0:01	0:00	0:01	00:00:03
s641	0:00	0:02	0:00	0:33	00:06:38
s713	0:01	0:02	0:01	1:13	03:17:28
s820	0:06	0:02	0:01	0:11	00:00:32
s832	0:06	0:02	0:02	0:14	00:00:58
s953	0:03	0:06	0:02	1:39	00:01:44
s1196	0:02	0:07	0:04	4:32	00:44:29
s1238	0:03	0:08	0:04	4:57	00:46:24
s1488	0:13	0:07	0:03	1:34	00:04:54
s1494	0:13	0:07	0:03	1:40	00:04:54

Table 2.6 Run time comparison of sub-procedures (at least 1000 det. faults)

Circuit	TG for Tran.	Prep. Proc.	FSim Proc.	Heur. Proc.	Bran. Proc.
s1423	0:04	0:27	1:59	04:41:14	72:47:46
s5378	0:25	0:13	2:38	00:38:41	12:04:31
s9234	03:10	01:45	00:40:15	01:04:21	07:04:12
s13207	03:08	12:04	00:48:17	09:00:58	19:34:27
s35932	18:15	37:20	00:18:50	01:03:45	04:56:58
s38417	08:32	09:24	08:26:17	25:23:10	191:48:27
s38584	44:11	15:06	09:03:16	15:10:42	37:42:17

It can be seen from Table 2.1 and Table 2.2 that most of the transition path delay faults in the fault list are proven to be detected or undetectable. The number of aborted faults can be reduced by increasing the run time limit for the branch-and-bound procedure and the backtracking limit during test generation for transition faults. Given enough time, all the aborted faults should be identified as either detected or undetectable.

The sub-procedures of the developed test generation procedure are compared as follows. Table 2.3 and Table 2.4 show the number of detected transition path delay faults in each sub-procedure. The first column identifies the circuit by name. The second column shows the upper bound on the number of detectable transition path delay faults after undetectable faults were identified by the preprocessing procedure. The third to the fifth column show the number of detected transition path delay faults for the fault simulation procedure, the number of detected transition path delay faults for the dynamic compaction heuristic procedure, and the number of detected transition path delay faults for the branch-and-bound procedure.

Table 2.5 and Table 2.6 show the comparison of run time of each sub-procedure. The first column identifies the circuit by name. The second to the sixth column show the run time of test generation for transition faults, the run time of the preprocessing procedure, the run time of fault simulation, the run time of the heuristic procedure, and the run time of the branch-and-bound procedure.

Several observations can be made from Table 2.3, 2.4, 2.5 and 2.6. First, a large number of undetectable transition path delay faults are identified during the preprocessing procedure, and this procedure is more important for large circuits. Second, part of the detectable transition path delay faults can be detected by tests for transition faults, although the number may decrease for larger circuits. Third, the fault simulation procedure and dynamic compaction heuristic procedure contribute significantly to the final number of detected transition path delay faults while the total run time of these sub-procedures is much shorter than that of the branch-and-bound procedure. All of these show the efficiency of the developed procedure, compared with a pure complete deterministic test generation procedure.

3. PATH SELECTION BASED ON STATIC TIMING ANALYSIS CONSIDERING INPUT NECESSARY ASSIGNMENTS

In this chapter, a static timing analysis based path selection method is described. Fault detection conditions are taken into consideration during static timing analysis by using input necessary assignments. The arrival times of signals are refined during static timing analysis, and more accurate path delays that are closer to those obtained during test application can be obtained. The correlation between static timing analysis and timing of tests on silicon is therefore enhanced. Using input necessary assignments can also identify undetectable faults. As a result, a set of potentially detectable path delay faults, which are associated with critical paths based on more accurate estimates of the path delays that can be exhibited by a test set, is obtained.

3.1. Introduction

Since the number of paths throughout a circuit can be exponential in the number of lines, it is impractical to enumerate all the paths for test generation in large circuits. As a result, when the deterministic test generation procedure for transition path delay faults was performed on larger circuits in section 2.4, we only targeted a subset of the paths by considering the transition path delay faults from the longest paths to the shorter ones until at least 1000 detected faults were found. In addition, the number of undetectable faults can be very high, as shown in Table 2.2. To address these two issues, path selection procedures [30]-[45] select a subset of critical paths for test generation. Using the unit delay model, it is possible to select the longest paths in the circuit [30]. This method was used in section 2.4. It is also possible to consider every line in the circuit and select one of the longest paths going through the line [31][32]. To avoid selecting undetectable path delay faults, several path selection procedures that identify undetectable path delay faults

were described in [33]-[36]. Procedures that estimate path delays through timing analysis were described in [37]-[45]. Since both the delay difference between gates of different types and the interconnect delays are taken into account, more accurate path delay is obtained and so is the critical path identification. Possible variations in operating conditions such as voltage and temperature were taken into account during path selection in [37]. Statistical timing analysis techniques were used to incorporate information about deep sub-micron manufacturing defects, process variations and noise effects during path selection in [38]-[43], path correlations were considered in [43], and multiple input transitions were considered in [44]. To address the issue of false paths, several false path aware timing analysis procedures were proposed in [45]-[48].

Dynamic timing analysis can achieve more accurate estimates of path delays than static timing analysis because a set of input patterns is simulated to exercise all the paths in the circuit, as described in [38] and [49]. However, the computational complexity of generating such input patterns can be very high and the simulation is time-consuming for large circuits. Therefore, static timing analysis is preferred due to its fast run time and acceptable accuracy as mentioned in [44]. However, two limitations should be considered when static timing analysis is used for path selection. The first limitation is that the delay of a target path can be overestimated compared with the delay obtained during test application on silicon. During static timing analysis, the delay of a target path is calculated with all the lines in the circuit unspecified. During test application, values are assigned to all the lines in the circuit by a test for the path delay fault associated with the target path. These values propagate a transition along the path in a way that the corresponding path delay fault is considered as detected. The propagation conditions can be robust, strong non-robust, weak non-robust, and so on. Even with non-robust tests, the delay of a path may be smaller than the delay estimated when detection conditions are not considered. As a result, static timing analysis may not select path delay faults whose delays are the highest during test application, given that delays are modified due to the logic values assigned in order to satisfy the detection conditions of the faults. The second limitation is that static timing analysis cannot identify undetectable path delay faults.

The procedure described in this chapter addresses the inaccuracy of static timing analysis in predicting delays during test application without incurring the increased runtime of dynamic timing analysis. In general, we use the conditions necessary for fault detection to refine the static timing analysis process. Feeding these conditions back to the static timing analysis tool enhances the correlation between static timing analysis and actual timing of tests on silicon. The correlation may be further enhanced by considering process variations, but this issue will not be discussed.

Specifically, the procedure uses an existing static timing analysis tool to perform selection of the most critical path delay faults, allowing it to use a state-of-the-art process for estimating path delays. It addresses the limitations mentioned above and achieves more accurate timing estimates under the tests for the selected path delay faults by taking into account the conditions that need to be satisfied in order to detect a path delay fault. To ensure that these conditions can be considered by the static timing analysis tool, they are represented using what are called input necessary assignments [16]. Input necessary assignments are input values that must be assigned in order to detect the fault, and they can be given to the static timing analysis tool as input. They are derived by performing simple implications, without performing test generation. For a target path delay fault f_p that is identified as critical by traditional static timing analysis, the procedure finds the input necessary assignments of f_p and provides them to the static timing analysis process for path delay recalculation. The resulting delay is closer to the path delay that will occur during test application, compared with the delay obtained through traditional static timing analysis. The procedure also uses static timing analysis to identify other path delay faults whose delays are at least as high as that of f_p under the input necessary assignments of f_p. This information is used for updating the set of selected path delay faults. In addition, as discussed in [16], during the process of finding input necessary assignments of f_p, it is possible to determine that f_p is undetectable. The procedure uses this information to avoid the selection of undetectable faults.

We use the transition path delay fault model to develop the path selection procedure. The procedure can also be applied using a conventional path delay fault model, or to assess path delays when generating timing-aware tests for transition faults.

3.2. Input Necessary Assignments

Necessary assignments are values that a test for a fault must assign to lines in the circuit [29]. Input necessary assignments of a fault indicate the values that a test for the fault must assign to the input lines [16]. Input necessary assignments were computed in [16] as part of a broadside test generation process for path delay faults. The input necessary assignments were computed in polynomial time and provided a unified framework for identifying undetectable faults and for generating tests for detectable faults in [16].

To compute input necessary assignments for transition path delay faults in this chapter, we use a procedure similar to the procedure described in [16] and the preprocessing procedure described in section 2.3.2. As mentioned in chapter 2, a transition path delay fault f_p is associated with a path $p=g_1\text{-}g_2\ldots\text{-}g_k$ and a $v \rightarrow v'$ transition on g_1. We denote by $TR(f_p):=\{tr_1(f_p), tr_2(f_p), \ldots, tr_k(f_p)\}$ the set of transition faults associated with f_p. We denote by $InNecAssign(f_p)$ the set of input necessary assignments of f_p. Using the terminology from [16], each entry of $InNecAssign(f_p)$ has the form q[i]a, where q is an input of the combinational logic (a primary input or the output of a state variable), $i \in \{1,2\}$ is the pattern index (the first or second pattern of a broadside test), and $a \in \{0,1\}$ is the value of q. In [16], input necessary assignments are defined for the primary inputs under both patterns of a broadside test, and for present-state variables only under the first pattern. This is due to the fact that, in a broadside test, the present-state variables under the second pattern are implied by the first pattern. Thus, they cannot be specified arbitrarily under a broadside test. For the discussion in this chapter, we retain the constraints imposed by a broadside test, but we collect input necessary assignments corresponding to the primary inputs and the present-state variables under both patterns of the test. This will be useful when the input necessary assignments are given to a static timing analysis tool. The procedure for computing input necessary assignments proceeds in four steps.

Step 1 identifies f_p as undetectable if a transition fault in $TR(f_p)$ is undetectable. All the undetectable transition faults are identified by performing deterministic test

generation for transition faults. This step is not required if conventional path delay faults are targeted.

Step 2 finds the necessary assignments of faults in TR(f_p). For a transition fault $tr_i(f_p)$ in TR(f_p), a test must assign $g_i=v_i$ under the first pattern and $g_i=v_i$' under the second pattern. The simple forward and backward implications of these two assignments are used as the necessary assignments of $tr_i(f_p)$. Step 2 identifies f_p as undetectable if a conflict exists between the necessary assignments of faults in TR(f_p). Otherwise, the necessary assignments of all the faults in TR(f_p) are merged into a set denoted by DetCon(f_p). Each entry in DetCon(f_p) has the same form q[i]a as entries in InNecAssign(f_p) except that q can be any line in the circuit. Entries in DetCon(f_p) that assign values to input lines are added to InNecAssign(f_p).

Step 3 adds to DetCon(f_p) the propagation conditions that off-path inputs of f_p must satisfy. In order to propagate a transition fault on f_p from a gate input to a gate output, the off-path inputs of the gate must assume non-controlling values under the second pattern. For each gate that p goes through, every input line g of the gate except the one on p adds to DetCon(f_p) an entry of the form g[2]v, where v is the non-controlling value of the gate. Step 3 identifies f_p as undetectable if the implications of entries in the updated set DetCon(f_p) are not compatible with each other. Otherwise, newly specified lines and their values are added to DetCon(f_p) and InNecAssign(f_p) is updated.

Step 4 attempts to identify additional input necessary assignments of f_p by assigning both 0 and 1 to every unspecified input q under every pattern of the test. It identifies f_p as undetectable if the implications of both 0 and 1 on q are incompatible with entries in DetCon(f_p). Otherwise, an additional input necessary assignment can be found if only the implications of one assignment on q is compatible with entries in DetCon(f_p). It keeps performing the process on every unspecified input until f_p is identified as undetectable or no additional input necessary assignments can be found.

After these four steps, we either have f_p identified as undetectable or a set of input necessary assignments for it. In the latter case we refer to f_p as potentially detectable. We use the term potentially detectable since the fault may be undetectable even though its

input necessary assignments do not conflict. If an undetectable fault is selected, it will be identified as undetectable when test generation is performed for it.

3.3. Path Selection Procedure

In this section, a path selection procedure which is based on static timing analysis and input necessary assignments is described. The procedure consists of the following steps. It first uses traditional static timing analysis to obtain a set of critical paths. Each path is associated with a transition at its source, defining a path delay fault. Static timing analysis thus yields a set of path delay faults denoted by F_{Po}. The procedure computes input necessary assignments for the faults in F_{Po} and removes from F_{Po} faults that it can identify as undetectable based on their input necessary assignments. A required number of path delay faults are selected from the resulting F_{Po} as the initial set of selected path delay faults. The set of selected path delay faults is denoted by Target_PDF. For each target path delay fault f_p in Target_PDF, the procedure uses static timing analysis to recalculate the path delay under the input necessary assignments of f_p. It also identifies additional paths whose delays are at least as high as that of f_p under the input necessary assignments of f_p. These paths are used to update Target_PDF.

To select N path delay faults, the procedure uses traditional static timing analysis to select M>N paths. After removing undetectable faults, N faults are selected from the remaining ones. If fewer than N faults are obtained, M can be increased. With N faults in the selected set Target_PDF, the procedure updates the set gradually based on more accurate estimates of path delays.

A similar effect can be achieved by using traditional static timing analysis to select a set of K>M>N path delay faults, removing undetectable faults and updating the delays of the remaining faults as described in this chapter (but without adding new faults to the set), and then selecting the N most critical paths from the set. However, this would require K to be large enough to accommodate all the possible delay changes that may occur when input necessary assignments are used. If K is underestimated, the accuracy of the path selection procedure will be affected, and this will go undetected by the procedure. With the developed procedure, there is no need to determine K in advance. In addition, the

developed procedure that adds faults to Target_PDF computes input necessary assignments only until it finds N potentially detectable faults. If the set of selected faults is not updated based on recalculated delays, input necessary assignments must be found for K>M>N path delay faults.

In this section, we first describe how to perform static timing analysis with input necessary assignments considered, and then describe the path selection procedure in more detail. We use paths and path delay faults interchangeably since paths are associated with specific transitions at their sources.

3.3.1. Static Timing Analysis Considering Input Necessary Assignments

For each path delay fault f_p in Target_PDF, we use PrimeTime from Synopsys to perform static timing analysis and calculate the delay of the path p associated with f_p under the input necessary assignments of f_p as follows.

PrimeTime accepts specified input values only if an input is specified under both patterns of a test. Therefore, of all the input necessary assignments in InNecAssign(f_p), we only consider cases where both q[1]v and q[2]w appear in InNecAssign(f_p), for v, w $\in$[0,1]. For every such line q, we provide the assignment vw on q to PrimeTime by using the "set_case_analysis" command. Specifically, we use

set_case_analysis 0 for vw=00,

set_case_analysis rising for vw=01,

set_case_analysis falling for vw=10, and

set_case_analysis 1 for vw=11.

Static timing analysis is then performed in PrimeTime under the input necessary assignments of f_p. A ranked list of paths is produced and each path in the list has its delay under the input necessary assignments of f_p. We run PrimeTime such that it reports enough critical paths so that the path p associated with f_p is also reported. The list is used as described in section 3.3.2.

3.3.2. Path Selection

As mentioned earlier, the procedure starts by applying traditional static timing analysis to obtain an initial set of path delay faults F_{Po} of size M, for a constant M. M is

selected such that it is large enough to ensure that the required number of path delay faults can be selected from F_{Po} to initialize Target_PDF. Target_PDF is initialized based on F_{Po} as follows. Path delay faults in F_{Po} are considered in a decreasing order of their delays. Every potentially detectable path delay fault f_p in F_{Po} is added to Target_PDF until Target_PDF includes a predetermined number N of faults. Other potentially detectable path delay faults that have the same delay as the N^{th} fault added to Target_PDF are also added to Target_PDF.

Next, we update Target_PDF as follows. For each path delay fault f_p in Target_PDF, we recalculate the path delay under the input necessary assignments of f_p as described in section 3.3.1. From the ranked list of paths produced by PrimeTime based on f_p, paths whose delays are at least as high as that of f_p under the input necessary assignments of f_p are obtained. These paths are important since they can be as critical as f_p under the tests that detect f_p. To ensure that the most critical paths are selected, we consider every such fault f_p'. If f_p' is potentially detectable, we add it into Target_PDF if it is not already included. Every newly added path delay fault is processed in the same way as the faults already included in Target_PDF. We continue processing every path delay fault in Target_PDF until each path delay fault has its delay under its input necessary assignments and no new path delay faults are added to Target_PDF. Path delay faults in Target_PDF are sorted afterwards based on their recalculated delays, and N faults that have the highest delays can be selected for test generation. Fig. 3.1 shows the flow chart of the entire path selection procedure.

We take ISCAS89 circuit s13207 as an example to illustrate the procedure. Suppose that 16 most critical path delay faults in s13207 are expected for test generation. 1500 paths in s13207 are considered during traditional static timing analysis and the corresponding path delay faults are included in F_{Po}. Most of the faults are identified as undetectable and removed from consideration. The path delay faults initially included in Target_PDF are the first 16 faults shown in Table 3.1. The last 8 path delay faults are added to Target_PDF by the developed procedure as described later.

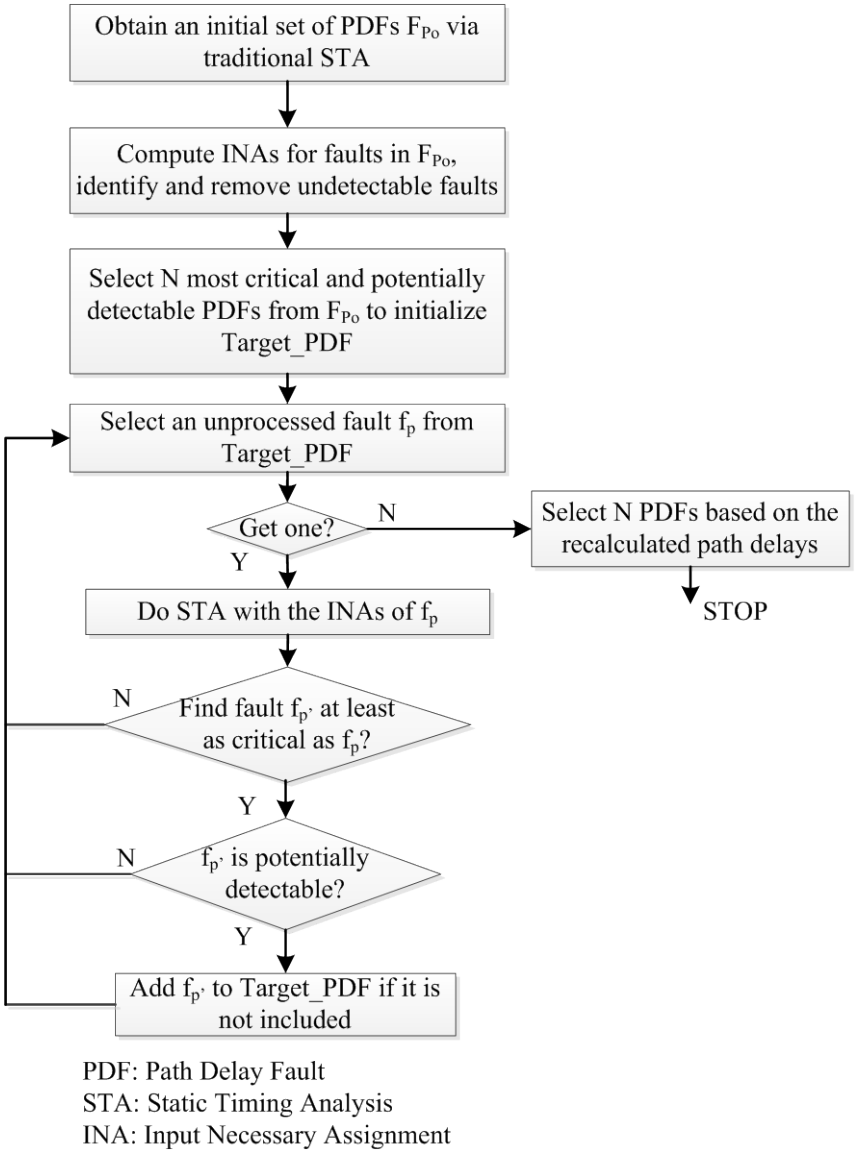

Fig. 3.1 Path selection procedure

In Table 3.1, the columns from left to right list the path delay faults by index, the delay obtained through traditional static timing analysis of each path delay fault, the recalculated delay of each path delay fault under the input necessary assignments of the fault, and newly identified potentially detectable path delay faults whose delays are at least as high as that of the target fault under the input necessary assignments of the target fault, where the target fault is the one shown in the leftmost column.

Table 3.1 Path selection in s13207

Path delay faults	orignial (ns)	final (ns)	new paths
f_p1	4.13	4.12	-
f_p2	4.11	4.10	-
f_p3	4.03	4.02	-
f_p4	4.01	4.00	-
f_p5	4.00	4.00	-
f_p6	3.93	3.92	-
f_p7	3.93	3.87	-
f_p8	3.91	3.91	-
f_p9	3.91	3.90	-
f_p10	3.89	3.86	-
f_p11	3.88	3.87	-
f_p12	3.88	3.84	-
f_p13	3.87	3.86	-
f_p14	3.87	3.81	-
f_p15	3.85	3.82	f_p17, f_p18
f_p16	3.85	3.82	-
f_p17	3.84	3.81	f_p19
f_p18	3.84	3.80	-
f_p19	3.83	3.83	f_p20, f_p21, f_p22, f_p23
f_p20	3.84	3.80	f_p24
f_p21	3.84	3.82	-
f_p22	3.83	3.80	-
f_p23	3.83	3.80	-
f_p24	3.81	3.76	-

After traditional static timing analysis, 16 potentially detectable path delay faults f_p1-f_p16 whose delays are among the highest in F_{Po} are included in Target_PDF. The path delay of each fault f_pi ($1 \leqslant i \leqslant 16$) is recalculated under its input necessary assignments. When f_p15 is considered, two new potentially detectable path delay faults f_p17 and f_p18 that have the same delay as f_p15 under the input necessary assignments of f_p15 are identified and added to Target_PDF. The path delay of each newly added fault is then recalculated under its input necessary assignments. A new potentially detectable path delay fault f_p19 that has a larger delay than f_p17 under the input necessary assignments of f_p17 is identified and added to Target_PDF. Similarly, four new potentially detectable path delay faults f_p20, f_p21, f_p22 and f_p23 that have larger delays than f_p19 under the input

necessary assignments of f_p19 are identified and added to Target_PDF. When f_p20 is targeted, f_p24 is identified and added to Target_PDF since it has the same delay as f_p20 under the input necessary assignments of f_p20. When f_p24 is targeted, no new path delay faults are identified and the procedure terminates. After the developed procedure, we have 24 path delay faults in Target_PDF.

Informally, the procedure obtains the transitive closure of the initial set of path delay faults based on the relations between path delays. Transitivity here implies the following: if f_pi is in the initial set, f_pj has a higher delay than f_pi under the input necessary assignments of f_pi, and f_pk has a higher delay than f_pj under the input necessary assignments of f_pj, then f_pk is added to the set (f_pj is added as well). In addition to the traditional static timing analysis applied in the beginning, static timing analysis considering input necessary assignments will be applied a number of times equal to the number of path delay faults in the set Target_PDF.

It can be observed from Table 3.1 that the delays of most path delay faults decrease after recalculation (the delays never increase since the use of input necessary assignments constrains the values that can be assigned to circuit lines). In addition, the rank of a path delay fault based on its delay changes. Considering two faults, their ranks can change in three ways. (1) Faults that have the same rank according to traditional static timing analysis may have different ranks after delay recalculation. For example, f_p6 and f_p7 have the same delay according to traditional static timing analysis, but f_p6 becomes more critical than f_p7 after delay recalculation. (2) Faults that have different ranks according to traditional static timing analysis may have the same rank after delay recalculation. For example, f_p14 has a larger delay than f_p17 according to traditional static timing analysis, but they are equally critical after delay recalculation. In this case, f_p17 was not included in the initial set of target faults, but it will be added to the set by the developed procedure. (3) Faults that have different ranks according to traditional static timing analysis may reverse their ranks after delay recalculation. For example, f_p7 has a larger delay than f_p9 according to traditional static timing analysis, but f_p9 becomes more critical than f_p7 after delay recalculation. As a result, the developed procedure may remove certain faults from the set of selected faults.

Because the rank of a fault changes and a new fault can be identified as critical after delay recalculation, the path delay faults selected for test generation through the developed procedure can be different from those selected through traditional static timing analysis. To select the 16 most critical and potentially detectable path delay faults in s13207 for test generation, f_p1-f_p16 are selected through traditional static timing analysis. However, through the developed procedure, f_p1-f_p13, f_p15-f_p16, f_p19 and f_p21 should be selected. f_p14 is discarded since it becomes less critical after delay recalculation. Two new faults f_p19 and f_p21 are selected instead. f_p21 is included since it has the same delay as f_p15 and f_p16.

3.4. Experimental Results

The path selection procedure was implemented in C++. PrimeTime and a simplified TSMC 0.18um technology library were used for static timing analysis. Experiments were conducted on ISCAS89 and ITC99 benchmark. For every circuit, we applied the procedure to select 100, 200, …, 1000 most critical path delay faults.

Table 3.2 compares the number of path delay faults in Target_PDF before and after path delay recalculation when different numbers of path delay faults are expected for test generation. In Table 3.2, the first column identifies the circuits by names and shows the size of F_{Po} in parentheses for every circuit. The number of path delay faults in Target_PDF before delay recalculation is shown in row “original”, and the number of faults in Target_PDF after delay recalculation is shown in row “final”. Taking s1423 and column “400” as an example, 400 potentially detectable path delay faults that have the highest delays are expected for test generation. Through traditional static timing analysis, 413 path delay faults are selected into Target_PDF. The last 13 path delay faults have the same delay as the 400^{th} fault. After delay recalculation, 425 path delay faults are included in Target_PDF. It can be observed from Table 3.2 that for many circuits, the final size of Target_PDF is larger than the corresponding original size. This is because additional path delay faults that are at least as critical as faults in the initial Target_PDF are identified by the developed procedure.

Table 3.2 Path group size comparison

Circuit		**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
s1423	original	101	202	304	413	502	605	707	806	909	1014
(54974)	final	109	206	325	425	546	707	737	841	941	1071
s5378	original	102	219	335	415	511	606	735	820	990	1076
(14802)	final	102	219	339	575	575	606	853	853	1335	1335
s9234	original	107	206	336	439	503	600	730	813	906	1005
(27738)	final	108	584	668	755	810	852	923	923	923	1005
s13207	original	101	202	300	420	501	609	700	808	905	1011
(80000)	final	173	308	382	841	1152	1232	1281	1404	1458	1532
s38417	original	105	205	306	432	511	626	742	812	1001	1001
(80000)	final	205	359	476	552	626	1001	1324	1517	1845	1845
s38584	original	114	209	310	409	548	651	773	874	972	1097
(80000)	final	114	209	310	426	557	670	773	890	981	1097
b11	original	100	210	301	404	505	608	707	805	904	1006
(57690)	final	100	211	301	410	506	610	713	805	905	1006
b12	original	105	200	300	401	506	600	701	800	908	1001
(223426)	final	113	201	307	405	507	602	701	800	915	1036

N most critical and potentially detectable path delay faults can be selected based on the recalculated path delays from the expanded Target_PDF. As mentioned in section 3.3.2, due to the newly identified path delay faults and the change of the ranks of path delay faults based on their delays, these N selected path delay faults may differ from those selected by traditional static timing analysis. For each circuit in Table 3.2, when $i\times 100$ ($1\leqslant i\leqslant 10$) path delay faults are expected for test generation, we select a set of the $i\times 100$ most critical path delay faults from the set Target_PDF obtained through the developed procedure, and compare these faults with the $i\times 100$ potentially detectable and most critical path delay faults selected based on the path delays obtained through traditional static timing analysis, i.e. the faults included in Target_PDF before delay recalculation. We exclude faults that can be identified as undetectable from comparison to show how the accuracy of path delay calculation affects the set of selected path delay faults. Some faults appear in both sets, while other faults are unique to one set. We count only the faults that are unique to one set. Table 3.3 shows this number. For example, if 500 path delay faults are selected for s1423, the developed method will select 15 path delay faults that are not selected by traditional static timing analysis with faults identified

as undetectable excluded. It can be observed from Table 3.3 that for most circuits, a different set of path delay faults is obtained through the developed procedure compared with the one obtained through traditional static timing analysis. The difference between the two sets is more significant for some circuits.

Table 3.3 Number of different path delay faults

Circuit	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
s1423	1	1	6	13	15	6	2	5	1	13
s5378	0	6	11	6	11	0	26	10	8	13
s9234	1	8	2	9	3	36	3	4	1	0
s13207	3	20	19	20	35	37	37	54	83	65
s38417	46	38	4	4	3	14	18	55	32	110
s38584	0	0	0	1	1	1	0	1	2	0
b11	0	2	1	4	4	2	3	4	1	0
b12	4	1	0	5	1	2	0	0	7	19

Table 3.4 Path delay comparison of s13207

Path delay(ns)	**f_p7**	**f_p10**	**f_p12**	**f_p14**	**f_p15**	**f_p18**	**f_p31**	**f_p36**	**f_p39**	**f_p42**
original	3.93	3.89	3.88	3.87	3.85	3.84	3.83	3.82	3.81	3.81
final	3.87	3.86	3.84	3.81	3.82	3.80	3.77	3.78	3.76	3.77
after TG	3.87	3.85	3.83	3.81	3.81	3.79	3.77	3.78	3.76	3.77
diff	0.06	0.03	0.04	0.06	0.03	0.04	0.06	0.04	0.05	0.04
diff_unit	2	1	1.3	2	1	1.3	2	1.3	1.7	1.3

To show how the developed procedure improves the accuracy of path delay calculation, we select a few critical path delay faults in s13207, generate a test for each selected fault, and compare its delay under the test with the delay obtained through traditional static timing analysis and the delay recalculated by the developed method. The result is shown in Table 3.4. In Table 3.4, the row “original” lists the path delay calculated by traditional static timing analysis for each path delay fault. The row “final” lists the delay recalculated by static timing analysis under the input necessary assignments of the fault. The row “after TG” lists the delay obtained by using static

timing analysis under a test for the fault. The row “diff” shows the difference between the path delays in rows “original” and “final”. The delay is shown in ns. In addition, for the technology considered, the lowest delay of any gate is the rising delay of an inverter, and it is equal to 0.03ns. Considering this as a unit delay, the row “diff_unit” shows the delay in the form of the number of inverters. It can be observed from Table 3.4 that for each path delay fault, the “original” delay is always larger than the “after TG” delay and the “final” delay is between the two delays. For all these 10 faults, the “final” delays are closer to the “after TG” delays. Taking f_p14 for example, the “final” delay is 0.06ns closer to the “after TG” delay than the “original” delay, i.e., the developed procedure improved the accuracy of the path delay of f_p14. For the technology considered, 0.06ns is equivalent to two inverter delays. Therefore, the delay of the path decreases by two inverter delays, which demonstrates the impact of the developed method on path delay.

Table 3.5 Path delay comparison

Circuit	**Pct. 1 %**	**Pct. 2 %**
s1423	83.9	78.9
s5378	32.4	39.2
s9234	38.1	61.15
s13207	98.9	86.34
s38417	64.2	29.44
s38584	14.3	21.68
b11	38	56.32
b12	85.75	88.82

For each circuit listed in Table 3.2, we select 1000 most critical paths by applying the developed procedure and compare the “original” delay, “final” delay and “after TG” delay for each path. The results are shown in Table 3.5. The first column identifies the circuits by names. The second column shows the percentage of path delay faults whose “original” delays are different from the “after TG” delays. Out of such faults, the percentage of faults whose “final” delays are closer to “after TG” delays is shown in the third column (We ignore faults whose “original” delays are the same as the “after TG” delays since the developed procedure cannot improve the accuracy of delay calculation in

such a case). It can be observed from Table 3.5 that for a large portion of the selected paths in a circuit, the developed procedure achieves estimates of path delays that are closer to the delays under tests for the path delay faults.

4. BUILT-IN GENERATION OF FUNCTIONAL BROADSIDE TESTS CONSIDERING PRIMARY INPUT CONSTRAINTS

This chapter describes a built-in functional broadside test generation method for a circuit that is embedded in a larger design. The functional constraints on the primary input sequences of the circuit are taken into consideration by using the functional input sequences of the design. The switching activity during built-in test generation is bounded within the peak switching activity that can occur in the circuit under the functional input sequences. An optional DFT approach based on state holding for improving fault coverage is also described. Experimental results show that the method can achieve high transition fault coverage for benchmark circuits using simple hardware.

4.1. Introduction

The deterministic broadside test generation method and path selection method described in previous chapters are applicable to off-line test generation, where tests are generated before test application. Typically, the tests are stored in an external tester and applied to the circuit by the tester. For large circuits with high clock frequency and complexity, delay testing via external tester can be expensive due to the following reasons. (1) In order to capture the delay defects that may fail the circuit during functional operation using a broadside test, the launch and capture clock edges should be triggered very fast especially for at-speed testing. As a result, the tester is required to provide a high-speed and accurate test clock. However, it is expensive to implement such a tester. One solution is to use on-chip clock source to provide a high-speed test clock as discussed in [50]-[54], so that the tester only needs to provide slower shifting clock and control signals. (2) Tests are stored in the memory of the tester before being applied. Considering that each pattern of a broadside test contains the values of all state variables

in the circuit, the test volume can be tremendous and may exceed the capability of the memory especially for large circuits. To address this issue, test compression techniques were proposed in [55]-[58] to alleviate the memory requirement in external testers. Built-in test generation [59], a design-for-test technique in which a circuit can be tested by extra test logic added to it, is a cost-effective solution to address both the issues. Using built-in test generation has the following advantages. (1) It facilitates at-speed testing. The added test logic is implemented using the same technology as the circuit, and it can share the functional clock of the circuit. Accurate timing behavior can be achieved during test application so that the circuit can be tested at its real operation speed. (2) It reduces test data volume. All the tests are generated on-chip by the test logic. The amount of memory required in the tester for storing test data is reduced. (3) It improves test quality. It is easy to apply a large number of tests with built-in test generation so that more detects, modeled or un-modeled, can be detected. N-detection [60] is naturally achieved and better test quality can be obtained.

Built-in test generation techniques were described in [61]-[68] to generate scan-based two-pattern tests for delay faults. In these techniques, arbitrary states can be scanned in during the application of the two-pattern tests. This makes it possible to bring the circuit into states that cannot be reached during functional operation. Although high fault coverage may be achieved since lines can be exercised under non-functional operation conditions, overtesting may occur due to the following reasons. (1) The switching activity in the circuit may be significantly higher than that under functional operation conditions. The excessive switching activity during non-functional operation conditions requires higher current which may cause the power supply voltage to drop and thus fail the circuit [18][19]. It also leads to higher power dissipation during test application than normal, which may cause permanent damage to the circuit. The problem can be severe under built-in test generation since the circuit operates at its real operation speed during test application. (2) Slow propagation paths that are never exercised during functional operation conditions can be sensitized under non-functional operation conditions and fail the circuit, as described in [20]. Due to overtesting, a circuit that

operates correctly under functional operation cannot pass the test, which results in unnecessary yield loss.

Several low-power built-in test generation techniques were described in [69]-[72] to reduce the power dissipation during test application. However, these methods do not guarantee that the power dissipation during test application would match the power dissipation that is possible during functional operation. Both higher and lower power dissipation are undesirable. To address the power dissipation as well as overtesting issues by creating functional operation conditions, a built-in test generation method was described in [73] to generate what are called functional broadside tests. Assuming that the primary inputs are unconstrained, a functional broadside test is a broadside test whose scan-in state is a reachable state, or a state that the circuit can enter during functional operation [21]. Under a broadside test, the state under the second pattern is the next-state obtained in response to the first pattern. Therefore, the state under the second pattern of a functional broadside test is also a reachable state. As a result, delay faults are detected under functional operation conditions when functional broadside tests are applied. Overtesing is therefore eliminated, and the power dissipation during the clock cycles where delay faults are detected is bounded within that possible during functional operation. In the method from [73], primary input sequences are generated on-chip and applied to the circuit starting from a known reachable state. The circuit traverses only reachable states, and functional broadside tests can be obtained from these primary input sequences and the corresponding reachable states.

A circuit is typically embedded in a larger design that constrains its primary input sequences. Fig. 4.1 shows two connected blocks B1 and B2 that may be part of a larger design. B2 is the target circuit, and its primary inputs are driven by part of the primary outputs of B1. During functional operation of the design, certain functional constraints can be imposed by B1 on the primary inputs of B2. Such functional constraints can affect the set of reachable states and the state-transitions that B2 can make during functional operation. Without considering these constraints, certain state-transitions that cannot occur during functional operation may be allowed during test application by the method in [73]. As a result, the switching activity during test application may exceed that

possible during functional operation, and overtesting may occur. Ignoring primary input constraints is acceptable when the circuit is designed to operate correctly as a stand-alone circuit. When this is not the case, it is necessary to consider the primary input constraints for built-in generation of functional broadside tests.

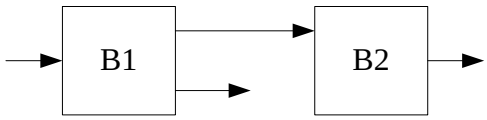

Fig. 4.1 Example of an embedded block

In this chapter, we extend the built-in functional broadside test generation method from [73] to address the issue of primary input constraints for a circuit that is embedded in a larger design. To take primary input constraints into account, it is possible to first extract the constraints imposed on the circuit as described in [74][75], and then extend the primary input sequence generation logic to incorporate the functional constraints. However, it is typically not possible to completely represent the functional constraints in closed form and synthesize simple hardware to satisfy them [76]. To avoid the effort for constraint extraction and the synthesis of complex hardware, we use functional input sequences of the complete design to capture the constraints. Functional input sequences may be generated for other purposes such as speed binning or design verification. If they are not available, they can also be derived using high level simulations of application programs. Functional broadside tests can be extracted from functional input sequences for embedded logic blocks, as described in [76][77]. In this chapter, the primary input constraints captured by functional input sequences are taken into account by using the peak switching activity that can occur in the circuit under these sequences. Among the state-transitions that can be used for on-chip test generation in [73], only those whose switching activities are no higher than the peak switching activity under the functional input sequences are eligible for use. The developed method generates primary input

sequences that only allow the circuit to make eligible state-transitions starting from a reachable state.

By avoiding the use of unreachable states, functional broadside tests may not detect delay faults that can be detected by unrestricted broadside tests. Although such delay faults may not cause the circuit to fail during functional operation, they may accelerate the deterioration of the circuit and affect its long-term reliability. In addition, detecting such faults can be important for failure diagnosis and process improvement. In this chapter, we provide an optional DFT method based on state holding to detect such faults and improve fault coverage. State holding keeps the values of some state variables from changing in certain clock cycles during on-chip test generation. As a result, gates in the fanout cones of the unchanging state variables are likely to keep their values, and the switching activity may be reduced. State holding was used to reduce the power dissipation during the application of structural scan-based tests in [78]-[80] and to improve delay fault coverage in [80]. In this chapter, it is used to introduce unreachable states to detect faults that cannot be detected by functional broadside tests. Although high switching activity may be caused by the introduced unreachable states, it may be compensated by state holding to some extent. To avoid excessive switching activity, only tests whose switching activities are no higher than the peak switching activity under the functional input sequences can be generated on-chip. A simple simulation-based procedure is described to select sets of state variables for holding. Experimental results show that for benchmark circuits, the developed built-in test generation method can achieve high transition fault coverage using simple hardware.

4.2. Generic Built-in Test Generation

This section reviews the generic built-in test generation method, which typically requires three additional logic blocks for a circuit: a test pattern generator (TPG), an output response analyzer (ORA), and a controller, as shown in Fig. 4.2. The TPG logic generates test patterns for the primary inputs and scan chains. The ORA logic compacts and analyzes the test responses observed at the primary outputs and scan chains to determine the correctness of the circuit. The control logic controls the circuit, the TPG

and the ORA logic so that test application and response analysis can be conducted properly.

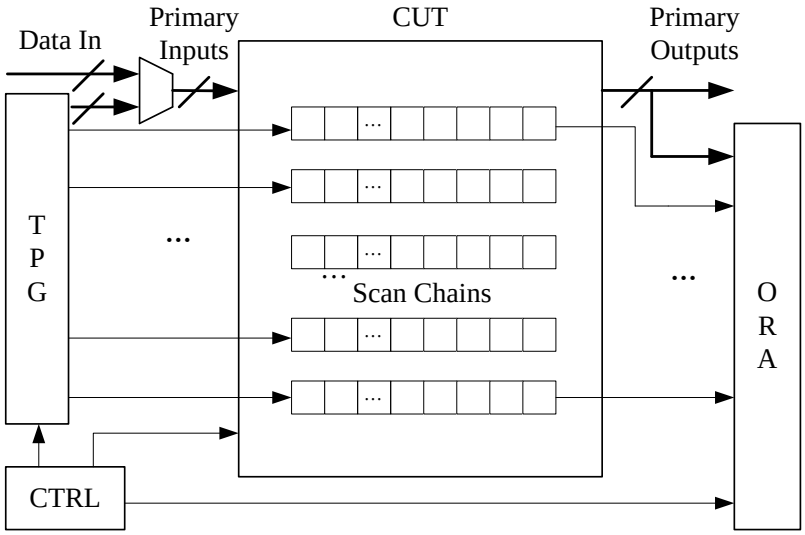

Fig. 4.2 Generic built-in test generation architecture

The TPG logic is usually implemented based on a linear feedback shift register (LFSR) [59] whose states are used as pseudo-random patterns. An n-stage LFSR is constructed from n D flip-flops and a number of modulo-2 adders (XOR gates), as shown in Fig. 4.3. C_i implies whether there is a connection between Q_n and the modulo-2 adder, for $1 \leq i < n$. If $C_i=1$, Q_n is connected to the modulo-2 adder. Otherwise, the modulo-2 adder can be considered as an interconnect wire. Under a particular combination of the values of C_i for $1 \leq i < n$, the LFSR can cycle through all possible 2^n-1 states except the all-0 state. The probability that 0 or 1 appears on each LFSR bit is 1/2. Various techniques such as LFSR-reseeding [81], bit fixing [82], bit flipping [83], and weighted random pattern generation [84]-[87] were developed for LFSR based TPG logic so that the pseudo-random patterns generated by the TPG logic can achieve higher fault coverage.

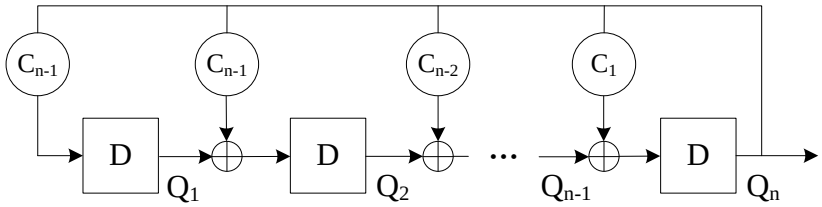

Fig. 4.3 An n-stage LFSR

The ORA logic is usually implemented by using a multiple-input signature register (MISR) [59]. A MISR is derived from an LFSR as shown in Fig. 4.4. D_i is an input of the MISR, and C_i implies the connection between Q_n and the modulo-2 adder, for $1 \leqslant i \leqslant n$. The test response of the circuit is captured by the MISR and compacted into its next-state value. The final state of the MISR is compared with the expected value after test application. Faults are detected if a mismatch is identified.

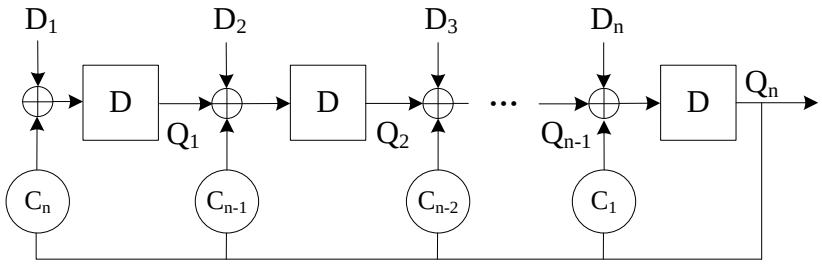

Fig. 4.4 An n-stage MISR

4.3. Built-in Generation of Functional Broadside Tests with Unconstrained Primary Input Sequences

This section reviews the built-in functional broadside test generation method from [73]. The architecture of the method is shown in Fig. 4.5. Different from the generic built-in test generation architecture shown in Fig. 4.2, the TPG logic in this method only generates test patterns for the primary inputs of the circuit.

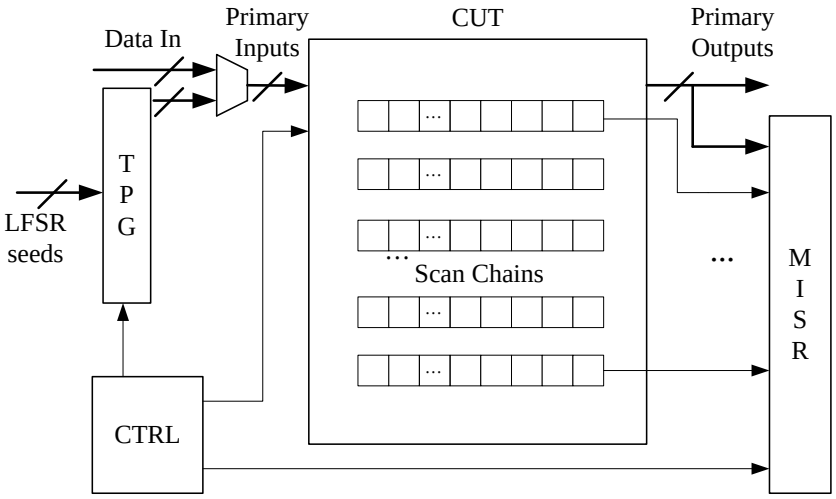

Fig. 4.5 Built-in generation of functional broadside tests

In this method, the circuit is first initialized into a reachable state $s_{initial}$. The TPG logic generates a primary input sequence P of a fixed length L. Let P=p(0)p(1)p(2)⋯p(L-1), where p(i) is the primary input vector at clock cycle i, for $0 \leqslant i < L$. P is applied to the circuit in functional mode and takes it through a state sequence S=s(0)s(1)s(2)⋯s(L), where $s(0)=s_{initial}$ and s(i) is the next-state the circuit enters when its primary inputs are driven by p(i-1) and its present state is s(i-1), for $0 < i \leqslant L$. A functional broadside test can be defined by any two consecutive time units from the primary input sequence P and its corresponding state sequence S. The test that starts at clock cycle i is denoted by t(i)=<s(i), p(i), s(i+1), p(i+1)>. The final state of t(i) is s(i+2).

The application of t(i) takes the circuit through states s(i), s(i+1) and s(i+2). The application of t(i+1) takes the circuit through states s(i+1), s(i+2) and s(i+3). An overlap of s(i+1) and s(i+2) occurs between t(i) and t(i+1). In order to apply t(i+1) after t(i) is applied, special hardware is needed to bring the circuit back to s(i+1) from s(i+2). To avoid such hardware, it is required in this method that there is no overlap between any two tests. Tests are applied every 2^q clock cycles, where $q \geq 1$. A $\log_2 L$-bit clock cycle counter is used to track the current clock cycle during the application of P, and its rightmost q bits are fed into a q-input NOR gate to generate an apply signal that indicates

when to apply the tests, as shown in Fig. 4.6. In this chapter, we have q=1 so that the largest number of functional broadside tests can be obtained. The rightmost bit of the clock cycle counter is used as the test apply signal and no extra NOR gate is needed.

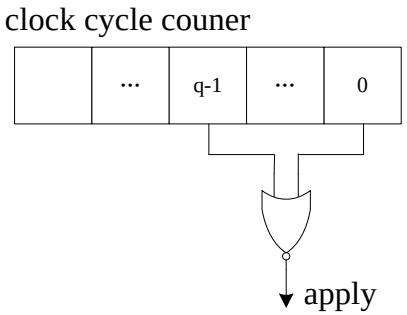

Fig. 4.6 Clock cycle counter and test apply signal generation in [73]

When t(i) is applied at clock cycle i, the primary output vector y(i+1) produced by the circuit in response to <s(i+1), p(i+1)> at clock cycle i+1 and the final state of the test s(i+2) at clock cycle i+2 are captured by the MISR. s(i+2) is shifted into the MISR over a number of clock cycles equal to the length of the longest scan chain. The circuit is brought back to s(i+2) by using circular shift so that the test application process can be continued.

The TPG logic in this method is implemented by using an LFSR whose states are used as pseudo-random vectors for the primary inputs of the circuit. The pseudo-random primary input sequence is modified by inserting additional logic gates based on a primary input cube C to avoid what is called repeated synchronization [88]. Repeated synchronization occurs when a primary input value causes a state variable to have a certain value. This value is repeated every time the primary input value appears in the primary input sequence, potentially preventing faults from being detected. The value C(i) of primary input i under C is the value that should appear more often on input i, and it is calculated by a software procedure in the following way. A specified value 0(1) is assigned to input i with all the other inputs and the present-state variables unspecified.

The number of specified next-state variables is then counted. If assigning 0(1) to input i synchronizes fewer state variables than assigning 1(0), we have C(i)=0(1). Otherwise, we have C(i)=x.

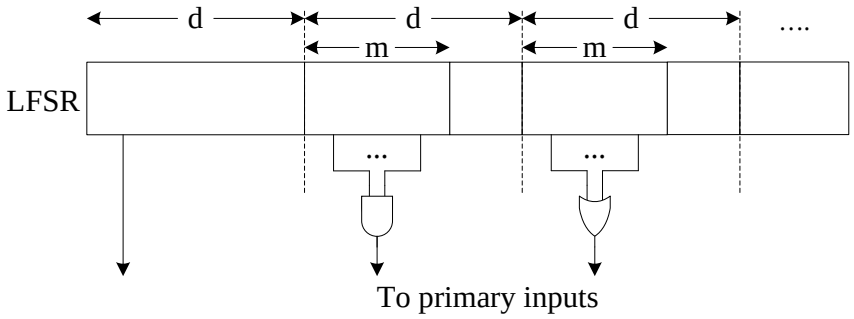

Fig. 4.7 The TPG logic in [73]

Fig. 4.7 shows the TPG logic in the method from [73]. To reduce the correlation between adjacent primary inputs, a distinct set of d LFSR bits is used to determine the values for each primary input. The number of LFSR bits is denoted by N_{LFSR}. For a circuit with N_{PI} primary inputs, we have $N_{LFSR}=d \cdot N_{PI}$ in this method. According to the primary input cube C, if C(i)=0(1), m out of the d bits allocated for primary input i are used to bias the probabilities of 0 and 1 on input i to avoid repeated synchronization, where $2 \leqslant m \leqslant d$. If C(i)=0, the m bits are fed into an m-input AND gate. In this case, a 0 is more likely to appear on input i, and the probability for a 0 is $1-1/2^m$. If C(i)=1, the m bits are fed into an m-input OR gate. In this case, a 1 is more likely to appear on input i, and the probability for a 1 is $1-1/2^m$. If C(i)=x, no logic gate is inserted and one of the d bits is used to drive input i directly.

To avoid using an LFSR whose length is proportional to the number of primary inputs, we use the following approach in this chapter. An LFSR with a fixed number of bits is used for the TPG logic. The LFSR drives a shift register whose states are used for driving the primary inputs of the circuit, as shown in Figure 4.8.

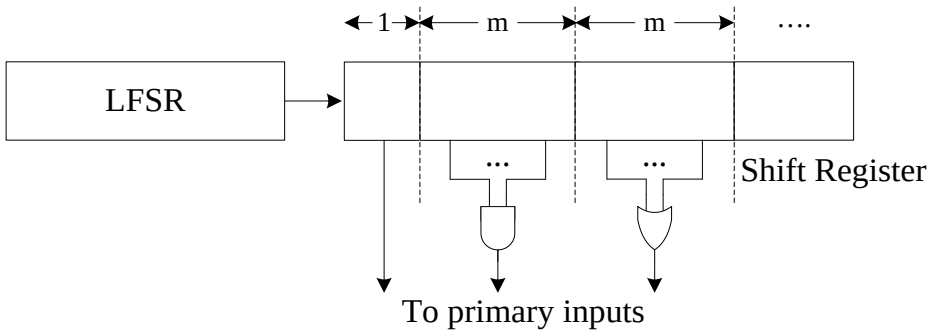

Fig. 4.8 The TPG logic in the developed method

For an input i, if C(i)=0(1), a distinct set of m shift register bits is used to determine the values for it. All the m bits are fed into an m-input AND(OR) gate so that 0(1) is more likely to appear on input i. If C(i)=x, a single shift register bit is used to determine the values for input i. The shift register contains $m \bullet N_{SP} + (N_{PI}-N_{SP})$ bits, where N_{SP} is the number of primary inputs whose values under C are specified. After the LFSR is initialized, it takes a number of clock cycles equal to the size of the shift register to initialize the shift register before primary input sequence generation.

Multiple primary input sequences are applied in [73] by using different LFSR seeds. A simulation-based seed selection procedure selects useful seeds among random seeds. For a random seed, the procedure computes the primary input sequence obtained from it, and checks whether the resulting tests can detect additional faults. If so, the seed is selected. Otherwise, the seed is discarded and a new random seed is considered. The procedure continues until the last U sequences cannot detect additional faults for a constant U. The number of selected seeds is then reduced. We reduce the number of selected seeds by using a procedure similar to reverse order fault simulation called forward-looking fault simulation [89].

4.4. Built-in Generation of Functional Broadside Tests with Constrained Primary Inputs

This section describes the developed built-in functional broadside test generation method that considers primary input constraints for on-chip test generation.

As discussed earlier, primary input constraints affect the set of state-transitions that the circuit can make during functional operation. Since the constraints cannot be represented in closed form and satisfied by using simple hardware, functional input sequences are used in this paper to capture them. The peak switching activity that can occur in the circuit under the functional input sequences is used to bound the switching activities of the functional broadside tests generated on-chip. The circuit may traverse states that it cannot traverse during functional operation with constrained primary inputs. However, the states are not arbitrary in the sense that the circuit can traverse them with unconstrained primary inputs during functional operation. In addition, the switching activity will match the switching activity that is possible during functional operation with constrained primary inputs. This alleviates overtesting caused by excessive switching activity. The peak switching activity obtained under the functional input sequences is denoted by SWA_{func}.

For a TPG sequence P=p(0)p(1)p(2)…p(L-1), we use SWA(i) to denote the switching activity during clock cycle i. SWA(i) is defined as the percentage of lines whose values in clock cycle i are different from their values in clock cycle i-1. SWA(0) is undefined. It is possible to identify subsequences of P such that the switching activities of the corresponding functional broadside tests do not exceed SWA_{func}. Let S=s(0)s(1) s(2)…s(L) be the state sequence the circuit traverses starting from s(0) under P. A subsequence $P_{k,w}$=p(k)p(k+1)⋯p(w-1) ($0 \leqslant k < w \leqslant L$) of P yields functional broadside tests t(k), t(k+2), t(k+4), ⋯. The tests are considered as acceptable if $SWA(i) \leqslant SWA_{func}$, for k<i<w. Although not all the clock cycles are important for test application in terms of the switching activity, for simplicity, it is required that all would satisfy the switching activity bound.

Table 4.1 shows an example. The columns in Table 4.1 show the state s(i), the primary input vector p(i), and the switching activity SWA(i) during clock cycle i. The

switching activity that is higher than SWA_{func} is marked in bold. To avoid the excessive switching activity at clock cycles j+1 and u+1, it is possible to use $P_{0,j}$, $P_{j+1,u}$ and $P_{u+1,L}$ for application of functional broadside tests. Note that p(j) and p(u) are avoided in order to avoid the transitions from s(j) to s(j+1) and from s(u) to s(u+1). When a subsequence $P_{k,w}$ is applied for on-chip test generation, a new LFSR seed should be loaded so that the TPG logic can generate the subsequence starting from p(k). In addition, the circuit must be initialized into s(k). This can be done by shifting s(k) into the scan chains. However, extra memory is required for storing the scan-in state. The amount of memory due to storage of scan-in states can be large if the number of state variables and the number of stored scan-in states are high.

Table 4.1 Example of primary input subsequence selection

Clock cycle i	**s(i)**	**p(i)**	**SWA(i)**
0	s(0)	p(0)	-
1	s(1)	p(1)	SWA(1)
…	…	…	…
j-1	s(j-1)	p(j-1)	SWA(j-1)
j	s(j)	p(j)	SWA(j)
j+1	s(j+1)	p(j+1)	**SWA(j+1)**
j+2	s(j+2)	p(j+2)	SWA(j+2)
…	…	…	…
u-1	s(u-1)	p(u-1)	SWA(u-1)
u	s(u)	p(u)	SWA(u)
u+1	s(u+1)	p(u+1)	**SWA(u+1)**
u+2	s(u+2)	p(u+2)	SWA(u+2)
…	…	…	…
L-1	s(L-1)	p(L-1)	SWA(L-1)
L	s(L)	-	SWA(L)

Several different reachable states can be used as initial states if the amount of required memory for storing these states is not a concern. We only use one reachable state $s_{initial}$ to initialize the circuit in this chapter. We attempt to find primary input sequences so that when they are applied to the circuit starting from $s_{initial}$, the switching

activity during each clock cycle does not exceed SWA_{func} and the fault coverage is as high as possible. In the example of Table 4.1, we only use $P_{0,j}$ and discard other subsequences. To avoid the potential loss in fault coverage, we extend $P_{0,j}$ as described next.

We construct a primary input sequence from segments where each segment is obtained through the TPG logic using a different LFSR seed. We use $P_{multi}=P_{seg}(0)P_{seg}(1)\ldots P_{seg}(N_{seg}-1)$ to denote a multi-segment primary input sequence, where $P_{seg}(i)$ is a primary input segment for $0\leq i<N_{seg}$, and N_{seg} is the number of segments included in the entire sequence. During the application of P_{multi} to a circuit starting from $s_{initial}$, different LFSR seeds are loaded at certain clock cycles to generate the primary input segments. The state of the circuit is held by deactivating the clock driving the circuit when a new LFSR seed is loaded, so that the application of the new primary input segment can start from the final state of the previous segment. The primary input segments are selected so that during the application of P_{multi}, the switching activity in each clock cycle is no higher than SWA_{func}.

A simulation-based procedure selects $P_{seg}(i)$ and constructs P_{multi} as follows. Initially we have $P_{multi}=\emptyset$. A primary input sequence P of length L is generated using the TPG logic based on a random LFSR seed. The procedure applies P to the circuit starting from state s_{start} through logic simulation. If i=0, $s_{start}=s_{initial}$. Otherwise, s_{start} is the final state of $P_{seg}(i-1)$. The procedure examines the switching activity in each clock cycle under P until the first violation $SWA(j+1)>SWA_{func}$ is identified at clock cycle j+1. Since the tests are obtained every two consecutive clock cycles, we have $P_{seg}(i)=P_{0,j}$ if j is even, or $P_{seg}(i)=P_{0,j-1}$ if j is odd, so that the final state of $P_{seg}(i)$ is the final state of the last test obtained from $P_{0,j}$. Then the procedure checks whether the tests obtained from $P_{seg}(i)$ detect additional faults. If so, $P_{seg}(i)$ is concatenated to P_{mulit} and the procedure starts selecting $P_{seg}(i+1)$. Otherwise, the current seed fails and a new seed is considered for selecting $P_{seg}(i)$ again. The procedure stops constructing P_{multi} if the last R seeds fail to select $P_{seg}(i)$ for a constant R. To obtain more tests, the procedure attempts to construct multiple multi-segment primary input sequences. An attempt fails if $P_{seg}(0)$ cannot be selected. The procedure stops constructing new multi-segment primary input sequences

when the last Q attempts fail for a constant Q. Figure 4.9 shows the flow chart of the entire construction procedure.

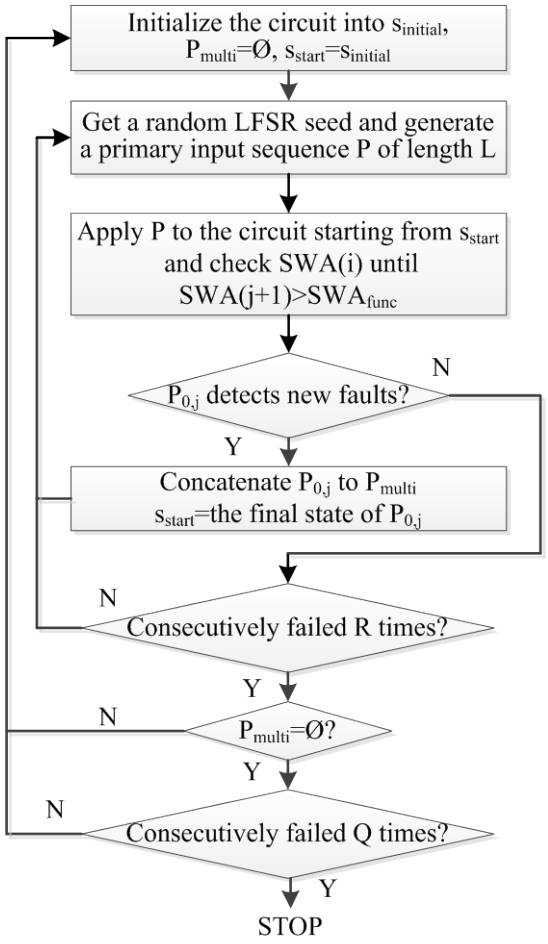

Fig. 4.9 The multi-segment primary input sequence construction procedure

Using multi-segment primary input sequences, the circuit only needs to be initialized into $s_{initial}$ before a new multi-segment primary input sequence is applied. To apply the multi-segment primary input sequences on-chip, a $\log_2 L_{max}$-bit clock cycle counter tracks the current clock cycle and generates the test apply signal every two clock cycles, where L_{max} is the length of the longest primary input segment. A $\log_2 L_{sc}$-bit shift counter tracks the number of shift operations during circular shifting, where L_{sc} is the length of the

longest scan chain. A $\log_2 N_{segmax}$-bit segment counter tracks the number of applied segments, where N_{segmax} is the largest number of segments contained in a multi-segment primary input sequence. A $\log_2 N_{multi}$-bit sequence counter tracks the number of applied multi-segment primary input sequences, where N_{multi} is the number of sequences. The clocks for the TPG logic, the counters and the circuit are gated and controlled by a finite state machine, so that the TPG logic and the counters can operate simultaneously or not with the circuit under different operation modes such as seed loading, shift register initialization, circuit initialization, primary input sequence application, and circular shifting. For example, when the clock cycle counter reaches the length of the current segment and a new segment needs to be applied, the clock that drives the circuit is disabled so that the state of the circuit is held. The clock for the TPG logic is still enabled so that a new LFSR seed can be loaded and the shift register can be initialized. The clock for the circuit is then enabled again so that the circuit can operate simultaneously with the TPG logic for the application of the new segment.

4.5. Built-in Test Generation with State Holding

This section describes an optional DFT method based on state holding for fault coverage improvement. The method can be used after applying the functional broadside tests generated on-chip if necessary. By keeping the values of some state variables from changing in certain clock cycles during on-chip test generation, state holding may introduce unreachable states that can be used to detect faults that cannot be detected by functional broadside tests.

4.5.1. State Holding

State holding can be implemented by the structure shown in Fig. 4.10. A latch-based clock gating cell is used to gate the clock for a state variable. When the state holding enable signal Hold_en is high, the clock input of the state variable is 0 when a clock edge arrives and no data can be captured. The value of the state variable is therefore held. Multiple state variables can be held simultaneously by sharing the same gated clock signal.

A state variable can be held over arbitrary number of consecutive clock cycles as long as the holding enable signal is high. However, in order to define a broadside test t(i)=<s(i), p(i), s(i+1), p(i+1)> based on the state sequence resulting from state holding, we require that no state variable is held during the transition from s(i+1) to s(i+2) to avoid potential fault coverage loss. The reason is that the fault effects activated by <s(i+1), p(i+1)> are expected to be captured by the state variables during the transition from s(i+1) to s(i+2). If state holding is performed, the fault effects may not be captured and the fault will not be detected.

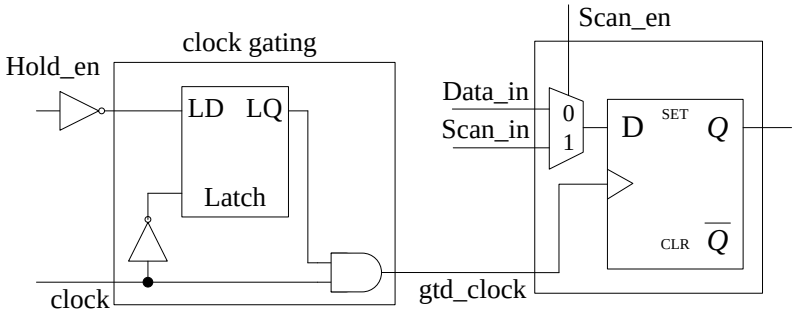

Fig. 4.10 Implementation of state holding

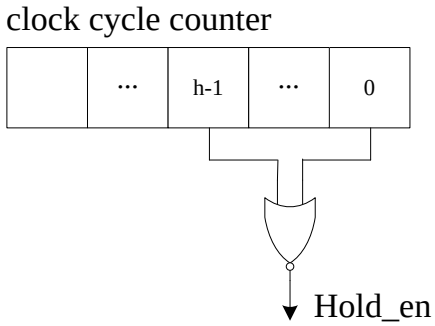

Fig. 4.11 Holding enable signal generation

State holding can be performed periodically during on-chip test generation. In this chapter, we perform state holding every 2^h clock cycles, where $h \geq 1$. The clock cycle

counter is used to generate the holding enable signal by inserting an h-input NOR gate at its rightmost h bits as shown in Fig. 4.11, so that state holding can be performed in the next clock cycle. Given a predetermined value for h and a set of state variables for holding, the multi-segment sequence construction procedure described in section 4.4 is used to construct the multi-segment primary input sequences and select LFSR seeds for the given set. The impact of holding state variables in the given set during on-chip test generation is taken into account when the state sequence under a primary input sequence is computed via logic simulation. SWA_{func} is used to ensure that the switching activity in each clock cycle under the multi-segment primary input sequences is no higher than it, so that possible excessive switching activity caused by unreachable states can be avoided.

4.5.2. Set Selection for State Holding

Multiple sets of state variables can be used for holding. State variables in different sets are controlled by different holding enable signals. In this chapter, a new set is used only after all the multi-segment primary input sequences for the current set have been applied.

We use a simulation-based set selection procedure to select the sets of state variables for holding. Let Set_{ini} be the set containing all the state variables in the circuit, and let F_r be the set of faults that cannot be detected by the functional broadside tests generated in section 4.4. The procedure selects a number of subsets of Set_{ini}. Each selected subset can help detect additional faults in F_r. A state variable can be included in different selected subsets. However, compared with the case where the state variable is included in only one selected subset, more gating logic is needed on the clock of the state variable so that it can be held properly every time a subset it belongs to is enabled for holding. Such gating logic may affect the performance of the clock network under functional mode. To avoid this issue, the procedure only considers non-overlapping subsets of Set_{ini}.

The set selection procedure first partitions Set_{ini} into non-overlapping subsets, and then selects the subsets that can help detect additional faults in F_r. We use Det to denote the detecting ability of a subset, i.e. the number of faults in F_r detected by tests resulting from holding the subset during on-chip test generation. In order to partition Set_{ini} in a way such that more faults in F_r are likely to be detected by holding the resulting subsets, the

procedure first examines the detecting abilities of different subsets of Set_{ini} from larger ones to smaller ones in the following way. Fig. 4.12 shows a full and complete binary tree with a height of H, where each node represents a subset $Set_{i,j}$ of Set_{ini}. We have the root $Set_{0,0}=Set_{ini}$. A pair of child nodes $Set_{i+1,2j}$ and $Set_{i+1,2j+1}$ are obtained by randomly partitioning their parent node $Set_{i,j}$ into halves. The detecting ability of every subset is examined from the root node to the leaf nodes until each node $Set_{i,j}$ has its $Det_{i,j}$ associated with it. To compute $Det_{i,j}$, the procedure first constructs multi-segment primary input sequences for $Set_{i,j}$, and then simulates the tests obtained from the multi-segment sequences on F_r. The number of detected faults in F_r is the value of $Det_{i,j}$.

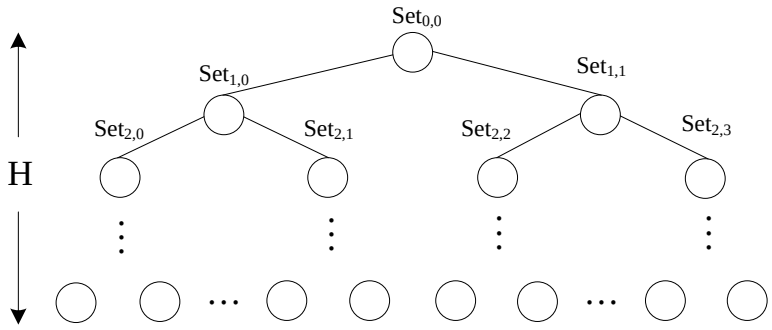

Fig. 4.12 Full and complete binary tree for set selection

After obtaining the detecting ability for each node, the procedure checks all the nodes from the leaves to the root to decide whether a subset should be partitioned or not. Take node $Set_{i,j}$ for example. If $Set_{i,j}$ is a leaf node, $Set_{i,j}$ is set to Ø if $Det_{i,j}=0$, and $Set_{i,j}$ remains the same if $Det_{i,j}>0$. If $Set_{i,j}$ is a parent node, its two child nodes $Set_{i+1,2j}$ and $Set_{i+1,2j+1}$ are checked as follows. If $Det_{i,j} \leq \max\{Det_{i+1,2j}, Det_{i+1,2j+1}\}$, which indicates that more faults can be detected if $Set_{i+1,2j}$ and $Set_{i+1,2j+1}$ are held separately, we have $Set_{i,j}=\{Set_{i+1,2j}, Set_{i+1,2j+1}\}$, i.e. $Set_{i,j}$ is partitioned into $Set_{i+1,2j}$ and $Set_{i+1,2j+1}$, and $Det_{i,j}=\max\{Det_{i+1,2j}, Det_{i+1,2j+1}\}$. If $Det_{i,j}>\max\{Det_{i+1,2j}, Det_{i+1,2j+1}\}$, which indicates that more faults can be detected if $Set_{i+1,2j}$ and $Set_{i+1,2j+1}$ are held together, $Set_{i,j}$ and $Det_{i,j}$ remain the same. Note that $Set_{i,j}=Set_{i+1,2j} \cup Set_{i+1,2j+1}$ is not used in this case since it may

cause $Set_{i,j}$=$Set_{i+1,2j}$ if $Set_{i+1,2j+1}$ is already updated to Ø. By performing the operations on each node, Set_{ini} can be partitioned into a number of non-overlapping subsets after $Set_{0,0}$ is processed. Then for each of such subsets, the procedure constructs multi-segment primary input sequences for it, and selects the subset if its resulting tests can detect additional faults in F_r.

By using the set selection procedure, N_h non-overlapping sets of state variables can be selected for holding. In order to perform on-chip test generation with the N_h sets to hold, extra hardware is required in addition to that required for applying multi-segment primary input sequences as described in section 4.4. We use a $\log_2 N_h$-bit set counter to track the number of used sets and a $\log_2 N_h$ to N_h decoder to select a set, as shown in Fig. 4.13. A new set of state variables is enabled for holding when the sequence counter reaches the number of multi-segment primary input sequences for the current set. The on-chip test generation with state holding terminates when the set counter reaches N_h.

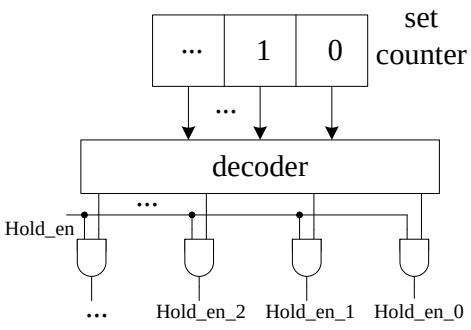

Fig. 4.13 Set selection signal generation

4.6. Experimental Results

The developed built-in test generation method was implemented and applied to ISCAS89, ITC99 and IWLS2005 benchmark circuits. Fastscan from Mentor Graphics was used for logic and fault simulation. The method was implemented using Perl, Tcl, and CShell scripts.

Table 4.2 lists the benchmark circuits used for the experiments and shows the parameters of the circuits. The columns from left to right show the name of the benchmark circuit, the number of primary outputs N_{PO}, the number of primary inputs N_{PI}, the number of primary inputs N_{SP} whose values are specified in the primary input cube C (or the number of logic gates inserted to avoid repeated synchronization), and the number of state variables N_{SV}.

We implemented the hardware required for the developed method in Verilog. The MISR and the shift register on the primary inputs were not included. Primary inputs of an embedded block are typically driven by registers, and the registers can be reused by the developed method. Extra shift register bits may be needed for a primary input whose value is specified in the primary input cube C to avoid repeated synchronization. However, the number of such primary inputs is small, as shown in Table 4.2. The extra logic gates inserted to avoid repeated synchronization were included for area calculation.

Table 4.2 Parameters for benchmark circuits

Circuit	**N_{PO}**	**N_{in}**	**N_p**	**N_{SV}**
s35932	320	35	1	1728
s38584	278	12	2	1164
b14	54	32	0	215
b20	22	32	0	430
spi	45	45	3	229
wb_dma	215	215	17	523
systemcaes	129	258	1	670
systemcdes	65	130	1	190
des_area	64	239	0	128
aes_core	129	258	2	530
wb_conmax	1416	1128	8	770
des_perf	64	233	0	8808

In order to evaluate the area overhead, the benchmark circuits and the hardware for the developed method were logic synthesized by using Design Compiler from Synopsys and a 0.18um generic library. As a result, the transition fault coverage achieved for a benchmark circuit may be different from that achieved in other works. For a small circuit, the area overhead of built-in test generation in general, considered as a percentage of the

circuit area, may be high. Only larger benchmark circuits where the area overhead of built-in test generation is acceptable are listed in Table 4.2.

Primary input constraints are created for benchmark circuits by connecting pairs of circuits such that all the primary inputs of one circuit are driven by the primary outputs of the other. The target circuit is the one whose primary inputs are constrained [76][77]. When two circuits are paired, it is ensured that the number of primary outputs of the driving block is no less than the number of primary inputs of the target circuit. We consider all possible combinations of the benchmark circuits listed in Table 4.2. We also allow the driving block to be a duplication of the target circuit if it does not have more primary inputs than primary outputs. In addition, we consider the case where there are no primary input constraints by using a block named “buffers”, which is a group of buffers placed at the primary inputs of the target circuit, as the driving block for comparison. We assume that all the benchmark circuits can be initialized into the all-0 state. The initialization can be performed by shifting in the all-0 state or asserting a global reset if it is available.

To determine the value of SWA_{func}, the complete design is simulated under 30 functional input sequences of length 30000 generated by the TPG logic. For simplicity, if the target circuit is not driven by “buffers”, we use the TPG logic designed for the driving block as the TPG logic for the complete design. Otherwise, the TPG logic for the target circuit is used. In this case, the value of SWA_{func} indicates the peak switching activity in the target circuit when there are no primary input constraints. After the simulation of functional input sequences, the target circuit is considered alone with its own TPG logic.

We have N_{LFSR}=32 and m=3 for the LFSR and shift register configuration. For the built-in generation of functional broadside tests considering primary input constraints, the multi-segment primary input sequence construction procedure stops constructing a sequence when it consecutively fails 3 times to select a segment, i.e. R=3. It stops constructing new sequences when the last 5 attempts fail, i.e. Q=5. The value of L is selected so that it is suitable for the target circuit. The number of state variables in the benchmark circuits varies from 128 to 8808. We assume that a circuit can have no more

than 10 scan chains and the length of a scan chain should be at least 100. All the scan chains are of approximately equal length.

Table 4.3 shows the results of the built-in generation of functional broadside tests considering primary input constraints. The first column shows the name of the target circuit and the total number of transition faults after fault collapsing in parentheses. The second column shows the length of the longest scan chain. The third column shows the name of the driving block. In addition to “buffers”, the driving blocks that cause the highest and lowest SWA_{func} are listed for every target circuit in order to show a range of possible results under primary input constraints. For wb_conmax, only the case where it is driven by a duplication of itself is listed since other circuits have fewer primary outputs than its primary inputs. When “buffers” is the driving block, the multi-segment primary input sequences are constructed with no constraint on the switching activity. Therefore, each primary input segment is of length L.

The fourth column shows the number of multi-segment primary input sequences N_{multi}. The fifth column shows the maximum number of segments contained in a multi-segment sequence. The sixth column shows the length of the longest primary input segment. The seventh column shows the value of SWA_{func}. The switching activity is given as a percentage of switching lines in the circuit during a state-transition. The eighth to the tenth column shows the number of selected LFSR seeds, the number of applied tests, and the peak switching activity during test application. The eleventh column shows the achieved transition fault coverage. The lowest fault coverage is obtained in the case where SWA_{func} is the lowest, and this case is included for every circuit. The twelfth column shows the area of the hardware required for on-chip test generation. The thirteenth column shows the area overhead, given as a percentage of the hardware in the circuit.

It can be observed from Table 4.3 that for the benchmark circuits, SWA_{func} is lower when the target circuit is driven by a block other than “buffers”. This demonstrates the influence of primary input constraints on the switching activity during functional operation. Tests whose switching activities exceed SWA_{func} are not acceptable when primary input constraints are considered. In cases where SWA_{func} does not decrease much

under primary input constraints, compared with the peak switching activity in the case of no primary input constraints, there is no or a small loss of fault coverage. One reason is that tests whose switching activities are higher than SWA_{func} may not necessarily detect additional faults. The exclusion of such tests does not affect the fault coverage. Another reason is that multi-segment primary input sequences use more LFSR seeds and allow the circuit to traverse longer state sequences to compensate for the potential fault coverage loss, such as the case where b14 is driven by systemcdes. However, when SWA_{func} decreases much, such as the case where s35932 is driven by spi, a noticeable fault coverage loss may occur. The reason is that many tests that can improve the fault coverage are excluded because of higher switching activity.

The number of applied tests varies from hundreds to hundreds of thousands for different target circuits. This is because the primary input sequences are based on random pattern generation and the target circuits have different numbers of random pattern resistant faults. It can also be observed that the area of the required hardware does not change much for different target circuits, and the area overhead is smaller for larger circuits.

To show how the fault coverage is improved by using state holding, we consider the cases in Table 4.3 where the fault coverage achieved by functional broadside tests is lower than 90%. The set selection procedure in section 4.5.2 was used to select the subsets of state variables for state holding first. A full and complete binary tree with a height of 6 was used for each target circuit during set selection. We have R=Q=1 when the multi-segment sequence construction procedure was used to compute the detecting ability for a subset, and we have R=3 and Q=5 when it was used to construct primary input sequences for a selected subset. State holding was performed every 4 clock cycles during on-chip test generation.

The results of built-in test generated with state holding are shown in Table 4.4. The first column shows the name of the target circuit. The second column shows the name of the driving block. The third to the twelfth column show the number of sets of state variables selected for holding, the total number of state variables included in the selected sets, the number of multi-segment primary input sequences applied, the maximum

number of segments contained in a multi-segment primary input sequence, the length of the longest primary input segment, the number of selected LFSR seeds, the number of tests applied on-chip, the peak switching activity during test application, the additional transition fault coverage contributed by using state holding, and the final transition fault coverage. The thirteenth column shows the area of the hardware required for performing both the built-in generation of functional broadside tests and the built-in test generation with state holding. The fourteenth column shows the area overhead, given as a percentage of the hardware in the circuit.

It can be observed from Table 4.4 that by using state holding, a noticeable fault coverage improvement can be achieved. Although unreachable states may be introduced by state holding, the switching activity during test application is always bounded within SWA_{func}. In addition, the area overhead does not increase much when state holding is also performed, compared with that when only the built-in generation of functional broadside tests is performed, i.e. the extra area overhead caused by state holding is small.

In summary, the developed method can achieve high fault coverage and bounded switching activity during test application for the benchmark circuits using simple hardware. However, several limitations should be considered when using the developed method. (1) The method uses switching activity to evaluate the deviations of a state-transition during on-chip test generation from the state-transitions that can occur during functional operation. Although overtesting caused by excessive switching activity can be alleviated since state-transitions whose switching activities exceed that possible during functional operation are excluded, certain state-transition that can never occur during functional operation may still be allowed during on-chip test generation. As a result, overtesting caused by slow paths sensitized by non-functional operation conditions may occur. (2) The set selection procedure described in section 4.5.2 does not guarantee that the highest fault coverage improvement can be achieved by holding the selected sets, and unnecessary state variables can be included in the selected subsets. (3) All the benchmark circuits used in this chapter are single-clock-domain designs, in which it is straightforward to obtain functional broadside tests every two consecutive clock cycles from the primary input sequences and the corresponding state sequences. However, for

circuits with multiple clock domains, the method cannot be directly applied since the frequency difference between clock domains needs to be taken into account, which may complicate the test application strategy and control logic.

Table 4.3 Results of built-in test generation considering primary input constraints

Circuit	**L_{sc}**	**Driving block**	**N_{multi}**	**N_{segmax}**	**L_{max}**	**SWA_{func} %**	**N_{seeds}**	**N_{tests}**	**SWA %**	**FC %**	**HW Area (um^2)**	**Area Over. %**
s35932 (22698)	173	buffers	1	1	6000	43.48	1	3000	39.93	94.94	12455	1.73
		aes_core	1	1	6000	43.33	1	3000	39.93	94.94	12455	1.73
		spi	19	6	44	23.08	48	254	23.08	87.33	12599	1.75
s38584 (30844)	117	buffers	22	50	18000	35.46	104	936000	33.70	84.65	14252	2.53
		des_area	22	50	18000	34.21	104	936000	33.70	84.65	14252	2.53
		wb_conmax	63	25	1996	30.61	194	38134	30.61	82.38	13674	2.43
b14 (20236)	108	buffers	10	12	12000	42.65	33	198000	41.76	80.72	13652	5.68
		systemcdes	15	16	12000	41.63	37	222000	41.12	80.72	13652	5.68
		aes_core	16	18	12000	39.44	49	274591	39.44	80.23	13795	5.73
b20 (45126)	108	buffers	32	44	6000	39.66	118	354000	37.21	79.05	13924	2.77
		aes_core	32	44	6000	39.53	118	354000	37.21	79.05	13924	2.77
		spi	62	19	1884	31.91	147	40911	31.91	78.23	13641	2.71
spi (10970)	115	buffers	126	8	18000	23.34	188	1692000	23.26	93.20	14035	9.24
		wb_conmax	118	8	18000	21.58	159	1341126	21.47	92.66	14035	9.24
		wb_dma	83	8	18000	15.58	221	832220	15.58	90.13	14035	9.24
wb_dma (13842)	105	buffers	24	16	18000	23.28	66	594000	22.44	70.36	14421	5.61
		wb_conmax	16	18	12596	18.26	52	90583	18.26	68.75	14273	5.55
		s35932	13	22	12498	17.82	61	49845	17.82	68.33	14273	5.55

Table 4.3 Results of built-in test generation considering primary input constraints (cont.)

Circuit	**L_{sc}**	**Driving block**	**N_{multi}**	**N_{segmax}**	**L_{max}**	**SWA_{func} %**	**N_{seeds}**	**N_{tests}**	**SWA %**	**FC %**	**HW Area (um^2)**	**Area Over. %**
systemcaes (29272)	112	buffers	18	1	18000	19.62	18	162000	19.49	75.86	13401	3.18
		wb_conmax	20	1	18000	19.39	20	180000	19.27	75.84	13401	3.18
		s35932	15	11	18000	18.10	43	76267	18.10	74.46	13826	3.27
systemcdes (10222)	100	buffers	1	5	1000	42.97	5	2500	40.69	99.77	12309	9.15
		wb_dma	1	5	1000	42.43	5	2500	40.69	99.77	12309	9.15
		s38584	1	5	1000	40.87	5	2500	40.69	99.77	12309	9.15
des_area (17800)	128	buffers	1	4	1000	39.99	4	2000	39.83	99.84	12096	7.28
		wb_conmax	1	5	1000	39.79	5	2500	39.42	99.84	12273	7.37
		des_area	119	9	22	29.96	288	578	29.96	98.97	12747	7.64
aes_core (79316)	106	buffers	2	8	1000	32.83	9	4500	31.42	99.94	12475	1.76
		wb_conmax	2	8	1000	32.74	9	4500	31.42	99.94	12475	1.76
		s35932	2	8	1000	31.46	9	4500	31.42	99.94	12475	1.76
wb_conmax (146970)	110	buffers	71	10	18000	17.69	131	1179000	16.87	92.03	14367	1.28
		wb_conmax	29	15	18000	15.76	143	384086	15.76	90.17	14076	1.25
des_perf (318412)	881	buffers	1	3	1000	37.46	3	1500	36.74	99.99	12676	0.27
		wb_conmax	3	11	470	35.74	16	1434	35.74	99.99	13118	0.27
		s38584	21	8	62	32.03	51	1049	32.03	97.86	12972	0.27

Table 4.4 Results of built-in test generation with state holding

Circuit	Driving block	N_h	N_{bits}	N_{multi}	N_{segmax}	L_{max}	N_{seeds}	N_{tests}	SWA %	FC Imp. %	Final FC %	HW Area (um^2)	Area Over. %
s35932	spi	1	1728	6	6	56	27	286	23.08	5.62	92.95	12760	1.77
s38584	buffers	2	1164	27	13	18000	69	621000	32.90	5.27	89.92	14755	2.62
	des_area	2	1164	27	13	18000	69	621000	32.90	5.27	89.92	14755	2.62
	systemcaes	2	1164	65	25	18000	118	342442	30.61	5.65	88.03	14899	2.64
b14	buffers	4	22	16	13	12000	33	198000	41.33	13.45	94.17	14753	6.11
	systemcdes	4	27	20	12	12000	43	258000	41.01	13.40	94.12	14915	6.17
	systemcaes	6	33	32	12	12000	58	284372	39.44	13.83	94.06	15603	6.44
b20	buffers	1	430	7	18	6000	27	81000	36.99	10.70	89.75	14085	2.80
	spi	1	430	7	18	6000	27	81000	36.99	10.70	89.75	14085	2.80
	s38584	1	430	26	14	6000	51	72498	31.91	10.55	88.78	14092	2.80
wb_dma	buffers	6	507	20	10	18000	56	504000	22.38	6.49	76.85	16082	6.21
	wb_dma	6	507	46	14	18000	97	275198	18.26	8.01	76.76	16405	6.33
	s35932	6	507	28	6	18000	74	145768	17.82	7.85	76.18	16242	6.27
systemcaes	buffers	2	670	29	1	18000	29	261000	20.05	7.25	83.11	13905	3.29
	wb_conmax	2	670	30	1	18000	30	270000	19.39	7.27	83.11	13905	3.29
	s35932	2	670	40	5	18000	70	302386	18.09	8.31	82.77	14648	3.46

5. CONCLUSIONS

The rapid increase in clock frequency and complexity of digital integrated circuits necessitates delay testing. This dissertation presented methods for three aspects of delay testing in scan-based circuits: deterministic test generation for a new path delay fault model, path selection for test generation, and built-in generation of functional broadside tests.

We first described a deterministic broadside test generation procedure for transition path delay faults. The transition path delay fault model captures both small and large delays along a path. The detection of a transition path delay fault requires that all the individual transition faults along the path are detected by the same test. To reduce the computational complexity of test generation, five sub-procedures were used: a deterministic test generation procedure that generates tests for transition faults and identifies undetectable transition faults, a preprocessing procedure that identifies undetectable transition path delay faults without performing test generation, a fault simulation procedure that identifies transition path delay faults that can be detected by the tests for transition faults, a dynamic compaction heuristic procedure that generates tests without backtracking on decisions made for previously detected faults, and a complete branch-and-bound procedure that backtracks on previously made decisions. Experimental results showed that for most of the transition path delay faults in benchmark circuits, either a test is found or the fault is identified as undetectable.

Next, we described a procedure based on static timing analysis to select critical paths for test generation. The procedure considers input necessary assignments during static timing analysis to obtain path delays that are closer to those that can be obtained under tests for path delay faults. This is based on the observation that traditional static timing analysis process does not take into consideration logic conditions that are necessary for

detecting a path delay fault. Such conditions are important since they may affect the path delays. The input necessary assignments are a subset of these logic conditions. Using input necessary assignments to refine arrival times of signals enhances the correlation between static timing analysis and timing of tests on silicon. Input necessary assignments can also be used for identifying undetectable faults. For a set of path delay faults obtained through traditional static timing analysis, the procedure calculates more accurate path delays, which are closer to the delays that can be obtained under tests that detect them. It also identifies path delay faults, whose delays are at least as high as the selected path delay faults under the input necessary assignments of the selected faults, in order to ensure that the most critical paths during test application can be selected.

Finally, a built-in test generation method for functional broadside tests was described for a circuit embedded in a larger design, taking the primary input constraints on the circuit into consideration. Functional input sequences for the design are used to capture the primary input constraints. Primary input sequences are generated on-chip and applied to the circuit starting from a reachable initial state. Tests are obtained from the primary input sequences and the corresponding state sequences. The primary input constraints are satisfied by ensuring that the peak switching activity that can occur under the primary input sequences is no higher than that possible under the functional input sequences. An optional DFT method based on state holding was also described to improve fault coverage. The method introduces unreachable states by keeping the values of some state variables from changing in certain clock cycles during on-chip test generation. As a result, faults that cannot be detected by functional broadside tests may be detected. Experimental results showed that high transition fault coverage can be achieved by the developed method for benchmark circuits using simple hardware.

5.1. Future Work

For the developed built-in functional broadside test generation method, there are several directions we can work on in the future to improve the method.

In the developed method, switching activity is used as a metric to evaluate the deviations of a state-transition during on-chip test generation from the state-transitions

under functional operation conditions. The primary input constraints are satisfied by bounding the switching activity during on-chip test generation within the peak switching activity that can occur in the target circuit under the functional input sequences for the design. Overtesting caused by excessive switching activity can therefore be alleviated. However, overtesting caused by slow paths sensitized by non-functional operation conditions may still occur. An alternative metric to evaluate the deviations is pattern of signal-transitions [90]. Pattern of signal-transitions was defined in [90] as a set of the switching lines during a state-transition, and each line in the set is associated with a specific transition. The size of the set is the switching activity during the state-transition. Using pattern of signal-transitions, we can require that a state-transition is allowed during on-chip test generation only if its pattern of signal-transitions is a subset of the pattern of signal-transitions of a state-transition that occurs under the functional input sequences. This requirement not only guarantees that the switching activity of the state-transition is no higher than that possible during functional operation, but also guarantees that only signal transitions that can occur during functional operation are allowed. As a result, overtesting caused by both excessive switching activity and slow paths sensitized by non-functional operation conditions can be alleviated.

Since the set selection procedure in the developed method cannot guarantee to achieve the highest fault coverage improvement and unnecessary state variables may be included in the select sets, an advanced procedure can be developed so that the achieved fault coverage improvement is the highest and no unnecessary state variable is selected.

The developed method was applied on single-clock-domain designs. For circuits with multiple clock domains, the frequency difference between clock domains must be taken into account during on-chip test generation. The clock domains should operate at their own speeds so that reachable states can be obtained properly. In addition, multi-cycle tests may be needed to detect both intra-clock-domain and inter-clock-domain faults. This implies more complicated test application strategy and built-in test generation control logic. Investigations are needed so that the method can be applied on multi-clock-domain circuits.

LIST OF REFERENCES

LIST OF REFERENCES

[1] J.A. Waicukauski, E. Lindbloom, B. Rosen and V. Iyengar, "Transition Fault Simulation", *IEEE Design and Test*, Apr. 1987, pp.32-38.

[2] J. P. Lesser and J. J. Shedletsky, "An Experimental delay test generator for LSI logic", *IEEE Trans. on Computers*, March, 1980, pp.235-248.

[3] G. L. Smith, "Model for delay faults based upon paths", *Proc. Int. Test Conf.*, 1985, pp. 342-349.

[4] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits", *IEEE Trans. Computer-Aided Design*, Vol. 6, No. 5, Sept. 1987, pp.694-703.

[5] S. M. Reddy, M. K. Reddy, and V. D. Agrawal, "Robust tests for stuck-open faults in CMOS combinational logic circuits", *Proc. Int. Symp. on Fault-Tolerant Computing*, 1984, pp.44-49.

[6] E. S. Park and M. R. Mercer, "Robust and nonrobust tests for path delay faults in combinational circuits", *Proc. Int. Test Conf.*, 1987, pp.1027-1034.

[7] N. Jha and S. Gupta, "Testing of digital systems" (Cambridge University Press, 2003)

[8] Y. Shao, I. Pomeranz, and S. M. Reddy, "On generating high quality tests for transition faults", *Proc. Asian Test Symp.*, 2002, pp.1-8.

[9] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability", *J. of Des. Autom. and Fault Tolerant Comput.*, 1978, Vol. 2, pp.165-178.

[10] S.Dasgupta, R. G. Walther, T. W. Williams, and E. B. Eichelberger, "An enhancement to LSSD and Some Applications of LSSD in Reliability, Availability and Serviceability", *Proc. Fault-Tolerant Compt. Symp.*, 1981, pp.880-885.

[11] S. Patil and J. Savir, “Skewed Load Transition Test: Part I, Calculus”, *Proc. Int. Test Conference*, 1992, pp.705-713.

[12] J. Savir and S. Patil, “On Broad-Side Delay Test”, *Proc. VLSI Test Symp.*, 1994, pp.284-290.

[13] Xijiang Lin, Ron Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli, “High-frequency, at-speed scan testing”, *IEEE Design & Test of Computers*, Vol. 20 , No. 5, 2003, pp.17-25.

[14] I. Pomeranz and S. M. Reddy, “Transition Path Delay Faults: A New Path Delay Fault Model for Small and Large Delay Defects”, *IEEE Trans. VLSI Syst.*, Vol. 16, No. 1, January 2008, pp.98-107.

[15] S. Hassoun and T. Sasao, Logic Synthesis and Verification, Springer, 2002.

[16] I. Pomeranz and S. M. Reddy, “Input Necessary Assignments for Testing of Path Delay Faults in Standard-Scan Circuits”, *IEEE Trans. VLSI Syst.*, Vol. 19, No. 2, November 2009, pp. 333-337.

[17] X. Wen, Y. Yamashita, S. Kajihara, L.-T. Wang, K. K. Saluja, and K. Kinoshita, “On low-capture-power test generation for scan testing”, *Proc. VLSI Test Symp.*, 2005, pp.265-270.

[18] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger, “A Case Study of IR-Drop in Structured At-Speed Testing”, *Proc. Int. Test Conf.*, 2003, pp.1098-1104.

[19] S. Sde-Paz and E. Salomon, “Frequency and Power Correlation between At-Speed Scan and Functional Tests”, *Proc. Int. Test Conf.*, 2008, pp.1-9.

[20] J. Rearick, “Too much delay fault coverage is a bad thing”, *Proc. Int. Test Conf.*, 2001, pp. 624–633.

[21] I. Pomeranz and S. M. Reddy, “Generation of functional broadside tests for transition faults”, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, Oct. 2006, pp. 2207-2218.

[22] J. L. Carter, V. S. Iyengar, and B. K. Rosen, “Efficient test coverage determination for delay faults”, *Proc. Int. Test Conf.*, 1987, pp.418-427.

[23] I. Pomeranz, S. M. Reddy, and J. H. Patel, “On double transition faults as a delay fault model”, *Proc. Great Lakes Symp. VLSI*, 1996, pp.282-287.

[24] K. Heragu, J. H. Patel, and V. D. Agrawal, “Segment delay faults: A new fault model”, *Proc. 14th VLSI Test Symp.*, 1996, pp.32-39.

[25] M. Sharma and J. H. Patel, “Testing of critical paths for delay faults”, *Proc. Int. Test Conf.*, 2001, pp.634-641.

[26] P. Goel and B. C. Rosales, “Test Generation and Dynamic Compaction of Tests”, *Digest of Papers 1979 Test Conf.*, 1979, pp.189-192.

[27] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, “SMART and FAST: Test Generation for VLSI Scan-Design Circuits”, *IEEE Design & Test of Computers*, Vol. 3, No. 4, 1986, pp.43-54.

[28] I. Pomeranz, L. N. Reddy, and S. M. Reddy, “COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits”, *Proc. Int. Test Conf.*, 1991, pp.194-203.

[29] J. Rajski and H. Cox, “A method to calculate necessary assignments in algorithmic test pattern generation”, *Proc. Int. Test Conf.*, 1990, pp.25-34.

[30] M. H. Schultz, K. Fuchs, and F. Fink, “Advanced automatic test pattern generation techniques for path delay faults”, *Proc. Int. Symp. Fault-Tolerant Comput.*, 1989, pp.44-51.

[31] W.-N. Li, S. M. Reddy, and S. K. Sahni, “On path selection in combinational logic circuits”, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 1, Jan. 1989, pp.56-63.

[32] W. Qiu, J. Wang, D.M.H. Walker, D. Reddy, et al, “K longest paths per gate (KLPG) test generation for scan-based sequential circuits”, *Proc. Int. Test Conf.*, 2004, pp.223-231.

[33] A. Murakami, S. Kajihara, T. Sasao, I. Pomeranz, and S. M. Reddy, “Selection of potentially testable path delay faults for test generation”, *Proc. Int. Test Conf.*, 2000, pp.376-384.

[34] Y. Shao, S. M. Reddy, I. Pomeranz, and S. Kajihara, “On selecting testable paths in scan designs”, *J. Electron. Testing-Theory Appl.*, Aug. 2003, pp.447-456.

[35] S. Padmanaban and S. Tragoudas, “A critical path selection method for delay testing”, *Proc. Int. Test Conf.*, 2004, pp.232-241.

[36] S. Padmanaban and S. Tragoudas, “Efficient Identification of (Critical) Testable Path Delay Faults Using Decision Diagrams”, *IEEE Trans. Comput. Aided Des.of IC and Syst.*, Vol. 24, 2005, pp.77-87.

[37] B. Seshadri, I. Pomeranz, S.M. Reddy, and S. Kundu, “On Path Selection for Delay Fault Testing considering Operating Conditions”, *Proc. Europ. Test Workshop*, 2003, pp.141-146.

[38] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng, “Path selection and pattern generation for dynamic timing analysis considering power supply noise effects”, *Proc. Int. Conf. Comput. Aided Des.*, 2000, pp.493-496.

[39] J.-J. Liou, K.-T. Cheng, and D.A.Mukherjee, “Path Selection for Delay Testing of Deep Sub-Micron devices using Statistical Performance Sensitivity Analysis”, *Proc. VLSI Test Symp.*, 2000, pp.97-104.

[40] J.-J. Liou, A. Krstic, L-C. Wang, and K.-T. Cheng, “False-Path-Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation”, *Design Autom. Conf*., 2002, pp.566-569.

[41] L.-C. Wang, J.-J. Liou, and K.-T. Cheng, “Critical Path Selection for Delay Fault Testing Based Upon a Statistical Timing Model”, *IEEE Trans. Comput. Aided Des. of IC and Syst.*, VOL. 23, Nov. 2004, pp.1550-1565.

[42] V. Iyengar, J. Xiong, S. Venkatesan, V. Zolotov, et al, “Variation-aware performance verification using at-speed structural test and statistical timing”, *Proc. Int .Conf. Comput. Aided Des.*, 2007, pp.405-412.

[43] Z. He, T. Lv, H. Li, and X. Li, “Fast path selection for testing of small delay defects considering path correlations”, *Proc. VLSI Test Symp.*, 2010, pp.3-8.

[44] S. Tsai and C.-Y. Huang, “A false-path aware Formal Static Timing Analyzer considering simultaneous input transitions”, *Design Autom. Conf*., 2009, pp.25-30.

[45] I-D. Huang and S. K. Gupta, “Selection of Paths for Delay Testing”, *Asian Test Symp.*, 2005, pp.208-215.

[46] Y. Kukimoto and R. K. Brayton, “Timing-Safe False Path Removal for Combinational Modules”, *Proc. Int. Conf. Comput. Aided Des.*, 1999, pp.544-549.

[47] E. Goldberg and A. Saldanha, “Timing Analysis with Implicitly Specified False Paths”, *Proc. Int. Conf. VLSI Design*, 2000, pp.518-522.

[48] S. Zhou, B. Yao, H. Chen, Y. Zhu, M. Hutton, T. Collins, and S. Srinivasan, “Improving the Efficiency of Static Timing Analysis with False Paths”, *Proc. Int. Conf. Comput. Aided Des.*, 2005, pp.527-531.

[49] M.Muraoka, H.Iida, H.Kikuchihara, M. Murakami, and K. Hirakawa, “ACTAS: An Accurate Timing Analysis System for VLSI”, *Des. Auto. Conf*. 1985, pp.152-158.

[50] M. Beck, O. Barondeau, O. Barondeau, F. Poehl, X. Lin, and R. Press, “Logic Design for On-Chip Test Clock Generation-Implementation Details and Impact on Delay Test Quality”, *Proc. Conf. on Des., Autom. and Test in Europe*, 2005, pp.56-61.

[51] H. Furukawa, X. Wen, L.-T. Wang, B. Sheu, Z. Jiang, and S. Wu, “A Novel and Practical Control Scheme for Inter-Clock At-Speed Testing”, *Proc. IEEE Int. Test Conference*, 2006, pp.1-10.

[52] X. Fan, Y. Hu, and L.-T. Wang, “An On-Chip Test Clock Control Scheme for Multi-Clock At-Speed Testing”, *Proc. IEEE Asian Test Symp.*, 2007, pp.341-348.

[53] X. Lin and M. Kassab, “Test Generation for Designs with On-Chip Clock Generators”, *Proc. IEEE Asian Test Symp.*, 2009, pp.411-417.

[54] B. Keller, K. Chakravadhanula, B. Foutz, and et.al., “Low cost at-speed testing using On-Product Clock Generation compatible with test compression”, *Proc. IEEE Int. Test Conference*, 2010, pp.1-10.

[55] A. Jas, J. Ghosh-Dastidar, Mom-Eng Ng, and N. A. Touba, “An efficient test vector compression scheme using selective Huffman coding”, *IEEE Trans. Comput. Aided Design of IC and Syst.*, 2003, pp.797-806.

[56] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand, “Data compression for multiple scan chains using dictionaries with corrections”, *Proc. of Int. Test Conf.*, 2004, pp.926-935.

[57] N. A. Touba, “Survey of Test Vector Compression Techniques”, *IEEE Design & Testing of Computers*, 2006, pp.294-303.

[58] H. Fang, C. Tong, B. Yao, X. Song, and X. Cheng, “CacheCompress: A Novel Approach for Test Data Compression with Cache for IP Embedded Cores”, *Proc. Int. Conf. on Comp. Aided Des.*, 2007, pp.509-512.

[59] M. Abramovici, M. Breuer, and A. Friedman, *Digital System Testing and Testable Design*, IEEE Press, Piscataway, NJ, 1990.

[60] I. Pomeranz and S.M. Reddy, “On n-detection test sets and variable n-detection test sets for transition faults”, *Proc. VLSI Test Symp.*, 1999, pp.173-180.

[61] C.A. Chen and S.K. Gupta, “BIST test pattern generators for two-pattern testing-theory and design algorithms”, *IEEE Trans. on Comput.*, 1996, pp. 257-269.

[62] P. Girard, C. Landrault, V. Moreda, and S. Pravossoudovitch, “An optimized BIST test pattern generator for delay testing”, *Proc. VLSI Test Symp.*, 1997, pp. 94-100.

[63] C.A. Chen and S.K. Gupta, “Efficient BIST TPG Design and Test Set Compaction for Delay Testing via Input Reduction”, *Proc. Intl. Conf. on Comp. Design*, 1998, pp. 32-39.

[64] N. Mukherjee, T.J. Chakraborty, and S. Bhawmik, “A BIST scheme for the detection of path-delay faults”, *Proc. Intl. Test Conf.*, 1998, pp. 422-432.

[65] W. Li, C. Yu, S.M. Reddy, and I. Pomeranz, “A scan BIST generation method using a markov source and partial bit-fixing”, *Proc. Design Auto. Conf.*, 2003, pp. 554-559.

[66] S. Pateras, “Achieving At-Speed Structural Test”, *IEEE Design & Test of Comput.*, Vol. 20, Issue 5, 2003, pp. 26-33.

[67] H. Lee, I. Pomeranz, and S. M. Reddy, “Scan BIST Targeting Transition Faults Using a Markov Source”, *Proc. Intl. Symp. on Quality Electronic Design*, 2004, pp. 497-502.

[68] V. Gherman, H.-J. Wunderlich, J. Schloeffel, and M. Garbers, "Deterministic Logic BIST for Transition Fault Testing", *Proc. Euro. Test Symp.*, 2006, pp. 123-130.

[69] P. Girard, "Survey of Low-Power Testing of VLSI Circuits", *IEEE Design & Test of Computers*, May/June 2002, pp. 80-90.

[70] P. Rosinger, B. M. Al-Hashimi, and N. Nicolici, "Dual Multiple-Polynomial LFSR for Low-Power Mixed-Mode BIST", *IEEE Proc.-Computers and Digital Tech.*, Vol. 150, Issue 4, 2003, pp. 209-217.

[71] J. Lee and N. A. Touba, "Low Power BIST based on Scan Partitioning", *Proc. Symp. on Defect and Fault Tolerance in VLSI Syst.*, 2005, pp. 33-41.

[72] S. Wang, "A BIST TPG for Low Power Dissipation and High Fault Coverage", *IEEE Trans. on VLSI Syst.*, July 2007, pp. 777-789.

[73] I. Pomeranz, "Built-In Generation of Functional Broadside Tests Using a Fixed Hardware Structure", *IEEE Trans. on VLSI Systems*, Jan. 2013, pp. 124-132.

[74] V.M.Vedula and J.A. Abraham, "FACTOR: a hierarchical methodology for functional test generation and testability analysis", *Proc. Design, Autom. and Test in Europe Conf.*, 2002, pp. 730-734.

[75] S.K.S. Hari, V.V.R. Konda, V. Kamakoti, V.M. Vedula, and K.S. Maneperambil, "Automatic constraint based test generation for behavioral HDL models", *IEEE Trans. VLSI Syst.*, 2008, pp. 408-421.

[76] I. Pomeranz, "Generation of Functional Broadside Tests for Logic Blocks With Constrained Primary Input Sequences", *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 3, Mar. 2013, pp. 442-452.

[77] I. Pomeranz, "Functional broadside tests for embedded logic blocks", *IET Comput. Digit. Tech.*, 2012, pp. 223-231.

[78] L. Whetsel, "Adapting scan architectures for low power operation", *Proc. Int. Test Conf.*, 2000, pp. 863-872.

[79] P. M. Rosinger, B. M. Al-Hashimi, and N. Nicolici, "Scan Architecture with Mutually Exclusive Scan Segment Activation for Shift and Capture Power Reduction", *IEEE Trans. on Comput.-Aided Design*, July 2004, pp. 1142-1153.

[80] Z. Zhang, S.M. Reddy, I. Pomeranz, J. Rajski, and B.M. Al-Hashimi, "Enhancing delay fault coverage through low-power segmented scan", *IEEE European Test Symp.*, 2006, pp. 21-28.

[81] S. Venkataraman, J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers", *Proc. of Int. Conf. on Comput.-Aided Design*, 1993, pp.572-577.

[82] N. Touba and E. McCluskey, "Altering a Pseudo-random Bit Sequence for Scan-based BIST", *Proc. of Int. Test Conf.*, 1996, pp.167-175.

[83] H. J. Wunderlich and G. Kiefer, "Bit-flipping BIST", *Proc. of Int. Conf. on Comput.-Aided Design*, 1996, pp.337-345.

[84] H.J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns", *IEEE Trans. on Comput.-Aided Design*, 1990, pp.584-593.

[85] F. Muradali, V.K. Agarwal, and B. Nadeu-Dostie, "A New Procedure for Weighted Random Built-in Self Test", *Proc. of Int. Test Conf.*, 1990, pp.660-669.

[86] B. Reeb and H.J. Wunderlich, "Deterministic Pattern Generation for Weighted Random Pattern Testing", *Proc. of Europ. Design and Test Conf.*, 1996, pp.30-36.

[87] H.S. Kim, J. Lee, and S. Kang, "A New Multiple Weight Set Calculation Algorithm", *Proc. of Int. Test Conf.*, 2001, pp.878-884.

[88] I. Pomeranz and S. M. Reddy, "Primary Input Vectors to Avoid in Random Test Sequences for Synchronous Sequential Circuits", *IEEE Trans. on Comput.-Aided Design*, Jan. 2008, pp.193-197.

[89] I. Pomeranz and S. M. Reddy, "Forward-Looking Fault Simulation for Improved Static Compaction", *IEEE Trans. on Comput.-Aided Design*, Oct. 2001, pp.1262-1265.

[90] I. Pomeranz, “Signal-Transition Patterns of Functional Broadside Tests”, *IEEE Trans. on Computers*, 2012, pp.1-8.

APPENDIX

A. IMPLEMENTATION OF THE DEVELOPED METHODS

This section briefly describes how the major steps of the developed methods were implemented and the experiments were conducted. All the benchmark circuits used for the experiments are standard benchmark circuits. The benchmark circuits in different formats, such as bench format and RTL VHDL/Verilog format, are available online.

The deterministic broadside test generation method for transition path delay faults was implemented based on an existing self-developed software package for test generation. The five sub-procedures of the developed method were implemented in C++ on top of the test generator and fault simulator for transition faults included in the package. The test generator and fault simulator for transition faults only accept circuits in a special format called the MIX format. In order to conduct the experiment, a format convertor, which was also included in the package, was used to translate the benchmark circuits from bench format into MIX format first. Then the developed method can be applied to the circuits.

Table A.1 List of used commercial tools

Name	Vender	Use
Design Compiler	Synopsys	Logic synthesis
PrimeTime	Synopsys	Static timing analysis
DFTAdvisor	Mentor Graphics	Scan insertion
Fastscan	Mentor Graphics	Logic/fault simulation
Modelsim	Mentor Graphics	RTL logic simulation

The implementation of the path selection method and the built-in functional broadside test generation method involves some commercial tools. Table A.1 lists the commercial tools used in this dissertation, the vendor names, and the uses of the tools.

In the path selection method, PrimeTime was used to perform static timing analysis, and the process for finding input necessary assignments was implemented in C++ based on the self-developed software package. A Perl script was used to translate a circuit from Verilog format, which can be accepted by PrimeTime, into MIX format. The mapping information between the netlists of different formats was also provided by the script so that given a path in one format, it is easy to find the same path in the other format. A CShell script was used to stitch the static timing analysis process and the process for finding input necessary assignments so that the developed path selection method can be performed correctly.

To conduct the experiment, Design Compiler was first used to logic synthesize a benchmark circuit in Verilog format from RTL into gate level, using a simplified technology library. Then the netlist was translated into MIX format. After performing static timing analysis using PrimeTime, a ranked list of critical paths can be reported in a text file with the most critical path on the top. A Perl script was used to extract the critical paths from the text file and translate them from Verilog format into MIX format. Such critical paths were then fed into the process for finding input necessary assignments if they were never processed. For each such critical path, the process computed the input necessary assignments of its corresponding path delay fault and reported them in a text file if the fault was potentially detectable. These input necessary assignments were then translated into Verilog format and fed into PrimeTime so that the path delay can be recalculated.

In the built-in test generation method, the behavior of the TPG logic was simulated by a simulator written in C++. The fault list generation, logic simulation, and fault simulation were accomplished via Fastscan. For a primary input sequence generated by the TPG simulator, its corresponding state sequence was obtained by simulating the primary input sequence cycle by cycle using Fastscan controlled by a Tcl script for state sequence calculation. Both the primary input sequences and the corresponding state sequences were given in text files. A Perl script was used to extract functional broadside tests every two consecutive clock cycles from the primary input sequences and the state sequences. The obtained functional broadside tests were then simulated by Fastscan on

the fault list. The switching activity of each test can also be reported into a text file by Fastscan. A Perl script was used to check the switching activity of every test and extract the primary input segment under which the switching activity in every clock cycle does not exceed SWA_{func}. A CShell script was used to stitch the TPG simulator, Fastscan calls and Perl scripts so that the developed built-in test generation method can be performed correctly.

To conduct the experiment, Design Compiler was first used to logic synthesize a benchmark circuit in Verilog format from RTL into gate level, using a simplified technology library. Then DFTAdvisor was used to insert scan structure into the circuit. The primary input cube C can be calculated through logic simulation using Fastscan controlled by a Tcl script for primary input cube calculation. To obtain the value of SWA_{func}, 30 sequences of length 30000 were computed by the TPG simulator. The driving block was simulated under these sequences using Fastscan, and 30 corresponding sequences can be obtained at the primary outputs of the driving block. The target circuit was then simulated under the primary output sequences of the driving block, and the peak switching activity can be reported by Fastscan. After that, the CShell script was invoked to perform the developed built-in test generation method. To evaluate the area overhead, the built-in test generation logic was implemented in Verilog. The correctness of the Verilog code was verified through simulation by using Modelsim. The Verilog code was then logic synthesized into gate level by using Design Compiler which can also report the area of the logic.

VITA

VITA

Bo Yao received his B.S. degree in Electronics Engineering from Tsinghua University, Beijing, China in 2005, and his M.S. degree in Electrical Engineering and Computer Science from Peking University, Beijing, China in 2008. Since 2008, he has been at the school of Electrical and Computer Engineering, Purdue University, pursuing a Ph.D. degree.

From March 2012 to February 2013, he was a visiting researcher at Intel Corporation, Hillsboro, OR, USA. His research interests include fault models, test generation, design-for-test, and design verification.