AT-SPEED PATH DELAY TEST

A Thesis

by

SWATI CHAKRABORTY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Duncan M. H. Walker
Committee Members,	Rabi N. Mahapatra
	Sunil P. Khatri
Head of Department,	Dilma Da Silva

May 2015

Major Subject: Computer Engineering

Copyright 2015 Swati Chakraborty

ABSTRACT

This research describes an approach to test metastability of flip-flops with help of multiple at-speed capture cycles during delay test. K longest paths per flip-flop test patterns are generated, such that a long path on one clock cycle feeds a long path on the next clock cycle, and so on. Traditional structural delay tests do not test whether time borrowing or stealing is working correctly, since only a single at-speed cycle is tested.

To detect path delay faults for the multi-cycle paths, it is necessary to start a path at a register and end at a register while passing through another register, testing the longest paths between each pair of registers. This requires three or more at-speed cycles, rather than the two of traditional Launch on Capture test. This produces power supply noise closer to functional mode, and permits the testing of flip-flop metastability and time-borrowing latches, that cannot be tested by any other structural test technique. The path generation algorithm uses the circuit structure, and then the paths are sequentially justified using Boolean Satisfiability algorithms.

The algorithm has been implemented in C++ on an Intel Core i7 machine. Experiments have been performed on various ISCAS benchmark circuits in both robust and non-robust path generation technique to evaluate our approach.

ACKNOWLEDGEMENTS

I would like to express my profound gratitude and respect for my M.S. Committee Chair and adviser Dr. Duncan M. (Hank) Walker. I would like to thank him for his guidance, support, encouragement and patience throughout the course of this research.

I am grateful to my Committee Members Dr. Rabi N. Mahapatra and Dr. Sunil P. Khatri and would like to express my thanks and respect to them for their suggestions and encouragement.

I would also like to thank all the faculty, staff and friends in the department.

Finally, thanks to my parents and my husband for their love and support throughout this journey.

TABLE OF CONTENTS

Page

ABSTRACT ii

ACKNOWLEDGEMENTS iii

TABLE OF CONTENTS iv

LIST OF FIGURES vi

LIST OF TABLES viii

1. INTRODUCTION 1

1.1 Path Delay Test 1
1.1.1 Delay Test Problem 1
1.1.2 Path Sensitization 3
1.1.3 Robust and Non-Robust Path Delay Tests 3
1.2 Scan-based Delay Test 4
1.2.1 Muxed-D Scan Approach 5
1.2.2 Enhanced Scan Approach 6
1.3 At Speed Testing 7
1.3.1 Launch on Shift 7
1.3.2 Launch on Capture 8
1.4 KLPG Algorithm 10
1.4.1 Pseudo-functional KLPG 12
1.5 Boolean Satisfiability 13
1.5.1 MiniSat 15
1.5.2 Use of SAT in CodGen 16
1.6 Structure of the Thesis 16

2. MOTIVATION 17

2.1 Time Borrowing and Time Stealing 17
2.2 Metastability 19
2.3 At Speed Test Approach 20
2.4 Faster than At-Speed Delay Test 21
2.5 Other Related Work 22

3. IMPLEMENTATION 23

3.1 Test Generation Strategy for Enhanced CodGen for Multiple Capture Cycles 23
3.2 Aim at Scan-flops 26
3.3 Finding Fan-in and Fan-out Cone 27
3.4 Complete Path over Multiple Capture Cycles 28
3.5 Tracking the Number of Paths Generated 29
3.6 Time-frame Expansion 29

4. RESULTS 31

4.1 Paths Generated across Different Capture Cycles for Robust and Non-Robust Case 31
4.2 Fault Coverage across Different Capture Cycles for Robust and Non-Robust Case 37

5. CONCLUSIONS AND FUTURE WORK 44

REFERENCES 46

LIST OF FIGURES

Page

Figure 1. Delay fault problem definition ... 2

Figure 2. Untestable path ... 3

Figure 3. Scan-based test ... 5

Figure 4. Muxed-D scan cell ... 5

Figure 5. Muxed-D scan design ... 6

Figure 6. Enhanced-scan design ... 7

Figure 7. Launch on Shift ... 8

Figure 8. Launch on Capture ... 9

Figure 9. Justification in LOC ... 9

Figure 10. KLPG algorithm ... 10

Figure 11. Drop in power supply voltage during delay test ... 13

Figure 12. Pseudo functional test ... 13

Figure 13. Time borrowing ... 17

Figure 14. Metastability ... 19

Figure 15. At speed functional test ... 20

Figure 16. Multiple capture cycles in at-speed path delay test ... 24

Figure 17. Flow-chart for CodGen with multiple capture cycles ... 26

Figure 18. Scan-flop with fan-outs ... 27

Figure 19. Complete path over multiple capture cycles ... 28

Figure 20. # Paths in circuits for different capture cycles for robust case ... 33

Figure 21. Longest path length in different capture cycles for different circuits for robust case 34

Figure 22. # Paths in circuits for different capture cycles for non-robust case 36

Figure 23. Longest path length in different capture cycles for different circuits in non-robust case 37

Figure 24. Fault coverage across different capture cycles in robust case 39

Figure 25. Fault coverage trend in circuit s1494 across different capture cycles for robust case 40

Figure 26. Fault coverage across different capture cycles in non-robust case 41

Figure 27. Fault coverage trend in circuit s1494 across different capture cycles for non-robust case 42

LIST OF TABLES

Page

Table 1. # Paths Generated, Longest Path Length, Time for Robust Case, K=1 32

Table 2. # Paths Generated, Longest Path Length, Time for Non-Robust Case, K=1 35

Table 3. Fault coverage in robust case 38

Table 4. Fault coverage in non-robust case 41

1. INTRODUCTION

1.1 Path Delay Test

Delay testing is used to test delay faults that affect the maximum operating speed of an integrated circuit. The delay can be modeled by a delay fault model. One of them is path delay fault model. A path is sequence of gates in the circuit from a primary input to a primary output and there is a transition at each gate [1]. The gate input on the path is the on-input and the other inputs are side-inputs [1]. In path delay fault model, a path has delay fault if the delay of the path exceeds some specified duration [2] [3]. The delay of a path is the amount of time needed to propagate a signal from the start gate of the path to the end gate. Many studies have been done to test longest paths in a circuit [4][5][6][7][8][9].

As the number of paths in a circuit is exponential in terms of the circuit size, identification of the longest sensitizable paths through each gate or line is extremely difficult [10]. To make the test tractable k longest paths per gate are tested in [10]. This test accounts for both local delay defects (e.g. a logic gate is slow) and global process variation (several different paths through a gate might be the slowest).

1.1.1 Delay Test Problem

The delay test requires two test patterns or vectors for launching transitions. The first vector is the initialization vector and the second vector is the test vector. Transitions are launched into the circuit through the primary inputs (PIs) and pseudo primary inputs

(PPIs) and the responses are captured through primary outputs (POs) and pseudo primary outputs (PPOs) [1].

Figure 1 below illustrates the concept of delay fault through gates. When a rising transition is put at the input of an inverter, output of the inverter experiences a falling transition. The delay between the rise and fall transition is determined by the characteristic of the gate. The shaded region in the picture represents the time in which the output of the inverter is expected to complete the transition caused by the transition at the input. When the path from the input to output of the inverter experiences an additional amount of delay, the falling transition at the output gets shifted outside of the shaded region. This is characterized as the delay fault.

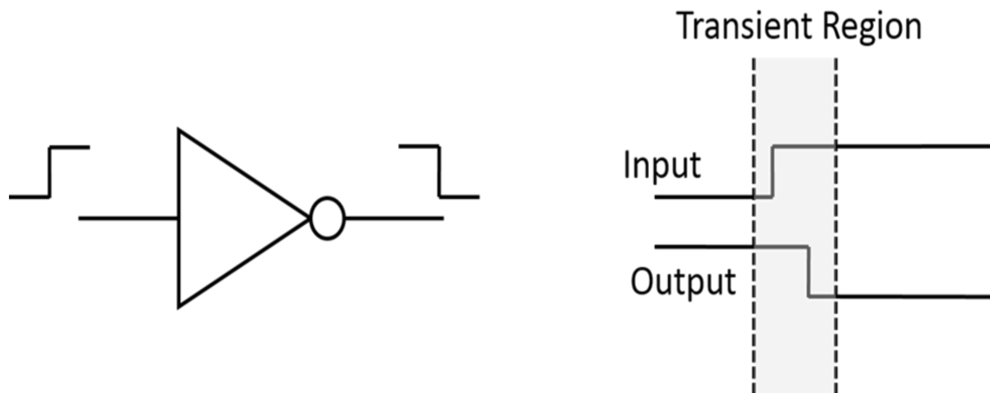

Figure 1. Delay fault problem definition

A combinational logic has various paths. The delay of a given path varies with the number of gates in the path and also by the fan-outs of a given gate. The path with the longest delay in the circuit is called the critical path. The critical path of a circuit defines the maximum attainable speed of operation. A delay fault is registered in the circuit when one or more path delay is more than the clock period of the circuit.

1.1.2 Path Sensitization

"A path is said to be testable if a rising/falling transition can propagate from the primary input to the primary output associated with the path, under certain sensitization criteria" [11] [12] [13] [14][1]. "If a path is not testable, it is called an untestable or false path" [15] [16][1]. In case of static sensitization of paths, all the side inputs of the gates for the path under test should have non-controlling values [2]. Figure 2 below shows a false path a-c-d which cannot be sensitized because b needs to have a non-controlling value of 1 for the AND gate and a non-controlling value of 0 for the OR gate in order to propagate transition along path a-c-d [1].

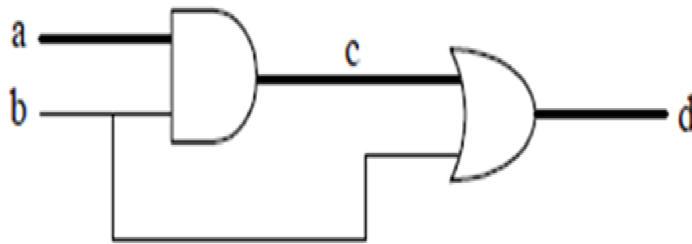

Figure 2. Untestable path [1]

1.1.3 Robust and Non-Robust Path Delay Tests

Depending on the sensitization criteria, a path can be robustly testable or non-robustly testable [1]. A robust test will detect a path delay fault irrespective of other delays in the circuit. However a non-robust test will detect path delay fault if no other path delay fault is present. For the non-robust test, condition of static sensitization

should be satisfied along with the condition that the test vector pair will produce the required transition at the start of path under test [2].

1.2 Scan-based Delay Test

To test a circuit, several scan flops are inserted into the design for observability and controllability of the circuit under test. This scan flops are then connected into a scan-chain. The circuit can be operated in normal functional mode or in scan mode. In the functional mode, the output will be the functional output. The scan flops will have no role to play in this case. For testing purposes, there are two distinct operations, first is to load the test vector and the second is to capture the response. First the scan-mode is enabled, and the test vector is shifted to the scan register. Then test mode is turned on, in which the combinational block gets the previously loaded values from the scan flops. In the next clock cycle, the output scan-flops capture responses from combinational block and the design is set to scan mode. The results are then shifted out of scan-chain to be compared against expected responses. Figure 3 explains the scan-chain operation.

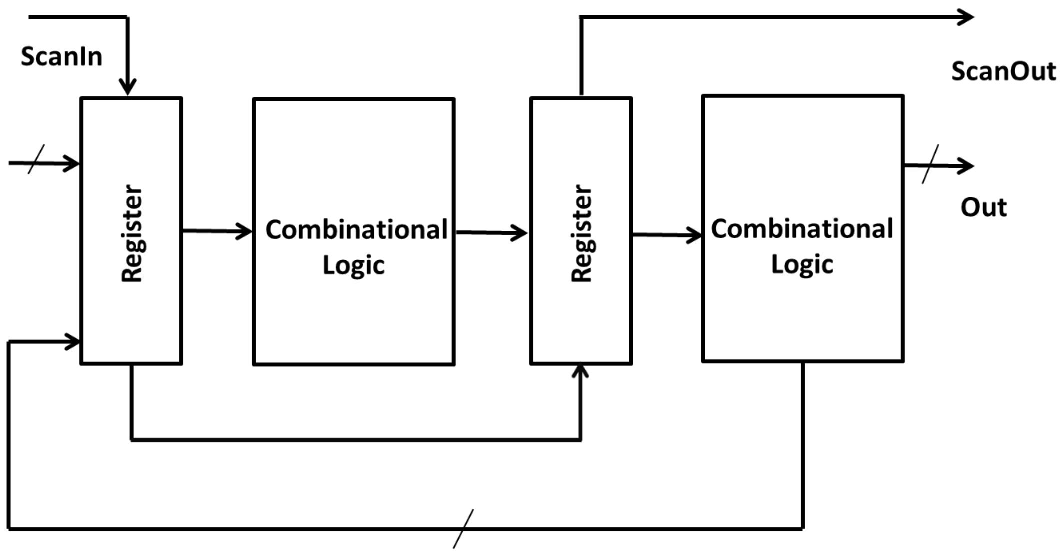

Figure 3. Scan-based test [17]

Since the flip-flop can hold a single value, to make them apply two patterns there are two common approaches, one is muxed-D scan and the other is enhanced scan [1].

1.2.1 Muxed-D Scan Approach

Muxed-D scan approach utilizes a muxed-D scan cell. Muxed-D scan cell, shown in Figure 4, has a 2:1 multiplexer at the input of a D flip-flop. The select input of the multiplexer is a scan-enable (SE) signal which selects between the functional data (DI) and scan-input (SI).

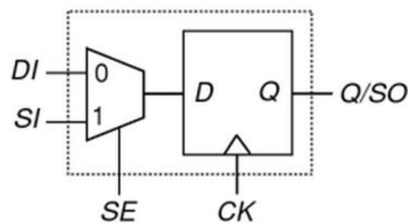

Figure 4. Muxed-D scan cell [1]

From Figure 5 we can see that since the outputs of a muxed-D scan-cell is connected to the input of next muxed-D scan-cell, when SE is 1, they function as a single scan-chain. SE value of 0 is used to capture responses from the combinational logic into the flops.

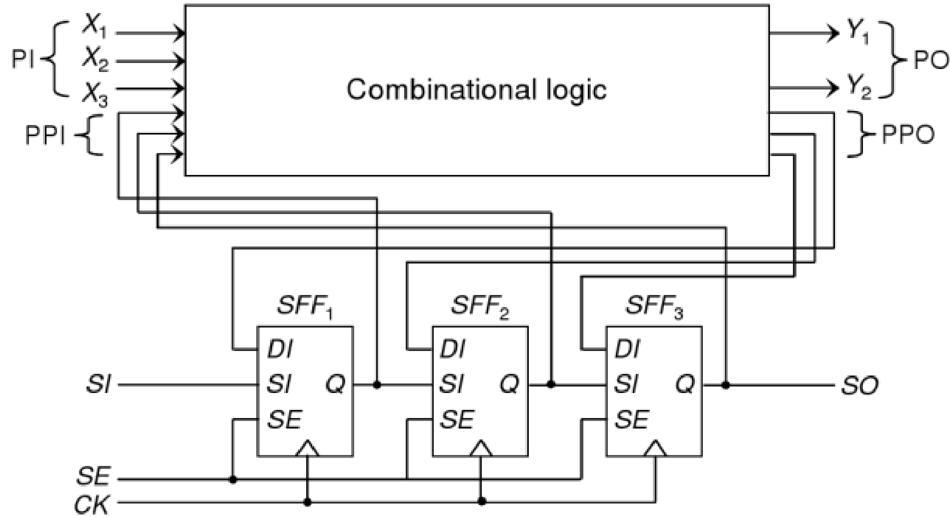

Figure 5. Muxed-D scan design [1]

1.2.2 Enhanced Scan Approach

In enhanced scan design, we can apply an arbitrary pair of vectors. In the design as shown in Figure 6, when the UPDATE signal is 1, the first vector applied to scan-flops (SFF) is transferred from the scan-flops to the latches (LA). Next UPDATE signal is set to 0, and the second vector is loaded into the scan-flops. Once the vector is loaded, UPDATE is made 1 again, and the output response is captured at the scan-cells. Enhanced scan approach has high delay fault coverage. The hardware overhead is the downside of enhanced scan.

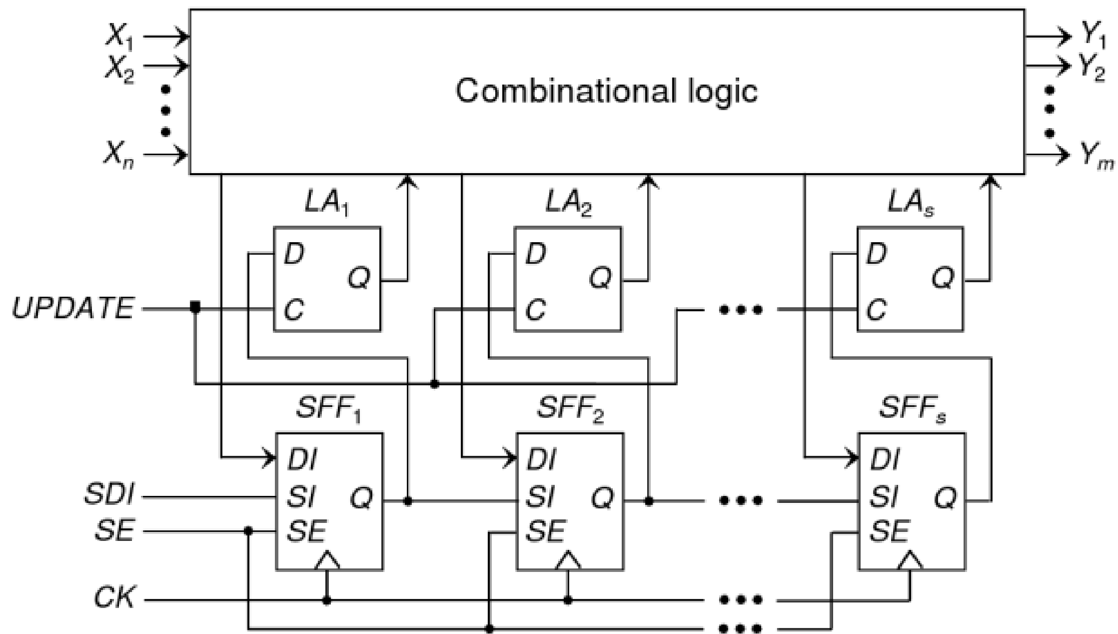

Figure 6. Enhanced-scan design [1]

1.3 At Speed Testing

The scan-design provides at-speed testing for high speed and high frequency circuits [1]. Launch On Shift (LOS) and Launch On Capture (LOC) are two at-speed test schemes [1]. To detect transition fault or path delay fault in intra-clock domain or inter-clock domain, either of the two could be used.

1.3.1 Launch on Shift

In Launch on Shift (LOS) approach as shown in Figure 7, the last shift clock pulse is used to launch transition and capture clock pulse is used to capture the response. In this approach, the scan enable signal switches its value between the launch and capture clock pulse [1].

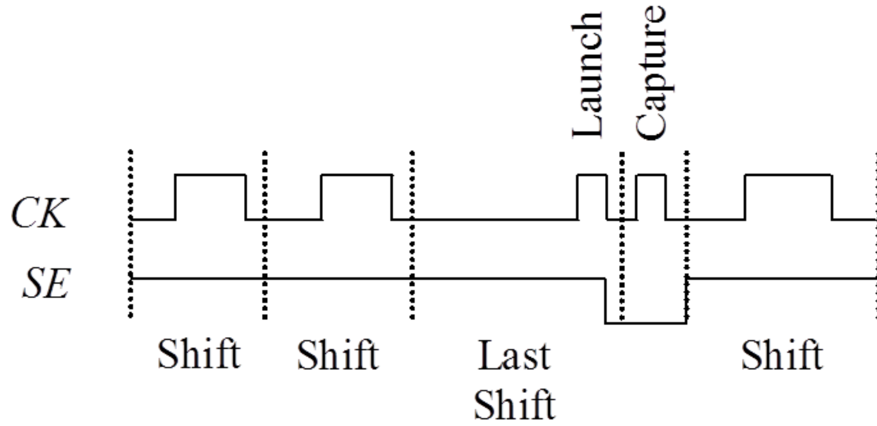

Figure 7. Launch on Shift [1]

1.3.2 Launch on Capture

The traditional structural test in high-speed circuits shifts the test pattern slowly into the flip-flops organized as scan chains, during which time the scan enable (SE) signal is held up. The SE signal is then switched low, so that the circuit is in functional mode, and applies two at-speed cycles to launch and capture the test results. This is referred to as launch-on-capture (LOC) test (Figure 8).

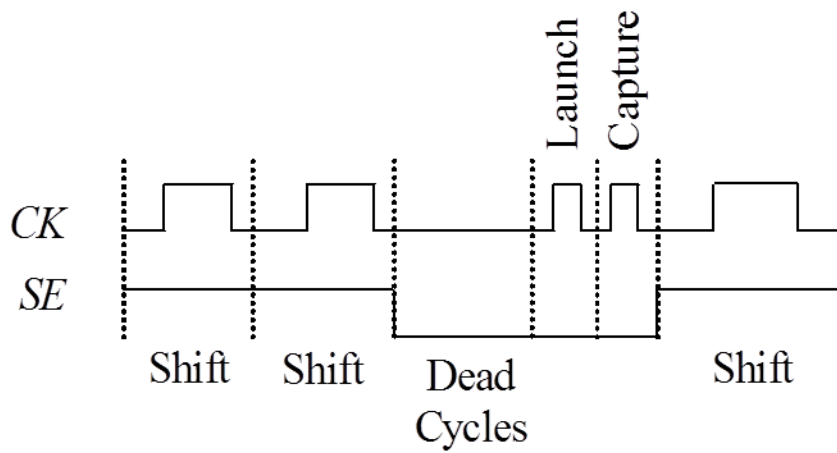

Figure 8. Launch on Capture [1]

In LOC approach the test vector is to be justified back by one time-frame as shown in Figure 9. The initialization vector V1 is generated first, then the next vector V2 is generated such that a transition can be launched.V2 is a function of the vector V1′, where V1 and V1′ are same except that they are shifted by one time-frame. The assignments in V1 and V1′ should not be conflicting [18].

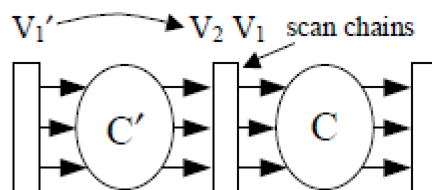

Figure 9. Justification in LOC [18]

1.4 KLPG Algorithm

The KLPG algorithm [10] aims at generating K longest paths through each gate in a combinational circuit. The paths start at primary inputs and ends at primary outputs. Paper [18] describes the KLPG algorithm for scan-based sequential circuits. For the sequential circuits the launch point is a scan-flop and the path is grown until it reaches a capture point which is another scan-flop. The paths that have been generated are subjected to a final justification phase. The KLPG algorithm has been implemented in CodGen.

The flowchart of the algorithm is given in Figure 10.

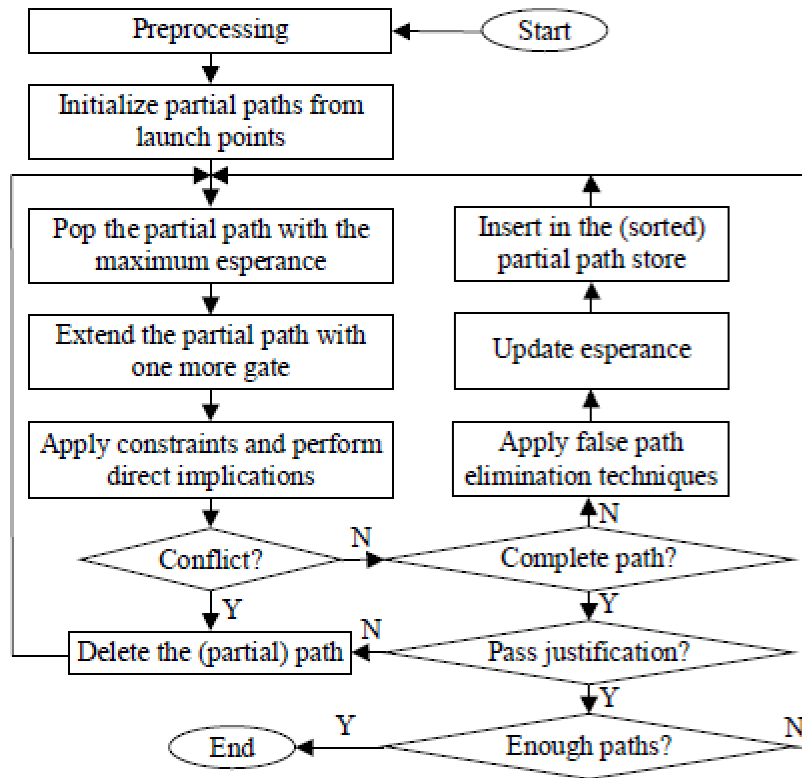

Figure 10. KLPG algorithm [18]

The three main steps of the KLPG algorithm are path initialization, path growth and path justification. Given a circuit, we need to find out the sequential observability and controllability values of the gates in the design. The observability value determines how easily we can observe the outputs of the gates and controllability is a measure of how easily we can control the input values. The gates near to the primary outputs are more observable than those near the primary inputs, whereas gates near the primary inputs are more controllable than those near to primary outputs. So in order to compute observability and controllability values, we need to levelize the circuit, which will give the maximum distance of the gate from a primary input. We also need to compute esperance of the gates. Thus observability, controllability, esperance, fan-in and fan-out cones of each gate are calculated in initialization phase.

During the path growth stage, each gate is added to the pre-existing partial paths if it meets the sensitization criteria. These partial paths are all saved and stored in the partial path store sorted according to esperance value. The esperance value is the upper bound limit on the delay when the partial path grows to a complete path. During this stage, direct implications are also performed to get the outputs of other gates.

When a partial path reaches a scan-cell, it becomes a complete path and final justification is performed on it to check whether all the assigned values are compatible. Test patterns are obtained in this final step.

After the paths are generated and test vectors found, the test vectors are compacted to reduce the number of test patterns. Compaction could be of two types: static compaction and dynamic compaction. Static compaction is performed after the test

generation. The paper [19] presents dynamic compaction implementation for KLPG algorithm. This dynamic implementation does not consider one pattern generated against the other. It saves the paths in a path pool and whenever a new path is generated, the assignments for the new path are compared against those in the path pool. By doing dynamic compaction, the pattern count had been reduced.

The coverage value gives us an idea of how many faults had been detected over total number of faults. Larger the coverage better is the test. To increase the fault coverage, top-off transition fault test patterns can be used [1].

1.4.1 Pseudo-functional KLPG

During the time that the circuit is switching from scan to functional mode, the currents in the off-chip connections fall to their quiescent values. When the at-speed cycles are applied, the current demand of the chip rises quickly, but the off-chip inductance limits the speed that current can be supplied, leading to dI/dt power supply voltage droop on the chip [20]. This causes the chip to operate more slowly than in functional mode. In Figure 11 , we could see delay test induced drop in power supply voltage. So there is a chance that the circuit will operate slowly and good chips may fail the delay test.

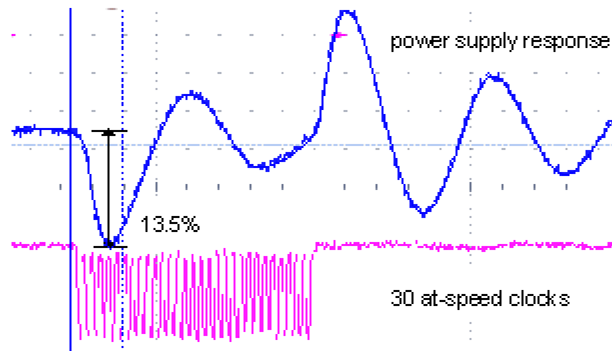

Figure 11. Drop in power supply voltage during delay test [20]

The solution is to apply a number of medium-speed preamble cycles after the test has been scanned in, before the launch and capture. Since these preambles are in sequential mode, and filter out most non-functional activity, this test in the KLPG algorithm is referred as pseudo functional KLPG test [21]. The timing diagram is shown in Figure 12.

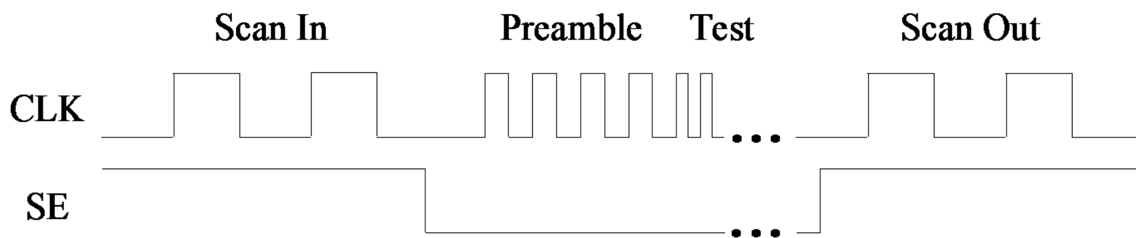

Figure 12. Pseudo functional test [21]

1.5 Boolean Satisfiability

The use of Boolean Satisfiability in generating the test patterns for the circuits under test has been shown in [22]. The Conjunctive Normal Form (CNF) generation of an AND gate is described in [22]. If Z=X.Y, then the formula can be written as

$(Z \rightarrow (X.Y))((X.Y) \rightarrow Z)$. Next all implications are transformed to disjunctions. Hence the formula for an AND gate is obtained as $(\sim Z+X)(\sim Z+Y)(\sim X+\sim Y+Z)$. In the CNF, each sum is a clause. The task is to find an assignment of X, Y and Z such that the formula evaluates to true. Clauses with two variables are said to be in 2CNF, and clauses with three variables are in 3CNF. While 2CNF can be solved in polynomial time, 3CNF is a NP-Complete problem.

The work in [22] describes how to extract the Conjunctive Normal Form (CNF) formulas for the faulted and un-faulted circuits. The XOR of the two outputs is included in the formula to account for the fact that the XOR output will be one, if the two outputs differ. Although satisfying a CNF formula (SAT) is a NP-Complete problem, most of the clauses used in the described case are binary clauses.

Applying SAT for test generation is a problem because of the difficulty to incorporate real delay values. This is avoided in [21] by using a mixed structural-functional approach, where the paths are generated with structural approach, and during path justification SAT engine is used. In [23] several techniques are presented to speed up the path generation with the SAT solvers. The techniques presented are circuit simplification, Dynamic SAT Solving (DSS), Circuit Observability Don't Cares (Cir-ODC) and Approximate Observability Don't Cares (AODC). In DSS, the structural information of the circuit is used to speed up SAT solution time. In the path delay test generation, to speed-up SAT, only the clauses affecting the concerned fan-in and fan-out cones are turned on, other clauses are turned off.

In the paper [24], authors discuss in detail about the implementation of SAT. The SAT components should determine how to represent the internal data structures, a policy on direct implication of assignments and the way search for the assignments is to be done to satisfy the solver. For assignments to the literals, first either a true or false value is assigned to that literal, and each of the clauses are evaluated for that assignment. Conflicts in literal assignments are resolved by backtracking some of the assignments and the search should be continued again to find satisfying assignments. Whenever some conflict is detected, the clause is added to the learnt clause set, which will be used in future decision making process. But care should be taken to see that this learnt clause set does not become too big, as time can be wasted on searching a big learnt clause set. So the SAT solvers generally prune these clauses.

Cir-ODC is described in detail in [25]. If there is a signal which does not have any effect in the output of the design with certain logic constraints, then those logic constraints are don't-care condition related to the signal. The use of Cir-ODC also helps to speed-up delay test generation. For optimization, it is necessary to find compatible ODCs. But generation of compatible ODCs is complex. An efficient algorithm to find approximate ODCs is presented in [26].

1.5.1 MiniSat

MiniSat is a minimalistic, open-source SAT solver [27]. It has been used in CodGen because of its modifiability, efficiency and ease of integration.

1.5.2 Use of SAT in CodGen

The implementation of SAT in CodGen has been described in [21]. The delay test requires two vectors for the launch and capture. So for launch on capture, for a signal in the circuit, two Boolean variables are used to represent the signal in two time-frames. Similarly for the pseudo-functional test, the signal has to be represented in more than two time-frames. If the primary inputs are fixed, only one Boolean variable can be used for all the time frames for the primary inputs. Several features, such as dynamic SAT solving are present in CodGen [23].

1.6 Structure of the Thesis

In this thesis, we propose a multiple at-speed cycle KLPG algorithm which will be useful for testing metastability of flops and time-borrowing of latches. The thesis is organized as follows: In Chapter 2, we present the motivation of the work. In Chapter 3, we present the implementation strategy for the multiple at-speed capture cycle KLPG algorithm. Chapter 4 discusses the results, and Chapter 5 concludes the research.

2.1 Time Borrowing and Time Stealing

In high-speed CMOS circuits, some paths may take longer than one clock cycle to propagate. This is enabled by time borrowing and time stealing in latch-based designs [28]. Time borrowing happens in "the case where a logical partition utilizes time left over (slack time)" by a partition in the previous clock cycle [28]. Time stealing occurs where "a logical partition utilizes a portion of the time allotted to" the partition in the next cycle [28].

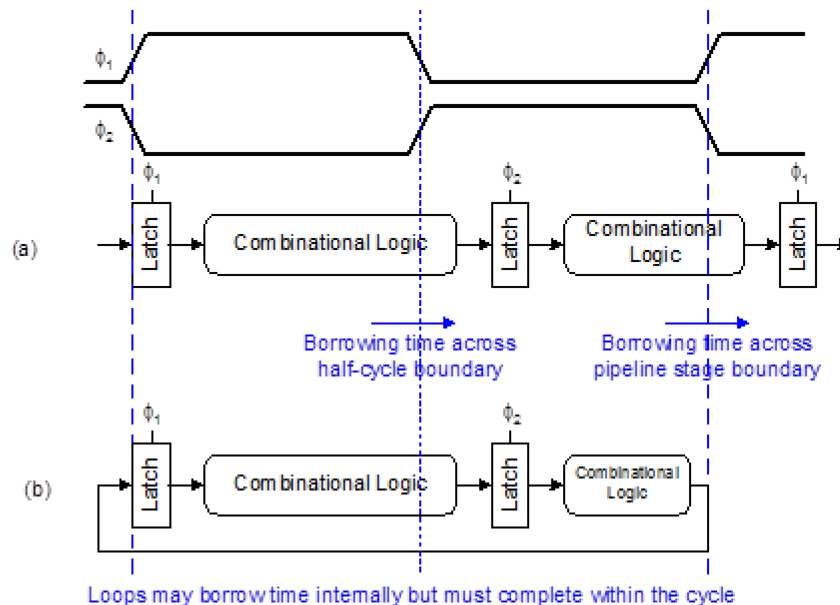

Figure 13. Time borrowing [29]

Figure 13 shows the concept of time borrowing. Figure 13 (a) shows borrowing time across the pipeline stage boundary and Figure 13 (b) shows borrowing time across half-cycle boundary.

Time borrowing in a design can be either intentional or unintentional [30]. For those cases with intentional time borrowing, the design for test is already taken care of. But even in this case, the actual time-borrowing will vary depending on the test vectors and the chip under test. The unintentional cases of time-borrowing latches in a chip arise from variations in fabrication process. Because of this, signals may not arrive at the inputs of gate at the required time. This causes problem for the test generation. To test time-borrowing latches, multiple paths that start and end at a latch have to be concatenated [30].

So time borrowing allows one partition to use more time than is available at the cost of another partition. Hence, time borrowing paths become multi-cycle paths. Traditional structural delay tests do not test whether time borrowing or stealing is working correctly, since only a single at-speed cycle is tested. To detect path delay faults for the multi-cycle paths, it is necessary to start a path at a register and end at a register while passing through another register, testing the longest paths between each pair of registers. This requires three or more at-speed cycles, rather than the two of traditional LOC test.

2.2 Metastability

The arrival time of a signal is the time at which the signal arrives at an endpoint. The required time is the time before which the signal can arrive without affecting the clock period. The difference between arrival and required time is known as the timing slack. If timing slack is positive, it means that the signal can still arrive later. If the slack is negative, the arrival time needs to be improved.

When a path is late arriving at a flip-flop, the input signal may change at the same time that the flip-flop is clocked, violating the flip-flop's setup time. This may cause the flip-flop to enter a metastable state, before it eventually resolves to a 0 or 1. This causes the flip-flops output transition to be slow, which could cause a delay fault in the following at-speed cycle. Current delay tests do not detect this situation. Using three or more at-speed cycles will detect this case.

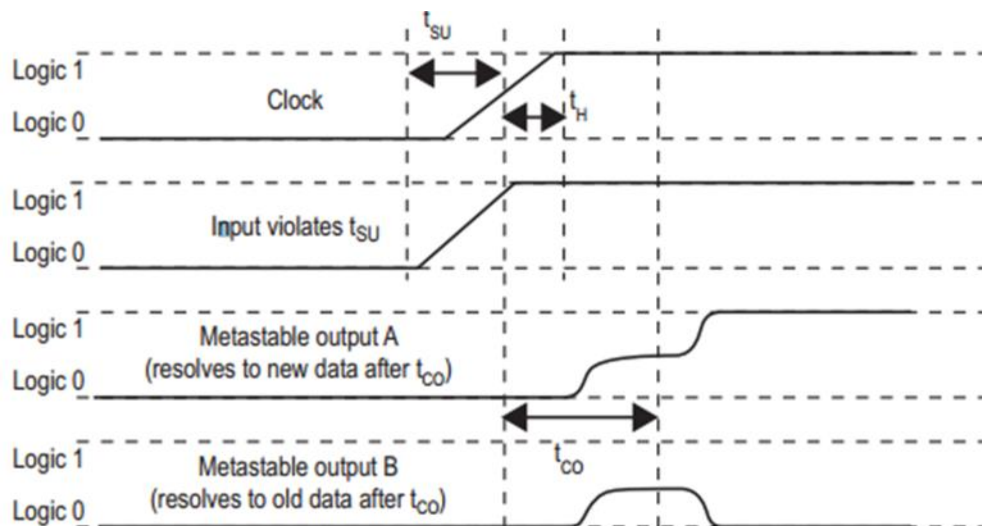

Figure 14. Metastability [31]

In Figure 14, we see that set-up is violated. The metastable output A finally settles to logic 1 after the clock to output time (tco) and metastable output B settles to logic 0 after tco.

2.3 At Speed Test Approach

Since scan launch and capture clocks are much slower than the functional clocks, scanning patterns in and out of the circuit is very slow which make the tests take a long amount of time. To make the tests faster, several at speed capture cycles can be used. This is done in this research and we term the test at-speed test. The timing diagram is shown in Figure 15, with the functional test cycles being preceded by the preamble cycles.

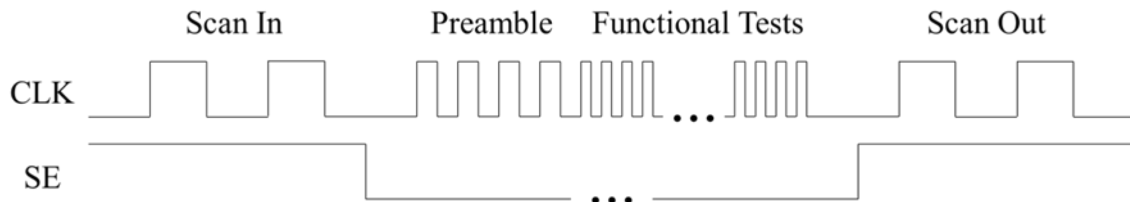

Figure 15. At speed functional test

The challenge for the at-speed test is that we have to justify the paths and the assignments over multiple time-frames. So we first find a longest path starting and ending at a flop in the first launch and capture cycle, in the next capture cycle we have to start finding a long path from the scan-out of the flop where the previous path ended,

while obeying the necessary assignments for the previous clock cycle. This constraints the path search, and as we have seen in our experimental results, the number of paths that we found in successive capture cycle decreases.

The three advantages of at-speed approach are:

- It can be used to test time-borrowing in case of latches.
- It can test metastability of flip-flops because a late arriving transition can result in a metastable flip-flop and the resulting output violation will be captured in next capture cycles.
- It can further help to filter out power-supply noise because of the extra functional cycles.

One of the decisions that had to be taken regarding path finding is when to do the final justification. One of the approaches could be to do the final justification after we have found the paths over a certain number of capture cycles, or we could justify the path after each capture cycle. We have used the second approach in this research, as by justifying after each capture cycle we could save time by eliminating paths upfront and eliminate the effort of expanding them.

2.4 Faster than At-Speed Delay Test

There are some delay test methods which selects shortest paths through a fault site and the paths can have timing slack [1]. Faster than at-speed testing approach can be used when the test patterns are generated for a path with timing slack [1]. But the drawbacks of this approach are the additional power supply noise generated to operate at

frequency higher than at-speed, and some good chips could be rejected because they fail at faster than at-speed test [1].

2.5 Other Related Work

The work in [32] proposes a metric which gives testability of the circuit for path delay faults. They have used this measure in scan-based Built-in Self-Test (BIST) path delay testing.

The work in [30] [33] considers the case of time-borrowing in latches in high speed circuits and proposes structural delay testing in those cases. It targets long paths in successive blocks for delay testing.

The work in [34] discusses about canary logic. The canary flop helps to prevent timing error in the design. Comparison of the values in main flip-flop and the canary flip-flop determines the correctness of operation. It is an alternative to Razor [35] logic. The use of canary logic in preventing age-related timing violation of the circuit has been shown in [36].

3. IMPLEMENTATION

3.1 Test Generation Strategy for Enhanced CodGen with Multiple Capture Cycles

The KLPG algorithm which has been implemented in CodGen assumed only one at-speed capture cycle, so only generates timed paths for one launch and one capture cycle. We have extended the KLPG algorithm to generate longest paths across multiple at-speed capture cycles. First, a longest path between a launch and capture flip-flop (or latch) is found by KLPG. We then justify this path. Then we extend this path from the capture flip-flop to the next capture flip-flop. This process continues for as many at-speed cycles as desired. In previous implementations of CodGen, time frame expansion only goes back in time, assuming that the at-speed path is captured at the last cycle. In this work we must go back in time for justification, but also forward in time as we add more at-speed clock cycles. These forward expansions are done using the necessary assignments of the prior frames, in order to trim off sequentially false paths. For each additional cycle, we justify the path, in order to avoid wasting time in generating sequentially false paths. The search space for the path finding across multiple at-speed cycles is shown in Figure 16, assuming that we start from a flip-flop (the “fault site”).

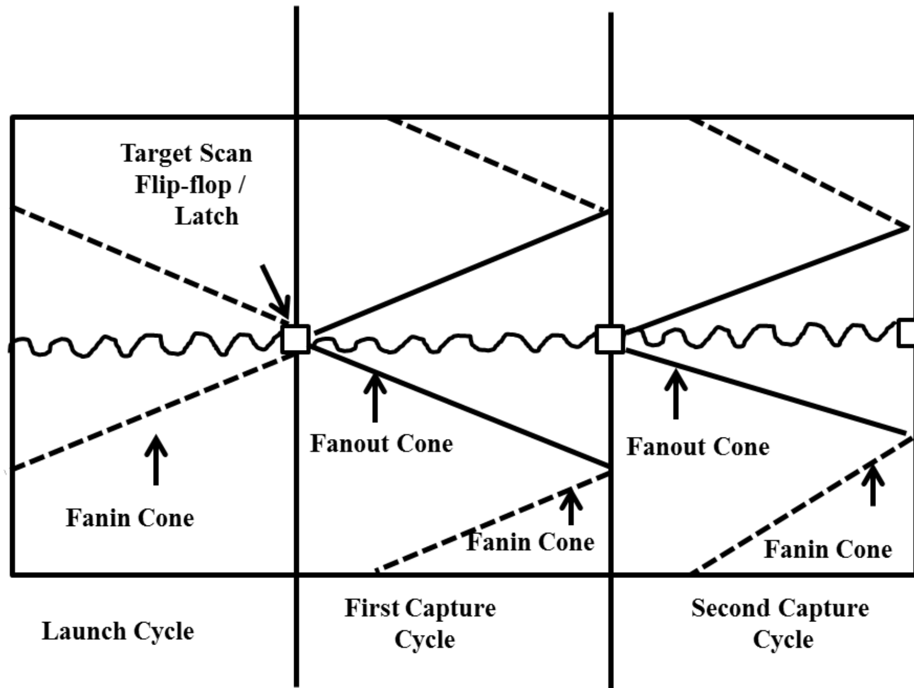

Figure 16. Multiple capture cycles in at-speed path delay test

The steps in the enhanced KLPG algorithm are explained in detail below.

Path Generation in launch and capture cycle: In our work, KLPG algorithm has been modified to generate K longest paths aiming to generate K longest paths per flop through each fan-out. During initialization step of KLPG algorithm [10], the metrics for observability, controllability, esperance as well as the fan-in and fan-out cone for the starting gate is calculated. The starting gate is a scan-cell which will be stored in a partial path structure. This partial path will grow to become a complete path. All such partial paths will be stored in a path store. We extend a partial path with the maximum esperance by adding one more gate to it. We perform direct implication on the outputs and side inputs of the newly added gate. If direct implication is passed, the gate is added to extend the partial path and we assign values to the time-frames. When a partial path

reaches another scan-cell, the path is a complete path. This complete path is then justified with SAT.

Path extension in next capture cycles: Once this path passes justification, we have to again start extending the path from that capture flop. The necessary assignments for this step will move forward by one time-frame. Once again we need to find the gates in the fan-out cone of this scan-cell. Then we extend the partial path by adding one more gate to it, in a similar manner as before until it reaches another scan-cell. This completes the path in the second capture cycle. The path is then justified. This process is repeated for additional capture cycles as needed.

Final Justification and Compaction: If a path passes final justification, then that path is reported and the necessary test pattern is generated, and compacted. Compaction could be either static or dynamic.

The flowchart for the enhanced CodGen with multiple capture cycles is shown in Figure 17.

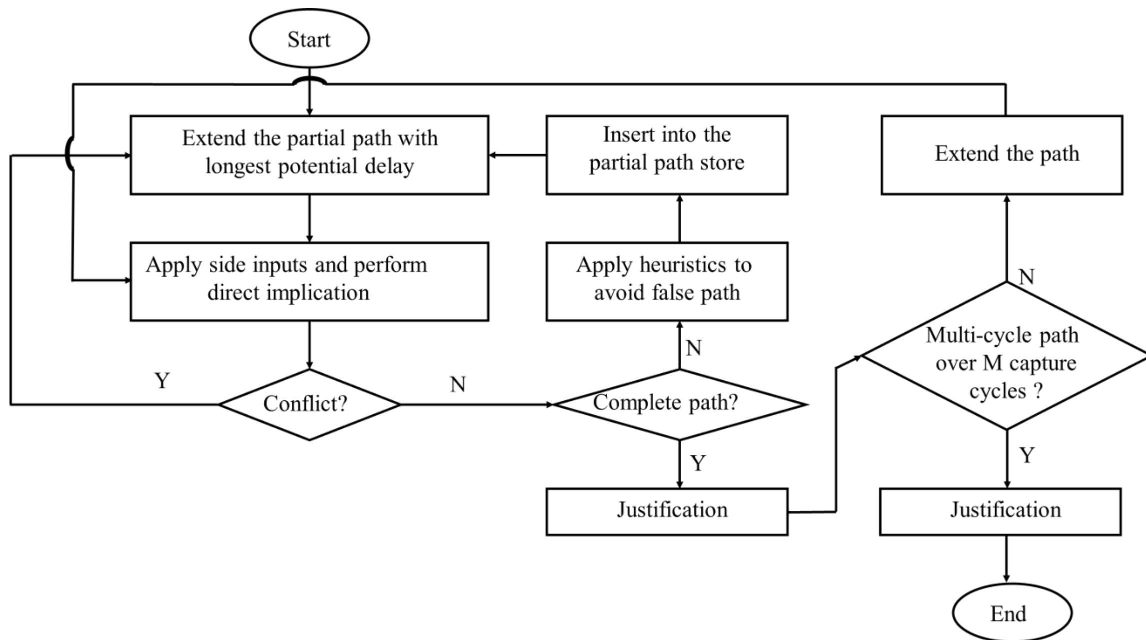

Figure 17. Flow-chart for CodGen with multiple capture cycles

3.2 Aim at Scan-flops

In order to generate K longest paths per gate previous versions of CodGen were targeting each of the gates. But in our work, since we are concerned about generating paths across multiple flops in order to determine metastability, we aim to generate K longest paths through each flop for each fan-out. Hence in our work, we have a stricter bound on the number of paths generated as we target only the flops. Figure 18 shows a scan flip-flop with its fan-outs.

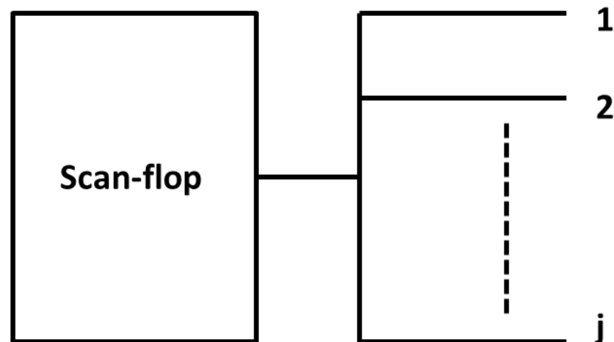

Figure 18. Scan-flop with fan-outs

If a scan-flop has j fan-outs, the number of maximum possible paths for this scan-flop is twice of K*j, to account for rising and falling transition at each fan-out. It may be noted that while targeting scan-flop SFF1, when we find a path from scan-flop SFF1 to another scan-flop SFF2 and again to another scan-flop SFF3, we increase the counter for the rise or fall paths generated only for the target scan-flop SFF1.

3.3 Finding Fan-in and Fan-out Cone

In the previous implementations with a launch and capture cycle, path generation would start at a launch point which is a scan-flop and would stop on reaching another scan-flop. So fan-in and fan-out cones are computed for the target scan flop. The path which reaches a scan-flop is a complete path.

In our implementation, since we are generating paths over multiple capture cycles, we pass across flops. When the first complete path gets generated for the launch and first capture cycle, we have reached a scan-flop. Since we will be extending the path from this capture flop, we would need to compute the fan-in and fan-out cones of this

scan-flop. The gates in the fan-in and fan-out cone of this scan-flop will be marked appropriately and other gates would be marked blocked to facilitate path extension for the next capture cycle.

3.4 Complete Path over Multiple Capture Cycles

A path starting from a scan-flop becomes a complete path on reaching another scan-flop. This requires a launch and a capture cycle. In our work, since we are concerned with finding path over multiple capture cycles, we will continue our path finding over multiple capture cycles and hence the paths that we find have more than two scan-flops for more than one capture cycle (Figure 19).

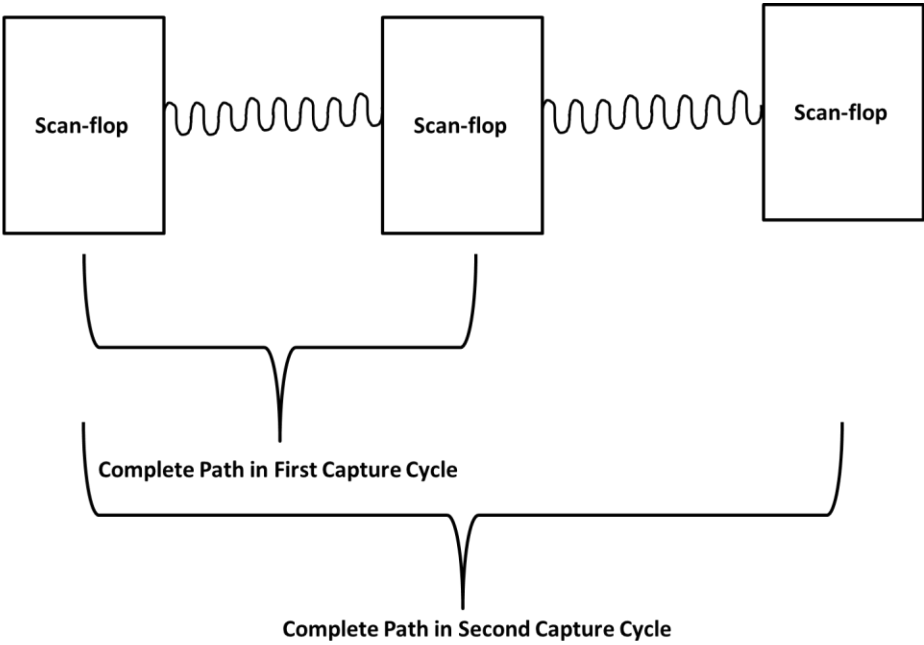

Figure 19. Complete path over multiple capture cycles

3.5 Tracking the Number of Paths Generated

There is a maximum limit for which a path can be extended through a particular fan-out of the target gate. After that limit is crossed, the partial path pool is emptied. Similarly, the partial path pool is emptied when K longest paths for both rising and falling transition had been found for a particular fan-out of the target gate. In order to improve runtime for path generation across multiple capture cycles, whenever we expand a partial path with a rising transition for a particular fan-out of the target flop, we check whether K longest paths with rising transition from that fan-out of the target flop has been generated or not. If it had already been generated, we discard the growth of the partial path into a complete path and remove that path from the path pool. However if the limit on K longest paths with falling transition had not been reached with that fan-out, we keep on expanding the partial paths with falling transition at that fan-out of the target flop. The process is similar when we reach the K longest path check limit for the falling transition at a fan-out of the target flop but the limit for the rising transition has not been reached.

3.6 Time-frame Expansion

The sequential circuit is unrolled in time to apply the ATPG procedures available for the combinational circuits to them. This is known as time-frame expansion [2]. In the unrolled model, the combinational circuit is used twice, one for a current clock cycle, and the other for a previous clock cycle.

In the implementation of CodGen, first a rising or falling transition is launched from a flop. So if we number the time-frames as 0 and 1, for a rising transition frame 0

would have a value of 0 and frame 1 would have a value of 1. For a falling transition, frame 0 is assigned 1 and frame 1 is assigned 0. Direct implications are then performed in the fan-out cones of this gate. If direct implication fails, then we would not be able to assign those values in the flop from where our path begins. Direct implication could fail whenever we cannot assign value in any of the time-frames. We could also have some don't cares in time-frames of certain gates. As we expand the path, by adding more gates to the path, we continue to assign values to the time-frames. When a complete path is achieved, final justification is performed.

For the at-speed cycles, apart from first launch and capture cycle, other capture cycles are present as well. So for the launch and capture cycle we have to assign values to the frames as discussed, then from the next capture cycle onwards, we have to note that we can assign values to the next frame by having constraints in place for the previous frames. For simplicity in justification, whenever we move forward by one capture cycle, justification is performed with the previous capture cycle. The assignments in each successive frame become more constrained. We justify the assignments already in the frames before we assign values for next capture cycles. This is done to avoid extra time in expanding paths which would ultimately fail later on.

4. RESULTS

The modified KLPG algorithm for different at-speed cycles has been implemented in Visual C++ on an Intel Core i7 machine. Experiments have been performed on the ISCAS 89 Sequential Benchmark Circuits. Both robust and non-robust cases have been considered to generate the test patterns. The primary inputs to the circuits have a fixed value which will be useful for a low-cost tester. In our experiments there are no preamble cycles and K is 1.

4.1 Paths Generated across Different Capture Cycles for Robust and Non-Robust Case

We have run the at-speed KLPG algorithm for different number of at-speed capture cycles, and tabulated the number of multi-cycle paths generated and time (h:m:s) to generate the paths. The longest path length generated at the capture cycles have been tabulated as well as the breakup in the lengths of the path in each of the cycles. The experimental results obtained for the test generation for the robust case is given in Table 1.

Table 1. # Paths Generated, Longest Path Length, Time for Robust Case, K=1

Circuit	First Capture Cycle			Second Capture Cycle			Third Capture Cycle		
	# Paths	Longest Path Length	Time	# Paths	Longest Path Length	Time	# Paths	Longest Path Length	Time
s1488	91	16	0:03	74	28=14+14	0:15	46	37=11+13+13	0:31
s1494	105	16	0:03	78	28=14+14	0:14	41	33=11+8+14	0:35
s5378	535	19	0:36	274	32=18+14	1:09	258	34=15+14+5	1:31
s9234	504	51	2:16	249	58=40+18	7:31	163	97=43+11+43	9:00
s13207	1028	50	3:22	321	81=48+33	7:21	122	110=50+31+29	9:51
s15850	1224	58	7:02	543	67=29+38	1:23:58	201	77=17+35+25	1:40:36
s38417	3624	41	26:58	1294	59=30+29	6:38:44	662	76=30+25+21	5:06:56

Table 1 captures the number of paths, longest path length and time to generate the paths for three capture cycles for robust test generation for the circuits. As we can see, the number of paths generated decreases as we increase the number of capture cycles for all of the circuits. This observation is illustrated with the help of a bar-chart (Figure 20).

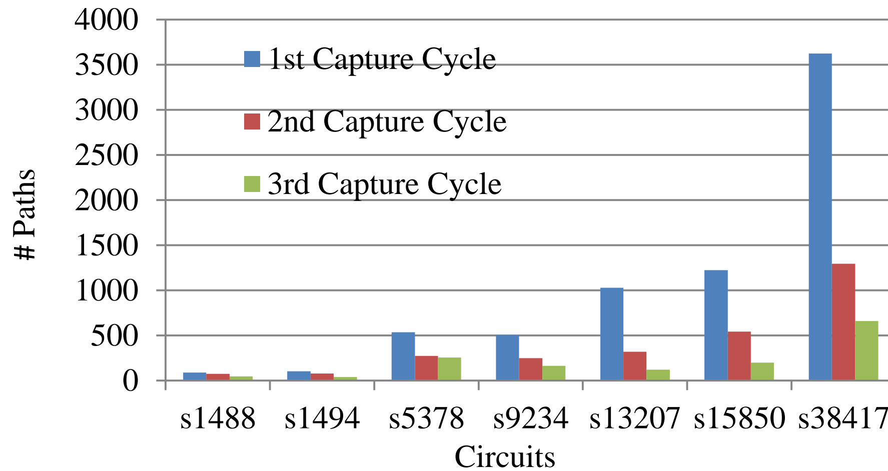

Figure 20. # Paths in circuits for different capture cycles for robust case

The time to generate the paths increases with each additional cycle as we have more constraints in expanding the path. For the smaller circuits the time to generate tests for third capture cycle is double that of second capture cycle. But as the circuit size grows, the time to generate these paths does not increase much from the second capture cycle to the third capture cycle. This is because the number of paths that we will consider expanding from the second capture cycle onwards has already been reduced from the first capture cycle. Another reason is, the try limit to generate paths through fan-out of the target gate reaches quickly for the third capture cycle, and some paths are aborted before extending.

As we can observe from Table 1, with increasing capture cycles, the longest path length increases as expected. In the columns for the longest path length, we have shown the breakup of the lengths of the path at each capture cycle. The longest path length found has been reported as $x = x1+x2+x3$, where xi gives the path length found at i^{th} capture cycle. The path length $x1$ gives the longest path that has been generated from the

target scan-flop. The subsequent path lengths are obtained by growing this path by trying to add more gates to this longest path until it reaches another scan-flop. The longest path length obtained in different capture cycles for the different circuits is shown in Figure 21.

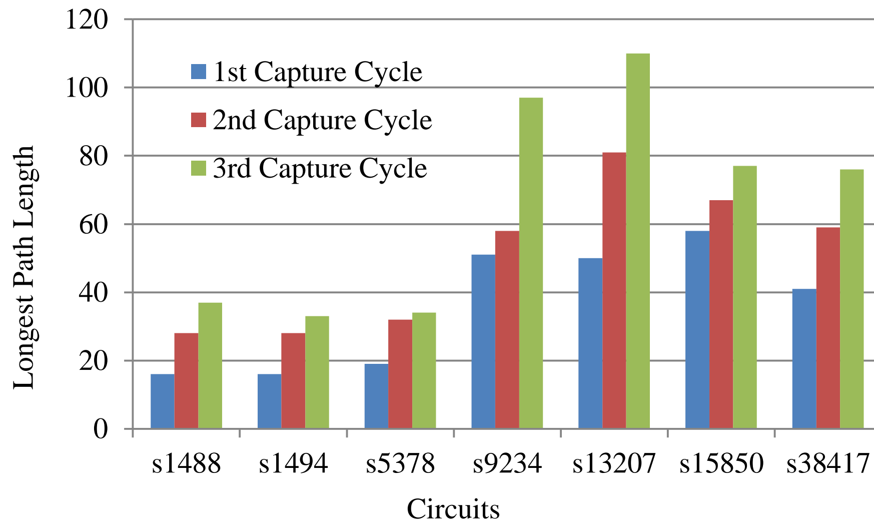

Figure 21. Longest path length in different capture cycles for different circuits for robust case

In Table 1, we could further observe that for circuit s9234, longest path length in second capture cycle is 58=40+18, and in third capture cycle it is 97=43+11+43. This implies that the longest length path found in third capture cycle has path length 54 till the second capture cycle. So this means that either we have not been able to extend the longest path of length 58 for the next capture cycle or even if we could have extended the path of length 58 from the second capture cycle, it is still not the longest path in the third capture cycle.

Table 2 presents the experimental results for the non-robust test generation procedure.

Table 2. # Paths Generated, Longest Path Length, Time for Non-Robust Case, K=1

Circuit	First Capture Cycle			Second Capture Cycle			Third Capture Cycle		
	# Paths	Longest Path Length	Time	# Paths	Longest Path Length	Time	# Paths	Longest Path Length	Time
s1488	226	16	0:06	116	27=16+11	0:21	72	38=14+11+13	0:53
s1494	227	16	0:07	115	27=16+11	0:18	82	34=8+13+13	1:08
s5378	546	23	0:32	281	26=14+12	1:23	264	34=23+2+9	1:32
s9234	865	52	2:29	456	65=52+13	12:12	200	78=44+11+23	20:31
s13207	1751	59	5:21	507	87=47+40	12:47	130	117=47+37+33	26:49
s15850	2304	61	9:55	794	84=47+37	2:21:07	150	74=27+10+37	2:21:08
s38417	4982	41	35:09	2227	62=31+31	6:01:17	1051	86=33+27+26	10:33:53

We observe from Table 2 that the number of paths generated decreases as we have more capture cycles. In the third capture cycle, the number of paths that are found is less than that in the second capture cycle. This is because path extensions are aborted more in the third capture cycles due to try limits being reached. The bar-chart (Figure 22) below shows the number of paths found in different capture cycles across the circuits.

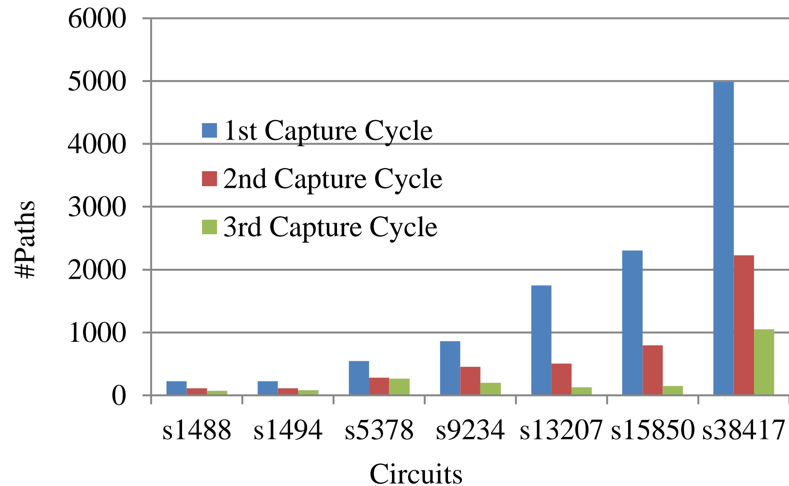

Figure 22. # Paths in circuits for different capture cycles for non-robust case

From Figure 22 we observe that we find less than half of the paths in second capture cycle compared to the first capture cycle. This is especially true for the bigger circuits. The decrease in the count of the paths is even more from the second capture cycle to the third capture cycle in the bigger circuits compared to the smaller ones.

If we analyze the time needed to generate paths for the different capture cycles in non-robust cases, we observe as before that time to generate the paths increases as we have more capture cycles. However, as observed before, the increase in time for path generation from second capture cycle to third capture cycle is not much for the bigger circuits, due to the limits being reached in our trial for path extension and paths getting aborted. This is a reason why we get fewer paths for some circuits in the second capture cycle for the non-robust cases.

Figure 23 shows the longest path length obtained for each capture cycle for different circuits in non-robust test generation strategy.

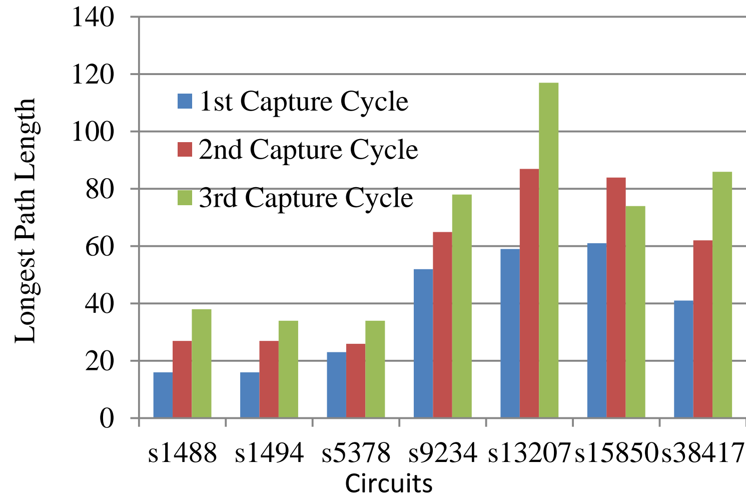

Figure 23. Longest path length in different capture cycles for different circuits in non-robust case

As seen from the Figure 23, the longest path length increases with more capture cycles, except in case of circuit s15850. The longest path length found in the third capture cycle in this case is slightly less than that found in the second capture cycle.

So in general we could conclude that as we have more capture cycles, the number of paths decreases with each capture cycle and the time to generate the path increases in both robust and non-robust case.

4.2 Fault Coverage across Different Capture Cycles for Robust and Non-Robust Case

The fault coverage result that we present here is the transition fault coverage that we have obtained by running the at-speed KLPG algorithm across different capture cycles. In the transition fault coverage study, the maximum number of possible transition

faults is twice the number of lines in the circuits. Among them, some faults are non-detectable. The fault coverage is given by the ratio of detected faults by our test generation approach and the total detectable faults in the circuit. The results for fault coverage in the different capture cycles for the robust case test generation is shown in the following table.

Table 3 contains the total number of paths generated for each capture cycle, average length of the paths found and the fault coverage for the robust case.

Table 3. Fault coverage in robust case

Circuit	First Capture Cycle			Second Capture Cycle			Third Capture Cycle		
	# Paths	Average Path Length	Fault Coverage %	# Paths	Average Path Length	Fault Coverage %	# Paths	Average Path Length	Fault Coverage %
s1488	91	10.45	33.12	74	19.1	35.95	46	26.97	30.83
s1494	105	10.19	35.57	78	19.28	36.02	41	27.46	26.13
s5378	535	11.2	51.87	274	17.99	34.98	258	22.41	33.61
s9234	504	18.57	28.4	249	28.46	21.71	163	48.61	16.95
s13207	1028	13.98	34.06	321	30.55	18.49	122	59.73	9.09
s15850	1224	19.78	37.19	543	34.42	27.27	201	43.23	13.26
s38417	3624	17.74	32.16	1294	31.54	19.49	662	45.98	10.15

As seen from Table 3, for each circuit, fault coverage generally decreases as we increase the number of capture cycles. The relation between the fault coverage percentages obtained across different capture cycles for each of the circuits for the robust case is presented in Figure 24.

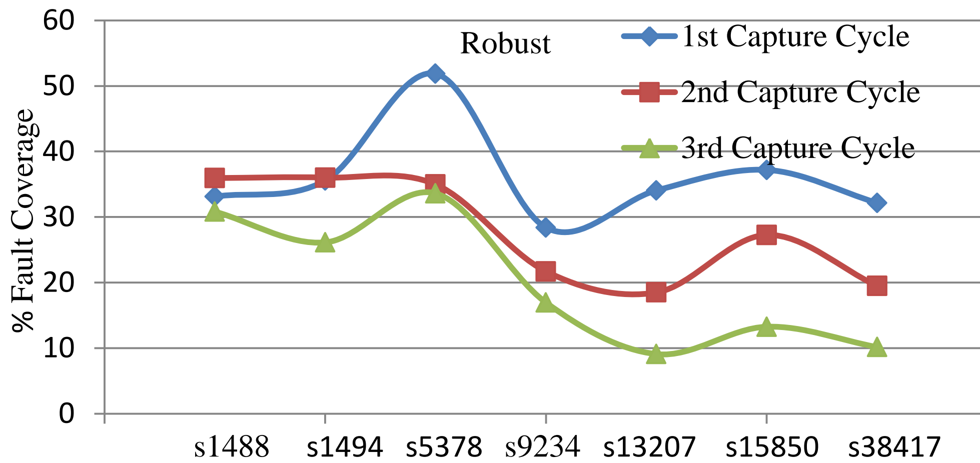

Figure 24. Fault coverage across different capture cycles in robust case

We see from Figure 24 that fault coverage is more in first capture cycle and then decreases with each capture cycle for all circuits except s1488 and s1494. The fact that fault coverage decreases with each capture cycle for the circuits is explained by the fact that we find less paths in each subsequent capture cycle. For circuits such as s1494, the fault coverage in first capture cycle is less than the coverage found in second capture cycle, although from the Table 3 we could see that the number of paths found in first capture cycle is more than the number of paths found in the second capture cycle. This is explained if we take into consideration the average length of the paths found in the different capture cycles. Figure 25 depicts the number of paths found, average length of the paths found and the fault coverage in each of the capture cycles for the circuit s1494.

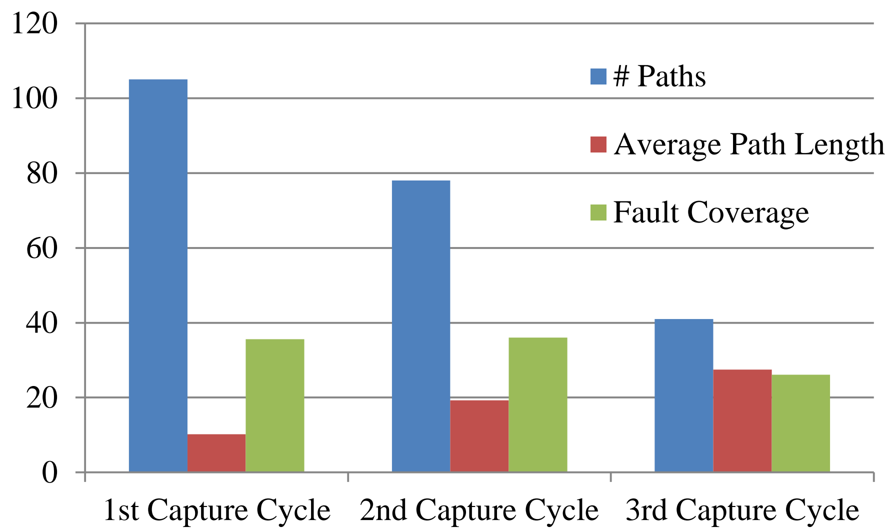

Figure 25. Fault coverage trend in circuit s1494 across different capture cycles for robust case

From Figure 25 we could see that, the reason why the fault coverage in s1494 for the second capture cycle is slightly more than in the first capture cycle despite the fact that more paths are found in first capture cycles is because, the average path length in second capture cycle is more. So basically in our at-speed test with multiple capture cycles, fault coverage at each capture cycle depends on both the number of paths found and the average length of each path.

This dependency on the average path length is the reason, why for most of the circuits although the number of paths in second capture cycle decreases to about half, the fault coverage does not decrease to such extent. However we find that for the third capture cycle, the fault coverage is always below that of second capture cycle. This is because although the average path length found in third capture cycle is always greater

than that of second capture cycle, the number of paths found in the third capture cycle is too less.

Table 4 below presents the fault coverage results for the non-robust case for the different circuits.

Table 4. Fault coverage in non-robust case

Circuits	First Capture Cycle			Second Capture Cycle			Third Capture Cycle		
	# Paths	Average Path Length	Fault Coverage %	# Paths	Average Path Length	Fault Coverage %	# Paths	Average Path Length	Fault Coverage %
s1488	226	10.19	36.33	116	19.36	46.64	72	28.3	39.08
s1494	227	10.2	35.12	115	19.2	44.78	82	28.95	40.71
s5378	546	11.69	40.73	281	17.04	34.54	264	22.23	32.99
s9234	865	15.91	22.25	456	28.61	30.48	200	47.72	16.46
s13207	1751	12.55	31.06	507	29.75	26.22	130	56.56	11.69
s15850	2304	16.42	35.94	794	33.88	30.54	150	43.75	11.04
s38417	4982	16.05	25.33	2227	31.5	29.49	1051	48.53	15.66

The trend in fault coverage in different circuits for different capture cycles for non-robust case is presented in Figure 26.

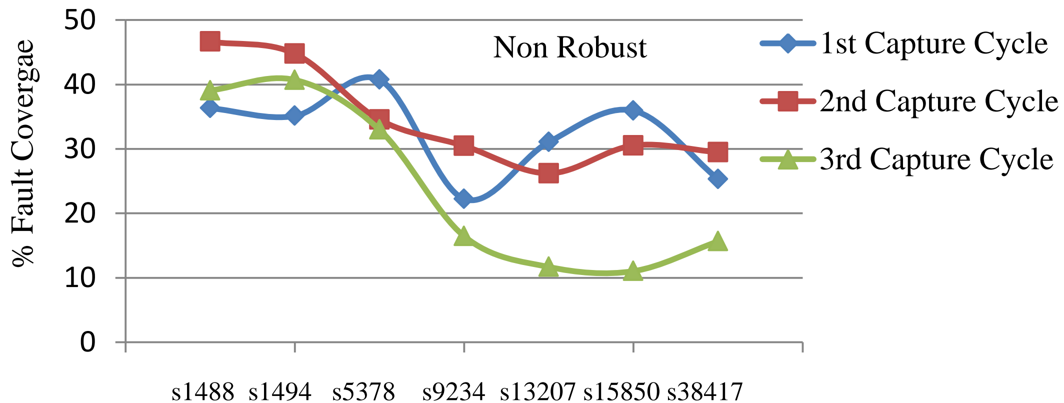

Figure 26. Fault coverage across different capture cycles in non-robust case

As before, we observe from Figure 26 that the fault coverage decreases with increase in the number of capture cycles in most of the cases in non-robust cases. In few cases the fault coverage in second capture cycle is more than the fault coverage in first capture cycle, because the average path length in second cycle is more than that in first cycle. So again in non-robust case, we observe that the fault coverage in a capture cycle depends on both the number of paths found and the average lengths of the paths. The fault coverage for the third capture cycle is too less because of the less number of paths found.

The below figure depicts the number of paths found, average length of the paths and the fault coverage for each capture cycle for the circuit s1494 in the non-robust case.

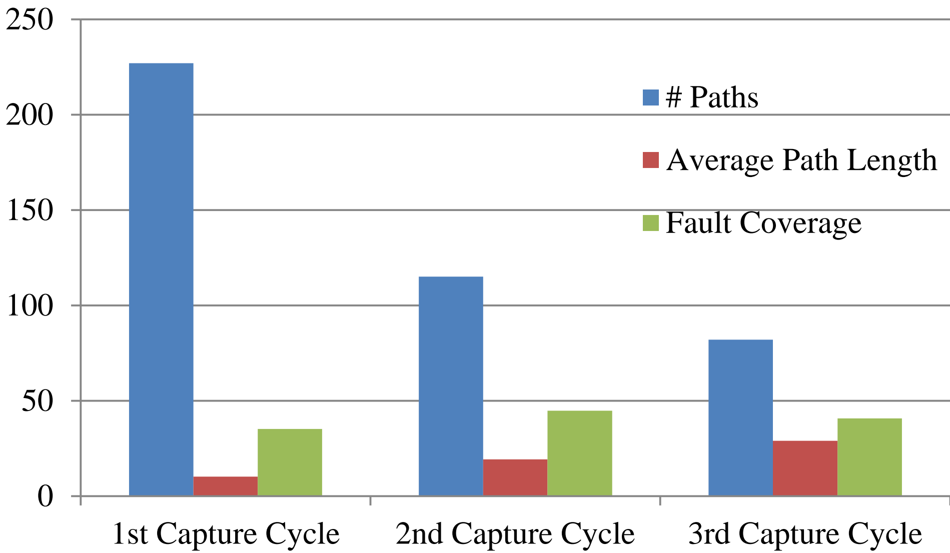

Figure 27. Fault coverage trend in circuit s1494 across different capture cycles for non-robust case

From Figure 27 we observe that for the circuit s1494, the fault coverage for second capture cycle is more than that of the first capture cycle, because the average path length in second capture cycle is more although the number of paths found is less in second capture cycle even in the non-robust case. The fault coverage in third capture cycle is also higher than the fault coverage in first capture cycle because the average path length in third capture cycle is more than that we have obtained for the first capture cycle, despite the fact that the number of paths found in third capture cycle is about one-third the number of paths found in the first capture cycle. Similarly, we observe that the fault coverage in third capture cycle is close to the value of fault coverage in second capture cycle, as the number of path being less is balanced by the average lengths of the path being greater.

So overall for the fault coverage in both robust and non-robust cases we conclude that the fault coverage generally decreases with more capture cycles as we have less number of paths with more capture cycles. Sometimes the fault coverage in second capture cycle is greater than or comparable to the fault coverage value in the first capture cycle because of the increase in average path length in second capture cycles. However the fault coverage value is poor for the third capture cycle in both robust and non-robust cases as there are very few paths generated compared to previous cycles. So ideally if the number of paths generated is comparable to that in the previous capture cycles, we could have better fault coverage in later capture cycles as the average length of the paths in later capture cycles is higher.

5. CONCLUSIONS AND FUTURE WORK

In conclusion, we can say that we have demonstrated an automatic test pattern generation approach for testing scan-flop metastability and scan-latch time-borrowing. Although the test patterns generated by the multi-cycle path approach may not have high transition fault coverage with increase in the number of capture cycles, but they will be useful in testing today's high frequency circuits. Further study can be made to analyze paths that are covered in each capture cycle. Since we have tested longest testable paths in and out of every flip-flop, and with multiple paths being tested per flip-flop due to multiple capture cycles, we have good metastability coverage.

Another interesting area of study would be to increase fault coverage obtained by allowing multiple capture cycles. The transition fault coverage results that we have obtained in our experiments shows that although the decrease in fault coverage is not in the same rate as the number of paths found at each capture cycle, there is still scope to improve the fault coverage. But as our primary goal was not improvement of the coverage, we have left it aside for future work. To get a better transition fault coverage, we could generate patterns to test metastability or time-borrowing first, then we could top-off with transition fault patterns. In future, we could work on getting the most small delay defect coverage with relatively fewer test patterns.

In the future, we plan to run experiments on bigger circuits and with varying values of K. We also plan to run these experiments by varying the primary inputs. Although varying the primary inputs may result in increase of the test cost, we plan to

see the effectiveness of our approach in this case. Finally we need to implement dynamic compaction in our at-speed KLPG and evaluate pattern count.

In our experiments, we have not considered the pseudo-functional cycles. Further study could be done to see the effect of preamble cycles in multi-cycle path generation. Our work on multiple at-speed cycles could be integrated with the work on path delay testing for non-scan cells.

In future, we would like to decrease the runtime needed for path generation across multiple capture cycles. We could speed up the path generation time by speeding up the time needed for justification. So in future, experiments on multi-cycle path generation could be performed by using various speed-up techniques.

REFERENCES

[1] L.-T. Wang, C. E. Stroud, N. A. Touba, “System-on-Chip Test Architectures Nanometer Design for Testability,” Ch.2, Ch. 6, Morgan Kaufmann 2010.

[2] M. L. Bushnell, V. D. Agrawal, “Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits,” Springer 2000.

[3] G. L. Smith, “Model for Delay Faults Based Upon Paths,” IEEE International Test Conference, Oct. 1985, pp. 342-349.

[4] W. N. Li, S. M. Reddy, S. K. Sahni, “On Path Selection in Combinational Logic Circuits,” IEEE Trans. On Computer-Aided Design, vol. 8, no. 1, Jan. 1989, pp.56-63.

[5] A. K. Majhi, V. D. Agrawal, J. Jacob, L. M. Patnaik, “Line Coverage of Path Delay Faults,” IEEE Trans. on VLSI Systems, vol. 8, no. 5, Oct. 2000, pp. 610-613.

[6] A. Murakami , S. Kajihara, T. Sasao, I. Pomeranz, S.M. Reddy, "Selection of Potentially Testable Path Delay Faults for Test Generation," IEEE International Test Conference, 2000, pp. 376-384.

[7] Y. Shao, S.M. Reddy, I. Pomeranz, S. Kajihara, "On Selecting Testable Paths in Scan Designs," IEEE European Test Workshop, 2002, pp. 53-58.

[8] K. Fuchs, F. Fink, M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," IEEE Trans. on Computer-

Aided Design of Integrated Circuits and Systems, 1991, vol.10, no.10, pp.1323-1335.

[9] M. Sharma, J. H. Patel, "Finding a Small Set of Longest Testable Paths that Cover every Gate," IEEE International Test Conference, 2002, pp. 974-982.

[10] W. Qiu, D. M. H. Walker, "An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit", IEEE International Test Conference, Sept. 2003, pp. 592-601.

[11] C. Lin, S. Reddy, "On Delay Fault Testing in Logic Circuits," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.6, no.5, Sept. 1987, pp. 694-703.

[12] P. McGeer, R. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," Proc. ACM/IEEE Design Automation Conference, June 1989, pp. 561-567.

[13] J. Benkoski, E. Meersch, L. Claesen, H. Man, "Timing Verification using Statically Sensitizable Paths," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, , vol.9, no.10, Oct. 1990, pp.1073-1084.

[14] H. Chang, J. Abraham, "VIPER: An Efficient Vigorously Sensitizable Path Extractor," Proc. ACM/IEEE Design Automation Conference, June 1993, pp.112-117.

[15] 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," Proc. ACM/IEEE Design Automation Conference, 2002, pp. 566-569.

[16] J. Liou, L.-C. Wang, K.-T. Cheng, "On Theoretical and Practical Considerations of Path Selection for Delay Fault Testing," Proc. IEEE/ACM International Conference on Computer Aided Design, 2002, pp. 94-100.

[17] J. Rabaey et al., Digital Integrated Circuits: A Design Perspective, Prentice-Hall, 1996.

[18] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, “K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits,” IEEE International Test Conference, Oct. 2004, pp. 223-231.

[19] Z. Wang, D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test," IEEE VLSI Test Symposium, 2008, pp. 243-248.

[20] P. Pant, J. Zelman, “Understanding Power Supply Droop During At-Speed Scan Testing,” IEEE VLSI Test Symposium, May 2009, pp.227-232.

[21] K. Bian, D. M. H. Walker, S. Khatri, S. Lahiri, “Mixed Structural-Functional Path Delay Test Generation and Compaction,” IEEE International Symposium Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Oct. 2013, pp. 7-12.

[22] T. Larrabee, “Test Pattern Generation Using Boolean Satisfiability,” IEEE Trans. Computer-Aided Design, vol. 11, no. 1, Jan. 1992, pp. 4-15.

[23] K. Bian, D. M. H. Walker, S. Khatri, "Techniques to Improve the Efficiency of SAT based Path Delay Test Generation," IEEE International Conference on VLSI Design, Mumbai, India, Jan. 2014, paper A1-2.

[24] N. Eén, N. Sörensson, "An Extensible SAT-solver," SAT 2003: 502-518.

[25] Z. Fu, Y. Yu, S. Malik, "Considering Circuit Observability Don't Cares in CNF Satisfiability," Proc. Design Automation and Test in Europe, 2005, pp. 1108-1113.

[26] N. Saluja, S. P. Khatri, "A Robust Algorithm for Approximate Compatible Observability Don't Care (CODC) Computation," Proc. ACM/IEEE Design Automation Conference, 2004, pp. 422-427.

[27] N. Eén, N. Sörensson, "The MiniSat Page, Introduction". Retrieved from minisat.se on March 2015.

[28] K. Bernstein et al., "High Speed CMOS Design Styles", Ch. 8, Kluwer Academic Publishers 1999.

[29] N. Weste, D. Harris, "CMOS VLSI Design", Ch. 7, Pearson Addison-Weasley, 2004. Retrieved from http://www.aw- bc.com/info/weste/assets/downloads/ ch7.pdf.

[30] K. Y. Chung, S. K. Gupta, "Structural Delay Testing of Latch-Based High-Speed Pipelines with Time Borrowing," IEEE International Test Conference, Oct. 2003, pp. 1089-1097.

[31] Altera Corporation, "Understanding Metastability in FPGAs". Retrieved from http://www.altera.com/literature/wp/wp-01082-quartus-ii-metastability.pdf

on March 2015.

[32] H.-C. Tsai, K.-T. Cheng, V.D. Agrawal, "A Testability Metric for Path Delay Faults and its Application," Proc. Asia and South Pacific Design Automation Conference, 2000, pp. 593-598.

[33] K. Y. Chung, S. K. Gupta, “Low-cost Scan-based Delay Testing of Latch-based Circuits with Time Borrowing,” VLSI Test Symposium, 2006, pp. 8-15.

[34] T. Sato, Y. Kunitake, "A Simple Flip-Flop Circuit for Typical-Case Designs for DFM," 8th International Symposium on Quality Electronic Design, 2007, pp. 539-544.

[35] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, T. Mudge, “A Self-Tuning DVS Processor Using Delay-Error Detection and Correction”, IEEE Journal of Solid-State Circuits, April 2006, vol. 41, no.4, pp. 792-804.

[36] N. Shah, R. Samanta, M. Zhang, J. Hu, D. M. H. Walker, “Built-In Proactive Tuning System for Circuit Aging Resilience,” IEEE International Symposium on Defect and Fault-Tolerance of VLSI Systems, Oct. 2008, pp. 96–104.