ATPG and DFT Algorithms for Delay Fault Testing

Xiao Liu

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirement for the degree of

Doctor of Philosophy
in
Computer Engineering

Dr. Michael S. Hsiao, Chairman

Dr. R. Michael Buehrer

Dr. Dong S. Ha

Dr. Thomas L Martin

Dr. Mark M Shimozono

July 19, 2004
Blacksburg, Virginia

Keywords: testing, delay testing, transition fault, ATPG, DFT,

ATPG and DFT Algorithms for Delay Fault Testing

Xiao Liu

Abstract

With ever shrinking geometries, growing metal density and increasing clock rate on chips, delay testing is becoming a necessity in industry to maintain test quality for speed-related failures. The purpose of delay testing is to verify that the circuit operates correctly at the rated speed. However, functional tests for delay defects are usually unacceptable for large scale designs due to the prohibitive cost of functional test patterns and the difficulty in achieving very high fault coverage. Scan-based delay testing, which could ensure a high delay fault coverage at reasonable development cost, provides a good alternative to the at-speed functional test.

This dissertation addresses several key challenges in scan-based delay testing and develops efficient Automatic Test Pattern Generation (ATPG) and Design-for-testability (DFT) algorithms for delay testing. In the dissertation, two algorithms are first proposed for computing and applying transition test patterns using stuck-at test vectors, thus avoiding the need for a transition fault test generator. The experimental results show that we can improve both test data volume and test appli-

cation time by 46.5% over a commercial transition ATPG tool. Secondly, we propose a hybrid scan-based delay testing technique for compact and high fault coverage test set, which combines the advantages of both the skewed-load and broadside test application methods. On an average, about 4.5% improvement in fault coverage is obtained by the hybrid approach over the broad-side approach, with very little hardware overhead. Thirdly, we propose and develop a constrained ATPG algorithm for scan-based delay testing, which addresses the overtesting problem due to the possible detection of functionally untestable faults in scan-based testing. The experimental results show that our method efficiently generates a test set for functionally testable transition faults and reduces the yield loss due to overtesting of functionally untestable transition faults. Finally, a new approach on identifying functionally untestable transition faults in non-scan sequential circuits is presented. We formulate a new dominance relationship for transition faults and use it to help identify more untestable transition faults on top of a fault-independent method based on static implications. The experimental results for ISCAS89 sequential benchmark circuits show that our approach can identify many more functionally untestable transition faults than previously reported.

To my parents, my wife and my daughter, who have given me endless support and happieness...

I Love You!

Acknowledgment

I would like to express my deepest appreciation to my advisor, Dr. Michael S. Hsiao, for his constant guidance, encouragement and support in helping me to complete this work. I am highly indebted for his insightful advice, which help me meander through my research and life in States. To me, Dr. Hsiao is much more than a research advisor, but also one of the best friends in my life. I will cherish the experience learning and working with him forever.

In addition, I would like to thank Dr. Richard M. Buehrer, Dr. Dong S. Ha, Dr. Thomas L Martin and Dr. Mark M Shimozono, for serving on my committee.

I am extremely thankful to Dr. Sreejit Chakravarty, Dr. Seongmoon Wang, Dr. Paul J. Thadikaran and Dr. Srimat T. Chakradhar for their help and guidance on my research.

I would like to thank my friends who have made my stay at graduate school enjoyable and people in the Proactive research group who have made every aspect of research exciting and interesting to me.

Finally, I would like to thank my beloved family for their unconditional love and support on me. Without their invaluable motivation and help, I would not be who I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main objective of traditional test development has been the attainment of high stuck-at fault coverage. However, the presence of some random defects does not affect a circuit's operation at low speed while it may cause circuit malfunction at rated speed. This kind of defect is called the delay defect. With ever shrinking geometries, growing metal density and increasing clock rate of chips, delay testing is gaining more and more industry attention to maintain test quality for speed-related failures. The purpose of a delay test is to verify that the circuit operates correctly at a desired clock speed. Although application of stuck-at fault tests can detect some delay defects, it is no longer sufficient to test the circuit for the stuck-at faults alone. Therefore, delay testing is becoming a necessity for today's IC manufacturing test.

In the past, testing a circuit's performance was typically accomplished with functional test patterns. However, developing functional test patterns that attain satisfactory fault coverage is unacceptable for large scale designs due to the prohibitive development cost. Even if functional test patterns that can achieve high fault coverage are available, applying these test patterns at-speed for high speed chips requires very stringent timing accuracy, which must be provided by very ex-

pensive automatic test equipment (ATEs). The scan-based delay testing where test patterns are generated by an automatic test pattern generator (ATPG) on designs that involve scan chains is increasingly used as a cost efficient alternative to the at-speed functional pattern approach to test large scale chips for performance-related failures [BRS+02, SBG+02]. Design-for-testability (DFT)-focused ATEs [COM00, ROB00], which are designed and developed to lower ATE cost by considering widely used DFT features of circuits under test (CUTs) such as full and partial scan are emerging as a strong trend in the test industry.

Several delay fault models have been developed, such as transition delay fault [WLRI87], gate delay fault [CIR87, PR97], path delay fault [SMI85], and segment delay fault models [HPA96]. A transition fault at node X assumes a large delay at X such that the transition at X will not reach the latch or primary output within the clock period. The path delay fault model assumes a small delay at each gate. It models cumulative effect of gate delays along a specific path, from a primary input to a primary output. If the cumulative delay exceeds the slack for the path, then the chip fails. Segment delay fault targets path segments instead of complete paths. Among these fault models, the transition delay fault model [WLRI87] is most widely used in industry for its simplicity. ATPGs and fault simulators that are developed for stuck-at faults can be reused for transition delay faults with minor modifications. Unlike the path delay fault model [SMI85] where the number of target faults is often exponential, the number of transition delay faults is linear with the number of circuit lines. This eliminates the need for critical path analysis and identification procedures, which are necessary for the path delay fault model. The gate delay model [CIR87, PR97] is similar to the transition delay fault model in that the delay fault is lumped at one gate in the CUT. However, unlike the transition delay model which does not take into account fault sizes, the gate delay model

takes into account fault sizes. The segment delay fault model [HPA96] is a trade-off between the path delay fault and transition delay fault models.

Detection of a delay fault normally requires the application of a pair of test vectors; the first vector, called *initialization vector*, initializes the targeted faulty circuit line to a desired value and the second vector, called *launch vector*, launches a transition at the circuit line and propagates the fault effect to primary output(s) and/or scan flip-flop(s) [WLRI87, SB87].

1.1 Motivation

In general, (non-scan) functional testing can be impractical for larger circuits since large test sets may be required to achieve a desirable fault coverage. As a result, at-speed AC scan testing has been widely used in the industry to detect delay-induced defects. Compared to functional testing, scan-based testing for delay faults can decrease the overall ATPG complexity and cost, since both controllability and observability on the flip-flops are enhanced.

Although delay fault testing has been researched for years, most research on delay fault testing has focused on the combinational part of the circuits. However, due to limited controllability of state inputs when standard scan is employed, applying these techniques to standard scan designs is not straightforward. Traditionally, three different approaches, **Enhanced-Scan** [DS91], **Skewed-Load** [SAV92a] and **Broadside** [SP94b], have been used to apply two-vector tests to standard scan designs. They differ in their way of storing and applying the second vector of each vector pair.

Several techniques for reducing test data volume and test application time for single cycle scan-tests have been presented in the literature [KBB+01, HP99,

KOE91, CC01, LCH98, DT00]. These methods assume that only 5-10% of the bits are fully specified. Unspecified bits are filled to detect the easy to detect faults. Different codes to compress the information in the tester and decompressing them on chip [KBB+01, CC01, DT00] or using partitioning and applying similar patterns to the different partitions [HP99, LCH98] have been proposed. The techniques proposed here compliment the work on compressing individual vectors. All of these techniques can also be used to compress the individual vectors comprising the transition test sets. They, however, do not address the data explosion in going from stuck-at tests to transition tests or the test application time issue with transition tests.

Nevertheless, the drawback of scan-based delay tests lies in two areas: hardware overhead and potential yield loss due to overtesting. In [REA01], the author reported that scan-based testing may fail a chip due to the delay faults that do not affect the normal operation, and thus it is unnecessary to target those functionally unsensitizable faults. In other words, we want to avoid failing a chip due to a signal transition/propagation that was not intended to occur in the functional mode. In [LKC00a, LKC00c], the authors also addressed the impact of delay defects on the functionally untestable paths on the circuit performance. Moreover, a scan test pattern, though derived from targeting functionally testable transition faults, can incidentally detect some functionally untestable transition faults if the starting state is an unreachable state.

Furthermore, several papers [RM01, REA01, MaLB00] have discussed the relationship between functional testing and scan-based testing. However, from our knowledge, currently there is no quantitative analysis on functional untestable transition faults and scanning testing. In this dissertation, we describe a novel constrained ATPG algorithm for transition faults. Two main contributions of our work

are: (1) the constrained ATPG only targets the functionally testable transition faults and *minimizes* detection of any identified functionally untestable transition faults; (2) the constrained ATPG can identify more functionally untestable transition faults than the conventional transition ATPG tools. The first contribution (the constrained ATPG) enables us to derive transition vectors that avoid illegal starting states, while the second contribution helps us to maximize the state space that we need to avoid. Because we want to avoid launching and propagating transitions in the circuit that are not possible in the functional mode, a direct benefit of our method is the reduction of yield loss due to overtesting of these functionally untestable transitions. Our experimental results showed that significantly more functionally untestable transition faults can be avoided in the final test set.

1.2 Our Work

In the effort to investigate novel and efficient delay fault testing algorithms and techniques, several ideas on delay fault testing has been explored. This dissertation addresses the following problems on developing novel and efficient ATPG and Design-for-testability (DFT) algorithms for all three approaches for scan-based delay testing.

1. Explosion in test data volume and test application time
2. Lower fault delay fault coverage
3. High complexity in delay fault ATPG
4. The overtesting problem in scan-based delay testing
5. Functional vs. scan-based delay testing

We first present two efficient transition fault ATPG algorithms [LHCT02a, LHCT03], in which we compute good quality transition test sets using stuck-at test vectors. Experimental results obtained using the new algorithms show that there is a 20% reduction in test set size compared to a state-of-the-art native transition test ATPG tool, without losing fault coverage. Other benefits of our approach, viz. productivity improvement, constraint handling and design data compression are also highlighted.

Our second contribution [LHCT02b,LHCT04] is on the techniques to reduce data volume and application time for scan-based transition test. We propose a novel notion of transition test chains to substitute the conventional transition pattern and combine this idea with the ATE repeat capability to reduce test data volume. Then a new DFT technique for scan testing is presented to address the test application issue. Our experimental results show that our technique can improve both test data volume and test application by 46.5% over a commercial ATPG tool.

Thirdly, a novel scan-based delay test approach [WLC03, WLC04], referred to as the hybrid delay scan, is proposed. The proposed scan-based delay testing method combines advantages of the skewed-load and broad-side approaches. The hybrid approach can achieve higher delay fault coverage than the broad-side approach. Unlike the skewed-load approach whose design requirement is often too costly to meet due to the fast scan enable signal that must switch in a full system clock cycle, the hybrid delay scan does not require a strong buffer or buffer tree to drive the fast scan enable signal. Hardware overhead added to standard scan designs to implement the hybrid approach is negligible. Since the fast scan enable signal is internally generated, no external pin is required. Transition delay fault coverage achieved by the hybrid approach transition was equal to or higher than that achieved by the broad-side for all ISCAS 89 benchmark circuits [BBK89]. On an average,

about 4.5% improvement in fault coverage was obtained by the hybrid approach over the broad-side approach.

Next, we propose a new concept of testing only functionally testable transition faults in Broadside Transition testing via a novel constrained ATPG [LH03a, LH03b]. Illegal (unreachable) states that enable detection of functionally untestable faults are first identified, and this set of undesirable illegal states is efficiently represented as a Boolean formula. Our constrained ATPG then uses this constraint formula to generate Broadside vectors that avoid those undesirable states. In doing so, our method efficiently generates a test set for functionally testable transition faults and minimizes detection of functionally untestable transition faults. Because we want to avoid launching and propagating transitions in the circuit that are not possible in the functional mode, a direct benefit of our method is the reduction of yield loss due to overtesting of these functionally untestable transitions.

Finally, we proposed a new approach on identifying functionally untestable transition faults in non-scan sequential circuits. A new dominance relationship for transition faults is formulated and used to identify more sequentially untestable transition faults. The proposed method consists of two phases: first, a large number of functionally untestable transition faults is identified by a fault-independent sequential logic implications implicitly crossing multiple time-frames, and the identified untestable faults are classified into three conflict categories. Second, additional functionally untestable transition faults are identified by dominance relationships from the previous identified untestable transition faults. The experimental results for ISCAS89 sequential benchmark circuits showed that our approach can quickly identify many more functionally untestable transition faults than previously reported.

In short, the topics we have investigated include:

1. Transition Fault ATPG Based on Stuck-at Test Vectors
2. Efficient Transition Testing using Test Chains and Exchange Scan
3. Hybrid Scan-based Delay Testing
4. Constrained ATPG for Broadside Transition Testing
5. Functional Untestable Transition Faults Identification

The rest of the dissertation is organized as follows. First, we give a brief review of the preliminaries on delay fault testing in Chapter 2. Then, a novel transition fault ATPG based on stuck-at test vectors is discussed in Chapter 3. Chapter 4 presents two techniques to further reduce the test data volume and test application time for scan-based transition test. In Chapter 5, a novel scan-based delay test approach, referred to as the hybrid delay scan, is proposed. Then,we proposed a new concept of testing only functionally testable transition faults in Broadside Transition testing via a novel constrained ATPG in Chapter 6. Chapter 7 presents a novel approach to identify functional untestable transition faults using implication and transition dominance relationship. Finally, Chapter 8 concludes the dissertation.

Chapter 2

Preliminaries for Delay Testing

Several delay fault models have been developed for delay defects: transition delay fault [WLRI87], gate delay fault [CIR87, PR97], path delay fault [SMI85], and segment delay fault models [HPA96]. A transition fault at node X assumes a large delay at X such that the transition at X will not reach the latch or primary output within the clock period. The path delay fault model assumes a small delay at each gate. It models cumulative effect of gate delays along a specific path, from a primary input to a primary output. If the cumulative delay exceeds the slack for the path, then the chip fails. Segment delay fault targets path segments instead of complete paths.

Among these fault models, the transition delay fault model [WLRI87] is most widely used in industry for its simplicity. ATPGs and fault simulators that are developed for stuck-at faults can be reused for transition delay faults with minor modifications. Unlike the path delay fault model [SMI85] where the number of target faults is often exponential, the number of transition delay faults is linear to the number of circuit lines. This eliminates the need for critical path analysis and identification procedures, which are necessary for the path delay fault model.

Gate delay model [CIR87, PR97] is similar to transition delay fault model in that the delay fault is lumped at one gate in the CUT. However, unlike transition delay model which does not take into account fault sizes, gate delay model takes into account fault sizes. Segment delay fault model [HPA96] is a trade-off between path delay fault and transition delay fault models.

2.1 Functional Delay Testing

In the past, testing circuit's performance was typically accomplished with functional test patterns(i.e. testing a microprocessor with instruction sequences [LKC00b, LKC00d, CS01, KLC+02]), in which the input signals to the CUT are determined by its functionality. This result in a much smaller set of vector pairs applicable for delay testing. Also, after the application of certain test sets, some registers/flip-flops may not be enabled in the immediate next cycle, and thus delay fault effects propagated to them can not be latched and will be lost. Therefore, developing functional test patterns that attain satisfactory fault coverage is unacceptable for large scale designs due to the prohibitive development cost. Even if functional test patterns that can achieve high fault coverage are available, applying these test patterns at-speed for high speed chips requires very stringent timing accuracy, which can be provided by very expensive automatic test equipments (ATEs). The scan-based delay testing where test patterns are generated by an automatic test pattern generator (ATPG) on designs that involve scan chains is increasingly used as a cost efficient alternative to the at-speed functional pattern approach to test large scale chips for performance-related failures [BRS+02, SBG+02]. Design-for-testability (DFT)-focused ATEs [COM00, ROB00], which are designed and developed to lower ATE cost by considering widely used DFT features of circuits under test (CUTs) such as

full and partial scan are emerging as a strong trend in test industry.

2.2 Scan-based Delay Testing

Traditionally, three different approaches have been used to apply two-vector tests to standard scan designs. They differ in the way of storing and applying the second vector of each vector pair. **Enhanced-Scan** [DS91]. **Skewed-Load** [SAV92a] and **Broadside** [SP94b].

2.2.1 Enhanced-scan

In the first approach, enhanced-scan [DS91], two vectors (V1, V2) are stored in the tester scan memory. The first scan shift loads V1 into the scan chain. It is then applied to the circuit under test to initialize it. Next, V2 is scanned in, followed by an apply and subsequently a capture of the response. During shifting in of V2 it is assumed that the initialization of V1 is not destroyed. Therefore enhanced-scan transition testing assumes a hold-scan design [DS91].

Enhanced scan transition test has two primary advantages: coverage and test data volume. Since enhanced scan testing [DS91] allows the application of any arbitrary vector pair to the combinational part of a sequential circuit. Hence, complete fault coverage can be attained.

Tester memory requirement is also important, and considerable attention is being paid to reduce the tester memory requirement for s@ tests. The problem is far worse for transition tests as the following data shows. In [HBP01] it was reported that for skewed load transition tests for an ASIC, the s@ vector memory requirement was 8.51M versus 50.42M for transition test. This implies an increase

of a factor of 5.9.

The downside of using enhanced-scan transition test is that special scan-design, viz. hold-scan that can hold two bits, is required. This may leads to higher area overhead, which may prevent it from using widely in ASIC area. However, in microprocessors and other high performance circuits that require custom design, such cells are used for other reasons. In custom designs, the circuit often is not fully decoded, hold scan cells are used to prevent contention in the data being shifted, as well as preventing excessive power dissipation in the circuit during the scan shift phase. Furthermore, if hold-scan cells are used, the failing parts in which only the scan logic failed can often be retrieved; thus enhancing, to some extent, the diagnostic capability associated with scan DFT. Therefore, for such designs enhanced-scan transition tests is preferred. This is our motivation for investigating good ATPG techniques for enhanced-scan transition tests.

Therefore, we can see that the two vectors (V1,V2) are independent to each other. However, in the next two approaches (skewed-load and broadside), the second vector is derived from the first vector.

2.2.2 Skewed-load

In the second approach, referred to as the skewed-load [SP93] or launch-from-shift approach, The initialization vector of a test vector pair is first loaded into scan chain by n consecutive *scan shift operations*, where n is the number of scan flip-flops in the scan chain, in the same fashion as a stuck-at test vector is loaded into the scan chain. The last shift cycle when a test vector is fully loaded into the scan chain CUT, is referred as the *initialization cycle*. The second vector is obtained by shifting in the first vector (initialization vector), which is loaded into the scan chain,

by one more scan flip-flop and scanning in a new value into the scan chain input. Note that the scan enable signal stays at logic high during the launch cycle in the timing diagram shown in Figure 5.2 (a). At the next clock cycle (capture cycle), the scan enable signal switches to logic low and the scan flip-flops in the scan chain are configured in their normal mode to capture the response to the scanned in test vector. Since the capture clock is applied at full system clock speed after the launch clock, the scan enable signal, which typically drives all scan flip-flops in the CUT, should also switch within the full system clock cycle. This requires the scan enable signal to be driven by a sophisticated buffer tree or strong clock buffer. Such design requirement is often too costly to meet. Furthermore, meeting such a strict timing required for the scan enable signal may result in longer design time.

Since the second vector of each vector pair is obtained by shifting in the first vector by one more scan flip-flop, given a first vector, there are only two possible vectors for the second vector that differs only at the value for the first scan flip-flop whose scan input is connected to the scan chain input. This *shift dependency* restricts the number of combinations of test vector pairs to $2^n \times 2$ [SB91] in standard scan environment, where n is the number of scan flip-flops in the scan chain. If there is a transition delay fault that requires a 1 at state input si_{i-1} in an initialization vector and requires a 0 at state input si_i in the corresponding launch vector to be detected, then that fault is untestable by the skewed-load approach (assume that the scan chain is constructed by using only non-inverting outputs of scan flip-flops).

2.2.3 Broadside

In the third approach, referred to as the broad-side [SP93, SP94a] or launch-from-capture, Similar to the skewed-load approach, the initialization vector of a test vec-

tor pair is first loaded into scan chain by n consecutive *scan shift operations*, where n is the number of scan flip-flops in the scan chain, in the same fashion as a stuck-at test vector is loaded into the scan chain. Then, the second vector is obtained from the circuit response to the first vector.

Hence, the scan flip-flops are configured into the normal mode by lowering the enable signal before every launch cycle (see Figure 5.2 (a)). Since the launch clock following an initialization clock need not be an at-speed clock, the scan enable signal does not have to switch to logic low at full system clock speed between the initialization clock and the launch clock. Note that in the broad-side approach, launch vectors are applied when scan flip-flops are in their normal mode. In other words, the at-speed clocks, the capture clock after the launch, is applied to scan flip-flops while the scan flip-flops stays in their normal mode. Hence, the scan enable signal does not have to switch between the launch cycle and the capture cycle when clocks are applied at full system clock speed. Hence, the broad-side approach does not require at-speed transition of the scan enable signal and can be implemented with low hardware overhead.

Even though the broad-side approach is cheaper to implement than the skewed-load approach, fault coverage achieved by test pattern sets generated by the broad-side approach is typically lower than that achieved by test pattern sets generated by the skewed-load approach [SP94a]. Test pattern sets generated by the broad-side approach are typically larger than those generated by the skewed-load approach [SBG$^+$02]. In order to generate two vector tests for the broad-side approach, an ATPG with sequential property that considers two full time frames is required. On the other hand, test patterns for the skewed-load approach can be generated by a combinational ATPG with little modification. Hence, higher test generation cost (longer test generation time) should be paid for the broad-side approach.

Since in the broad-side approach, the second vector is given by the circuit response to the first vector, unless the circuit can transition to all 2^n states, where n is the number of scan flip-flops, the number of possible vector that can be applied as second vectors of test vector pairs is limited. Hence, if a state required to activate and propagate a fault is an invalid state, i.e., the state cannot be functionally justified, then the transition delay fault is untestable. Typically, in large circuits that have a large number of flip-flops, the number of reachable states is only a small fraction of 2^n states. Due to this reason, transition fault coverage for standard scan designs is often substantially lower than stuck-at fault coverage.

2.3 Summary

Among the three approaches for applying delay tests, broadside suffers from poor fault coverage [SP94b]. Since there is no dependency between the two vectors in Enhanced scan, it can give better coverage than skewed-load transition test. Skewed-load transition tests also lead to larger test data volume. Compared to stuck-at tests, the increase in the number of vectors required for enhanced scan to get complete coverage is about 4X [LHCT02a] For skewed-load transition test, it has been observed that the data volume for an ASIC has an increase of 5.9X [HBP01].

For most circuits, test sets generated by the skewed-load approach achieve higher fault coverage than those generated by the broadside approach [SAV94]. Sizes of test pattern sets generated by the skewed-load approach are also typically smaller than those generated by the broad-side approach [SBG$^+$02]. However, the skewed-load approach requires higher hardware overhead and may require longer design times (see Section 5.3).

We summary the advantages and disadvantages of the three approaches in table 2.1.

Table 2.1: Enhanced-scan vs. Skewed-load vs. Broadside

	Enhanced-scan	*Skewed-load*	*Broadside*
Fault Coverage	Highest	Higher	Lowest
Test Set Size	Smallest	Smaller	Largest
ATPG Complexity	Lowest	Higher	Highest
ATPG Time	Shortest	Shorter	Longest
Scan-cell Type	Hold-scan	Standard	standard
Hardware Overhead	Highest	Higher	Lowest
Overtesting Ratio	Highest	Higher	Lowest

Chapter 3

Transition Fault ATPG Based-on Stuck-at Test Vectors

3.1 Motivation

Enhanced scan transition test has two primary advantages: coverage and test data volume. Among the three types of transition tests, broadside tests suffer from poor fault coverage [SAV94, SP94b]. Enhanced-scan, in general, gives better coverage than skewed-load transition test. This stems from the fact that, unlike skewed-load tests, there is no dependency between the two vectors in enhanced-scan test pattern.

Tester memory requirement is also important, and considerable attention is being paid to reduce the tester memory requirement for s@ tests. The problem is far worse for transition tests as the following data shows. In [HBP01] it was reported that for skewed load transition tests for an ASIC, the s@ vector memory requirement was 8.51M versus 50.42M for transition test. This implies an increase of a factor of 5.9. This increase in tester memory requirement is also true if enhanced-scan transition tests are used. We generated s@ and enhanced-scan transition tests

for the ISCAS85 and ISCAS89 benchmark circuits using a state-of-the-art commercial ATPG tool, and the results are shown in Table 3.1. Both s@ and transition test

Table 3.1: Storage Increase.

$circuit$	$Stuck-at$ $vectors$	$Transition$ $Patterns$	$Expansion$
C1908	129	263	4.08
C2670	116	198	3.41
C3540	179	366	4.09
C5315	124	248	4
C6288	36	90	5
C7552	237	378	3.19
S5378	266	490	3.68
S9234	410	837	4.08
S13207	485	1002	4.14
S15850	464	924	3.98
S35932	75	137	3.65
S38417	1017	1927	3.79
S38584	727	1361	3.74

vectors were generated with compression option turned on. Note that each pattern for enhanced-scan transition test requires two vectors. The data shows an expansion of anywhere between 4X and 5X in tester memory requirement for transition tests when compared with that required for s@ tests.

The downside of using enhanced-scan transition test is that special scan-design, viz. hold-scan, is required. For designs using custom logic, such as high performance microprocessors, the circuit is often not fully decoded. In such a scenario, to avoid contention during the scan-shift phase, hold-scan design is used even for applying s@ tests. Therefore, for such designs enhanced-scan transition tests is preferred. This is our motivation for investigating good ATPG techniques for enhanced-scan transition tests.

The rest of the chapter is organized as follows. Section 3.2 explains the

bounding of the transition fault coverage based on s@ test set. In Section 6.4 heuristics to compute a much smaller set of transition patterns with the achievable coverage is discussed. Experimental result comparing these heuristics is discussed in Section 3.4. Section 3.5 compares results to a state-of-the-art transition fault ATPG tool. In addition to coverage, the fact that we are reusing the s@ patterns has a number of additional benefits pertaining to productivity improvement, constraint handling and design data compression. These are discussed in Section 3.6. Finally, Section 3.7 concludes the chapter.

3.2 Achievable Coverage using S@ Tests

There is a close correlation between s@ vectors and enhanced-scan transition patterns. The condition to detect a transition fault is more stringent than for s@ faults in that an extra initialization vector is required. For example, a transition pattern for *slow-to-fall* fault on line L requires an *initial vector* that *initializes* L to 1, done by a test vector that excites L s@0, and a *test vector* for detecting fault L s@1. The resulting vector pair causes a falling transition and propagates the fault effect to an observable node. Transition pattern for L *slow-to-rise* can be similarly composed.

If the s@ fault coverage of a test set T is α, then the enhanced-scan transition pattern set derived by using only vectors in T has transition fault coverage of at most α. To understand this, suppose we have an undetected s@-0 fault, then there must be at least one slow-to-rise transition fault that cannot be detected no matter how we rearrange the test vectors in the test set. Therefore, the s@ coverage of a given test set T is the upper bound for transition fault coverage, if the transition patterns are composed from vectors in T.

The conditions under which the upper-bound is not achieved are as follows. A *slow-to-rise* fault at X is not detectable even when X *s@-0* is detectable if and

only if there does not exist any s@ test in the given s@ test set, T, which sets X to 0. Similarly, a *slow-to-fall* fault at X is not detectable even when X *s@-1* is detectable if and only if there does not exist any s@ test in T which sets X to 1. In both cases an initialization vector does not exist.

Table 3.2: Results for Exhaustive Patterns.

Circuit	[HRP97]	S@	Trans	Exhaust Pairwise	
	Vec	FC(%)	FC(%)	Vec	TFC(%)
c880	128	100	95.51	16256	100
c1355	198	99.77	94.23	39006	99.77
c1908	143	99.67	93.03	20306	99.67
c3540	202	96.29	88.68	40602	96.27
c5315	157	99.56	96.91	24492	99.54
c6288	41	99.42	97.60	1640	99.19
S344	31	100	89.31	930	100
S382	41	100	90.50	1640	100
S526	82	99.93	90.99	6642	99.93
S832	179	99.20	82.12	31862	99.20
S1196	197	99.97	87.52	38612	99.97
S1423	97	99.11	95.12	9312	99.11
S5378	332	98.77	93.65	109892	98.40
S35932	78	90.50	89.97	6006	90.50
S38417	1207	99.67	97.59	1455642	99.66
S38584	893	95.34	91.63	796556	95.02

To determine the maximal transition coverage obtainable by exhaustively pairing all s@ vectors was computed. Results are shown in Table 3.2. STRATEGATE [HRP97] stuck-at test sets were used. Columns 3, 6 of Table 3.2 imply that by including all possible combination of vector pairs, the transition fault coverage reach the s@ coverage in most cases. In a few cases, such as C3540, C5315, S5378 and S38584, the transition fault coverages are slightly lower. However, the differences are negligible.

The staggering number of exhaustive pairwise vectors in Table 3.2 clearly

make it infeasible to use all such pairs as transition patterns. To alleviate this high cost, we present two heuristics to select a considerably smaller set of s@ vectors to achieve the same transition fault coverage.

3.3 Transition ATPG algorithms

3.3.1 Fault-list-based Extension

This technique is based on a greedy approach, illustrated by the following example. Let us consider a circuit with four gates (eight s@ faults) and a test set consisting of 5 vectors V1, V2, V3, V4, V5. The excited and detected s@ faults by the test set (simulated with fault-dropping) are shown in Table 3.3.

Table 3.3: Dictionary with fault-dropping.

$Vector$	$Excited\ Faults$	$Detected\ Faults$
V1	a-s-1,b-s-0,c-s-0,d-s-1	a-s-1,b-s-0
V2	b-s-1,c-s-1,d-s-1	c-s-1,d-s-1
V3	a-s-0,c-s-0	a-s-0,c-s-0
V4	b-s-1	b-s-1
V5	d-s-0	d-s-0

By applying the test sequence in the given order, only two transition faults (falling-transition fault and rising-transition fault on gate c) can be detected. However, we can obviously see that the following vector-pairs can detect additional transition faults:

1. (V3,V1) detects falling-transition on a;
2. (V1,V3) detects rising-transition on a;
3. (V1,V4) detects falling-transition on b;
4. (V4,V1) detects rising-transition on b;

5. (V5,V2) detects falling-transition on d;
6. (V1,V5) detects rising-transition on d;

Including these 6 additional test patterns (instead of all possible 20 pairs) is sufficient to detect all transition faults.

Next, we describe how these 6 test patterns are selected. First, we pick an undetected transition fault f. Without loss of generality, assume the fault f slow-to-rise. We look for the first vector that sets node f to logic 0, and then complete this vector-pair by searching for the first vector that detects the s@-fault f s-a-0. In the above example, for the fault a slow-to-fall, the first vector that sets a to logic 1 (excite a s@-0) is V3. The first vector in the dictionary that detects a s-a-1 is V1. Thus the pair (V3, V1) is constructed for the transition fault a slow-to-fall. The algorithm then picks the next undetected transition fault. The complete procedure for the fault-list-based approach is outlined below:

1. Assume that we have a s@ test set $T = \{T_1, \ldots, T_N\}$. Perform transition fault simulation using original test set $\{\langle T_1, T_2\rangle, \langle T_2, T_3\rangle, \ldots, \langle T_{N-1}, T_N\rangle\}$. Compute $undet\_TR$, the set of undetected transition faults.
2. Compose the useful subset U of the s@ faults implied by $undet\_TR$ as follows. If X *slow-to-rise* or *slow-to-fall* fault $\in undet\_TR$ then both X s@-0 and s@-1 are included in U.
3. Perform s@ fault simulation using T on the s@ faults in U. For each undetected s@ fault in U, record the *first vector* in T that excites it (*greedy initial vector*) and the *first vector* that detects it (*greedy test vector*).
4. Iterate through the transition faults in $undet\_TR$, and for each fault add a test pair to the transition test set. If the fault is X *slow-to-rise* then add the vector

pair $\langle V_i, V_j \rangle$ where: V_i is the *greedy initial vector* for X s@ 1; and V_j is the *greedy final vector* for X s@ 0. If the fault is X *slow-to-fall* then add the vector pair $\langle V_i, V_j \rangle$ where: V_i is the *greedy initial vector* for X s@ 0; and V_j is the *greedy final vector* for X s@ 1.

3.3.2 Priority-based Extension

In the previous *fault-list-based* algorithm, only one excitation vector and one detection vector is recorded per fault in the dictionary. Thus, only one vector pair can be formed for a transition fault. Stated differently, a different vector pair may detect the same target transition fault and maximize detection of remaining undetected transition faults at the same time. Table 3.4 shows the excitation and detection dictionary obtained by simulation *without* fault-dropping on the same circuit as in Table 3.3. If we consider all the combinations of test vectors for transition faults in Table 3.4,

Table 3.4: Dictionary without fault-dropping.

$Vector$	$Excited\ Faults$	$Detected\ Faults$
V1	a-s-1,b-s-0,c-s-0,d-s-1	a-s-1,b-s-0
V2	b-s-1,c-s-1,d-s-1	c-s-1,d-s-1
V3	a-s-0,c-s-0	a-s-0,c-s-0
V4	a-s-0,b-s-1	a-s-0,b-s-1
V5	a-s-0,d-s-0	a-s-0,d-s-0

we can see that:

1. (V3,V1), (V4,V1), (V5,V1) detect falling-trans. on a;
2. (V1,V3), (V1,V4), (V1,V5) detect rising-trans. on a;
3. (V1,V4) detects falling-transition on b;
4. (V2,V1), (V4,V1) detect rising-transition on b;

5. (V5,V2) detects falling-transition on d;
6. (V1,V5), (V2,V5) detect rising-transition on d;

If we select the test patterns intelligently, 4 test patterns (V1,V4), (V1,V5), (V5,V2) and (V4,V1) suffice to detect all transition faults.

The assumption behind this heuristic is that a pattern which detects a hard-to-detect fault may detect many other faults as well (some could be easy-to-detect). The heuristic, named *Priority-based extension algorithm*, processes the hard-to-detect faults first. The **priority of a transition fault** at node N, with respect to the s@ test set T, is defined to be the number of times node N s@-0 and N s@-1 faults are detected by the test set T. The transition fault with the lowest priority is one that is most difficult to detect and is therefore at the top of the list.

Because fault-simulation without fault-dropping could be expensive, we restrict simulation of only those undetected transition faults missed by the original s@ test set. This more complete dictionary also enables us to identify essential vectors that must be included in the final test set. The priority-based extension algorithm is described next.

1. Assume that we have a s@ test set $T = \{T_1, \ldots, T_N\}$. Perform Transition Fault Simulation on the original test set $\{\langle T_1, T_2\rangle, \langle T_2, T_3\rangle, \ldots, \langle T_{N-1}, T_N\rangle\}$. Record all undetected transition faults $undet\_TR$.
2. Compose the useful subset U of the s@ faults implied by $undet\_TR$ as follows. If X *slow-to-rise* or *slow-to-fall* fault $\in undet\_TR$, then both X s@-0 and s@-1 are included in U.
3. Perform s@ fault simulation, *without fault dropping*, using T on the s@ faults in U. For each transition fault in $undet\_TR$ calculate its priority index.

4. Sort the transition faults in $undet\_TR$ in increasing order of their priority. The transition fault with the lowest priority is one that is most difficult to detect and is therefore at the top of the list. It is targeted first.

5. For each target transition fault:

   (a) If its priority number is 0, then mark the fault as undetectable (by the given s@ test).

   (b) Else, select a test pair that can detect the fault. If there exists more than one vector in either the detection list or excitation list, select the test pair that detects most faults and has not been included in the test set.

Using the example shown in Table 3.4 again, vector $V5$ is the only test for d s@ 0. Therefore, $V5$ must be the second vector in the transition test pattern for the fault d slow-to-rise. Although each of $V1, V2, V3$ can set d to 0, we select $V1$ as the initial vector since it detects more s@ fault than the other two. Thus, the pair selected is $(V1, V5)$.

3.3.3 Compaction

After using any of the two heuristics we have a set of test-pairs $\{(U_1, U_2), (U_3, U_4), \ldots, (U_{n-1}, U_n)\}$. They were inserted into the test set in the order given above. We next simulate the test-pairs in the reverse order starting with (U_{n-1}, U_n), (U_{n-3}, U_{n-2}), $\ldots$, (U_1, U_2). If a pair (U_{i-1}, U_i) does not detect any new faults then it is dropped.

3.4 Experimental Results

The algorithms described in the previous sections have been implemented in C++. Experimental data are presented for ISCAS85 and full-scan versions of ISCAS89, on a 1.7GHz Pentium 4 PC with 512 MB of memory, running the Linux Operating

Table 3.5: Results for Fault-List-Based Extension.

Ckt	$S@$ $vectors$	$Trans$ $Pat.$	$Exten$ $time(s)$	$Comp$ $Pat.$	$Comp$ $time(s)$	$Trans$ $FC(\%)$	$Total$ $Time$
c880	128	333	0.23	256	0.24	100	0.47
c1355	198	587	0.43	420	0.47	99.77	0.90
c1908	143	580	0.41	437	0.74	99.67	1.15
c3540	202	1057	1.52	796	3.91	96.27	5.43
c5315	157	628	1.98	556	3.00	99.54	4.98
c6288	41	302	1.30	229	1.50	99.19	2.80
S344	31	132	0.04	102	0.03	100	0.07
S382	41	166	0.04	114	0.07	100	0.11
S526	82	285	0.10	224	0.13	99.93	0.23
S832	179	662	0.35	475	0.41	99.20	0.76
S1196	197	686	0.56	521	0.71	99.97	1.27
S1423	97	382	0.34	288	0.48	99.11	0.82
S5378	332	1397	2.98	1109	5.08	98.40	8.06
S35932	78	507	11.34	417	22.70	90.50	34.04
S38417	1207	4654	52.36	3805	100.76	99.66	153.12
S38584	893	4386	68.33	3475	175.62	95.02	243.95

System. Tables 3.5 and 3.6 report the results for the fault-list-based and priority-based heuristics, respectively. All times given are in CPU seconds. In these two tables, for each circuit, the number of s@ vectors is given first, followed by the initial number of transition patterns computed. Next, the runtime for the pre-compression phase is shown. This time includes the "dictionary computation time". The next two columns report the number of transition patterns after compression and the compression time needed. The total transition coverage and total time are reported in the final two columns. One additional column (rightmost in Table 3.6) shows the percentage improvement the priority-based algorithm has over the fault-list-based algorithm in terms of test data volume. For example, in Table 3.6, the initial STRATEGATE s@ test set has 197 vectors for fully scanned S1196 circuit, the pre-

Table 3.6: Results for Priority-based Extension.

Ckt	$S@$ $vectors$	$Trans$ $Pat.$	$Exten$ $time(s)$	$Comp$ $Pat.$	$Comp$ $time(s)$	$Trans$ $FC(\%)$	$Total$ $Time$	Improve (%)
c880	128	257	0.25	203	0.23	100	0.48	20.70
c1355	198	461	0.53	336	0.42	99.77	0.97	20.00
c1908	143	440	0.51	375	0.57	99.67	1.08	14.19
c3540	202	755	2.06	629	3.06	96.27	5.12	22.91
c5315	157	421	2.14	396	2.11	99.54	4.25	28.78
c6288	41	268	1.36	190	1.19	99.19	2.55	20.53
S344	31	96	0.05	84	0.04	100	0.09	17.65
S382	41	120	0.05	101	0.08	100	0.13	11.40
S526	82	209	0.09	176	0.12	99.93	0.21	21.43
S832	179	450	0.42	386	0.38	99.20	0.80	18.74
S1196	197	532	0.71	428	0.69	99.97	1.40	17.85
S1423	97	278	0.40	237	0.43	99.11	0.83	17.71
S5378	332	1069	3.40	924	4.29	98.40	7.69	16.68
S35932	78	399	12.53	347	17.15	90.50	29.68	16.79
S38417	1207	3562	58.31	3152	77.86	99.66	136.17	17.16
S38584	893	3288	76.73	2785	132.80	95.02	209.53	19.86

compressed transition pattern count is 532. These set of patterns were computed in 0.71 seconds from the 197 patterns using the priority-based heuristic. This initial set of patterns was compressed down to 428 in 0.69s. The total transition fault coverage was 99.97%.

From the transition fault coverage numbers in Tables 3.5, 3.6 and Table 3.2, it is clear that the s@ fault coverage serves as the maximum achievable if we restrict ourselves only to the provided s@ tests.

For most circuits, the results by the priority-based algorithm, prior to compaction, are *already* better than those obtained by the fault-list-based followed by compaction. As indicated in the right-most column of Table 3.6, the average improvement over all benchmark circuits was 18.90%.

The efficiency of the priority-based heuristic is competitive with the fault-list based heuristic, even though fault simulation without fault-dropping on selected faults is needed. We observed that the pre-compaction phase of the priority-based algorithm takes longer than the fault-list-based algorithm. However, since the compaction phase takes about 60% of the time, the total time required by the priority-based algorithm was often shorter due to a smaller starting test set.

3.5 Comparison With Commercial ATPG

We have shown so far that reusing s@ vectors to compose transition patterns is effective in composing transition patterns. An important question arises is "Can this approach be a substitute for a native-mode transition fault ATPG?" In this section we compare the priority based heuristic with a state-of-the-art commercial transition fault ATPG.

The experiment consisted of generating two test sets using the commercial tool. A compressed s@ test set, T_1, and a compressed transition test set, T_2. Then, we applied the priority-based algorithm (and compression) technique to the s@ test set T_1 to obtain another transition test set, T_3. The results are tabulated in Table 3.7.

In Table 3.7, for each circuit, the number of s@ vectors produced by the commercial ATPG is first given. Next, the number of transition patterns generated, transition fault coverage, and ATPG time are listed for the commercial ATPG and our priority-based heuristic alternately. Note that the data storage and the test application time will be proportional to twice this pattern count. The final column reports the improvement in test data volume, as well as test time, is calculated as follows:

$$Data\ Volume\ Improve = 100 * \frac{|T_2| - |T_3|}{|T_2|}$$

For three cases the commercial ATPG performs better than our heuristic. For C5315 the difference is marginal. In the other two cases, C6288 and S35932, the highly compressed s@ vectors T_1 leaves very little room for the heuristic to pick from a variety of vectors. These scenarios are due to the fact that these circuits are very deep and narrow circuits. If we consider the trend that circuits today are becoming flatter and broader to meet cycle time goals, s@ tests for newer high performance circuits will not have these characteristics. From that perspective these two examples can be ignored. In the other cases, there is a substantial improvement in the test data volume. For S9234, the compression is 50%. The average compression over all circuits, including those with negative improvement, is 19.59%.

Table 3.7: Comparison with Native Transition ATPG Tool.

	T_1 (S@	Pattern Count		Coverage		Time		Data Vol.
Ckt	Vectors)	T_2	T_3	T_2	T_3	T_2	T_3	Improve
C1908	129	263	249	99.72	99.72	4.2	2.04	5.32
C2670	116	198	145	78.61	79.26	4.3	4.37	26.77
C3540	179	366	317	82.94	87.62	6.8	10.71	13.39
C5315	124	248	249	96.63	97.05	4.8	9.06	-0.40
C6288	36	90	111	98.95	98.54	3.6	4.28	-23.33
C7552	237	378	322	90.95	91.61	11.0	15.64	14.82
S5378	266	490	316	86.57	87.51	5.3	13.43	35.51
S9234	410	837	412	68.62	70.58	14.7	82.87	50.78
S13207	485	1002	444	80.53	82.29	27.4	70.95	55.69
S15850	464	924	522	84.98	85.76	28.0	101.30	43.51
S35932	75	137	183	90.02	90.33	94.3	61.86	-33.57
S38417	1017	1927	1027	89.92	91.19	115.6	293.30	46.71

T_1: s@ test set generated by commercial ATPG T_2: transition test set generated by commercial ATPG

T_3: transition test set generated by our priority-based algorithm using T_1

Columns 5 and 6 give the coverage numbers. In all cases, except for C6288, our priority heuristic gives a higher fault coverage than the commercial ATPG. For

C3540 the difference is about 5%. Columns 7 and 8 give the execution times for the commercial ATPG and our technique. The commercial ATPG was run on a different machine, and we cannot conclude which is faster. The data presented simply shows that the run times are comparable.

3.6 Additional Benefits of Reusing s@ Vectors

The approach of constructing enhanced-scan transition patterns using only s@ vectors has several additional benefits. Firstly, prior to using the test-patterns in manufacturing, the transition patterns and the vectors of which it is comprised must be validated. Since the s@ vectors have already gone through that process, the patterns in the transition tests computed by our algorithm can skip this validation process. Note, however, that transition patterns still have to be validated on silicon against yield loss. Thus, although we have not eliminated pattern validation from the flow, the overhead has been considerably reduced. From a project execution point of view this is a tremendous productivity boost.

Secondly, none of our test cases contain any contention since they are fully synthesized logic. For designs where contention can occur, a tool usually extracts such contention-causing constraints which are fed to the ATPG tool. The ATPG tool then generates tests that satisfies these constraints. It is a well-known fact that ATPG tools work very hard to satisfy such constraints and that adversely affects the run time, coverage and test set size. Our approach scores over native mode transition fault ATPG in that it leverages off the "hard work" of the s@ ATPG phase. Since all the s@ vectors are contention free the transition test set our algorithm generates is also contention free.

Finally, increasing numbers of IP cores are now shipped with their test sets as part of their design data. Compressing these test sets to contain the design data

is becoming increasingly important. If transition tests are added to the test data, then the test data volume problem is compounded. Since we are using the s@ test set we need to store only one copy of the vectors that are used several times. The transition test set can, then, be expressed as a sequence of pointers to these set of vectors. Thus, with a modest increase over the storage requirement over the s@ test set we can also ship the transition test set.

Table 3.8: Improvement on Data Storage Requirement

Ckt	S Cells	S@ Vec.	Trans Pat.	S@ Str.	Tr Pointer Str.	% of S@ Str.(ptr)	% of S@ Str.(pat)	Factor Red.
C2670	157	116	145	18212	9280	50.96	250	4.91
C5315	178	124	249	22072	15936	72.20	401.61	5.56
C7552	207	237	322	49059	20608	42.01	271.73	6.47
S5378	228	266	316	60648	20224	33.35	237.59	7.13
S9234	250	410	412	102500	26368	25.72	200.98	7.81
S13207	790	485	444	383150	28416	7.42	183.09	24.69
S15850	684	464	522	317376	33408	10.53	225	21.38
S35932	2048	75	183	153600	11712	7.63	488	64
S38417	1742	1017	1027	1771614	65728	3.71	201.97	54.44

Quantitative data is presented in Table 3.8. In this table, the second column gives the number of scan cells. We assume that the primary inputs and the state elements are all part of the scan chain. We have only included circuits in which the number of scan cells exceed 100. Our method will become more effective as the number of scan cells increase as will become clear shortly. The next column gives the number of s@ vectors followed by the number of transition patterns. The transition pattern set assumed is the set computed by the priority-based algorithm, where each transition pattern consists of two vectors. The fifth column gives the number of bits of storage required to store the s@ vectors. We next assume that instead of storing the vectors in the transition patterns we use a pair of 32-bit pointers indicating the location of the s@ vectors. The total storage required to store the

sequence of pointer-pairs representing the transition patterns is shown in column 6. Column 7 gives the percentage increase, above and beyond the s@ pattern storage, required to store the transition pattern pointer set. Note that the incremental storage requirement varies from about 7.6% to 72.2% with the average being about 28.17%. On the other hand, we could have stored the explicit transition patterns by storing the vectors in each transition pattern. The incremental storage requirement is shown in Column 8. The average increase in storage requirement is about 273%. Finally, Column 9 shows the factor reduction in the incremental storage required by the two methods. The average reduction is a factor of 21.82 (2182%) with the highest reduction is a reduction of a factor of 64.

3.7 Summary

We presented two algorithms, Fault-List-based extension and Priority-based extension, for composing transition patterns from vectors in a s@ test set. The priority-based algorithm was shown to be superior to the fault-list based algorithm. It was demonstrated that a high quality transition pattern sets can be obtained, bypassing the need for a native mode transition fault ATPG. Experimental comparison with a native mode transition fault ATPG tool showed the proposed heuristics resulted in 20% smaller pattern set while achieving the same or higher transition fault coverage. We discussed the additional advantages of reusing the s@ vectors in pattern validation, constraint handling and reducing design data in the context of IP cores.

Chapter 4

Efficient Transition Testing Using Test Chains and Exchange Scan

4.1 Background and Motivation

As we introduced before, transition tests can be applied in three different ways: **Broadside** [SAV94], **Skewed-Load** [SAV92a] and **Enhanced-Scan** [DS91]. For broadside testing (*also called functional justification*), a vector is scanned in and the functional clock pulsed to create the transition and subsequently capture the response. For each pattern in broadside testing only one vector is stored in tester scan memory. The second vector is derived from the first by pulsing the functional clock. For skewed-load transition testing, an N-bit vector is loaded by shifting in the first N-1 bits, where N is the scan chain length. The last shift clock is used to launch the transition. This is followed by a quick capture. For skewed-load testing also, only one vector is stored for each transition pattern in tester scan memory; the first vector is a shifted version of the stored vector. Finally, for enhanced-scan

transition testing, two vectors (V1, V2) are stored in the tester scan memory. The first scan shift loads V1 into the scan chain. It is then applied to the circuit under test to initialize it. Next, V2 is scanned in, followed by an apply and subsequently a capture of the response. During shifting in of V2 it is assumed that the initialization of V1 is not destroyed. Therefore enhanced-scan transition testing assumes a hold-scan design [DS91].

Table 4.1: Test Data Volume Comparison

$Name$	$S@Test$	$TransTest$	$Expansion$
c3540	179	366	4.09
c5315	124	248	4
c6288	36	80	5
c7552	237	378	3.19
s13207	485	1002	4.13
s15850	464	924	3.98
s35932	75	137	3.65
s38417	1017	1927	3.79
s38584	727	1361	3.74

Among the three kinds of transition tests, broadside suffers from poor fault coverage [SP94b]. Since there is no dependency between the two vectors in Enhanced scan, it can give better coverage than skewed-load transition test. Skewed-load transition tests also lead to larger test data volume. Compared to stuck-at tests, the increase in the number of vectors required for enhanced scan to get complete coverage is about 4X (Table 4.1). This data, collected using a state-of-the-art commercial ATPG tool, shows the number of stuck-at vectors and the number of enhanced scan transition patterns for each circuit. Note that each transition pattern consists of two vectors. For skewedload transition test, it has been observed that the data volume for an ASIC has an increase of 5.9X [HBP01].

A drawback of enhanced-scan testing is that it requires hold-scan scan-cells.

However, in microprocessors and other high performance circuits that require custom design, such cells are used for other reasons. In custom designs, the circuit often is not fully decoded, hold scan cells are used to prevent contention in the data being shifted, as well as preventing excessive power dissipation in the circuit during the scan shift phase. Furthermore, if hold-scan cells are used, the failing parts in which only the scan logic failed can often be retrieved; thus enhancing, to some extent, the diagnostic capability associated with scan DFT. In this chapter, we will consider only enhanced-scan transition tests.

A number of techniques on reducing test data volume and test application time for single cycle scan-tests have been presented in the literature [KBB+01, HP99, KOE91, CC01, LCH98, DT00]. These methods assume that only 5-10% of the bits are fully specified. Unspecified bits are filled to detect the easy to detect faults. Different codes to compress the information in the tester and decompressing them on chip [KBB+01, CC01, DT00] or using partitioning and applying similar patterns to the different partitions [HP99, LCH98] have been proposed. The techniques proposed here compliments the work on compressing individual vectors.

In our previous work [LHCT03], we presented two algorithms for computing transition test patterns from generated s@ test vectors, which can reduce the test set size by 20%. Nevertheless, there has not been much work on addressing the explosion in data and application time for transition tests. To tackle this problem, we first propose novel techniques to generate effective transition test chains based on stuck-at vectors, thus the need for a separate transition ATPG is eliminated. Next, we propose methods to reduce the transition test data volume and test application time. Our optimization techniques consider factors across test patterns for transition tests. The first technique uses the ATE repeat option to reduce the test data volume and requires the use of transition test chains, rather than randomly apply the indi-

vidual transition test patterns. We describe the transition test chains and present a novel algorithm for computing such chains. Experimental results show an average data volume reduction of 46.5%, when compared with the conventional enhanced scan transition test data volume computed by COM, a commercial ATPG tool. The above technique does not necessarily decrease test application time. To reduce test application time, a new DFT technique called exchange scan is proposed . Combining exchange scan with transition test chains reduces both test application time and test data volume by 46.5%, when compared with a conventional transition test set computed by COM.

Nevertheless, one of the drawbacks of scan-based delay tests is possible yield loss due to overtesting. In [REA01], the author showed that scan-based testing may fail a chip due to the delay faults that do not affect the normal operation, and thus it is unnecessary to target those functionally unsensitizable faults. In [LKC00a, LKC00c], the authors also addressed the impact of delay defects on the functionally untestable paths on the circuit performance. Moreover, a scan test pattern, though derived from targeting functionally testable transition faults, can incidentally detect functionally untestable faults if the starting state is an arbitrary state (could be an unreachable state). In this chapter, we use a low-cost implication engine to compute the functionally untestable faults and address the problem of overtesting in our graph-formulated transition ATPG algorithm.

The rest of the chapter is organized as follows. In Section 4.2, the ATE model we use is described. Section 4.2.1 proposes a novel transition test chain formulation, which is mapped into a weighted transition pattern graph traversal problem. A new DFT technique to reduce test application time by reducing the number of scan loads is presented in Section 4.3 . Section 4.4 describes a novel method to address the problem of incidentally overtesting of functionally untestable faults

in our ATPG algorithm. Section 4.5 presents a summary of all the experimental results. Finally Section 4.6 concludes the chapter.

4.2 ATE Model

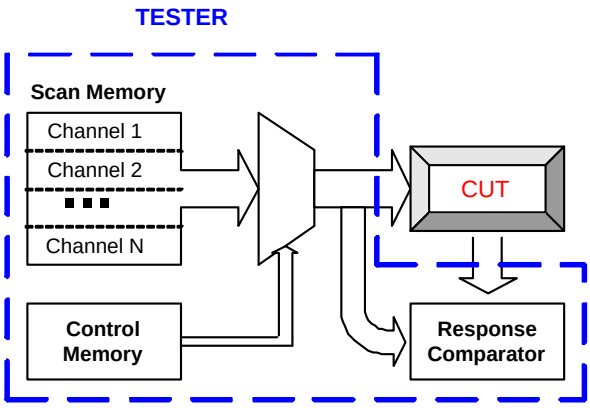


Figure 4.1: Tester Memory Model

Figure 4.1 is an abstraction of the tester model we use. ATE storage consists of two parts: scan and control memory. Scan memory is divided into several channels. Each channel consists of three bits. For each clock cycle of the scan shift operation, the scan memory contains the bit to be scanned in, the expected response bit from the circuit under test (CUT) and an indication of whether this bit of the response is to be masked or not, indicated by M or U in the figure 4.2. Figure 4.2(a) shows the data stored for a single scan channel for the test set {V1,V2,V3,V4,V5,V6,V7,V8,V9,V10}, with Rj being the expected response for Vj. The control memory controls the shift and the comparison operation. The scan memory depth required is (N+1)*S bits, for a test set of size N and scan length S.

The enhanced scan transition test set in Table 4.2 consists of 6 transition test

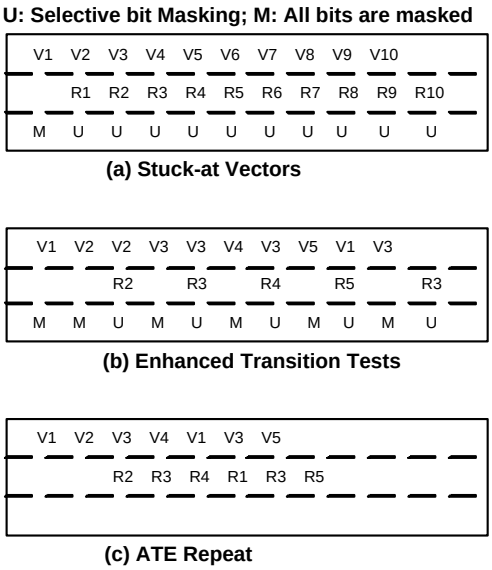


Figure 4.2: ATE Storage Model

Table 4.2: Enhanced Scan Transition Test Set Example

$Vector1$	$Vector2$	$Response$
V1	V2	R2
V2	V3	R3
V3	V4	R4
V3	V5	R5
V1	V3	R3*
V4	V3	R3**

patterns. Each pattern consists of a pair of vectors and the expected response to the test vector. The first pattern in our example consists of the pair (V1, V2) and the response R2 to V2. Storage of the test data in the scan memory is shown in Figure 4.2(b). Storage depth required is N*2*S+N*S bits. The control sequence for this test set shown in Table 4.3 is very repetitive and stored in the tester scan memory. Row 1 of the table states that V1 is scanned in and applied to the CUT. Row 2 states that V2 is scanned in, applied to the CUT, and the response R2 captured. Row 3

Table 4.3: Control Sequence of Transition Test

(i)Shift in V1;(ii)Apply;
(i)Shift in V2;(ii)Apply;(iii)Capture;
(i)Shift in V2;Shift Out and Compare R2;(ii)Apply;
(i)Shift in V3;(ii)Apply;(iii)Capture;
(i)Shift in V3;Shift Out and Compare R3;(ii)Apply;
(i)Shift in V4;(ii)Apply;(iii)Capture;
(i)Shift in V3;Shift Out and Compare R4;(ii)Apply;
(i)Shift in V5;(ii)Apply;(iii)Capture;
(i)Shift in V1;Shift Out and Compare R5;(ii)Apply;
(i)Shift in V3;(ii)Apply;(iii)Capture;
(i)Shift in V4;Shift Out and Compare R3*;(ii)Apply;
(i)Shift in V3;(ii)Apply;(iii)Capture;
(i)Shift out and compare R3**;

states that the first vector of the next vector pair, i.e. V2, is scanned in while the response R2 of the previous test pattern is scanned out and compared against the expected response. Once the scan operation is complete, the new vector is applied to the CUT. The rest of the entries can be similarly interpreted.

4.2.1 ATE Repeat and Transition Test Chains

4.2.2 ATE Repeat

There is considerable redundancy in the information stored in the tester. In Figure 4.2(b), V2, V3 are used several times in the test sequence. Ideally, one copy of the information should suffice. However, storing one copy of a vector and reusing it in any random order requires the ATE to index into random locations in the scan memory, which is currently not available. Limited reuse of the information stored however is possible. In Figure 4.2(b), two copies of V2 are stored in consecutive locations in the scan memory. It is possible to store just one copy of V2 and scan in

V2 as often as possible during consecutive scan cycles. Similarly, we can replace two copies of V3 in locations 4, 5, from the left of the scan memory in Figure 4.2(b), with just one copy of V3. Further reduction of the number of copies of V3 is not possible. Thus, we store the sequence {V1, V2*, V3*, V4, V3, V5, V1, V3} and repeatedly scan in the vectors marked with the flag *. Information about vectors that needs to be scanned in multiple times is stored in control memory. Thus, 8 instead of 10 vectors need to be stored.

In the above example, the scan storage requirement was reduced at a price. Since vectors that are scanned in repeatedly do not form a regular pattern, the control memory requirement can increase drastically. To avoid such an increase in control memory we impose a restriction that, except for the first vector and last vector stored in the scan memory, every vector is scanned in *exactly* twice. In Figure 4.2(c), the sequence {V1, V2, V3, V4, V1, V3, V5} is stored. Assuming that all but the first and last vectors are scanned in twice, the set of transition test patterns applied is (V1, V2,), (V2, V3), (V3, V4), (V4, V1), (V1, V3,), (V3, V5). This set of patterns includes all the test patterns of Figure 4.2(a) as well as (V4, V1). Thus, by storing only 7 vectors (instead of 10 vectors), all the transition tests can be applied. For this example, not only is control memory requirement lower but the number of vectors stored is also reduced. Sequences, in which all but the first and last vectors are scanned in twice are defined to be *transition test chains*.

4.2.3 Transition Test Chains via Weighted Transition Graph

Computing transition test chains is different from computing a set of transition test patterns as is conventionally done. A novel algorithm, called *weighted transition graph algorithm*, to compute such chains is discussed next.

The algorithm constructs transition test chains from a given stuck-at test set. Instead of computing a set of vector-pairs and chain them together as alluded to in the above examples, the problem is mapped into a graph traversal problem. The algorithm builds a weighted directed graph called the *weighted transition-pattern graph*. In this graph, each node represents a vector in the stuck-at test set; a directed edge from node Vi to Vj denotes the transition test pattern (Vi, Vj) and its weight indicates the number of transition faults detected by (Vi, Vj). Directed edges may potentially exist between every node pair, resulting in a large (possibly complete) graph. In order to reduce the time required to construct the graph, only the subset of the faults missed by the original stuck-at test set are considered. The graph construction procedure is discussed next.

We start with the stuck-at test set T=$\{T_1 ... T_N\}$.

1. Perform transition fault simulation using the stuck-at test set as a transition test set $\{(T_1, T_2), (T_2, T_3)...(T_{N-1}, T_N)\}$ to compute *undet*, the set of undetected transition faults.
2. Deduce the subset *U* of the stuck-at faults implied by *undet* as follows. If X slow-to-rise or slow-to-fall fault $\in$ *undet* , then both X stuck-at-0 and stuck-at-1 are included in *U*.
3. Perform stuck-at fault simulation, without fault dropping, using the stuck-at test set *T* on the stuck-at faults in *U*. For each stuck-at fault f in *U*, record the vectors in *T* that excite f and the vectors that detect f. Also, for each vector, the faults excited and detected by it are recorded.
4. The weighted directed graph contains a node corresponding to each of the stuck-at tests in *T*. The directed edge, from V_i to V_j, is inserted if the corresponding test pattern (T_i, T_j) detects at least one transition fault in *undet*.

The weight of (T_i, T_j) is the number of transition faults in *undet* detected by (T_i, T_j).

For example, consider a circuit with 5 gates (10 stuck-at faults) and a stuck-at test set consisting of 4 vectors V1, V2, V3, and V4. The excitation and detection dictionary obtained by simulation without fault dropping are as shown in Table 4.4.

Assuming the test set order to be {V1, V2, V3, V4}, only 3 transition faults (slow-to-fall at c, e and slow-to-rise at c) are detected. However, using Table 4.4, we can make the following observations by combining non-consecutive vectors: (V1, V3) detects a slow-to-fall; (V3, V1) detects a slow-to-rise; (V1, V4) detects d slow-to-fall; (V4, V2) detects d slow-to-rise; (V4, V1) detects a slow-to-rise, b slow-to-fall; and (V2, V4) detects b slow-to-rise, e slow-to-rise and d slow-to-fall. This results in the transition-pattern graph of Figure 4.4 .

Table 4.4: Fault Dictionary without Fault-Dropping

$Vector1$	$ExcitedFaults$	$DetectedFaults$
V1	a-s-0,b-s-1,c-s-1,d-s-0,e-s-0	a-s-0,b-s-1
V2	b-s-1,c-s-0,d-s-0,e-s-1	c-s-0,d-s-0,e-s-1
V3	a-s-1,c-s-1,	a-s-1,c-s-1
V4	a-s-1,b-s-0,d-s-1,e-s-0	b-s-0,d-s-1,e-s-0

Unlike general graphs, this weighted transition graph has a specific property, which formulates the following theorem.

Theorem 1 *Faults detected by pattern (V_i, V_j) and faults detected by pattern (Vj, Vk) are mutually exclusive.*

Proof: We prove this by contradiction. Without loss of generality, consider fault f *slow-to-fall*detected by (V_i, V_j). Thus, V_i excites f *s-a-0*(sets line f to 1)

GENERIC WEIGHTED TRANSITION GRAPH ALGORITHM

```
construct the weighted transition pattern graph;
initialize P to T={T1...Tn}
while(TFC<100%) &&(iteration num<MAX))
    BEGIN
      identify an edge(Vi,Vj) with the heaviest weight;
      append vectors Vi,Vj to set set P;
      for all the edges starting from Vj
          BEGIN
            look for the edge (Vj, Vk) having the heaviest weight;
            append vector Vk to the test set P;
          END
    END
```

Figure 4.3: Generic Weighted Transition Graph Algorithm

and V_j detects f *s-a-1*. Assume (V_j,V_k) also detects f *slow-to-fall*. Then, the initial vector V_j must set line f to 1, which is a contradicts.

An Euler trail in the transition-pattern graph traverses all the edges in the graph exactly once. By inserting a minimal number of edges to the graph, an Euler trail that traverses all edges would tempt us to think that this is the best test chain. However, this actually leads only to a sub-optimal solution. Traversing edge (Vi,Vj) is equivalent to selecting test (Vi,Vj). Once edge (Vi,Vj) is traversed, i.e test (Vi, Vj) is selected, it detects a number of transition faults. This alters the weights on other edges and even removes some of the edges. Per Theorem 1, edges whose weights do not change are those originating from Vj. To improve the solution, the edge weights should be updated after traversing each edge. A preliminary version of the algorithm is outlined in Figure 4.3 where P is the transition test chain computed by the algorithm from the given stuck-at test set T.

The idea behind this algorithm is as follows: we are looking for a smallest

test sequence that can cover all the detectable faults by traversing the weighted transition pattern graph. In a traditional weighted graph, traversing any edge in the graph will *not* affect the weight on other edges; therefore, an Euler trail, which traverse each edge exactly once, will be the optimum solution. But in our case, traversing any edge in the weighted transition pattern graph may result in a number of transition faults be detected and removed from the fault dictionary. Thus, the weight on other edges, which also detect these faults, will be altered and must be updated by simulation.

For example, in the original weighted transition pattern graph on the left of Figure 4.4, (V2,V4,V1) is the heaviest-weight test chain of length 3. After traversing this chain, 5 transition faults (a slow-to-rise, b slow-to-rise, b slow-to-fall, d slow-to-fall and e slow-to rise) have been detected. The updated graph is shown on the right of Figure 4.4. It should be noted that in addition to removing the edges (V2,V4) and (V4,V1), other two edges ((V1,V4) and (V3,V1)) are also removed from the graph. This is because the fault (d slow-to-fall), which can be detected by (V1,V4) has been detected by selecting the test chain (V2,V4,V1). Therefore, the edge (V1,V4) can be removed from the weighted pattern graph because it has no contribution to the future selection of edges. Similar idea can be applied on the edge (V3,V1) as well. Finally, all the 7 undetected faults in Table 4.4 are detected with the test chain {V2, V4, V1, V3, V4, V2}.

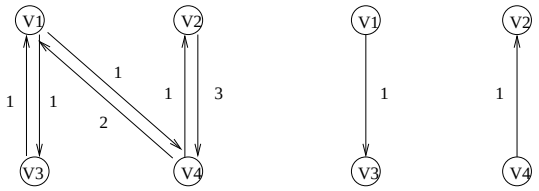


Figure 4.4: Weighted Transition-Pattern Graph Example

Several optimizations were applied to the generic version of the algorithm to improve the results. First, we investigated the impact of different lengths of test chain on the number of faults detected. Instead of considering one edge at a time, we use Theorem 1 to inspect path segments of length 3. This reduces the amount of simulation required and a transition test chain is generated by incrementally concatenating the vector-chains of length 3. While this solution is better than simply concatenating vector pairs for each of the remaining undetected transition faults, it still may not be optimal. When the test chain length increases beyond 3, the difference between the actual number of transition faults detected by the chain and the sum of edge weights in the chain can increase. This difference determines whether it is worthwhile to use continue extending the chains. Thus, using longer chains reduces the number of graph updates (hence reducing run time of the algorithm), but it will increase the size of the final solution. Our experiments suggest that the chain lengths of 3 or 4 give the best results.

Secondly, expanding on the essential test definition for stuck-at fault from [HP99], we define an essential vector for transition faults to be any test vector that excites (or detects) at least one transition fault that is not excited (or detected) by any other test vector in the test set. All essential vectors must occur in the transition test chain at least once. We include essential vectors early in the chaining process. The transition test chain generation process is divided into two phases by constructing two weighted transition pattern graphs.

1. Identify all essential vectors, generate the transition-pattern graph using only essential vectors and construct the test chains only with the essential vectors in the graph. Append that to the initial transition test chain P in the generic version of the weighted transition graph algorithm.

2. Reduce the number of faults by dropping faults detected in the first step. Generate the transition-pattern graph for the remaining faults and extend the partial transition test chain from previous step as described in step 3 of the generic weighted transition graph algorithm.

The number of edges in the second step of the modified algorithm is significantly reduced, because most of the edges incident on the essential vectors have been traversed in the previous step and thus removed.

During the test pattern generation procedure, some of the faults detected by the earlier test vectors may also be detected by the test patterns generated later. Therefore, vectors added early in our transition test set might become redundant. To identify such redundant patterns, we perform a *reverse-order pair-wise compaction*. After (U_1, U_2, U_3), (U_4, U_5, U_6), ... , (U_{n-2}, U_{n-1}, U_n) are generated, they are appended to the original stuck-at test set in that order. The test patterns (U_{n-1}, U_n), (U_{n-2}, U_{n-1}),..., (U_1, U_2) are simulated in the reverse order in which it was generated. If neither (U_{i-1}, U_i), (U_i, U_{i+1}) detects any additional fault, then U_i is redundant and can be eliminated. Note that eliminating a vector U_i will give rise to a new transition test pattern (U_{i-1}, U_{i+1}); therefore, we follow this by performing a forward-order pair-wise compaction step to further reduce the size of the test sequence.

4.3 Exchange Scan

We saw that by using transition test chains and ATE repeat can reduce test data storage. Data presented in experimental result will show the average reduction to be about 46.5%. However, test application time can sometimes increase due to

repeating on every vector in the chain. To reduce test application time, while still retaining the improvement in data storage, we propose a new DFT technique.

Instead of using the ATE repeat option to re-use the vector, an low overhead alternative is possible in which each vector is only scanned in once. Reducing scan-in operations reduces test application time. The net result is a reduction in both the data storage requirement and the test application time.

The block diagram of a conventional hold-scan system is shown in Figure 4.5. The scan cells, each of which consists of two parts: System Latch and Shadow MSFF are chained together to form two related registers: SYSTEM REGISTER and SHADOW REGISTER. During normal operation, the SYSTEM REGISTER is in use and the SHADOW REGISTER does not play any role. For scan testing options, three scan operations– SCAN_SHIFT, SCAN_LOAD and SCAN_STORE, are supported.

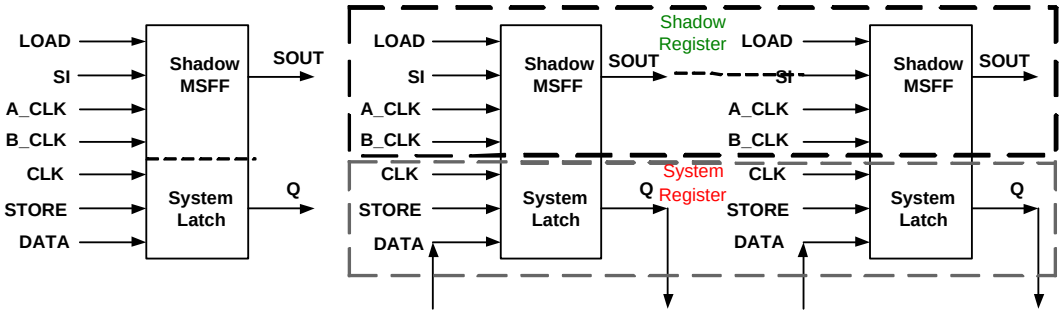


Figure 4.5: Block Diagram of Hold-Scan System

Assume the scan cell implementation of Figure 4.6. For SCAN_SHIFT, the A_CLK and B_CLK are pulsed so that data passes from the SI to the SOUT. For SCAN_STORE, the STORE signal is pulsed to transfer the data from SOUT to Q. The content of SHADOW REGISTER is transferred to the SYSTEM REGISTER. In SCAN_LOAD the contents of the SYSTEM REGISTER is transferred to the

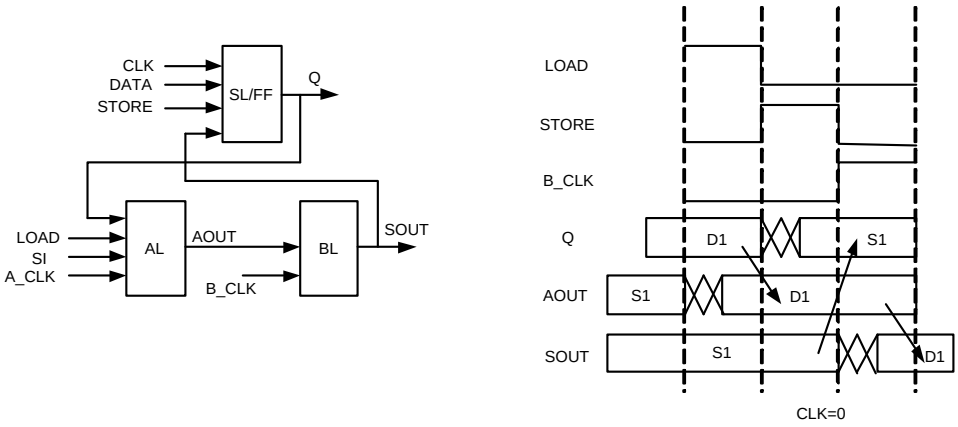


Figure 4.6: Hold Scan Cell and Exchange Scan Timing

SHADOW REGISTER. Pulsing LOAD transfers data from Q to AOUT. Pulsing the BCLK to transfer the data from AL to BL.

The new operation SCAN_EXCHANGE exchanges the data between the SHADOW and the SYSTEM registers, without destroying either of them. Pulsing LOAD transfers the contents of SL to AL. Pulsing STORE transfers the contents of BL to SL. Pulsing B_CLK to transfer the content of AL to BL, follows this. The corresponding timing diagram is shown in Figure 4.6 (b). No additional hardware or signal is needed to support the exchange operation. It may require the global scan controller to be modified slightly to realize the exchange operation. The exchange operation takes about three clock cycles.

SCAN_EXCHANGE, for the transition test chain {V1, V2, V3, V4} is used as follows. Test-pairs applied are: (V1, V2), (V2, V3), (V3, V4). The expected response on application of V2 is R2, V3 is R3 and V4 is R4. The sequence of operations, without exchange scan, is shown in the first column of Figure 4.7. *Capture R2* implies that the response of the CUT on application of V2 is latched on to the SYSTEM REGISTER. Scan in V2, Scan Out R2 implies SCAN_SHIFT wherein

WITHOUT EXCHANGE	WITH EXCHANGE
Scan in V1	Scan in V1
Store V1	Store V1
Scan in V2	Scan in V2
Store V2	Store V2
Capture R2	Capture R2
Load R2	Exchange (R2, V2)
Scan in V2, Scan out R2	Scan in V3, Scan out R2
Store V2	Store V3
Scan in V3	Capture R3
Store V3	Exchange (R3, V3)
Capture R3	Scan in V4, Scan out R3
Load R3	Store V4
Scan in V3, Scan out R3	Capture R4
Store V3	Exchange (R4, V4)
Scan in V4	Scan in V5, Scan out R4
Store V4	
Capture R4	
Scan in V4, Scan out R4	

Figure 4.7: Scan Operation with/without exchange

V2 is scanned out and response of the CUT, from the previous pattern is compared to R2.

The second column of Figure 4.7 shows the sequence of operations using SCAN_EXCHANGE. Once V2 is loaded into the SHADOW REGISTER, the subsequent store and capture operations do not destroy the contents of the SHADOW register. So, the SCAN_LOAD operation that destroys the contents of the SHADOW REGISTER is replaced by the SCAN_EXCHANGE operation. It exchanges the contents of the SHADOW REGISTER and the SYSTEM REGISTER. Thus, the captured response is transferred to the SHADOW REGISTER and V2 is applied to the circuit under test as the *initial* vector for the next test pair. We can therefore skip the sequence of operations that rescans V2 and stores it. In addition, the SCAN_SHIFT of the response from V2, i.e. R2, can now be merged with the

SCAN_SHIFT of the final vector of the next pattern V3. The net effect of this is that we can replace an entire scan operation with a 3-cycle SCAN_EXCHANGE operation. Considering that the SCAN_SHIFT operation may take 1000 or more clock cycles this overhead of the SCAN_EXCHANGE operation is negligible and will be neglected from our calculations. Note that transition test chains, but not ATE repeat capability in the testers, is required to realize the gains of the exchange scan operation. If the ATE Repeat capability is available, each of the vectors that are stored can be compressed, as discussed in [Keller 2001], and the benefits of transition test chains can be realized using exchange scan. Our experimental result show that both test data volume and test application time decrease by 46.5%, compared to a commercial tool.

4.4 Constrained ATPG to Minimize Overtesting

In this section, we present a novel algorithm to address the possible yield loss due to overtesting of functionally untestable faults in Enhanced-scan testing. A transition fault is functionally untestable if either launching of the transition or the propagation of its effect is impossible in the functional mode, due to constraints imposed by the circuit. Such constraints include the requirement for an illegal/unreachable state in the test pattern, or the two state combination in the enhanced-scan pattern is functionally impossible. Because the enhanced-scan model assumes total independence between the two vectors in the test pattern, some functionally untestable transition faults may become detected. Figure 4.8 illustrates the scenario. Since an enhanced-scan pattern consists of (State1, V1, State2, V2), if either (1) State1/State2 is illegal or (2) the State1/State2 pair is not a valid state transition combination, the transition pattern may detect some of the functionally untestable faults. To avoid detection of

such functionally untestable faults, we must make sure that these two scenarios do not arise.

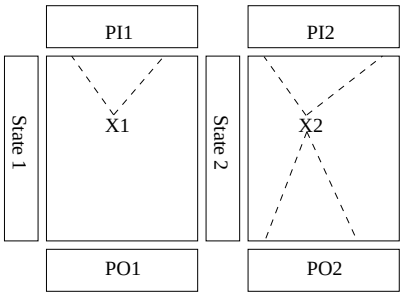


Figure 4.8: Arbitrary starting states in Enhanced Scan

We first describe a low-cost implication based functionally-untestable-fault identification method without involving any sequential ATPG. Then we try to minimize the overtesting of these functionally untestable faults in our graph-formulated ATPG algorithm described in section 4.2.1.

In general, identifying functionally untestable fault in sequential circuits is of the same complexity as sequential ATPG. In [HSI02], a method for identifying untestable stuck-at faults in sequential circuits by maximizing local conflicting value assignments has been proposed. The technique first computes a large number of logic implications across multiple time-frames and stores them in an implication graph. Then the algorithm identifies impossible combinations of value assignment locally around each gate in the circuit and those redundant stuck-at faults requiring such impossibilities.

For identifying functionally untestable transition faults, in addition to searching for the impossibilities locally around each gate, we also check the excitability of the initial value in the previous time frame. In other words, if opposing values on a signal in two consecutive time frames are not possible, we would know that

no transition can be launched on that signal in the functional mode. Furthermore, if the corresponding s@ fault becomes unobservable due to the imposed constraints, the transition fault also would be unobservable. In employing this technique, we can identify a large set of untestable transition faults in the circuit.

The identified functionally untestable transition faults are mapped onto the graph traversal problem. We build the weighted transition pattern graph for the given stuck-at test set as before. Again, in the graph, each node represents a vector in the stuck-at test set and a directed edge from Vi to Vj denotes the transition test pattern (Vi,Vj). To address the problem of overtesting functionally untestable faults, each edge in the weighted transition pattern graph has *two* weights: W1 indicates the number of functionally untestable faults detected by test pattern (Vi,Vj) and W2 represents the number of functionally testable faults detected by the pattern. Therefore, our target is to achieve the highest transition fault coverage, while minimizing the overtesting of functionally untestable faults.

The heuristic we used to minimize the overtesting is presented next. Assume the stuck-at test set T=T_1...T_n is given.

1. Identify the functionally untestable fault set *R* using the transition implication tool.

2. Perform transition fault simulation using the stuck-at test set as a transition test set $\{(T_1, T_2), (T_2, T_3)...(T_{N-1}, T_N)\}$ to compute *undet*, the set of undetected transition faults.

3. Deduce the subset *U* of the stuck-at faults implied by *undet* as follows. If X slow-to-rise or slow-to-fall fault $\in$ *undet* , then both X stuck-at-0 and stuck-at-1 are included in *U*.

4. Categorize the transition faults in *U* into two parts: U_1=*U*∩*R*, standing for the functionally untestable fault set in *U* and U_2=*U*-U_1 standing for the functionally testable fault set in *U*.

5. Build the fault dictionary for U_1 and U_2 respectively, using the fault simulation without fault-dropping. Generate the weighted pattern graph with W_1 and W_2 on each edge.

6. Compute the weight W on each edge, using the formula W=F(W_1, W_2, threshold). For small circuits, we set the threshold to Zero and for bigger circuits, we predetermined the threshold as a small fraction of the number of functionally untestable faults in the circuit.

7. Greedily construct transition test chains with the maximum W and append the transition test chains to the original stuck-at test set *T*.

Function *F* is defined as follows:

$$F = \begin{cases} W_2 - W_1 & \text{if } W_1 \leq threshold \\ 0 & \text{otherwise} \end{cases}$$

Let us look at the example in Section 4.2.1 again. Suppose the transition fault, a slow-to-rise, is a functionally untestable fault, then the weighted pattern graph in Figure 4.4 will be modified to Figure 4.9. Again, recall that every vertex in the figure is a test vector, there exists an edge between two vertex if they detect at least one fault. Different from the previous example, every edge now has two weights, namely W_1 and W_2, which are the number of functionally testable faults and functionally untestable faults detected by the pair of test vectors, respectively. So, from the fault dictionary, we can see that (V4,V1) detect one functional testable faults (b slow-to-fall) and one functionally untestable fault (a slow-to-rise). There-

fore, the weights on edge (V4,V1) are W_1=1 and W_2=1. Similarly, (V3,V1) detects only a slow-to-rise, which is a functionally untestable fault; correspondingly, the weights on edge (V3,V1) are W_1=1 and W_2=0.

Therefore, selecting a transition test chain no longer depends on merely the total number of transition faults it can potentially detect. We must distinguish the number of functionally testable faults and number of functionally testable faults each chain can detect. For instance, consider the transition test chain {V2, V4, V1, V3, V4, V2} generated in Section 4.2.1; although all the functionally testable faults will be detected, the functionally untestable fault (a slow-to rise) will also be detected. To avoid the overtesting of this functionally untestable fault, we generate the transition test chain by avoiding traversal of the edges containing non-zero-W_1. In doing so, the new transition test chain {V2,V4,V2,V1,V3} (that avoids traversing non-zero W_1 edges) will now only detect 5 out of the 6 remaining functionally testable faults because all the vectors that can detect b slow-to-fall also detect the functionally untestable fault a slow-to-rise. Consequently, a slight drop in fault coverage of the functionally testable faults may result.

We can also relax this condition of avoiding all non-zero W_1 edges to increase coverage of functionally testable faults. In essence, we set a threshold on W_1 such that we will still consider some non-zero W_1 edges, but we make sure that detection of such untestable faults is bounded by the threshold value.

4.5 Experimental Results

The weighted transition graph algorithm, with all the optimizations described above, was implemented in C. Experimental data are presented for ISCAS85 and full-scan versions of ISCAS89 benchmarks, on a 1.7GHz Pentium 4 with 512 MB of mem-

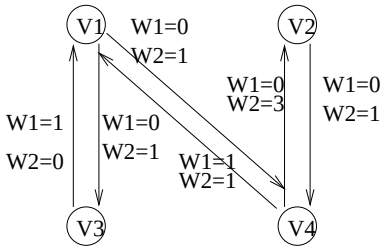


Figure 4.9: Weighted Pattern Graph with two Weights

ory, running the Linux Operating System.

Table 4.5: Results with different chain Lengths

	2		3		4		5		6		7	
$Circuit$	ID	AC	ID	AC	ID	AC	ID	AC	ID	AC	ID	AC
s344	38	38	64	64	90	72	111	89	143	89	164	103
s382	24	24	43	43	63	44	75	56	92	61	111	61
s832	60	60	87	87	114	89	136	95	158	95	180	103
s1196	47	47	78	78	116	93	146	105	177	105	208	119
s1423	36	36	60	60	79	79	106	95	129	118	153	124
s5378	59	59	103	103	149	142	190	144	235	172	284	176
s35932	1072	1072	1175	1175	1270	1247	1382	1261	1475	1312	1564	1334
s38417	132	132	190	190	274	249	380	285	439	296	500	319

In Table 4.5, results for full-scan version of ISCAS89 circuits, with different chain lengths, are presented. For each benchmark, the ideal (ID), given by the sum of the edge weights, and actual (AC) faults detected by the chains are shown. The difference between the ideal (ID) and actual (AC) increases with the chain length. For example, for circuit s5378, when the chain length is 2, the ideal and actual numbers of detected transition faults are the same. Likewise, when the chain length is increased to 3, they are still equal as explained by the proposed Theorem. When we increase the chain length beyond 3, the actual number of detected transition faults start to differ, as some of the faults detected by this last chain segment may

be detected by the first 2 pairs.

Table 4.6: Results with/without Essential Vectors

$Circuit$	S@ Set	Without essential vectors				with essential vectors			
		$Tran.$ $Tests$	$Comp.$ $Tests$	$Time$ (s)	TFC $(\%)$	$Tran.$ $Tests$	$Comp.$ $Tests$	$Time$ (s)	TFC (%)
c1355	198	928	285	3.51	99.77	915	270	2.82	99.77
c1908	143	918	318	4.57	99.67	966	298	3.32	99.67
c3540	202	1222	515	25.43	96.27	1181	514	22.06	96.27
c5315	157	816	342	11.80	99.54	762	313	9.79	99.54
c6288	141	310	122	5.60	99.19	334	120	5.70	99.19
s344	31	135	63	0.37	100	207	64	0.37	100
s832	179	988	310	4.36	99.20	937	292	2.78	99.20
s1196	197	1004	362	5.24	99.97	1022	358	4.38	99.97
s1423	97	566	186	2.25	99.11	528	177	2.09	99.11
s5378	332	1672	722	35.73	98.40	1685	722	29.76	98.40
s35932	78	542	196	133.13	90.50	633	197	133.01	90.50
s38417	1207	5142	2682	1073.03	99.66	5208	2686	858.85	99.66

Table 4.6 presents the results of the weighted transition-graph algorithm with and without using essential vectors. In Table 4.6, column 2 gives the number of stuck-at vectors in the original STRATEGATE test set, followed by the results for our algorithm without using essential vectors. The final four columns show the results when essential vectors are used. For each approach, the number of transition vectors produced is shown first. Next, the number of compacted test vectors and its transition fault coverage are shown. Note that the compaction step achieves considerable reduction without losing fault coverage. The transition fault coverage achieved with or without essential vectors are the same, as indicated in column 6 and column 10 of the table; only the test set sizes are different in the two approaches. In most cases, the use of essential vectors yields smaller test sets. However, because this is a greedy heuristic, optimality is not guaranteed. The execution time with

essential vectors is also generally shorter due to the quick elimination of a large number of faults detected by the essential vectors. The extra computation needed in s38417 was due to the lack of trimming in the weighted transition pattern graph. Because our target was to show the proof of concept that stuck-at vectors can be chained through the proposed WTG, we did not explicitly target reduction in the execution time.

4.5.1 Experimental Results for ATE Repeat

Table 4.7: ATE Repeat vs. COM

CKT	Storage				TF COV		CPU Time			IMPROVEMENTS	
	COM		WT_GR		COM	WT_GR	COM	WT_GR		$Str.$	Time
	$s@$	$Tran$	3	4				3	4		
c1908	177	526	353	346	99.7	99.72	4.2	3.33	3.47	32.89	-34.22
c2670	167	396	185	184	78.6	79.26	4.3	9.25	9.99	53.28	6.57
c3540	247	732	410	419	82.9	87.62	6.8	19.45	20.11	43.99	-12.02
c5315	213	496	310	323	96.6	97.05	4.8	10.20	10.77	37.50	-25.00
c6288	47	180	130	118	99	98.54	3.6	3.72	3.62	27.77	-44.44
c7552	348	756	428	431	91	91.61	11	28.34	28.21	43.38	-13.23
s5378	391	980	455	464	86.6	87.51	5.3	37.66	37.76	53.57	7.14
s9234	630	1674	644	646	68.6	70.58	14.7	319.18	324.20	61.53	23.06
s13207	662	2004	681	679	80.5	82.29	27.4	292.89	295.30	66.02	32.06
s15850	641	1848	729	749	85	85.76	28	350.00	358.58	60.55	21.10
s35932	81	274	224	209	90	90.33	94.3	87.56	86.30	18.29	-63.50
s38417	1449	3854	1555	1547	89.9	91.19	116	1416.82	1406.83	59.65	19.30

In Table 4.7 we compare our results with results using COM, a commercial ATPG tool. We first tabulate the data for the storage required (STORAGE). Both the size of the stuck-at test set and the number of transition test vectors for COM are presented. These were generated using the dynamic compaction option of COM. Thus, for C1908 we need to store a total of 526 vectors. WT_GR used the stuck-at test set generated using COM without compaction. The next two columns show the transition test chain lengths obtained using the proposed algorithm, with chain length 3 and 4, respectively. Thus, for C1908 we need to store 353 vectors or 346

vectors depending on the chain length used in the algorithm. The storage improvement obtained using transition test chains and the ATE repeat option is shown in column 11, where chain length of 3 was assumed. Thus, for C1908, the storage improvement is calculated as 100* (526 - 353)/(526). Note the substantial reduction in all cases. The average reduction in scan memory requirement is 46.5%.

Columns 6 and 7 compare the transition fault coverage obtained by weighted transition graph (WT_GR) and COM. Note that there is no loss in fault coverage using WT_GR. Columns 8, 9 and 10 compares the CPU time required by COM and the two versions of our algorithm. For most of the circuits, COM is much faster. The last column of the table shows changes in the test application time. Recall that for a given transition test chain all but the first and last vectors are scanned in twice. Therefore, the test application time gain for C1908 is computed as 100*(526-2*353)/(526) = -34.22. This implies a 34.22% increase in test application time if transition test chains with ATE repeat are used. However, in a number of cases the test application time actually decreases by a significant amount. The average increase in test application time is 6.9%.

Next, we will present the results for reducing the extra test application time with exchange scan.

4.5.2 Experimental Results for Exchange Scan

Benefits of using the exchange scan, versus COM are reported in Table 4.8. Transition test chains were computed using our heuristic with chain length set to 3. The improvement in both test application time and test data volume are reported in column 4. Thus, for C1908, the test application time reduction is calculated as 100*(526 - 353)/(526) = 32.89%. We note that now there is a substantial reduction

Table 4.8: ATE Weighted Transition Pattern Graph Algorithm vs. COM

	STORAGE		IMPROVEMENT
$CIRCUIT$	COM	WT_GR	DATA,APPTIME(%)
c1908	526	353	32.89
c2670	396	185	53.28
c3540	732	410	43.99
c5315	496	310	37.50
c6288	180	130	27.77
c7552	756	428	43.38
s5378	980	455	53.57
s9234	1674	644	61.53
s13207	2004	681	66.02
s15850	1848	729	60.55
s35932	274	224	18.29
s38417	3854	1555	59.65

in both the scan memory requirement and the test application time, compared to COM. The average reduction in both test application time and data storage requirement is 46.5%.

Results from Table 4.8 and Table 4.7 are illustrated graphically in Figure 4.10. For each circuit, the data storage requirement and test application time are plotted for the conventional ATE, ATE repeat, and Exchange scan.

4.5.3 Experimental Results for Constrained ATPG

In Table 4.9, we compare the results of the weighted transition graph algorithm with and without considering the constraints on functionally untestable faults. The number of functionally untestable faults identified by our implication engine is presented in column 2, followed by the percentage of functionally untestable faults for each circuit. The next three columns show the transition fault coverage by the STRATEGATE stuck-at vectors: the total transition fault coverage, the functional

testable fault coverage and the overtesting fault ratio are given, respectively. Next, the results without considering the functionally untestable faults are shown. And the final three columns tabulate the results on the transition fault coverage while considering the constraint on functionally untestable faults.

For instance, for circuit s5378, our low-cost transition implication engine identified 3695 out of the total 15322 faults are functionally untestable. So the percentage of functionally untestable faults is 3695/15322=24.12%. In other words, only 75.88% (1-24.12%) of the faults can be detected in the functional mode.

When looking at the STRATEGATE stuck-at test set for s5378, we found that it can detect 92.96% of the transition faults. While most of the functionally testable faults can be detected (75.87% out of total 75.88%), the overtesting ratio for functionally untestable faults is 17.29%. If we ignore the overtesting factor and only target those functionally testable faults, our weighted transition graph algorithm can improve the total transition fault coverage to 94.16% but at the cost of the overtesting ratio of 18.29%. Finally, if we impose the constraint of minimizing the overtesting of functionally untestable faults in our graph algorithm, we can reduce the overtesting ratio to only 3.75% and still capture most of the functionally testable faults. Only 75.88%-74.86=1.02% of the functionally testable faults are missing. Note that for this Constrained ATPG, we do not include the original sequence of s@ vectors in our final test set, since the original order of vectors can potentially detect many functionally untestable faults. Results for other benchmark circuits can be explained in a similar manner.

Table 4.9: Results with/without Constraint

Ckt	Red fault	Red ratio (%)	Orig. S@ vectors			Without Constraint			Constrained ATPG		
			FC (%)	FFC (%)	OVT (%)	FC (%)	FFC (%)	OVT (%)	TFC (%)	FFC (%)	OVT (%)
s344	66	6.5	88.0	82.6	5.5	98.5	92.1	6.4	93.4	91.8	1.6
s832	96	4.3	72.5	69.9	2.5	94.3	90.8	3.5	91.0	90.5	0.5
s1196	5	0.1	86.2	86.1	0.1	94.7	94.6	0.1	91.3	91.3	0
s1423	387	9.4	94.4	86.5	7.9	98.1	89.9	8.3	88.6	83.3	4.7
s5378	3695	24.1	93.0	75.1	17.9	94.2	75.8	18.3	84.3	74.9	3.8
s35932	11255	11.2	86.6	85.3	1.4	89.6	87.9	1.7	89.1	87.4	1.7
s38417	32086	27.0	97.4	72.1	25.2	97.6	72.4	25.3	87.9	69.3	18.6

4.6 Summary

We presented effi cient techniques to reduce test data volume and test application time for transition faults. First, we propose a novel transition test chain formulation via a weighted transition pattern graph. Only s@ ATPG is needed to construct the necessary test chains for transition faults. By combining the proposed transition test chain and ATE repeat capability to reduce the test data volume by 46.5%, when compared with the conventional approach. The second technique that replaces the ATE repeat option with Exchange Scan improves both test data volume and test application time by 46.5%. In addition, we address the problem of yield loss due to incidental *overtesting* of functionally untestable transition faults, By formulating it into a constraint in our weighted pattern graph, we can effi ciently reduce the overtesting ratio. The average reduction on the overtesting ratio is 4.68%, with a maximum reduction of 14.5%.

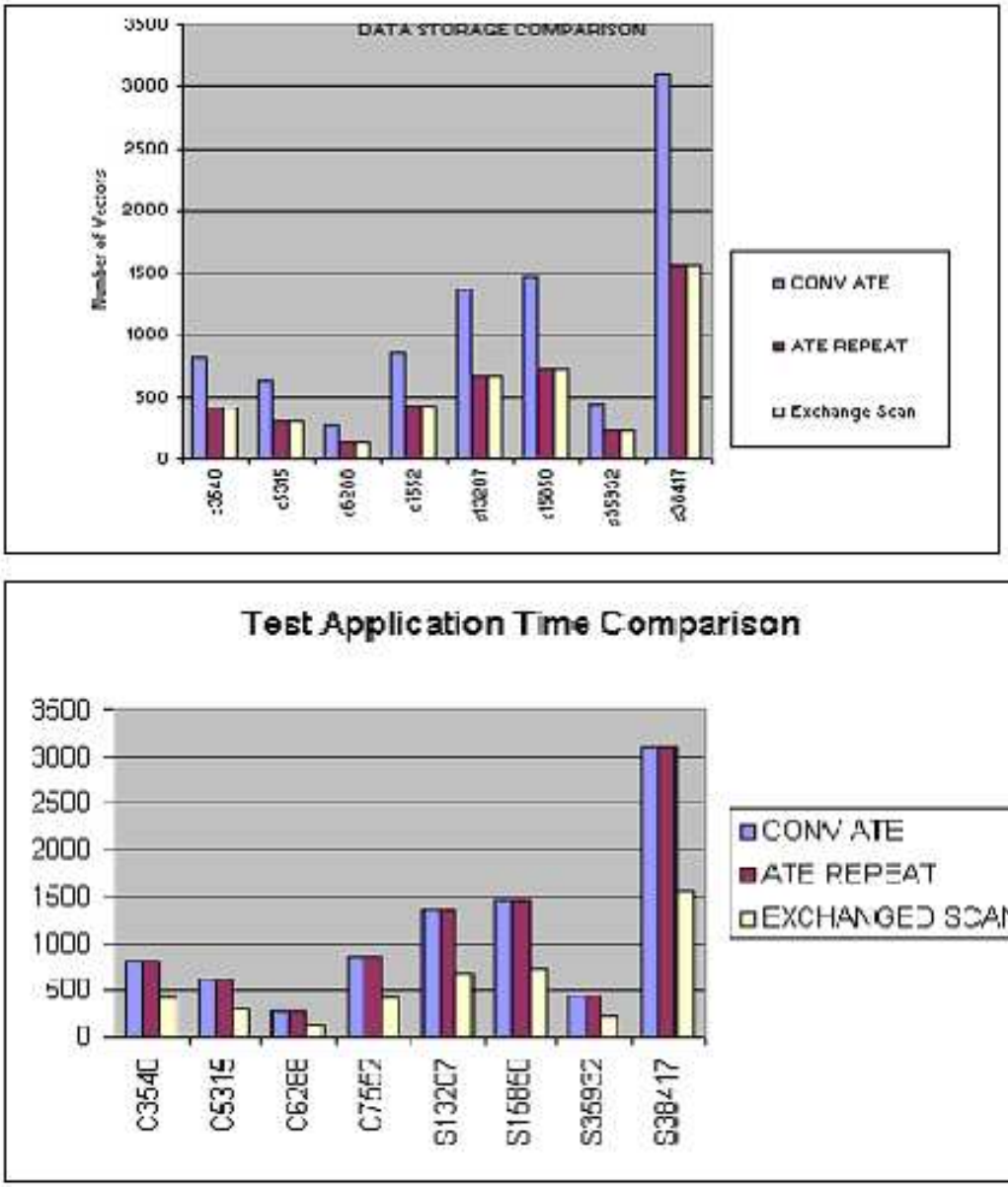


Figure 4.10: Graphical Experimental Results

Chapter 5

Hybrid Scan-based Delay Testing

5.1 Introduction

With ever decreasing geometry sizes and increasing clock speeds, ascertaining correct operation of digital circuits at desired speed is becoming a necessity rather than an option to maintain product quality level. In the past, testing circuit's performance was typically accomplished with functional test patterns. However, developing functional test patterns that attain satisfactory fault coverage is unacceptable for large scale designs due to the prohibitive development cost. Even if functional test patterns that can achieve high fault coverage are available, applying these test patterns at-speed for high speed chips requires very stringent timing accuracy, which can be provided by very expensive automatic test equipments (ATEs). The scan-based delay testing where test patterns are generated by an automatic test pattern generator (ATPG) on designs that involve scan chains is increasingly used as a cost effi cient alternative to the at-speed functional pattern approach to test large scale chips for performance-related failures [BRS+02, SBG+02]. Design-for-testability (DFT)-focused ATEs [COM00, ROB00], which are designed and developed to lower ATE cost by considering widely used DFT features of circuits under test (CUTs) such as full and partial scan are emerging as a strong trend in test industry.

In this chapter, we present a novel scan-based delay test approach that combines advantages of the skewed-load and broad-side approaches. In the hybrid approach, only a small set of selected scan flip-flops are controlled by the skewed-load approach and the rest of scan flip-flops are controlled by the broad-side approach. Hardware overhead to implement the proposed technique is substantially lower than that for the traditional broad-side approach and comparable to that for the traditional broad-side approach. No additional external pin is required to implement the proposed approach. The proposed hybrid approach can achieve higher coverage than the traditional broad-side approach. Sizes of test pattern sets generated by the hybrid approach are comparable to those of test pattern sets generated by the skewed-load approach. ATPG run times of the proposed hybrid approach are typically shorter than those of the traditional broad-side approach. Although the proposed technique is applicable to other delay fault models, in this chapter we focus only on transition delay fault model.

The rest of this chapter is organized as follows. Definitions and notations that are used in the rest of chapter are described in Section 5.2. Section 5.3 gives an in-depth comparison between Skewed-load and Broadside. The motivation and key idea of the proposed hybrid approach are described in Section 5.4. The selection criterion used to select the flip-flops to be controlled by the skewed-load approach is described in Section 5.5. The implementation of the circuit that generates the fast scan enable signal, which is used to drive flip-flops that are controlled by the skewed-load approach, from the slow scan enable signal is discussed in Section 5.6. Section 5.7 reports experimental results. Finally, Section 5.8 gives the conclusions.

5.2 Definitions and Notations

Figure 5.1 (a) describes the Huffman model of a sequential CUT, which employs full scan, has l primary inputs, $pi_1, pi_2, \ldots, pi_l$, m primary outputs, $po_1, po_2, \ldots, po_m$, n state inputs , $si_1, si_2, \ldots, si_n$, and also n state outputs, $so_1, so_2, \ldots, so_n$. Each pair of state output

so_i and state input si_i, where $i = 1, 2, \ldots, n$ are connected through a scan flip-flop D_i to constitute a feedback loop. The scan flip-flops are controlled by a scan enable signal to configure it into either its shift mode or normal mode. We assume that the scan chain is constructed with muxed scan type flip-flops. A muxed type scan flip-flops consists of a regular flip-flop and a multiplexer whose output is connected to the input of the regular flip-flop. The select input of the multiplexer selects between the normal data input and the scan input. When the normal data input is selected, the scan flip-flops is configured into its normal mode and when the scan input is selected, the scan flip-flop is configured into its shift mode.

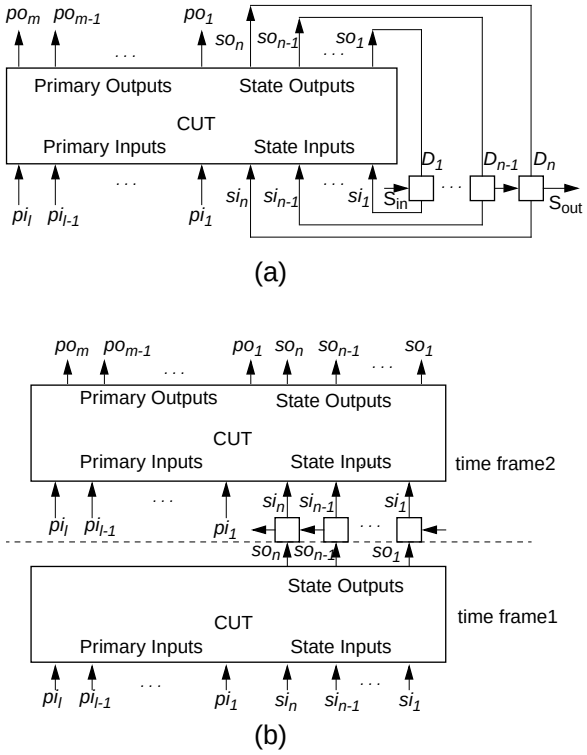


Figure 5.1: An Example Full-Scan Circuit (a) Original Circuit (b) Two Frame Version

When the circuit employ full scan, test vector pairs for transition delay faults can be

generated by running a combinational ATPG for a two time frame version of the circuit. A two time frame version of the original circuit shown in Figure 5.1 (a) is shown in Figure 5.1 (b) where the state outputs of the first time frame copy are connected to the state inputs of the second time frame copy, i.e., state outputs so_i, where $i = 1, 2, \ldots, n$, of the first time frame copy are connected to state inputs si_i of the second time frame copy.

The scan chain input of the scan chain is connected to the scan input of scan flip-flop D_1 and the scan output of D_1 is connected to the scan input of D_2, and so on, and finally the scan output of D_n is connected to the scan chain output. Hence, when the scan chain is in the shift mode, the value at state input si_i of the second time frame is the same as the value at state input si_{i-1} of the first time frame. Unlike state inputs, we assume that primary inputs are fully controllable, i.e., completely independent vectors can be applied to primary inputs at any two consecutive test cycles.

5.3 Skewed-load vs. Broadside

Although delay fault testing has been researched for years, most researches on delay fault testing have focused on combinational circuits. However, due to limited controllability of state inputs when standard scan is employed, applying these techniques to standard scan designs is not straightforward. Test procedures of the two traditional approaches to apply test vector pairs to standard scan designs are illustrated in the timing diagrams shown in Figure 5.2. In both skewed-load and broad-side approaches, the initialization vector of a test vector pair is first loaded into scan chain by n consecutive *scan shift operations*, where n is the number of scan flip-flops in the scan chain, in the same fashion as a stuck-at test vector is loaded into the scan chain. The last shift cycle when a test vector is fully loaded into the scan chain CUT, is referred as the *initialization cycle* (see Figure 5.2 (a) and (b)). The clock speed during scan shift operations is typically lower than the full system clock speed. The launch vector is applied after the CUT is stabilized from switching caused by

applying the initialization vector. The response to the launch vector is captured into scan flip-flops at the next clock cycle. Note that unlike the launch clock after the initialization clock, the capture clock is applied at full system clock speed after the launch clock.

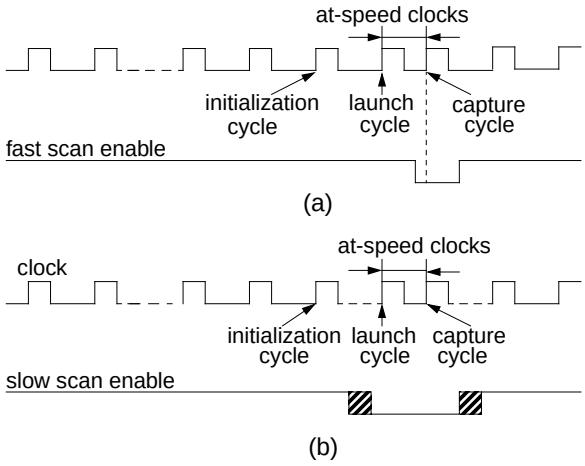


Figure 5.2: Scan-based Delay Test Diagram (a) Skewed-load Approach (b) Broad-side Approach

The second vector is derived from the first vector in both approaches. In the skewed-load approach [SP93], the second vector is obtained by shifting in the first vector (initialization vector), which is loaded into the scan chain, by one more scan flip-flop and scanning in a new value into the scan chain input. Note that the scan enable signal stays at logic high during the launch cycle in the timing diagram shown in Figure 5.2 (a). At the next clock cycle (capture cycle), the scan enable signal switches to logic low and the scan flip-flops in the scan chain are configured in their normal mode to capture the response to the scanned in test vector. Since the capture clock is applied at full system clock speed after the launch clock, the scan enable signal, which typically drives all scan flip-flops in the CUT, should also switch within the full system clock cycle. This requires the scan enable signal to be driven by a sophisticated buffer tree or strong clock buffer. Such design requirement is often too costly to meet. Furthermore, meeting such a strict timing required for the scan enable

signal may result in longer design time.

Since the second vector of each vector pair is obtained by shifting in the first vector by one more scan flip-flop, given a first vector, there are only two possible vectors for the second vector that differs only at the value for the first scan flip-flop whose scan input is connected to the scan chain input. This *shift dependency* restricts the number of combinations of test vector pairs to $2^n \times 2$ [SB91] in standard scan environment, where n is the number of scan flip-flops in the scan chain. If there is a transition delay fault that requires a 1 at state input si_{i-1} in an initialization vector and requires a 0 at state input si_i in the corresponding launch vector to be detected, then that fault is untestable by the skewed-load approach (assume that the scan chain is constructed by using only non-inverting outputs of scan flip-flops).

In the broad-side approach, the second vector is obtained from the circuit response to the first vector. Hence, the scan flip-flops are configured into the normal mode by lowering the enable signal before every launch cycle (see Figure 5.2 (a)). Since the launch clock following an initialization clock need not be an at-speed clock, the scan enable signal does not have to switch to logic low at full system clock speed between the initialization clock and the launch clock. Note that in the broad-side approach, launch vectors are applied when scan flip-flops are in their normal mode. In other words, the at-speed clocks, the capture clock after the launch, is applied to scan flip-flops while the scan flip-flops stays in their normal mode. Hence, the scan enable signal does not have to switch between the launch cycle and the capture cycle when clocks are applied at full system clock speed. Hence, the broad-side approach does not require at-speed transition of the scan enable signal and can be implemented with low hardware overhead.

Even though the broad-side approach is cheaper to implement than the skewed-load approach, fault coverage achieved by test pattern sets generated by the broad-side approach is typically lower than that achieved by test pattern sets generated by the skewed-load approach [SP94a]. Test pattern sets generated by the broad-side approach are typically larger

than those generated by the skewed-load approach [SBG+02]. In order to generate two-vector tests for the broad-side approach, an ATPG with sequential property that considers two full time frames is required. On the other hand, test patterns for the skewed-load approach can be generated by a combinational ATPG with little modification. Hence, higher test generation cost (longer test generation time) should be paid for the broad-side approach.

Since in the broad-side approach, the second vector is given by the circuit response to the first vector, unless the circuit can transition to all 2^n states, where n is the number of scan flip-flops, the number of possible vectors that can be applied as second vectors of test vector pairs is limited. Hence, if a state required to activate and propagate a fault is an invalid state, i.e., the state cannot be functionally justified, then the transition delay fault is untestable. Typically, in large circuits that have a large number of flip-flops, the number of reachable states is only a small fraction of 2^n states. Due to this reason, transition fault coverage for standard scan designs is often substantially lower than stuck-at fault coverage.

Enhanced scan testing [DS91] allows the application of any arbitrary vector pair to the combinational part of a sequential circuit. Hence, complete fault coverage can be attained. However, since this technique requires enhanced scan cells, which can hold two bits, the disadvantage of enhanced scan testing is high area overhead of enhanced scan cells. In this chapter, we refer to faults that are not testable under standard scan environment but testable under full enhanced scan environment as *dependency untestable faults*. Particularly, dependency untestable faults that are not testable due to shift dependency, which may exist when the skewed-load approach is used, are referred as *shift dependency untestable faults* and dependency untestable faults that are not testable due to function dependency, which may exist when the broad-side approach is used, are referred as *function dependency untestable faults*.

5.4 Key Idea

As described in Section 5.3, although the skewed-load approach has several advantages (higher fault coverage, smaller test sets, and ease of test generation) over the broad-side approach, the broad-side approach is the only choice of scan-based test method in many cases due to difficulty meeting design requirements of the skewed-load approach [SBG+02]. However, using the broad-side approach to avoid high hardware overhead will incur significantly higher test cost and result in lower test quality for most designs than the skewed-load approach.

The cost of test application is directly determined by the size of test set to be applied. As size and complexity of chips grow, size of test sets also tend to grow. Hence, generating compact test sets is a very important objective of test developers. Due to large test volume required to achieve satisfactory coverage, transition fault coverage is often compromised for acceptable test volume. In most compaction algorithms, don't cares in test vectors, which are not assigned binary values, play an important role in compacting test vectors. Test compaction techniques can be classified as dynamic [GR79, PRR91] and static compaction [GR79, CL95] according to when compaction of test vectors is performed. In dynamic compaction, which performed during test generation, don't cares, which are not specified in a vector generated to detect a fault, are specified to detect more faults. Test vectors that have many don't cares can be easily merged together by a static compaction technique, which is performed on pre-generated test patterns as a post-processing step, to reduce the number of patterns in the final test pattern set.

Figure 5.3 illustrates backtrace operations during test generation process to generate test vector pairs for delay faults. Suppose that we want to generate a test vector pair for the slow-to-rise (STR) fault at line l. Assume that state input si_i should be assigned a 1 to activate the fault and propagate the activated fault effect to observation point(s) (primary and scan outputs). If we use the broad-side approach, we may need to specify a large number of state inputs in time frame 1 to set si_i to a 1 in time frame 2. On the other hand,

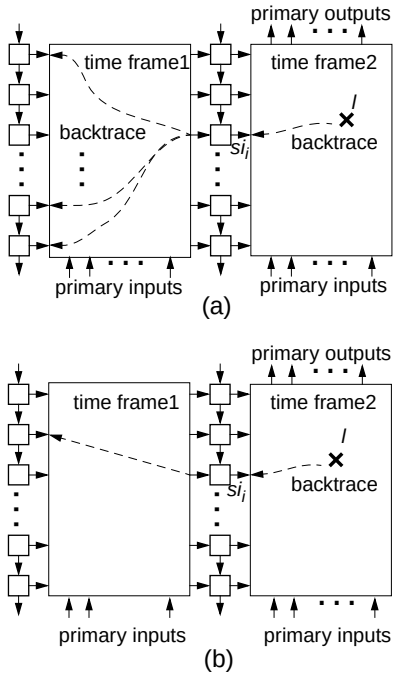


Figure 5.3: Assignment of State Inputs in Time Frame 2 (a) Broad-side Approach (b) Skewed-load Approach

if we use the skewed-load approach, the assigning a 1 to si_i in time frame 2 is achieved by setting only si_{i-1} to a 1 in time frame 1 (see Figure 5.3 (a)) and no backtrace is required in time frame 1. Hence, test patterns generated by the skewed-load approach typically have more don't cares, i.e. fewer specified bits, than those generated by the broad-side approach. This implies that test patterns generated by the skewed-load approach have more room for compaction. Indeed, sizes of test sets generated for the skewed-load approach are typically smaller than those of test sets generated by the broad-side approach [SBG+02].

Switching the scan enable signal that drives all scan flip-flops in a large circuit within one full system clock cycle requires a strong clock buffer or a buffer tree. But a scan enable signal that drives only a small number of flip-flops (say, 100 flip-flops) can switch in one full system clock cycle without any strong buffer or buffer tree. If only a small set

of state inputs of the circuit require a large number of inputs to be specified in time frame 1 to be set to binary values in time frame 2 (and the rest of state inputs can be set to binary values in time frame 2 by specifying only small numbers of inputs in the time frame 1), then controlling only those state inputs by the skewed-load approach and controlling the rest of state inputs by the broad-side approach will generate test patterns that have many don't cares. Since we drive only the small set of scan flip-flops by a separate scan enable signal, the separate scan enable signal can switch in one system clock cycle without being driven by a strong buffer or a buffer tree. The scan enable signal that drives the rest of scan flip-flops that are controlled by the broad-side approach signal need not switch at-speed. In consequence, by using our hybrid approach, we can take advantages of the skewed-load approach without having to meet costly design requirement required by the traditional skewed-load approach. In the rest of this chapter, scan flip-flops that are controlled by the skewed-load approach are referred to as *skewed-load flip-flops* and flip-flops that are controlled by the broad-side approach are referred to as *broad-side flip-flops*.

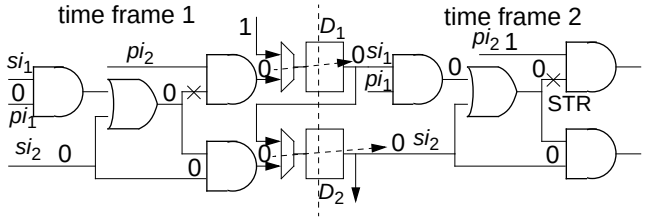


Figure 5.4: Function Dependency Untestable Fault

Another advantage of the proposed hybrid approach is that some faults that are not testable by the traditional broad-side approach due to function dependency can become testable by the hybrid approach. For example, consider the STR fault at line l in the circuit shown in Figure 5.4. In order to initialize the STR fault, line l should be set to a 0 in time frame 1. Assume that when the select input of multiplexer of a scan flip-flop, which selects between the scan input and the normal data input, is set to a 0, then the scan flip-flop is configured into normal mode and the normal data input of the multiplexer is selected.

Therefore, the 0 at line l in time frame 1 propagates to state input si_1 in time frame 2 through the AND gate at the next cycle. The 0 at input si_1 in turn propagates to line l in time frame 2. However, in order to activate the STR fault in time frame 2, line l should be set to a 1. Hence, the STR fault is not testable when the broad-side approach is used. Now, assume that we control scan flip-flop D_1 by the skewed-load approach, i.e., the select input of multiplexer of D_1 is set to a 1 to select the scan input. If the scan input is assigned a 1, then the 1 at the scan input propagates to input si_1 in time frame 2 rather than the 0 at the normal data input. When both primary inputs pi_1 and pi_2 are assigned 1's in time frame 2, the STR fault at line l can be detected at scan output so_1.

Finally, the proposed approach can speed up ATPG process. Let us revisit Figure 5.3. Again let us assume that state input si_i should be set to a binary value v, where $v = 0$ or 1, in time 2 to activate a target fault and propagate the fault effect. When the broad-side approach is used, this need to assign a large number of inputs (primary or state inputs) to binary values in time frame 1. If PODEM [GOE81] algorithm is used to generate test patterns, this will be achieved by a series of backtrace operations in time frame 1. Whether setting si_i to a 1 in time frame 2 leads to a solution to generate a test vector pair for the target fault at line l cannot be determined until state si_i is assigned a v in time frame 2. If setting si_i to a 1 in time frame 2 does not leads to a solution, it is found only after a lot of CPU time has been already spent on backtrace operations and following forward implications. On the other hand, if the skewed-load approach is used, setting state input si_i to binary value v can be achieved by simply setting state input si_{i-1} in time frame 1 and forward implying the assignment at state input si_{i-1}. This requires no backtrace operations in time frame 1. If setting state input si_i to v in time frame 2 does not lead to a solution, then the ATPG can immediately backtrack without wasting time on backtrace operations and following forward implications.

5.5 Selecting Skewed-load Flip-flops

We use controllability measures, which are similar to SCOAP [GT80], as a criterion to select scan flip-flops to be controlled by the skewed-load approach. We select M state outputs, where M is determined from the number of flip-flops the scan enable signal can drive without being driven by a strong buffer or buffer tree, that have highest 0 or 1 controllability measure and control the selected M scan flip-flops by the skewed-load approach. The 0 (1) controllability measure of line l, $C_0(l)$ ($C_1(l)$), is the minimum number of primary and state inputs to be specified to set line l to a 0 (1). Controllability measures of line l are defined as:

$$C_v(l) = \begin{cases} 1 & \text{if } l \text{ is a primary input} \\ 1 & \text{if } l \text{ is a state input} \\ \min_{l_a}\{C_c(l_a)\} & \text{if } v = c \oplus i \\ \sum_{l_a} C_{\bar{c}}(l_a) & \text{otherwise,} \end{cases} \tag{5.1}$$

where l_a and l are respectively inputs and outputs of a gate with controlling value c and inversion i. If a value c, when applied to an input of a gate, determines the value at the output of the gate regardless of the values applied to its other inputs, then the value is said to be the *controlling value* of the gate. Note that the controllability measures are calculated on one time frame version of the circuit.

If we select the skewed-load flip-flops considering only controllability measures of corresponding state outputs, then it may introduce shift dependency untestable faults. If a scan flip-flop drives the fanout cone that its immediate predecessor scan flip-flops also drives, then controlling the scan flip-flop by the skewed-load approach may introduce shift dependency untestable faults (see Section 5.3). Hence when we select flip-flops to be controlled by the skewed-load, we should also consider shift dependency relation between adjacent scan flip-flops to avoid introducing shift dependency untestable faults.

If state input si_i drives no fanout cones that its immediate predecessor state input si_{i-1} drives, then si_i is said to be *independent* and controlling such state inputs si_i by the

skewed-load approach does not introduce any shift dependency untestable faults [SAV92a]. When we use the skewed-load approach for all scan flip-flops in the circuit, it is enough to consider only one time frame to identify independent state inputs. However, in the hybrid approach, some of scan flip-flops are controlled by the broad-side approach. Hence, in order to guarantee no decrease in fault coverage due to introduction of shift dependency untestable faults, we have to consider two time frames to identify independent state inputs. However, according to our extensive experiments, decrease in fault coverage due to selecting state inputs that are independent in one time frame but not independent in two time frames to be controlled by the skewed-load approach is negligible. Since there are very few state inputs that are independent in two time frames, in the experiments whose results are shown in Section 5.7, we considered only one time frame during the independent state input identification process in order to select more skewed-load scan flip-flops. Figure 5.5 shows a pseudo code for the algorithm to select skewed-load scan flip-flops.

Selecting skewed-load scan flip-flops
compute $C1(l)$ and $C0(l)$ for every line l in the circuit;
for every state output so_i;
cost(so_i) = max($C1(so_i)$, $C0(so_i)$);
for every state input si_i;
    compute shift dependency relation with si_{i-1};
end for;
sort state outputs so_i by cost(so_i), in non-increasing order
j = 1;
for i = 1 to n; /* n is the number of state inputs in thecircuit */
    if si_i is indepdent of si_{i-1}, then
    select corresponding flip-flop D_i as a skewed-load flip-flop
    and increment j by 1;
    if ($j > M$) then exit the for loop;
end for;

Figure 5.5: Pseudo Code for Skewed-load Flip-flop Selection Algorithm

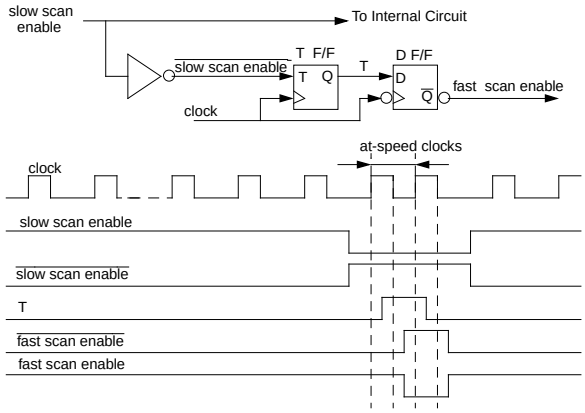


Figure 5.6: Fast Scan Enable Signal Generator

5.6 Generating Fast Scan Enable Signal

The *fast scan enable signal*, which drives skewed-load flip-flops, is internally generated from the *slow scan enable signal*, which comes from an external pin and drives broad-side flip-flop, and the system clock signal. Hence, no additional external pin is required for the fast scan enable signal. Figure 5.6 shows a schematic for the fast scan enable signal generator and waveforms of involved signals. As the schematic shows, the fast scan enable signal generator can be implemented with very little hardware. Since the fast scan enable signal is synchronized with the system clock, which is accessible from the clock pin of any flip-flop, the fast scan enable signal generator can be located anywhere in the chip to minimize routing from the fast scan enable signal generator to skewed-load flip-flops that are driven by the fast scan enable signal thereby reducing possible skew problem of the fast scan enable signal and routing overhead in congested chips. Furthermore, since the fast scan enable signal is synchronized, it is also possible to use multiple fast scan enable signal generators and multiple fast scan enable signals each of which drive different sets of skewed flip-flops. When there are many state outputs that have high controllability measures and hence all the scan flip-flops that have high controllability measures cannot be driven by a

single fast scan enable signal without a strong buffer, then multiple fast scan enable signals can be used to reduce load capacitance of fast scan enable signals by making each fast scan enable signal drive only small set of scan flip-flops. Since a fast scan enable generator is comprised of only two flip-flops, even hardware overhead of tens of fast scan enable generators will be significantly lower than a clock buffer or buffer tree.

5.6.1 Multiple Fast Scan Enable Signals

Some circuits have very small numbers of independent state inputs even if we consider one time frame for identification of independent state inputs. If such circuits have large numbers of state inputs that have high controllability measures, then reduction in test pattern set sizes and enhancement of fault coverage that are obtained by using the hybrid approach may not be significant. If we use multiple fast scan enable signals, then we can obtain large reduction in test set sizes and improvement in fault coverage even for such circuits. If we partition an entire delay test application task into several sub-phases and control a different subset of scan flip-flops by fast scan enable signals at each sub-phase, then faults that are not detected due to shift dependency in a sub-phase can be detected in other sub-phases.

Figure 5.7 shows a circuit to control multiple fast scan enable signals. The test register is used to control which sets of skewed-load flip-flops are drive by fast scan enable signals. In each sub-phase, the test register will be loaded with different values according to the sets of scan flip-flops that will be controlled by the skewed-load approach. Assume that the test register is loaded with 0100 in sub-phase 1. When the test register is loaded with 0100, only the set of scan flip-flops that are driven by scan_en2 will be controlled by the skewed-load approach and scan flip-flops controlled by other scan enable signals, scan_en1, scan_en3, and scan_en4, will be controlled by the broad-side approach. If scan flip-flops that are driven by scan_en2 drive state inputs that are not independent, then there may exist shift dependency untestable faults that are not detected in sub-phase 1. However, if the test flip-flop that controls scan_en2 is loaded with a 0 in sub-phase 2, then the shift dependency

untestable faults that are not detected in sub-phase 1 can be detected in sub-phase 2.

Decrease in fault coverage due to shift dependency can be further minimized by input separation techniques [SP93] and/or scan path test point insertion technique [WC03].

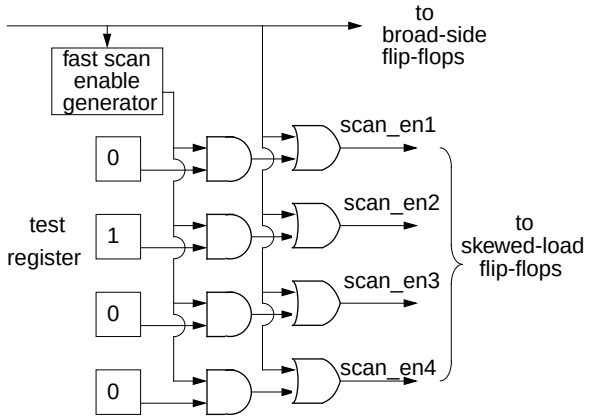


Figure 5.7: Multiple Fast Scan Enable Signal Control Circuit

5.7 Experimental Results

Table 5.1 shows experimental results for full scan versions of ISCAS 89 benchmark circuits. The experiments were conducted on a SUN Microsystem's Ultra 1 with 1 Giga bytes of memory. The column *# FFs* shows the number of scan flip-flops in each benchmark circuit. Results obtained by using the skewed-load, the broad-side, and the hybrid approach are compared for fault coverage (columns *% FC*), numbers of test vectors generated (columns *# Vec.*), and ATPG run times (columns *time*). Columns under the heading *Skewed-load* are results for the traditional skewed-load approach, columns under the heading *Broad-side* are results for the traditional broad-side approach, and columns under the heading *Hybrid* are results for the proposed hybrid delay scan approach. We used only one fast scan enable signal for all benchmark circuits and used M (the maximum number of skewed-load flip-flops) as 10% of total number of scan flip-flops in the circuit. However, only 1 or 2 flip-flops

Table 5.1: Experimental Results

CKT		*Skewed-load*			*Broad-side*			*Hybrid*			
Name	*# FFs*	*% FC*	*# Vec.*	*times*	*% FC*	*# Vec.*	*time*	*# SFFs*	*% FC*	*# Vec.*	*times*
s208	8	83.43	41	0.13s	69.23	28	0.29s	1	73.96	33	0.13s
s298	14	78.79	30	0.13s	79.17	30	0.39s	2	87.12	48	0.22s
s344	15	92.13	37	0.15s	92.88	36	0.35s	1	96.63	46	0.18s
s349	15	92.31	38	0.15s	93.04	42	0.42s	1	96.70	48	0.14s
s386	6	88.44	76	0.25s	66.33	47	0.89s	1	80.27	63	0.72s
s420	16	82.35	80	0.59s	68.24	53	1.32s	1	70.00	56	0.60s
s444	21	87.70	38	0.22s	77.01	42	1.32s	3	79.14	47	1.26s
s510	6	86.85	80	0.60s	84.13	79	2.26s	1	93.88	88	1.25s
s526	21	83.30	70	0.46s	60.89	58	2.11s	3	68.92	70	2.05s
s641	19	99.49	57	0.60s	95.67	83	9.33s	2	96.44	78	16.46s
s713	19	99.56	66	0.79s	94.32	91	17.20s	2	94.98	76	38.11s
s820	5	84.62	131	1.67s	77.49	133	10.46s	1	85.04	138	21.05s
s832	5	84.52	141	1.70s	77.13	135	11.43s	1	84.94	148	21.89s
s838	32	81.71	152	3.08s	67.55	104	6.73s	1	68.44	108	2.93s
s953	29	91.94	133	2.30s	92.43	134	6.49s	3	95.85	135	6.36s
s1196	18	99.91	227	4.72s	99.81	237	8.06s	2	99.81	217	2.36s
s1238	36	99.91	232	7.58s	99.72	233	12.10s	2	99.72	231	5.61s
s1423	74	95.41	113	12.44s	87.73	134	9m23s	8	88.48	129	14m7s
s1488	6	76.22	145	4.31s	85.59	151	24.13s	1	91.80	171	21.8s
s1494	6	75.88	145	4.25s	85.51	147	24.00s	1	91.81	170	22.45s
s5378	179	92.22	336	47.34s	93.02	365	9m2s	18	94.76	349	22m40s
s9234	228	91.89	605	12m50s	83.01	643	83m21s	23	84.98	598	126m27s
s13207	669	88.27	654	8m48s	77.74	620	24m48s	67	89.52	803	17m5s
s15850	597	92.53	595	22m6s	66.17	508	113m16s	60	72.56	524	166m1s
s35932	1728	100.0	125	7m11s	93.80	131	11m58s	173	99.58	175	6m40s
s38417	1636	98.57	1705	136m26s	96.80	1566	226m23s	164	96.84	1669	505m33s
s38584	1452	92.48	1097	82m15s	90.31	1463	759m47s	146	93.20	1528	898m31s

are selected as skewed-load flip-flops for many circuits since those circuits have very small numbers of independent inputs.

For all benchmark circuits, except for s1196 and s1238, fault coverage achieved by test pattern sets generated by our hybrid approach is higher than that achieved by test pattern sets generated by the broad-side approach. For example, for s13207, broad-side transition fault coverage is only 77.74% while the hybrid approach achieves as high as 89.52%. The average fault coverage improvement over all the ISCAS89 bench marks is 4.47% , with the highest improvement as 13.94%. It is notable that the hybrid approach test pattern sets achieve higher fault coverage than the skewed-load approach test pattern sets for s510, s820, s832, and s13207 for which the broad-side approach test pattern sets significantly lower fault coverage. This implies that when carefully designed, the hybrid approach can achieve even higher fault coverage than the skewed-load approach, which requires very high hardware overhead.

For some circuits, the hybrid approach test pattern sets are larger than the broad-side approach test pattern sets. However for those circuits, the hybrid approach test pattern sets substantially achieve higher fault coverage. For most circuits for which the hybrid and broad-side achieve similar fault coverage, hybrid approach test pattern sets are smaller than broad-side test pattern sets. The only exception is s38417 for which the hybrid approach test patterns attains similar fault coverage to broad-side approach test patterns. But the hybrid test set for s38417 has about 100 more patterns that the broad-side test set. We believe that this is due to inaccuracy in controllability measures as the criterion to select skewed-load flip-flops. Note that the 1 (0) controllability measure at a line reflect the minimum number of inputs to be specified to set the line to 1 (0). However, due to conflict with other necessary assignments during test generation processes, setting a line to a binary value by specifying only minimum number of inputs may not be possible. If state output so_i has a very high 1 controllability measure $C_1(so_i)$ but very low 0 controllability measure $C_0(so_i)$, then the corresponding scan flip-flop D_i will likely be selected as a skewed-load flip-flop since the

cost of so_i, $cost(so_i) = max\{C_0(so_i), C_1(so_i)\}$, is very high (see Section 5.5). However, if so_i is required to be assigned 0's in most test patterns, selecting scan flip-flop D_i as the skewed-load flip-flop will not help generate compact test pattern sets. A new cost function that can correct above limitations of the current cost function is under investigation.

The ATPG run time of the hybrid approach is comparable with that of the broad-side approach for most circuits except a few circuits such as s13207, s5378 and s38417. For s13207, the ATPG run time of the hybrid approach is substantially shorter than that of the broad-side approach and for 5378 and s38417, ATPG run time of the hybrid approach is substantially longer than that of the broad-side approach.

5.8 Summary

In this chapter, a novel scan-based delay test approach, referred as the hybrid delay scan, has been proposed. The proposed method combines advantages of skewed-load and broad-side approaches and can achieve higher delay fault coverage than the broad-side approach. By selecting only a small fraction of the state inputs as the skewed-load flip-flops, we avoid the costly design requirement in skewed-load approach due to the fast scan enable signal that must switch in a full system clock cycle.

Our experimental results show that for all the ISCAS 89 Benchmarks, the transition delay fault coverage achieved by hybrid approach is higher than or equal to that achieved by broadside load approach, with an average improvement of 4.47%. Due to limitation of the current cost function that is used as the criterion to selection skewed-load flip-flops, reduction in test pattern sets is not spectacular. sets. A new cost function that improves the current cost function is under investigation.

Some circuits have very small numbers of independent state inputs even if we consider one time frame for identification of independent state inputs. If such circuits have large numbers of state inputs that have high controllability measures, then reduction in test

pattern set sizes and enhancement of fault coverage that are obtained by using the hybrid approach may not be significant. If we use multiple fast scan enable signals, then we believe that we can obtain large reduction in test set sizes and improvement in fault coverage even for such circuits. If we partition an entire delay test application task into several sub-phases and control a different subset of scan flip-flops by fast scan enable signals at each sub-phase, then faults that are not detected due to shift dependency in a sub-phase can be detected in other sub-phases. We are currently investigating algorithms to partition skewed-load flip-flops into several subsets to further enhance fault coverage.

Chapter 6

Constrained ATPG for Broadside Transition Testing

6.1 Previous Work

The stuck-at fault model [ELD59] is insufficient for catching speed-related failures, as more chips are now more vulnerable to such failures due to higher clock rate, shrinking geometries, longer wires, increasing metal density, etc. The three most prevalent fault models for delay testings are: *transition fault* [WLRI87], *path delay fault* [SMI85], and *segment delay fault* [HPA96]. Among them, the transition fault model are most widely used in the industry due to its simplicity and similarity to the stuck-at fault model. In this chapter we only target at the transition fault model.

In general, (non-scan) functional testing can be impractical for larger circuits in that large test sets may be required to achieve a desirable fault coverage. As a result, at-speed AC scan testing has been widely used in the industry to detect delay-induced defects. Compared to functional testing, scan-based testing for delay faults can decrease the overall ATPG complexity and cost, since both controllability and observability on the flip-flops are enhanced. Nevertheless, the drawback of scan-based delay tests lies in two folds: hardware

overhead and potential yield loss. In [REA01], the author reported that scan-based testing may fail a chip due to the delay faults that do not affect the normal operation, and thus it is unnecessary to target those functionally unsensitizable faults. In other words, we want to avoid failing a chip due to a signal transition/propagation that was not intended to occur in the functional mode. Moreover, a scan test pattern, though derived from targeting functionally testable transition faults, can incidentally detect some functionally untestable transition faults if the starting state is an unreachable state.

Several papers [RM01, REA01, MaLB00] have discussed the relationship between functional testing and scan-based testing. However, from our knowledge, currently there is no quantitative analysis on functional untestable transition faults and scanning testing. In this chapter, we describe a novel constrained ATPG algorithm for transition faults. Two main contributions of our work are: (1) the constrained ATPG only targets at the functionally testable transition faults and *minimizes* detection of any identified functionally untestable transition faults; (2) the constrained ATPG can identify more functionally untestable transition faults than the conventional transition ATPG tools. The first contribution (the constrained ATPG) enables us to derive transition vectors that avoid illegal starting states, while the second contribution helps us to maximize the state space that we need to avoid. Because we want to avoid launching and propagating transitions in the circuit that are not possible in the functional mode, a direct benefit of our method is the reduction of yield loss due to overtesting of these functionally untestable transitions. Our experimental results showed that significantly more functionally untestable transition faults can be avoided in the final test set.

The rest of the chapter is organized as follows. Section 5.1 gives an overview of the three different scan-based transition test application techniques and explains the motivation of this work. Section 6.3 presents an implication engine we developed to identify a subset of the functionally untestable transition faults. Section 6.4 proposes a new constrained ATPG algorithm targeting at only functionally testable faults and simultaneously avoiding

the functionally untestable transition faults. Section 6.5 reports our constrained ATPG results, and compares them with a conventional transition ATPG engine. Finally, Section 6.6 concludes the chapter.

6.2 Background and Motivation

In ASIC area, transition tests are normally applied in two different ways: **Skewed-load** [SAV92a] and **Broadside** [SP94b].

For skewed-load transition testing (*also called last shift*) [SAV92a, SAV92b], an N-bit vector is loaded by shifting in the first N-1 bits, where N is the scan-chain length. The last shift clock is used to launch the transition. This is followed by a quick capture. For skewed-load testing, only one vector is stored for each transition pattern in tester scan memory; the second vector is a shifted version of the stored vector. Therefore, skewed-load testing is constrained by the correlation of the bits in the test pattern based on scan chain ordering. Figure 6.1 shows a simple example where the *slow-to-fall* on line d is untestable in Skewed-load testing. To detect the *slow-to-fall* fault on line d, we need to set

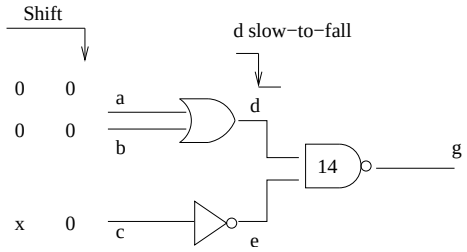
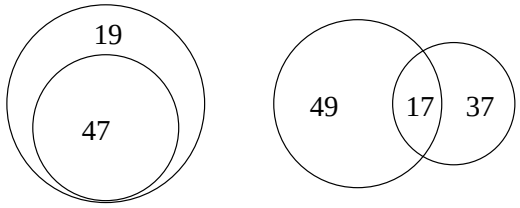


Figure 6.1: Untestable Fault in Skewed-load

the second vector V2=000 to detect the *d s-a-1*, therefore the initial vector must be 00X, the previously shifted version V2. However, this V1, 00X, is unable to initialize the line *d* to logic 1. Since 000 is the only vector that can detect the *d s-a-1* fault, thus *d slow-to-fall* is untestable in skewed-load testing. Based on this observation, skewed-load may miss some functionally testable faults because of the data dependency between the two vectors, resulting in *undertesting* of the functionally testable faults.

For broadside transition testing (*also called functional justification*) [SP94b], the first vector is scanned in and the second vector is derived as the circuit response to the first one. For broadside testing, after the first vector is scanned in and applied to the circuit, two clock cycles need to be pulsed: the first to launch the transition and the second to capture the circuit response. PI/PO changes would be made simultaneously with the first clock pulse if necessary [SBG+02]. Because it requires neither hold-scan design nor skewed shifting, this has been widely applied for transition testing. In this chapter, we will consider the Broadside model only.

Both Skewed-load and Broadside can suffer from yield loss due to a chip failing on detection of functionally untestable transition faults. Figure 6.2 shows an example of overtesting problem using traditional ATPG on ISCAS benchmark circuit s344. we can derive the following observations from figure 6.2:



(a)Broadside vs. Funct. (b)Skewed−load vs. Funct.

Figure 6.2: Untestable faults for S344

1. In part (a) of the figure, it shows that the set of 47 untestable faults by Broadside testing is a subset of the 66 functionally untestable faults.
2. In part (a) of the figure, Broadside testing will incidentally detect 19 faults that would be untestable in functional mode.
3. In part (b) of the figure, 54 transition faults cannot be tested in the skewed-load testing mode. Our results show that only 17 out these 54 faults are truly functionally

untestable; in other words, 37 (i.e., 54-17) functionally testable faults will be missed by skewed-load testing.

4. In part (b) of the fi gure, Skewed-load testing will incidentally detect 49 functionally untestable faults.

In general, we can relate the functional test and the various scan-based tests in terms of their untestable transition faults as depicted in Figure 6.3. In this fi gure, the circle at the

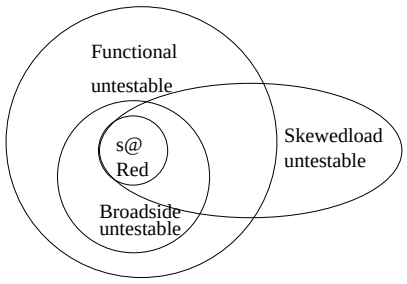


Figure 6.3: Functional testing vs. Scan-based testing

center depicts the set of redundant stuck-at fault set in the circuit, while each of the outer circle/oval represents the fault set that **cannot** be detected by a particular test method, Based on this relationship, some observations can be made:

1. For every redundant stuck-at fault in the circuit, there must be at least one corresponding functionally untestable transition fault, which is clearly untestable by any test method.
2. All scan-based test methods will incidentally detect some functionally untestable transition faults, because either the states they scan in may be functionally unreachable or the state combination is not functionally possible.
3. If a transition fault is untestable by the Broadside model, it will be defi nitely untestable in the function mode. Conversely, every functionally testable transition fault will be detectable under broadside testing.

4. Skewed-load can potentially miss some functionally testable faults due to the correlation between the vector and its shifted version.
5. Some of the untestable transition faults in Broadside may be detectable in skewed-load test, and vice versa.

Another reason we want to restrict our ATPG starting from reachable states is that we might exercise a false critical path, which is otherwise unexercisable in the functional operations.

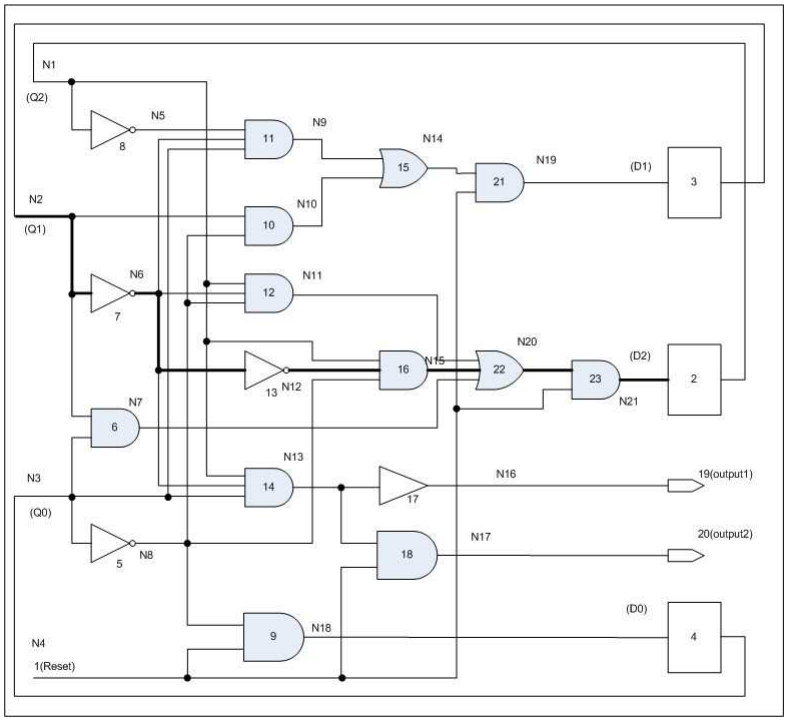


Figure 6.4: Modulo-6 counter

Figure 6.4 gives us an example for a modulo-6 counter, which counts from 000 to 101. Therefore, the other two higher values (110 and 111) will be unreachable states. We then exhaust all the input test vectors using a delay simulation. If we assume the delay on each gate of the circuit is always 1, we can observe that:

1. In function mode, the length of longest critical path is 4 i.e. the path of G5, G12, G22, G23.

2. But a path of length 5 (G7,G13, G16, G22, G23) could be exercised in scan testing by the test vector (1110)

We validated our gate-level delay simulation result with the spice simulator. The spice simulation is based on the TSMC 0.25um process CMOS model and the input frequency is set to be 20MHZ. We measure the propagation delay between the PI and OP along the longest path, when the signals cross 2.5 volts.

Table 6.1: Regular Counting Transitions

Transition	$LongestPathDelay$
$0000 \rightarrow 1001$	1.096ns
$1001 \rightarrow 1010$	0.615ns
$1010 \rightarrow 1011$	1.249ns
$1011 \rightarrow 1100$	0.623ns
$1100 \rightarrow 1101$	1.788ns

Table 6.2: Reset Transitions

Transition	$LongestPathDelay$
$1001 \rightarrow 0000$	0.204ns
$1010 \rightarrow 0000$	0.204ns
$1011 \rightarrow 0000$	0.203ns
$1100 \rightarrow 0000$	0.200ns
$1101 \rightarrow 0000$	0.312ns

Table 6.3: Illegal Transitions

Transition	$LongestPathDelay$
$1101 \rightarrow 1110$	2.232ns
$1110 \rightarrow 1111$	1.245ns
$1111 \rightarrow 0000$	2.207ns

From the spice simulation results, we can see that the longest critical path delay in the function mode is 1.788ns, while the longest path delay starting from illegal states could

be as long as 2.232ns. This result is in line with our previous gate-level delay simulation results. So it is import to prevent the ATPG from searching into illegal states.

Furthermore, even if we remove all the identified functionally untestable faults from the targeting list in our traditional ATPG, we may still incidentally detect some of them. We illustrates this scenario in the figure 6.5,which is a fragment of sequential circuit.

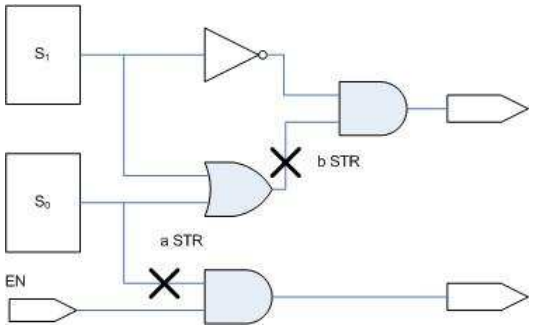


Figure 6.5: Example of Overtesting

Suppose the State S_1S_0=01 is an unreachable state in the sequential circuit. Then, the *slow-to-rise* transition fault on node b would be a functionally untestable fault, because S_1S_0 has to be set to 01 in oder to launch the transition on node b and propagate the fault effect to outputs. Therefore, the transition fault *b slow-to-rise* will NOT impair the functional performance.

But if we look at another functionally untestable faults *a slow-to-rise*, we can notice that the vector pair ($S_1S_0E_n$=X0X, $S_1S_0E_n$=X11) would be a good test for it. If we fill the unspecified bits randomly, we could have the test vector pair (000, 011) which will detect both *a slow-to-rise* and *b slow-to-rise*, although *b slow-to-rise* is actually functionally untestable and should NOT be detected. Therefore, in terms of avoid overtesting problem, vector pair (100, 111) would be a better choice that detects only *a slow-to-rise* but NOT *b slow-to-rise*. Similar idea can be also extended beyond transition fault model to path delay fault model.

6.3 Functionally Untestable Faults Identification

In general, functionally untestable transition fault identification in sequential circuits is of the same complexity as sequential ATPG, which is of exponential complexity in terms of the size of the circuit. In this section, we describe a novel untestable transition fault identification method by combining a transition fault implication engine and Broadside ATPG.

In [HSI02], a method for identifying untestable stuck-at faults in sequential circuits by maximizing local conflicting value assignments has been proposed. The technique first quickly computes a large number of logic implications across multiple time-frames and stores them in an implication graph. Then the algorithm identifies impossible combinations of value assignment locally around each gate in the circuit and those redundant stuck-at faults requiring such impossibilities.

For identifying functionally untestable transition faults, in addition to searching for the impossibilities locally around each gate, we also check the excitability of the initial value in the previous time frame. Thus, the implication engine in [HSI02] can be extended to quickly identify a large set of untestable transition faults in the circuit.

Although the transition fault implication engine helps us in identifying a large number of untestable transition faults in the circuit, it may be incomplete (i.e., not all untestable transition faults are identified). To avoid the high cost of calling a functional-mode sequential ATPG to identify the other untestable transition faults, we use a two-time-frame Broadside ATPG instead. As we discussed before, if a transition fault is untestable in Broadside testing, then it is definitely untestable in the function mode as well. So the transition fault implication engine and broadside ATPG can be combined to estimate the total number of functionally untestable transition faults. This saves us from having to invoke a full sequential ATPG. Figure 6.3

gives us a graphical illustration. In this figure, the outer rectangle represents the to-

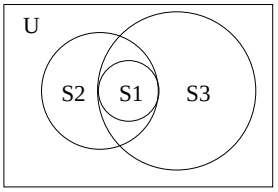


Figure 6.6: Approximation of Functionally Untestable Transition Faults.

tal number of functionally untestable transition faults in the circuit, and region S1 contains the redundant *stuck-at* faults identified by stuck-at fault implication-based method [HSI02]; region S2 contains the untestable transition faults by our new transition fault implication engine and S3 is the set of untestable transition faults identified by the Broadside ATPG. Note that the new implication-based method may identify some functionally untestable faults that ATPG misses, and vice versa. The union of S1, S2, and S3 can give us a close approximation of the functionally untestable transition faults within the circuit.

6.4 Constrained ATPG For Broadside Testing

In this section, we describe how we formulate the illegal states as a formula and use it to speed up the ATPG process to generate effective test vectors that avoid functionally untestable transition faults. A side benefit is that it also helps us to identify the functionally untestable faults (region S3 in Figure 6.3) for broadside testing.

As we described in Section 5.1, broadside vectors consists of initial state S1, primary input vectors PI1 and PI2. The intermediate state in the second time-frame is derived directly from S1 and PI1. PI2 in the second time-frame is independently applied. In our broadside ATPG, we unroll the sequential circuit to two time-frames

and attempt to generate a vector, which consists of state inputs (S), primary inputs in the 1st time frame (PI1) and primary inputs in the 2nd time frame (PI2). We denote a test vector V^j in the unrolled circuit as (S^j,$PI1^j$,$PI2^j$).

6.4.1 Problem Formulation

Given the set of functionally untestable transition faults, F_u, we want to make sure that the vectors generated will not incidentally detect any fault in F_u. A naive approach is to fault simulate the faults in F_u whenever a vector is obtained, and the ATPG engine would backtrack if some faults in F_u are incidentally detected. However, this naive approach can be computationally expensive. To reduce the expense, instead of focusing on F_u, we project each fault in F_u onto the state space to identify subspaces that will detect them. Subsequently, the ATPG only needs to avoid searching in the identified subspaces. We note that any state s that can detect any fault $f \in F_u$ would be an unreachable state, since fault f would otherwise be functionally detectable.

Figure 6.4.1 illustrates our broadside testing model. In this model, let us consider the detection of the *slow-to-rise* fault on line X, (We use $X1$ and $X2$ to represent line X in the first and second time-frame respectively.) We need to satisfy the following two objectives simultaneously:

1. Excite $X1$ s-a-1 fault in the first time-frame and detect $X2$ s-a-0 in the second time frame.

2. Avoid detection of any transition faults in F_u by making sure the search space does not overlap with the subspaces that can detect faults in F_u.

The second objective of avoiding detection of functionally untestable transition faults is key to the constrained ATPG. We first identify the state subspace that

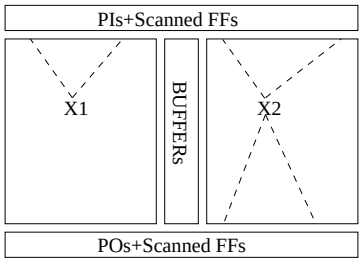


Figure 6.7: Broadside ATPG

can detect the functionally untestable transition faults, and this subspace is represented as a Boolean formula. Suppose that the circuit has n flip-flops s_1,...,s_n, a formula in conjunctive normal form (CNF) is used to represent the subspace. A CNF formula over the n binary variables is the conjunction (AND) of m clauses ω_1,...,ω_m, each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. Each clause in the formula represents a subspace that the ATPG must avoid. Therefore, at any given time during the ATPG search, no clause in the formula should evaluate to false (where each literal in the clause is valued to 0).

The subset of states that can detect any functionally untestable transition fault is obtained as follows.

1. Run the Implication Engine (TRANIMP) to identify the set of the functionally untestable transition faults (F_{u1}).

2. Fault simulate with a 5,000 random vector set, V_{5000}. Remove the easy functionally testable faults and record any vector in V_{5000} that detects one or more of the functionally transition faults in F_{u1}. Please such vectors in $V_{illegal}$.

3. For each vector in $V_{illegal}$, deduce the illegal state (IS) and negate it to construct the corresponding clause ω as a constraint.

4. Combine all the constraint clauses to form the CNF formula ϕ.

Because obtaining the complete set of states that can detect any functionally untestable transition fault is computationally expensive, we obtain only a subset of states via random simulation. As a result, there may still exist states outside the subset that may still detect a functionally untestable transition fault. However, our experiments showed that the subset is sufficient to significantly reduce the incidental detection of functionally untestable faults.

Table 6.4 illustrates an example of how the constrained CNF formula ϕ is constructed from the set of illegal states. During the random vector simulation, if a vector v^j detects one or more functionally untestable faults identified by the implication engine (TRANIMP), we record the state variables of v^j, deduce the corresponding constrained clause ω_j from the specified state variables. For example, in Table 6.4, three illegal states are reported. The first illegal state is $s_1 s_2 s_3 s_4 s_5 s_6 =$ 111XXX. The illegal state subspace can be represented simply as $s_1 \cdot s_2 \cdot s_3$. Negating this conjunction gives us the clause $\omega_1 = s_1' + s_2' + s_3'$. This clause essentially restricts any solution must fall within the subspace expressed by ω_1. The clauses for the other two illegal states can be obtained in a similar manner. Finally, the constrained CNF is formed by the conjunction of all the constrained clauses.

Table 6.4: CNF Formula Construction

Illegal States						$Constrained\ Clauses$
s_1	s_2	s_3	s_4	s_5	s_6	
1	1	1	X	X	X	$\omega_1 = s_1' + s_2' + s_3'$
X	1	0	X	1	X	$\omega_2 = s_2' + s_3 + s_5'$
X	0	0	X	X	0	$\omega_3 = s_2 + s_3 + s_6$

$\phi = \omega_1\omega_2\omega_3 = (s_1'+s_2'+s_3')(s_2'+s_3+s_5')(s_2+s_3+s_6)$

6.4.2 Constrained ATPG Algorithm

Next, we will discuss how the constrained CNF formula ϕ helps us to speedup the ATPG process (and identify extra functionally untestable faults also). During the ATPG process, we must make sure that no clause in ϕ ever evaluates to false (i.e.., all literals in a clause evaluates to false). Whenever we make a decision on a state variable s_k, we apply this decision assignment to all the constrained clauses in ϕ that contain s_k. Application of this assignment may result in some unit clauses (a unit clause is an unsatisfied clause with exactly one remaining unassigned literal left). This remaining literal is called an *implication.* The implied variable automatically becomes the next decision variable. We also check whether there is conflict (where one clause evaluates to false). If there is a conflict, backtrack immediately. A test vector is said to be generated for a transition fault X if it excites the fault X1 s-a-1 and detects faults X2 s-a-0, also it satisfies the constrained CNF ϕ.

Table 6.5: Implication on Decision Assignment

$CurrentDecision$	$ImpliedAssignment$
s_1=1	$None$
s_6=0	$None$
s_5=1	$None$
s_3=0	$s_2 = 0, s_6 = 1$
Backtrack s_3=1	$s_2 = 0$

Using the constrained CNF formula ϕ shown in Table 6.4, we explain the implication process on state variables in Table 6.5. After assigning s_1=1, we apply this assignment to the unsatisfied clauses containing s_1, no unit clause results, thus no implication can be made on other state variables. Next, suppose the ATPG makes the subsequent decisions s_6=0 and s_5=1. Applying these to ϕ still results in no implication. The next decision made by the ATPG is $s_3 = 0$. For clause ω_2, we can directly imply s_2=0 (because to satisfy clause (s_2'+s_3+s_5')=1, s_2 has to be 0).

Consequently, after the direct implication s_2=0, clause ω_3 evaluates to 0 because all literals in ω_3 has evaluated to 0. Therefore, we backtrack to the previous decision and assign s_3=1 and continue the ATPG process.

During the ATPG backtrace, an implication stack is dynamically updated to record the implication list of earlier backtrace choices similar to [HP98]. We maintain two dynamic implication lists: $Imp_{X1=0}$ for storing the implications that are necessary for setting X1=0, and $Imp_{X2=1}$ for storing implications necessary for setting X2=1. If there is conflict between $Imp_{X1=0}$ and $Imp_{X2=1}$, then we declare X *slow to rise* untestable. A conflict is observed when $X1 = v$ implies $X2 = v$ (v can be either logic 0 or 1). In other words, a transition is not possible on line X. Otherwise, we try to generate the test vector V for detecting X2 s-a-0. If V can incidentally excite X1 s-a-1 in the first time-frame as well, X *slow to rise* is detected. Otherwise, we continue to backtrace to excite X1 s-a-1. If not successful, we declare X *slow to rise* untestable.

6.5 Experimental Results

We implemented a constrained broadside ATPG based on PODEM [GOE81] in C++, as well as the implication-based untestable transition fault identification, also in C++. We further analyzed the effectiveness of our ATPG algorithm by comparing it with a conventional Broadside ATPG. Experimental data was collected for full-scan versions of ISCAS89 benchmark circuits on a 2.8GHz Pentium-4 with 512 MB of memory, running the Linux operation system.

First, Table 6.6 reports the functionally untestable transition faults identified by using our transition fault implication engine (TRANIMP). In order to see the effectiveness of the implication engine, we list the number of functionally untestable

Table 6.6: Functionally Untestable Faults Identified by Implication(TRANIMP)

circuit	$faults$	$1-TF$	$3-TF$	$5-TF$	$7-TF$
s344	1040	0	47	66	66
s1423	4288	33	387	387	387
s5378	15680	351	3673	3695	3695
s9234	29086	1327	6533	7415	7415
s13207	44130	1303	8900	14530	14540
s35932	103842	9536	11255	11255	11255

transition faults identified while considering different number of time frames for sequential circuit. The third column presents the number of untestable transition faults identified while only one time-frame is considered. The last three columns show the numbers of functionally untestable transition faults discovered while considering 3-time-frames, 5-time-frames and 7-time-frames, respectively. For example, in circuit s5378, one-time-frame implication found 351 functionally untestable transition faults. When the number of time-frames increases to 7, the number of functionally untestable faults identified increased to 7415.

Several interesting issues to note are listed below:

1. In general, the number of functionally untestable transition faults are much greater than the number of redundant stuck-at faults in the circuit.

2. The number of identified untestable transition faults increases with the number of time frames considered in the static implication graph.

3. Except for circuit s13207, the number of identified untestable transition fault saturates when the number of time frames increases to 7. Therefore, we can expect the number of untestable transition faults to not increase too much even if the number of time frames continue to increase.

Table 6.7 shows the (lack of) effectiveness of random vectors in avoiding

Table 6.7: Effectiveness of Random Vectors On Avoiding Functionally Untestable Faults

$circuit$	$Total$ $faults$	$FuncUnt$ $faults$	5000 RandVec		Pruned RandVec		
			OVT	DET	OVT	DET	UND
s344	1040	66	19	907	0	906	134
s1423	4288	387	160	2843	0	1163	3125
s5378	15680	3695	1415	9080	0	0	15680
s9234	29086	7415	1590	9875	0	0	29086
s13207	44130	14542	5306	12186	0	0	44130
s35932	103842	11255	1275	87328	0	49758	54084

detection of the functionally untestable transition faults. For each circuit, the total number of faults is first reported in column 2. Column 3 shows the number of functionally untestable faults identified by our implication engine (TRANIMP), column 4 reports the number of detected functionally untestable faults, and column 5 reports the coverage of the remaining faults. Then we remove those vectors which detect at least one functionally untestable faults from the test set and rerun the fault simulation. The results are reported under the *Pruned RandVec* columns. Obviously, for the pruned random vector set, it will not detect any identified functionally untestable faults, as shown in column 6. Columns 7 and 8 list the number of faults detected and missed by the pruned random vector set, respectively. It is interesting to see that for some of the circuits (s5378,s9234,s13207), all random vectors detect at least one functionally untestable fault! Therefore, if we want to reduce the yield loss by avoiding overtesting of functionally untestable faults, random vectors may not be very effective.

Table 6.8 reports the results from our constrained ATPG for Broadside testing and we compare it with a conventional non-constraint Broadside ATPG engine. We target only the faults that random vectors could not detect without incidentally detecting at least some functionally untestable transition faults. The sizes of

these remaining faults are first listed for each circuit under the second column. Columns 3 to 6 list the number of detected functionally testable faults, number of proved functionally untestable faults, number of aborted faults and test generation time for our constrained Broadside ATPG. The last four columns show the results when non-constrained ATPG is used. Although the execution time for constrained ATPG is longer than the non-constrained version, we significantly improve the quality of generated test vectors because the new test test only detect those functionally testable faults and avoid detecting of those functionally untestable ones. In other words, the vectors generated by the non-constrained ATPG detect **both** functionally testable and untestable faults. In addition, the constrained ATPG algorithm identified significantly more functionally untestable faults than the non-constrained ATPG. For example, in circuit s9234, our constrained ATPG identify 8095 functionally untestable faults out of the 29086 remaining potential functionally testable faults, while non-constrained ATPG only identify 3357 functionally untestable faults. Similarly, the number of aborted faults with our proposed method is also fewer. For instance, only 573 transition faults were aborted as opposed to 1396 transition faults in the non-constrained ATPG.

Table 6.8: Constrained ATPG Vs.Non-constrained ATPG

Ckt	$Rem.$	Constrained ATPG				Non-constrained ATPG			
	$flts$	DET	UNT	ABT	$Time(s)$	DET	UNT	ABT	$Time(s)$
s344	134	60	74	0	0.19	83	51	0	0.13
s1423	3125	2406	476	243	1621.65	2485	368	272	463.73
s5378	15680	10871	4270	539	5393.13	12999	2024	657	669.35
s9234	29086	20418	8095	573	8114.77	24333	3357	1396	2915.27
s13207	44130	21526	22549	55	19871.32	27150	16900	80	3148.62
s35932	54084	39952	14123	9	34754.80	39978	14089	17	6179.21

6.6 Summary

We presented a novel constrained broadside transition ATPG algorithm. We first identify the set of illegal (unreachable) states that enable detection of functionally untestable faults. Then, by formulating the illegal states as a constrained CNF formula in our ATPG process, we efficiently generated a higher quality test set detecting only those functionally testable faults and avoid overtesting of functionally untestable ones. The cost for the CNF formula construction is extremely low, making our formulation very practical. The constrained ATPG allows for earlier backtrack whenever an illegal state is encountered. In some circuits, significantly more functionally untestable transition faults have been identified. At the same time, more faults could be detected without incidental detection of functionally untestable transition faults. With a test set that reduces launching of transitions that are functionally impossible, we believe our method offers a practical solution to avoid overtesting of these functionally impossible transitions, thus reducing yield loss.

Chapter 7

On Identifying Functionally Untestable Transition Faults

7.1 Introduction

Higher clock rate, shrinking geometries, increasing metal density, etc. introduces various defects that stuck-at test cannot screen out. Therefore, delay fault testing, which verifies that the CUT operate correctly at desired speed, is becoming a necessity to maintain the product quality level. However, (non-scan) functional delay testing can be impractical for larger circuits in that large test sets may be required to achieve a desirable fault coverage. As a result, at-speed AC scan testing has been widely used in the industry to detect delay-induced defects. Compared to functional testing, scan-based testing for delay faults can decrease the overall ATPG complexity and cost, since both controllability and observability on the flip-flops are enhanced. But some of the functionally untestable faults which do not impair normal operation of the circuit may become testable in scan testing [REA01]. This scenario

is known as overtesting and may result in yield loss [LH03a]. Unfortunately, identifying the functionally untestable transition faults in large-scale sequential circuit can be prohibitively expensive, because it is of the same complexity as sequential ATPG, which is of exponential complexity in terms of the size of the circuit.

Much work [PR94, AC95, PR96, LPR98, RPLB99, WPR01, IA96, ILA96, PAS01, HSI02, SH03] has been published on identifying untestable and redundant stuck-at faults in both combinational and sequential circuits in the last decades. In general, these methods can be classified into two types: fault-oriented methods and fault-independent methods.

For fault-oriented methods, [AC95] presents two theorems on identifying untestable faults in sequential circuits. The single-fault theorem states that if a single fault injected in the last timeframe of a iterative logic array model for the sequential circuit is untestable, then the fault would be sequentially untestable. The multi-fault theorem states that an untestable multi-fault in the logic array corresponds to an untestable single fault in the sequential circuits. In [RPLB99], three new procedures were introduced as an extension to the theorems in [AC95] to help identify additional undetectable and redundant faults. And [WPR01] uses sensitizability of partial paths to determine redundant faults and incorporate some new features (such as blockage learning, dynamic branch ordering and fault grouping) to identify more redundant single stuck-at faults.

Most of the fault-oriented algorithms mentioned above are ATPG-based, which spend a lot of computational efforts on identifying untestable faults via exhaustive search. To alleviate the problem, approaches that do not rely on traditional branch-and-bound search algorithm have been proposed. In FIRE [IA96], the authors presented a fault-independent algorithm, which identifies a set of untestable faults that require conflicting value assignment on a single line for detection in a

combinational circuit. The idea is extended in FIRES [ILA96] for sequential circuits, which uses illegal state information as an extra criterion for untestable faults identification. Then, the authors of [CP96] proposed a generalized FIRES algorithm, which identifies c-cycle redundancies without simplifying assumption and state transition information. And on top of FIRES algorithm, [HSI02] proposed a multiple-node implication approach to maximize local conflicting value assignment for the purpose of untestable faults identification.

For delay faults, several approaches [BI94, LMB97, HPA97, CRP03, SH04] have been proposed on functionally untestable delay faults identification. In [BI94], a general delay fault model, which allows the delay faults to persist for many cycles, is introduced, and algorithms are developed to identify redundancies of arbitrary size. In [LMB97], a path-independent implication-based approach is proposed for identifying non-robust untestable path delay faults in combinational circuits. Additional general implication-based algorithms are given in [HPA97] on identifying untestable segment delay faults and robustly untestable, non-robustly untestable and functionally unsensitizable path delay faults. Most recently, approaches proposed in [CRP03, SH04] combine the implication-based approach and ATPG-based technique to identify more untestable transition faults in sequential circuits. It has also been noticed that the size of the transition fault (in terms of clock cycles) has to be considered for sequential circuits under at-speed testing [CHE93]. The author in [CHE93] points out that different transition fault sizes will cause different faulty circuit behavior. However, the number of untestable transition faults decreases significantly when the size of the fault increases from one cycle to multiple cycles [CRP03]. Therefore, we will assume the delay defect size is shorter than one clock cycle in this chapter.

In this chapter, we present a new approach on identifying functionally untestable

transition faults in non-scan sequential circuits. The proposed method consists of two phases: first, a large number of functionally untestable transition faults is quickly identified by using a fault-independent logic implications implicitly crossing multiple time-frames and classified into three conflict categories. Then, the technique identifies additional functionally untestable transition faults by finding the dominated fault sets on the previous identified untestable transition faults. The experimental results for ISCAS89 sequential benchmark circuits showed that our approach can identify many more functionally untestable transition faults than the implication-based method and a ATPG-based method previously reported.

The rest of the chapter is organized as follows. Section 7.2 reviews the logic implication and fault dominance, which will be used in the rest of chapter. Section 7.3 describes our proposed two-phase approach for identifying functionally untestable transition faults in sequential circuits. Section 7.4 gives our experimental results on ISCAS89 benchmark circuits and compares it with the implication-based method and another ATPG-based identification approach. Finally, Section 7.5 concludes the chapter.

7.2 Preliminaries

7.2.1 Static Logic Implication

Static logic implication (also called static learning) is a procedure which performs implications on both value assignment (0 and 1) for each node in the circuit. Since the effectiveness of implication-based untestable fault identification highly depends on the completeness of the implications learned, it is critical to have a large set of implications with each node in the circuit. A number of works have been reported

on implication computation. In [RK90], a 16-value logic algebra and deduction list method was proposed to determine necessary node assignments in ATPG. Transitive closure procedure was used in [CA93] to identify indirect implication on implication graphs. A more complete implication learning algorithm is based on recursive learning [KP94], however, the depth of recursion must be kept low to keep the computation time within reasonable bounds. [ZRP97] introduced the extended backward implication to identify additional nontrivial implications. And more recently, a compact implication graph [ZNP01] has been proposed for sequential circuits, which spans multiple time-frames without suffering from memory explosion. In our work, direct, indirect and extended backward implications are used. Although direct implications can be easily learned, indirect and extended backward implications require extensive usage of contrapositive and transitive laws. The discovery of indirect and extended backward implications are nontrivial and can help us in both ATPG process and untestable fault implications. The following terminology will be used:

1. $[a, v, t]$: assign logic value *v* to node a in timeframe t, where $v \in 0, 1$. When t=0, [N, *v*, 0] is also expressed as [N, *v*].

2. $[a, v] \rightarrow [b, u, t]$: assignment to logic value *v* on node a in the current time frame (t=0) implies value u assigned to node b in time frame t .

3. *impl*[a, *v*, t]: the implication set of assigning logic value *v* to node a in the timeframe t.

4. implication graph: a directed graph, where each node corresponds to a circuit node assignment [node, value], each directed edge denotes an implication and the weight on the edge represents the relative time-frame associated with the

implication.

5. transitive law: If $[a, v] \rightarrow [b, u, t_1]$ and $[b, u] \rightarrow [c, w, t_2]$, then we also have $[a, v] \rightarrow [c, w, t_1 + t_2]$.

6. contrapositive law: If $[a, v] \rightarrow [b, u, t]$,then $[b, u\prime] \rightarrow [a, v\prime, t]$.

7. a/*v*: the stuck-at-*v* fault on node a.

We will use Figure 7.1 to illustrate the direct implication, indirect implication and extended backward implication. Without losing generality, we look at the *impl* [g,1].

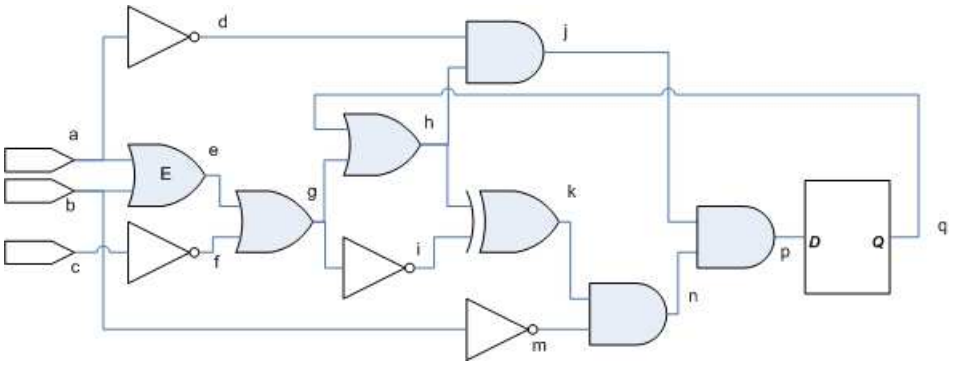


Figure 7.1: Example sequential circuit

1. Direct Implication

   By traversing the direct fan-in and fanout of gate*G*, we can easily find $[g, 1] \rightarrow \{[e, 1], [f, 1], [h, 1], [i, 0]\}$. Similarly, $[f, 1] \rightarrow [c, 0]$. Applying transitive law to it, the set of direct implications associated with g=1 is:

   impl [g,1]= {[g,1], [c,0], [e,1], [f,1], [h,1], [i,0]}

2. Indirect Implication

   Although [h,1] or [i,0] could not imply anything on node k individually,together, they will imply [k,1]. This is called an indirect implication, which is learned through circuit simulation. [k,1] will be added into the implication list for [g,1], which makes

   impl [g,1]= {[g,1], [c,0], [e,1], [f,1], [h,1], [i,0], [k,1]}

3. Extended Implication

   Finally, extended backward implications apply to unjustified gates in the implication list. In our example, gate *E* will be an unjustified gate, since the output signal of E is specified as 1 ([e,1]), but none of its input (i.e. a, b) was specified yet. Thus, E is a candidate for the application of extended backward implication.

   To compute extended implications on gate E, first, one of the unspecified inputs (node a) is set to logic value 1, we simulate [a,1] together with the *impl*[g,1]. And four new implications are found: {[d,0], [j,0], [p,0], [q,0,1]}. Similarly, if we simulate [b,1] together with the *impl*[g,1], we can have the new implication {[m,0], [n,0], [p,0] [q,0,1]}. Since the new implications [p,0] and [q,0,1] are common between the learned values during the simulation of {[a,1], *impl*[g,1]} and {[b,1], *impl*[g,1]}, they are added into the implication list for [g,1]. Therefore, the complete implication for [g,1] is now:

   impl [g,1]= {[g,1], [c,0], [e,1], [f,1], [h,1], [i,0], [k,1], [p,0], [q,0,1]}

These implications are stored in a compact implication graph as shown in Figure 7.2.

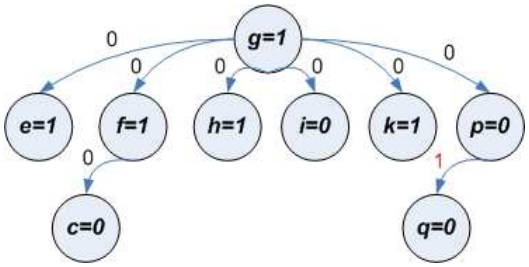


Figure 7.2: Implication Graph for [g,1]

In the implication graph, each node corresponds to a circuit node assignment [node, value], each directed edge denotes an implication and the weight on the edge represents the relative time-frame associated with the implication.

7.2.2 Fault Dominance

The following definition for **dominated fault** was given in [ABF90]. We will use the same definition in this chapter.

Definition 1 *Let T_g be the set of all tests that detect a fault* **g***. We say that a fault* **f** **dominates** *the fault* **g** *iff* **f** *and* **g** *are functionally equivalent under T_g.*

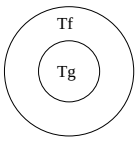


Figure 7.3: Test set T_f and T_g

Figure 7.3 gives us a graphic illustration of the test sets T_f and T_g when fault f dominates another fault g. From the figure, we can see that if f dominates g, then any test t that detects g will also detect f ($T_g \subseteq T_f$). Therefore, in ATPG process it

is unnecessary to consider the dominating fault f, because we automatically obtain a test that detects f by deriving a test to detect g. Particularly, if fault **f** and fault **g** dominates each other, then we say these two faults are equivalent.

Furthermore, if we look at the Figure 7.3 from a different perspective, we can observe that if fault f is untestable, then fault g must be untestable as well. Otherwise, there must be at least one test that can detect both fault g and f.

7.3 Our Approach

In this section, we describe our proposed method that consists of two phases. In Phase 1, a large number of functionally untestable transition faults is quickly identified by the fault-independent logic implications across multiple time-frames and classified into three conflict categories. However, the implication-based algorithm in Phase 1 may not be complete. Therefore, in Phase 2, additional functionally untestable transition faults are identified by finding the dominated fault sets on the previous identified untestable transition faults.

7.3.1 Phase 1: Untestable Transition Fault Identification with Implication

In Phase 1, the algorithm similar to [IA96, ILA96] is used, which identifies a set of untestable stuck-at faults that requires conflicting value assignment for detection in a circuit. However, unlike [IA96, ILA96], where implications did not span multiple times, our method computes sequential implications quickly without explicit unrolling of the circuit. In [IA96, ILA96], two sets of faults, set_0 and set_1 are computed with respect to a given node n:

set_i=set of faults that require $[n, i]$ as a necessary condition for excitation or propagation.

Then $set_0 \cap set_1$ is the set of faults that require both $[n, 0]$ and $[n, 1]$ simultaneously for detection, which must then be untestable.

For transition faults, our transition implication engine computes four sets of transition faults, ini_set_0, ini_set_1, cap_set_0, and cap_set_0 with respect to a given node n:

ini_set_i= set of transition faults that require $[n, i, -1]$ as a necessary condition for initialization.

cap_set_i= set of transition faults that require $[n, i]$ as a necessary condition for launch or propagation.

We further make the following three definitions:

Definition 2 *A transition fault ℓ slow-to-rise(slow-to-fall) is said to be* **sequentially uninitializable** *if there exists no input sequence T such that the targeted line ℓ could be initialized to logic 0(1) functionally.*

Based on Definition 2, all the transition faults belonging to $ini\_set_0 \cap ini\_set_1$ for any given node n would be sequentially uninitializable because they require both $[n, 0, -1]$ and $[n, 1, -1]$ to set the target node to initial value in the immediate preceding time frame.

Definition 3 *A transition fault ℓ slow-to-rise(slow-to-fall) is said to be* **sequentially uncapturable** *if there exists no input sequence T such that the stuck-at-0(stuck-at-1) fault on line ℓ could be sequentially launched and propagated.*

Based on Definition 3, all the transition faults belonging to $cap\_set_0 \cap cap\_set_1$ for any given node n would be sequentially uncapturable, because they require both $[n, 0]$ and $[n, 1]$ to launch or propagate the target transition fault.

Definition 4 *A transition fault ℓ slow-to-rise(slow-to-fall) is said to be* **sequentially constrained** *if the initialization of ℓ in time frame i and capture of fault effect on ℓ in time frame $i+1$ require conflicting value assignments.*

Thus, the sequentially constrained transition faults are those that are either (1) sequentially initializable but uncapturable in the immediate next time frame, or (2) sequentially capturable but not initializable in the immediate preceding time frame. Note that these three types of untestable transition faults are *mutually exclusive*. Therefore, the sequentially constrained transition faults for any given node n are $(ini\_set_0 \cup cap\_set_0) \cap (ini\_set_1 \cup cap\_set_1)$.

Below is the High-level description of the implication-based algorithm used in Phase 1.

1. Construct sequential implication graph through static learning
2. Using the single-line-conflict and maximizing local conflict algorithms, identify a set of functionally untestable transition faults, $S0$.
3. Classify the faults in $S0$ into three categories based on the definitions given above:
   (a) Type1: Sequentially uninitializable transition faults
   (b) Type2: Sequentially uncapturable transition faults
   (c) Type3: Sequentially constrained transition faults

7.3.2 Phase 2: Dominated Untestable Faults Identification

In Phase 2, we define three types of transition fault dominance relationship and describe how they help identify additional functionally untestable transition faults based on those identified untestable faults in Phase 1.

Definition 5 *Let TI_g and TI_f be the sets of all single time-frame vectors (FFs + PIs) that initialize transition faults* **g** *and* **f***, respectively. We say that transition fault* **f Initialization-dominates** *transition fault* **g** *iff $TI_g \subseteq TI_f$.*

Definition 6 *Let TC_g and TC_f be the sets of all single time-frame vectors that capture transition faults* **g** *and* **f***, respectively. We say that transition fault* **f Capture-dominates** *transition fault* **g** *iff $TC_g \subseteq TC_f$.*

Definition 7 *Let TI_g and TI_f be the sets of all single time-frame vectors that initialize transition faults* **g** *and* **f***, respectively, and let TC_g and TC_f be the sets of all single time-frame vectors that capture transition faults* **g** *and* **f***, respectively. We say that transition fault* **f Constrain-dominates** *transition fault* **g** *if the following two conditions are both met:*

1. $TI_g \subseteq TI_f$.
2. $TC_g \subseteq TC_f$.

We will use the circuit in Figure 7.4 to illustrate the three types of dominance relationship.

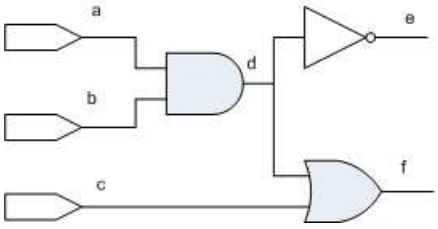


Figure 7.4: segment of sequential circuit

In Table 7.1, we show the test vectors that could initialize the transition faults in Figure 7.4. From the table, we can see that *d slow-to-rise* **Initialization-dominates** *a slow-to-rise, b slow-to-rise and e slow-to-fall*. This could be derived

from the fact that all the vectors, which could initialize *a slow-to-rise, b slow-to-rise or e slow-to-fall* could also initialize the *d slow-to-rise* as well. Similarly we can have *f slow-to-fall* initialization-dominates *c slow-to-fall*.

Table 7.1: Initialization Vectors

Fault	*Primary Inputs*		
	a	*b*	*c*
a slow-to-rise	0	X	X
a slow-to-fall	1	X	X
b slow-to-rise	X	0	X
b slow-to-fall	X	1	X
c slow-to-rise	X	X	0
c slow-to-fall	X	X	1
d slow-to-rise	0	X	X
	X	0	X
d slow-to-fall	1	1	X
e slow-to-rise	1	1	X
e slow-to-fall	0	X	X
	X	0	X
f slow-to-rise	0	X	0
	X	0	0
f slow-to-fall	X	X	1
	1	1	X

In Table 7.2, we show the test vectors that could capture the transition faults in Figure 7.4. From the table, we can see that *d slow-to-rise* **Capture-dominates** *a slow-to-rise, b slow-to-rise and e slow-to-fall*. This could be derived from the fact that all the vectors, which could capture *a slow-to-rise, b slow-to-rise or e slow-to-fall* could also capture the *d slow-to-rise* as well. Similarly, *f slow-to-fall* capture-dominates *c slow-to-fall* and *a slow-to-rise* capture-dominates *d-slow-to-rise*.

Table 7.2: Capture Vectors

Fault	*Primary Inputs*		
	a	*b*	*c*
a slow-to-rise	1	1	X
a slow-to-fall	0	1	X
b slow-to-rise	1	1	X
b slow-to-fall	1	0	X
c slow-to-rise	0	0	1
c slow-to-fall	0	0	0
d slow-to-rise	1	1	X
d slow-to-fall	0	X	X
	X	0	X
e slow-to-rise	0	X	X
	X	0	X
e slow-to-fall	1	1	X
f slow-to-rise	X	X	1
	1	1	X
f slow-to-fall	0	X	0
	X	0	0

The complete initialization dominance relationship and capture dominance relationship are stored in the two graphs as shown in Figures 7.5 and 7.6. In the dominance graph, each node corresponds to a transition fault in the circuit, each directed edge denotes a dominance relationship. If we consider Figures 7.5 and 7.6 together. We have the following observations:

1. *d slow-to-rise* dominates *a slow-to-rise*, because both TI and TC for *d slow-to-rise* are the superset of TI and TC for *a slow-to-rise*, respectively.

2. However, *a slow-to-rise* does **NOT** dominate *d slow-to-rise*, because test vector X0X could initialize d slow-to-rise but not a slow-to-rise.

3. It is notable that *d slow-to-rise* and *e slow-to-fall* dominates each other. There-

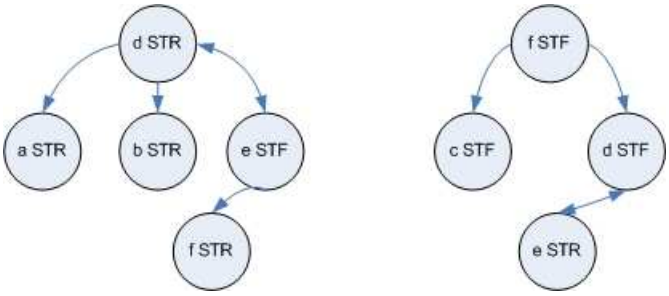


Figure 7.5: Initialization Dominance Graph

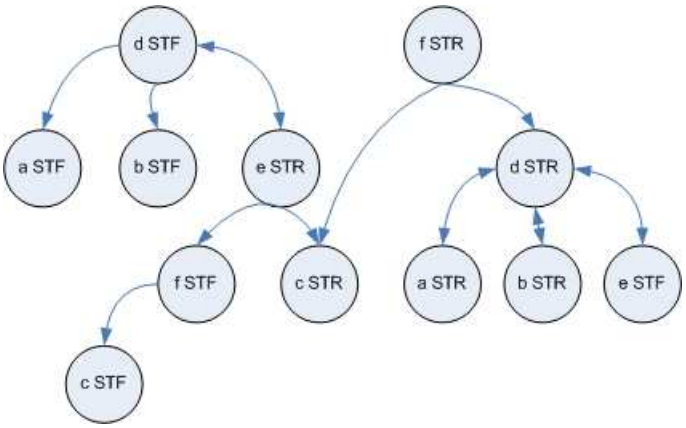


Figure 7.6: Capture Dominance Graph

fore, they are equivalent transition faults.

The dominated untestable faults identification algorithm used in Phased 2 is given as follows:

1. Construct the fault-initialization dominance graph (INI_DOM) and the fault-capture dominance graph (CAP_DOM).
2. S1=S2=S3=EMPTY
3. for every identified untestable transition faults *f* in *S0*:

(a) case Type1: Search the fault-initialization dominance graph (INI_DOM) to identify dominated untestable faults by *f* and add them in (*S1*)

(b) case Type2: Search the fault-capture dominance graph (CAP_DOM) to identify dominated untestable faults by *f* and add them in (*S2*)

(c) case Type3:

$tfset_0$ = untestable transition faults that are initialization-dominated by *f*

$tfset_1$ = untestable transition faults that are capture-dominated by *f*

The untestable fault set: **S3**= $\mathbf{S3} \cup (\mathbf{tfset_0} \cap \mathbf{tfset_1})$

4. The final, complete set of functionally untestable transition faults: $\mathbf{S} = \mathbf{S0} \cup \mathbf{S1} \cup \mathbf{S2} \cup \mathbf{S3}$

7.4 Experimental Results

The algorithms explained above were implemented in C++. Experimental data are collected for non-scan ISCAS89 sequential benchmark circuits, on a 2.0GHz Pentium 4 with 512 MB of memory, running the Linux Operating System. The result of our proposed two-phase procedure on ISCAS 89 benchmark circuits are reported in Table 7.3, For each circuit, we first give the number of transition faults in the circuit. Next, we show the number of functionally untestable transition faults identified in Phase 1. In column 4–7, we report the number of sequentially uninitializable transition faults, sequentially uncapturable transition faults, sequentially constrained transition faults and total number of functionally untestable transition faults identified by our proposed approach, respectively. Next, column *Time* reports the corresponding CPU execution time for Phase 2 procedure.

From this table, many additional functionally untestable transition faults have been identified using our approach. For example, in circuit s386, the implication-only algorithm in Phase 1 identified 195 untestable transition faults, and Phase 2 found 91 additional untestable ones, making a total of 286 faults identified as untestable, in only 0.02 second extra execution time. The number of additional functionally untestable transition faults identified in large sequential circuits (except for s35932) could be significant. For example, in s38417r, the implication-based technique found 6434 functionally untestable transition faults, while the proposed method identified totally 7106, which is 672 more, in only 12 seconds. For s35932, since the implication-based procedure already identified all the functionally untestable transition faults, no additional faults could be identified by the new technique. Overall, the proposed two-phase approach improve the number of identified functionally untestable transition faults by 5.8% over the implication-based method in less than 1 minutes totally.

In Table 7.4, we compare our results with the implication-based method and a ATPG-based approach [CRP03] for some of the large ISCAS89 benchmark circuits. After the first two columns for circuit name and number of transition faults in the circuit, we show the number of untestable transition faults identified by Chen etc. in [CRP03], using a ATPG-based scheme. Then, we give the number of untestable ones identified by implications only and our new technique, respectively. In the last column, we show the percentage improvement our approach has over the ATPG-based method.

For three circuits (S1238, s1488 and s1494) out of the 13 large benchmark circuits, the ATPG-based method [CRP03] performed better. In all the other cases, our proposed technique identifies more functionally untestable transition faults than the other two methods. For example, in circuit s38417r, our technique identi-

fied 7106 functionally untestable transition faults, which is more than doubling of 3075, the number of untestable transition faults identified in [CRP03]. Overall, our new approach identified 43% more functionally untestable transition faults than the ATPG-based method in [CRP03].

7.5 Summary

In this chapter, we present a new approach on identifying functionally untestable transition faults in non-scan sequential circuits. We formulate a new dominance relationship for transition faults and use it to help us identify more untestable transition faults. The proposed technique consists of two phases: first, a large number of functionally untestable transition faults was identified by using the fault-independent logic implications implicitly crossing multiple time-frames and classified into three categories according to the sequential constraint on their initialization and capture requirement. Then, additional functionally untestable transition faults were identified by finding the dominated fault sets on the identified untestable transition faults. The experimental results for ISCAS89 sequential benchmark circuits showed that the proposed two-phase approach identified 5.8% more functionally untestable faults than implication-based method and 43% over the ATPG-based approach previously reported.

Table 7.3: Experimental Results using Implication & Dominance

CKT		*#Functional Untestable Faults Identified*					*Time*
Name	*#Faults*	*Phase1*	*Phase1+Phase2*				*(s)*
			#UnInit	*#UnCap*	*#Constr*	*#Total*	
s298	812	129	8	8	141	157	0.01
s344	980	66	11	10	52	73	0.01
s386	1078	195	6	16	264	286	0.02
s444	1180	226	6	22	231	259	0.02
s526	1390	345	0	1	400	401	0.03
s641	2078	66	0	0	88	88	0.03
s713	2206	208	40	81	120	241	0.03
s832	2244	96	0	8	131	139	0.03
s953	2516	81	0	0	109	109	0.03
s1196	3232	5	0	0	8	8	0.04
s1238	3254	58	0	50	13	63	0.04
s1423	3992	387	0	33	463	496	0.07
s1488	4196	22	0	0	38	38	0.06
s1494	4196	30	0	11	34	45	0.05
s5378	14964	3695	1090	1002	1742	3834	0.70
s9234	28174	7415	284	1358	6389	8031	3.14
s13207r	41758	14542	2397	1255	11488	15140	18.52
s15850r	49654	16402	416	1190	15741	17347	13.86
s35932	96930	11255	0	9536	1719	11255	3.86
s38417r	116310	6434	376	577	6153	7106	12.09
s38584r	111578	14923	1919	5975	8031	15925	15.17
Sum	492722	76580	6553	21133	53355	81041	52.64

Table 7.4: Comparison With Previous Work

CKT		*Func. Unt. faults*			*Improvement*
Name	*#Faults*	*[CRP03]*	*Impl*	*Ours*	*(%)*
s953	2516	101	81	109	7.9
s1196	3232	2	5	8	300
s1238	3254	82	58	63	-23.2
s1423	3992	273	387	496	81.7
s1488	4196	241	22	38	-84.2
s1494	4196	262	30	45	-82.8
s5378	14964	1645	3695	3834	133.1
s9234	28174	5600	7415	8031	43.4
s13207r	41758	10637	14542	15140	42.3
s15850r	49654	10259	16402	17347	69.1
s35932	96930	8903	11255	11255	26.4
s38417r	116310	3075	6434	7106	131.1
s38584r	111578	14450	14923	15925	10.2
Sum	480754	55530	75249	79397	43.0

Chapter 8

Conclusions

In this dissertation, we addressed some of the key problems in delay testing (such as explosion in test data volume and application time, lower fault coverage, higher ATPG complexity and overtesting problem in scan-based delay testing) and developed several novel and efficient ATPG and Design-for-testability (DFT) algorithms for delay testing.

First, we presented two algorithms, Fault-List-based extension and Priority-based extension, for composing transition patterns from vectors in a s@ test set. The priority-based algorithm was shown to be superior to the fault-list based algorithm. It was demonstrated that high quality transition pattern sets can be obtained, bypassing the need for a native mode transition fault ATPG. Experimental comparison with a native mode transition fault ATPG tool showed the proposed heuristics resulted in 20% smaller pattern set while achieving the same or higher transition fault coverage. We discussed the additional advantages of reusing the s@ vectors in pattern validation, constraint handling and reducing design data in the context of IP cores.

Secondly, We proposed efficient techniques to reduce test data volume and

test application time for transition faults. First, we proposed a novel transition test chain formulation via a weighted transition pattern graph. Only s@ ATPG was needed to construct the necessary test chains for transition faults. By combining the proposed transition test chain and ATE repeat capability to reduce the test data volume by 46.5%, when compared with the conventional approach. The second technique that replaces the ATE repeat option with Exchange Scan improves both test data volume and test application time by 46.5%. In addition, we address the problem of yield loss due to incidental *overtesting* of functionally untestable transition faults, By formulating it into a constraint in our weighted pattern graph, we can efficiently reduce the overtesting ratio. The average reduction on the overtesting ratio is 4.68%, with a maximum reduction of 14.5%.

Thirdly, a novel scan-based delay test approach, referred as the hybrid delay scan, has been proposed. The proposed method combines advantages of skewed-load and broad-side approaches and can achieve higher delay fault coverage than the broad-side approach. By selecting only a small fraction of the state inputs as the skewed-load flip-flops, we avoid the costly design requirement in skewed-load approach due to the fast scan enable signal that must switch in a full system clock cycle. Our experimental results show that for all the ISCAS 89 Benchmarks, the transition delay fault coverage achieved by hybrid approach is higher than or equal to that achieved by broadside load approach, with an average improvement of 4.47%.

Next, we presented a novel constrained broadside transition ATPG algorithm. We first identify the set of illegal (unreachable) states that enable detection of functionally untestable faults. Then, by formulating the illegal states as a constrained CNF formula in our ATPG process, we efficiently generated a higher quality test set detecting only those functionally testable faults and avoid overtesting

of functionally untestable ones. The cost for the CNF formula construction is extremely low, making our formulation very practical. The constrained ATPG allows for earlier backtrack whenever an illegal state is encountered. In some circuits, significantly more functionally untestable transition faults have been identified. At the same time, more faults could be detected without incidental detection of functionally untestable transition faults. With a test set that reduces launching of transitions that are functionally impossible, we believe our method offers a practical solution to avoid overtesting of these functionally impossible transitions, thus reducing yield loss.

Finally, we present a new approach on identifying functionally untestable transition faults in non-scan sequential circuits. We formulate a new dominance relationship for transition faults and use it to help us identify more untestable transition faults. The proposed technique consists of two phases: first, a large number of functionally untestable transition faults was identified by using the fault-independent logic implications implicitly crossing multiple time-frames and classified into three categories according to the sequential constraint on their initialization and capture requirement. Then, additional functionally untestable transition faults were identified by finding the dominated fault sets on the identified untestable transition faults. The experimental results for ISCAS89 sequential benchmark circuits showed that the proposed two-phase approach identified 5.8% more functionally untestable faults than implication-based method and 43% over the ATPG-based approach previously reported.

Bibliography

[ABF90] M. ABRAMOVICI, M. A. BREUER, and A. D. FRIEDMAN. *Digital Systems Testing and Testable Design*. Computer Science Express, 1990.

[AC95] V. D. AGRAWAL and S. T. CHAKRADHAR. Combinational ATPG Theorems for Identifying untestable faults in synchronous Sequential Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 14(9):1155–1160, Sep. 1995.

[BBK89] F. BRGLEZ, D. BRYAN, and K. KOZMINSKI. Combinational Profiles of Sequential Benchmark Circuits. In *Proceedings IEEE International Symposium on Circuits and Systems*, 1989.

[BI94] D. BRAND and V. S. IYENGAR. Identification of Redundant Delay Faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 13(5):553–565, MAY 1994.

[BRS+02] D. BELETE, A. RAZDAN, W. SCHWARZ, R. RAINA, C. HAWKINS, and J. MOREHEAD. Use of DFT Techniques In Speed Grading a 1GHz+ Microprocessor. In *Proceedings IEEE International Test Conference*, pages 1111–1119, 2002.

[CA93] S. T. CHAKRADHAR and V. D. AGRAWAL. A transitive closure algorithm for test generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 12(7):1015–1028, JULY 1993.

[CC01] A. CHANDRA and K. CHAKRABARTY. Frequency Directed Run Length(FDR) Codes with Application to System on a Chip Data Compression. In *Proceedings VLSI Testing Symposium*, pages 42–47, 2001.

[CHE93] K.-T. CHENG. Transition Fault Testing for Sequential Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 12(12), Dec 1993.

[CIR87] J. L. CARTER, V. S. IYENGAR, and B. K. Rosen. Efficient Teset Coverage Determination for Delay Faults. In *Proceedings IEEE International Test Conference*, pages 418–427, 1987.

[CL95] J.-S. CHANG and C.-S. LIN. Test Set Compaction for Combinational Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 14(11):1370–1378, November 1995.

[COM00] S. COMEN. DFT-focused chip testers: What can they really do? In *Proceedings IEEE International Test Conference*, page 1120, 2000.

[CP96] W. CAO and D. K. PRADHAN. Sequential Redundancy Identification Using Recursive Learning. In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 56–62, 1996.

[CRP03] G. CHEN, S. M. REDDY, and I. POMENRANZ. Procedures for Identifying Untestable and Redundant Transition Faults in Synchronous

Sequential Circuits. In *Proceedings IEEE International Conference on Computer Design*, pages 36–41, 2003.

[CS01] L. CHEN and S.DEY. Software-Based Self-Testing Methodology for Processor Cores . *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 20(3):369–390, MARCH 2001.

[DS91] B. DERVISOGLU and G. STONG. Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement. In *Proceedings IEEE International Test Conference*, pages 365–374, 1991.

[DT00] D. DAS and N. A. TOUBA. Reducing Test Data Volume Using External/BIST Hybrid Test Patterns. In *Proceedings IEEE International Test Conference*, pages 115–122, 2000.

[ELD59] R.D. ELDRED. Test Routing Based on Symbolic Logical Statement. *Journal ACM*, 6:33–36, January 1959.

[GOE81] P. GOEL. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Trans. on Computers*, Vol. C-30(3), March 1981.

[GR79] P. GOEL and B. C. ROSALES. Test Generation & Dynamic Compaction of Tests. In *Dig. Papers Test Conference*, pages 182–192, 1979.

[GT80] L. H. GOLDSTEIN and E. L. THIGPEN. SCOAP: Sandia Controllability/Observability Analysis Program. In *Proceedings IEEE-ACM Design Automation Conference*, pages 190–196, 1980.

[HBP01] F. F. HSU, K. M. BUTLER, and J. H. PATEL. A Case Study of the Illinois Scan Architecture. In *Proceedings IEEE International Test Conference*, pages 538–547, 2001.

[HP98] I. HAMZAOGLU and J. H. PATEL. New Techniques for Deterministic Test Pattern Generation. In *Proceedings VLSI Testing Symposium*, pages 446–452, 1998.

[HP99] I. HAMZAOGLU and J. H. PATEL. Reducing Test Application time for full scan embedded cores. In *29th International symposium on Fault-Tolerant Computing*, pages 260–267, 1999.

[HPA96] K. HERAGU, J. H. PATEL, and V. D. AGRAWAL. Segment delay faults: a new fault model. In *Proceedings VLSI Testing Symposium*, pages 32–39, 1996.

[HPA97] K. HERAGU, J. H. PATEL, and V. D. AGRAWAL. Fast Identification of Untestable Delay Faults Using Implication. In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 642–647, 1997.

[HRP97] M. S. HSIAO, E. M. RUDNICK, and J. H. PATEL. Sequential circuit test generation using dynamic state traversal . In *ProceedingsEuropean Design and Test Conference*, pages 22–28, 1997.

[HSI02] M. S. HSIAO. Maximizing Impossibilities for Untestable Fault Identification. In *ProceedingsIEEE Design, Automation and Test in Europe Conference and Exhibition*, pages 949–953, 2002.

[IA96] M. A. IYER and M. ABRAMOVOCI. FIRE: a Fault Independent Combinational Redundancy Algorithm. *IEEE Trans. VLSI Systems*, 4(2):295–301, June 1996.

[ILA96] M. A. IYER, D. E. LONG, and M. ABRAMOVICI. Identifying Sequentially Redundancies without Search . In *Proceedings IEEE-ACM Design Automation Conference*, pages 259–266, 1996.

[KBB+01] B. KELLER, C. BARNHART, V. BRUNKHORST, F. DISTLER, A. FERKO, O. FARNSWORTH, and B. KOENEMAN. OPMISR: The Foundation of Compressed ATPG Vectors. In *Proceedings IEEE International Test Conference*, pages 748–757, 2001.

[KLC+02] A. KRISTIC, W. C. LAI, L. CHEN, K. T. CHENG, and S. DEY. Embedded software-based self-testing for SoC design. In *Proceedings IEEE-ACM Design Automation Conference*, pages 355–360, 2002.

[KOE91] B. KOENEMAN. LFSR-Coded Test Patterns for Scan Designs. In *IEEE European Test Conference*, pages 237–242, 1991.

[KP94] W. KUNZ and D. K. PRADHAN. Recursive Learning: a new implication technique for efficient solutions to CAD problems-test, verification and optimization. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 13(9):1149–1158, SEPT 1994.

[LCH98] K-J. LEE, J-J. CHEN, and C-H HUANG. Using a Single Input to Support Multiple Scan Chains. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 74–78, 1998.

[LH03a] X. LIU and M. S. HSIAO. Constrained ATPG for Broadside Transition Testing. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 175–182, 2003.

[LH03b] X. LIU and M. S. HSIAO. Constrained Transition Fault ATPG to Reduce Yield Loss in SOCs. In *the 3rd Annual Emerging Information Technology Conference*, 2003.

[LHCT02a] X. LIU, M. S. HSIAO, S. CHKRAVARTY, and P. J. THADIKARAN. Novel ATPG Algorithms for Transition Faults . In *Proceedings of the IEEE European Test Workshop*, pages 47–52, 2002.

[LHCT02b] X. LIU, M. S. HSIAO, S. CHKRAVARTY, and P. J. THADIKARAN. Techniques to Reduce Data Volume and Application Time for Transition Test . In *Proceedings IEEE International Test Conference*, pages 983–992, 2002.

[LHCT03] X. LIU, M. S. HSIAO, S. CHAKRAVARTY, and P. J. THADIKARAN. Efficient Transition Fault ATPG Algorithms Based on Stuck-at Test Vectors. *Journal of Electronic Testing Theory and Applications*, 19(4):437–445, August 2003.

[LHCT04] X. LIU, M. S. HSIAO, S. CHAKRAVARTY, and P. J¿ THADIKARAN. Efficient Techniques for Transition Testing. *IEEE Trans. on Design Automation of Electronic System*, 2004.

[LKC00a] W. C. LAI, A. KRISTIC, and K. T. CHENG. On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set. In *Proceedings VLSI Testing Symposium*, pages 15–20, 2000.

[LKC00b] W. C. LAI, A. KRISTIC, and K. T. CHENG. On testing the path delay faults of a microprocessor using its instruction set. In *Proceedings VLSI Testing Symposium*, pages 15–20, 2000.

[LKC00c] W. C. LAI, A. KRISTIC, and K. T. CHENG. Test Program Synthesis for Path Delay Faults in Microprocessor Cores. In *Proceedings IEEE International Test Conference*, pages 1080–1089, 2000.

[LKC00d] W. C. LAI, A. KRISTIC, and K. T. CHENG. Test program synthesis for path delay faults in microprocessor cores. In *Proceedings IEEE International Test Conference*, pages 1080–1089, 2000.

[LMB97] Z. LI, Y. MIN, and R. K. BRAYTON. Efficient Identification of Non-Robustly Untestable Path Delay Faults. In *Proceedings IEEE International Test Conference*, pages 992–997, 1997.

[LPR98] X. LIN, I. POMENRANZ, and S. M. REDDY. On Finding undetectable and Redundant Faults in Synchronous Sequential Circuits. In *Proceedings IEEE International Conference on Computer Design*, pages 5–7, 1998.

[MaLB00] P. MAXWELL and I. HARTANTO an L. BENTZ. Comparing Functional and Structural Tests . In *Proceedings IEEE International Test Conference*, pages 400–407, 2000.

[PAS01] Q. PENG, M. ABRAMOVICI, and J. SAVIR. MUST: Multiple Stem Analysis for Identifying Sequentially Untestable Faults. In *Proceedings IEEE International Test Conference*, pages 839–846, 2001.

[PR94] I. POMENRANZ and S. M. REDDY. On Identifying undetectable and Redundant Faults in Synchronous Sequential Circuits. In *Proceedings VLSI Testing Symposium*, pages 8–14, 1994.

[PR96] I. POMENRANZ and S. M. REDDY. On Removing Redundancies from Synchronous Sequential Circuits with Synchronizing Sequences. *IEEE Trans. on Computers*, Vol. 45(1):20–32, Jan. 1996.

[PR97] A. K. PRAMANICK and S. M. REDDY. On the Fault Coverage of Gate Delay Fault Detecting Tests. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 16(1):78–94, January 1997.

[PRR91] I. POMERANZ, L. N. REDDY, and S. M. REDDY. COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits. In *Proceedings IEEE International Test Conference*, pages 194–203, 1991.

[REA01] J. REARICK. Too much Delay Fault Coverage is a Bad Thing. In *Proceedings IEEE International Test Conference*, pages 624–633, 2001.

[RK90] J. RAJSKI and H. KOX. A method to calculate necessary assignments in ATPG. In *Proceedings IEEE International Test Conference*, pages 25–34, 1990.

[RM01] J. REARICK and P. MAXWELL. Deception by Design: Fooling Ourselves with Gate-level Models . In *Proceedings IEEE International Test Conference*, pages 921–929, 2001.

[ROB00] G. D. ROBINSON. DFT-focused chip testers: What can they really do? In *Proceedings IEEE International Test Conference*, page 1119, 2000.

[RPLB99] S. M. REDDY, I. POMENRANZ, X. LIN, and N. Z. BASTURKMEN. New Procedures for identifying Undetectable and Redundant Faults in Synchronous Sequential Circuits. In *Proceedings VLSI Testing Symposium*, pages 275–281, 1999.

[SAV92a] J. SAVIR. Skewed-Load Transition Test: Part I, Calculus. In *Proceedings IEEE International Test Conference*, pages 705–713, 1992.

[SAV92b] J. SAVIR. Skewed-Load Transition Test: Part I, Coverage. In *Proceedings IEEE International Test Conference*, pages 714–722, 1992.

[SAV94] J. SAVIR. Broad-side Delay Test. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 13(8):1057–1064, August 1994.

[SB87] M. H. SCHULZ and F. BRGLEZ. Accelerated Transition Fault Simulation. In *Proceedings IEEE-ACM Design Automation Conference*, pages 237–243, 1987.

[SB91] J. SAVIR and R. BERRY. At-Speed Test is not Necessarily an AC Test. In *Proceedings IEEE International Test Conference*, pages 722–728, 1991.

[SBG+02] J. SAXENA, K. M. BUTLER, J. GATT, R. R, S. P. KUMAR, S. BASU, D. J. CAMPBELL, and J. BERECH. Scan-Based Transition Fault Testing - Implementation and Low Cost Test Challenges. In

Proceedings IEEE International Test Conference, pages 1120–1129, 2002.

[SH03] M. SYAL and M. S. HSIAO. A Novel, Low-cost Algorithm for Sequentially Untestable Fault Identification. In *ProceedingsIEEE Design, Automation and Test in Europe Conference and Exhibition*, pages 316–321, 2003.

[SH04] M. SYAL and M. S. HSIAO. Identifying Untestable Transiiton Faults in Latch Based Designs with Multiple Clocks. In *to appear in the Proceedings IEEE International Test Conference*, 2004.

[SMI85] G. L. SMITH. Model for Delay Faults Based Upon Paths. In *Proceedings IEEE International Test Conference*, pages 342–349, 1985.

[SP93] J. SAVIR and S. PATIL. Scan-Based Transition Test. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 12(8), August 1993.

[SP94a] J. SAVIR and S. PATIL. Broad-Side Delay Test. *IEEE Trans. on Computer-Aided Design of Integrated Circuit and System*, Vol. 13(8), August 1994.

[SP94b] J. SAVIR and S. PATIL. On Broad-Side Delay Test. In *Proceedings VLSI Testing Symposium*, pages 284–290, 1994.

[WC03] S. WANG and S. T. CHAKRADHAR. A Scalable Scan-Path Test Point Insertion Technique to Enhance Delay Fault Coverage for Standard Scan Designs. In *To appear in Proceedings IEEE International Test Conference*, 2003.

[WLC03] S. WANG, X. LIU, and S. T. CHAKRADHAR. Hybrid Delay Scan: A Low Hardware Overhead Scan-based Delay Test Technique for Compact and High Fault Coverage Test Sets. Technical Report 2003-L078, NEC USA, 2003.

[WLC04] S. WANG, X. LIU, and S. T. CHAKRADHAR. Hybrid Delay Scan: A Low Hardward Overhead Scan-based Delay Test Technique for High Fault Coverage and Compact Test Sets. In *ProceedingsIEEE Design, Automation and Test in Europe Conference and Exhibition*, pages 949–953, 2004.

[WLRI87] J. A. WAICUKAUSKI, E. LINDBLOOM, B. K. ROSEN, and V. S. IYENGAR. Transition Fault Simulation. *IEEE Design & Test of Computers*, pages 32–38, April 1987.

[WPR01] CHEN WANG, I. POMENRANZ, and S. M. REDDY. REDI: An Efficient Fault Oriented Procedure to identify Redundant Faults in Combinational Circuits . In *Proceedings IEEE International Conference on Computer-Aided Design*, pages 370–374, 2001.

[ZNP01] J. ZHAO, J. A. NEWQUIST, and J. PATEL. A Graph Traversal Based Framework for Sequential Logic Implication with an Application to C-cycle Redundancy Identification. In *Proceedings International Conference on VLSI Design*, pages 163–169, 2001.

[ZRP97] J. ZHAO, M. RUDNIK, and J. PATEL. Static Logic Implication with Application to Fast Redundancy Identification . In *Proceedings VLSI Testing Symposium*, pages 288–293, 1997.

VITAE

Xiao Liu was born and raised in Wuhan, Hubei, China. He received his B.S. in major of electrical engineering and minor of English from Huazhong University of Science and Technology, China, in 1996 and M.S. in electrical engineering in 1999 from the same university. He had been a Ph.D candidate, under the supervision of Dr. Michael S. Hsiao, in the Department of Electrical and Computer Engineering at Rutgers, the State University of New Jersey between 1999 and 2001. Since 2001, he has been continuing his Ph.D study with Dr. Hsiao in the Bradley Department of Electrical and Computer Engineering at Virginia Polytechnic Institute and State University. He recently got a job in Texas Instruments as a DFT lead in its ASIC group in Dallas, TX. His current research interests include transition fault ATPG, delay testing and design-for-testability.