High Quality Transition and Small Delay Fault ATPG

Puneet Gupta

Keywords: Path Delay Tests, Non-Robust Tests, Robust Tests, Transition faults, Delay Test, ATPG

HIGH QUALITY TRANSITION AND SMALL DELAY FAULT ATPG

PUNEET GUPTA

ABSTRACT

Path selection and generating tests for small delay faults is an important issue in the delay fault area. A novel technique for generating effective vectors for delay defects is the first issue that we have presented in the thesis. The test set achieves high path delay fault coverage to capture small-distributed delay defects and high transition fault coverage to capture gross delay defects. Furthermore, non-robust paths for ATPG are filtered (selected) carefully so that there is a minimum overlap with the already tested robust paths. A relationship between path delay fault model and transition fault model has been observed which helps us reduce the number of non-robust paths considered for test generation. To generate tests for robust and non-robust paths, a deterministic ATPG engine is developed. To deal with small delay faults, we have proposed a new transition fault model called As late As Possible Transition Fault (ALAPTF) Model. The model aims at detecting smaller delays, which will be missed by both the traditional transition fault model and the path delay model. The model makes sure that each transition is launched as late as possible at the fault site, accumulating the small delay defects along its way. Because some transition faults may require multiple paths to be launched, simple path-delay model will miss such faults. The algorithm proposed also detects robust and non-robust paths along with the transition faults and the execution time is linear to the circuit size. Results on ISCAS'85 and ISCAS'89 benchmark circuits shows that for all the cases, the new model is capable of detecting smaller gate delays and produces better results in case of process variations. Results also show that the filtered non-robust path set can be reduced to 40% smaller than the conventional path set without losing delay defect coverage.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Michael Hsiao for his direction, support and motivation throughout this work. I would also like to thank Dr. Dong S. Ha and Dr. Shukla for graciously serving on my thesis committee. Last, but not the least by any means, I would like to thank my friends who have made my stay at graduate school enjoyable and the people in my research group who have made every aspect of research exciting and interesting for me.

CONTENTS

CHAPTER 1 .... **1**

INTRODUCTION **1**

1.1 PREVIOUS WORK **2**

1.2 THESIS OUTLINE **4**

CHAPTER 2 .... **5**

PRELIMINARIES **5**

2.1 DELAY FAULT MODEL **5**

2.1.1 STUCK-AT FAULTS *5*

2.1.2 TRANSITION FAULTS *6*

2.1.3 PATH DELAY FAULTS *7*

2.2 METHODS TO TEST DELAY FAULTS **9**

2.2.1 ENHANCED SCAN *9*

2.2.2 BROADSIDE OR LAUNCH-FROM-CAPTURE *9*

2.2.3 SKEW LOAD OR LAUNCH-FROM-SHIFT *10*

2.3 GENETIC ALGORITHMS (GA'S) **12**

2.4 DELAY TEST SET SIZE **14**

CHAPTER 3 .... **15**

HIGH QUALITY DELAY TESTS USING PATH FILTERING **15**

3.1 DELAY TEST METHOD **17**

3.1.1 ROBUST TEST GENERATION PHASE *18*

3.1.2 NON-ROBUST TEST GENERATION PHASE *19*

3.1.3 TRANSITION FAULT ATPG *20*

3.2 CLUSTERING OF PATHS TO REDUCE TEST SET SIZE 22

3.3 RESULTS 24

CHAPTER 4 31

ALAPTF: A NEW TRANSITION FAULT MODEL AND THE ATPG ALGORITHM 31

4.1 ALAPTF: THE NEW FAULT MODEL 32

4.2 ATPG FOR ALAPTF MODEL 36

4.2.1 PRE-PROCESSING 36

4.2.2 ALAPTF ATPG 37

4.3 EXPERIMENTAL RESULTS 39

CHAPTER 5 44

CONCLUSION AND FUTURE WORK 44

REFERENCE 46

LIST OF FIGURES

FIGURE 1 EXAMPLE CIRCUIT FOR TRANSITION FAULTS 7
FIGURE 2 ROBUST AND NON-ROBUST PDFS 9
FIGURE 3 ILA REPRESENTATION FOR BROADSIDE CAPTURE 10
FIGURE 4 WAVEFORM FOR SCAN-ENABLE AND CLOCK FOR THE BROAD SIDE METHOD 10
FIGURE 5 ILA REPRESENTATION FOR SKEW LOAD 11
FIGURE 6 WAVEFORM FOR SCAN-ENABLE AND CLOCK FOR THE SKEW LOAD METHOD 11
FIGURE 7 UNIFORM CROSSOVER 12
FIGURE 8 FLOWCHART FOR A BASIC GA BASED STUCK-AT FAULT ATPG 13
FIGURE 9 A CIRCUIT MODEL WITH 2 PATHS 16
FIGURE 10 RELATIONSHIP BETWEEN ROBUST AND NON-ROBUST PATHS 16
FIGURE 11 ALGORITHM FOR ROBUST ATPG 19
FIGURE 12 A SAMPLE CIRCUIT FOR CLUSTERING 22
FIGURE 13 CLUSTERING HELPS IN NR PATH FILTERING 23
FIGURE 14 PERCENT REDUCTION IN # OF VECTORS USING CLUSTERING 25
FIGURE 15 PERCENTAGE REDUCTION IN THE # OF PATHS USING FILTERING 29
FIGURE 16 IDEAL PLOT FOR GATE COVERAGE WITH VARYING VALUES OF DELAY 32
FIGURE 17 A TRANSITION FAULT NEED NOT PROPAGATE THROUGH A ROBUST/NR PATH 33
FIGURE 18 ALAPTF MODEL 34
FIGURE 19 SPECIAL CASE FOR ALAPTF 35
FIGURE 20 PRE-PROCESSING STEP FOR ALAPTF ATPG 36
FIGURE 21 FLOWCHART FOR FINDING V2 FOR ALL PATHS STARTING FROM A GIVEN PI 38
FIGURE 22 PERCENTAGE OF GATES FOR WHICH TRANSITION REACHES LATER BY USING TNEW COMPARED WITH TOLD 41
FIGURE 23 PERCENTAGE OF GATES FOR WHICH TRANSITION REACHES LATER BY USING TNEW COMPARED WITH TPATH 42
FIGURE 24 PERCENT GATE COVERAGE WITH VARYING AMOUNTS OF DELAYS 43

LIST OF TABLES

TABLE 1 UNTESTABLE ROBUST AND NON-ROBUST PATHS 24

TABLE 2 ATPG RESULTS FOR ROBUST PATHS 26

TABLE 3A. ATPG RESULTS FOR NR PATHS WITH AND WITHOUT FILTERING OF NR PATHS.. 27

TABLE 3B. ATPG RESULTS FOR NR PATHS WITH AND WITHOUT FILTERING OF NR PATHS 27

TABLE 3C. ATPG RESULTS FOR NR PATHS WITH AND WITHOUT FILTERING OF NR PATHS 28

TABLE 3D. ATPG RESULTS FOR NR PATHS WITH AND WITHOUT FILTERING OF NR PATHS 28

TABLE 4 TFC AND THE TOTAL TEST SET SIZE 30

TABLE 5 RESULTS OF THE PROPOSED ATPG ALGORITHM 40

TABLE 6 NUMBER OF FAULTS DETECTED BY VARYING AMOUNTS OF ADDED DELAY 42

CHAPTER 1

INTRODUCTION

Increasing performance requirements motivated testing for the correct temporal behavior, commonly known as delay testing [1]. Delay faults can be modeled in a number of different ways, among which the most common are the Path Delay Fault (PDF) model [2] and the transition fault model [3-4]. Test patterns for transition faults and PDFs consist of a pair of vectors $\{V_1, V_2\}$ where V_1 is required to initialize the target node and V_2 is required to launch the appropriate transition at the target node and propagate it to an observation point, such as a primary output (PO). In PDF, cumulative effect of gate delays along the path is considered whereas in the transition fault, every transition (both $1 \rightarrow 0$ and $0 \rightarrow 1$) can be modeled as 2 stuck-at faults. A transition fault models excessive delay on a single node in the circuit. The test that delivers a rising (falling) transition to a node and sensitizes a path from that node to an observation point will detect a slow-to-rise (slow-to-fall) transition fault at that node. That same test may detect a path delay fault associated with the particular route into and out of the node in question. Conversely, a PDF test may also detect some transition faults. Nevertheless, a complete transition test set may not detect all critical paths; likewise, a test set that exercises longest paths may not detect all transition faults. In general, the transition fault model is for capturing gross defects whereas PDF model is for detecting small defects. Thus to achieve high delay defect coverage we require both high path and transition fault coverage.

In the thesis, both the aspects of delay tests, i.e. PDF's and Transition faults has been studied and new algorithms have been proposed for increasing the quality of tests. We know that typically a circuit can have exponential number of paths. Obtaining high PDF coverage may require testing of a large number of paths, many of which overlap with one another. To improve the delay test sets, we make an attempt to generate tests such that they not only have high robust path coverage for the critical paths, but the test set is also capable of detecting other delay fault models (such as transition faults) that were missed by the critical path analysis. Since the number of paths can be very large for practical circuits, we try to generate tests for a filtered path set. The idea behind the work is to reduce the number of Non-Roust (NR) paths to be considered for test generation without losing PDF coverage. Hence, instead of selecting NR paths based on their lengths, we discard all NR paths that significantly overlap with previously tested robust paths

The second aspect of delay test relates to transition faults. We have proposed a new transition fault model called **A**s **L**ate **A**s **P**ossible (ALAP) Transition Fault Model. We know that the traditional transition fault model detects gross delays on the circuit nodes. If robust tests are possible for all the paths in the circuit, we will not need any additional delay tests. However, since very few paths are robustly testable, there are some delays, which cannot be captured by both transition and path delay fault models. Consider the condition when there are some small delay defects distributed inside a circuit. If the nodes lie on a robustly untestable path or a less critical path, then the path delay model may miss on those faults. The segment delay fault model might also miss the fault because there might not be a path along which the effect may be propagated. Furthermore, launching of a transition may not be either robust or non-robust. Hence, we want a model that can accumulate the effect of these small delays on a particular node, which can then be tested by the traditional transition fault model. This accumulation of small delays can be modeled by launching each transition fault as late as possible at the fault site. The notion of ALAP can be implemented by making sure that a transition fault is launched via one of the longest robust segment ending at the fault site.

1.1 PREVIOUS WORK

A large amount of work has been done in the past relating to the detection of both the Transition Faults and the Path Delay Faults (PDFs). Due to the potential large number of paths in a circuit, selection of paths and the ways to test them has always been important. A large amount of work has also been done in the area of classification of different path delay tests. PDFs were broadly classified in two ways [5]: Robust PDF and Non-Robust (NR) PDF. A more detailed classification was presented in RESIST [6] where the paths were classified in five basic categories, i.e. GRT (General Robust Tests), GNRT (General Non-Robust Tests), HFRT (Hazard Free Robust Tests), RRT (Restricted Robust tests) and RNRT (Restricted Non-Robust Tests). The disadvantage of using such a detailed classification is that we need to have a very elaborate value system. The complexity of the ATPG engine also increases with this classification. Both DYNAMITE [7] and RESIST are deterministic based ATPG engines. DYNAMITE applies improved redundancy identification techniques. All the paths are stored in a path tree. A stepwise path sensitization procedure identifies sets of redundant path delay faults without enumerating them. The limitation of the method is that the path tree is impractical for large circuits. To improve upon this limitation, the authors in RESIST proposed a recursive method to identify redundant paths and for the test generation. No path tree is required. Although the method is effective in detecting a large number of paths, no path selection has been used. Hence circuits with large number of paths become difficult to test.

Another famous ATPG engine is called FSIMGEO [8]. It is a simulation based ATPG engine for PDFs, but this misses the delay faults on the less critical paths. To overcome this, segment delay faults were considered and studied in [9]. In this the authors study the technique of covering delay defects on untestable critical paths by robustly testing their longest possible segments that are not covered by any of the testable critical path. The disadvantage of this scheme is that there are a large number of untestable critical paths and generating NR tests for all can be futile.

A different approach for the selection of critical paths has been presented in [10-11]. In [11] the authors try to generate a longest path passing through each gate. To achieve this they have presented a graph traversal algorithm that takes a weighted Directed Acyclic Graph (DAG) *G*, a graph vertex *v* and a path length *l* as input and traverses all those paths in *G* that pass through *v* and have a length *l*. There are some search pruning methods also involved in the algorithm. Some of the difficulties in using the method are as follows:

1) The method is not useful of the circuits having a large number of long untestable paths. Since the algorithm checks of all the possible long paths passing through each gate, the techniques becomes expensive for circuits like c6288.
2) The sets LRF (LRB) contain a number *w* which represents that there is segment of length *w* starting from the node (some input) and ending at some output (the node). But since these numbers are purely based on structural length or delay information, the longest path length found by these sets for each gate can be impractical. In other words, those long paths may not be testable and the algorithm wastes a large amount of time for theses paths.
3) Since the longest path passing through a gate may actually be one of the shortest paths in the whole circuit, this technique does not guarantee a proper coverage of the critical paths if only these paths are considered.

Another way to select specific paths to be tested is presented in [12-13]. This is a statistical based approach where critical paths are selected based on the statistical properties of the already detected paths. The problem with this method is that since the path selection is based on the statistics of the process variation, we need to have a different path set for each individual chip which is impractical.

1.2 THESIS OUTLINE

An Outline of the rest of the thesis is as follows:

- Chapter 2 outlines the basic definitions and terminology used. It gives the details about the different types of delay tests, the different ways to apply delay tests and the basics of simulation based ATPG algorithms. Some vector storage method schemes have also been defined in Section 2.4.
- Chapter 3 describes the generation of high quality delay tests using the appropriate selection of paths. A new incremental based algorithm for transition faults has also been presented. Results are presented in Section 3.3.
- Chapter 4 presents the idea of As Late As Possible transition fault model to detect small delay defects using transition fault model. This method is a marriage of path delay faults and transition faults. The method is effective in testing small delays. The results for this are presented in Section 4.3.
- Chapter 5 concludes the work with an overview and presents some recommendations for future work.

CHAPTER 2

PRELIMINARIES

This Chapter presents the basics of delay faults. The differences between the Robust and Non-Robust Path Delay faults as well as Transition faults have been described. Different approaches used for the application of tests for Transition Faults have been presented. Essentials of genetic algorithm along with different types of fitness functions are presented in Section 2.3.

2.1 DELAY FAULT MODEL

Physical failures and fabrication defects cannot be easily modeled mathematically. As a result, these failures and defects are modeled as logical faults. All the different types of delay fault models can be modeled in terms of stuck-at faults. Hence let us just look into the basic definition of single stuck-at faults.

2.1.1 STUCK-AT FAULTS

It has been shown that stuck-at fault tests are effective in capturing a wide range of defects on manufactured chips. This model represents faults caused by opens, shorts with power or ground, and internal faults in the components driving signals that keep them stuck-at a logic value (1/0). To test for stuck-at faults, two steps are involved. The first step is to generate a test vector that excites the fault and the next step is to propagate the faulty effect to a primary output or a scan flip-flop. Automatic test pattern generation (ATPG) tools are typically used to generate the test vectors. It is relatively easy to generate patterns for stuck-at faults and pattern volume is also comparatively low.

Now let us consider basic types of delay fault models. As described previously, delay faults can be broadly classified into Transition and Path Delay Faults.

2.1.2 TRANSITION FAULTS

The Transition Fault model is similar to the stuck-at fault model in many respects. The effect of a transition fault at any point P in a circuit is that a rising or a falling transition at P will not reach an observable output such as a scan flip-flop or a primary output within the desired time. Because of the nature of these faults, a node can have a slow-to-rise or slow-to-fall transition fault. A slow-to-rise fault at a node means that any transition from 0 to 1 on the node does not produce the correct result when the device is operating at its maximum operating frequency. Similarly, a slow-to-fall fault means that a transition from 1 to 0 on a node does not produce the correct result at the desired frequency.

In any circuit, *slack* of a path can be defined as the difference between the clock period when the circuit outputs are latched and the propagation delay of the path under consideration. For a gate level delay fault to cause an incorrect value to be latched at a circuit output, the size of the delay fault must be such that it exceeds the slack of at least one path from the site of the fault to the site of an output pin or scan flip-flop. If the propagation delays of all paths passing through the fault site exceed the clock period, such a fault is referred to as a *gross delay fault* [4].

Any test pattern that successfully detects a transition fault comprises of a pair of vectors $\{V_1, V_2\}$, where V_1 is the initial vector that sets a target node to the initial value, and V_2 is the subsequent vector that not only launches the transition at the corresponding node, but also propagates the effect of the transition to a primary output or a scan flip-flop [14]. Let us consider an example circuit to understand the exact definition of a transition fault. Consider a small circuit shown in Figure 1. It has 2 NAND gates and one Primary Input and 2 Observation points. By the application of the shown vector pair, we can see that there is a rising transition at nodes *x* and *y*. These transitions are propagated to the PO's and they manifest themselves as 2 falling transitions detecting a total of 4 transition faults *viz. rising TR faults at x and y* and *falling TR faults at w and z.*

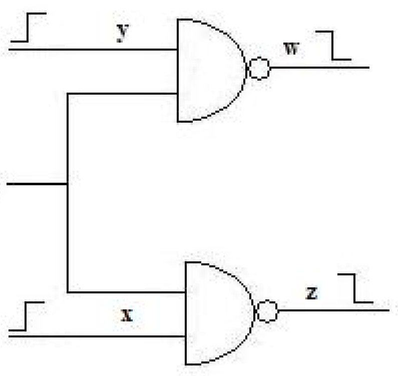


Figure 1 Example circuit for transition faults

2.1.3 PATH DELAY FAULTS

A physical path *P* is an interconnection of gates from an input (PI or a scan flop) to observation point (PO or a scan flop). A rising (falling) path P^r *(P^f)* is defined as the path corresponding to a rising (falling) transition starting at the PI. The polarity of the transition for each gate on the path depends on the inversion parity along that path. *Path Length* is defined, as the number of gates in a given path *P*. *Segment S* is a contiguous section of *P*. A segment can start and end at any point in the given path *P*.

Paths delay fault model is used to detect small distributed delays along a path. Faults in a circuit due to process variations can manifest themselves as small delays which individually do not make the circuit faulty. Since the extra delay at each gate is small, transition fault model is incapable of detecting such faults. As evident from the name, PDFs are used to detect error on the specified paths. Since number of paths is usually large, selection of paths is critical in generation of PDF tests. PDFs are more complex to model than transition faults but they can be used to test the defects in the critical paths.

There are a large amount of variations of the path delay fault model. They can be broadly classified into two categories:

A) Non-Robust (NR) Path Delay Fault:

This class of paths is *statically sensitizable* [5]. A path P is said to be statically sensitizable if there exist at least one input vector which stabilizes all side inputs of P at non-controlling value (NVC). However, NR tests cannot guarantee the detection of the fault in the presence of other faults. Without making any assumption on the component delay values, guaranteed tests exist for only a subset of these non-robustly testable PDFs, and are called *validatable non-robust tests* [5]. A more formal definition of NR tests is as follows: A vector pair {V_1, V_2} is said to detect a path P non-robustly, iff:

i) If the vector pair launches a transition (rising/falling) at the beginning of the path **AND**
ii) All the off-paths inputs along the paths have a NCV for V_2.

NR tests are easy to generate as compared to other forms of PDFs. Vector V_1 only launches the transition at the start of the path which can always be controlled since it is either a PI or a scan flop. The only disadvantage of NR tests is that they can be invalidated in the presence of other faults.

B) Robust Path Delay Faults:

These types of paths can be tested independently of side path delays. Hence if all the paths in a circuit are robustly testable then we will not require any other kind of delay tests. They are a more constraint form of NR delay tests and cannot be invalidated by other delays. Numerous classifications of robust path delay faults exits [6] e.g., hazard free robust tests, single/multiple input changing tests, and single/multiple path propagating tests. The essential requirements of a path P to be robustly testable are:

i) The vector pair {V_1, V_2} should be a NR test for P **AND**
ii) Whenever the on-path input of a gate G along the path transitions from a NCV to a controlling value (CV), then all the off-path inputs of G should be held at a steady NCV.

Robust tests are difficult to generate and a large number of paths in a circuit are usually robustly untestable (Chapter 3).

Let us consider an example to understand the definitions of NR and Robust test. Consider the circuit in Figure 2. The rising path P_1^r= PI-1-2-3-4-5-PO has a path length LP_1 = 7. P_1 is robustly and non-robustly testable as shown in the figure. Now consider another path P_2^r = PI-1-2-3-6-5-PO which also has LP_2 = 7. To robustly test P_2, the off-path input of gate 6 must be a steady one, which is not possible in this case. Hence P_2 is robustly untestable but it is non-robustly testable.

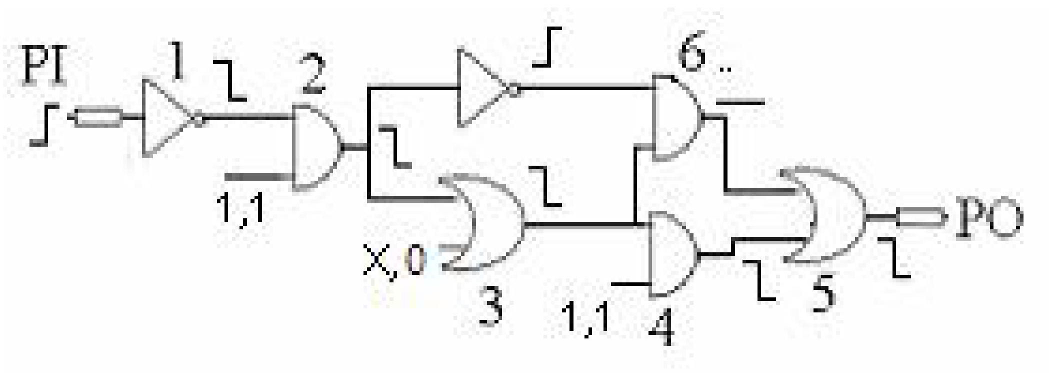


Figure 2 Robust and Non-Robust PDFs

2.2 METHODS TO TEST DELAY FAULTS

As mentioned before, delay tests require a vector pair to detect a fault. Since the patterns must be applied at the rated speed, at-speed testing is needed. For full scan circuits, both the vectors in the scan flip-flops must be ready for consecutive time frames to ensure at-speed testing. Several different methods are used to apply the vectors at-speed. The three most common ones are as follows:

2.2.1 ENHANCED SCAN

This the type of method in which both the vectors are shifted in during the shift process to the scan flops. Special scan flops are required for this method so that they cab store 2 values at a time. After both the vectors are shifted in, scan-enables go low and the clock is pulsed two times. The response is then shifted out. The advantage of this method is that we can achieve higher coverage since both the vectors are controllable, but the technique has a high overhead of special scan flops called hold-scan flops [15].

2.2.2 BROADSIDE OR LAUNCH-FROM-CAPTURE

Broadside [3] is the most common form of delay fault application method. The method requires only one vector to be shifted in during the shift cycle. Vector V_2 is the circuit response obtained by the application of V_1. Since the second vector is the functional response of the first vector, this method is also known as *functional justification method*.

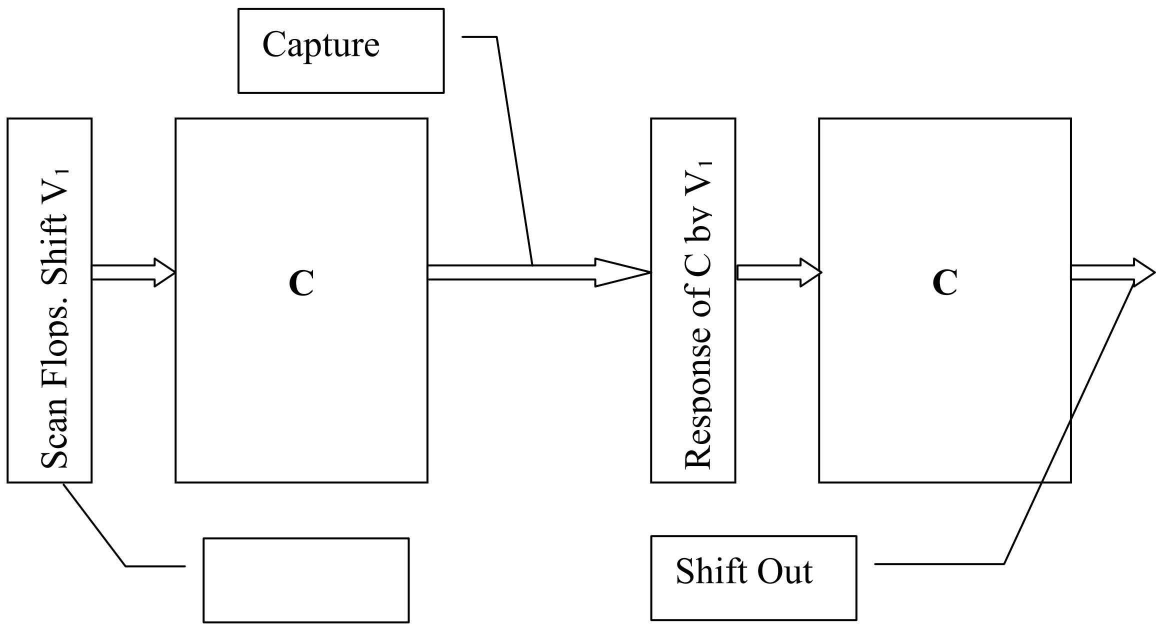


Figure 3 ILA representation for Broadside Capture

Figure 3 shows an ILA representation of a circuit C. During the broadside operation, we shift-in the first vector, and then the system clock is pulsed twice to make a launch and a capture which is then shifted out during the shift-out operation. The waveforms for the whole operation can be plotted as shown in Figure 4.

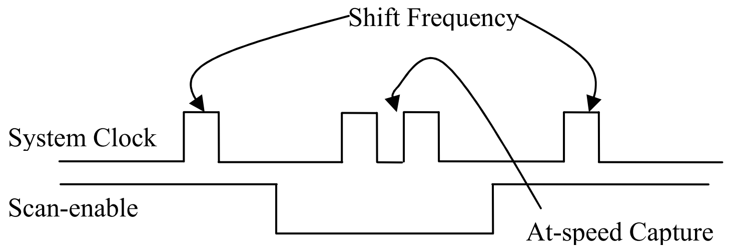


Figure 4 Waveform for scan-enable and Clock for the Broad side method

2.2.3 SKEW LOAD OR LAUNCH-FROM-SHIFT

As the name specifies, this methods uses a shifted version of the first vector as its second vector. Hence the second vector is no more the functional response of the circuit. Instead after shifting in the first vector, we pulse the system clock and make the scan-enable go low. After the first clock, scan-enable is again made to go to high and then comes the capture clock.

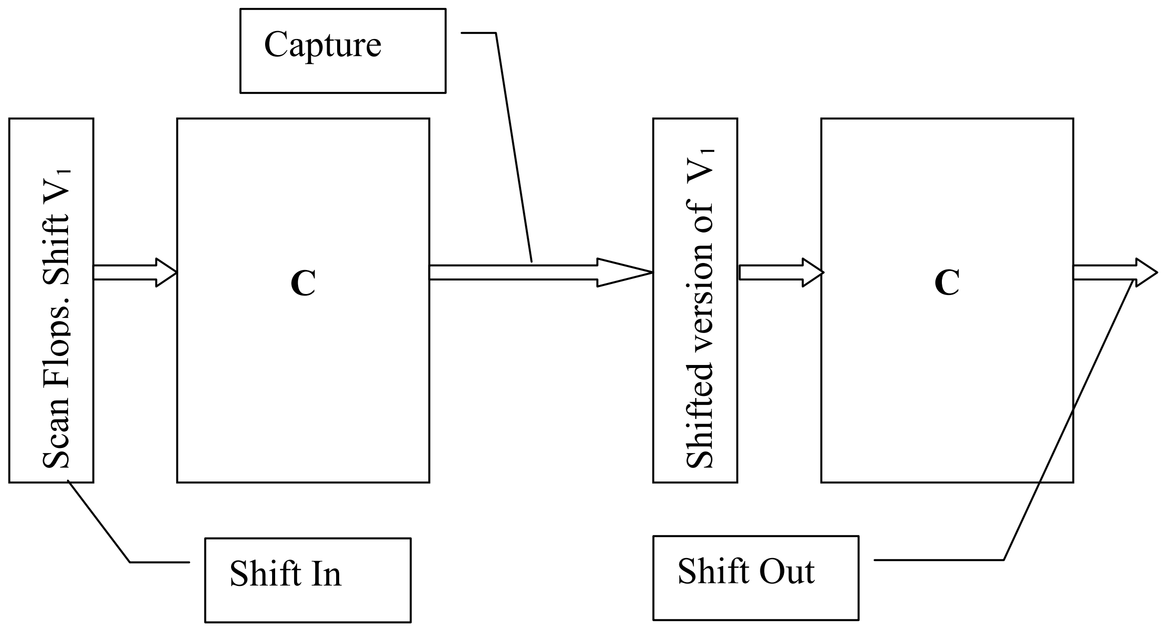


Figure 5 ILA representation for Skew Load

Hence the second vector is the one bit sifted version of the first. The advantage of this method is that it produces better coverage as compared to broadside since the second vector is partially controllable. But the disadvantage is that the scan-enable to switch state exactly between the 2 at-speed system clocks. This is practically difficult because of the clock skew problem; it's difficult to make a switch right in between of two system clocks. The ILA representation and the weave forms for the clock and the scan-enable are shown in Figure 5 and Figure 6 respectively.

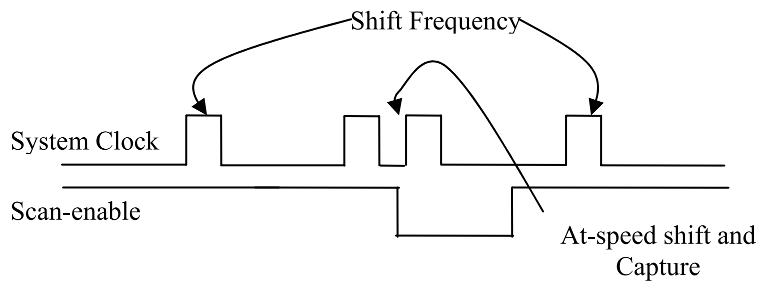


Figure 6 Waveform for scan-enable and Clock for the Skew Load method

2.3 GENETIC ALGORITHMS (GA'S)

GA's [16-18] revolve round the same framework as nature has supplied us. There is very strong parallelism between the genetic evolution in nature and that in this algorithm. Here there is an initial population and three basic operators i.e. selection, crossover, and mutation. Each individual is a vector and the best one is evolved over generations from selection and crossover. The initial population is selected randomly and then the selection and crossover is done in order to improve the performance of an individual. The law of the "survival of the fittest" applies here also. Weak vectors are dropped and not used for crossover. The selection of a fitness function is very critical in this algorithm. A bad fitness function can lead to an individual, which does not give good fault coverage whereas a good fitness function can avoid this. To find an individual with the desired properties sometimes become difficult and there can be several generations evolved before an individual is obtained. The three basic string operations do not require a lot of execution time but the calculation of the fitness of an individual is very time consuming.

Several, selection methods can be applied to select an individual. In the GA always two parents give rise to two children and hence the size of the population is maintained constant. In *tournament selection*, two individuals from the population are selected in random and there fitness function is compared. The best among the two becomes parent one (P1) and similar process is applied to find the parent two (P2). The two are then crossed over to get two children (C1 and C2). In *Uniform cross over*, a mask is taken which is chosen randomly and depending on the mask, crossover is done. For e.g. as shown in Figure 7 the bits of two parents are swapped if the mask has a one, otherwise it its left as it i

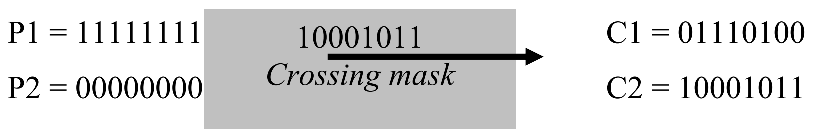
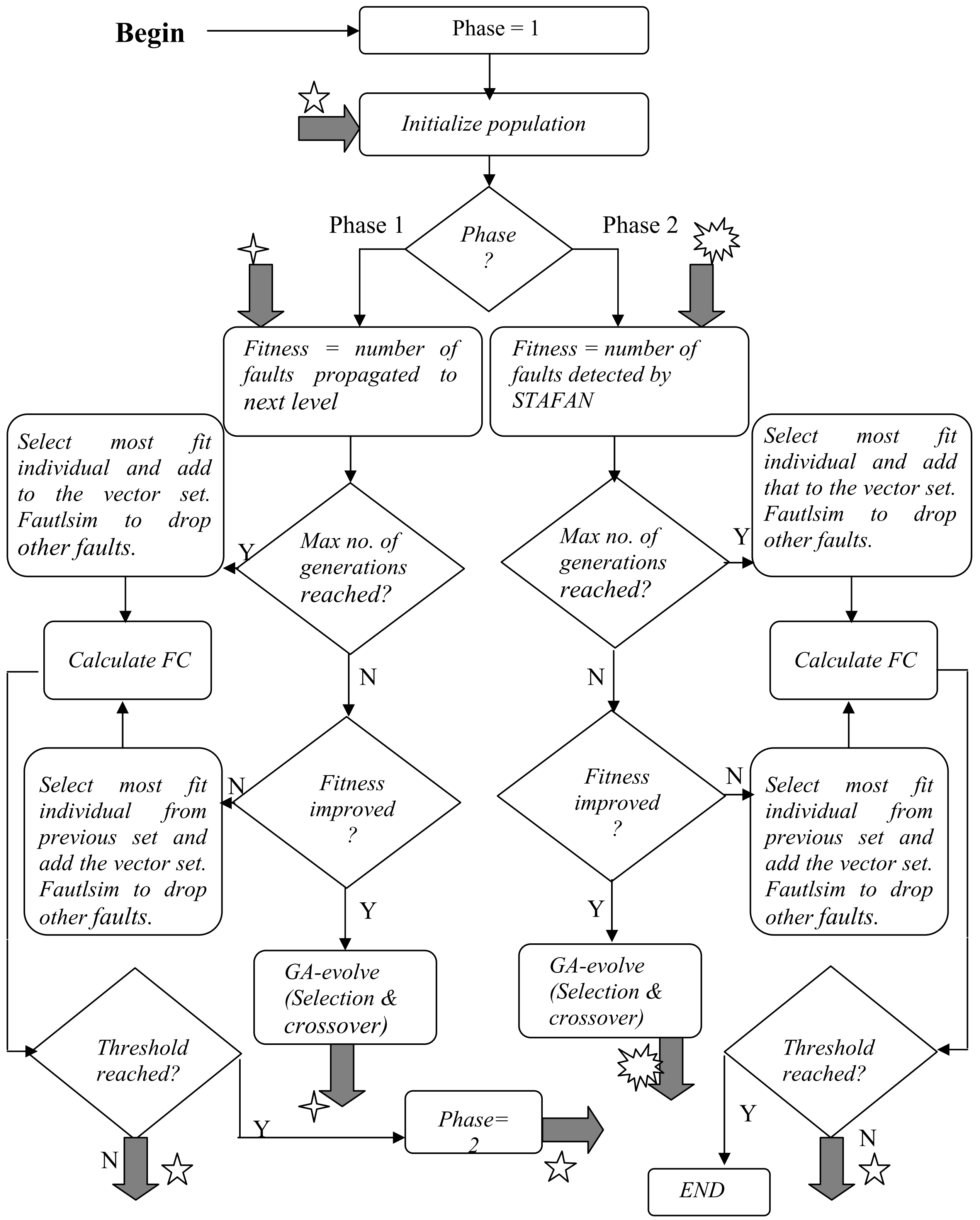


Figure 7 Uniform Crossover

A basic GA based stuck-at ATPG flowchart is shown in Figure 8. It's a two phase algorithm and hence it has two different fitness functions and the program switches its fitness function after a threshold is reached. We have used GA's to develop a Transition Fault ATPG engine. A simulation method was also used to generate PDFs which is discussed in Chapter 3.



Begin
Phase = 1
Initialize population
Phase ?
Phase 1
Phase 2
Fitness = number of faults propagated to next level
Fitness = number of faults detected by STAFAN
Select most fit individual and add to the vector set. Fautlsim to drop other faults.
Select most fit individual and add that to the vector set. Fautlsim to drop other faults.
Max no. of generations reached?
Max no. of generations reached?
Y
Y
N
N
Calculate FC
Calculate FC
Fitness improved ?
Fitness improved ?
N
N
Select most fit individual from previous set and add the vector set. Fautlsim to drop other faults.
Select most fit individual from previous set and add the vector set. Fautlsim to drop other faults.
Y
Y
GA-evolve (Selection & crossover)
GA-evolve (Selection & crossover)
Threshold reached?
Threshold reached?
Y
Phase= 2
N
Y
N
END

Figure 8 Flowchart for a basic GA based stuck-at fault ATPG

2.4 DELAY TEST SET SIZE

As mentioned earlier, a scan based delay test needs two vectors for testing a given transition. Hence, any given test set *v* having *n* test patterns can be represented as:

V={(v11,v12),(v21,v22),……(vi1,vi2),……(vn1,vn2)}.

Vector Reusable Test Set (VRTS) [19] is a special form of vector storage in which a test set *T* having *m* elements can be represented as:

T={(v11,v12),(v12,v22),……(v(i-1)2,vi2), …… (v(m-1)2,0vm2)}

where $m<n$. Thus, instead of storing *2n* vectors for test set *v*, we only need to store *m* vectors. Since this method reuses the vector space, it is called VRTS.

CHAPTER 3

HIGH QUALITY DELAY TESTS USING PATH FILTERING

Since the number of paths can be very large for practical circuits, only some selected paths are considered for test generation. The general criterion for the selection of paths is based on there lengths (structural or delay based). This way some of the critical paths can be tested and hence selection of paths is very important for the process of test generation. The idea behind this work [20-21] is to reduce the number of Non-Robust (NR) paths to be considered for test generation without losing Path Delay Fault (PDF) coverage. Hence, instead of selecting NR paths based on their lengths, we discard all NR paths that significantly overlap with previously tested robust paths. This concept can be understood by considering Figure 9. It shows a circuit model with two paths originating from two PIs and ending at two different POs. Let path P_1 (PI1-PO1) be longer than P_2 (PI2-PO2) and let us assume that P_1 is robustly testable whereas P_2 is robustly untestable. The overlap of P_1 and P_2 is L_{over}. We know that since we can test P_1 robustly, the region of overlap is also tested robustly. If L_{over} is greater than some preset threshold, then the delay due to the non-overlapping portion of P_2 alone will not likely make P_2 faulty (if the defects on the non-overlapping segment are small distributed delay defects). Hence, the likely fault that can make P_2 faulty is a large delay present in the non-overlapping section. By making sure that the test set covers the transition faults associated with these gates, we can discard many paths like P_2, reducing the total number of paths needed to be considered for test generation. Due to this observation, a high-quality delay test set also should achieve a high Transition Fault Coverage (TFC).

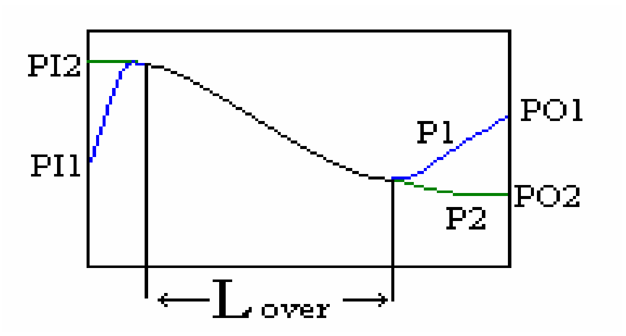


Figure 9 A circuit model with 2 paths

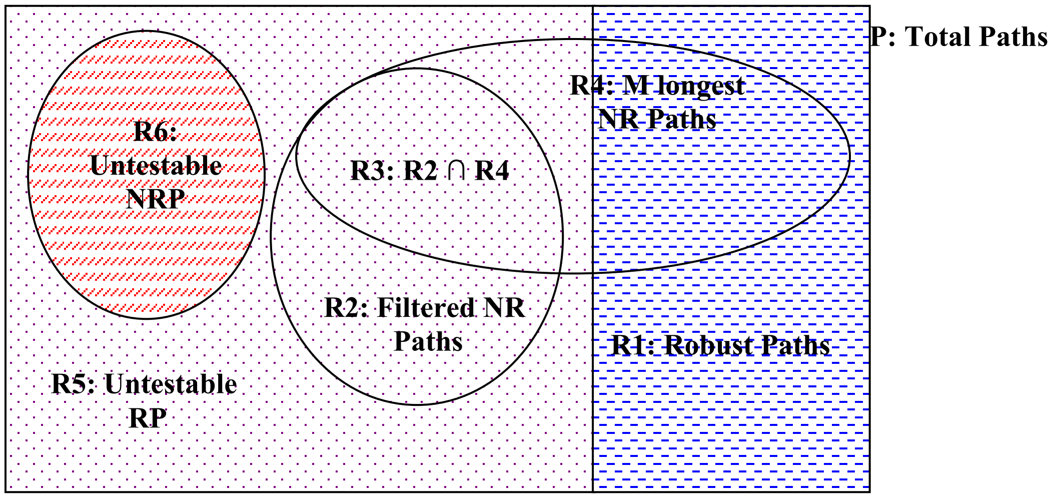


Figure 10 Relationship between robust and non-robust paths

A general relationship between paths can be deduced from the Venn diagram of Figure 10. Out of the total possible P paths in the circuit, region R1 (right rectangle) represents robust paths in the circuit, while Region R5 (left rectangle) represents the robustly untestable paths. R6 represents the untestable NR paths. R6 is a subset of R5 because untestable NR paths are also robustly untestable. Region R4 represents the M longest paths considered for ATPG. Note that this set contains some robust and some non-robust paths. Nevertheless, not all paths from R4 need to be tested, since many of them overlap with already tested robust paths, as explained earlier. Using our proposed filtering technique, we choose paths more intelligently. Let us suppose that the region R2 contains the set of NR paths that do not overlap with the tested robust paths. Then, the region R3 (overlapping between R2 and R4) contain paths which are both long and do not overlap with an already tested robust path. Thus, while selecting NR paths for test generation, we want to select paths from R3, rather than all of NR paths in R4. If the test set can accommodate more patterns, we can choose additional paths from R2. Results show that there has been a reduction in the NR path set by as much as 40% for some circuits.

We have also proposed clustering of paths to reduce the test set size by considering multiple compatible paths together for test generation. Results showed that clustering of paths reduces test set data by about 40%. Untestable paths are dropped in the initialization phase and reusable vector storage schemes [19] have been used to further reduce the test set size.

3.1 DELAY TEST METHOD

We want to select a small number of paths for ATPG and still want to have high quality test set. A robustly tested path detects small-distributed delays along the path, and hence NR path that overlap significantly with this path may become futile to test. By considering the transition fault model along with the PDF model, we can compute a measure for selecting NR paths for ATPG. The two fault models can be related to each other by the following observation.

Observation 1: By making sure that a test set has high transition fault coverage, many of the NR paths that overlap largely with already tested robust paths need not be tested.

We will explain the observation with the following example: Using Figure 9, consider the case when a large portion of a NR path (e.g. P_2) overlaps significantly with an already robustly tested path (e.g. P_1). One of the following two scenarios can occur:

1) *The non-overlapping portion of P_2 has a small delay (according to distributed delay model)*: Since $LP_1 \geq LP_2$, and P_1 is already tested robustly, this small delay alone is not likely to make P_2 faulty. In other words, since a large portion of P_2 overlaps with P_1, this added delay is most likely to be detected via the test of path P_1. Hence we don't need to consider P_2 separately for ATPG, similar to those less critical or shorter paths that we do not consider (region P-R1-R2-R4-R6 of Figure 10).

2) *The non-overlapping portion of P_2 has a large (lumped) delay*: This large delay can always be tested by using the transition fault model at the nodes of the non-overlapping portion of P_2.

Thus, in order to have high quality delay test, we need to have high robust path coverage, high NR path coverage and high transition fault coverage. To achieve this efficiently, we have designed a 3-phase ATPG. All 3 phases are described in the following sub-sections.

3.1.1 ROBUST TEST GENERATION PHASE

The first step towards the test generation for robust paths is to enumerate the robustly testable paths for which tests are to be generated. We use an implication base technique much similar to [22] for the removal of all the untestable robust paths. This implication-based technique can be best understood by a simple example. Consider the circuit of Figure 2 which is again shown here for convenience. To robustly test the path Pr = PI-1-2-3-6-5-PO, there is a transition from a non-controlling (NCV) to a controlling value (CV) at the input of gate 6. Hence, the off-path inputs of gate 6 should have a steady NCV for both V1 and V2, which imply a constant ‘0’ at the output of gate 2. This is a conflict and hence Pr is robustly untestable. This implication-based analysis identifies a large number of untestable robust paths for most of the circuits.

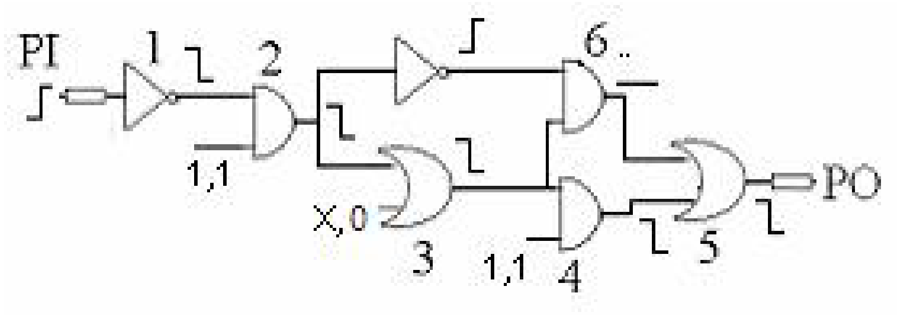


Figure 2 Robust and Non-Robust PDFs (Redrawn from Chapter 2)

After removing paths that are robustly untestable, we want to generate tests for the N longest paths. The algorithm used for doing this is shown in Figure 11. The function essential_values analyzes a given path P and finds the values needed by V1 and V2 on all the gates of P and stores them in vectors val0 and val1, respectively. It also finds the essential off-path values under V2. For example, in Fig. 1 for path Pr = PI-1-2-3-4-5-PO values in val0 = gate1=1, gate2=1, gate3=1, gate4=1, gate5=1, and val1 = gate1=0, gate2=0, gate3=0, gate4=0, gate5=0. val1 also contains nodes corresponding to side input of gate2 to be logic 1 and the side input of gate4 to be logic 1 as well. Since the function needs to satisfy the values in val0 and val1, only 3-valued logic simulation is needed. A vector pair is produced for P if Vi and Vi+1 satisfy all the values in val0 and val1 corresponding to P respectively. All other paths detected by <Vi-1,Vi> and <Vi,Vi+1> are then dropped. The final test set produced after considering all N paths is called T_R.

```
robust_ATPG(){
  For all paths P not detected {
    essential_values(P,val0,val1);
    Generate vector $V_i$ (values in val0 need to be satisfied)          //only need to do logic simulation
   If $V_i$ generated {
    Generate vector $V_{i+1}$ (values in val1 needs to be satisfied)      //only need to do logic simulation
      If $V_{i+1}$ generated {
       Add $V_i$ and $V_{i+1}$ to the test set T
      Drop all path detected by vector pairs <$V_{i-1}$,$V_i$> and <$V_i$,$V_{i+1}$> }
  }}
```

Figure 11 Algorithm for Robust ATPG

3.1.2 NON-ROBUST TEST GENERATION PHASE

An implication-based approach similar to that used to enumerate robust paths is used to first drop all the paths that cannot be tested non-robustly. However, this implication based technique poses restrictions on the values required by V2 only. After dropping the identified untestable NR paths, we further remove additional NR paths that satisfy the following two conditions according to *observation 1*:

1) The NR path PNR overlaps with an already detected robust path PR with an amount greater than a preset threshold Δ_{NR}. The overlapping section should be contiguous.

2) L(PR) $\geq$ L(PNR).

After dropping paths based on above criteria, we can drop a large number of NR paths. But the number of paths dropped depends on the number of robust paths detected. Higher robust path coverage generally translates to more NR paths dropped. We also drop additional paths that are incidentally detected by the robust test set TR generated in Section 3.1.1

Once we have the set of filtered NR paths, we generate test for the longest M paths (if the number of paths is still large). The algorithm for NR path ATPG is similar to that of robust path ATPG used in Section 3.1.1, except that NR condition is enforced. Hence for V1, the ATPG needs to satisfy only the conditions at the PI. The final test set produced after the end of this function is called T_{R+NR}.

3.1.3 TRANSITION FAULT ATPG

The test set produced so far may not have high transition fault coverage (TFC) since we did not target some NR paths (by observation 1) that overlap with an already detected robust path. The dropped NR paths can still cause a delay fault if a large delay defect is present on the nodes of the path that can be captured by using the transition fault model.

Genetic Algorithm (GA) is used for the transition fault ATPG. The advantages of GA over conventional deterministic approach are:

(1) Multiple transition faults can be easily targeted simultaneously, and
(2) A good quality vector set without backtracking can be produced in a reasonably shorter time.

GA has been used before for stuck-at faults [16-18]. Calculation of fitness function through multiple fault simulation is a bottleneck in the efficiency of GA's. We developed an ATPG called ***Incremental Propagation Based ATPG***, which circumvents the problem of fault simulation required for the calculation of fitness function. The algorithm is divided into three phases. Instead of generating tests that will guarantee the detection of some faults, we generate tests incrementally.

All the transition faults detected by the test set T_{R+NR} are dropped initially and the TFC achieved by T_{R+NR} is defined as $TFC_{phase\ 0}$.The three phases are described as follows.

Phase I: In this phase of the ATPG, we generate test patterns that will only launch the targeted transitions. We try to maximize the launch coverage L1 in this phase and add all the vectors produced to the test set. Hence at the end of the first phase we have a set T0={v_1, v_2, v_3……v_N }, where vectors v_1 to v_N are stored in the VRTS fashion. Now a Transition Fault Simulation is performed using this vector set and all the detected faults are dropped. Therefore now we have N vectors, which have launch coverage of L1 and transition fault coverage of $TFC_{phase\ I}$. For most of the circuits L1 is near to 100%. For every transition fault *f* that is launched, we keep track of the vector number in a *vec_num* database, which launched *f*. This information is later used in phase III. The importance of phase I come from the fact that a large number of transition faults are easy to detect and we want to drop all the easy faults as soon as possible so as to save execution time. Moreover the database *vec_num* produced in this phase helps reduce the time to regenerate V1 for faults that are hard to detect in the later phases.

Phase II: In the second phase, we generate VRTS such that the first vector excites as many undetected faults *f* as possible and the corresponding next vector excites as many opposite of *f* as possible and also propagates them to k levels ahead from the fault site; where k is the iteration number within phase II. Note here that the second vector need not propagate the fault to a primary output. Hence after the end of first iteration within phase II, we have another test set T1={v_{N+1}, v_{N+2}, v_{N+3}…v_{N+m}}. Transition fault simulation is again performed on T1 and the detected faults are dropped.

After the end of k iterations (k in worst case can be equal to maximum number of levels in the circuit) we have k test sets. These can be appended together to get the test set T = {(T1, T2, T3,…, …, Tk)}.Thus the TFC of phase II is :

$$\boldsymbol{TFC_{phase\ II} = \sum(TFC(Ti)) + \Phi}$$

$$\boldsymbol{And\ \Phi = TFC(V_{T1}^{\alpha 1}, V_{T2}^{1}) + TFC(V_{T2}^{\alpha 2}, V_{T3}^{1}) + ... + TFC(V_{T(k-1)}^{\alpha(k-1)}, V_{Tk}^{1})}$$

Where αi is the number of vector in test set Ti and the V_{Ti}^{x} represents vector number X of test set Ti. The term Φ accounts for the TFC for the vectors that are at the boundary of the two VRTS, Ti and Ti+1.

Phase III: This phase targets the remaining hard-to-detect transition faults and is a fault dependent phase. Unlike the other two phases it adds a vector pair for each detected fault. In this phase every undetected transition fault is considered separately and test is generated for it. Fitness of an individual is defined as the number of fault events produced. Once a vector pair {V_1, V_2} is generated for a transition fault *f* , we drop all the other undetected faults that might be detected by {V_1, V_2}. From the *vec_num* database generated in phase I, it is easy to find the vectors that launch the transition. We don't have to waste effort in regenerating vector V_1. Hence, if a transition fault *f* has a database entry in *vec_num*, then we only need to generate V_2. The TFC at the end of this phase is given by $TFC_{phase\ III}$.

Hence, after the end of all the three phases the final TFC is:

$$\boldsymbol{TFC = TFC_{phase\ 0} + TFC_{phase\ I} + TFC_{phase\ II} + TFC_{phase\ III}}.$$

And the final test set is called $T_{R+NR+TF}$.

3.2 CLUSTERING OF PATHS TO REDUCE TEST SET SIZE

It follows from Section 3.1 that tests are generated for each path separately and two vectors are added for each path detected. Although additional paths detected by an added vector pair are dropped, using an optimization called *clustering*; we can further reduce the vector space. All the paths are clustered based on their compatibility with each other. Then, instead of considering one path at a time, we consider a whole cluster at a time. Two paths are clustered if none of the values in val0 and val1 of both the paths contradict each other. It is to be noted that two paths need not overlap each other for being compatible. In order to limit the compatible path space, we only combine a path P_i with P_j such that $j \leq i$, where the initial ordering of paths can be arbitrary. In our case the initial ordering of paths was the same as the order in which paths are generated. Moreover, cluster size was limited to 50 for each path due to memory limitations.

Once clustering is done based on path compatibility, we generate test for a whole group. The algorithm targets the first path in the cluster. Once it is detected, we try to fill the remaining don't care values of the produced vector such that another paths in the cluster also gets detected. It is a form of compaction with the exception that vectors are modified dynamically based on the clustered paths. The concept of clustering can be best understood by the following example.

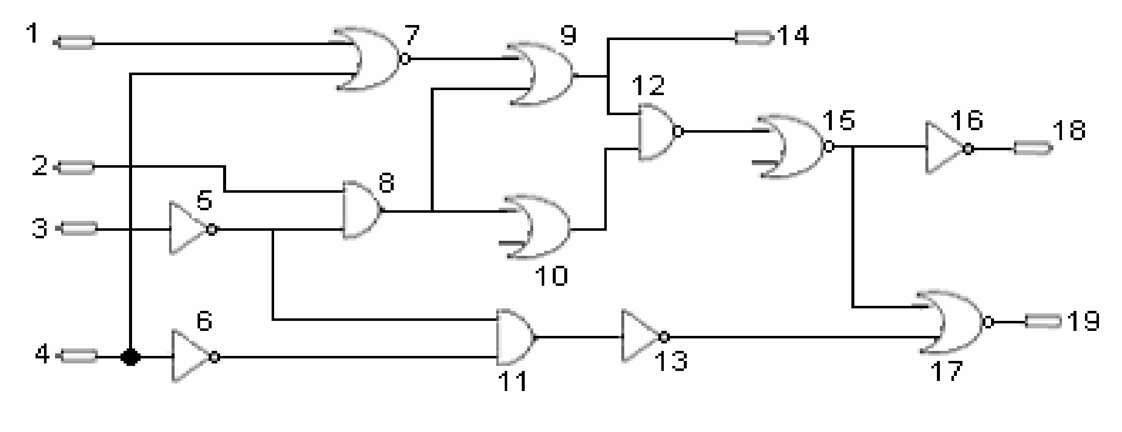


Figure 12 A Sample Circuit for clustering

Consider the circuit of Figure 12. Let paths P_1, P_2, P_3 and P_4 be defined as:

P_1^r = 1-7-9-14
P_2^f = 4-6-11-13-17-19
P_3^f = 2-8-9-12-15-16-18
P_4^f = 2-8-10-12-15-16-18

Without clustering, we will require 8 vectors to detect all the 4 paths. But with clustering the *compatibility relations (C)* are as follows:

P_1***C*** (P_1, P_3); P_2 ***C*** (P_2, P_3, P_4); P_3 ***C*** (P_1, P_2, P_3); P_4 ***C*** (P_2, P_4).
Hence the cluster of paths will be as follows:
Group 1: P_1, P_3; Group 2: P_2, P_3; Group 3: P_3; Group 4: P_4;

Note that group 2 does not contain P_4 since P_3 is not compatible with P_4. Suppose a test {V_1, V_2} is generated for group 1, which detected both path P_1 and P_3. Hence the final test set will be reduced to only 6 vectors. Thus clustering can help reduce number of vectors.

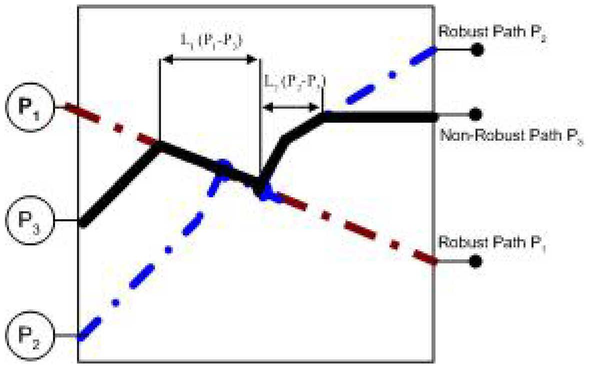


Figure 13 Clustering helps in NR path filtering

Since each vector pair using clustering detects more paths, the filtered path set may be further reduced. Consider a vector pair detecting two robust paths P_1 and P_2 that share at-least a small common segment and a NR path overlaps them as shown in Figure 13. Let segment S_{L1} of length L1 be the overlap of NR path with P_1 and segment S_{L2} with length L2 be the overlap of NR path with P_2. Further assume that L1 and L2 are both less than Δ_{NR} but $L1+L2 > \Delta_{NR}$. If the segments S_{L1} and S_{L2} are contiguous, we can drop the NR path and further enhance the definition of observation 1 made in Section 3.1. Thus clustering not only reduced the test set volume, but can also improve the process of filtering NR paths.

3.3 RESULTS

This section presents the results for combinational and full scan sequential *ISCAS'85* and *ISCAS'89* benchmark circuits. The programs were written using C++ and were simulated on a 1.7GHz, Pentium 4, running the Linux operating system. For the calculation of static implications, an implication engine presented in [23] was used. Table 1 presents the analysis on robust and NR paths. The second column shows the total number of paths present in each circuit. The paths include both rising and falling paths. Since the implication engine is not complete, we cannot say anything about the paths that were not detected as untestable. Column 3 gives the untestable NR paths (P_{UNR}) and column 4 reports the number of untestable robust paths (P_{UR}). Since untestable NR paths $\subseteq$ untestable robust paths, the number of robust paths (NR) needed to be considered for test generation is P-P_{UR} and number of paths considered for NR path ATPG is P_{UR}–P_{UNR}. The results of Table 1 suggest that there are a large number of paths that cannot be tested robustly or non-robustly.

Table 1 Untestable Robust and Non-robust paths

Circuit	# of paths P	Untestable NRP(P_{UNR})	Untestable RP(P_{UR})
C880	17284	163	326
C2670	1359920	1190899	1322192
C5315	2682610	2026131	2205279
S641	3488	1079	1280
S1196	6196	1289	1976
S1238	7118	2725	2793
S1423	89452	41102	52923
S1488	1924	0	0
S5378	27084	3645	3645
S9234	489708	419108	446665
S38584	2161446	1646624	1926898
S35932	394282	334713	355494
S38417	2783158	1469251	1795854

After filtering out untestable paths, we generate tests for longest N robust paths. Table 2 reports the results for robust ATPG. Column 2 of Table 2 gives N for various circuits using the deterministic algorithm. The upper limit on N was chosen to be 5000. Fewer robust paths are chosen if there were not 5000 robust paths in the circuit (e.g. S1196). Next, we report the results of the ATPG without and with clustering, respectively. The effect of clustering can be seen by comparing the number of detected paths and the number of vectors generated. For all cases, the number of vectors generated using clustering is less than the number of vectors generated without clustering, without any loss in the path coverage. This is because a group of paths are considered together for ATPG rather than targeting individual paths. Figure 14 shows the percentage decrease in the number of vectors because of clustering. It compares the results with the percentage reduction in the number of vectors using deterministic and Genetic Algorithm (GA) based approach. For most of the cases the former performs better than the GA based approach. But for some circuits like s35932, GA produces a large reduction in the number of vectors. This is because GA performs better for circuits which are not random pattern resistant. The average reduction of vector size is about 40%. Execution times for big circuits are about 4-5 times more with clustering but are still under limits. The test set produced after robust path ATPG is called V_{RD}.

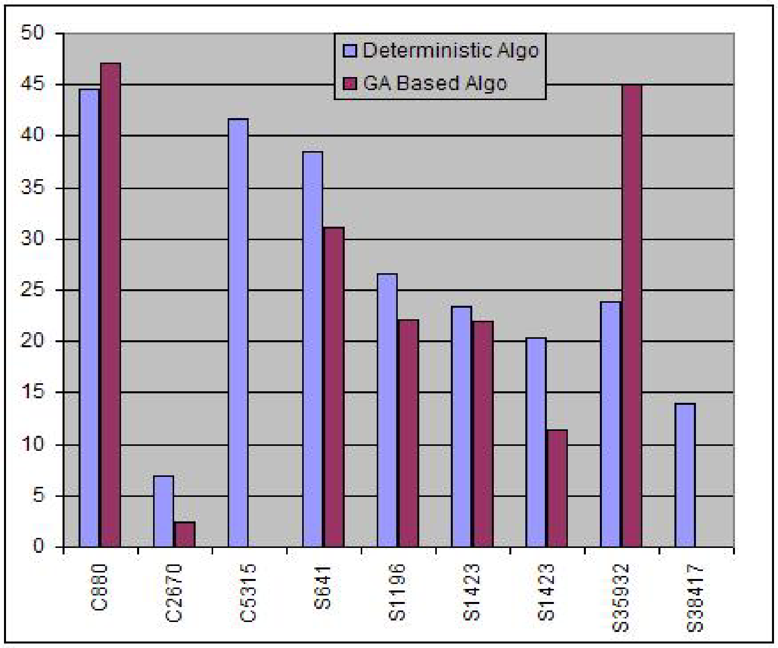


Figure 14 Percent reduction in # of vectors using clustering

Table 3a and 3b presents the ATPG result for NR paths. First, we filter out the NR paths that overlap with the tested robust paths. Then, we select all paths that are at least 85% longer than the longest path in the filtered set. Δ_{NR} (overlap threshold required to drop the path) was kept to be a path dependent quantity.

Table 2 ATPG results for robust paths

Circuit	# Paths (N)	Without Clustering			Clustering		
		#Det	#Vec	T_C(s)	#Det	#Vec	T_S(s)
C880	5000	4728*	5756	5.83	4728*	3190	29.27
C1355	5000	337	674	101.1	337	674	785.1
C2670	5000	3742	3744	73.99	3742	3484	242.81
C5315	5000	14*	24	10.85	14*	14	77.37
C7552	5000	34*	68	15.67	34*	68	166.67
S641	2208	2096*	1328	1.09	2096*	818	5.67
S1196	4220	3710*	2404	2.02	3710*	1766	11.26
S1423	5000	4822*	3934	10.64	4822*	3136	57.01
S1238	4325	3665*	2392	2.33	3665*	1832	10.1
S5378	5000	4048*	4186	11.24	4048*	2560	54.44
S9234	5000	3085	4458	337.36	3085	2394	5416.7
S35932	5000	4851*	2512	39.6	4851*	1914	130.44
S38417	5000	3638	5266	20895.23	3638	4536	66685.32

*** Rest all of the paths were proven to be untestable by the ATPG.**

Clustering was again used for this ATPG. Table 3 (a-d) shows results for the NR paths with and without filtering with varying values of Δ_{NR}. Specifically, we report results for Δ_{NR} of 100%, 90%, 80%, and 70%. The second column under NR path ATPG gives the number of NR paths detected/number of NR paths considered of ATPG (M_{NR}). Note here that Δ_{NR} = 100% means that no filtering has been done.

Paths in the set M_{NR} are covered under region 4 with reference to Figure 10 and also $M_{NR} \subseteq P_{NR}$. In our experiments we choose P_{NR} (# NR Paths) such that $P_{NR} \cap N_R = \Phi$. For most of the circuits the coverage is high at the cost of the addition of few extra vectors (V_{NR}) to the already present vector set V_{RD}.

The column P_{NRF} under *'NR filtered path ATPG'* represents the paths after filtering with varying values of Δ_{NR}. As Δ_{NR} decreases from 90% to 70%, the number of paths (M_{NRF}) goes on decreasing for almost all the circuits. This also results in the reduction of the number of vectors. Hence we can infer that filtering at a correct threshold not only decreases the number of vectors but also increases the delay quality vector set.

Table 3a. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=100% (No Filtering)			
	#NR paths (P_{NR})	#Det/M_{NR}	# Vec (V_{NR})	T_{NR}(s)
C880	163	129/129	100	0.85
C1355	5504	2752/4672	1896	23.3
C2670	131293	44760/63448	3452	12675.1
C5315	179148	15101/39447	4500	846.95
C7552	90442	3983/11030	3644	187.9
S641	201	11/11	0	0.09
S1196	147	6/12	0	0.14
S1423	11821	939/939	512	9.22
S9234	27557	590/6736	22	6909.16
S5378	0	0	0	0.0
S35932	20781	11888/12320	0	43.32

Table 3b. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=90%			
	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec (V_{NRF})	T_{NRF}(s)
C880	34	34/34	28	0.28
C1355	5504	2752/4672	1896	23.3
C2670	131293	44760/63448	3452	12675.1
C5315	179148	15101/39447	4500	846.95
C7552	90442	3983/11030	3644	187.9
S641	176	6/6	0	0.08
S1196	146	6/12	0	0.14
S1423	9386	392/392	114	4.06
S9234	27557	590/6736	22	6909.16
S5378	0	0	0	0.0
S35932	20781	11888/12320	0	43.31

Table 3c. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=90%			
	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec V_{NRF}	T_{NRF} (s)
C880	34	34/34	28	0.28
C1355	5504	2752/4672	1896	23.3
C2670	131247	44760/63448	3452	12675.1
C5315	179140	15095/39439	4500	962.11
C7552	90442	3983/11030	3644	187.9
S641	18	1 / 2	0	0.07
S1196	130	3/9	0	0.14
S1423	8544	286/286	90	3.01
S9234	27501	534/6680	22	6445.80
S5378	0	0	0	0.0
S35932	17693	8944/9232	0	33.82

Table 3d. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=90%			
	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec V_{NRF}	T_{NRF} (s)
C880	34	34/34	28	0.28
C1355	5504	2752/4672	1896	23.3
C2670	126740	40792/59468	3426	11635.1
C5315	178986	15033/39285	4482	839.72
C7552	90379	3956/10967	3684	185.86
S641	0	0	0	0.0
S1196	92	0/1	0	0.14
S1423	7107	1175/1445	750	22.75
S9234	27286	350/6496	22	6869.85
S5378	0	0	0	0.0
S35932	14763	6158/6302	0	28.88

Using a filtered set of paths enables us to detect a better path set, which is small and hence easy to detect. We can see that for some big circuits the reduction in path size (P_{NR}-P_{NRF}) is about 40%. Set $M_{NRF} \subseteq P_{NRF}$ and is essentially region 3 of Figure 10.

The increase in the number of paths when the threshold is reduced to 70% in circuit S1423 and C7552 can be explained by the definition of M_{NRF}. Since after filtering, the longest path remaining had a small length, set M_{NRF} for Δ_{NR} = 70% is greater than set M_{NRF} for Δ_{NR} = 80%.

The percentage decrease in the number of filtered paths produced with (Δ_{NR}=90%, 80%, 70% and 60%) and without filtering (Δ_{NR}=100%) is plotted in Figure 15 for various circuits. For almost all the circuits there has been a reduction in the number of paths required for testing to achieve high delay coverage. As the overlap threshold is increased, the reduction becomes lower since now there are more constraints on the filtering process. For example, number of paths reduce to about 75% with Δ_{NR} = 60% and about 20% with Δ_{NR} = 90%. We can also see that the number of vectors required are more with Δ_{NR}=100% (no filtering) than Δ_{NR}=70% in almost all the cases and the execution times are always less with the help of filtering. This proves that the concept of filtering helps us reduce number of vectors with an increase in delay coverage.

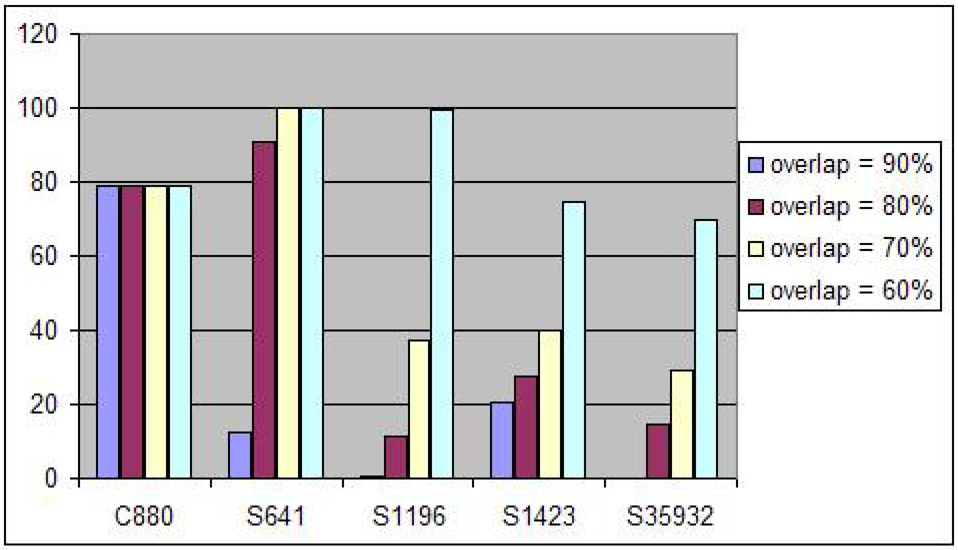


Figure 15 Percentage reduction in the # of paths using filtering

Table 4 presents the results for the transition fault coverage achieved. The second column presents the TFC for the vector set generated so far (V_{RD}+V_{NRF}=70%). We still need to perform transition fault ATPG for some circuits to account for the faults that V_{RD}+V_{NRF}=70% did not detect. For most of the circuits, the additional number of vectors (V_{TF}) added to the previous test set (V_{RD}+V_{NRF}) are very few since a lot of transition faults are detected while generating tests for the robust paths. For cases such as s35932, we don't need to add any additional vector. The incremental propagation based ATPG produces a high TFC for almost all the circuits in a reasonable amount of time. The last column presents the total number of vectors and total time taken to generate the whole vector set. The time is the sum of T_D + T_{NRF} + T_{TF} and the total vectors produced is the sum of V_{RD}+ V_{NRF} + V_{TF}. These final test sets achieve high robust coverage for the 5000 longest robust paths, high non-robust coverage for the filtered NR paths that do not significantly overlap with tested robust paths, and high transition coverage.

Table 4 TFC and the Total Test Set Size

Circuit	TFC (%) V_{RD} +V_{NRF}	Transition Fault ATPG			Final Test set	
		TFC (%)	#Vec (V_{TF})	T_{TF}(s)	#Vec ($V^{‡}$)	T (s) †
C880	98.69	100.0	22	4.16	3240	33.71
C1355	97.14	99.76 1	160	22.88	2730	831.28
C2670	82.9	87.83	296	317.0	7206	12194.91
C5315	99.1	99.54	22	45.79	4518	962.88
C7552	93.08	96.14	1069	1200.0	4821	1552.53
S641	100.0	100.0	0	0.0	818	5.67
S1196	99.84	100.0	9	1.64	1775	13.04
S1238	96.77	97.26	95	15.48	3231	72.49
S1423	98.2	99.2	122	29.1	2704	61.95
S9234	71.14	90.89	2705	3978.8	5122	10085.3
S5378	93.4	98.23	434	196.1	2994	250.54
S35932	90.5	90.5	0	0.0	1914	159.32

T(s) † = T_{RD} + T_{NRF} + T_{TF}; $V^{‡}$ = V_{RD} + V_{NRF} + V_{TF}.

CHAPTER 4

ALAPTF: A NEW TRANSITION FAULT MODEL AND THE ATPG ALGORITHM

As **L**ate **A**s **P**ossible Transition fault model is helpful in detecting small delay faults that cab be missed by the traditional Path delay fault model and the Transition fault model. The new ALAPTF model differentiates itself from the traditional transition fault model in the way that the new model is capable of detecting delays of much smaller sizes. At the same time, it can also outperform the PDF tests because not all paths have robust tests. For non-robust tests, only the transition at the start of the path is sure, while transitions along the path are not guaranteed. Moreover, the new model launches a transition such that the robust launching segment is the longest possible. Hence, the percentage of gates for which the transition reaches later by using the new ALAPTF model is much higher as compared to either the PDF or the traditional transition fault model. Since we propagate small delays to accumulate at gate nodes, the ALAPTF model is capable of detecting very small delays due to process variations along the circuit nodes.

An ideal plot for this measure will look something like Figure 16. In this figure, the X-axis denotes the size of the delay defect, and the Y-axis represents the percentage of gates within the circuit under test (CUT). Two curves are shown in the figure: the old (traditional) transition fault model and the new ALAPTF model. Because the traditional model aims at capturing gross delay defects, larger defect sizes will naturally translate to a higher coverage. However, with our new model, we expect the coverage to be much higher for smaller delay sizes than the coverage by the traditional transition fault model, if not also higher than PDF models. A side benefit for the proposed approach is that the longest robust and non-robust path delay faults through each gate will also be simultaneously detected. Results reported in Section 4.4 show that with the new model, much smaller delay defects could be captured when compared with PDF tests or traditional transition tests. The computation effort is only of the order of circuit size.

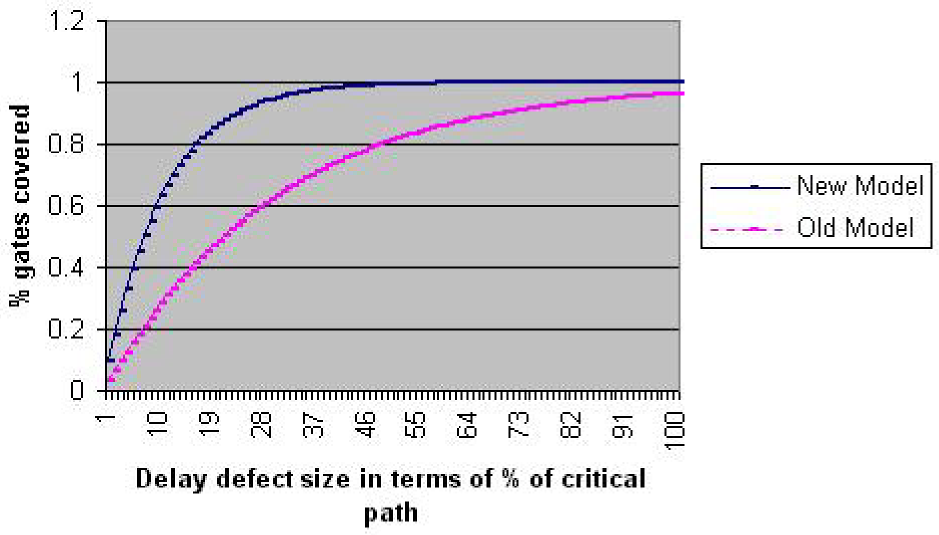


Figure 16 Ideal plot for gate coverage with varying values of delay

4.1 ALAPTF: The New Fault Model

Lemma 1: A transition fault can be launched robustly, non-robustly, or neither through the segment PI-fault site.

Proof: Consider a slow-to-rise transition fault at the output of a simple two-input OR gate G. This transition can be launched by having rising transitions at both inputs of G. Hence, none of the 2 paths are robustly/non-robustly tested. It can also be launched by having a transition on one input, while the other input is at a steady 0, forming a robust test for one of the paths. A non-robust test can be constructed in a similar manner for a slow-to-fall transition on gate G.

Lemma 2: A detected transition fault might not be detected by a robust/non-robust segment starting from the fault site f to a PO.

Proof: Consider the circuit shown in Figure 17. A slow-to-fall transition fault at G1 is propagated to the PO and hence detected, but both the paths are blocked (robustly or non-robustly) due to off-path inputs at gate G2.

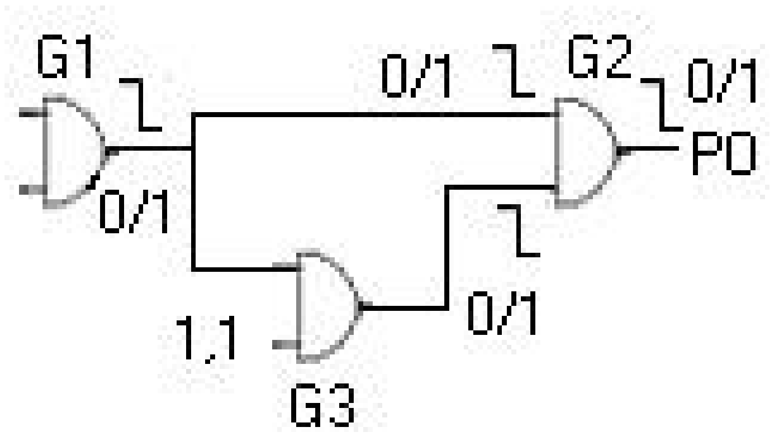


Figure 17 A transition fault need not propagate through a Robust/NR path

We know that the PDF model is used to capture small-distributed delay defects whereas the transition fault model is used to capture gross delay defects. From *Lemma 1* and *Lemma 2*, because transition fault detection may be neither robust nor non-robust, we can formulate a methodology to combine the small delay defects present before a fault site and propagate them as one transition fault. Please note that even the segment delay model may fail, because the overall path may be non-robustly untestable.

To implement this methodology, we have proposed a new model for transition faults in this Chapter. Consider the case shown in Figure 18. It shows a structural path P of length L_p (with solid lines). Let us assume that this path is robustly untestable and there are some small delay defects present at node D_1 through D_4, such that $D_1 + D_2 + D_3 + D_4 > T_{clock}$. The robust path delay model cannot capture this delay fault because P is robustly untestable. Moreover the traditional transition fault model may not capture this fault since each $D_{i\ (i=1,2,3,4)} < T_{clock}$ and in general each transition is launched via a short path from the PIs and its effect is propagated through the shortest propagatable paths to a PO. Hence, we need a fault model, which can group the effect of the entire D_i's and present them as a lumped delay at node D_{i+1}. The traditional transition fault model can then test this lumped delay defect.

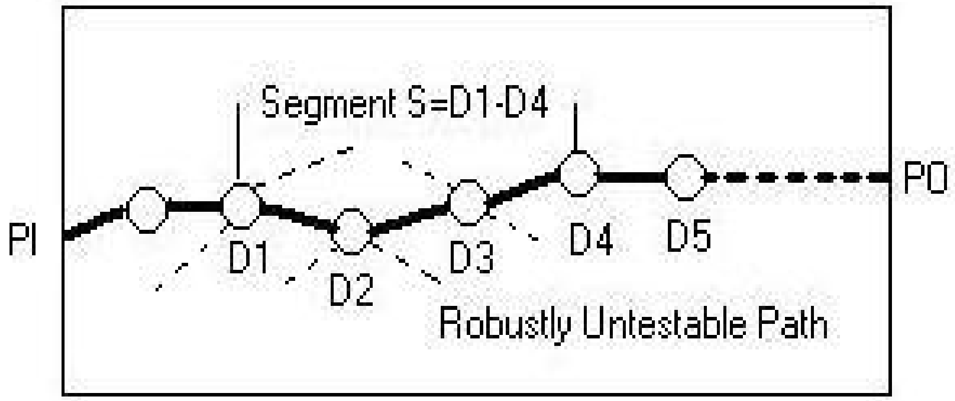


Figure 18 ALAPTF Model

The new transition fault model, which considers the above-mentioned problem, is called As Late As Possible Transition Fault (ALAPTF) Model. A fault *f* under this model is detected if:

1) The fault *f* is launched at the fault site as late as possible. This means that the fault should be launched by one of the longest robust segment ending at the fault site. Note that the segment needs not start from a PI. Hence, from Figure 18, we want the fault at gate D_5 to be propagated robustly along the segment $S=D_1-D_4$ if this is the longest testable segment present for D_5. This will ensure that the small delay defects of all the nodes in the robust segment S, gets accumulated at the final node which can then be tested traditionally.

2) The fault is propagated to a PO from one of the longest paths starting from the fault site. This will make sure that we cover long paths of propagation and hence can detect the fault more efficiently.

3) The traditional transition fault model detects *f* via this ALAPTF path.

Some of the advantages of using ALAPTF are as follows:

1) Since the transition is launched through one of the longest robust segment, all the small delay defects before the fault site along with the gross delay defects on the fault site will be detected.

2) The new test set will be able to capture defects of much smaller size than traditional transition fault model.

3) The complexity is linear to the circuit size, which is much smaller than PDF or segment delay models, which could be exponential to the circuit size.

4) The algorithm will simultaneously detect longest robust and non-robust PDFs (if they exist) through each gate. Note that if the PDFs are not testable, we are still able to launch the transition as late as possible neither robustly nor non-robustly as explained in Lemmas 1 and 2.

A special case of ALAPTF model is the condition when the robust segment does not start from a PI. This condition arises when all the robustly testable segments $S_{PI\text{-}FS}$ (from PI to transition Fault Site) are smaller than a robustly testable segment not starting from a PI. This condition can only occur when there are re-convergent fanouts and the inversion parity at the re-converging gate is same. This condition is explained in Figure 19.

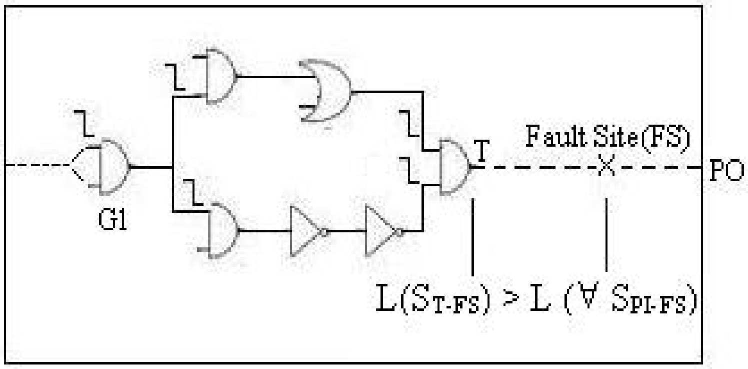


Figure 19 Special Case for ALAPTF

We can see that both the branches of gate G1 re-converge with the same inversion parity at T. Both of the paths are robustly and non-robustly untestable. Hence, under the condition $L_{(}S_{T\text{-}FS)} >$ $L_{(}\forall\ S_{PI\text{-}FS)}$, the latest transition that can arrive at the Fault Site can come from a non-PI segment, i.e. $S_{T\text{-}FS}$. Note that the two paths are each non-sensitizable, but are co-sensitizable. The difference between our technique and the segment delay model is also evident from Figure 19. Since the segment delay model requires the entire path to be at least non-robustly testable, the transition fault model does not require the segment $S_{FS\text{-}PO}$ to be tested non-robustly.

4.2 ATPG FOR ALAPTF MODEL

The test generation focuses on finding a test for each fault such that the fault is launched as late as possible and is propagated to a PO through a long path. To implement ALAPTF we use a two-step procedure. The first step is a pre-processing step based on simulation, whereas the second is the deterministic ATPG process.

4.2.1 PRE-PROCESSING

The first step towards ATPG is to estimate the latest time at which a fault can be launched at each fault location. This information will be used in the second stage of the ATPG. A simple topological model can also be used (*LRB* set in [10]) but since a large amount of long paths are robustly untestable, this approach can be very expensive if every topological path needs to be verified for robust sensitizability. To overcome this problem and to estimate a better solution of the problem, we used a Genetic Algorithm (GA) based technique. We consider a unit gate delay model (every gate has a delay of one unit) for the pre-processing step. However, other more elaborate delay models can be used also. The pseudo algorithm is shown in Figure 20.

```
Void pre_processing() {
 For each gate g {
   Late_1[g] = GA (g,1) //a 'one' should reach g ALAP
   //fitness = time at which 'one' reaches the gate
   Late_0[g] = GA (g,0) //a 'zero' should reach g ALAP
   //fitness = time at which 'zero' reaches the gate
   }
}
```

Figure 20 Pre-processing step for ALAPTF ATPG

The method finds a time that a logic 1 and logic 0 can reach each gate. Since it is a unit gate delay model, the latest time will directly correspond to the structural length of the segment. For example, if we find out that for a gate *g*, a '1' will reach latest at time unit 10, then there is definitely a segment S of length 10 ending at *g* such that we can propagate a robust rising transition at *g* along S. The primary difference between finding the LRB set [10] for each gate and this approach is that the length produced for each gate is simply not the topological maximum. Moreover, the lengths produced are indicative of the segment lengths that may or may not start from a PI. Since we are using a GA based approach, the lengths produced are not guaranteed to be the maximum. But, the length L produced here for a gate G guarantees that there exists a segment of length L ending at G, such that, we can propagate a robust transition along it.

4.2.2 ALAPTF ATPG

The second step towards finding the vector set is the following deterministic ATPG algorithm. To produce a test for all the transition faults so that ALAPTF criterion is satisfied, we use a reverse approach, i.e. first generate a test for a given path and then drop all the faults detected via the produced test vector. We use an approach similar to that of RESIST [6] to generate test for all the paths starting from a given PI. Since we want the transition to launch and propagate from a long path, we generate test for all the paths and drop the faults once a vector pair is generated. The following steps explain the algorithm.

1) Since the vector V_2 is same for both robust and non-robust test for a given path, we generate V_2 first via a recursive search.

2) Many paths starting from the same PI are structurally similar; hence we use a RESIST-based approach to produce a test for a path, by reducing re-computation of side-inputs for the common segments along the paths. In our algorithm, we restrict ourselves to finding V_2 only. Our approach for generating V_2's for a given PI and transition, for all paths starting from PI, can be understood by the flowchart of Figure 21.

3) After generating V_2, compute the values required by the gates on P by V_1 so that P becomes robustly testable and store them in Set_V1.

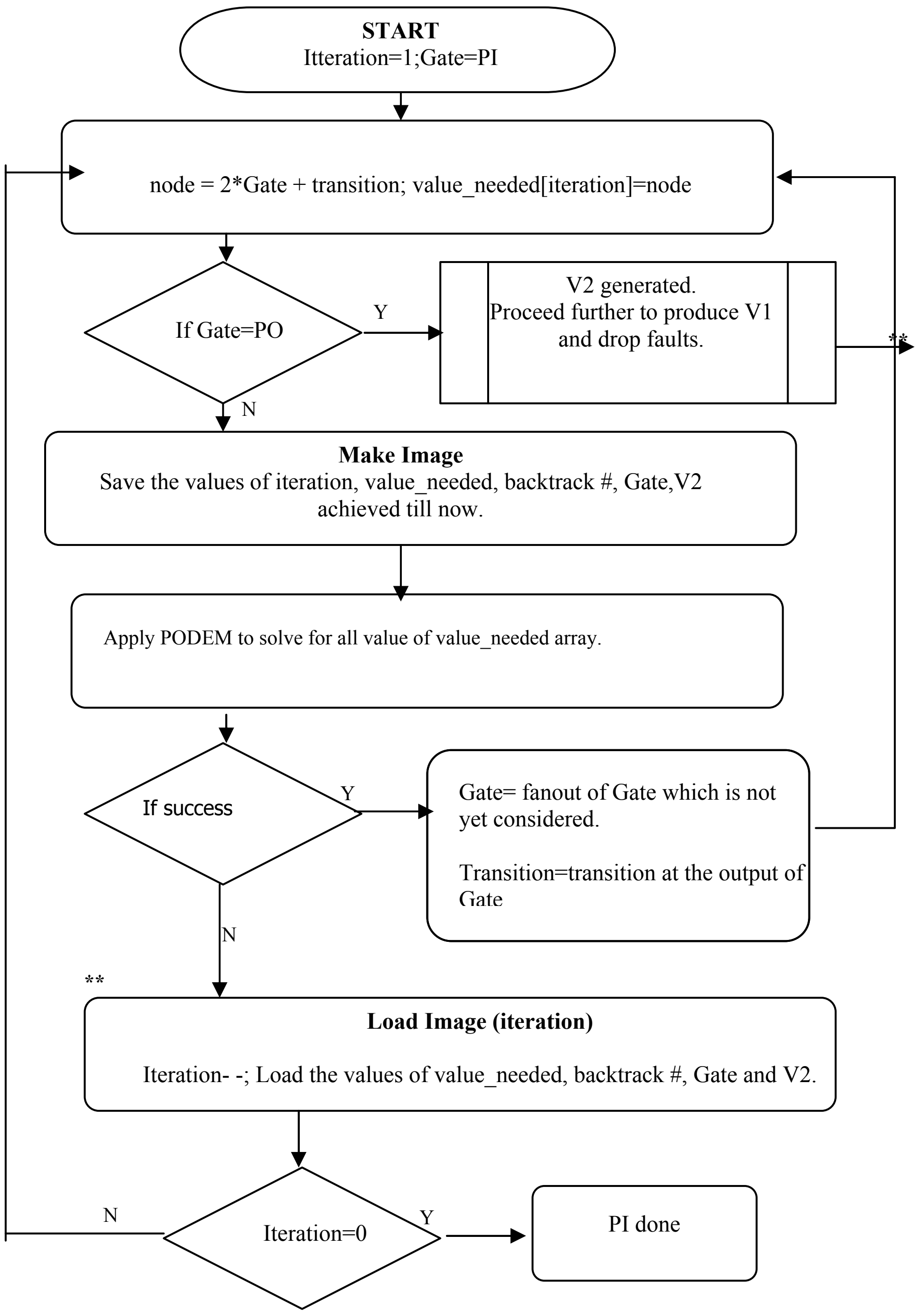

START
Itteration=1;Gate=PI
node = 2*Gate + transition; value_needed[iteration]=node
If Gate=PO
Y
N
V2 generated.
Proceed further to produce V1 and drop faults.
**
Make Image
Save the values of iteration, value_needed, backtrack #, Gate,V2 achieved till now.
Apply PODEM to solve for all value of value_needed array.
If success
Y
N
Gate= fanout of Gate which is not yet considered.
Transition=transition at the output of Gate
**
Load Image (iteration)
Iteration- -; Load the values of value_needed, backtrack #, Gate and V2.
Iteration=0
N
Y
PI done

Figure 21 Flowchart for finding V2 for all paths starting from a given PI

4) Since we want to launch every transition ALAP, we want to satisfy as many contiguous values as possible in the set Set_V1.

5) If all the values of Set_V1 are satisfied, then the test becomes a robust test for P. If the first value is satisfied (which is simply an assignment to PI), then it is a non-robust test for P.

6) Drop all the transition faults detected by the vector pair <V_1, V_2>, if the faults satisfy all the conditions required by the ALAPTF model, i.e.

    a. The fault should be detected in the traditional sense.
    b. The fault should be launched by at least a segment of length ≥ the length found in pre-processing step.
    c. The fault should be propagated through one of the longest paths through the fault site.

7) If no test is found for a fault f such that it satisfies all 3 criterions then we add a vector that at least detects the fault traditionally.

8) For circuits like c6288 where there are about 217 potentially testable paths, we abort on a PI as soon as we have tested at least 5000 paths starting from it.

At the end of the process we have a test set that has a maximum robust and non-robust coverage (if no paths are aborted) along with high transition fault coverage (TFC). If we want to have high TFC alone, we can simply keep the vectors that detect at least one fault in step 7.

4.3 EXPERIMENTAL RESULTS

This section presents the results for combinational and full scan sequential ISCAS'85 and ISCAS'89 benchmark circuits. The programs were written using C++ and were simulated on a 1.8GHz, Pentium 4, 512MB RAM machine, running the Linux operating system.

Table 5 presents the results obtained by our approach for ISCAS'85 and full-scan ISCAS'89 benchmark circuits. Second column reports the transition fault coverage for the corresponding circuits. It should be noted that not all faults were detected by the ALAPTF criterion. Some of them were detected by relaxing the detection criterion since those faults do not satisfy the first two ALAPTF detection criteria. We can see that for all the circuits, the TFC reached is the maximum TFC that can be achieved. The third and the fourth column give the robust and the NR coverage. Since the ATPG is a deterministic algorithm, the path coverage is maximum for most of the cases. For c6288, we aborted on each PI node after every 5000 paths. The last column gives the execution time in seconds. The time is under limits even in the case of circuits such as c1355 and c5315, which have a large number of paths. This is because of the recursive nature of the algorithm due to which we need not generate test for each path from the start every time. Hence, one of the advantages of using this algorithm is that in only one pass, we can find tests for all three models of delay tests.

Table 5 Results of the proposed ATPG Algorithm

Circuit	TFC(%)	# of Robust Paths Detected*	# of NR Paths Detected*	Time(s)
C880	100.0	16489	16652	19.4
C1355	99.7	200462	841613	4533.2
C2670	85.06	33156	130638	4287.0
C5315	99.54	186635	341634	10564.2
C6288†	99.18	45853	305894	2223.1
C7552	95.74	192021	274920	3531.2
S641	99.05	2092	2266	8.41
S713	94.2	2066	4922	31.4
S1196	100.00	3710	3759	9.73
S1423	99.17	33981	45182	210.7
S5378	97.87	19398	21919	306.2
S9234	89.78	38593	59830	6615.9
S35932	90.5	38372	58657	4891.7

*** If all the vectors are considered. † Only 5000 paths per PI are considered.**

To prove that the new test set T_{new} launches the transitions as late as possible, we compared T_{new} with the test set (T_{old}) produced by the traditional transition fault ATPG. Figure 22 plots the result for various circuits. The y-axis represents the percentage of gates on which a transition reaches later by T_{new} as compared to T_{old}. This shows that T_{new} is indeed better in performance that T_{old}. There are on average 30% gates for which transition reaches later by T_{new}. The test set was then simulated for the same parameter against a path delay fault test set T_{path}. This set was produced by the method presented in [10]. Figure 23 shows the corresponding result. Here we can see that for roughly 20% of the gates, the transitions were launched later. Moreover, the technique presented in [10] was not able to generate tests for circuits having a large number of long untestable paths (e.g. c6288), but the technique presented here is able to generate test for these circuits as well.

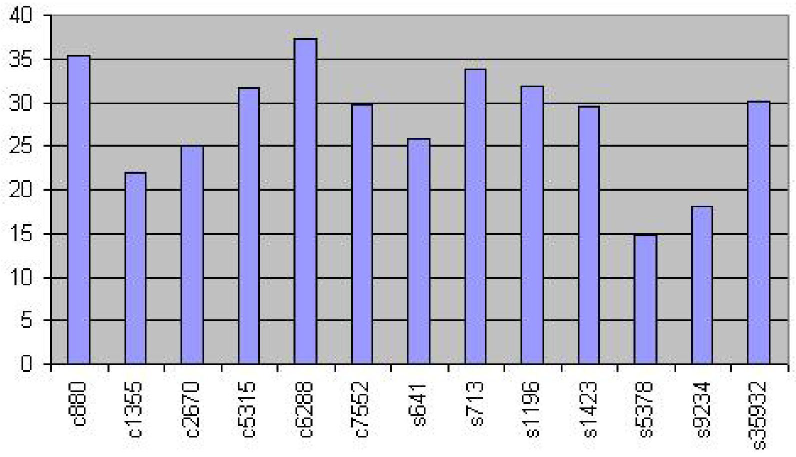


Figure 22 Percentage of gates for which transition reaches later by using T_{new} compared with T_{old}

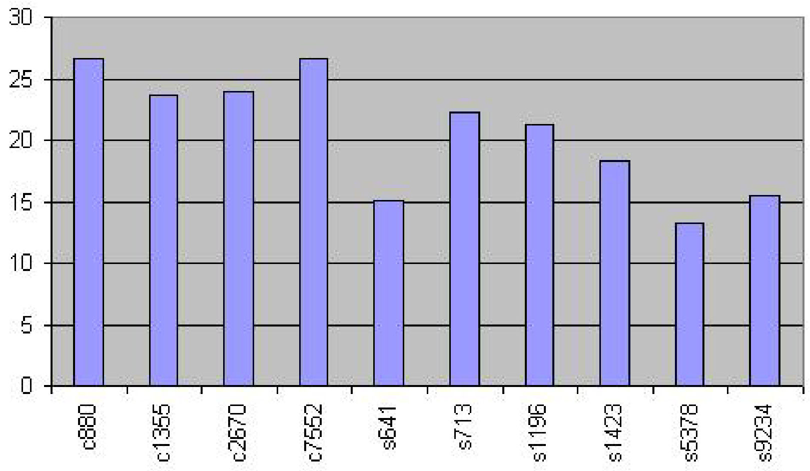


Figure 23 Percentage of gates for which transition reaches later by using T_{new} compared with T_{path}

In order to show that the T_{new} can detect small delays as compared to T_{old}, we inserted random delays on each gate. We then simulate the good and the faulty circuit by T_{new} and T_{old} to see if the defects could be captured by either of the test sets. The sizes of the defects were varied for this experiment. We considered a delay model in which the nominal delay of each gate *g* was kept equal to Dg = 5+ # of fanin(*g*)+ # of fanout(*g*). This model can be changed and we can also use real values of delay of each gate. A Gaussian distribution with standard deviation (SD) of 10.0 was used to generate random delay sizes. *Wichman-Hill transform* [24] was used to generate a uniform random variable, which was then converted to a Gaussian random variable by using the *Box-Muller* [24] method. Table 6 reports the results for some of the benchmark circuits. Let the nominal delay of the critical path in the circuit be L_d. Then, for the second column (1%) of Table 6, we shift the Gaussian probably distribution curve by an amount equal to 1% of L_d. This means that the amount of delay added on each gate is about 1% of L_d and can vary up to ± 10 units (because of SD). We can see that as the size of the delay increases, the number of gates for which we get a faulty response increases with both T_{new} and T_{old}. But T_{new} is capable of detecting delay faults on more number of gates even for smaller values of delays. As the delay approaches infinity, both the test sets will detect the same number of gates, since both have the same TFC.

Table 6 Number of faults detected by varying amounts of added delay

Ckt	*1%*		*5%*		*10%*		*50%*		*80%*	
	T_O	T_N	T_O	T_N	T_O	T_N	T_O	T_N	T_O	T_N
C880	0	1	0	68	54	80	249	258	350	350
S641	0	0	0	0	0	86	155	243	338	344
S713	0	0	0	0	0	0	123	231	347	356
S1196	0	23	29	71	57	86	322	323	533	534
S1423	0	60	0	82	0	100	67	345	403	496

The results of Table 6 are also plotted in a 3-D diagram under Figure 24. We can see that the curves of T_{old} and T_{new} are apart at the beginning and become closer as the delay size reaches a large value. The curves should be compared with the predicted curves of Figure 16. It is evident that the plot of Figure 24 closely resembles to Figure 16.

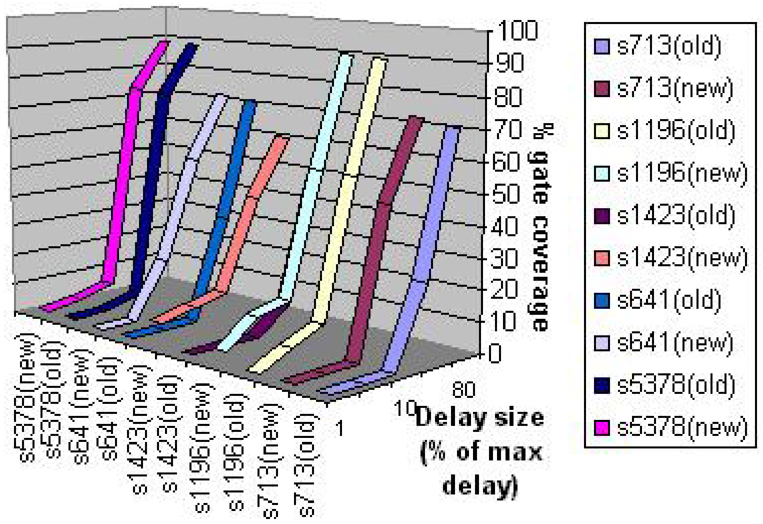


Figure 24 Percent gate coverage with varying amounts of delays

CHAPTER 5

CONCLUSION AND FUTURE WORK

Although a large amount of work has been done in the past relating to path delay faults and transition faults, delay tests still remain one of the greatest challenges in the fielding of testing. Due to the new 90nm and 65nm technologies, delay testing is becoming more and more important. New methods are required to test small delay faults along with the standard transition faults. We have tried to come up with some alternate techniques for path selection and testing of small delay faults which might overcome some of the problems faced by the previous work done in this area. In the thesis we have addressed the following issues:

- A high quality Delay Fault ATPG based on proper path selection was presented. Robust paths, non-robust paths and transition faults were considered for ATPG. Since the number of paths in circuits can be huge, measures are taken to select specific paths for ATPG. Selecting non-robust paths based on their lengths can be non-optimal and hence we drop non-robust paths that significantly overlap with an already-tested robust path, and the results show that the final test set is rich in all three aspect of delay testing. In other words, the obtained test sets capture both gross and distributed delay defects in the circuits. For transition fault ATPG, a special incremental propagation algorithm is proposed to reduce the vector space and generate a high Transition Fault Coverage test vector. Results for ISCAS’85 and full-scan ISCAS’89 benchmark circuits show that the filtered non-robust path set can be reduced to 40% smaller than the conventional path set without losing delay defect coverage. Clustering of paths has been shown to improve the fault coverage and also reduce the number of vectors by 40%.

- A new transition fault model called As Late As Possible Transition Fault (ALAPTF) is presented. The model is useful in detecting small delay defect which can be missed by both the transition and path delay fault model. The algorithm uses both Genetic Algorithms and Deterministic Algorithms for the test generation. A recursive method for the ATPG ensures a high PDF coverage along with the transition fault coverage. Even circuits like c6288 can be processed using this method with some constraints. The new model has been shown to perform better than the existing transition fault model and the path delay fault model. At the same TFC as produced by the older model, the ALAPTF model, finds a test set which launches the transitions on each gate as late as possible. For about 30% times the transition reaches later by using the new fault model which means that the transition are detected via long paths. It is also capable of detecting much smaller delays (delays much smaller than the critical path delays). The algorithm proposed also detects robust and non-robust paths along with the transition faults and the execution time is linear to the circuit size.

REFERENCE

1) A. K. Majhi and V.D. Agrawal, "Delay Fault Model and Coverage", Proceedings of the VLSI Design Conference, 1998.

2) G. L. Smith "Model for delay faults based upon paths," International Test Conference, 1985.

3) J. Savir and S. Patil, "On Broad Side Delay Test", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 2, Issue: 3, Sept. 1994, Pages: 368 – 372.

4) 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.

5) K. T. Cheng, and H. C. Chen, "Classification and identification of non-robust untestable path delay faults", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 15 , Issue: 8 , Aug. 1996, Pages: 845 – 853.

6) K. Fuchs, M. Pabst and T. Rossel, "RESIST: A recursive test pattern generation algorithm for path delay faults considering various test classes", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 13, Issue: 12, Dec. 1994 Pages: 1550 – 1562.

7) K. Fuchs, F. Fink and M. Schulz, "DYNAMITE: An efficient automatic test pattern generation system for path delay faults ", IEEE Transactions on CAD Computer-Aided Design of Integrated Circuits and Systems, Volume: 10, Issue: 10, Oct. 1991 Pages: 1323 – 1335.

8) S. Yihe and W. Qifa, “FSIMGEO: A Test Generation Method for Path Delay Fault Test Using Fault Simulation and Genetic Optimization”, ASIC/SOC Conference, 2001. Proceedings. 14th Annual IEEE International, 12-15 Sept. 2001, Pages: 225 – 229.

9) M.Sharma and J.H. Patel, “Testing of Critical Paths for Delay Faults”, International Test Conference, 2001. Proceedings. 30 Oct.-1 Nov. 2001, Pages: 634 – 641.

10) M. Sharma, J.H. Patel, “Finding a small set of longest testable paths that cover every gate”, International Test Conference, 2002. Proceedings. 7-10 Oct. 2002 Pages: 974 - 982

11) W. Qiu and D. M. H. Walker, “An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit”, International Test Conference, 2003.

12) J.J. Liou, L.C. Wang, K.T. Cheng, “On theoretical and practical considerations of path selection for delay fault testing”, International Conference on Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM, 10-14 Nov. 2002, Pages: 94 - 100.

13) J.J. Liou, A. Krstic, L.C. Wang, K.T. Cheng “False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation”, Design Automation Conference, 2002. Proceedings. 39th , 10-14 June 2002, Pages: 566 – 569

14) M. L.Bushnell and V. D. Agrawal, “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits”, Kluwer Academic Publishers, Boston, 2000.

15) I. Dervisoglu and G.Stong, “Design for Testability: Using Scan-path Techniques for Path-delay Test and Measurement”, Test Conference, 1991, Proceedings., International , 26-30 Oct 1991, Pages:365

16) E. M. Rudnick, J. H. Patel, G.S. Greenstein and T. M. Niermann, “A Genetic Algorithm Framework for Test Generation,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 16 , Issue: 9 , Sept. 1997, Pages:1034 – 1044.

17) D.G. Saab, Y.G. Saab and J.A. Abraham, “CRIS: A test cultivation program for sequential VLSI circuits”, International Conference on Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers. 1992 IEEE/ACM, 8-12 Nov. 1992, Pages:216 - 219.

18) M. S. Hsiao, E. M. Rudnick, and J. H. Patel, “Application of genetically-engineered finite-state-machine sequences to sequential circuit ATPG”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 17 , Issue: 3 , March 1998, Pages: 239 – 254.

19) X. liu, M. S. Hsiao, S. Charravarty, P. J. Thadikaran, “Novel ATPG Algorithms for Transition Faults”, The Seventh IEEE European Test Workshop, 2002. Proceedings., 26-29 May 2002, Pages: 47 – 52.

20) P. Gupta and M. S. Hsiao, “High Quality ATPG for Delay Defects”, International Test Conference, 2003. September, 2003, Page: 584-591.

21) P. Gupta and M. S. Hsiao, “High Quality Delay Testing”, Proceedings of the IEEE Concurrent and Defect-Based Testing Workshop, April 2003.

22) K. Heragu, J.H. Patel, V.D. Agrawal, “Fast Identification of Untestable Delay Faults using Implications”, International Conference on Computer-Aided Design, 1997. Digest of Technical Papers. 1997 IEEE/ACM, 9-13 Nov. 1997, Pages: 642 – 647.

23) M. S. Hsiao, "Maximizing impossibilities for Untestable fault identification”, Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, 4-8 March 2002, Pages: 949 - 953.

24) W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing, (Cambridge University Press, Cambridge, 1988.

VITA

Puneet Gupta was born in Udaipur, a town in Rajasthan, India. He did his schooling from Udaipur, India. He joined Regional College of Engineering, Warangal under the Kakatiya University in 1997, to obtain technical education in the area of electronics and communications. After graduating with a bachelor's degree in May 2001, he joined Virginia Polytechnic Institute and State University in August 2001 for a Masters degree in the Bradley Department of Electrical and Computer Engineering. He joined Dr. Michael Hsiao and his research group in January 2002 and has since then been involved in research related to hardware testing. He currently got a job in Cadence Design Systems as a Senior Member Technical Staff in there Test Design Automation Group in Endicott, NY. He is working currently on delay test ATPG methodologies for the Cadence Encounter Test Platform. His technical interests include VLSI testing, verification and VLSI design. His long term goals are to contribute to the scientific community in the areas of hardware testing and verification.