Spring 2013

Compaction mechanism to reduce test pattern counts and segmented delay fault testing for path delay faults

Sharada Jha
University of Iowa

COMPACTION MECHANISM TO REDUCE TEST PATTERN COUNTS AND SEGMENTED DELAY FAULT TESTING FOR PATH DELAY FAULTS

by

Sharada Jha

An Abstract

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

May 2013

Thesis Supervisor: Professor Sudhakar M. Reddy

ABSTRACT

With rapid advancement in science and technology and decreasing feature size of transistors, the complexity of VLSI designs is constantly increasing. With increasing density and complexity of the designs, the probability of occurrence of defects also increases. Therefore testing of designs becomes essential in order to guarantee fault-free operation of devices.

Testing of VLSI designs involves generation of test patterns, test pattern application and identification of defects in design. In case of scan based designs, the test set size directly impacts the test application time which is determined by the number of memory elements in the design and the test storage requirements. There are various methods in literature which are used to address the issue of large test set size classified as static or dynamic compaction methods depending on whether the test compaction algorithm is performed as a post-processing step after test generation or is integrated within the test generation. In general, there is a trade-off between the test compaction achievable and the run-time. Methods which are computationally intensive might provide better compaction, however, might have longer run times owing to the complexity of the algorithm.

In the first part of the thesis we address the problem of large test set size in partially scanned designs by proposing an incremental dynamic compaction method. Typically, the fault coverage curve of designs ramp up very quickly in the beginning and later slows down and ultimately the curve flattens towards the tail of the curve. In the initial phase of test generation a greedy compaction method is used because initially there are easy-to-detect faults and the scope for compaction is better. However, in the later portion of the curve, there are hard-to-detect faults which affect compaction and we propose to use a dynamic compaction approach. We propose a novel mechanism to identify redundant faults during dynamic compaction to avoid targeting them later. The

effectiveness of method is demonstrated on industrial designs and test size reduction of 30% is achieved.

As the device complexity is increasing, delay defects are also increasing. Speed path debug is necessary in order to meet performance requirements. Speed paths are the frequency limiting paths in a design identified during debug. Speed paths can be tested using functional patterns, transition n-detect patterns or path delay patterns. However, usage of functional patterns for speed path debug is expensive because generation of functional patterns is expensive and the application cost is also high because the number of patterns is large and requires functional testers.

In the second part of the dissertation we propose a simple path sensitization approach that can be used to generate pseudo-robust tests, which are near robust tests and can be used for designs that have multiple clock domains. The fault coverage for path delay fault APTG can be further improved by dividing the paths that are not testable under pseudo robust conditions, into shorter sub-paths. The effectiveness of the method is demonstrated on industrial designs.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

COMPACTION MECHANISM TO REDUCE TEST PATTERN COUNTS AND SEGMENTED DELAY FAULT TESTING FOR PATH DELAY FAULTS

by

Sharada Jha

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

May 2013

Thesis Supervisor: Professor Sudhakar M. Reddy

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

PH.D. THESIS

_____

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

Sharada Jha

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the May 2013 graduation.

Thesis Committee: _____
Sudhakar M. Reddy, Thesis Supervisor

_____
David R. Andersen

_____
Mona K. Garvin

_____
Jon G. Kuhl

_____
Hantao Zhang

To My Family

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude and appreciation to all those who have challenged, supported and encouraged me in the entire course of PhD. First of all, I am deeply grateful to my thesis advisor, Professor Sudhakar Reddy and would like to extend my gratitude for all the expert guidance, valuable professional advice and help throughout my thesis. To work with him was a great opportunity for me and has been a great learning experience. He has been a constant support throughout the various phases of PhD. I would like to thank the dissertation committee members Professor David R. Andersen, Professor Mona K. Garvin, Professor Jon G. Kuhl and Professor Hantao Zhang for their time and effort for my dissertation. It is an honor for me to have them as my committee members.

I would like to express my sincere appreciation to Ramesh Sharma, Sanjay Sengupta, Yonsang Cho, Kameshwar Chandrasekar and Weixin Wu at Intel who guided and extended their valuable knowledge which helped me in my research work.

I would like to thank my colleagues Xiaoxin Fan, Amit Kumar, Joseph Howard and others at University of Iowa who made my stay at Iowa an enjoyable one. I would particularly like to thank my friends Swathi Kode, Hema Kumari Achanta and Pallavi Marrapu for all their help and support.

I would like to express my deepest and everlasting gratitude to my parents and my brothers for their love, support and encouragement throughout my life.

I would like to specially thank my husband, Saroj Kumar Jha, for his love, care and encouragement. His contribution towards my thesis is immeasurable.

ABSTRACT

With rapid advancement in science and technology and decreasing feature size of transistors, the complexity of VLSI designs is constantly increasing. With increasing density and complexity of the designs, the probability of occurrence of defects also increases. Therefore testing of designs becomes essential in order to guarantee fault-free operation of devices.

Testing of VLSI designs involves generation of test patterns, test pattern application and identification of defects in design. In case of scan based designs, the test set size directly impacts the test application time which is determined by the number of memory elements in the design and the test storage requirements. There are various methods in literature which are used to address the issue of large test set size classified as static or dynamic compaction methods depending on whether the test compaction algorithm is performed as a post-processing step after test generation or is integrated within the test generation. In general, there is a trade-off between the test compaction achievable and the run-time. Methods which are computationally intensive might provide better compaction, however, might have longer run times owing to the complexity of the algorithm.

In the first part of the thesis we address the problem of large test set size in partially scanned designs by proposing an incremental dynamic compaction method. Typically, the fault coverage curve of designs ramp up very quickly in the beginning and later slows down and ultimately the curve flattens towards the tail of the curve. In the initial phase of test generation a greedy compaction method is used because initially there are easy-to-detect faults and the scope for compaction is better. However, in the later portion of the curve, there are hard-to-detect faults which affect compaction and we propose to use a dynamic compaction approach. We propose a novel mechanism to identify redundant faults during dynamic compaction to avoid targeting them later. The

effectiveness of method is demonstrated on industrial designs and test size reduction of 30% is achieved.

As the device complexity is increasing, delay defects are also increasing. Speed path debug is necessary in order to meet performance requirements. Speed paths are the frequency limiting paths in a design identified during debug. Speed paths can be tested using functional patterns, transition n-detect patterns or path delay patterns. However, usage of functional patterns for speed path debug is expensive because generation of functional patterns is expensive and the application cost is also high because the number of patterns is large and requires functional testers.

In the second part of the dissertation we propose a simple path sensitization approach that can be used to generate pseudo-robust tests, which are near robust tests and can be used for designs that have multiple clock domains. The fault coverage for path delay fault APTG can be further improved by dividing the paths that are not testable under pseudo robust conditions, into shorter sub-paths. The effectiveness of the method is demonstrated on industrial designs.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1 INTRODUCTION

With the advancement in science and technology, the VLSI designs today are becoming more complex day by day. Testing of manufactured chips is very essential as it directly impacts the cost. There are various types of errors – design errors, fabrication errors, fabrication defects and physical failures. Design errors could be because of incomplete or inconsistent specification, design rule violations. Fabrication errors can be caused by wrong components, improper wiring, and improper soldering that could lead to shorts of interconnects. Fabrication defects happen due to imperfections in manufacturing process. Physical failure can happen during the lifetime of a system attributed to wear-out or environmental factors or process variations. Apart from these types of errors coding bugs also can lead to incorrect design and can cause errors. Thus it becomes essential to test the devices for any defects after manufacturing.

1.1 Motivation

In this work we address two issues related to testing devices for failures – (i) large test set size which impacts test application time and test storage requirements and (ii) usage of functional patterns for speed path debug which is very expensive.

With increasing device complexity, testing complexity increases which in turn increases cost of testing. Testing scan-based designs demands smaller test set sizes because the test application time for such circuits directly depends on the number of memory elements in the schips in a given amount of time and thus fewer testers would be needed. Test pattern compaction plays a very important role in reducing the cost of testing very large designs by reducing the test application time. This also impacts the storage requirements of the test patterns in the testing hardware. In order to address this issue, we propose a dynamic compaction technique which reduces the test pattern size.

The second issue that is addressed in this work is identifying speed path failures in design to meet high performance requirements. Generation of functional patterns for all the critical paths is difficult and expensive. Functional patterns when used for speed path debug have the disadvantage of long times for debug since functional patterns typically are long and may take several cycles to reach an observed point. Therefore test patterns for path delay faults can be used to address this issue. Typically the robust fault coverage is very low for designs and thus it is not possible to cover all critical paths with robust tests. To address this issue, a method of dividing a path into sub-paths and generating tests for the sub-paths is proposed.

Section 1.2 discusses various fault models and the test generation process.

1.2 Background

1.2.1 Fault Models

Physical defects occur in chips during the chip fabrication process. There can be various types of defects like signal line breaks, lines shorted to ground, delayed signal propagation, etc. As there is a huge number of types of defects it is very difficult to generate tests for all types of defects. Defects may or may not cause device failure. A fault is a representation of a defect at the abstracted function level [2]. A good fault model reflects the behavior of the defects closely and is easy to analyze. It also should be computationally effective in fault simulation and test generation process [3]. Fault models are technology independent and thus changes in technology do not change the test generation for the faults. The different types of fault models are stuck-at fault model, bridge fault model, transition fault model, path delay fault model, open and short fault model, some of which are described below:

- Stuck-at fault model:

  Stuck-at fault model is one of the earliest fault models and has been used for a very long time. A stuck-at fault is a fault that forces a constant logic value (1

or 0) on a signal line in the circuit, called stuck-at-1 or stuck-at-0 [5]. A short between the signal line and ground line is modeled by a stuck-at-0 fault and a short between a signal line and power line is modeled by a stuck-at-1 fault. The signal line can be a primary input, primary output, inputs and outputs of internal gates, fanout stems and fanout branches. A circuit that has n lines can have 2n single stuck-at faults and $3^n - 1$ possible multiple stuck-at fault combinations [2, 4]. Stuck-at fault model is used in the current work for test pattern compaction. However, the technique is independent of the underlying fault model.

- Bridge fault model:

  Bridge fault model is another important fault model. This is a commonly occurring type of fault. A bridge fault occurs when two signal lines are shorted unintentionally [6][7]. The shorts have a finite resistance. A bridge fault affects the voltage on both the signal lines involved in a bridge. Modeling and test generation of bridges is a challenging issue and thus there are several simplified models that have been developed for test generation. The wired AND/wired OR bridge fault model a short defect between two signal lines where the bridged nodes take a logic value which is the AND (OR) of the bridged signal lines. The four-way bridge fault model models various scenarios where the effect of the bridge fault depends on the relative strengths of the gates involved in the bridge [8].

- Transition fault model:

  In the transition fault model, the time taken for a transition from input of gate to its output exceeds the specified limit [9]. The number of transition faults in a circuit is linear to the number of circuit lines. Just as the stuck-at fault model, there are two types of transition faults – slow-to-rise and slow-to-fall. A two pattern test is required to activate a transition fault, where the first

pattern sets the fault site to the initial value and the second pattern is required to launch the transition and once the fault is activated, it is propagated to an output.

- Path delay fault model:

  The path delay fault models the cumulative effect of the delays along a path in the circuit. If the cumulative delay exceeds the clock period for the path, then the test pattern that fails the chip is said to detect the path delay fault. A two pattern test is required to detect a path delay fault [1] which creates a transition at the input of the path. The transition at the input of the path is propagated along the path by satisfying necessary off-path conditions.

1.2.2 Test Generation

Test generation is the process of generating an effective set of test patterns by which a high fault coverage can be achieved for a given fault model. The main objective of test generation is to generate patterns that will detect defects in a chip. Since the number of defects in a circuit is really large and generating test for all of them would be unrealistic, test pattern generators target the faults which are an abstract representation of defects. Test pattern generation consists of the following steps –

- Fault activation:

  Fault activation sets the signal value on a line opposite to that produced by the fault at the faulty site in the faulty circuit. For example, in order to activate a stuck-at 1 fault on line l, 0 needs to be assigned to line l in order to excite the fault.

- Fault propagation:

  This is the second phase where the fault effect is propagated by sensitizing at least one path from the fault site to a primary output or a scan cell. In order to propagate the fault effect from the fault site, there could be one or more gates

through which the fault effect can be propagated. These choices are called as d-frontiers. D-frontiers are gates that have an output value of X and have a fault effect on at least one of the inputs. The fault effect is propagated by assigning non-controlling values to inputs other than the input that has the fault effect. In order to propagate the fault effect from a fault site to an output, there should be at least one x-path between the two nodes. An x-path is a path between two gates of a circuit where the output value of all the gates is X. If there exists no x-path between the fault site and any primary output or scan cell, then a test cannot be generated for the fault.

- Justification:

  Justification is the process of specifying primary input values or scan cell values in order to produce the signal values required for fault activation and fault propagation. The justification process is carried out by assigning necessary values at the inputs of gates for which the output values are specified to be 1 or 0 during fault activation and propagation, but are not implied by the input values. The set of all such gates is called as j-frontier.

The effectiveness of the test set produced is measured in terms of fault coverage for the given fault model, the number of test patterns generated. The number of test vectors generated directly impacts the test application time.

1.2.3 Design for Testability

Test costs can be attributed to test pattern generation, fault simulation, generation of fault location information, test equipment cost, cost related to testing process which is the time required to detect and/or isolate a fault. The costs associated with testing could be high and can even exceed design costs. In order to limit the testing costs and to simplify testing a device, design for testability (DFT) techniques are used to ensure that a device is testable. Controllability and observability play an important role in generating a

test for a circuit. Controllability is the ability to obtain a required signal value at each gate in a circuit by setting certain values on the circuit's inputs. Observability is the ability to determine a signal value at on any gate of the circuit by controlling the inputs of the circuit and observing its outputs.

Basic testing infrastructure consists of three components: circuit under test, automatic test equipment (ATE) and ATE memory to store test patterns and expected test responses obtained by automatic test pattern generation (ATPG) tools as shown in Figure 1.1. In order to test a given circuit (CUT), test patterns are applied at the inputs of the circuit and the output values obtained are compared with the test responses stored in the ATE memory. A circuit is considered to be fault free if the output response of the circuit matches with the output response stored in the ATE.

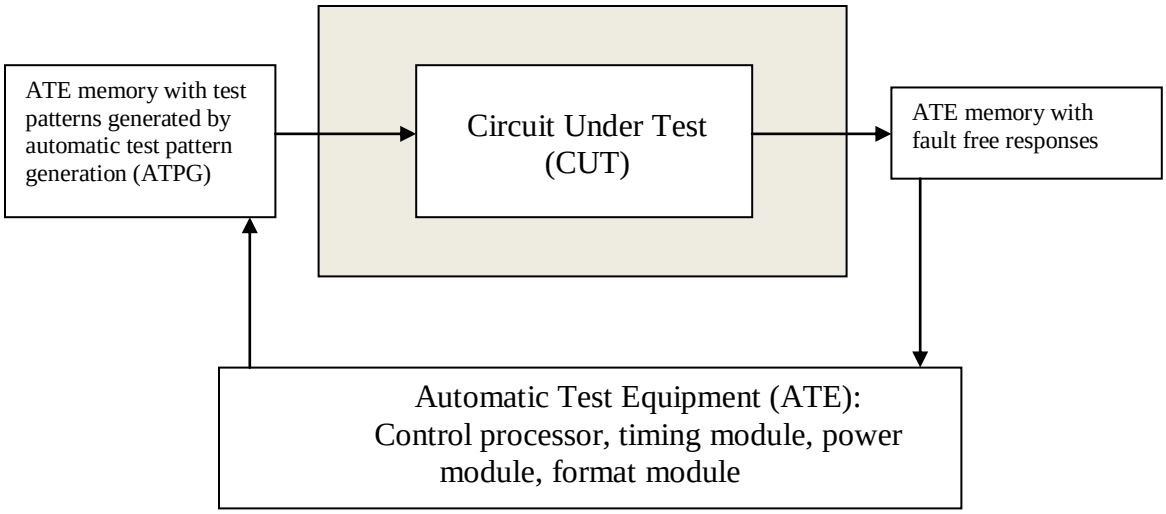


Figure 1.1 Manufacturing test of a circuit [54]

Testing combinational circuit is easier than testing sequential circuits since the primary inputs can be set to required values and primary outputs can be observed. Testing

sequential designs is challenging because it takes several test cycles to get desired values on latches or flip-flops. This can also lead a large number of faults being untestable. In order to cope with low controllability and observability of sequential designs, design for testability (DFT) techniques is employed. The testability of a device improves with DFT methods since it enhances the controllability and observability of sequential elements. This is achieved by introducing scans in the design.

Scan design is the most widely used structured DFT method that is used to improve the controllability and observability of the storage elements in sequential design. This is achieved by converting the sequential design into a scan design and the design is operated in functional mode and test modes. In functional mode, the circuit operates in functional configuration by turning off all the test signals. During test mode, a test mode signal is applied which converts all the flip-flops in the design into one or more shift registers called scan chains. This improves the controllability of the flip-flops as they can now be set to desired state during the test mode by shifting in appropriate values.

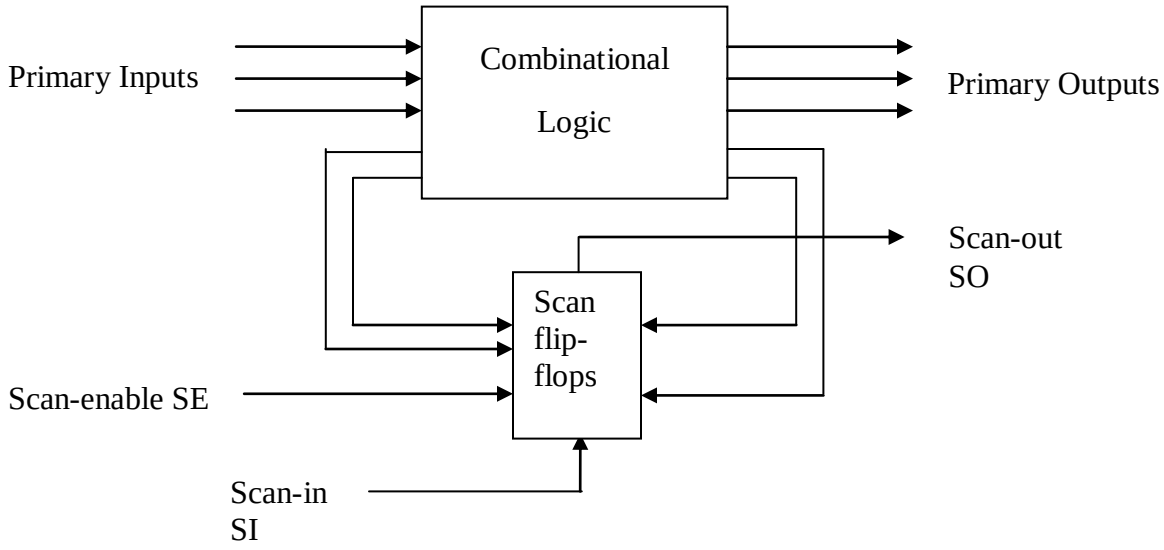


Figure 1.2 Scan based design

The observability of the flip-flops is also enhanced since the states of the flip-flops can be observed by shifting out the content of the scan chains.

There are two types scan designs: full scan and partial scan designs. In full-scan design all the storage elements are converted into scan cells and combinational ATP can be used for test generation. In partial scan designs a portion of the storage elements is converted into scan cells and sequential ATPG is required for test generation since the design is still sequential since the complete design is not converted into scan design. The designs used in the current work are partially scan designs.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 2 presents a dynamic compaction technique to address large test pattern size. Chapter 3 presents path delay fault test generation methodology in partially scanned designs and method to improve robust fault coverage. Chapter 4 draws the conclusions.

CHAPTER 2 INCREMENTAL DYNAMIC COMPACTION TECHNIQUE

In this chapter we review various compaction techniques in the literature and present an incremental dynamic compaction approach to reduce test pattern size. In order to address the long run times of the approach we also present a static untestability and reasoning analysis method based on [15, 56]. Experimental results on industrial designs demonstrate the effectiveness of this technique.

2.1 Introduction

Earlier the major focus of research was to generate a complete test set efficiently for a given design. Several test generation algorithms have been proposed over the years [11-15]. Over the past two decades, the effort is directed towards minimizing the size of the test pattern set produced. The problem of finding the minimum test size for an irredundant combinational circuit by itself is proven to be NP-hard [17]. There are several compaction algorithms in the literature that are based on various heuristics, for example – test generation based on independent fault set and compatible fault sets[19-21], double detection[19, 22], reverse order fault simulation[23], rotating backtrace[20]. Every new methodology is targeted towards getting better test size reduction and thus closing the gap further to the lower bound.

The size of the test set directly impacts the test storage requirements and test application time, especially for circuits using scan design. The test application time is directly proportional to the product of the number of test patterns and the number of scan cells in the longest scan chain [19]. This necessitates generation of small test sets.

The complexity of the compaction process plays an important role in test compaction. There are computation-intensive procedures proposed in the literature that produce minimal size test sets close to the lower bound [21, 24, 25]. For instance, in [9] tests are generated repeatedly which can detect several faults at the same time so as to

replace previous tests found. Though these methods produce small test sets, they are not suitable to large designs. Methods based on simple and efficient heuristics can be found in [20, 23, 26].

Test pattern compaction is aimed at generating a pattern set in which a test detects as many faults as possible. There are two ways of compaction – static and dynamic and are described below.

- Static Compaction

  Static compaction is applied as a post processing step to already generated test sets, to reduce the test set size further and therefore is independent of the test generation process. Static compaction is performed after all the patterns are generated and this is independent of test generation.

- Dynamic compaction

  Dynamic compaction is incorporated within the test generation process where a test cube is generated for a fault and the generated test cube is added as constraints to the next targeted fault. The advantage of dynamic compaction over static compaction is that it reduces the time required for post-processing step for compacting patterns. The dynamic compaction begins with a fault which is on top of previously ordered fault list, called as primary fault. The primary fault is targeted for test generation and if a test is generated for the fault, another fault called secondary fault is picked and a test generation for the fault is attempted. The test generation tries to generate a test for the secondary fault with the primary input values and scan cell values specified by previously generated test vector. The test generation for the secondary fault specifies only the unspecified values remaining from the previous test vector. This process is repeated for all the remaining faults remaining in the fault list or all the inputs are specified [20]. The unspecified inputs of the resulting test vector at the end are then random filled with 1s and 0s. This process is repeated with a different primary fault and the

entire process of is repeated with remaining secondary faults and the process continues until all the faults in the fault list are tried as primary faults.

2.2 Review of Previous Work

In this section below, some of the static and dynamic compaction methods are reviewed from literature.

2.2.1 Reverse Order Fault Simulation

Reverse order fault simulation is a very simple and effective method of compacting test patterns. The test patterns are simulated in reverse order of test generation, wherein a test that was generated later is simulated earlier [5]. If a test does not detect any new faults when it is simulated, then the test is dropped from the test set. Reverse order fault simulation is a widely used static compaction technique [23, 28] suitable for combinational ATPG. But reverse order simulation is most beneficial when applied to effective tests. Effective tests are obtained by simulating a test set in the order in which it was generated with fault simulation used for fault dropping. A test generated by deterministic test generation method is fault simulated to drop the faults it detects and thus every new test detects faults not detected by previous tests. If the test set is generated by non-deterministic method like random test generation, fault simulation with fault dropping needs to be additionally done so as to get effective tests. Once the effective tests are obtained, these can be used for reverse order fault simulation. However, if the additional pass of fault simulation is not applied to non-deterministic patterns, the reverse order simulation would only identify effective tests of the test patterns in the reverse order which would not achieve as much test size reduction as obtained when applied to effective tests [29].

2.2.2 Forward-looking reverse order fault simulation

Forward-looking reverse order fault simulation [29] is an improvement over reverse order fault simulation. This method records the information about the first test in the test set that detects a fault for the first time. This is obtained by simulating the test set in the order in which it was generated combined with fault dropping. With this information, tests can be further dropped during reverse order fault simulation. This can be illustrated as follows. Consider a test set T = $\{t_0, t_1, \ldots, t_{n-1}\}$ and let F = $\{f_0, f_1, \ldots, f_{p-1}\}$ be the set of target faults. During forward looking reverse order fault simulation, every test in T is simulated in the reverse order with fault dropping similar to reverse order fault simulation. Before simulating a test t_i, it is first determined whether it is necessary to simulate a pattern, and if it is not necessary then it is dropped without simulating the pattern. The decision to ascertain whether a test t_i is necessary or not is based on the information of the first test that detects each test during the original order. Let the detection vector index of a test that detects a fault for the first time during forward order be denoted as detindex (f_i). This can be obtained during fault simulation of F. During forward looking reverse order fault simulation, before simulating a test, the detection vector index of every fault $fi \in F$, is compared against the current test index. If the current test index i is greater than the detection vector index of all the faults in F, then the test is dropped without simulating it because the faults that will be detected by the test t_i will be detected by later tests during reverse order fault simulation. If a fault $fi \in F$ is such that the detection vector index of f_i is the same as current test index, then the pattern is simulated because there will be no further tests that would detect f_i

2.2.3 Reverse Order Test Compaction (ROTCO)

Reverse Order Test Compaction[27] is a method similar to reverse order fault simulation in [23] but with the difference that the test vectors are allowed to be "modified" in the process, thereby increasing the possibility of detecting faults that were

detected by earlier test vector which could potentially result in a smaller test set. This is based on the following reason. During the process of test generation, after every test is generated, fault simulation is performed and all the faults detected by the test are dropped. The faults dropped include the target faults for which the test was found and the faults that were detected additionally due to random filling of unspecified bits in the test vector. Only small number of faults is typically detected by the specified bits of the last vectors in the test set which are detected for the first time by the test vectors. There are a large number of unspecified inputs which can be specified in such a way that the faults detected by test that are generated earlier during test generation would be detected by the later tests. Therefore, the tests generated earlier during test generally could be possibly dropped. The order in which the vectors are processed is in the reverse order of test generation.

Table 2.1 Before and after reverse order test compaction [27]

Before reverse order test compaction		After reverse order test compaction	
Test	Fault	Test	Fault
t_1	f1, f2	t_1	f1, f2
t_2	f3	t_{21}	f2, f3
t_3	f4	t_{31}	f1, f4

The complexity of ROTCO is much less than the complexity of complete test generation because the specified values in a test vector are left unchanged.

The test compaction procedure can be explained with an example given in [27]. Consider an irredundant circuit with four faults $\{f_1, f_2, f_3, f_4\}$. Let the faults be detected by three test vectors as shown in the Table 2.1 above. The test t_3 can be extended into a

test t_{31} which detects faults that are detected by earlier vectors by using the unspecified inputs of t3. Similarly if t2 can be extended to t_{21} such that it can detect f2 apart from f3, then t1 can be dropped from the test set. This is shown in Table 2.1. This would not be possible with reverse order fault simulation.

The following information is required for ROTCO-

1) The test set $T = \{t_1, t_2, \ldots, t_k\}$.
2) Fault detected by each test vector, which are not detected by earlier test vectors. This is obtained by fault simulation which drops faults that are detected by a test vector. Let F_i be the set of faults detected by a test t_i
3) In order to distinguish the inputs specified by test generation from the inputs specified by random filling of unspecified bits, the unspecified bits filled by random filling should be given as x_0 or x_1 (x_0(x_1) stands for input randomly set to 0(1))

The tests are considered in reverse order. All the inputs with value x_0 or x_1 are changed to x. The fault lists F_1, F_2,…. F_{i-1} are considered in increasing order of the size of the fault lists. The order within a fault set is not restricted. The test generation starts with specified inputs specified by test vector t_k until t_1. After attempting to generate a test t_{k1} which possibly detects other faults apart from F_k, the unspecified inputs that are still x are changed back to their original value (x_0 or x_1). Since some of the unspecified values might get specified in the test generation, some of the faults in F_i may no longer be detected. So, fault simulation is carried out for the modified t_i and if the faults in F_i are no longer detected by the test vector, the test vector is restored to its original values. This ensures that the fault coverage is maintained as before. If the modified t_i detects all the faults in F_i, it replaces the original test in the test set. All the faults in F_1, F_2,…, F_{i-1} are fault simulated and all the faults that get detected are removed from their respective fault lists and are added to fault list F_i. If a fault list becomes empty in this process, the

respective test is dropped from the test set. This process is repeated for all ti in test set T. The remaining vectors left after this procedure is the reduced test set for the circuit.

The test set size reduction when ROTCO is applied after test vectors generated using PODEM algorithm [13] is up to 56% on ISCAS 85 circuits and PLA benchmark circuits in [27]. Further when ROTCO was performed over COMPACTEST [20] with reverse order fault simulation, the test set size reduction was up to 20%. This is because the test sets produced by COMPACTEST is already compacted to a great extent and the number of faults detected by a test vector on an average is very large. This method is applicable for combinational patterns only.

2.2.4 COMPACTEST

In COMPACTEST [20], the authors propose a test generation method which uses independent faults for fault ordering, a test compaction method and a dynamic line justification method to generate tests that detect large number of faults and hence reduce the test set size. The importance of independent fault sets in the reduction of test size has been established in [31, 32]. An independent fault set is defined as a set of faults for which there exists no test that detects any pair of faults in the set. Independent fault sets are very useful for test generation because the smallest test set size cannot be smaller than the size of the largest independent fault set. The problem of computing the set of independent faults of maximum cardinality in a circuit is np-hard [16]. Algorithms to compute the set of independent faults of maximum cardinality (MIFS) in a circuit is discussed in [20, 32].

During the pre-processing step of COMPACTEST, an ordered fault list is derived using MIFS for fanout free regions (FFRs) for collapsed fault set. The largest MIFS is placed at the top of the fault list, which is followed by the next largest MIFS and so on. The remaining collapsed faults which do not belong to any MIFS are added to the end of the fault list. During the computation of MIFS for the circuit, information regarding the

subsets of faults that can be potentially tested by the same vector is gathered. Basically, every fault f in an MIFS of an FFR is associated with other faults in the fanout-free region which can potentially be tested along with the fault f.

2.2.4.1 Maximal Compaction Procedure

The compaction procedure of COMPACTEST is described as follows. The fault at the top of the ordered fault list is selected to be targeted. This is called the primary fault. It is attempted to generate a test for the primary fault if possible. The information gathered during the pre-processing stage to find faults that can be targeted with the same fault in the FFR region to generate a test is utilized during the test generation process for the primary fault. As a result, a test vector is generated which detects a primary fault along with possibly additional faults in the FFR region. Once the test vector is generated, it is then maximally compacted to maximize the number of unspecified values in a test vector before targeting the next fault. The maximal compaction happens as follows. A primary input p is selected from the set of specified values in the test vector whose value is specified for the first time by the fault f targeted most recently. The value specified for the primary input is then complemented and implication is performed for the modified test vector. This is to ensure that the modified test vector still detects the fault f and if it still detects the fault f, the primary input p is marked, and otherwise it is left unmarked. The value of the primary input is restored to its original value as in the original test vector. This process is repeated for all the primary inputs specified for the first time during the test generation for f. Once all the primary inputs are tried, the primary inputs that were marked during the compaction procedure are then unspecified. This can be explained by an example from [20].

Consider a stuck at 0 fault on line a in the circuit shown in Figure 2.1. Let (1111) be the test vector generated for a stuck-at-0. The maximal compaction procedure starts by complementing the value of the primary input a, from 1 to 0. Implication is performed to

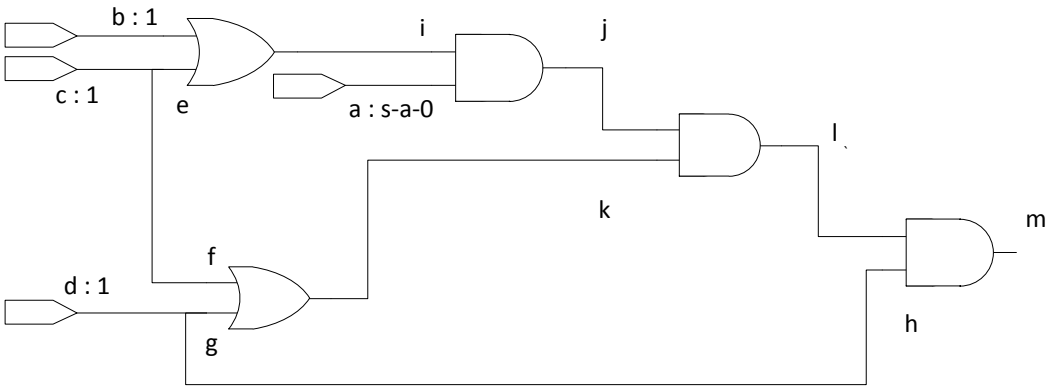


Figure 2.1 Example for maximal compaction [20]

check whether the fault a stuck-at 0 is still detected under the test (0111). Since a stuck-at 0 is not activated in this case, the input a is left unmarked and the value of input a is restored to 1. The next primary input b is considered and the value of 1 is flipped to 0 and implication is performed. Since the fault a stuck-at-0 still gets detected under the modified vector (1011), input b is marked. In a similar way, input is complemented and the same check is performed and since 1101 detects a stuck-at 0 it is also marked. Input 4 is left unmarked because 1110 does not detect a stuck-at 0. Thus the compacted vector is (1xx1), where inputs a and d are left unmarked and inputs b and c are marked.

The number of unspecified values obtained in the maximal compacted vector is independent of the order in which the inputs are processed since every input is processed beginning from the same initial state where all the other specified inputs in the original test vector remain the same except the current primary input on which the check is being performed. The resulting compacted vector contains the original vector, but not necessarily the fault for which the original test vector was generated will be detected by the specification of the unspecified values later.

After the primary fault f is targeted and the above compaction procedure is performed, the next fault in the ordered fault list picked for target for test generation. This fault is called secondary fault. The test generation process begins with the specified values in the test vector generated for the previous targeted fault and a test is generated for the secondary fault if possible by assigning values to only the unspecified inputs of the test vector. Once a test is generated for the secondary fault, additional faults present in the FFR region of the targeted secondary fault are tried for test generation to maximize the number of faults detected by the test vector. After this, the specified inputs of the test vector that were assigned during test generation of the secondary fault, are processed for maximal compaction. If the test generation for the targeted secondary fault was not possible, all the primary inputs specified during the process of test generation for the secondary fault are unspecified.

This process is repeated until either all the faults in the fault list have been tried as secondary faults or all the inputs are specified in the test vector. Once either of the conditions is met, if there are unspecified inputs left, then they are randomly specified and the test vector is fault simulated. All the faults detected by the test vector are dropped from the fault list. The process of test generation and maximal compaction is repeated with the next primary fault which is at the top of the fault list. This process continues until either all the faults have been tried or the fault list is empty.

2.2.4.2 Rotating backtrace

The backtrack process is modified in way such that different paths are sensitized each time a line needs to be justified on a line. In the process, different faults along the various paths are potentially detected by the test vector generation. Rotating backtrace works as follows. Every gate is associated with a counter which is initialized to 0. During backtrack, whenever the output of a gate needs to be justified to a value which can be obtained by setting any of the inputs to the controlling value, the backtrace procedure

selects an input which is given by the counter. The counter is then incremented modulo n, where n is the number of inputs to the gate. Thus, each time a value needs to be justified at the output of a gate; it is done by setting a different input line. If the input selected by the counter already is specified, another input is selected and the counter is not incremented. In addition to the rotating backtrace using a counter to select an input, controllability measures can also be used to bias the selection of inputs.

The experimental results with the heuristics of COMPACTEST applied on ISCAS-85 and ISCAS-89 benchmark circuits demonstrate a 50% reduction in the test set size on an average when compared to the test patterns generated on a test generator using PODEM and a deductive fault simulator, with reverse order fault simulation performed at the end of both the methods. There is a 2X increase in run-time with the COMPACTEST procedure of compaction. There is not much reduction in size of the test set when reverse order fault simulation is performed on the test set produced by COMPACTEST. This is illustrated the fact that COMPACTEST produces test patterns which are irredundant.

2.2.5 Double Detection

In Double detection [22], a dynamic compaction method is proposed. Double detection is based on using the unspecified input values during test generation process to increase the possibility of obtaining and then later dropping redundant test vectors. In order to accomplish this, a fault is detected twice before it is dropped from the fault list. For every fault in the fault list the following information is recorded:

1) The fault index
2) The information about number of times a fault is detected is stored in a variable called *check* and is coded as follows-

   check = 0 implies fault is not detected yet,

   check = 1 implies the fault has been detected once,

   check = 2 implies the fault has been detected at least twice

check = 3 is when a fault is aborted during test generation due to backtrack limit

3) The test vector that detects a fault for the first time is stored in a variable called *tvector*

Deterministic test generation happens until there exists a fault with *check* is 0 exists. A fault f_1 with *check* = 0 is chosen as a primary fault that is to be targeted. After a test is generated for the primary fault f_1, another fault f_2 is selected. f_2 is called secondary target fault. The test vector that was generated for f_1 is attempted to be extended by specifying additional inputs and is possible a test is generated for fault f_2. This process is repeated until the test vector is fully specified or there are no more additional faults can be detected. Fault with *check* = 0 are selected first as secondary faults, followed by faults with *check* = 1. A fault is aborted when it is a primary fault there is no test that can be obtained within the given backtrack limit. The test generation procedure in [22] differs from [20] in which the faults with *check* = 1 are not selected as secondary faults and fault simulation is not performed on such faults.

After a test is generated, the test vector is simulated and the number of faults detected for the first time by the test are stored in a variable one_check. When additional vectors are generated, if the variable *check* of a fault increases from 1 to 2, one_check for the test vector is decreased by one. Therefore, one_check represents the number of faults detected only by the test vector. If one_check for a test vector is greater than 0, it means that the test vector is essential and cannot be dropped. If one_check is 0, then the faults detected by the test vector are also detected by other vectors. This is shown in Table 2.2. Once test generation is completed, the redundant test vectors are reduced as follows. Vectors with one_check is greater than 0 are simulated first since these vectors detect faults that cannot be detected by any other vector in the test set. Then the remaining vectors whose one_check is 0 are simulated in reverse order compared to their original order in which they were generated. This way redundant test vectors are dropped and

Table 2.2 Example of double detection [22]

Test vector	Faults detected	one_check
t_1	f_1, f_3	1
t_2	f_1, f_2	0
t_3	f_2, f_4	0
t_4	f_4, f_5	1

additional redundant vectors can be dropped by recalculating the variables *check* and *one_check* and *tvector*. This process is repeated until one_check for all test vectors becomes greater than 0. This method along with dynamic fault ordering and rotating backtrace [22] was compared against test patterns generated by the method used in [30] where redundant elements are removed from circuits using test pattern generation. The test pattern size is 50.7% smaller than [30] and the CPU time required is 3.6X times the time in [30].

2.2.6 Essential Fault Reduction Method

Hamzaoglu and Patel in [47] propose an essential fault reduction (EFR) technique for generation of compact test sets in combinational circuits for single stuck-at faults and a heuristic for estimation of minimum stuck-at fault test set size. These algorithms together with dynamic compaction method of [22] are incorporated into the test generation system in [48]. This method found better lower bounds and generated smaller test sets than the methods of [49, 50]. EFR algorithm is an improvement over Two_by_One (TBO) [19, 51] and Essential Fault Pruning (EFP) algorithms [52].

Some of the definitions used in [47] are defined as follows. A test vector is called an *essential vector* if it detects at least one fault that is not detected by any other test vector in the test set. An *essential fault* of a test vector is a fault that is detected only by

the test vector in the test set. A test vector is considered *redundant* relative to a given test set, if it does not detect any essential faults. An essential fault f of a test vector t_i is said to be *pruned* if a test vector $tj \neq ti$ in the test set is replaced by a new test vector t_j' that detects the essential fault *f*, essential faults of t_j and faults detected only by t_i and t_j.

A pair of faults is *compatible* if they can be detected by the same test vector. If the two faults cannot be detected by the same vector, they are called as *incompatible*. An incompatibility graph for a given set of faults is defined as $IG(FS) = (V, E)$ where $V = \{v_i = f_i | 1 \leq i \leq n\}$ and $E = \{e_j = (v_k, v_l) | v_k$ and v_l are incompatible, $1 \leq k \leq n$ and $1 \leq l \leq n\}$[18, 52, 19, 21].

Once the initial test set is generated, EFR algorithm is used repetitively to prune essential faults of each test vector as much as possible. If all the essential faults of a test vector are pruned then a test is redundant and can be dropped from the test set. The TBO algorithm compacts tests by replacing two test vectors with a new one. This is accomplished by finding a test vector that detects the essential faults of both the vectors and the faults detected only by the two vectors. If this is not achievable by TBO, it may be achieved by three_by_two algorithm which replaces three test vectors with two new ones if possible. However, the *N_by_M* algorithm could be computationally expensive since in the worst case, it may involve $O(T^N)$ checks where *T* is the number of initial test vectors generated.

EFP algorithm reduces the number of tests by pruning the essential faults of each test vector and if all the essential faults of a test vector are pruned, the vector can be dropped since it is redundant. EFP achieves better performance than TBO since it allows a test vector to prune its essential faults by replacing more than one vector in the test set. EFP tries to generate a test vector for $O(FXT)$ fault sets, where *F* is the number of essential faults and *T* is the number of test vectors generated initially. Generally *F* is larger than *T*, therefore EFP is more expensive than TBO. For N>2, the N_by_M algorithm is however more expensive than EFP.

The limitation of TBO and EFP approaches is that they carry out a localized greedy search by focusing on removal of one test at a time by pruning its essential faults. If the algorithm fails to prune even one of the essential faults for a test t_i, the original test is recovered. This restriction may prevent elimination of another test vector t_j from the test set because the essential faults of t_j may be incompatible with essential faults of all other tests in the test set. However, the essential faults of t_j may be compatible with those of t_i except one of them. If t_i were allowed to prune the incompatible essential fault then the essential faults of t_j can be pruned and thus the test can be dropped from the test set.

EFR algorithm overcomes the limitation of TBO and EFP by reducing the number of essential faults for a test vector by pruning the essential faults as many as possible. The method does not stop when it fails to prune one of the essential faults and goes ahead pruning other essential faults of the test vector. This can be explained with an example in [47] shown in Figure 2.2. Let the test set $T = \{t_1, t_2, t_3, t_4\}$ detect faults as given in the Figure 2.2. The incompatibility relation is also shown in Figure 2.2. TBO and EFP methods cannot reduce the test set size. EFR can reduce the size of the test set by replacing test vectors t_1 with t_1' which detects f_2 and f_3, t_3 with t_3' that detects f_1, f_5 and f_6 and t_4 with t_4' that detects f_4 and f_7. It can be observed that t_2 now is redundant and thus it can be dropped.

EFR can be used iteratively for further compaction. EFR has a worst case complexity as that of EFP and if used iteratively, the worst case complexity is $O(FXTXI)$, where I is the number of iterations. The execution time is reduced based on a new incompatibility relation for stuck-at faults. This is based on the fact that even though a fault is pair-wise compatible with all the faults in a given fault set, it may be incompatible

if the faults are targeted together. The incompatibility relation is defined as follows. For a given set of faults $FS = \{f_i | 1 \leq i \leq n\}$, the new incompatibility graph is defined as $IG(FS) = (V, E)$ where $V = \{v_i \subset FS | 1 \leq i \leq n\}$ and $E = \{e_j = (v_k, v_l)|$ the faults in v_k are incompatible with the faults in v_l, $1 \leq k \leq n$ and $1 \leq l \leq n\}$. This is used to speed up EFR algorithm. The iterations of the EFR algorithm are stopped as soon as the minimum test set size is reached instead of iterating a pre-determined number of times.

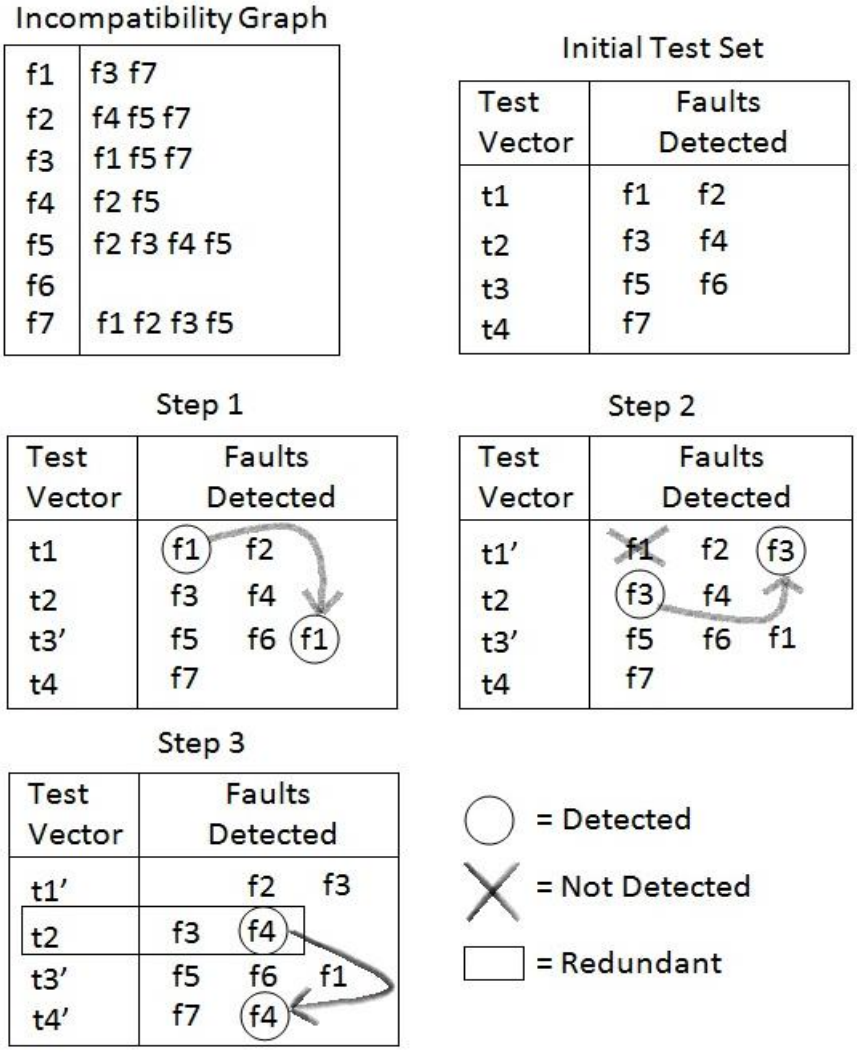


Figure 2.2 EFR example [47]

2.2.7 Dynamic Test Vector Compaction

The dynamic compaction approach in [53] differs from traditional dynamic compaction approach. Typically in dynamic compaction, once test is generated for a fault, the next fault is picked from the fault list and is targeted for test generation. In [53] however, after a test is generated for a fault, instead of selecting the next untested fault from the fault list and generating a test for the fault, a compaction procedure called COMPACT is used. In the compaction procedure, a test T_i is compared with other tests generated earlier and if T_i is compactable with any of the previous tests T_j, where j<i then T_j is replaced with $T_i \cap T_j$. If T_i is not compactable the test is added to the test vector set. This compaction is performed repetitively and test compaction is obtained.

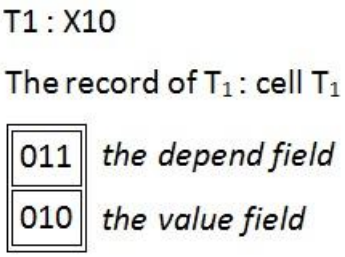


Figure 2.3 Test T_i : X10 recording [53]

An efficient data structure is used for the comparison of test vectors. The data structure contains an N dimension table, where N is the expected number of tests. Each test vector is identified by index as T_i and belongs to a cell sequence of type *test-vector*. The *i*th test vector can be obtained from the table TEST as TEST[i]. Each T_i record is a cell which contains two fields called *depend* and *value* both being integer type. The first field *depend* specifies the "significance" of a PI, i.e., if the value is 0 or 1 then it is considered to be significant and *depend* is set to 1 and if value is X then *depend* is 0. The second field *value* contains the test vector values for the corresponding PIs and if *value* is

1, it is always significant and if it is 0, then *depend* field is looked-up. This is illustrated in Figure 2.3. An integer consisting of 32 bits can store upto 32 PI values. The values of PIs are stored with two integers as explained above.

In order to check for compactability between a test generated *test* and for a test generated previously TEST[i] from the table, the *depend* fields are compared first and then the *value* fields are compared. Once the comparisons are done and if the test generated *test* is compactable with the test TEST[i], the test is compacted with TEST[i]. In this way, the test generated *test* is compared with all the previously generated tests. If the test is not compactable with any of the tests, then the test is added to the table TEST. Before picking the next fault for targeting, the compacted result or the generated test *test* is fault simulated.

The advantage of this method is it is very simple and the memory requirements are minimal since the values of the inputs are stored in two integers. The compaction achieved by this method is 40% for smaller circuits and about 50% for larger circuits (over 1000 gates) when compared with tests generated using PODEM.

2.3 The Proposed Method

In this section, we present an incremental dynamic compaction approach that uses a cube merging mechanism with dynamic compaction. We also propose a reasoning analysis approach to drop redundant secondary faults from being targeted again in order to improve the run-time.

2.3.1 Motivation

The importance of minimum size test sets has been observed in the previous section and we have reviewed methods from literature to achieve this. Test set size impacts the test storage requirements and test application time. Test application time is directly proportional to the product of the number of tests and the number of scan cells in

the longest scan chain in the design. In this work, an incremental approach is proposed which addresses the issue of growing test pattern size.

The fault coverage curve for the designs typically ramps up vigorously in the beginning due to random fault detections and slows down ultimately becoming almost flat after certain fault coverage is reached. Therefore, towards the tail end of the fault coverage curve there are very few faults detected per pattern. The existing test generator uses cube merging initially which is a greedy way of compaction and we use focused dynamic compaction after a threshold fault coverage after which the ramp slows down so as to utilize the benefit of random detections in the initial portion of the fault coverage curve and get the benefit of dynamic compaction in the tail of the curve.

In this work we present an incremental dynamic compaction method and propose methods to reduce run-time. The contributions of the work are:

1) Propose a test compaction approach that utilizes the benefit of both cube merging and dynamic compaction by initially performing cube merging/greedy compaction and later switching to dynamic compaction in the tail of the fault coverage curve.
2) Static untestability analysis to identify and skip targeting secondary faults that cannot be activated and/or propagated with a given testcube.
3) Reasoning analysis to identify untestable faults during dynamic compaction which are untestable independent of constraints set by dynamic compaction

2.3.2 Preliminaries

In this section the fault coverage curve is discussed and cube merging approach is described that is used during the initial ramp of the fault coverage curve. The basic incremental dynamic compaction with static untestability and reasoning analysis to reduce run time for dynamic compaction is presented later.

2.3.2.1 Cube merging Method

In this section we describe the cube merging technique existing in the test pattern generator similar to [53] which is used to compact patterns on the fly as they are generated. Once a test is generated for a fault, the raw pattern is stored in a bin. A bin is a data structure to hold a compacted pattern resulting by merging of raw patterns. The number of bins is programmable and can be specified at the beginning of ATPG. A test cube is a pattern generated by the ATPG and the unspecified values are not specified yet. A new pattern generated is merged with the first available bin with which the test cube is compatible. In order to find whether a pattern is compatible with the pattern in a bin, the scan cell values and the primary input values are compared. If there are no conflicting values then the pattern is merged with the pattern in the bin.If there is no such bin available with which the raw pattern can be merged, the pattern is stored in the next available empty bin. If there is no empty bin remaining, then a bin that has the most number of tests compacted is written out after filling the unspecified bits with 1s and 0s. The compacted pattern that is written out is simulated and all faults that are detected by the pattern are dropped and are not targeted later. The overall flow of cube merging is illustrated as a flowchart in Figure 2.4.

The process of cube merging can be explained with an example as shown in Figure 2.5. Consider a two bin datastructure for storing test cubes. The number of testcubes merged is initialized to 0 for every bin. When testcube t_1 is generated, it is stored in the first bin, i.e. bin 0 as shown in Figure 2.5(b), since it is the first compatible bin available and the number of testcubes merged is updated to 1. When testcube t_2 is generated, since it is compatible with the testcube in the first bin, it is merged with the testcube already existing in the bin and the merged pattern is stored in the bin. The number of testcubes merged is updated to 2 for bin 0. When testcube t_3 is generated, since it is not compatible with the pattern in bin 0, it is stored in the next compatible bin, i.e. bin 1 as shown in Figure 2.5(b). The number of testcubes merged is changed from 0 to 1

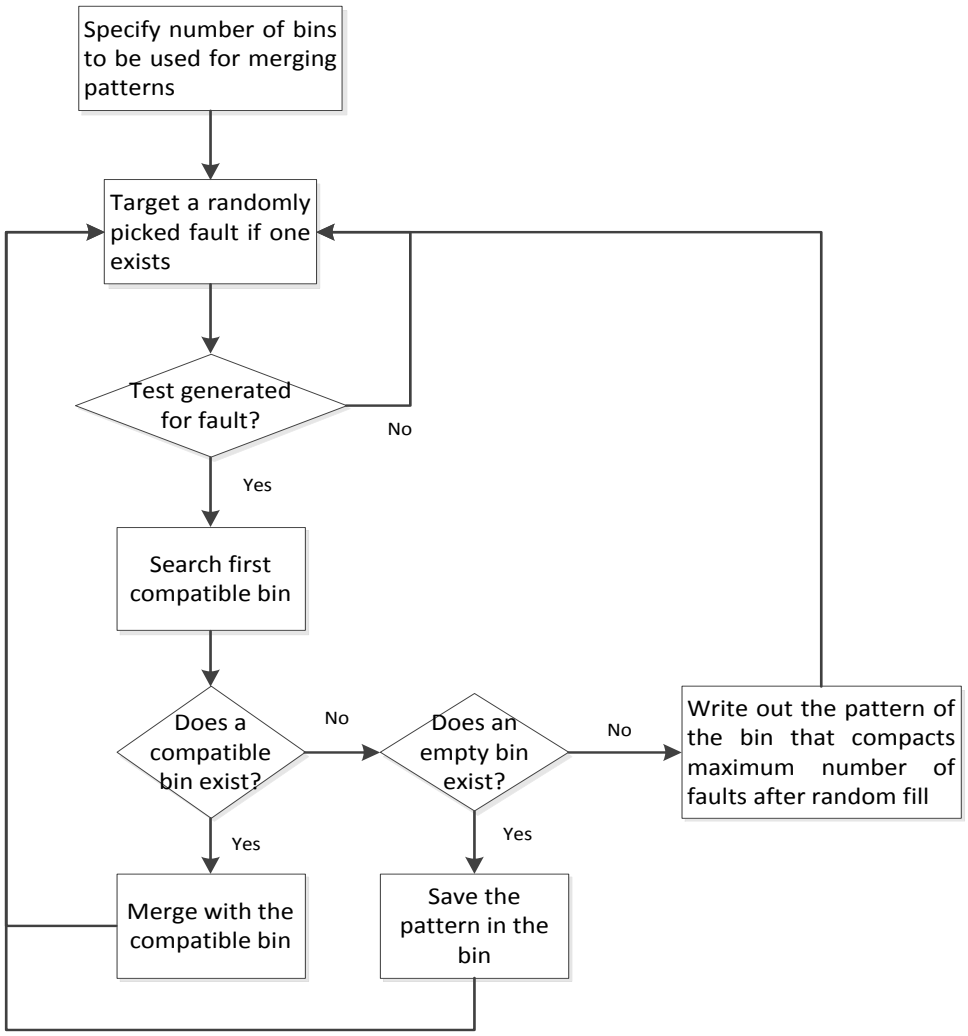


Figure 2.4 Flow diagram for cube merging technique

for bin 1. When testcube t_4 is generated, it is incompatible with both bin 0 and bin 1 and since there are only two bins available, testcube in bin 0 is picked to be fault simulated since it has the maximum number of testcubes merged. Once the testcube is selected for fault simulation, the unspecified bits are random filled and the pattern is then fault simulated and the detected faults are dropped from the fault list. The testcube t_4 is now stored in bin 0 as shown in Figure 2.5(c).

Test Index	Testcube
t_1	1X0X1X
t_2	110X1X
t_3	10X10X
t_4	11010X

(a) Test Cubes Generated

Bin Index	Merged testcube	Num cubes merged
0	110X1X	2
1	10X10X	1

<- Fault simulated after random fill

(b) Bin data Structure - 1

Bin Index	Merged testcube	Num cubes merged
0	110X1X	2
1	10X10X	1

(c) Bin Data Structure - 2

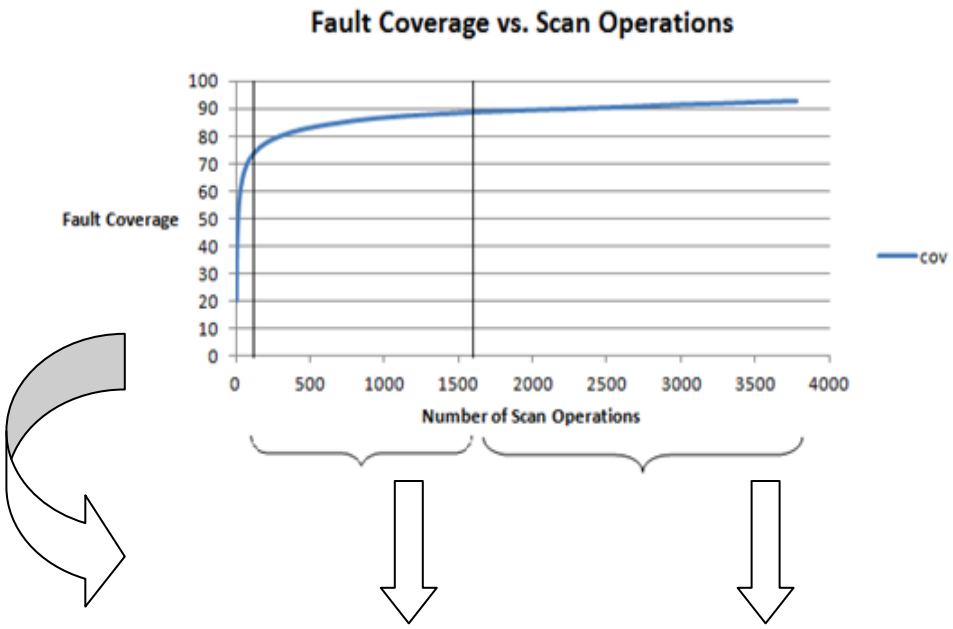
Figure 2.5 Example for Cube Merging

The next section describes fault coverage curve when the cube merging technique is used and the motivation to use dynamic compaction so as to generate compatible test cubes and reduce the pattern count.

2.3.2.2 Fault Coverage Curve

A fault coverage curve plots the fault coverage with respect to the number of tests or equivalently scan operations. A fault coverage curve when the above cube merging

method/greedy method of compaction is used is shown in Figure 2.6. Initially there is huge number of detections during fault simulation. The random filling of unspecified values in the pattern causes a large number of faults to be detected during fault simulation. There is a rapid increase in the fault coverage within very small number of scan operations.



	Initial	Ramp	Tail
Behavior	Large random detects on fault simulation	Effect of cube merging and random detects on fault simulation	Few random detects on fault simulation, patterns not merge-able
Solution	No compaction needed	Cube merging performs well	Cube merging not suitable and needs compaction friendly patterns

Figure 2.6 An Example Fault Coverage Curve

Compaction does not really have an impact on the number of fault detections. This is shown as the initial region of the fault coverage curve in Figure 2.6. The coverage ramp slows down after a few scan operations and there is lesser number of detections during

fault simulation. The compaction of patterns, by cube merging together with detections during fault simulations contribute to the fault detections in this region as shown as the ramp region in Figure 2.6. In the tail portion of the curve there is very little random detection during fault simulation. The patterns generated are not compaction friendly and thus very few patterns are compacted in every bin. Therefore, in order to generate compaction-friendly patterns dynamic compaction would aid in better compaction of patterns by generating a pattern incrementally.

In the next section, the incremental dynamic compaction procedure is described.

2.3.3 Incremental Dynamic Compaction

In the incremental dynamic compaction approach we present in this section, dynamic compaction is used on top of cube merging. Unlike various methods in the literature where dynamic compaction starts from the very beginning of test generation, in the incremental approach, we use the benefit of random detections by fault simulation in the initial region and cube merging to achieve a certain threshold coverage. Dynamic compaction can be run-time intensive because a fault may be targeted multiple times if the fault requires scan cells or primary input values that conflict with the existing values of the inputs obtained from the previous test pattern generation. This can increase the run-time if dynamic compaction procedure is performed from the beginning of test generation. Therefore, in this method, cube merging is performed until threshold fault coverage is reached in order to exploit the advantage of run-time of cube merging. The threshold coverage is based on the fault coverage curve and the threshold coverage is set to coverage slightly before the tail portion of the fault coverage begins. A random fault ordering is used while targeting secondary faults.

The backtrack limit for secondary faults is set to be less than the primary fault backtrack limit in order to avoid spending too much time on test generation for the

secondary faults. This is based on average backtrack limit for all the faults for which test is generated when cube merging is used.

The number of secondary faults num_{sec} to be targeted with a primary fault is limited and is specified as a user defined parameter. num_{sec} is a function of the average time spent per backtrack, the target test pattern compaction and run time targeted for the entire test generation. num_{sec} is calculated as follows.

$$AveTimePerBacktrack(A) = \frac{Total\ time\ spent\ for\ faults\ before\ dynamic\ compaction}{Total\ number\ of\ backtracks\ before\ dynamic\ compaction}$$

$$AverageTimePerSecFault(T_{avesec}) = A * Secondary\ Backtrack\ Count$$

$$\#Passes = TargetScanOpCount - ScanOpsAtThreshold,$$

$TargetScanOpCount$ is determined based on the desired test pattern compaction and $ScanOpsAtThreshold$ is the number of test patterns generated right before dynamic compaction is enabled at the user specified threshold fault coverage

#Number of secondary faults targeted per primary fault (2X-t)/(#Passes*T_avesec) where 2X is the time that can be afforded for entire test generation, and X is the time taken for test generation when only cube merging is used for test compaction.

num_{sec} is a lower bound on the number of secondary faults to be targeted along with a primary fault. Typically the number of faults targeted per primary fault is set slightly higher than the number obtained by the above equation.

2.3.4 Improving run time by Static Untestability Analysis

The run time with basic dynamic compaction described above is some times more than twice the time when only cube merging is used for test compaction. The long run

times can be attributed to targeting faults that cannot be either activated or propagated because of controlling values on the side inputs of the d-frontier gates. In order to address the long run time, the next fault that is picked randomly can be statically tested for activation condition violation and for any x-path blocks during fault propagation. This is accomplished by saving the values of gates during test generation whenever a test is generated. These values are used to check for activation violation check, for instance let a fault on line n be n stuck at 0 and the stored value for the previously generated test was 0, then if the fault n stuck at 0 is targeted with the scan cell and primary input values set by the previous test, then line n will attain a value 0 and the fault cannot meet its activation condition. Therefore, by looking up the stored value for the previous test, it can be determined if the fault can be activated or not and thus can help reduce the run time by avoiding targeting faults that cannot be activated. The check for activation violation is performed for the timeframes in a test cycle when the fault can be activated.

The check for x-path blocks is done in a similar manner by performing depth first search up to a few levels from the fault site and check for any controlling values on side-inputs that can block the fault effect propagation. X-path check is limited to a few levels only in order not to spend too much time in the depth first search traversal which may itself cause longer run-times. The traversal for x-path along a path stops either if the maximum number of levels is reached or if there is an x-path block or if there is a sequential gate. The traversal stops at a sequential gate because, depending on the clock of a sequential gate, the fault can be propagated in a later frame if not in the current timeframe.

2.3.5 Reasoning Analysis to drop redundant faults

Apart from the activation violation and x-path blocks, another factor that impacts the run time is redundant faults. Redundant faults are untestable faults and of two types – circuit untestable and constraint untestable [56]. Circuit untestable faults are redundant

faults and a test cannot be generated for such faults because of the nature of the circuit. Constraint untestable faults are untestable due to constraints set for ATPG which are conservative because the design is not mature enough. These constraints may be relaxed later by a DFT engineer to improve the test coverage. Test coverage is different from fault coverage. Test coverage accounts for the redundant faults and hence the number of redundant faults is subtracted during the calculation of coverage, whereas fault coverage is calculated for total number of faults. During dynamic compaction, when a redundant fault is targeted there is no test generated with the constraints set by the previous test generated and the fault is targeted again with another primary fault. Thus, a redundant fault can be potentially targeted multiple times which can lead to longer run times. We present a reasoning analysis approach based on the conflict driven learning approach in [15, 56]. For completeness purpose, the method of [15] is briefly described here. The conflict driven learning uses an AND/OR reasoning framework that is built during fault propagation and justification. During fault propagation, the fault effect D that needs to be propagated through the fanouts of a d-frontier gate form an OR relationship for the fanout gates since one of the fanouts is chosen for propagation. There exists an AND relationship between the parent d-frontier and the selected d-frontier since the fault effect needs to be propagated through both the gates in order to reach an observed point. The AND relationship holds true even for the off-path J-frontiers that need to be justified in order to propagate the fault through the D-frontiers. Every time a gate is assigned a value, a graph node is created for the gate corresponding to a timeframe. The graph nodes have AND relationship as described previously. OR choices do not have a graph node created and are maintained separately in a stack. Implication graph is created only for the portion of the circuit that is active during test generation so as to keep the size of the implication graph small.

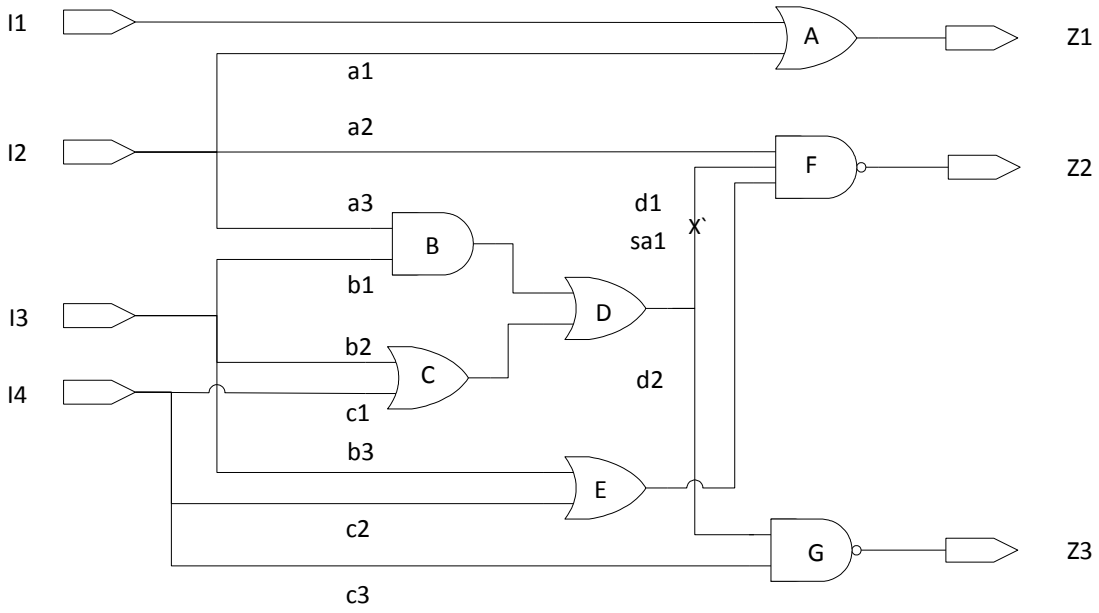


Figure 2.7 An example circuit showing a redundant fault

The implication graph is used to identify the set of implicants that imply a value on a gate. Traversing backward from a given node in an implication graph, the set of implicants that imply a specific gate value on a node can be identified. Whenever a conflict scenario occurs during test generation, backward traversal is performed starting from the conflicting nodes to identify the set of implicants that cause the conflict. During the backward traversal when a conflict occurs, the traversal begins from the conflicting nodes and stops at a node if it is a decision node or an implication node at a decision level lower than the highest decision level of the conflicting nodes. The backtrack process proposed in [15] is non-chronological and backtracks are based on reasons collected for conflicts. For a j-frontier J, let R be the set of implicants for J. Let C_1, C_2,…., C_n be the choices to justify J. If all the choices lead to conflict and let R_i be the corresponding reason set for the conflict at C_i, then backtrack can be made to decision level *l* where, *l* is the maximum decision level among all the decision levels of nodes in the reason set {R, R_1, R_2, …., R_n }.

Consider d1 stuck at 1, a redundant fault in the circuit shown in Figure 2.7. The implication graph generated during test generation for d1 stuck at 1 fault is as shown in Figure 2.8. It can be seen that there is a conflict on gate E and while tracing back from the conflicting nodes according to the conflict diagnosis procedure in [15], the reason set only consists of node N1 since all the nodes are implication nodes and of the same decision level. The first node is not considered since it is the reason for activation of the fault. Therefore, a redundant fault has no reasons in conflict diagnosis process and this can be used for the benefit of dynamic compaction to drop redundant faults determined during test generation.

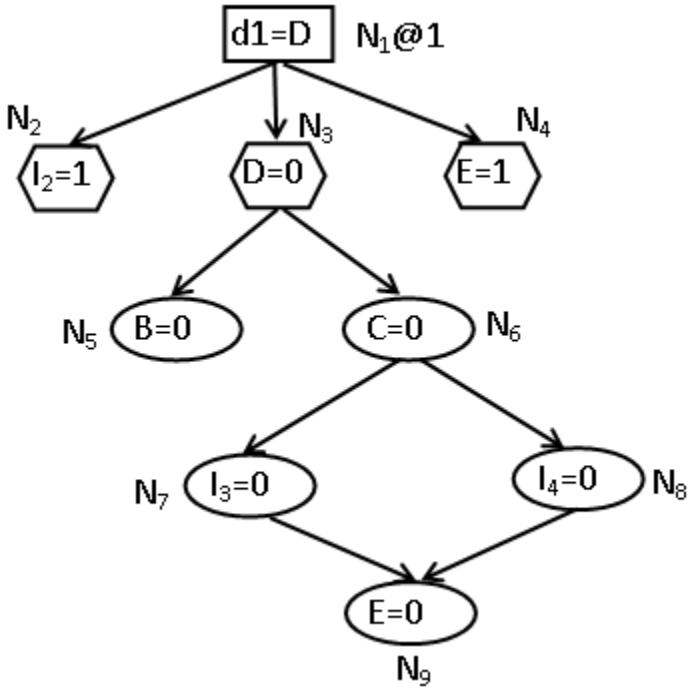


Figure 2.8 Implication graph generated during ATPG for testing d1 s-a-1

During incremental dynamic compaction, the primary input and scan cell values that are specified by previous test are attributed with a special property in order to determine if a

fault is untestable in the presence of the constraints set by the previous test or if the fault is untestable due to other reasons like pin constraints or functional constraints, etc. Consider a primary fault a1 stuck at 1 in Figure 2.7, for which test is generated by specifying I2=0 and I1=0. These specified primary values are applied as constraints to generate a test for the secondary fault B stuck at 0. It can be observed that B stuck at 0 requires I2=1 and I3=1 and since I2=1 conflicts with the already specified value I2=0, a test cannot be generated for B stuck at 0 under the input values specified by previous test. In order to differentiate the constraints specified by dynamic compaction from other types of constraints, let a dummy node of type DYN to represent dynamic compaction constraints be added as a fanin to the specified primary inputs I2 and I1 as shown in Figure 2.9.

The implication graph is shown in Figure 2.9. It can be observed that the backtrack procedure eventually ends in the node I2=0 and the reason for the fault B stuck at 0 being untestable is due to the constraints set by the process of dynamic compaction.

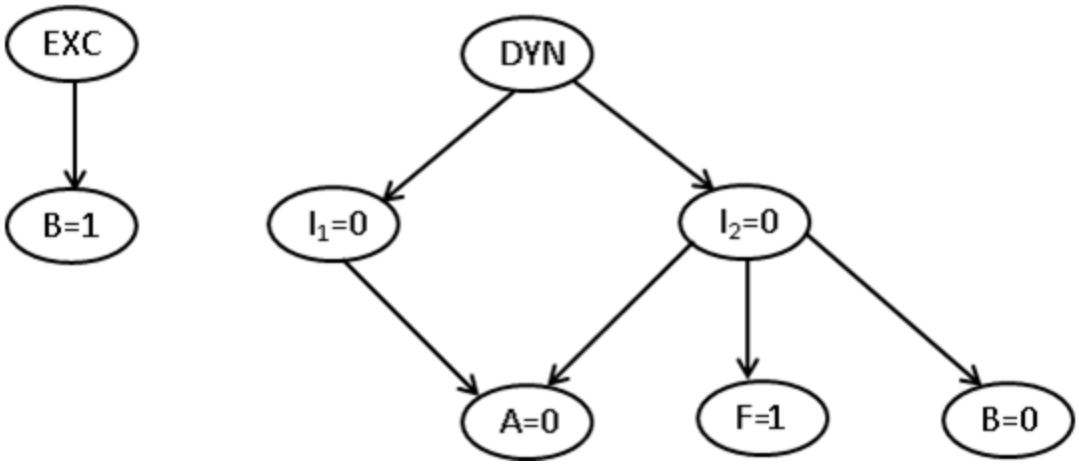


Figure 2.9 Implication graph generated during incremental dynamic compaction

The reason set R consists of the DYN node. In case fault d1 stuck at 1 which is a redundant fault is targeted as a secondary fault with primary fault a1 stuck at 0. The backtrack process will be independent of the dynamic compaction constraints and the reason set R will be empty. This can be used to drop redundant faults from being targeted again with a later primary fault.

2.3.6 Automatic Identification of Parameters for Dynamic Compaction

The main drawback of the above proposed procedure is that the various parameters for enabling dynamic compaction namely – threshold, secondary fault backtrack limit and number of faults to be targeted per primary fault are calculated manually and need to be provided to ATPG. This requires that first the test generation tool needs to be run with the cube merging method of compaction and then the parameters need to be evaluated, which is time consuming. The calculation of the parameters can be automated and can be evaluated during initial cube merging and the parameters can be identified on the fly during test generation.

In the automated method proposed here, cube merging is performed until a threshold is reached in order to exploit the advantage of run-time of cube merging. The decision point or the threshold is based on ratio of running average of unique detects (detections on faults simulation) and running average of unique targeted faults. Unique detections constitute the faults that are detected during fault simulation once the compacted pattern in the bin is to be fault simulated. These unique detections exclude the faults, which constitute the faults detected by the patterns of the merged test cube that is fault simulated. Therefore, the unique detects only include the detections that are unintentionally detected. The unique targeted faults are the faults corresponding to a merged pattern in a bin that is to be fault simulated. These faults consist of the ones that have not been detected yet and exclude the faults that were detected already due to fault

simulation of other patterns. This can happen when pattern in some other bin is fault simulated and the pattern accidentally detects additional faults, which belong to the current bin. The motivation to use this ratio is that it is more beneficial to use dynamic compaction in the region where the number of unique collateral detects with respect to the targeted faults is low. The ratio conceptually means that if the unintentional detections upon fault simulation per pattern fall to a small number, the tail region of the fault coverage curve is reached. This is when the patterns are not compatible and it is required to generate compaction friendly patterns and dynamic compaction is enabled beyond this threshold. In the proposed method we set the target ratio to 2. If the ratio is less than or equal to 2, dynamic compaction is enabled. In some cases, the number of patterns on the tester is restricted due to memory limitations. In such situations, the target ratio might not be reached yet. In such cases, dynamic compaction is enabled at 60% of the number of patterns specified to the tool.

Backtrack limit for the secondary faults is calculated as the average of backtracks used for all the faults for which test is found during cube merging. Once the threshold is reached, dynamic compaction is enabled and the secondary backtrack limit is set as described. Secondary fault backtrack limit is incremented each time a new primary fault is tried until the average backtrack+50% of average is reached.

The number of secondary faults to be targeted with a primary fault is to be restricted because this could impact the run-time for overall ATPG. This is determined during the initial portion of the test generation when cube merging is performed. This is given by the equation below.

#faults per pass = factor * (# Total faults targeted)/(# Test Patterns)

The ratio of total faults targeted and the number of test patterns gives an approximate estimate of the number faults that need to be targeted per primary fault or

per test pattern. The secondary faults targeted along with a primary fault, constitute of the faults of three categories –

- Test found faults – For these faults test is found with the current primary fault
- Aborted faults – Test generation is not successful for these faults because they are aborted due to limited backtrack limit
- Redundant faults – Test cannot be generated for these set of faults because these faults are untestable in the presence of the constraints set by the previous faults targeted with the current primary fault

The *Factor* in the equation scales the ratio of the total faults targeted per test pattern and is required because towards the tail end of the curve, the number of faults that need to be targeted is large in order to achieve test size reduction because the probability of generating a test in the tail is reduced because of the nature of the faults which are hard-to-detect. The *Factor* is set to 10 in our experiments.

During the initial phase when only cube merging is performed, the running average of the number of test patterns $comp_{ave}$ compacted or merged in the written out bin is computed. Once the threshold ratio is achieved, dynamic compaction is enabled. All the patterns in the bins that have compacted greater than or equal to the running average $comp_{ave,}$ number of patterns, are written out and fault simulated. All the patterns in the remaining bins are emptied and are not fault simulated because the patterns are not optimally compacted.

Apart from these three parameters which determine the amount of compaction achieved during dynamic compaction, the number of bins used during cube merging determines the compaction achieved during the initial phase of test generation. As the number of bins is increased, the compaction achieved also increases because there is greater possibility of finding a compatible bin or an empty bin. Therefore there is less frequent writing out of patterns from the bins and in turn greater compaction is achieved.

However, as the number of bins increases, the run-time also increases because it takes longer to identify a compatible bin.

2.4 Experimental Results

The incremental dynamic compaction technique was performed on industrial designs and the following experiments were conducted – (i) Comparison of basic incremental dynamic compaction over cube merging technique and (ii) Comparison of dynamic compaction with static untestability analysis and cube merging technique. The size of the various designs is shown in Table 2.3.

2.4.1 Comparison of basic incremental dynamic compaction over cube merging technique

The CONCAT procedure of [15, 56] is the underlying conflict learning procedure used in our experiments. The experiments are performed on industrial circuits and the results are shown in Table 2.4. Column 2 and 3 of Table 2.4 gives the test set sizes with cube merging method and with incremental dynamic compaction method respectively. Columns 4 and 5 represent the run times for the designs when run with cube merging method and with incremental dynamic compaction method respectively. Both the test set size and run-time is given at a given fault coverage. The number of scan operations is limited for the experiments and is curtailed at 10000 for smaller designs and 3000, 5000 for larger designs. The percentage decrease in test size when incremental dynamic compaction is compared with cube merging technique is specified in column 6. The run time for incremental dynamic compaction is specified as a multiple of the run time for cube merging method and is specified in column 7. The maximum coverage for given limit on scan operations is given in Table 2.5. Column 2 is the maximum coverage

Table 2.3 Approximate size of designs

Design	Gate Count
A	350k
B	5600k
C	850k
D	6700k
E	6200k
F	10000k
G	1200k
H	6000k
I	6500k

Table 2.4 Results for incremental dynamic compaction vs. cube merging technique

	Test Size		Time (hours)			
Design	CubeMerging	DynComp	CubeMerging	DynComp	%TestRed	Time(X)
A	10000	7899	2.6	3.55	21.02	1.37
B	10000	6336	66.25	71.25	36.64	1.07
C	10000	6486	10.83	38.11	35.14	3.52
D	5000	3408	212.17	289.21	31.84	1.36
E	3000	2218	109.31	144.65	26.06	1.32
F	5000	3157	271.5	342.16	36.86	1.26
G	7598	6885	7.45	30.68	9.38	4.11
H	4025	2874	26.68	75.06	28.59	2.81
I	4772	4021	60.5	82.58	15.73	1.36

Table 2.5 Maximum coverage achieved with cube merging and incremental dynamic compaction

Design	Cube Merging	DynComp
A	91.48	93.23
B	90.65	93.40
C	94.52	95.53
D	76.59	77.36
E	77.92	80.35
F	84.49	86.15
G	85.52	85.53
H	73.52	73.82
I	70.22	70.40

Table 2.6 Comparison of BDR and BDR+SUA approach

	Test Size		Time (hours)		
Design	BDC	BDC+SUA	BDC	BDC+SUA	TimeRed(%)
A	7899	7826	3.55	3.63	-2.25
B	6336	6305	71.25	65.83	7.6
C	6486	6502	38.11	35.43	7.03
D	3408	3379	289.21	316.38	-9.39
E	2218	2196	144.65	157.03	-8.55
F	3157	3129	342.16	323.21	5.54
G	6885	4890*	30.68	14.3	-
H	2874	3286*	75.06	81.46	-
I	4021	3336*	82.58	73.77	-

It can be observed that the basic dynamic compaction approach gives test pattern size reduction upto 36% with incremental dynamic compaction and the run time is upto 4 times the run time with cube merging technique. The target scan count reduction set for these experiments was 20%. The number of secondary faults to be targeted per primary fault is determined based on the equations described in previous section. The threshold fault coverage is determined based on the fault coverage curve for the cube merging technique. The secondary fault backtrack limit is based on average backtrack for testable faults.

2.4.2 Comparison of basic incremental dynamic compaction with static untestability analysis over cube merging technique

The experimental results for the runs with basic incremental dynamic compaction (BDC) and basic incremental dynamic compaction with static untestability analysis (BDC+SUA) is shown in Table 2.6. The test set sizes with BDC and BDC+SUA is shown in columns 2 and 3 respectively. Columns 4 and 5 show the test run times with BDC and BDC+SUA. It can be seen that the static untestability analysis helps in run time reduction for some of the smaller designs but adds to the run time for larger testcases. This is attributed to the additional time involved in saving the values of gates whenever a test is found, which can be large for larger circuits. In addition to this the depth first search traversal also is expensive in terms of time for larger circuits. The entries in column 3 that are star marked, is to indicate that the fault coverage is 0.3%, 0.5% and 0.07% less than the cube merging fault coverage.

2.4.3 Comparison of incremental dynamic compaction with automatic parameter identification method with cube merging technique

Since the basic dynamic compaction approach requires the parameters for dynamic compaction namely the threshold, number of secondary faults per primary fault

and the secondary fault backtrack limit to be provided to the tool manually, we proposed an automatic method of identification of parameters for dynamic compaction (API). Table 2.7 compares the test set size and run time obtained with API with cube merging. The number of bins used for cube merging and dynamic compaction is 32. The test set sizes with cube merging and API is shown in columns 2 and 3 respectively. Columns 4 and 5 show the test run times with cube merging and API respectively. The reasoning analysis is used to drop secondary faults that are identified to be redundant during dynamic compaction. This avoids the redundant faults from being targeted again and reducing run-time over head. From the results it can be observed that the test size reduction achieved is approximately 30% and the run-time is 2X times the run-time it takes for cube merging. From the table it can be seen that for testcases G and I, the fault coverage drops by 0.5% and 0.6%. Upon investigation, it was found that the difference in fault coverage was because of the faults detected accidentally during fault simulation. When the patterns are fault simulated, a sequence of patterns is fault simulated and therefore there are accidental detections of faults that are activated and propagated across test patterns. These are the faults for which single test cycle is not sufficient and require more than one test cycle to be detected.

When the number of bins is increased, the compaction achieved when cube merging is used typically increases. This happens because there is greater possibility of finding a compatible bin or an empty bin. Therefore the writing out of patterns from the bins is delayed and in turn greater compaction is achieved.

However, as the number of bins increases, the run-time also increases because it takes longer to identify a compatible bin. When the automatic parameter identification method of dynamic compaction was used with large number of bins i.e. 1000 bins, the run-time associated with dynamic compaction was very large because the number of number of faults per primary fault that is selected is very large because as the number of

Table 2.7 Comparison of results for API vs. cube merging technique for 32 bin size

	Test Size		Time (hours)			
Design	CubeMerge	Auto	CubeMerge	Auto	%TestRed	Time(X)
A	10000	5962	2.6	8.5	40.39	3.26
B	10000	6574	66.25	69.63	34.26	1.05
C	10000	6978	10.83	17.36	30.22	1.6
D	5000	3386	212.17	249.78	32.28	1.11
E	3000	2073	109.31	304.75	30.9	2.78
F	5000	3417	271.5	333.23	31.66	1.22
G	7598	4158	7.45	26.5	-	-
H	4025	3702	26.68	44	8.02	1.64
I	4772	2316	60.5	90.23	-	-

Table 2.8 Maximum coverage achieved with API and cube merging for 32 bin size

Design	Cube Merging	Auto
A	91.48	95.82
B	90.65	93.64
C	94.52	95.61
D	76.59	77.34
E	77.92	81.11
F	84.49	86.44
G	85.52	85.00
H	73.52	73.71
I	70.22	69.67

Table 2.9 Parameter selection for API method for 32 bin size

Design	Threshold	#Secbkt	#SecFaults
A	85.46/3780	5	50
B	89.29/6000	8	180
C	91.76/4898	73	90
D	75.22/2617	23	930
E	75.91/1800	50	880
F	83.16/3000	5	450
G	81.82/2250	17	210
H	72.12/2341	46	190
I	67.22/1161	22	910

bins increases, the number faults targeted or resolved per test pattern is really large compared to the smaller bin size case. Another factor that adds to run-time is that when dynamic compaction is enabled, all the patterns in the bins that have compacted greater than or equal to $comp_{ave}$ are fault simulated and the patterns in the remaining bins are discarded because the patterns in those bins are not optimally compacted. It was observed that a very small fraction of the patterns were actually fault simulated and most of the patterns in the bins were discarded. This implies that the effort spent in generating a test for those faults is lost and those faults are targeted again during dynamic compaction. In order to account for this, we use smaller bin size of 256 for the initial phase of test generation where only cube merging is used. Therefore, lesser number of patterns in the bins is discarded and lesser number of secondary faults per primary fault are targeted.

Table 2.10 compares the performance of API method when run with 256 bins against cube merging when run with 256 bins and 1000 bins. Columns 2 and 4 are the number of test patterns when cube merging is used with 256 bins and 1000 bins respectively. Columns 3 and 5 are the number of test patterns obtained when API is run

Table 2.10 Comparison of results for API(256 bin size) vs. cube merging technique (at 256 and 1000 bin size base coverage)

	Test Size				Time (hours)			
Design	Cube@ 256bins	Auto @256 BCov	Cube @1000bins	Auto @1000 BCov	Cube@ 256bins	Auto @256 BCov	Cube @1000bins	Auto @1000 BCov
A	10000	7695	10000	7976	3.4	52.5	3.75	55
B	10000	6595	10000	7076	58.85	124.75	70	174.61
C	10000	7559	8731	7624	13.25	40.28	11.61	41.03
D	4976	3852	4462	3359	214.6	867.68	205.18	800.25
E	5000	3289	3000	3342	210.16	383.5	215.5	424.35
F	3000	2264	3000	2720	321.5	421.9	260.5	938.63
G	6669	-	6693	-	-	-	-	-
H	3323	2014	2795	2080	23.21	59.71	21.5	62.8
I	2045	-	1247	1513	28.81	-	26.18	232.66

with 256 bins at the fault coverage achieved when cube merging is run with 256 bin size and 1000 bin size respectively. Columns 6 and 8 represent the time required for cube merging when run with 256 bins and 1000 bins respectively. Columns 7 and 9 give the time required when API is run with 256 bins at the fault coverage achieved when cube merging is run with 256 bin size and 1000 bin size respectively. It can be observed that the compaction achieved is >20% in most of the cases except testcases G and I. In case of G the fault coverage is 84.53%, the reason being the same as discussed previously. However, in the case of testcase I, when cube merging is run with 1000 bins, the fault coverage curve is steep and there is no tail portion. In this case dynamic compaction is not required because the cube merging method provides desired compaction and the motivation for the proposed method is to address long tail portion of the fault coverage

Table 2.11 Maxim Fault Coverage, Test size reduction and run-time overhead achieved with API and cube merging

	Cube Coverage			@256 bins Cube Cov		@1000 bins Cube Cov	
Design	256b	1000b	Auto	%ScanRed	Time(X)	%ScanRed	Time(X)
A	96.32	96.52	97.2	23.05	15.44	20.24	14.66
B	91.33	92.01	94.06	34.05	2.12	29.24	2.49
C	95.53	95.55	95.85	24.41	3.04	12.67	3.53
D	77.20	77.04	77.51	22.58	4.04	24.71	3.9
E	80.52	80.72	83.4	34.22	1.8	33.16	1.9
F	85.06	86.09	86.43	24.53	1.3	9.33	3.6
G	85.28	85.28	84.53	-	-	-	-
H	73.53	73.58	74.48	39.39	2.57	25.58	2.9
I	69.60	69.15	69.57	-	-	-21.33	8.88

curve and the design does not have a tail region. Table 2.11 tabulates the fault coverage achieved with cube merging method with 256 bin size, 1000 bin size and API method respectively. The later columns give the test set size reduction and run-time overhead compared with cube merging when 1000 bins are used.

2.5 Conclusion

In this work, we propose an incremental dynamic compaction approach that incorporates cube merging method with dynamic compaction enabled after certain threshold is reached. The threshold is determined internally during test generation and dynamic compaction is enabled. The parameters for dynamic compaction are determined on the fly during test generation. This approach is very efficient in test pattern count reduction. When small bin size of 32 is used, there is approximately 30% compaction achieved with the proposed incremental dynamic compaction method, with a run-time of

2X times the cube merging method. However, as the number of bins is increased, the run-time increases when dynamic compaction is used, but the proposed method provides greater than 20% compaction. Static untestability analysis is proposed to address long run-time problem to avoid targeting secondary faults that could have activation conflicts or x-path blocks. However, the method reduces the run-time in some cases and does not benefit in some other cases. Reasoning analysis method which uses an AND/OR reasoning graph is proposed which helps identify the cause for a secondary fault being redundant during dynamic compaction. Redundant faults that are independent of the constraints set by dynamic compaction are dropped and hence such faults are avoided from being re-targeted and thus the unwanted effort required to re-target the redundant faults is avoided.

The technique is effective in the fault coverage tail or designs with long fault coverage tail where compaction friendly patterns are necessary for hard to detect faults.

CHAPTER 3 PATH DELAY FAULT TESTING WITH SEGMENTED FAULT MODEL

In this chapter we present a test generation methodology for path delay faults suitable to testing multi-segment paths in partially scanned designs. In order to increase the delay fault coverage of pseudo-robust tests we propose a methodology to divide full-paths into sub-paths and generated patterns for sub-paths. Experiments conducted on industrial designs using the generated path delay fault detection patterns show that the generated patterns are more effective in identifying speed-path failures on silicon than n-detect transition atpg patterns currently used for this purpose.

3.1 Introduction

Due to increasing clock rate and scaling down of feature sizes, manufactured devices are subject to delay defects which affect the functional operation of the design when run at high frequency. Delay defects are mainly caused by process variations, increasing clock rates, increasing chip density. The process variations may lead to failure of device when run at higher clock rate for the specified time interval [33], however the device may function correctly at lower frequencies. This type of defect is modeled as delay fault. There are two types of delay fault models – lumped and distributed. Lumped type delay faults are treated as point defects and are considered to be concentrated at a gate output. Examples of lumped type delay fault model are transition faults and gate delay faults. Transition fault model assume the delay defect to affect only one gate in the circuit and the delay to be so large that the delay of any path passing through the fault site exceeds the cycle time. Gate delay fault model assumes that the delay is lumped at a gate in the circuit, however the assumption is that the delay on the gate only affects long paths through the fault site. The advantage of lumped delay fault model is that the number faults is linear in the number of gates in the circuit, test generation is relatively simple

because traditional stuck-at tests can be used with initialization vectors. The limitations of lumped delay fault model are that that since the fault model is based on the assumption that the delay is concentrated at a gate, the test for these faults may fail to detect delay faults resulting from a sum of several small delay defects. Distributed type delay fault is considered to be distributed along the path which accounts for the cumulative process variations. Example for distributed type delay fault model is path delay fault model. The advantages of distributed fault model are that it models the impact of cumulative delay variations, the tests for these faults can target critical paths of interest and can also target gate or transition faults. The major drawback of distributed fault model is that the number of paths in a circuit can be very large and the test generation process is relatively difficult because the test generation involves sensitizing and satisfying necessary off-path conditions in two time frames rather than one, which is the case for stuck-at atpg for all the gates on the path.

The focus of the work is test generation for path delay faults in partial scanned circuits and we propose a method by which the fault coverage of untestable paths in the design can be increased by progressively generating patterns for smaller sub-paths when pattern generation for a full path fails.

3.2 Preliminaries

Fault models are used to model physical defects so as to translate the problem of fault analysis from a physical problem into a logical problem [5]. Fault modeling also reduces the complexity as many different physical faults may be modeled by the same logical fault. There are various types of fault models of which the delay fault model is used in our work. A delay defect causes the delay of a path to exceed the clock period or the cycle time. The slack of a path is defined as the difference between the designed cycle time and the actual delay. When the size of the delay exceeds the slack of a path it leads to incorrect values at the circuit output. There are two types of delay fault models:

- Transition fault model:

Transition fault is a lumped delay fault model. In the transition fault model, the delay is considered so large that any path passing through the fault site exceeds the cycles time for the path [9]. A stuck at fault models the defect where a signal line is stuck at a value 0 or 1 permanently. A transition fault's behavior is similar to a stuck-at fault, however it limited to a finite duration. A transition fault is a gross delay fault where the propagation delay of all the paths passing through the fault site exceeds the cycle time [34]. There are two types of transition faults – slow-to-rise and slow-to-fall. There can be a maximum of 2N transition faults in a circuit where N is the number of nets in the circuit. A two pattern test is required to activate a transition fault, where the initializing vector sets the fault site to the initial value and the final vector launches the transition. For a slow-to-rise transition, the final vector is a stuck-at 0 test for the faulty line. Once the transition is launched, it is propagated to an output. Consider the circuit shown in Figure 3.1, where line e has a slow to fall transition fault. V1 = 11XX and V2 = 0x11 corresponding to inputs {a, b, c, d} is a two pattern test that detects the slow-to-rise fault on line e.

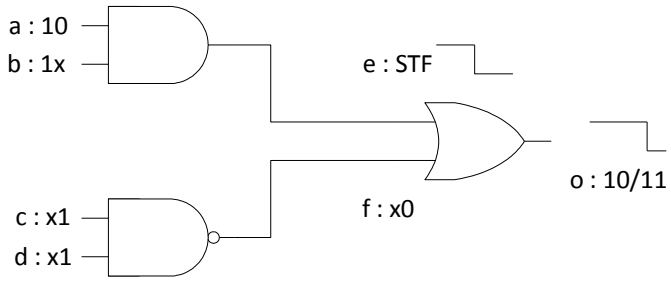


Figure 3.1 Transition fault Model

- Path delay fault model:

The path delay fault models distributed failures, which are typically caused by statistical variations in the manufacturing process. When the cumulative delay of a path exceeds the clock period for the path, it can cause chip failure. A lot of research has been done on various aspects of test generation and fault simulation of path delay faults. A path delay fault is tested using a two pattern test. The transition created by the test pattern is propagated along the path explicitly by sensitizing every gate of the path and satisfying necessary off-path conditions. The two popular ways in which a path can be sensitized are robust sensitization and non-robust sensitization.

1) Robust sensitization:

In robust sensitization the path is sensitized independent of the delay on the off-path inputs and the fault is detected even in the presence of delay on off-path inputs [35]. It is ideal sensitization and is practically difficult to achieve. Consider the path P = {A, C, E} shown in Figure 3.2. Signal lines A, C and E are the on-path inputs of the path. Signal lines B, D and F are the off-path inputs of the path.

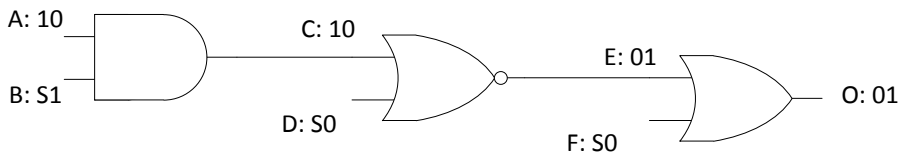


Figure 3.2 Robust Sensitization

The robust test to detect the path consists of a two pattern test - V1<S1> and V2<01> for inputs A and B, where S1 represents stable 1 value and S0 represents stable value 0 in both the time frames. V1 is the initialization vector

which sets the initial value of the transition and V2 is the final vector which launches the transition on the beginning of the path and activates the path delay fault. The test detects the path delay fault at the output independent of delay on the off-path inputs of the path.

2) Non-robust sensitization:

This assigns non-controlling values to the off-path inputs in the second vector of the test. Transition along every gate of the path is not ensured. Test can be invalidated by delays on off-path inputs under this type of sensitization [35]. Non-robust sensitization is shown in Figure 3.3. Vectors <0X> and <11> form a non-robust test for the path P. It can be seen that the test can get invalidated if the off-path input F has a delayed 10 transition. Therefore non-robust tests do not guarantee the detection of a path delay fault independent of delays in other parts of the circuit.

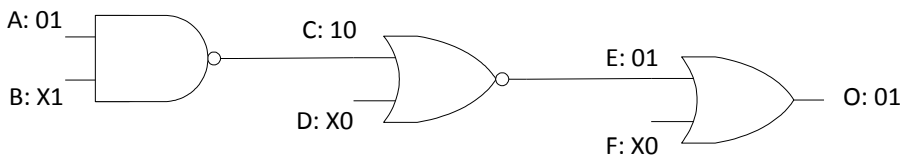


Figure 3.3 Non-robust Sensitization

3) Pseudo-robust sensitization:

This ensures that the final value of the off-path inputs is non-controlling value in both the vectors. This sensitization ensures transition on all the gates along the path. Due to hazards the final values of the gates may not be stable. Hazards are unwanted pulses or glitches that appear at internal signals or primary outputs and

are caused due to difference in delays along reconvergent signal paths when there is transition at inputs [3]. There are two types of hazards – static and dynamic. Static hazards are caused due to glitches in a steady signal [37]. Dynamic hazards result from multiple changes on a signal before attaining the steady state [37]. The test maybe invalidated in the presence of hazards on off-path inputs.

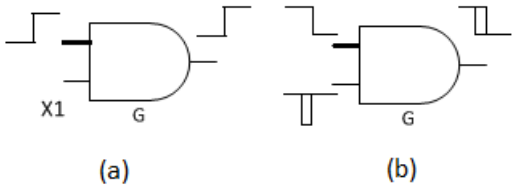


Figure 3.4 Pseudo-robust sensitization of AND gate for (a) Rising Transition (b) Falling transition

Pseudo-robust sensitization of AND gate, for rising and falling transition on on-path input shown by thick line is illustrated in Figure 3.4(a) and (b) respectively. It can observed from Figure 3.4(b) that a test for a delayed falling transition can be invalidated in the presence of a hazard on the off-path input. The output which would be a delayed falling transition may have a delay free transition due to the hazard on the off-path input. Throughout the proposed work, pseudo-robust sensitization is used as robust sensitization is practically difficult to achieve.

3.3 Review of Previous Work

In this section, various test generation techniques that have been developed for path delay faults in combinational as well as sequential logic circuits are discussed.

3.3.1 PODEM based Test generation for Path Delay Faults

In [33] a five-valued logic system is proposed to generate robust deterministic test patterns based on PODEM to detect path delay faults. The paper also gives the necessary and sufficient conditions for a two pattern test to be a robust test for a given path. Robust tests are important since the tests used to detect path delay faults should not get invalidated in the presence of delays on other paths of the chip. The excessive delays are caused by the device parameter variations due to random fluctuations during fabrication of the circuits. Initially, robust tests with six-valued logic were considered in [36] to detect path faults by two pattern test set, though the method of test generation for a given transition path was not given. The five-valued logic system proposed in [33] is a subset of the seven-valued logic system in [37].

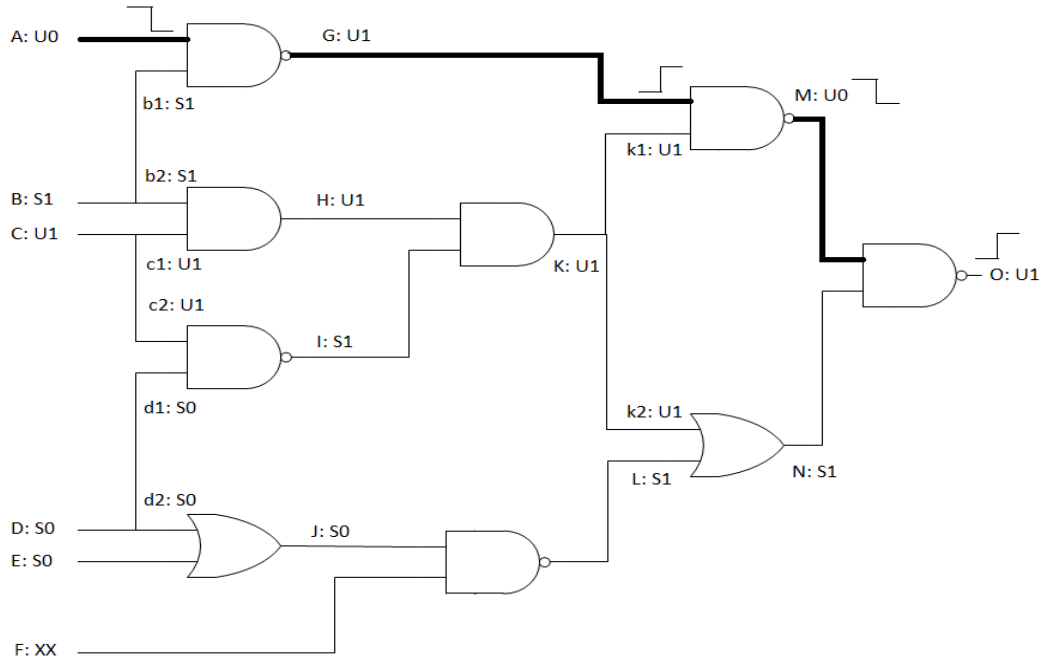


Figure 3.5 An example circuit

The five-valued logic values are {S0, S1, U0, U1, XX}. S1 (S0) represents hazard-free stable value on a line in the circuit where the initial value and final value on the line is 1(0). U1(U0) represents final value 1 of a signal line with initial value being 0 or 1 and there could be a hazard or transient between the two time frames. U1 (U0) includes signal value S1 (S0). XX represents signal value being X in both initial and final time frame. XX covers both U1 and U0.

Test generation for path delay faults in [33] is done similar to test generation for stuck-at faults with the difference that the test generation for path delay faults has the path determined upfront and so there is no need to determine path(s) to propagate the fault effect to an observe point unlike stuck-at faults. The necessary values required for the off-path inputs need to be justified. The PODEM based test generation approach in [33] is explained with an example below.

Consider the circuit in Figure 3.5 for which a robust test needs to be generated for path A-G-M-O with a falling transition at the input of the path by setting A to U0. The three objectives necessary to generate a test are determined by the necessary off-path conditions for lines b1, k1 and N which are required to have value S1, U1 and S1 respectively. These are ordered by priority as follows: set N to S1, b1 to S1 and k1 to U1. These values are justified as in PODEM and during backtrace when primary inputs are reached, only S1 and S0 values are tried if an internal line has an objective of S1 or S0 to be satisfied. This reduces the number of branches in the decision tree from four to two branches for a primary input. Only when the objective consists of setting a signal line covered by U1 or U0, a U1 or U0 is tried on primary inputs.

The advantage of this method is that the logic system used facilitates generation of minimum specified tests as a result of which the number of specified inputs is less and hence leads to better compaction of test patterns for the set of path delay faults.

3.3.2 DYNAMITE

DYNAMITE [35] is a test pattern generation system for path delay faults in combinational or scan-based circuits. DYNAMITE stands for Delay Fault Oriented Automatic Test Pattern Generation System and is based on the techniques proposed in SOCRATES [23]. The test pattern generator is capable of generating robust tests using a 10-valued logic system as well as non-robust tests for path delay faults based on a 3-valued logic. The major limitations of most of the path delay ATPG approaches is the large number of paths in a circuit which grows exponentially with depth and hence only a subset of paths can be targeted and needs to be determined prior to test generation in some way. The authors propose a new path sensitization procedure which identifies large number of redundant faults by single test generation attempt. The paths are stored in an efficient data structure called path tree. The path tree structure is described below.

3.3.2.1 Path Tree Structure

As the number of paths in a circuit could be very large, the paths need to be stored in an efficient manner. In [35], a tree data structure, in which portions of paths which are common to many paths from a primary input to a primary output are stored only once rather than allocating space for every path separately. According to the concept of path tree data structure, a structural path $SP = \{s1, s2, \ldots\ldots, sn\}$ is a path from signal s_1 to s_n where the signal $s1 \in (S_{PI} \cup S_{FO} \cup S_{XO})$ AND $sn \in (S_{PO} \cup S_{FO} \cup S_{XO})$ and signal si does not pass through a fan-out stem or output signal of XOR or XNOR gate. S_{PI} and S_{PO} is the set of all primary inputs and primary outputs respectively. S_{FO} corresponds to the set of all fan-out stems and S_{XO} is the set of all output signals of XOR and XNOR gates. Paths which are same structurally and only differ in the direction of transition at PI, is accounted for by storing the direction of transition in the common leaf node.

2.3.2.2 Path Sensitization Procedure

The path sensitization method is capable of processing large number of paths by identifying redundant path delay faults in a single ATG attempt. Consider a functional path $P = \{s_1^{t1}, \ldots s_n^{tn}\}$ that is picked from the path tree. The path P is partitioned into k functional sub-paths $S1, \ldots Sk$ based on the node sequence $(n1, \ldots, nk)$ for the path P in the path tree. Every sub-path Si of path P is sensitized consecutively based on robust or non-robust test conditions. Once a sub-path is sensitized, implication is performed. If there are no conflicts during the implication step, sensitization continues with the next sub-path in the target path. If a conflict occurs during implication then all the sub-paths identified to be redundant are dropped from the path tree and the first node of the node sequence corresponding to the sub-path is flagged. Therefore, all functional paths which contain sub-path with the flagged node and a transition that corresponds to the flagged node can be dropped. Dropping sub-paths from the tree leads enables release of memory which helps improve memory efficiency.

A major drawback of this approach is the huge amount of memory required for the path tree storage in case of large circuits.

3.3.3 Test Generation for Path Delay Faults in Non-scan Circuits

The authors in [37] propose a test generation method for path delay faults in synchronous sequential circuits. In order to generate a test for a path delay fault, the netlist model of the circuit under test is augmented with a logic block consisting of a pair of flip-flops and a few combinational gates. The gates in the logic block are driven by signals driving the inputs of the on-path gates of the path. The path delay fault is test by testing for a single stuck at fault in the logic block. The stuck-at fault in the block is activated and the fault effect is latched into the destination flip-flop once all the signals on the path are set in the states required for the test. Once the fault is activated, the fault effect is propagated to an observe point. Thus the test pattern sequence generated for the

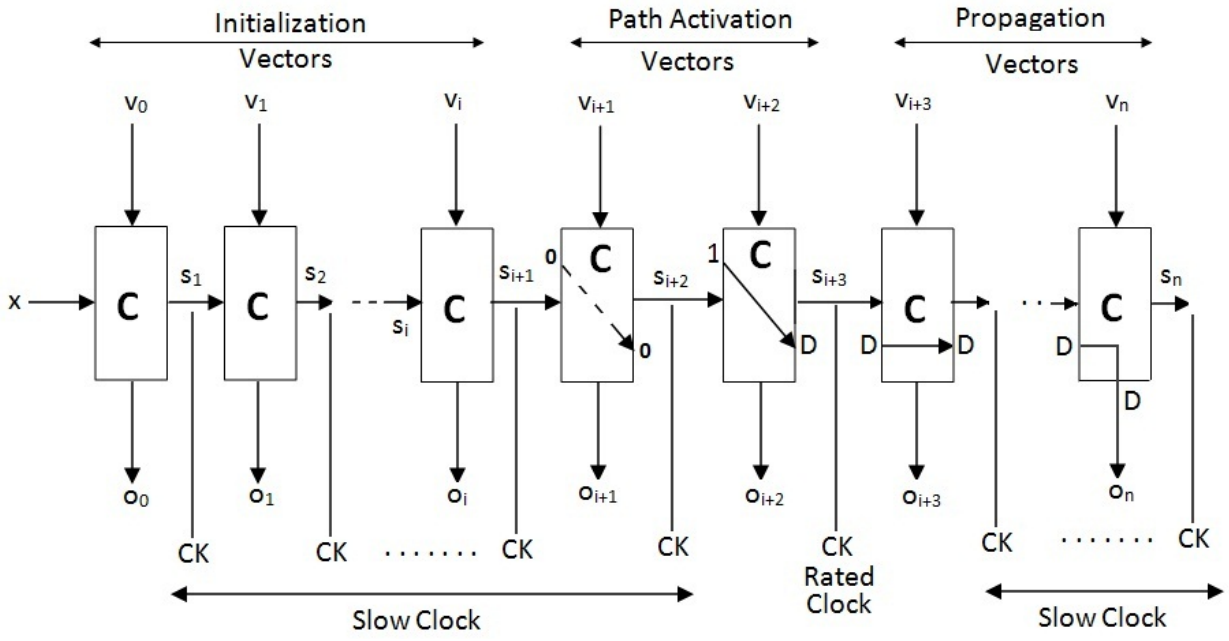


Figure 3.6 Testing for path delay fault [37]

stuck at fault performs initialization, path activation and fault propagation required for testing the path delay fault.

The path delay fault model used in [37] is similar to the one given in [38]. A path starts from a primary input or a flip-flop and ends at either a primary output or a flip-flop.

Test generation consists of three phases:

1) Initialization:

During this phase the flip-flops are set in the suitable states required for later phases by applying an initializing sequence. The initializing sequence v_0 to v_i as shown in Figure 3.6 from [37] brings the circuit into a known state. At the end of the initialization phase, the flip-flops are set to states necessary for path activation. The clock is run at a slower rate so as to ensure that the circuit is initialized irrespective of any delays.

2) Path Activation Phase:

The path is activated by creating a transition at the beginning of the path by the two vectors v_{i+1} and v_{i+2}. The path is mainly sensitized during the second vector and during

the first vector, the path is sensitized only through those gates that propagate a non-controlling value similar to that in [33]. Once v_{i+2} is applied, the flip-flops are clocked at the rated clock period. If there exists a delay on the signal arriving at the input of the flip-flop that exceeds the rated clock period, the value latched in the flip-flop is a faulty value otherwise the flip-flop latches the fault-free value.

3) Fault Propagation Phase:

The vectors v_{i+3}, …. are propagation vectors that propagate the latched value in the destination flip-flop of the path to an observe point. The propagation vectors are also applied at a slower clock as the initializing vectors to ensure fault-free operation.

2.3.3.1 Test Generation Model

For a given path and transition, the netlist is modified in which a test to detect a single stuck-at fault is generated which in turn detects the path delay fault. The stuck-at fault functions as follows-

1) The activation of the stuck-at fault must happen only when the activation vectors have been applied to the combinational logic. The initialization vectors precede the path activation vectors.
2) The stuck-at fault should not interfere with the normal operation of the circuit. Once the second vector for path activation is applied, the stuck at fault is activated and the fault effect is injected into the destination flip-flop of the path.
3) Once the fault effect is latched in the destination flip-flop, the stuck-at fault should have no effect on the circuit and should allow fault-free operation of the circuit during the fault propagation phase

In order to test a path between two flip-flops, a logic block consisting of a few combinational logic gates with two flip-flops are inserted into the circuit. A stuck-at 1 fault is introduced at the output of the added logic block. The fault effect from the stuck-at 1 fault is inserted using an AND or OR gate depending on the direction of the

transition. The AND or OR gate is inserted between the output of the combination path beginning from the source flip-flop and the input of the destination flip-flop.

The test generation system uses three programs – a path generator that generates the paths, a stuck-fault model builder that reads the path and builds two models for rising and transition fault and STEED, a sequential test generator. STEED generates test for the stuck-at fault and initialization sequence and the propagation vectors.

The main drawback of the approach is the complexity involved and is impractical for very large sequential circuits as run-time is long due to page faults.

3.3.4 NEST: A Non-enumerative Test Generation Method

In [40], the authors propose a test generation technique for path delay faults that is based on the method presented in [41] which generates tests for path delay faults of a given circuit without explicitly enumerating the paths. The major challenge in path delay fault testing is the exorbitantly large number of paths in the designs which in the worst case can be exponential in the number of lines in the circuit. In [41], a labeling procedure is used that assigns labels to appropriate lines in the circuit and counts the number of paths and the number of faults detected. The method proposed is based on the fact that a large number of path delay faults can be detected by propagating transitions robustly through portions of the circuits, without enumerating all the paths through which the transitions are propagated. The labeling procedure considers only single lines and not paths. Sub-circuits with large number of paths going through them and which can be tested simultaneously are identified. Test generation objectives are determined for every sub-circuit identified. These objectives ensure that a large number of faults are tested by the same test without having to enumerate the paths. Tests can be generated for paths that go through new lines which ultimately lead to covering the complete path delay faults in the design. By considering new lines in the design, only a number of faults equal to twice the number of lines in the circuit are targeted.

In order to detect a large number of faults, two lines l1 and l2 are selected in the circuit such that the pair (l1, l2) has the maximum number of paths. Test generation objectives to propagate transitions from l1 to l2 robustly through as many paths as possible are found. Additional objectives are added to robustly propagate transition from a primary input to l1 and l2 to primary output. Apart from these objectives, another constraint imposed on the selection of l1 and l2 is to detect a large number of paths between them by the same test. This however may not be possible always as there could be odd and even parity paths due to inverters. Thus, instead of selecting line l1 that has maximum number of paths to l2, the line l1 that has maximum of either odd parity or even parity paths is selected. Once line l1 is selected for every line l2, the pair (l1, l2) that has the maximum number of paths is chosen.

During test generation, a single path from l2 to l1 is traced back. The output of l_2 is assigned a transition that allows all the inputs of the gate to be assigned a transition based on Table 3.1. Therefore for g_2 being an AND gate, a 0x1 transition is assigned. If for a gate, only one of its inputs can be assigned a transition then an input that belongs to the selected paths from l1 to l2 is chosen. Test generation objectives are collected while tracing the path from l2 to l1 and if the objectives can be satisfied then they are stored. Once l1 is reached, backtrace is done from l1 to the last gate where there exists a choice among the inputs. A different input is selected and test generation objectives are determined for the sub-path from the selected input to l1. This process continues until all the paths from the decision point are checked and a decision point preceding it is selected and the process is repeated. If a decision point is reached a second time, backtrace stops there and hence each line is considered only once.

The major advantage of this approach is that the main drawback of the path delay fault model which is the large number of faults that needs to be considered is overcome by not having to enumerate all the faults. This method is very effective for highly testable circuits.

Table 3.1 Robust propagation requirements (output) [40]

Gate type	Output transition	
	0x1	1x0
AND	Any number of inputs	Single input
OR	Single input	Any number of inputs
NAND	Single input	Any number of inputs
NOR	Any number of inputs	Single input

2.3.5 Segment delay fault model

In [42], the authors present a method to overcome the problem of testing critical paths that are untestable. Critical paths in a circuit are paths with the largest delay and are important for delay fault testing because a delay defect on such paths can cause a timing violation on such paths. In general, very small number of critical paths is testable [42, 43]. In [42], the untestable critical paths are tested robustly by covering the delay defects on the longest possible segments that are not covered by any testable critical path. The path selection is based on fixing a threshold for a given circuit and selecting paths that have length greater than the threshold. The threshold for each circuit is set such that there are a reasonable number of critical paths for the circuit.

A functional segment is defined as follows-

Definition: A functional segment (segment) is a sequence of connected gates, $S = g_0^{to}, g_1^{t1}, \ldots g_k^{tk}$, where the gate g_0 (g_k) need not be a circuit primary input (primary output), and $t_i \in \{r, f\}$ is the transition on gate g_i.

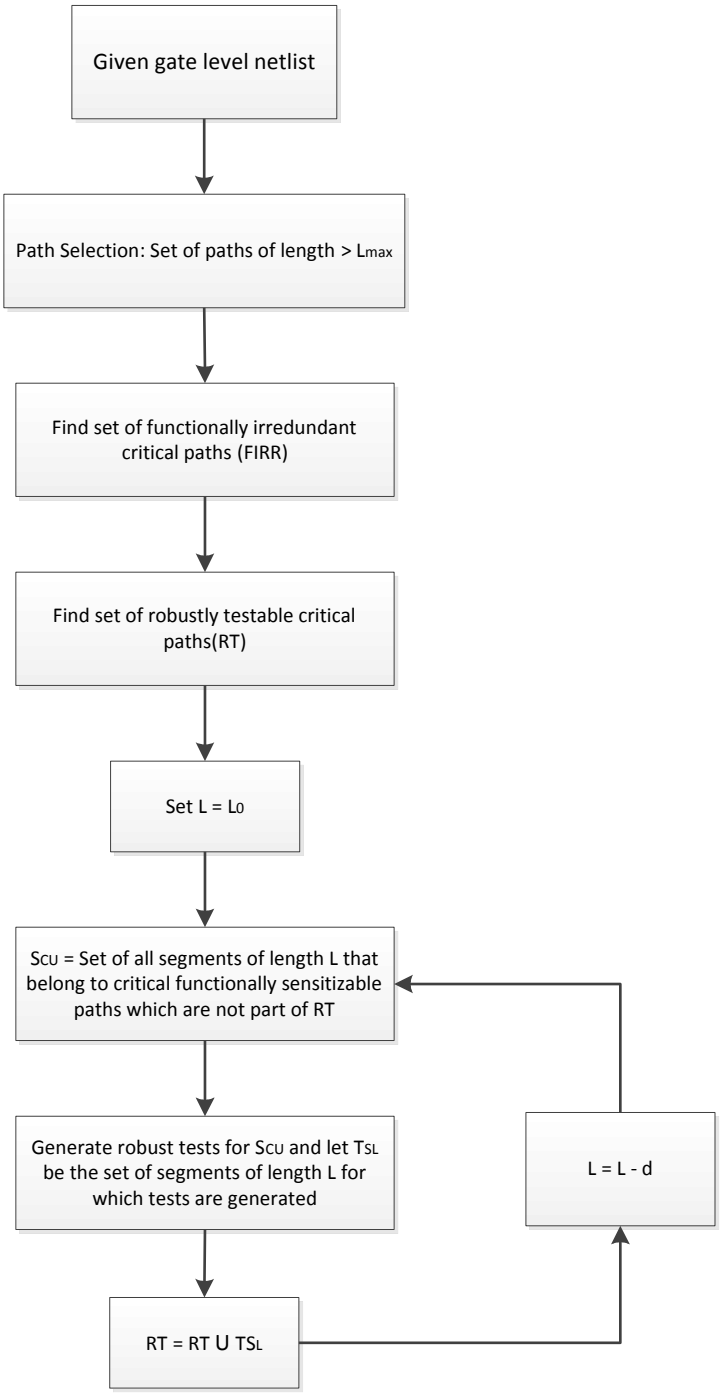


Figure 3.7 Atpg Flow [42]

The delay *D(S)* of a segment *S* is $D(S) = \sum_{i=0}^{k} \Delta_{g_i}^{t_i}$. All segments are assumed to be of fixed length $L(S)$ for segment *S*, where $L(S)$ is the number of gates in the segment S. A path begins from a primary input and ends at a primary output. If all the paths that pass through a segment S have a delay that exceeds the clock period, then the segment is said to be faulty. The problem of low robust coverage can be overcome by generating tests for uncovered segment which cannot be a part of any robustly testable. The overall flow to increase robust fault coverage by targeting uncovered segments of untestable but irredundant critical paths is shown in Figure 3.7.

The advantage of this approach is that a large number of paths for which robust tests cannot be generated can be covered by robustly testable segments. These segments implicitly cover more than 87% of the irredundant paths for ISCAS circuits in [42].

3.4 The Proposed Method

In this section, we propose an effective path delay pattern generation algorithm for multi-segment paths in partial scan designs. The proposed method introduces the concept of sustaining in order to sensitize a path which has intermediate sequential elements between the source and destination gates of the path. We also propose a method to increase the fault coverage for circuits which have low robust coverage by dividing the paths into sub-paths and generating tests for the sub-paths.

3.3.1 Motivation

The objective of this research is to propose an effective path delay atpg method to generate patterns for multi-segment paths. One of the applications of path delay patterns is they can be used for finding speed paths in a design. In high performance circuits, speed path debug is an important part of the design process in order to meet performance requirements [44, 45]. Speed paths are the frequency limiting paths identified during debug. Based on the causes of speed path failures the design practices can be improved.

Functional test patterns are used by debug engineers to identify speed paths in the design and provide accurate results. However, generation of functional patterns is expensive and the application cost is also high because the number of patterns is large and requires functional testers. Speed paths are identified by shrinking the clock period of each individual test cycle run for the failing test. This helps identify critical clock cycles when the shrinking of the clock causes failure. The use of functional test patterns may be time-consuming since the functional test patterns are very long and it might take many cycles to reach an observable node. It is important to identify the causes of speed failure so that the designers can develop strategies for better power and performance.

The various methods to test for speed paths are using functional patterns, n-detect transition patterns, path delay patterns. Since usage of functional patterns is not feasible due to the above reasons, the industry is interested in alternate methods to identify speed paths. One alternate method is to use n-detect transition tests. The disadvantages of n-detect transition atpg are that the size of the test pattern set can be very large and n-detect transition may or may not target critical paths in the design. Transition atpg also has the disadvantage that it may sensitize paths which are not critical and may lead to wrong paths being identified as speed paths. All these reasons motivate us to generate path delay patterns for speed path debug.

In the previous section various test generation methods for path delay faults were reviewed. It has been observed that it is challenging problem to sensitize a path consisting of sequential gates. In [37], it was seen that the method proposed consisted of inserting a complex logic block into the circuit to generate tests for paths in sequential circuit beginning from a flip-flop and ending in a flip-flop. In the work we present here, we propose a simple path sensitization approach that can be used to generate pseudo-robust tests, which are near robust tests and will be discussed in the next section. In the circuits today, there are multiple clock domains and thus source and destination latches of a path could be controlled by clocks of same or different domain. Assuming that a path

can be represented as a set of contiguous segments where a segment begins at a sequential boundary (with the exception of the first gate of the path which can be a primary input or combination gate), the issue of sensitization in multiple clock domains can be handled by sustaining the previous segment until the latch of the next segment becomes transparent i.e. when the clock is turned on.

The contributions of the work are:

1) In this work we present a path delay ATPG methodology as described above and compare the performance of the patterns generated by path delay ATPG with patterns generated by n-detect transition ATPG on silicon.
2) The coverage for some designs is still low, so we also propose a method by which the fault coverage of a design can be increased by dividing the paths into sub-paths and generate test for the sub-paths.

3.3.2 Test Generation Methodology

In this section the main procedures for test generation are described and we describe the path sensitization conditions and path sensitization approach.

3.3.2.1 Preliminary:

Path Definition:

In this section we define a path model and type of sensitization used in this work. A path P is defined as, $P = \{S_0, S_1, \ldots, S_n\}$, where P is a set of an ordered set of segments $S_0, S_1, \ldots, S_n$. A segment is an ordered set of gates, $S = g_0, g_1, \ldots, g_n$ and starts with a sequential element and ends at the gate driving the next sequential element. The first segment starts at a combinational element or a primary input and the last segment ends at any gate. An example for a path is shown in Figure 3.8. The path is shown as highlighted in the Figure 3.8. The path consists of segments S1 and S2. Segment S1 consists of latch L1 followed by gates G1 and G2 and since L2 is a

sequential it belongs to a different segment S2. S2 consists of latch L2 followed by gates G3 and G4. For combinational circuits a path can only have one segment whereas a sequential circuit could have one or more segments. The on-path inputs to the path are a, b, d, f, g and i. The off-path inputs of the path are clk1, c, e, clk2, h and j. A path can be sensitized by launching a transition at the beginning of the path and satisfying necessary off-path conditions specified by type of sensitization.

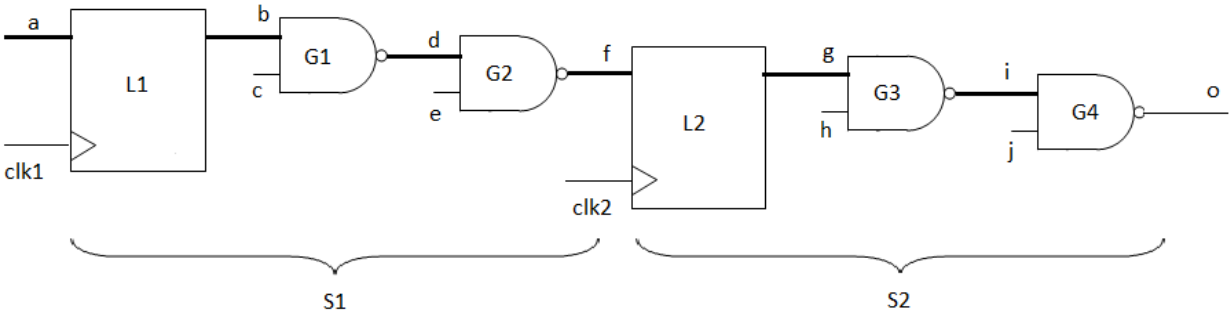


Figure 3.8 Example path delay fault

The different ways in which a path can be sensitized are robust sensitization and non-robust sensitization. We described pseudo robust sensitization previously, which has slightly relaxed conditions than robust sensitization and more stringent conditions than non-robust sensitization. We use pseudo-robust sensitization in this work.

3.3.2.2 Overall Path delay ATPG Flow

Path delay ATPG consists of three steps – path activation, path sensitization and fault propagation. Path activation creates the required transition on the first gate of the first segment or the first gate of the path. Once the transition is set on the output of the first gate of the path, the path is sensitized by assigning necessary values to the off-path

inputs depending on the type of sensitization. Once all the gates of the path are sensitized, the fault is propagated to a primary output or scan cell using sequential ATPG. The overall test generation flow is shown in Figure 3.9.

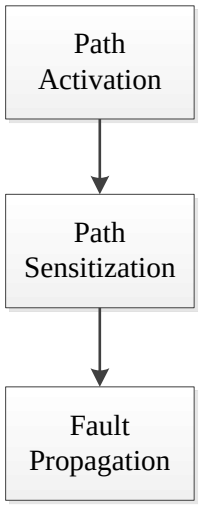


Figure 3.9: Overall Test Generation Flow

3.3.2.3 Path Sensitization

Once activation is completed, the path sensitization begins and the transition from the first gate of the first segment is propagated to the end of the segment or end of the path. Once the last gate of the last segment is sensitized the path sensitization is complete. In other words, path sensitization is achieved by sensitizing all the segments of the path. A segment is said to be sensitized if all the gates of the segment are sensitized. In turn a gate is said to be sensitized if the sensitization at the input of the gate is successfully propagated to the output of the gate under necessary off-path conditions. A gate that requires necessary off-path assignments to be made is called a p-frontier and is described below.

3.3.2.3.1 P-frontier

If a gate needs to be assigned appropriate off-path inputs either in previous or current frame, it is called a path frontier gate or p-frontier gate. A combination gate is a p-frontier if atleast one of the inputs is sensitized and the output of the gate is not sensitized yet and none of the off-path inputs prevents from propagating the transition from input of the gate to output of the gate. A sequential gate is a p-frontier if the output is not sensitized yet and one of the inputs is sensitized and clock is unknown. Sensitization of the path is achieved by assigning proper off-path inputs to all p-frontier gates. The pseudo-robust sensitization of AND, multiplexer and D-latch gates are discussed below:

1) Pseudo-robust sensitization of AND gate:

For a rising transition at the on-path inputs as shown in Figure 3.10(a), the final value of the off-path input should have a non-controlling value, whereas for a falling transition at the on-path input, the off-path input requires to have non-controlling value in both the first and second timeframes. Unlike robust sensitization where stable value is required on off-path input, pseudo- robust sensitization requires the off-path values to be the same in previous and current time frame. The pseudo-robust conditions for NAND, OR/NOR gates are similar and are not shown here.

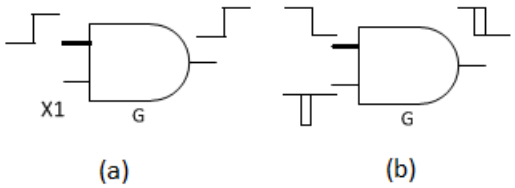


Figure 3.10: Pseudo-robust conditions for AND gate
a) rising transition and b) falling transition

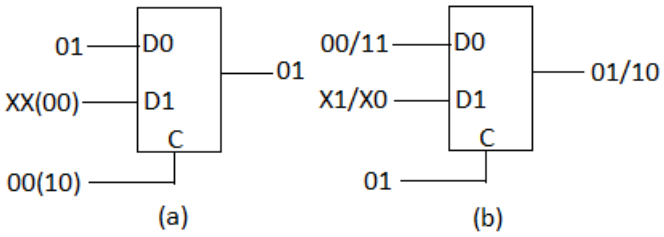


Figure 3.11 Pseudo-robust conditions for MUX gate
a) data pin on-path and b) control pin on-path

2) Pseudo-robust sensitization of MUX gate:

The pseudo-robust sensitization of a multiplexer when the data pin is on-path is shown in Figure 3.11(a). When input D0 is on-path and is sensitized, i.e. the gate driving the fanin-pin of the multiplexer is sensitized, then the control input should be 0 in the current frame and either 1 or 0 in the previous time frame. Inputs D1 and control pin are off-path. When the value on control pin is 0 in the previous and current time frames, input D0 is selected in both the time frames. Thus the delayed transition is observed at the output of the multiplexer. In this case the value on D1 has no impact on the output since D1 is not selected. The initial value of the transition can also be obtained from the off-path input, but the final value should be obtained from the on-path input. Therefore, when the control selects input D1 in previous time frame, the initial value at the output of the multiplexer should be set by input D1. When the control input has a delay free transition, it correctly sets the initial value of the transition at the output in the previous time frame and the delayed transition of the D0 input is observed at the output when control pin switches to 0 in current time frame. If there occurs a delay on control input, the initial value should still be set at the output. This is achieved by setting the same initial value of transition 0 on input D1 in both the timeframes.

When the control input is on-path, and D0 and D1 are off-path inputs, consider a 01 transition on the control input. The pseudo-robust condition of a multiplexer when the control pin is on-path is shown in Figure 3.11(b). The initial value of the transition at the

output is set by the D0 input as the control input selects D0 in previous time frame. So D0 should have the initial value of the transition 0(1). The final value of the transition at the output is set by the D1 input as the control input selects D1 in current time frame. When there is a delay on control input the value still remains 0 in the current time frame and thus selects input D0. So the output should still have the initial value of the transition and so the value of D0 should be maintained in both previous and current time frames.

3) Pseudo-robust sensitization of D-latch gate:

The pseudo-robust conditions for sensitizing a D-latch can be explained with an example shown in Figure 3.12. In this work we only evaluate sensitization conditions for the case when the data pin is on-path.

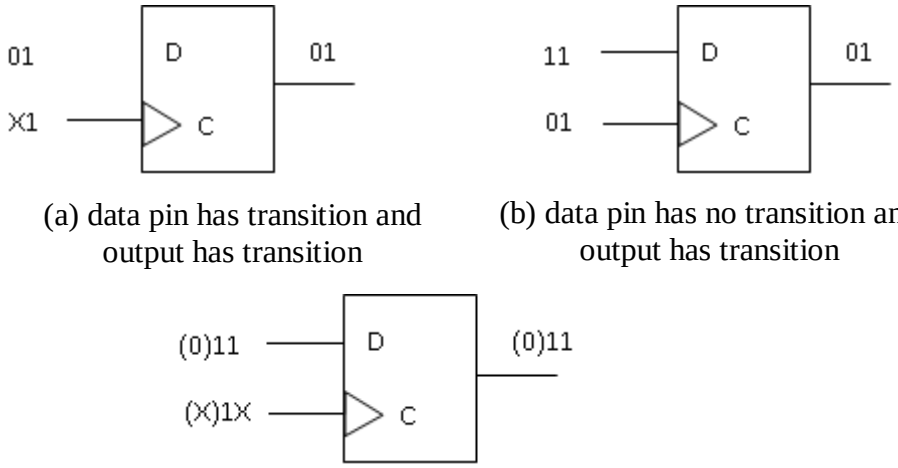
a) Data pin is on-path:

When the data pin is on-path, the gate can be sensitized under the following conditions:

i) When data pin is sensitized, i.e. the data input has a transition and the fanin input gate is sensitized in second frame, the gate can be sensitized if the clock is turned on in the second time frame and thus the gate has a transition at the output.

ii) If the data input is sensitized in both the timeframes and has same value in the previous and current time frame and the output of the gate has a transition when the clock turns on in current frame (clock is off in the previous frame in this case).

iii) If data pin has no transition and gate was sensitized in the previous time frame, the gate can be sensitized.



(a) data pin has transition and output has transition

(b) data pin has no transition and output has transition

(c) data pin has no transition and output has no transition and gate sensitized in previous frame

Figure 3.12 Pseudo-robust sensitization of D-latch for rising transition at data pin

Cases ii and iii will be explained in the next section. The above conditions for sensitizing a gate when the data pin is on-path is shown in Figure 3.12 (a), (b) and (c) respectively.

3.3.2.3.2 Sustaining

In the circuits today, there are multiple clock domains and thus source and destination latches of a path could be controlled by clocks of same or different domain. In addition to this, critical paths in a design may not be limited to latch to latch or flop to flop and can include more than one sequential along the path. In order to propagate the sensitization from one segment (S1) to the following segment (S2), the final value of the transition on segment S1or the sensitization of segment S1 needs to be maintained or sustained until the downstream latch is transparent and the final value of transition is latched. By sustaining the value the transition on the previous segment, the delay effect

on the previous segment is potentially maintained by setting non-controlling values on the off-path inputs until the next segment's latch becomes transparent.

In order to sustain a segment the final value of transition should be maintained on all the on-path pins of segment S1. The off-path inputs of segment S1 also should be sustained from the previous time frame. This is shown in Figure 3.13. Consider a three segment path as shown in Figure 3.13. The on-path inputs are shown with dark lines. In order to sensitize the path, segment S1 needs to be sensitized first. Since latch L1 is transparent in phase 2 of cycle 1, where one cycle has four time frames, the path can be activated by creating a transition on output of latch L1. Gates G1and G2 can be sensitized with pseudo-robust conditions on the off-path inputs. Thus by sensitizing every gate of the segment, S1 is sensitized. Since latch L2 is not transparent in phase 2 of cycle 1, segment S1 should be sustained in phase 3 when latch L2 is transparent. This is done by maintaining the final value of the transition on all the on-path inputs of gates of S1 and the off-path inputs should maintain the same value as the previous time frame. Once L2 is sensitized in phase 3, gates G3 and G4 can be sensitized with pseudo-robust conditions. Latch L3 is transparent in phase 1 of a cycle, so segment S2 should be sustained until phase 1 of cycle 2. Thus the path is sensitized in phase 1 of cycle 2.

In order to summarize the sensitization with respect to the conditions explained previously for D-latch, latch L1 is sensitized for the first time when clock turns on and the output has a transition. For the first gate of the path, since there is no on-path input, the gate is considered to be sensitized if the necessary transition is set at the output. After necessary off-path assignments are made to the gates of segment S1 in timeframe as latch L2 is not transparent in phase 2 of cycle 1, segment S1 is sensitized. Condition iii applies for latch L1 and gate is marked sensitized. Latch L2 is sensitized in phase 3 according to condition ii.

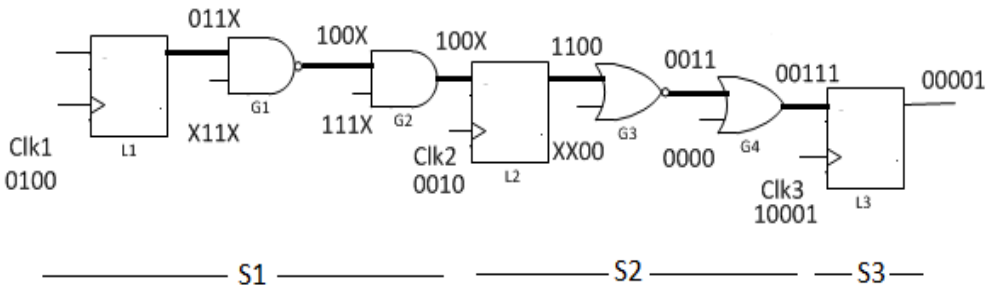


Figure 3.13 Path sensitization with Sustaining

3.3.2.4 Fault Propagation

The test generator uses a split circuit model. So, once the last gate of the segment is sensitized, there is a divergence created between good and the faulty machine at the output of the last gate. In order to propagate the transition to a scan node or output, sequential ATPG is used.

3.3.3 Segment Delay Fault Testing to Improve Fault Coverage

The coverage achieved by pseudo-robust sensitization may not be satisfactory in most cases and so we introduce the segment delay fault model similar to the method used in [42] to alleviate the problem of low coverage. All the faults that are targeted for test generation are classified into test found, aborted and redundant. The faults which are aborted or redundant are divided into sub-paths. These sub-paths are targeted for test generation. The motivation behind picking redundant faults for test generation is that, a complete path that is redundant under pseudo-robust condition may still be covered by tests generated for sub-divisions of the original path. Also, the delay may not be evenly distributed along the path. It is possible that the delay may be more concentrated in certain portions of the path. Thus the paths can be divided and tests can be generated for the sub-paths. The aborted faults are also divided into sub-paths because, the test may be

aborted when the entire path is targeted however, a test can be potentially generated for a portion of the path.

The faults in the given path delay fault list are first targeted and if a test is not generated for a fault then the path is divided into two half sub-paths. The two half-paths or segments are then targeted and tests are generated if possible. If test generation is not possible for a segment, the segment is further divided into half and the process repeats until a segment consists of only two gates. The overall flow for segmented delay fault testing is shown in Figure 3.14.

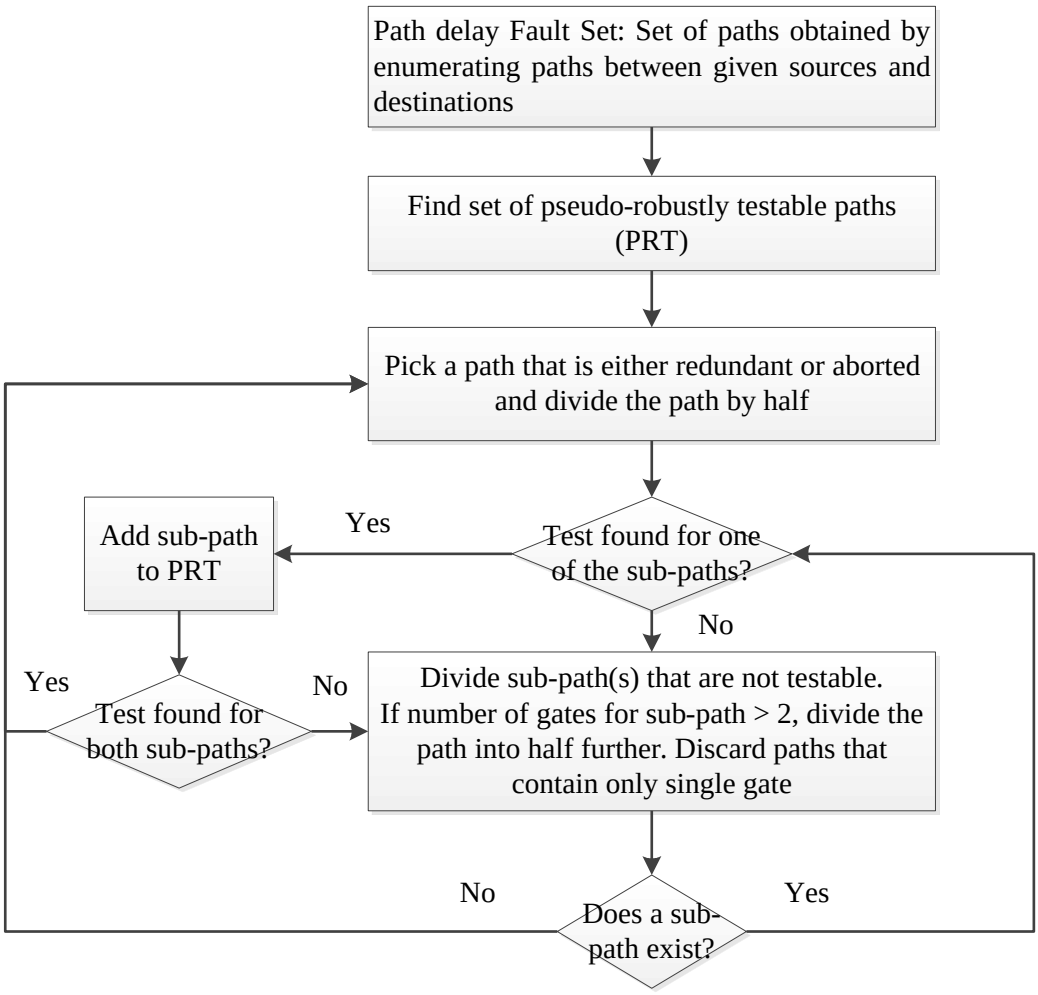


Figure 3.14 Overall flow for Segmented Delay Fault Testing

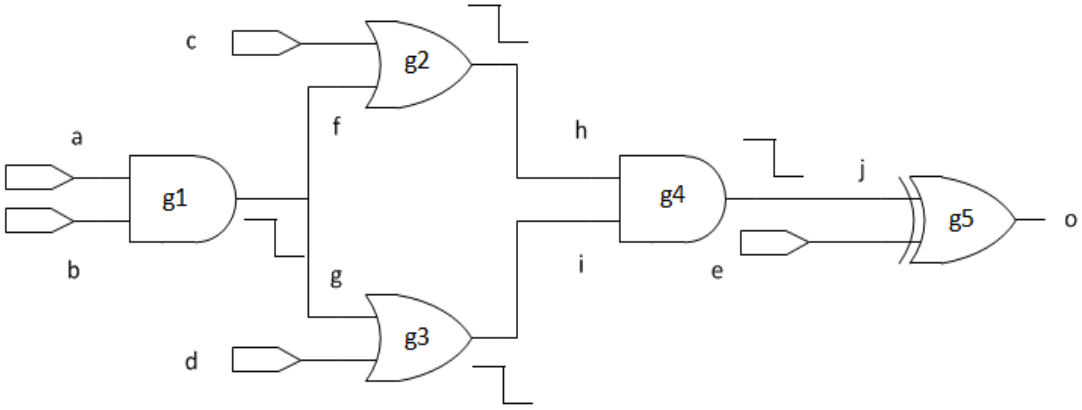


Figure 3.15 An example circuit depicting paths that are robustly untestable

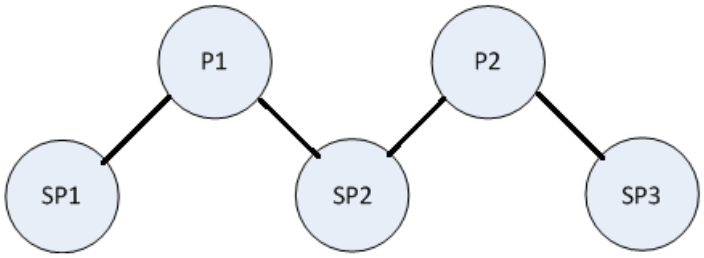


Figure 3.16 Tree structure for path storage

The paths are stored in an efficient tree structure so that duplication of paths can be avoided. Consider a simple circuit in the Figure 3.15. The paths P1 = {g1, g2, g4, g5} and P2 = {g1, g3, g4, g5} are robustly untestable. However, if we divide the path P1 into sub-paths SP1 = {g1, g2} and SP2 = {g4, g5} and path P2 into sub-paths SP3 = {g1, g3} and SP4 = {g4, g5}. It can be observed that the sub-paths SP2 and SP4 are the same and hence it is sufficient to store a single path to avoid duplication. The paths are stored in a tree structure as given in Figure 3.16. Every path (or sub-path) corresponds to a node in the tree. Path P1 is has two child nodes corresponding to SP1 and SP2. Once a node a

path is detected, all the sub-paths (child nodes) are marked detected and hence if an aborted or redundant fault consists a node that has one or more of the sub-paths already marked as detected, targeting them can be avoided.

3.4 Experimental Results

The experiments were carried out on industrial circuits and the effectiveness of the test patterns generated was demonstrated on silicon. The patterns generated by the path delay test generator were used for speed path debug and the performance was compared with n-detect transition fault tests. The paths were selected using the on die clock shrink mechanism [46] or critical path sourcing methodology. This methodology helps identifying candidates for critical paths by manipulating frequency or duty cycle of clock for one or more test cycles. The candidates for speed path are reported in terms of source and destination latches. This information is used by a path enumeration utility to enumerate the paths between the given source and destinations using depth first search.

Tests are generated for the paths using the proposed test generation process described in previous section. The results are shown in Table 3.2. Column 1 gives the circuit name, column 2 is the number of paths enumerated for the given sources and destinations identified by critical path sourcing methodology. Columns 3, 4 and 5 give the number of faults that are detected by robust sensitization, number of aborted faults and number of redundant faults.

Table 3.2 Fault Statistics

Circuit	Number of Paths	Num Faults det	Num faults abrt	Num faults Red
Circuit A	752	273	261	218
Circuit B	42	42	0	0

As can be seen that circuit B has 100% fault coverage while circuit A has a fault coverage of 30% and an effectiveness of 42%. Effectiveness is defined as the percentage of faults for which test is found with respect to total faults less the redundant faults. The test pattern set generated for Circuit A was run on silicon and the performance of path delay ATPG patterns was compared against transition n-detect patterns, where n=10. A two dimensional plot of frequency versus IDV is given in Figure 3.17. The graph plots frequency versus IDV for three sets of test patterns – functional test patterns, transition n-detect patterns and path delay atpg test patterns. IDV stands for Intra-die-variation which is a measure of process variations across the dies. Though the dies are from the same wafer but the behavior of dies varies due to process variations. The graph is plotted for 93 dies and each point in the graph denotes the maximum frequency at which a die fails. Path delay patterns identify failures at a frequency lower than the transition patterns. The closer the gap between the patterns (n-detect or path delay) and functional patterns, the more accurate are the patterns in identifying the speed paths. If the gap between the patterns is too large it implies that the patterns may not identify actual critical paths. When at-speed tests like transition ATPG or path delay patterns are used to test speed paths, there are power droop issues because of loading or unloading of values in the scan chains which might create illegal conditions in the design which would otherwise not be possible during normal function of the design [55]. The power droops are accounted for by providing guardbands.

In order to improve the fault coverage for circuit A, the path delay faults that are either aborted or redundant are divided into sub-paths and the procedure described in Section 3.3.3 is performed. For the 752 faults in Circuit A, there 2659 unique segmented path delay faults generated by dividing the path into sub-paths and avoiding duplicate faults in the fault list by using a tree structure. When path delay ATPG is performed on the faults with pseudo-robust sensitization conditions, the statistics of the test generation is reported in Table 3.3.

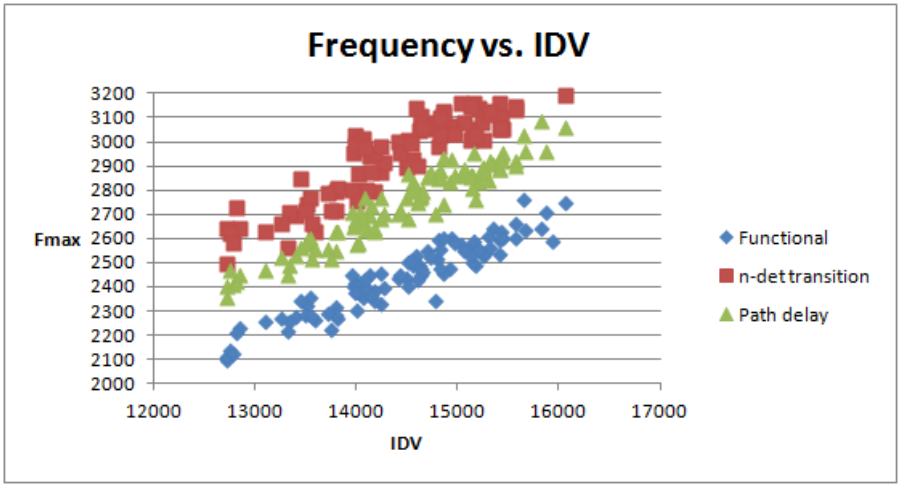


Figure 3.17 Frequency versus IDV

Table 3.3 Fault Statistics for path delay ATPG with segmented path delay faults

Level	Detections	Aborts	Redundants	Total Faults	%Det
0	273	261	218	752	36.3
1	1090	286	128	1504	72.4
2	2851	97	60	3008	94.7
3	5331	79	30	5440	97.9
4	722	2	0	724	99.7

The fault statistics reported in Table 3.3 represent the number of faults that are detected, aborted and redundant respectively in columns 2, 3 and 4 at each level shown in column 1. In the statistics reported, all the sub-paths are counted independently at each level and therefore even though the tree structure facilitates re-using the same node in the tree for multiple paths having the same sub-path, for reporting purpose we consider all the sub-paths at each level. It can be observed that twice the aborted and redundant faults in level k is equal to the total faults in the next level k+1. However, as we move to higher

levels, from level 3 to 4, this does not hold true because all the paths are not of the same length. This can be explained as follows. Consider a path P1 of length 4 and path P2 of length 8. The sub-paths enumerated for path P1 belong to level 1, whereas the sub-paths enumerated for path P2 belong to levels 1 and 2. Therefore, as we move on to higher levels the number of nodes in the next level may not be twice the number of nodes in previous level.

Without the segmented path delay ATPG, there are only 273 paths, for which test is found. It can be observed that, as we move to higher levels the percentage detections increase, which is expected because as the length of the path reduces, the chances of detecting a fault is more. This in turn potentially increases the chances of detecting a critical path. The quality of the patterns or the chances of detecting critical paths on silicon is determined by the number of detections at each level. If there are more number of detections in the earlier levels, it would potentially increase the chances of detecting critical paths.

3.4 Conclusion

In this work, we propose path delay ATPG algorithm for partial scan designs using pseudo-robust sensitization condition. Sensitization is propagated by sustaining segments where there are sequential elements on the path. Sustaining plays an important role in multiple clock domain designs. In order to improve the pseudo-robust fault coverage, we propose a method of dividing the paths into sub-paths and generating tests for the sub-paths. Path delay ATPG patterns can be used as an alternative to functional patterns for identifying speed path failures. Experiments on industrial designs demonstrate the effectiveness of path delay patterns over n-detect transition patterns.

In order to improve fault coverage, we propose to divide paths into shorter paths and generate tests for portions of the path for which test is not found. Since the pseudo-robust condition could be strict and it might not be possible to generate tests for entire

path, the path can be divided and test can be generated for the shorter paths assuming that the delay along the path may not be uniformly distributed and a path which is found to be aborted or redundant can potentially have a test for a shorter path. We demonstrate the effectiveness of this approach on industrial design.

CHAPTER 4 CONCLUSIONS AND FUTURE RESEARCH

4.1 Conclusions

With decreasing size of transistors, there are more number of gates integrated on the chip thereby increasing the density and complexity of the chip. Due to increasing complexity of designs, it necessitates the requirement to test the designs for defects. Testing of manufactured chips directly impacts the overall cost. In this work, we address two problems related to testing for failures on designs – (i) large test set size which directly impacts the test application time, storage requirements and testing cost and (ii) functional patterns when used for speed path debug could be really expensive

In chapter 2, we address the issues related to using large test set size for testing designs. The size of the test set directly impacts the test application time which is directly proportional to the product of the number of test patterns and the number of scan cells in the longest scan chain. In addition to test application time, the test set size also impacts the storage requirements. We propose an incremental dynamic compaction for partially scanned designs. Typically the fault coverage curve of designs ramp up quickly initially and slows down after some time and flattens in the tail portion of the curve. The cube merging method, which is the basic compaction initially used in the test generation tool, does not produce compaction friendly patterns in the tail of the curve. The proposed incremental dynamic compaction method is suitable for designs for which the fault coverage curve has a long tail by generating compaction friendly patterns using dynamic compaction after a certain threshold in the fault coverage curve is reached. The method exploits the benefit of cube merging in the initial region of the coverage curve and later switching to dynamic compaction. Initially we proposed a method in which the parameters for dynamic compaction are manually provided based on the fault coverage curve. We demonstrated the effectiveness of the method on industrial designs with test size reduction 36% and run time upto 4X times the cube merging method. We also

proposed static untestability analysis method to address long run time, which checks for activation violation and existence of a potential x-path to an observable point. Some designs benefited from the method while the others had longer run-times. In addition to static untestability analysis approach, a reasoning analysis method is proposed. This method drops secondary faults that are redundant hence avoiding re-targeting of redundant faults. As further enhancement, we automated the identification of parameters for dynamic compaction, where the parameters are evaluated during the initial phase of test generation where cube merging is performed. Experiments conducted on the industrial designs demonstrated the effectiveness of the method. The method provided 30% compaction with 2X times the run time of cube merging when 32 bins were used in both the methods. We conducted further experiments with larger number of bins. Since using automatic parameter identification method would be run-time intensive, we propose to use smaller bin size for the dynamic compaction as opposed to larger bin size used for cube merging. The effectiveness of the method is demonstrated on industrial designs.

In the completed research presented in Chapter 3, we presented a path delay fault test generation methodology for partially scanned designs which is used to generate test patterns to be used for speed path debug. There are various methods to test for speed paths - functional patterns, n-detect transition patterns, path delay patterns. However, the usage of functional patterns is not feasible because functional pattern generation is expensive and the application cost is also high because the number of patterns is large and functional testers are required for testing. In the proposed method, a simple path sensitization approach is presented, that can be used to generate pseudo-robust tests, which are near robust tests. In this work, a path is represented as segments, where typically each segment begins at a sequential and ends at gate driving the next sequential with some exceptions. The sensitization in multi-clock domain designs can be handled by sustaining the previous segment until the latch of the next segment becomes transparent

i.e. when the clock is turned on. It was observed that the path delay patterns demonstrated better performance when compared with n-detect transition atpg patterns. The path delay patterns identified speed path failures at lower frequencies than the n-detect transition patterns, thus providing more accurate critical path debug. Another issue that is handled in the proposed work is that the pseudo-robust fault coverage is low because of the strict conditions on the off-path inputs of gates. In order to improve the fault coverage, we propose a method where the paths that are aborted or redundant are sub-divided iteratively.

4.2 Future Work

The incremental dynamic compaction technique which is based on identifying a threshold and switching to dynamic compaction after cube merging successfully addressed the issue of long fault coverage tail by providing upto 30% compaction over the cube merging method in 2X run-time with bin size 32. The method is effective in the fault coverage tail where compaction friendly patterns are necessary. However, there are potential areas for improvement for the proposed work.

The run-time associated with the proposed method with smaller bin size is reasonable. However, as the number of bins used is increased, even though the compaction achieved is appreciable, but the run-times are long. The reason for the increased run-time is the number of secondary faults selected per primary fault is large in some of the cases, which does not benefit dynamic compaction, however impacts the run time. In some cases, it might benefit if the threshold is moved further down the tail of the fault coverage curve. Future research work can investigate potential solutions to improve run time when large bin sizes are used. Another issue that was observed was for two of the test cases, there was loss in fault coverage and the cause for the coverage loss was due to single test cycles being used. The current work can be extended to multi-test cycles.

REFERENCES

[1] Rudnick, E.M.; Patel, J.H.;, "Efficient techniques for dynamic test sequence compaction," *Computers, IEEE Transactions on* , vol.48, no.3, pp.323-330, Mar 1999

[2] M. L. Bushnell and V. D. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits. Boston: Springer, 2005.

[3] Laung-Terng Wang, Cheng-Wen Wu and Xiaoqing Wen, "VLSI Test Principles and Architectures: Design for Testability",The Morgan Kaufmann Publishers, 2006

[4] K. T. Cheng and V. D. Agrawal, Unified Methods for VLSI Simulation and Test Generation. Boston: Kluwer Academic Publishers, 1989.

[5] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Digital Systems Testing and Testable Design," IEEE Press, Piscataway, NJ, 1994.

[6] Millman, S.D.; Acken, J.M.; , "Diagnosing CMOS bridging faults with stuck-at, IDDQ, and voting model fault dictionaries," *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994* , vol., no., pp.409-412, 1-4 May 1994

[7] K.Y. Mei, "Bridging and Stuck-at Faults", in IEEE Transaction On Computers, vol. C-23(7, pp.720-727), 1974.

[8] Pomeranz, I.; Reddy, S.M.; , "Generation of Broadside Transition-Fault Test Sets That Detect Four-Way Bridging Faults," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.26, no.7, pp.1311-1319, July 2007

[9] Bernardi, P.; Reorda, M.S.; Bosio, A.; Girard, P.; Pravossoudovitch, S.; , "On the Modeling of Gate Delay Faults by Means of Transition Delay Faults," *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on* , vol., no., pp.226-232, 3-5 Oct. 2011

[10] Kwang-Ting Cheng; Krstic, A.; , "Current directions in automatic test-pattern generation," *Computer* , vol.32, no.11, pp.58-64, Nov 1999

[11] Cha, C.W.; Donath, W.E.; Ozguner, F.; , "9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits," *Computers, IEEE Transactions on* , vol.C-27, no.3, pp.193-200, March 1978

[12] Fujiwara, H.; Shimono, T.;, "ON THE ACCELERATION OF TEST GENERATION ALGORlTHMS," *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on* , vol., no., pp.350, 27-30 Jun 1995

[13] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, vol. C-30, no. 3, pp. 215–222, Mar. 1981.

[14] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," IBM Journal of Research and Development, vol. 10, no. 4, pp. 278–291, July 1966.

[15] Bommu, S.; Chandrasekar, K.; Kundu, R.; Sengupta, S.; , "CONCAT: CONflict Driven Learning in ATPG for Industrial designs," *Test Conference, 2008. ITC 2008. IEEE International*, vol., no., pp.1-10, 28-30 Oct. 2008

[16] B. Krishnamurthy and S. B. Akers, "On the Complexity of Estimating the Size of a Test Set," IEEE Trans. on Computers, vol. C-33, no. 8, pp. 750–753, Aug. 1984.

[17] Krishnamurthy, Balakrishnan; Akers, Sheldon B.; , "On the Complexity of Estimating the Size of a Test Set," *Computers, IEEE Transactions on* , vol.C-33, no.8, pp.750-753, Aug. 1984

[18] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the role of independent fault sets in the generation of minimal test sets," in *Proc. Int. Test Conf.*, Aug. 1987, pp. 1100–1107.

[19] Kajihara, S.; Pomeranz, I.; Kinoshita, K.; Reddy, S.M.; , "Cost effective generation of minimal test sets for stuck-at faults in combinational logic circuits," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1496–1504, Dec. 1995.

[20] I. Pomeranz, L. Reddy, and S. M. Reddy, "Compactest: A method to generate compact test sets for combinational circuits," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 194–203.

[21] G.-J. Tromp, "Minimal test sets for combinational circuits," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 204–209.

[22] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost effective generation of minimal test sets for stuck-at faults in combinational logic circuits," in *Proc. Design Automation Conf.*, June 1993, pp. 102–106.

[23] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 126–137, Jan. 1988.

[24] Y. Matsunaga, "MINT-An exact algorithm for finding minimum test sets," IEIC8 Trans. Fundamentals, vol. E76-A, pp. 1652-1658, Oct. 1993

[25] J.3. Chang and C.3. Lin, "Test set compaction for combinational circuits," in First Asian Test Symp., Nov. 1992, pp. 20-25

[26] P. Goel, and B. C. Rosales, "Test generation and dynamic compaction of tests," in Dig. Papers 1979 Test Con&, pp. 189-192, Oct. 1979

[27] "ROTCO: A reverse order test compaction technique," in 1992 ZEEE EURO-ASIC Con$, pp. 189-194, Sept. 1992.

[28] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Dept. Elect. Eng., Virginia Polytechnic Inst. State Univ., Blacksburg, VA, Tech. Rep. 12-93, 1993.

[29] I. Pomeranz and S. M. Reddy, "Forward-looking fault simulation for improved static compaction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1262–1265, Oct. 2001.

[30] S. Kajihara, H. Shiba, and K. Kinoshita, "Removal of redundancy in logic circuits under classification of undetectable faults," in *Proc. 22nd Fault-Tolerant Computing Syrnp.*, July 1992, pp. 263-270.

[31] S. B. Akers *et al.*, "on the role of independent fault sets in the generation of minimal test sets," *1987Int. Test Conf.*, pp. 1100-1107, Aug. 1987.

[32] S. B. Akers and B. Krishnamurthy, "On the application of test counting to VLSI testing," Computer Research Laboratory, Tektronix Laboratories, Technical Report No. CR-85-12, Apr. 1985.

[33] Chin Jen Lin; Reddy, S.M.; , "On Delay Fault Testing in Logic Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.6, no.5, pp. 694- 703, September 1987

[34] J.Savir and S. Patil, "Scan-Based Transition Test," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume: 12 , Issue: 8 , Aug. 1993 Pages: 1232 – 1241.

[35] Fuchs, K.; Fink, F.; Schulz, M.H.; , "DYNAMITE: an efficient automatic test pattern generation system for path delay faults," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.10, no.10, pp.1323-1335, Oct 1991

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

[37] Agrawal, P.; Agrawal, V.D.; , "A New Method for Generating Tests for Delay Faults in Non-Scan Circuits," *VLSI Design, 1992. Proceedings., The Fifth International Conference on*, vol.,no.,pp. 4-11, 4-7 Jan1992

[38] Malaiya, Yashwant K.; Narayanaswamy, Ramesh; , "Modeling and Testing for Timing Faults in Synchronous Sequential Circuits," *Design & Test of Computers, IEEE* , vol.1, no.4, pp.62-74, Nov. 1984

[39] Ghosh, A.; Devadas, S.; Newton, A.R.; , "Test generation and verification for highly sequential circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.10, no.5, pp.652-667, May 1991

[40] Pomeranz, I.; Reddy, S.M.; Uppaluri, P.; , "NEST: A Non-Enumerative Test Generation Method for Path Delay Faults in Combinational Circuits," *Design Automation, 1993. 30th Conference on* , vol., no., pp. 439- 445, 14-18 June 1993

[41] Pomeranz, I.; Reddy, S.M.; , "An efficient non-enumerative method to estimate path delay fault coverage," *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on* , vol., no., pp.560-567, 8-12 Nov, 1992

[42] Sharma, M.; Patel, J.H.; , "Testing of critical paths for delay faults," *Test Conference, 2001. Proceedings. International* , vol., no., pp.634-641, 2001

[43] Krstic, A.; Kwang-Ting Cheng; Chakradhar, S.T.; , "Identification and test generation for primitive faults," *Test Conference, 1996. Proceedings., International* , vol., no., pp.423-432, 20-25 Oct 1996

[44] Ruifeng Guo; Wu-Tung Cheng; Kun-Han Tsai; , "Speed-Path Debug Using At-Speed Scan Test Patterns," *Test Symposium, 2009 14th IEEE European* , vol., no., pp.11-16, 25-29 May 2009

[45] Killpack, K.; Natarajan, S.; Krishnamachary, A.; Bastani, P.; , "Case Study on Speed Failure Causes in a Microprocessor," *Design & Test of Computers, IEEE* , vol.25, no.3, pp.224-230, May-June 2008

[46] Josephson, D., Gottlieb, B. Advances in Electronic Testing – Challenges and Methodologies: Chapter 3 (Silicon Debug), pp. 77-108, Gizopoulos, D. (Editor), Springer, 2005, ISBN 0-387-29408-2.

[47] Hamzaoglu, I.; Patel, J.H.; , "Test set compaction algorithms for combinational circuits," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.19, no.8, pp.957-963, Aug 2000

[48] I. Hamzaoglu and J. H. Patel, "New techniques for deterministic test pattern generation," in *Proc. IEEE VLSI Test Symp.*, Apr. 1998, pp. 446–452.

[49] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark designs and a special translator fortran," in *Proc. Int. Symp. Circuits and Systems*, June 1985.

[50] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. Int. Symp. Circuits and Systems*, May 1989, pp. 1929–1934.

[51] Kajihara, S.; Pomeranz, I.; Kinoshita, K.; Reddy, S.M.; , "On compacting test sets by addition and removal of test vectors," *VLSI Test Symposium, 1994. Proceedings., 12th IEEE* , vol., no., pp.202-207, 25-28 Apr 1994

[52] J.-S. Chang and C.-S. Lin, "Test set compaction for combinational circuits," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1370–1378, Nov. 1995.

[53] Ayari, B.; Kaminska, B.; , "A new dynamic test vector compaction for automatic test pattern generation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.13, no.3, pp.353-358, Mar 1994

[54] Santiago Remersaro, "On low power test and DFT techniques for test set Compaction", Ph.D. Dissertation, University of Iowa, 2008.

[55] Pant, P.; Skeels, E.; , "Hardware hooks for transition scan characterization," *Test Conference (ITC), 2011 IEEE International* , vol., no., pp.1-8, 20-22 Sept. 2011

[56] Chandrasekar, K.; Bommu, S.; Sengupta, S.; , "Low Coverage Analysis using dynamic un-testability debug in ATPG," *VLSI Test Symposium (VTS), 2011 IEEE 29th* , vol., no., pp.291-296, 1-5 May 2011