# HEURISTICS BASED TEST OVERHEAD REDUCTION TECHNIQUES IN VLSI CIRCUITS

A Dissertation

by

AVIJIT CHAKRABORTY

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

DOCTOR OF PHILOSOPHY

Chair of Committee, Duncan M. H. Walker

Committee Members, Jiang Hu
Jyh-Charn Liu
Rabi Mahapatra

Head of Department, Scott D. Schaefer

May 2022

Major Subject: Computer Engineering

# ABSTRACT

The electronic industry has evolved at a mindboggling pace over the last five decades. Moore's Law [1] has enabled the chip makers to push the limits of the physics to shrink the feature sizes on Silicon (Si) wafers over the years. A constant push for power-performance-area (PPA) optimization has driven the higher transistor density trends. The defect density in advanced process nodes has posed a challenge in achieving sustainable yield. Maintaining a low Defect-per-Million (DPM) target for a product to be viable with stringent Time-to-Market (TTM) has become one of the most important aspects of the chip manufacturing process. Design-for-Test (DFT) plays an instrumental role in enabling low DPM. DFT however impacts the PPA of a chip. This research describes an approach of minimizing the scan test overhead in a chip based on circuit topology heuristics. These heuristics are applied on a full-scan design to convert a subset of the scan flip-flops (SFF) into D flip-flops (DFF). The K Longest Path per Gate (KLPG) [2] automatic test pattern generation (ATPG) algorithm is used to generate tests for robust paths in the circuit. Observability driven multi cycle path generation [3][4] and test are used in this work to minimize coverage loss caused by the SFF conversion process. The presence of memory arrays in a design exacerbates the coverage loss due to the *shadow* cast by the array on its neighboring logic. A specialized behavioral modeling for the memory array is required to enable test coverage of the shadow logic. This work develops a memory model integrated into the ATPG engine for this purpose. Multiple clock domains pose challenges in the path generation process. The inter-domain clocking relationship and corresponding logic sensitization are modeled in our work to generate synchronous inter-domain paths over multiple clock cycles. Results are demonstrated on ISCAS89 and ITC99 benchmark circuits. Power saving benefit is quantified using an open-source standard-cell library.

# ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude for my advisor Dr. Duncan M. (Hank) Walker. I would like to thank him for giving me the opportunity to pursue my research under his guidance. He has been an amazing mentor and has always provided me with support and technical guidance.

I am grateful to my committee members Dr. Mahapatra, Dr. Hu and Dr. Liu for their encouragement and support.

I would like to thank all the faculty members and the staff of the department of Computer Science and Engineering for their support to make my experience at Texas A&M University so special.

Finally, I would like to thank my parents and in-laws for their love and support. And most importantly, I want to thank my wife Swati to whom I will eternally be indebted for her support throughout the whole journey, for making so many sacrifices to enable me to complete my PhD research, and my two-year-old son Avyan for his joyful presence in our lives and being the reason for all the inspiration and the drive to do more.

# CONTRIBUTORS AND FUNDING SOURCES

This work was supervised by a dissertation committee consisting of Dr. Duncan M. (Hank) Walker [advisor] , Dr. Rabi Mahapatra and Dr. Jyh-Charn (Steve) Liu of the Department of Computer Science and Engineering and Dr. Jiang Hu of the Department of Electrical and Computer Engineering.

This work was partially supported by a scholarship from the Department of Computer Science and Engineering. The rest of the work was personally funded.

# TABLE OF CONTENTS

Page

ABSTRACT .....ii

ACKNOWLEDGEMENTS .....iv

CONTRIBUTORS AND FUNDING SOURCES .....v

TABLE OF CONTENTS .....vi

LIST OF FIGURES ..... viii

LIST OF TABLES .....x

1. INTRODUCTION .....1

2. PATH DELAY TEST .....6

2.1. Path Delay Test .....6
2.1.1. Delay Test Problem .....7
2.1.2. Path Sensitization .....10
2.1.3. Robust and Non-Robust Path Delay Test .....11
2.2. SCAN based Delay Test .....11
2.3. Clocking Scheme in At-Speed Scan Test .....13
2.3.1. Launch On Shift (LOS) .....13
2.3.2. Launch On Capture (LOC) .....14
2.4. KLPG Algorithm .....15
2.5. Observability Driven Path Extension .....19

3. HEURISTIC BASED SFF CONVERSION FOR POWER RECOVERY .....22

3.1. Prior work .....22
3.2. Heuristic Driven SFF Conversion .....23
3.2.1. Topological Properties of SCAN Design .....23
3.2.2. SFF Conversion Metrics .....26
3.2.3. Memory Shadow Logic Coverage .....33
3.2.4. Clock Domain Crossing .....40

4. EXPERIMENTAL RESULTS .....46

4.1. Experimental Results on the Topological Heuristics .....46
4.2. Memory Test Results .....61

4.3. Experimental Results on Clock Domain Crossing ........................................... 62

5.CONCLUSIONS AND FUTURE WORK ........................................................... 64

REFERENCES ........................................................................................... 66

# LIST OF FIGURES

Page

Figure 2.1 Delay Fault in a Circuit. Reprinted from [15]...............................................8

Figure 2.2 Sequential Logic Stage ......................................................................9

Figure 2.3 Timing Diagram for Sequential Operation ..............................................9

Figure 2.4 Path Sensitization...........................................................................10

Figure 2.5 SCAN FF Cell. Reprinted from [16] ....................................................11

Figure 2.6 SCAN Architecture. Reprinted from [16]...............................................12

Figure 2.7 Launch on Shift Clocking Scheme. Reprinted from [16] ...........................14

Figure 2.8 Launch on Capture Scheme. Reprinted from [16] ...................................14

Figure 2.9 Partial Path Growth. Reprinted from [16]...............................................15

Figure 2.10 KLPG Path Generation Flowchart. Reprinted from [2]............................16

Figure 2.11 Multi Cycle Path Generation Clocking Scheme .....................................20

Figure 2.12 Multi Cycle Path Growth in KLPG. Reprinted from [4] ...........................20

Figure 3.1 Topological Characteristics in a Generic Circuit.......................................24

Figure 3.2 Topology Dependent Path Justification...................................................25

Figure 3.3 Synthetic Benchmark Circuit..................................................................26

Figure 3.4 Connectivity Matrix...............................................................................29

Figure 3.5 SFF Conversion Pseudocode .................................................................31

Figure 3.6 Pseudocode for SFF Conversion Procedures.............................................32

Figure 3.7 At-Speed Path Generation through a Memory Array ..................................33

Figure 3.8 Memory Array Block Diagram.................................................................36

Figure 3.9 Memory CRAM Primitive........................................................................37

Figure 3.10 Synthetic Memory Benchmark Circuit ...................................................39

Figure 3.11 Path Generation in CDC Scenario ............................................................42

Figure 3.12 Clock Path Justification .................................................................................43

Figure 4.1 Path Profile in s5378 Benchmark Circuit .....................................................47

Figure 4.2 Coverage Loss with Varying DFF%..............................................................48

Figure 4.3 SFF Conversion Statistics................................................................................50

Figure 4.4 Multi-Round Conversion in s5378 ................................................................51

Figure 4.5 Tradeoff between SFF Conversion% and Pattern# Increase in ISCAS89 Benchmark Circuits ..........................................................................................57

Figure 4.6 Tradeoff between SFF Conversion% and Pattern# Increase in ITC99 Benchmark Circuits ..........................................................................................58

Figure 4.7 Leakage Power Saving Statistics ....................................................................60

Figure 4.8 Synthetic Benchmark Circuit with Multiple Clock Domains .........................63

# LIST OF TABLES

Page

Table 4.1 SFF Conversion Matrix with One Coda Cycle Clocking Method ................... 52

Table 4.2 Path Capture Statistics in s5378 ........................................................ 53

Table 4.3 Tradeoff between SFF Conversion and Test Coverage Loss ........................... 55

Table 4.4 Memory Test Statistics ....................................................................... 62

# 1. INTRODUCTION

The electronics industry has seen rapid advancement in the fabrication process over the last several decades. The push for power-performance-area (PPA) in all product segments has resulted into the development of bigger and more complex chips. Device variation in advanced process nodes has posed significant challenges for achieving sustainable yield for viable product. Achieving low defect-per-million (DPM) quality is of utmost important in enabling high yield. DFT plays a major role in ensuring product quality and reliability by allowing the discovery of faults in the design throughout the product life cycle. The DFT infrastructure however impacts product PPA. This research describes the approach of minimizing the PPA impact of SCAN test infrastructure with the help of circuit topology-based heuristics. This work extends the observability driven multi cycle path generation described in [4] by augmenting the method to determine the percentage of the scan flip-flops (SFFs) in the design that can be converted into D-type flip-flops (DFFs) in order to save area, power and delay with minimal loss in fault coverage. The heuristics to guide the conversion consider several aspects associated with the SFFs viz. controllability and observability of the gates in the fan-in cone of the SFF that are driven by SFFs in their respective fan-in cones, presence of memory arrays, presence of feedback loops in the paths, clock gating and multiple clock domains for clock-domain-crossing (CDC) aspects.

We use at-speed path delay test in our research for characterizing circuit performance. At speed path delay test is used to characterize a circuit for its specification frequency of operation. The operating frequency of a design is limited by the timing critical path. A timing critical path has the longest delay. The longest delay is determined by the total number of gates on the path and the total accumulated delay of individual stages along the path. The delay of each stage of the gates varies on Silicon due to the on-die variation of the device and interconnect properties. It is essential to check timing on such longest paths to detect faults caused by increased delay. The path delay test method detects faults on longest paths by launching a logic transition at a sequential flip-flop and capturing the transition at another sequential flip-flop. A path in the delay test is defined as a set of connected gates, starting at a SFF output, which is a pseudo primary input (PPI) and ending at a SFF input, which is a pseudo primary output (PPO). The total delay on a path is the cumulative delay of individual gates on that path. Hence, the path generated for the delay test is required to be *longest* to increase the probability of detecting a fault. A delay metric called *Esperance* [5] is utilized to generate a *longest* path in this work. *Esperance* is defined as the upper bound of the delay of a partial path which starts growing from a PPI and has not reached a PPO to become a complete path. The number of paths in a circuit grows exponentially with the size of the circuit. In order of keep the path generation tractable, the K longest paths per gate (KLPG) ATPG algorithm is used [2]. In order to be able to control and observe these flip-flops during test, the sequential elements need to be scannable flip-flop (SFF) cells. The D (delay or data) flip-flop (DFF) elements in the design are converted into SFF to enable path delay test. SFF cells are larger in area, power and delay compared to the DFF cells. Hence, the scan test capability has an area, power and delay impact on the functional design. Depending on the test infrastructure, the PPA overhead can be quite significant [6].

In this work, we developed heuristics based on the circuit topology to convert a full-scan design into a partial-scan [7][8][9] design by converting a subset of the SFFs into DFFs. Any path ending at these DFFs cannot be observed in test mode. We consider these paths to be *false* paths. These false paths result in test coverage loss. Observability driven multi-cycle path extension is used in this work to reclaim the lost test coverage as a result of the SFF conversion.

The KLPG algorithm is implemented in C++. The experiments are run on the ISCAS89 and ITC99 benchmark circuits. We used a 130nm open-source process design kit (PDK) and standard cell library [10] to quantify the leakage power benefit achieved by the SFF conversion process at zero coverage loss for these benchmark circuits.

The key contributions of this research work are summarized here. The subsequent chapters elaborate on the items summarized below.

- **Topological heuristics for SFF Conversion:** Circuit topology driven heuristics are developed in this work that are used to guide the SFF conversion process. SCOAP metrics and Path-Based-Analysis are used to determine the optimized subset of the SFFs in the circuit that can be converted into DFFs to maximize leakage power saving without resulting in coverage loss. Our work demonstrates significant leakage power saving without reducing the test coverage, at the cost of increased pattern count which is superior compared to the work  done in [7].
- **Memory Shadow Logic Test in KLPG:** We developed a combinational memory model (CRAM primitive) in our work. This model is used to determine memory behavior in the at-speed path delay test. It allows generation of at-speed paths

through the memory arrays and to sensitize logic gates driven by the memory data port. This work is an improvement over the memory test developed in KLPG described in [22] where logic synthesis was required to create a gate level netlist of the memory array for delay test.

- **Modeling of Clock Domain Crossing in KLPG:** Our work enables handling of the clock-domain-crossing in path delay test over multiple cycles in KLPG. We only model synchronous clock domains in our work. The clock domain frequencies can be different, while the edges are phase-aligned. We do not model asynchronous domains in our work, where no phase relationship exists between the clock domains. Our work is an improvement over the existing KLPG algorithm wherein only one clock domain is assumed for at-speed path generation.
- **Implementational Improvements:** A regular expression-based circuit parser was developed in this work. This parser is used to process circuits with embedded memory instances and generate the RTL that is compatible with the KLPG on-the-fly. We also implemented verbose logging in this work to help prove the validity of the implementation and provide logging capabilities found in the industry vendor tools. Both of these items are improvement over the existing solutions in KLPG. The existing parser had specific spacing requirements to be able to parse RTL files. The amount of logging was not sufficient to prove the validity of the implementation.

The dissertation is organized as following. In chapter 2, we introduce the path delay test and associated definitions. This chapter provides the background for our work and also alludes to the prior work done in this field. Chapter 3 elaborates on the motivation of the research work and the implementation of the topological heuristics. Chapter 4 discusses the experimental results and chapter 5 concludes the research with directions for future work

# 2. PATH DELAY TEST

In this chapter we introduce the basic concepts of the path delay test. We first define the nomenclature used in delay tests. The KLPG algorithm is summarized in a flowchart. The multi-cycle observability driven path extension concept is elaborated with the help of illustrative examples. The definitions serve as the background of the heuristic driven SFF conversion process described in the next chapter.

## 2.1. Path Delay Test

Delay test is used to characterize the performance of a circuit. The objective of the delay test is to discover delay faults in a circuit. Delay faults cause degradation in the circuit performance. The degradation in performance causes the delay tests to fail against the specification frequency targets. The delay fault can be modeled in various ways. We use the path delay fault model [11][12][13] in our work. Both localized and distributed delay faults can be detected using this model. The delay is measured on a *path* in the circuit. A path is defined as a set of connected gates between two sequential elements in one sequential stage of the circuit. The starting point of a path can be a primary input (PI) or a pseudo-primary input (PPI) which is the output of a sequential element, which are the flip-flops (FF) in the circuit. The end point of a path can be a primary output (PO) or the input of a FF. The delay test is a SCAN based test [14]. All the FFs in the design are converted into SFFs. These SFFs provide direct controllability and observability of the internal nodes in a design during the test mode. We consider the PIs in a design to be held constant and the POs ignored during the test, as is the case in a low-cost tester setup. The paths generated in our work start at a PPI and end at a PPO.

A logic transition is launched by the PPI and captured by the PPO. The input of a gate on which such transition appears is called an on-path input. All the other inputs of the gate are called the off-path inputs. For the launched transition to traverse through the gate, all the off-path inputs need to be held at non-controlling values. This requirement is called path sensitization. The path is said to have a delay fault if the arrival time of the transition at the PPO exceeds the specification time. The delay of the path is the accumulated stage delay of each gate on that path. It is intuitive that the longest paths are more susceptible to delay faults. These *long* paths are termed timing critical paths. These paths determine the maximum attainable frequency of a design. We generate such longest paths in our path delay test. The number of paths in a circuit grows exponentially with the circuit size. Identification of longest sensitizable paths through all the gates becomes extremely challenging. In order to limit the search space and improve the tractability of the test, K Longest Paths per Gate (KLPG) are generated. We can test local delay faults caused by a slow gate and globally distributed delay faults caused by process variation on a chip.

We summarize some of the key concepts in the path delay test in the following section.

### 2.1.1. Delay Test Problem

Figure 2.1 illustrates the delay fault in a combinational circuit. A set of vectors $v_1$ and $v_2$ are applied at the inputs ($x_1$, $x_2$, $x_3$) of the circuit.

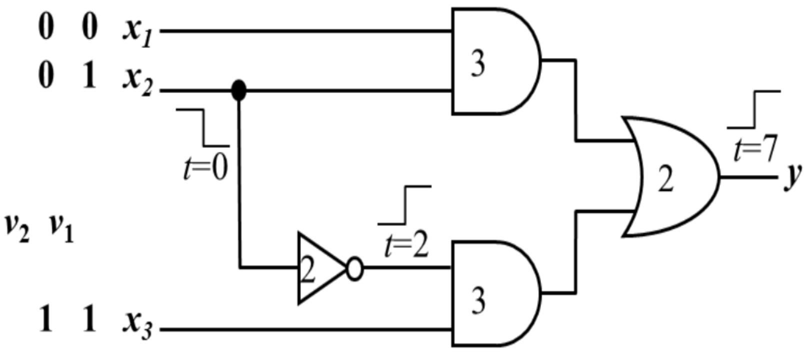


**Figure 2.1 Delay Fault in a Circuit. Reprinted from [15]**

The vector $v_1$ is called the initialization vector and the vector $v_2$ is called the test vector. The ($v_1$, $v_2$) vector pair launches the transition at the inputs for the delay test. The launched transition at the input of the circuit traverses through the gates in the design. The resulting logic transition can be observed at the output ($y$) of the circuit.  The specification delay of each of the gates is written inside the gate symbols. The accumulated delay of the longest path in the circuit is 7. The falling transition launched at the input $x_2$ at $t$=0, generates a rising transition at the output $y$ at $t$=7. If any of the gates on this path has additional delay, the value of the output at $t$=7 will remain zero. If the value of $y$ is compared against the specification value at $t$=7, the test will fail.

In industrial circuits, the logic connectivity is much more complex. The delay of the critical path depends on the fan-in and fan-out of the logic gates, loading at every stage, physical routing characteristics, device variation, etc.

Figure 2.2 illustrates a generic sequential stage in a circuit. FF1 and FF2 form the single cycle sequential stage.

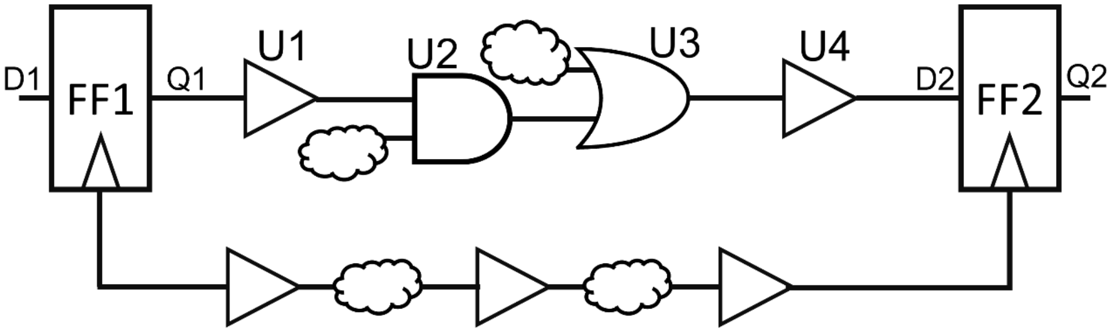


**Figure 2.2 Sequential Logic Stage**

The logic gates U1, U2, U3 and U4 represent the gates on the critical path. The fan-in logic cones of U2 and U3 gates are depicted as logic clouds in this figure. The clock path is represented using generic buffers and logic cloud in this figure. In order for this circuit to work correctly, the delay of the critical path cannot exceed the cycle time and setup constraints of the circuit. Figure 2.3 illustrates the delay fault scenario for this circuit.

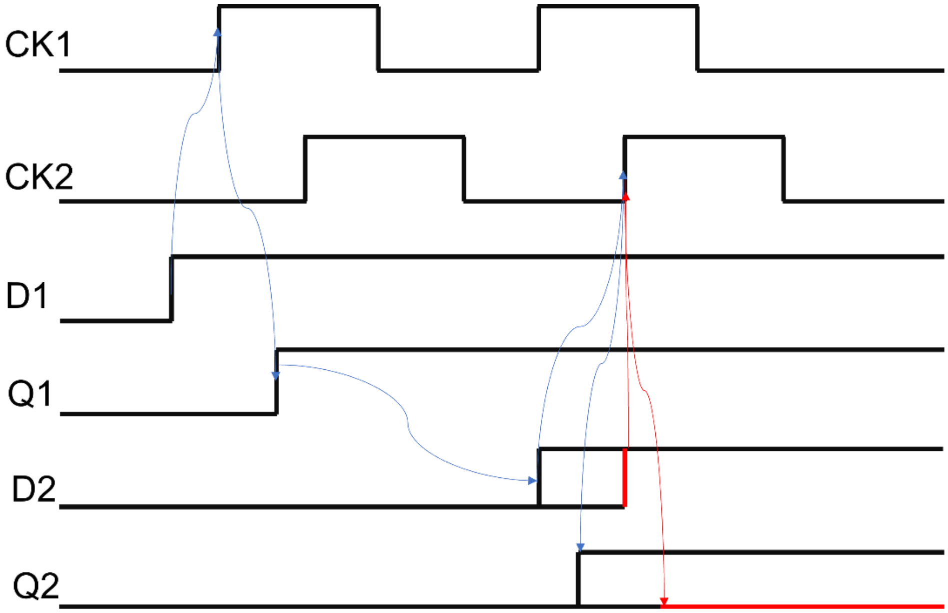


**Figure 2.3 Timing Diagram for Sequential Operation**

The data path delay from Q1 to D2 is determined by the gates delays of U1, U2, U3 and U4. If the rising transition at D2 arrives before the setup time of FF2 in the capturing cycle, the value of Q2 is captured correctly. This is depicted by the earlier transition on the D2 net in the figure. Any delay degradation along the critical path can push the rising transition on the D2 net which is depicted by the rising transition in red. If the rising transition reaches FF2 later than its setup time, the correct value will not be captured at Q2 which is depicted by the red line representing the logic zero. In this scenario, the sequential stage is said to have a delay fault.

### 2.1.2. Path Sensitization

A path can be tested for a delay fault only if it can be sensitized. For a logic transition to propagate through a gate, the off-path inputs of that gate need to be set to non-controlling values. For example, in Figure 2.2, the logic cone of U2 needs to set the off-path input of U2 to a logic ‘1’ for the transition launched by FF1 to traverse through it. Figure 2.4 illustrates a scenario wherein the critical path cannot be sensitized.

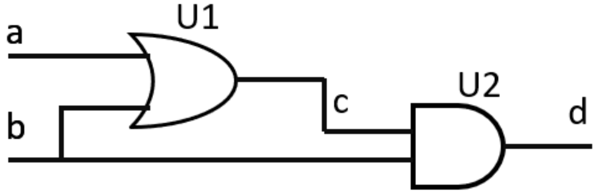


**Figure 2.4 Path Sensitization**

The critical path in the circuit in Figure 2.4 is from the input *a* through net *c* to the output *d*. In order to test the path *a*-*c*-*d*, the side input *b* of the gate U1 needs to be set to logic ‘0’ and the same input *b* needs to be set to logic ‘1’ for the gate U2. This requirement cannot be fulfilled since *b* is the common net between the gates U1 and U2. Hence the path *a*-*c*-*d* cannot be sensitized.

### 2.1.3. Robust and Non-Robust Path Delay Test

A path in delay test can be classified as “robustly testable” or “non-robustly testable” depending on the sensitization criterion. Delay faults can be present on multiple paths in a circuit. If the delay fault on a path can be tested irrespective of any delay faults on any other paths, the test is termed as robust. If the delay fault on that path can be tested only in absence of any other delay faults in the circuit, the test is termed as non-robust.

## 2.2. SCAN based Delay Test

SCAN based delay test is one of the most widely used delay test methods. In SCAN based test, all the FFs in the design are converted into SFFs. A SFF is commonly formed by adding a 2:1 multiplexer (MUX) on the data input pin D of a delay FF. The select signal of the MUX is called Scan Enable (SE). The inputs of this MUX are called Scan-In (SI) and Data-In (DI). The output of the SFF is called Q or Scan-Out (SO). The structure of a SFF is illustrated in Figure 2.5.

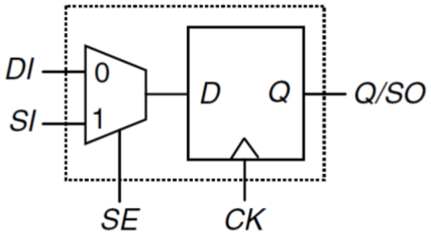


**Figure 2.5 SCAN FF Cell. Reprinted from [16]**

The use of these SFF cells results in PPA impact. The additional devices in the MUX result in more area, power and delay.

The SFFs in the circuit are connected in a daisy chain fashion called SCAN chain. The SO of an SFF is connected to the SI of the next SFF in the chain. The DI input is used in the functional mode to drive functional data into the SFF. Test vectors are driven from the SI input in the test mode. The low frequency test mode clock called scan clock is used during the scan test. The SE signal is asserted to enable the scan mode test. The scan architecture is illustrated in Figure 2.6.

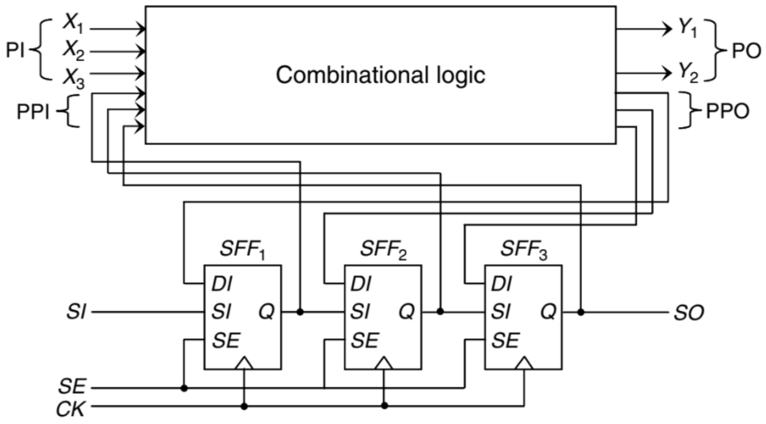


**Figure 2.6 SCAN Architecture. Reprinted from [16]**

The SFF outputs provide direct controllability to the internal circuit nodes and act as PPIs. The inputs of the SFFs provide direct observability to the internal circuit nodes and act as PPOs. The circuit operates in three modes. Normal, Shift and Capture. The SE signal is asserted first to put the design in test mode. The test vectors are shifted into the SFFs over several low-speed scan clock cycles. Once the vector is loaded into the chain, SE is de-asserted to put the circuit into functional or normal mode of operation. The test vector is applied to the combinational logic of the circuit from the PPI inputs in this mode. An At-Speed functional clock cycle is applied to generate the response of the test vector. SE is asserted again to put the circuit back into test mode. The response of the circuit is shifted out from the scan chain over several scan clock cycles.

Various SCAN architectures exist other than the Muxed-D scan design described above. We use Muxed-D scan design in our work. Hence, we limit our discussion only to this architecture.

### 2.3. Clocking Scheme in At-Speed Scan Test

The design under test (DUT) is tested at the specification frequency in the At-Speed Scan Test. More than one clock domain can exist in a DUT. The clock domains can either be synchronous or asynchronous with respect to each other. Two clock domains are classified as synchronous if the triggering edges of the clocks from these domains can be precisely aligned. In case this alignment is not possible, the clock domains are said to be asynchronous.

Two primary clocking schemes for testing inter- and intra-clock-domain at-speed delay faults are described below.

#### 2.3.1. Launch On Shift (LOS)

In this clocking scheme, a capture clock pulse is applied right after the last shift clock pulse. The capture clock pulse is clocked at the functional frequency. This scheme requires the SE signal to switch at the functional speed. This poses significant challenge in meeting the timing on the SE signal. Figure 2.7 illustrates this clocking scheme.

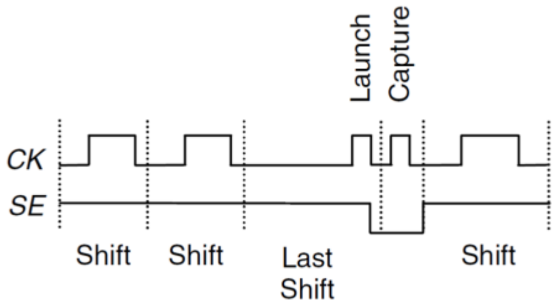


Figure 2.7 Launch on Shift Clocking Scheme. Reprinted from [16]

### 2.3.2. Launch On Capture (LOC)

In this clocking scheme, two consecutive capture cycles are used to launch the test vector and capture the response respectively. This clocking scheme does not impose any stringent timing requirement on the SE signal, since dead cycles can be inserted to give the SE signal time to settle. The SE signal can be de-asserted after the test vector has been shifted into the scan chain. It is required to be asserted again only after the double capture cycles. This clocking scheme typically requires more test vectors and provide relatively lower test coverage compared to the LOS scheme. However, due to the relaxed timing requirement, LOC is the preferred clocking scheme for high-speed circuits. Figure 2.8 illustrates the LOC clocking scheme.

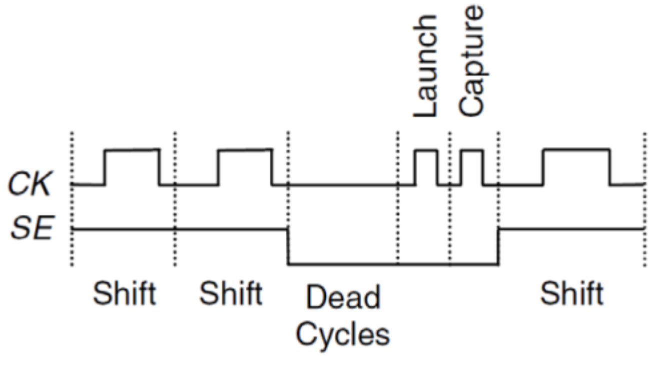


Figure 2.8 Launch on Capture Scheme. Reprinted from [16]

## 2.4. KLPG Algorithm

The total number of paths in a circuit grows exponentially with the size of the circuit. In order to maintain the tractability of the path delay test, the KLPG algorithm aims at generating K longest paths through a gate or a line in a combinational circuit. A complete path starts at a PI or a PPI and end at a PO or a PPO. In this work, we focus on the paths between a PPI and a PPO. A path is called a *partial* path that has started at a PPI but has not yet reached a PPO. Both rising and falling transition tests are performed. A timing metric called *Esperance* (French for "hope") is used to calculate the upper bound of the delay of a partial path. Figure 2.9 illustrates the concept of growing the partial path in KLPG.

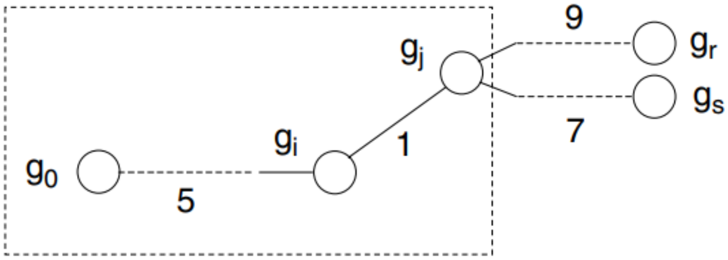


**Figure 2.9 Partial Path Growth. Reprinted from [16]**

A path is generated by traversing through the fan-out cone of the logic gates in a recursive manner. In each iteration, one gate is connected to the output of the gate in the previous stage until a PPO is reached. In Figure 2.9, the traversal starts at gate $g_0$. Only the gate $g_i$ is connected to the gate $g_0$. Hence, the fan-out (FO) of this gate is 1. The delay of the stage $g_o$-$g_i$ is 5. The accumulated delay of this partial path becomes 5 at this stage. In the next iteration, the gate $g_j$ is added to the partial path. The FO of gate $g_j$ is 2. In the next iteration, the gate $g_r$ is added to the partial path instead of the gate $g_s$ because the likelihood of the partial path that started at $g_0$ becoming a longest path is more if $g_r$ is added to the partial path since the stage delay of $g_j$-$g_r$ is more than the stage delay of $g_j$-$g_s$. The partial path growth continues from the FO of the gate $g_r$ in the same way described above until a PPO is reached.

The flow chart of the KLPG algorithm is depicted in Figure 2.10. The KLPG algorithm is divided into three main portions. Path initialization, Path growth and Path justification. This work concentrates primarily on the path generation aspect of the path delay test.

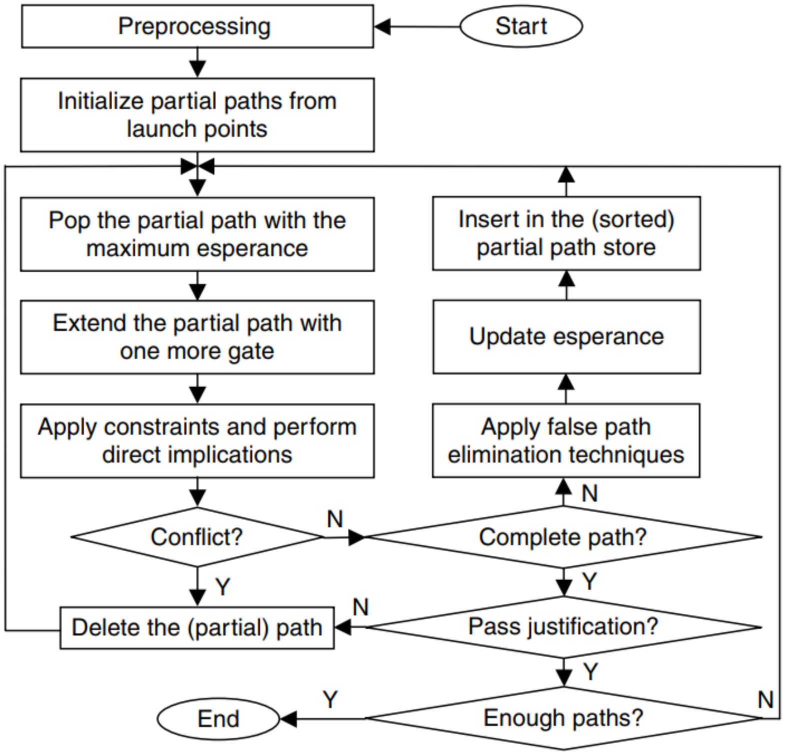

Preprocessing
Start
Initialize partial paths from launch points
Pop the partial path with the maximum esperance
Insert in the (sorted) partial path store
Extend the partial path with one more gate
Update esperance
Apply constraints and perform direct implications
Apply false path elimination techniques
N
Conflict?
N
Complete path?
Y
Y
Delete the (partial) path
N
Pass justification?
Y
End
Y
Enough paths?
N

**Figure 2.10 KLPG Path Generation Flowchart. Reprinted from [2]**

The Verilog of the circuit is read by a parser. The gate connectivity is stored into the data structure. SCOAP metrics [20] are calculated for the gates in the design. The gates are levelized [15] based on their distance from the PI and PPI to ensure proper calculation of the PERT delay [15] and the SCOAP metrics. The distance of a gate from the PI or PPI is calculated as the maximum of distances of the gate from the PI or PPI through all its inputs. Combinational Controllability (CC) and Observability (OBS) metrics are calculated for all the gates. The Controllability is a measure of how easy it is to set the logic state of a node in the circuit to ‘0’ or ‘1’. The Observability is a measure of how easy it is to observe the logic state of a node in the circuit. The CC values are annotated to the input pins of the gates and the OBS values are annotated to the output pins of the gates. The CC increases with the increasing distance from the PI or the PPI. The OBS increases with the increasing distance from the PO or the PPO. The CC of PI and PPI is ‘1’. The OBS of PO and PPO is ‘0’. These metrics are calculated for the gates based on their rank in the circuit. Lower-rank gates are processed for these metrics first before the higher-rank gates in the design.

The path generation step begins after the circuit initialization step is complete. The fan-out each of the gates is traversed recursively in this step. In each iteration of this traversal, one gate is added to the output of the gate in the previous stage to grow the *partial* path. The *Esperance* metric is used to ensure that the longest possible paths are generated in this step. A *partial path pool* is maintained to store the *partial* paths during the growth phase. This pool is sorted in the order of the calculated *Esperance* value. The partial path with the maximum *Esperance* value is popped in each iterative step and a new gate is added to it to grow the path further. If a gate has multiple fan-outs, the *partial* path splits at this point. With the addition of the new gate, *Esperance* is calculated, and the path pool is sorted accordingly. Logical constraints are updated each time a new gate is added to the *partial* path. In order to propagate the launched transition through the newly added gate, non-controlling values on the side inputs are calculated. The logic sensitization constraint is propagated throughout the circuit. If logical conflict prevents the *partial* path to be sensitized, the path growth stops, and the *partial* path is discarded from the pool. The next *partial* path with the maximum *Esperance* value is popped out from the pool and the path growth continues in the same manner. When a PPO is reached during the path growth process, the *partial* path is converted into a complete path and the path growth process ends for that *partial* path. The path growth continues until enough paths are generated which is bound by the value of K.

Justification is performed on the complete path by assigning logic values to the gates on the path. The miniSAT [17] solver is used in this work for this purpose. Compaction is run to reduce the number of the test patterns. Both static and dynamic compaction can be used. The static compaction is the faster approach because it does not require any circuit analysis. The dynamic compaction on the other hand has longer run time due to the requirement of dynamically making sure that a pattern can test the set of generated paths. The dynamic compaction also requires a pattern pool to be maintained for the analysis. It however produces fewer patterns.

The flowchart described above is used to generate at-speed paths over single at-speed cycles. Switching from the functional mode to the test mode causes high current demand from the SE signal switching. This can cause voltage droop in the circuit. The voltage droop causes circuit delay degradation causing false test failures. A number of low-speed cycles termed *preamble cycles* [18] are added before the at-speed test begins to ramp the power supply currents back up to the functional level. This approach helps eliminate non-functional states in the design. This test methodology is called pseudo functional KLPG (PKLPG) [2]. In this research, we do not use any *preamble* cycles, deferring them to future work.

## 2.5. Observability Driven Path Extension

During the scan test mode, the SFFs capture the logic response from the DUT to derive test pass/fail. If any at-speed path ends at a DFF, the captured response cannot be observed in the test mode. We consider these as *false* paths. The *false* paths result in coverage loss. The multi-cycle path generation approach described in [3][4] utilizes the observability metric to extend the at-speed paths that end at a DFF, over multiple lower than at-speed cycles, called *coda cycles*, such that they terminate at a SFF element.

The objective of the at-speed cycle path generation is to generate longest paths. The type of the capturing flop does not matter in this cycle. In the *coda* cycles, the OBS metric is used to extend the path on an SFF. The first launching flop in these cycles need not be an SFF. The at-speed paths captured at a DFF are extended through the most observable fan-out cone to an SFF such that the data captured by the DFF can be observed in the test mode. The extension of the at-speed paths onto SFFs regains the lost coverage.

The clocking scheme used for the multi cycle-based path extension is illustrated in Figure 2.11.

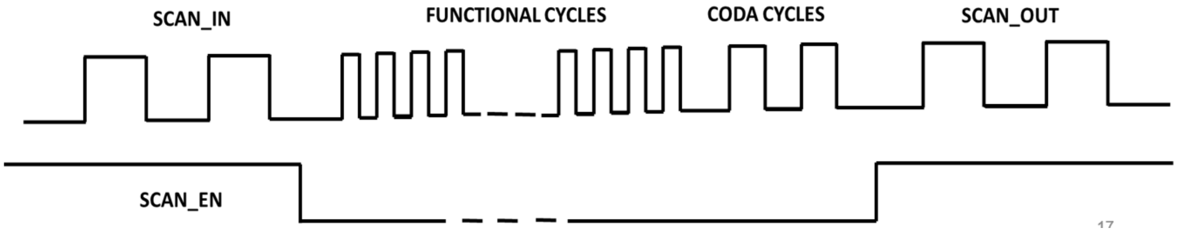


**Figure 2.11 Multi Cycle Path Generation Clocking Scheme**

The use of the *coda* cycles for path extension does not require any additional test hardware. On-chip clock generators produce test mode clocks. We assume that no additional design constraints are required in order to generate the *coda* cycle frequency. The main goal of the *coda* cycles is path extension. Any existing lower than at-speed cycle frequency can be used for this purpose. For simplicity, coda cycles can use the scan shift frequency.

Figure 2.12 illustrates the OBS driven path extension for a generic circuit. The combinational logic in a pipeline stage is depicted with a logic cloud.

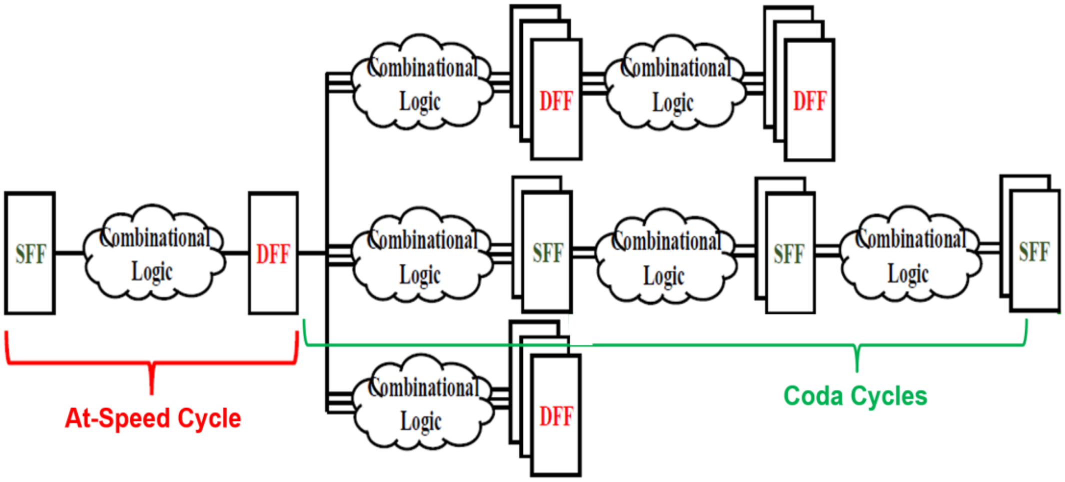


**Figure 2.12 Multi Cycle Path Growth in KLPG. Reprinted from [4]**

The path generation in this example starts at the launching SFF at the left in the figure. The capture FF is a DFF. This DFF fans out to three logic cones. All the FFs in the upper and the lower fan-out cones are DFFs over all of their sequential depths. The FFs in the middle fan-out cone are SFFs at a sequential depth of one, two and three. If a one *coda* cycle clocking scheme is used, the at-speed path can be extended to the first SFF in the middle fan-out logic cone. If this SFF would have been DFF, applying a second *coda* cycle would extend the at-speed path to the second SFF in that logic cone. No *coda* cycle scheme would be helpful in extending the at-speed path through the top and bottom fan-out logic cone. The OBS driven path extension comprehends the likelihood of finding SFFs in the fan-out logic cones in the circuit and attempts to traverse through the viable fan-out logic cones only.

It should be noted that the efficiency of the *coda* cycle path extension depends on the circuit topology. The path extension over multiple cycles needs to satisfy the logical implication and justification related constraints. Although path traversal over multiple *coda* cycles can discover SFFs in the fan-out cones, if the extended path cannot be justified, the at-speed path cannot be extended through that fan-out cone. This scenario would still result in coverage loss due to the presence of the DFF cells as the at-speed capturing FFs.

# 3. HEURISTIC BASED SFF CONVERSION FOR POWER RECOVERY

In this chapter, we describe the topological heuristics for the SFF conversion. We first present the prior work. We describe the topological dependency of the test coverage loss as a result of the SFF conversion. We then elaborate on the path-based analysis (PBA) method and Sandia Controllability Observability Analysis Program (SCOAP) based heuristics developed in this work. Later, we elaborate on the memory shadow logic test and clock domain crossing (CDC) methods developed in this work.

## 3.1. Prior work

Partial scan architecture has been described in various literature. The work done in [18] pre-selects the flip-flops in the design to be converted into SFFs. The process is iterative and guided by fault coverage. The results show very high run time in order to determine the right percentage of SFFs in the design on small benchmark circuits. Application of this approach on large circuits is not practical. Other partial scan approaches exist where multiple clocks are used to control the state of the DFF cells during test. The work in [19] describes a method of achieving partial scan without the use of multiple clocks. The work deploys a cycle breaking technique. However, it does not include scenarios where cycle breaking is not required. Hence the solution is effective on a specific circuit topology. The work done in [20] discusses topological checks that are used to convert SFFs into DFFs. An SFF ranking procedure is described in [21]. This work uses the LOS scheme for transition fault test using a commercial ATPG tool. A significant percentage of the SFFs could be converted in this work, however, it reports sizable pattern count increase and coverage loss with increasing SFF conversion. The notion of partial scan was originally introduced to incorporate as many SFFs in the design as possible within a limited die area. It did not consider the power saving aspect of SFF conversion. Our research focuses primarily on the PPA benefit of minimizing the scan test overhead and demonstrates power saving while minimizing the test coverage impact.

## 3.2. Heuristic Driven SFF Conversion

### 3.2.1. Topological Properties of SCAN Design

The OBS driven multi-cycle path extension mechanism described in the previous section can generate complete paths terminating at SFFs. For a given clocking scheme, all the intermediate FFs on such paths can be DFFs. We utilize this property to convert a subset of the SFFs in a design into DFFs. As mentioned before, the path extension over multiple *coda* cycles does not guarantee test coverage loss recovery unless the extended path can be justified. The logical constraints restricting the efficiency of the *coda* cycle-based scheme is a strong function of the circuit topology. We elaborate the concept with the help of Figure 3.1.

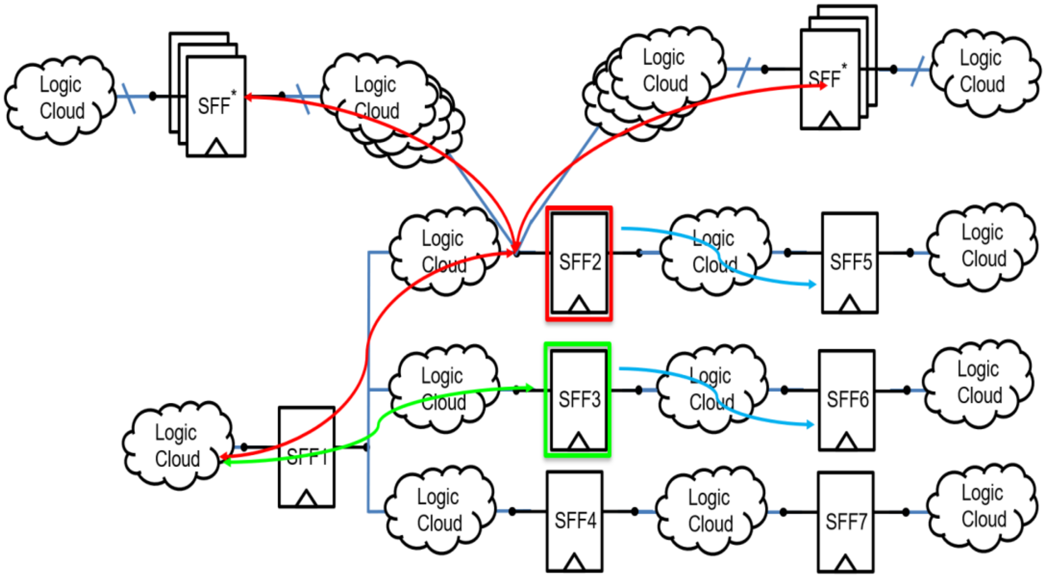


**Figure 3.1 Topological Characteristics in a Generic Circuit**

We use the notation SFF$^*$ to denote multiple SFFs present in the fan-in/fan-out cone of the logic clouds in the figure. We can observe that a large number of paths launched by the SFF* in the top left corner of the figure terminate at SFF2 in this circuit. For SFF3 and SFF4 on the other hand, only one at-speed path can be captured by SFF3 and SFF4 that is launched by SFF1. We assume that all the complete paths can be justified through the logic clouds. Since the circuit is full scan, we can achieve 100% test coverage.

If SFF3 or SFF4 is converted into a DFF, the test coverage impact would be small. However, if SFF2 is converted into a DFF, it would result in significant coverage loss. This clearly illustrates the dependency of the test coverage on circuit topology. The PBA method developed in this work ranks all the SFFs in the design based on the coverage impact resulting from their conversion, in order to guide the SFF conversion process to minimize the coverage impact.

If SFF3 and SFF4 are converted into DFF, we can apply one *coda* cycle to extend the paths to SFF6 and SFF7 respectively. We represent these paths with blue arcs in the figure. The justifiability of these paths would of course depend on the logic cloud on the extended paths. If SFF6 and SFF7 are also converted into DFFs, a single *coda* cycle scheme will not be able to reclaim the lost test coverage resulting from the SFF3 and SFF4 conversion. The number of FFs in the fan-out cones of SFF2-SFF4 is one in this figure. In practical circuits, this number will be more. With the increasing breadth of the fan-out cone of the SFFs, the likelihood of finding an SFF in subsequent stages increases significantly. The coverage recovery may not correlate with the fan-out breadth due to the logical constraints and inability to justify the paths over multiple cycles.

The following figure illustrates the issue in path justification using a simplified example.

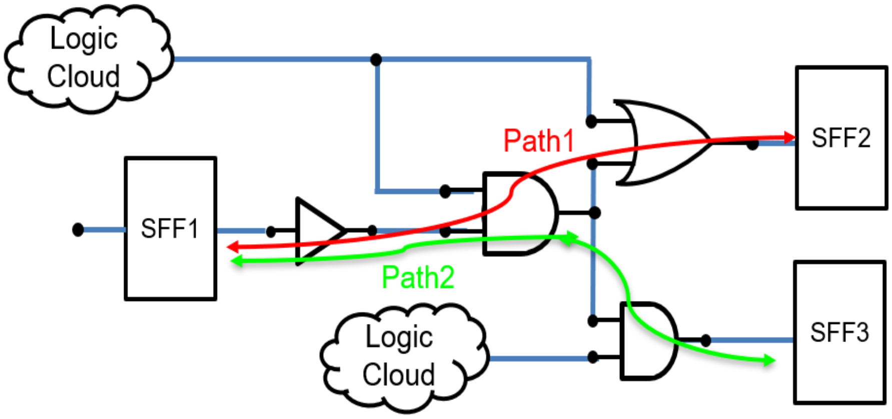


**Figure 3.2 Topology Dependent Path Justification**

It can be observed in Figure 3.2 that in order to justify the path from SFF1 to SFF2, the off-path input of the AND gate needs to be held at logic ‘1’ and the off-path input of the OR gate needs to be held at logic ‘0’. Both of these inputs are connected together and driven by the logic cloud. Clearly, both the gates cannot be sensitized at the same time. If the AND gate is sensitized by setting a logic ‘1’ at its side input, Path1 would not be justified. Assuming that the AND gate in the fan-in cone of SFF3 can be sensitized, only Path2 can be justified in this circuit.

### 3.2.2. SFF Conversion Metrics

We elaborate on the SCOAP based SFF conversion method in this section using a synthetic benchmark circuit in Figure 3.3.

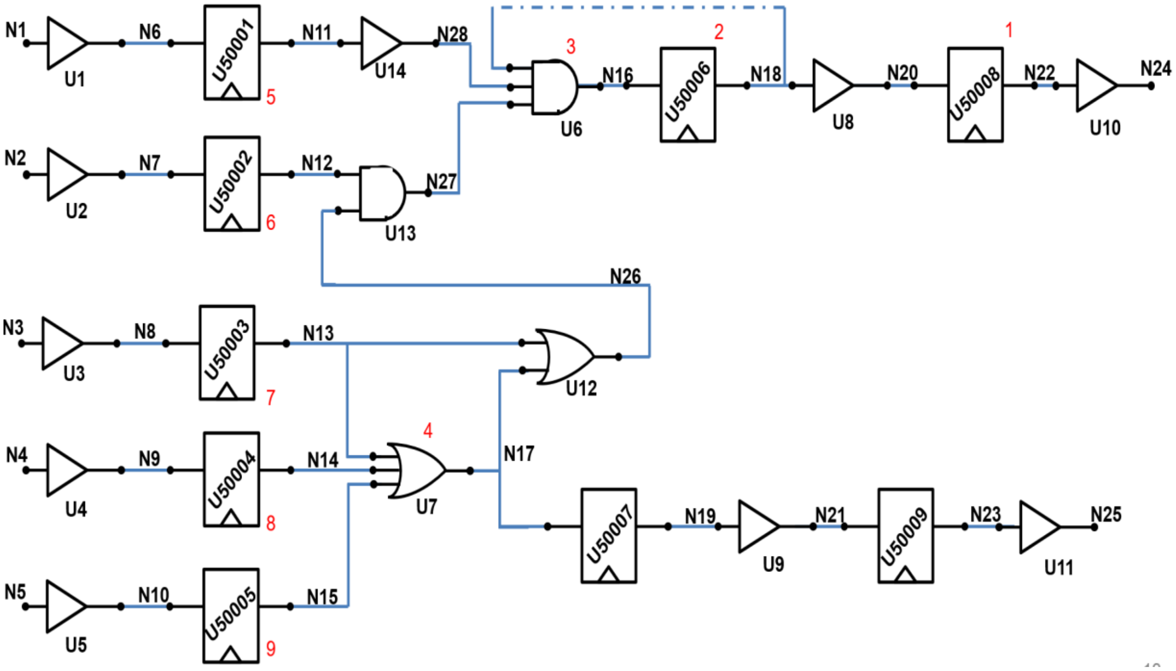


**Figure 3.3 Synthetic Benchmark Circuit**

Multiple rounds of SFF conversion are carried out to optimize the SFF cell conversion process. In the first round of conversion, we perform backward traversal from the input of all the SFFs through all the fan-in lines in the search of other SFFs. During this traversal, the controllability of each node is calculated as a guide to determine if an SFF found during the backward traversal should be converted into DFF. During the path delay test, *off-path* signals at a gate need to be held at non-controlling values to sensitize the path. If the *off-path* signals are driven by DFFs, we lose the ability to control them in test mode, causing coverage loss. Along with the controllability overhead, we use fan-in threshold to control the conversion process as well. With a higher threshold, a larger number of fan-in gates can be reached during the traversal.

The dotted line in Figure 3.3 represents two scenarios. When the line exists, a feedback loop from the output of the SFF marked ‘2’ (U50006) to its input through the gate AND gate (U6) marked ‘3’ exists. We use a fan-in threshold of 2 for this illustration. In the following example, the backward traversal starts at the input of the SFF U50006 and finds the AND gate U6. When the feedback loop is absent, the traversal proceeds to the input of U6 since the fan-in threshold is set to 2. Through the inputs of the gates U13 and U14, U50001 and U50002 are found. These SFFs are marked as viable SFFs that can be converted into DFFs. The SFFs U50003, U50004 and U50005 can also be reached starting from U50002. The traversal however stops at the fan-in cone of the gate U13 since the fan-in threshold is set to 2.

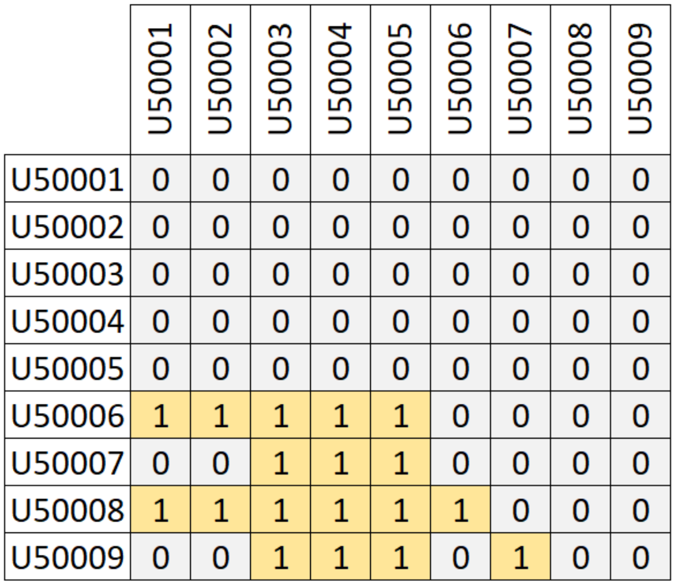
The fan-in threshold models the controllability overhead for a node in the circuit. It assumes a direct correlation between the fan-in width and the controllability of the node. This is more pessimistic compared to the SCOAP controllability metric. We combine the controllability metric with the fan-in threshold to determine whether the backtracking should continue through the input of a gate. A high fan-in gate with low controllability values for its inputs can pass the heuristic threshold compared to a low fan-in gate with much higher controllability values for its inputs. The controllability of the inputs of gate U7 is smaller than the inputs of gate U6. The fan-in depth of U7 is however larger than U6 in the above traversing example. When the controllability metric is combined with the fan-in threshold, the traversal result will be altered for the above example. A large fan-in width gate is likely to have a large number of SFFs in its fan-in cone. With increasing fan-in depth of such a gate, even a larger number of SFFs can be reached during the backward traversal from the inputs of such a gate. Converting the SFFs in such a fan-in cone would severely impact the test coverage. The heuristic would mark such SFFs in the viable SFF pool and not convert them into DFF.

After the first round of the SFF conversion is complete, subsequent rounds of checks are performed to reconvert a subset of the SFFs to address specific scenarios. In Figure 3.5, if the feedback loop from the output of U50006 to its input through gate U6 exists, U50006 will be excluded from the conversion list. The feedback loop in this figure is a simplified representation of a real-life circuit wherein the feedback logic can be a lot more complex and involve much wider and deeper logic cones.

It can be observed that U50001-U50005 in the example circuit act as launch-only SFFs. Converting these into DFFs would severely impact the test coverage. Hence, we exclude these SFFs as well. Other SFFs can be explicitly excluded from conversion due to design intent, such as debug.

Each of these reconversion steps can be individually controlled in the implementation of this work. It helps configuring the ATPG tool based on the specific test methodologies set by the design and test team.

In order to limit the search space in the SFF conversion process, we generate a connectivity matrix to store SFF connectivity information in the circuit. The connectivity matrix of Figure 3.3 is depicted in Figure 3.4.



| | U50001 | U50002 | U50003 | U50004 | U50005 | U50006 | U50007 | U50008 | U50009 |
|---|---|---|---|---|---|---|---|---|---|
| U50001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| U50002 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| U50003 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| U50004 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| U50005 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| U50006 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| U50007 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| U50008 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| U50009 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

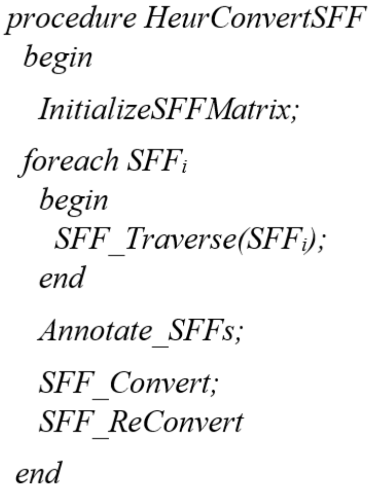
**Figure 3.4 Connectivity Matrix**

A value of one in a box indicates that the SFF in the row index position can be reached from the SFF in the column index position, e.g., U50008 can be reached from U50001 through U50006, whereas U50007 can be reach from U50003 through U50005. The entire row for the SFFs U50001 through U50005 is '0' in the connectivity matrix. This signifies that there are no SFFs in any of their fan-in cones. This indicates that these SFFs are launch-only SFFs. These SFFs can be marked as launch-only by referring to this matrix. Similarly, U50008 and U50009 do not have any SFFs in their fan-out cones. This can be derived from the entire column of zeros for these two SFFs in the connectivity matrix. For a particular clocking scheme, if these two SFFs are converted into DFFs, test coverage loss can be quite high since no further path extension is possible from the output of these two SFFs. The conversion heuristic can use this information and exclude these two SFFs from the conversion pool.

The connectivity matrix is further enhanced to store a *distance* attribute. This attribute represents the sequential depth between a pair of SFFs. This attribute is utilized by the PBA process to perform a what-if analysis to determine the optimal clocking scheme for a circuit. E.g., the distance of U50007 and U50009 from U50005 is one and two respectively. For a specific clocking scheme, the capturing SFF cannot be converted into DFF. For a launch SFF, all the SFFs that can potentially be the capturing SFF for the particular clocking scheme can be determined from the connectivity matrix. If the adjacent column has a non-zero value, the clocking scheme can be modified to add an additional *coda* cycle and extend the paths to the next SFF in the pipeline while converting the SFF into DFF, which was the capturing SFF in the earlier clocking scheme. This analysis is done during the PBA mode to determine the optimal clocking scheme to achieve maximum SFF conversion.

In summary, the SFF conversion heuristics evaluate the impact of converting an SFF on the test coverage using the SCOAP metric and the PBA method. The OBS driven multi-cycle path extension reclaims the lost coverage by extending the at-speed path over the *coda* cycles. The efficacy of the entire scheme in maximizing the SFF conversion while minimizing the coverage impact is heavily dependent on the circuit topology.

The SFF conversion procedure is summarized with the help of pseudocode depicted in Figure 3.5.

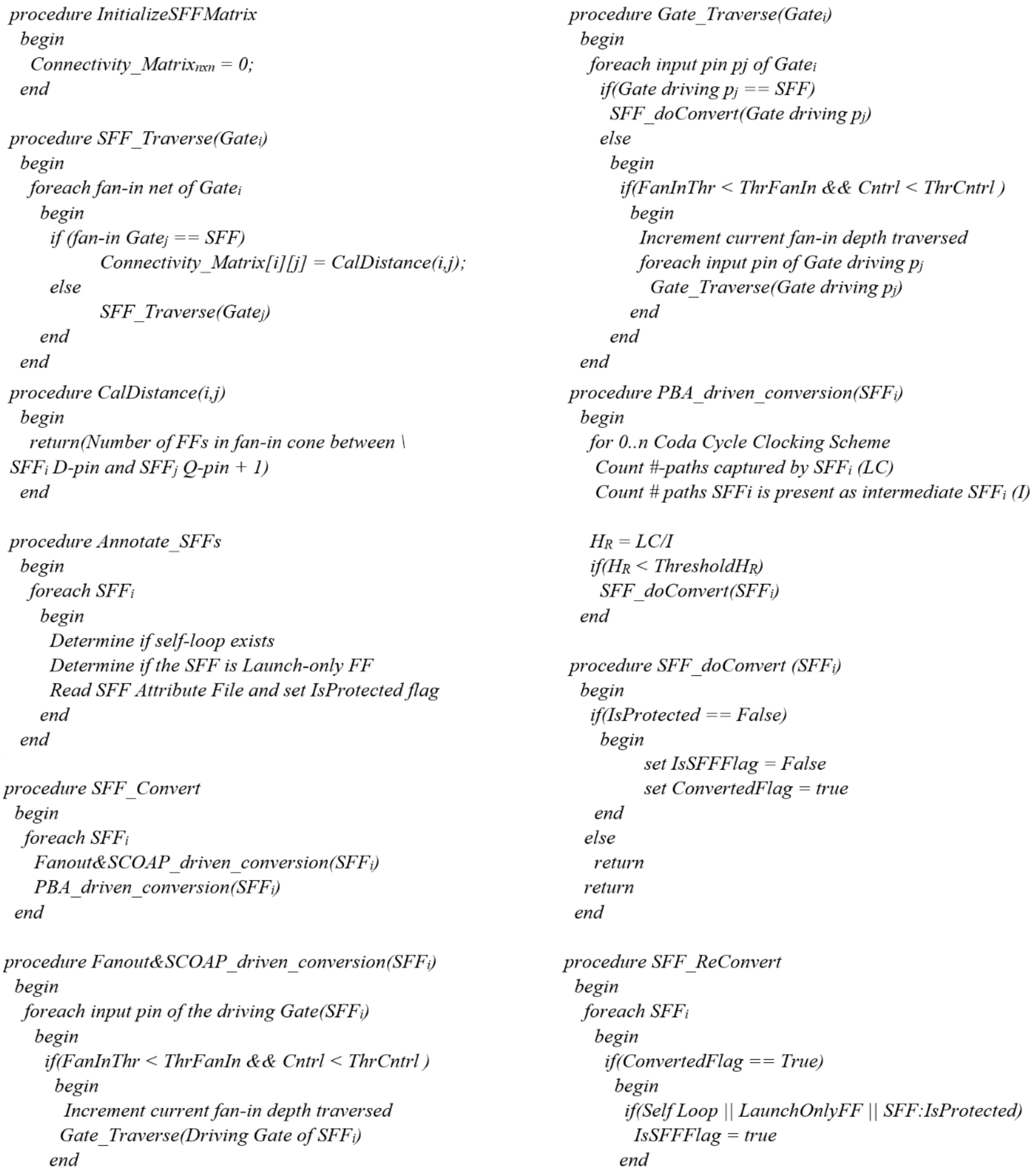


```
procedure HeurConvertSFF
  begin
    InitializeSFFMatrix;
  foreach $SFF_i$
    begin
      SFF_Traverse($SFF_i$);
    end
    Annotate_SFFs;
    SFF_Convert;
    SFF_ReConvert
 end
```

**Figure 3.5 SFF Conversion Pseudocode**

The main procedure *HeurConvertSFF* calls all the subsequent procedures in the code. The connectivity matrix is initialized first using the *InitializeSFFMatrix* procedure. The backward traversal procedure *SFF_Traverse* is called on all the SFFs in the circuit. All the SFFs are annotated with their topological and circuit intent attributes using the *Annotate_SFFs* procedure. The multi-round conversion procedure *SFF_Convert* is called followed by the reconversion procedure *SFF_ReConvert.*

The pseudocode for all the procedures are described in Figure 3.6.



```
procedure InitializeSFFMatrix
  begin
    Connectivity_Matrix$_{nxn}$ = 0;
  end

procedure SFF_Traverse(Gate$_i$)
  begin
    foreach fan-in net of Gate$_i$
      begin
        if (fan-in Gate$_j$ == SFF)
                Connectivity_Matrix[i][j] = CalDistance(i,j);
        else
                SFF_Traverse(Gate$_j$)
      end
  end
procedure CalDistance(i,j)
  begin
    return(Number of FFs in fan-in cone between \
SFF$_i$ D-pin and SFF$_j$ Q-pin + 1)
  end

procedure Annotate_SFFs
  begin
    foreach SFF$_i$
      begin
        Determine if self-loop exists
        Determine if the SFF is Launch-only FF
        Read SFF Attribute File and set IsProtected flag
      end
  end

procedure SFF_Convert
  begin
    foreach SFF$_i$
      Fanout&SCOAP_driven_conversion(SFF$_i$)
      PBA_driven_conversion(SFF$_i$)
  end

procedure Fanout&SCOAP_driven_conversion(SFF$_i$)
  begin
    foreach input pin of the driving Gate(SFF$_i$)
      begin
        if(FanInThr < ThrFanIn && Cntrl < ThrCntrl )
          begin
            Increment current fan-in depth traversed
            Gate_Traverse(Driving Gate of SFF$_i$)
        end

procedure Gate_Traverse(Gate$_i$)
  begin
    foreach input pin pj of Gate$_i$
      if(Gate driving p$_j$ == SFF)
        SFF_doConvert(Gate driving p$_j$)
      else
        begin
          if(FanInThr < ThrFanIn && Cntrl < ThrCntrl )
            begin
              Increment current fan-in depth traversed
              foreach input pin of Gate driving p$_j$
                Gate_Traverse(Gate driving p$_j$)
            end
        end
  end
procedure PBA_driven_conversion(SFF$_i$)
  begin
    for 0..n Coda Cycle Clocking Scheme
      Count #-paths captured by SFF$_i$ (LC)
      Count # paths SFFi is present as intermediate SFF$_i$ (I)

    $H_R$ = LC/I
    if($H_R$ < Threshold$H_R$)
      SFF_doConvert(SFF$_i$)
  end

procedure SFF_doConvert (SFF$_i$)
  begin
    if(IsProtected == False)
      begin
            set IsSFFFlag = False
            set ConvertedFlag = true
      end
    else
      return
    return
  end

procedure SFF_ReConvert
  begin
    foreach SFF$_i$
      begin
        if(ConvertedFlag == True)
          begin
            if(Self Loop || LaunchOnlyFF || SFF:IsProtected)
              IsSFFFlag = true
          end
```

**Figure 3.6 Pseudocode for SFF Conversion Procedures**

### 3.2.3. Memory Shadow Logic Coverage

Memory arrays are instantiated in a circuit as hard macros. A large percentage of the chip area can be attributed to the embedded memory arrays. The ATPG tool needs to handle the memory array behavior differently from a traditional logic gate. Unlike a combinational logic gate, the memory array has sequential behavior. Several different forms of memory arrays exist, e.g., SRAM, DRAM, Register File, ROM and CAM. These memories can have single or multiple read and write access ports as well. The memory arrays can either be a single or multi-clock cycle latency design. Depending on the sequential depth of the memory, number of ports and the associated cross-port logical constraints, the interface pins for the memory need to be controlled accurately during the test. Figure 3.7 illustrates the problem statement of the path delay test in the presence of memory arrays in the circuit.

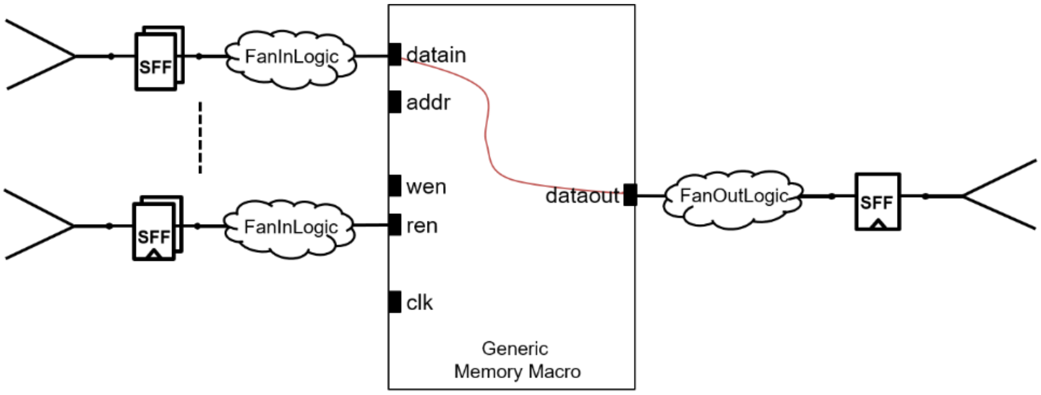


**Figure 3.7 At-Speed Path Generation through a Memory Array**

A potential longest path may start at any of the SFFs on the input side of the memory array in the figure. The path traverses through the fan-out cone of these SFFs and a subset of them may terminate at one of the memory inputs. Unlike a logic gate, the ATPG engine cannot determine the output value from the memory. In order to transfer the transition at the input of the memory array to the output port of the array, the state of the input ports and the functional behavior of the memory need to be defined accordingly.

The combinational logic that feeds into and out of a memory from scan cells is called the *shadow logic*. When the memory is modeled as a 'black box', all of the logic values that feed the memory are lost, and the outputs of the memory are unknown values. As a result, the logic between the scan cells and memory cannot be tested. It is said to be in the *shadow* of the memory.

Prior work in [22] used logic synthesis to generate a gate level representation of a memory array. It used the gate level netlist of the memory to generate longest paths through the memory. This work requires running a logic synthesis tool on the memory RTL, which is not an integrated part of the ATPG engine. This synthesized netlist greatly increased the size of the test model. Several software-based [23][24] and hardware-based [25][26] memory test solutions exist in the literature. In our work, we develop a behavioral model for the memory that is used to test the longest paths that terminate at the input of a memory array and also to sensitize the gates in the *shadow* of the memory array (its output logic-cone) to enable testing of the longest paths that pass through the gates driven by the memory output pins. This approach can be integrated with the existing DFT infrastructure and does not require additional hardware support. Use of LOC clocking scheme along with enhanced scan [27] technique can achieve transition fault coverage for the memory shadow logic. Utilizing bypass MUX at the output of the memory array circumvents the coverage loss for the shadow logic. All of these techniques have additional test overhead.

The behavioral modeling approach in our work minimizes the memory test overhead without requiring any additional DFT capabilities.

We implemented a new regular expression (regex) based RTL parser to process the memory instantiation in the design. The regex parser is a lot more generic compared to the existing parser in the ATPG code and can process generic text with varied number of white-spaces and delimiters in the design.

The memory test can be divided into two modes: RAM Sequential Test and Output Shadow Logic Test. In RAM Sequential Test mode, the behavior of the memory array instance in the RTL is modeled using a combinational logic representation, synchronized with the memory clock. We assume single cycle latency for the memories in our work. Only single ported Register File Array memories are considered in the experiments. The memory model is called a CRAM (combinational RAM) primitive. This model structurally defines the memory. During the path generation step of the test, the partial path that reaches a memory input port, can be extended *through* the CRAM structure. The red colored arc depicted in Figure 3.7 is modeled with multistage logic defined by the behavior of the memory. A generic functional block diagram is illustrated in Figure 3.8.

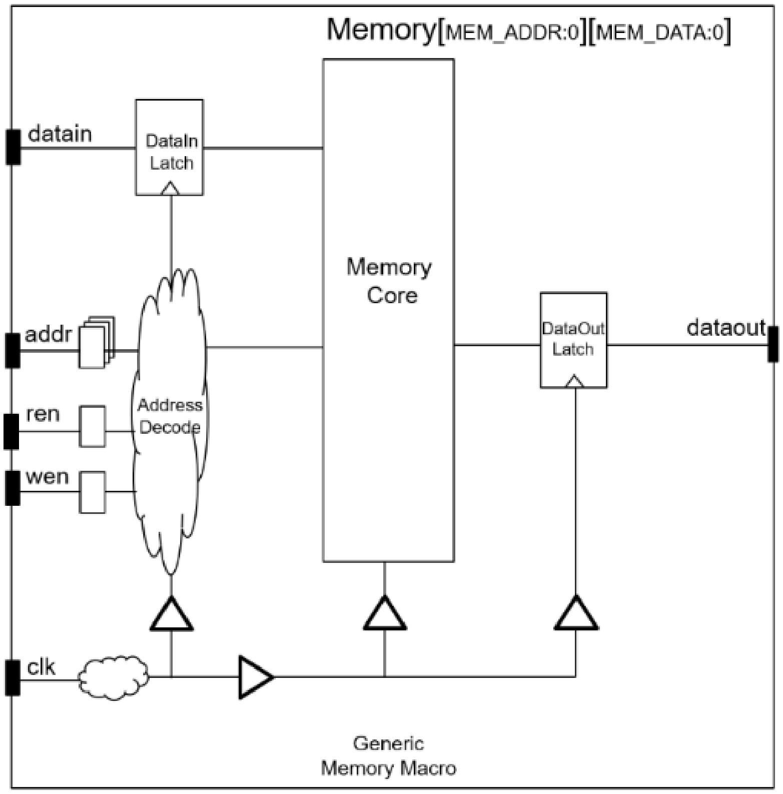


**Figure 3.8 Memory Array Block Diagram**

It is assumed that the input and output FFs are internal sequential elements and are not accessible to the scan chain in the design under test. The address and enable FFs are shown at the input of the decoder logic. The memory core block represents the two-dimensional bit-cell array along with read data path logic including a sense amplifier or a domino stage. Figure 3.8 depicts the block diagram of a single ported memory. For multiple ports, the address, enable and data signals are replicated over all the ports in the design. The test methodology for a multiport memory is similar to the process described in this section. An additional set of signal constraints and clocking requirement may exist to process the data from all the ports of the memory. The CRAM primitive for the memory array in Figure 3.8 is illustrated in Figure 3.9.

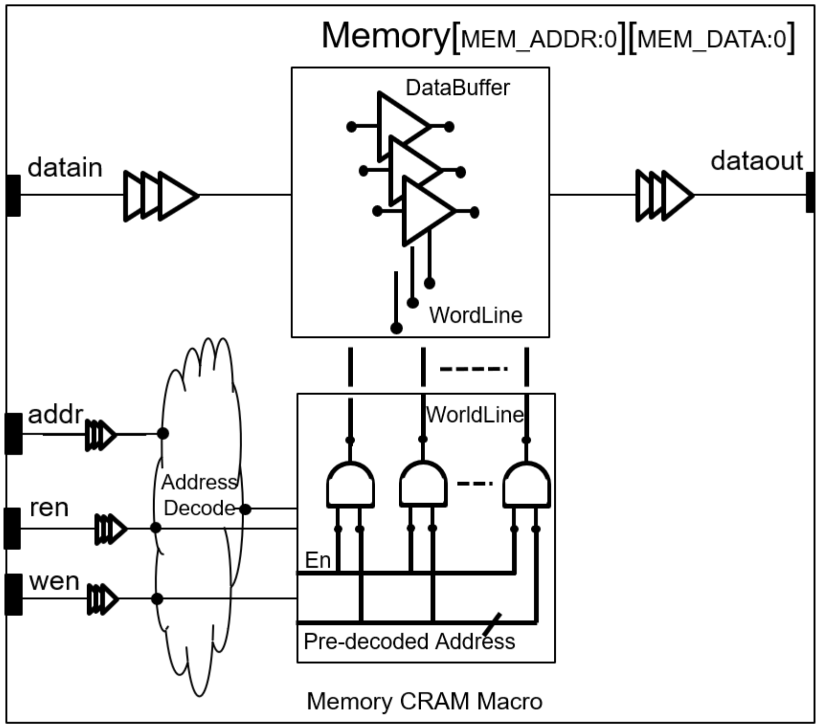


**Figure 3.9 Memory CRAM Primitive**

The boundary of the CRAM primitive interacts with the FFs in the interface of the memory. It can be observed that the sequential elements from the block diagram of the memory are replaced with combinational delay chains inside the CRAM primitive. These delay chains represent the internal sequential path delays of different memory sub-blocks in their functional mode. The CRAM model is divided into two modules; the address-decode and the memory-core. The address decode module consists of the decoder blocks used to generate the read and write word-line signals. These word-line signals are connected to the memory-core module. The memory-core module consists of the data-path logic. Buffer only representation of the bit cells is used in the figure for generality and ease of depiction. The paths from the data-in pins are considered for the write mode and the data-out pins are considered for the read mode. The word-line signals from the decode module are combined with the read/write path logic to generate the bit-cell selection logic for the corresponding mode of operation. The DFT infrastructure present in the test suite needs to comprehend memory test methods developed with the CRAM primitive to be able to control the read and the write operations on the memory array to setup the path delay test through and around the memory array.

In the RAM Sequential Mode, the paths are generated *through* the memory. The DFT associated with the memory array is used to put the memory into test mode. It also controls the write and the read from the memory array to model the transition propagation through the memory. This work uses a unit delay model for path generation. The delay through the memory array is modeled as a multiple of the number of gate delay stages in the CRAM model.

In the Output Shadow Logic Test mode, the memory array is used to sensitize the gates in its fan-out cone to enable the paths passing through those gates. Unlike the RAM sequential mode, the shadow logic test mode does not generate any path through the internal data-path of the memory. Hence the CRAM primitive is used to determine the logic state of the memory output ports to sensitize the gates present in the fan-out of the memory array. In this mode, we first determine the sensitization criterion for the gates that are driven by the memory output pins. A test vector is derived from this information. This test vector has the width of the memory data port. This vector is written into the memory.

Any random address can be chosen for this purpose. The same address location is then read out and the enable pins are masked to initiate the shadow logic test. This back-to-back write and read operations sensitize the gates driven by the memory data out port. Disabling the enable pins of the memory ensures that the off-path input pins (memory fan-out) do not change values during the shadow logic test. Figure 3.10 illustrates a custom benchmark circuit used to verify the memory test feature. The logic clouds in the picture represents random logic. The clouds on the functional and test address and enable paths are identical. The fanout logic clouds on the output ports of the memory array are chosen differently to cover all logic gate types. The logic clouds not interacting with the memory periphery are random as well. These logic clouds can be modified to model different depth of logic, logic cone width, etc.

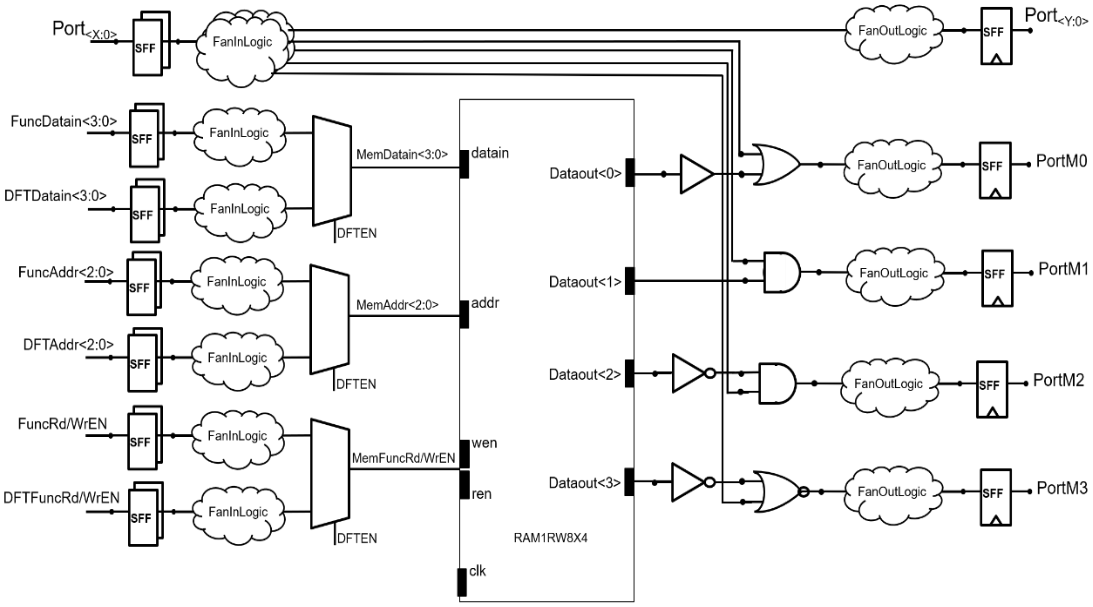


**Figure 3.10 Synthetic Memory Benchmark Circuit**

The circuit contains one instance of an 8X4 1RW memory. The paths that are generated for delay test can be categorized into three groups. One set of paths traverse from a subset of Port<X:0> through the fan-logic to Port<Y:0> ports. These paths do not interact with the memory array. The remainder of the paths are generated from the functional and test ports of the memory array, through the memory array, to PortM ports in the fanout logic. The third category of paths start at input ports and traverse through the gates which are connected to the memory data ports, to end at the output ports of the circuit. In the absence of a CRAM model, the paths that interact with the memory boundary cannot be generated. The RAM sequential mode is used to cover the category 2 paths and the output shadow logic mode is used to sensitize the gates in the memory array's output ports to cover the category 3 paths.

### 3.2.4. Clock Domain Crossing

Several clocking domains can exist in a circuit. We assume a synchronous clocking scheme in our work in which two clock domains can differ in their frequencies, but the rising edges are synchronized. The path generation in KLPG so far assumed only one clock domain. All the SFFs are assumed to be clocked by the same clock. We implement multiple clock domains by using a clock-attribute file that is read during the RTL parsing step. This file lists the SFFs per clock domain. In industrial circuits, individual clock domains can be disabled with the help of clock gating for power saving and debug purposes. Clock gating information is also captured in the clock-attribute file. We assume Integrated Clock Gating (ICG) cells in the design.

Path generation in a multi-clock circuit can be addressed in various ways. One approach is to generate paths within the same clock domain. In this approach, separate tests are run on the different clock domains in the circuit. Any paths with Clock Domain Crossing (CDC) cannot be tested using this approach. Test generation methods for CDC verification have been presented in the literature [28][29].

The path generation procedure used in our work remains unchanged from the traditional KPG algorithm described earlier. We introduce the concept of clock domains for all the SFFs in the design. In order to test CDC, the clock domain attribute needs to be used while generating at-speed and *coda* cycle paths. We assume that the static timing convergence across all the clock domains is ensured by design construction.

The CDC mode of path generation in this work is divided into two parts. The first part involves path generation and SFF marking for viable conversion into DFF without impacting the ability to generate inter clock domain paths. The second part involves clock tree justification.

We illustrate the CDC mode of path generation using the Figure 3.11. The synthetic circuit in this figure has three clock domains named CLK_1, CLK_2 and CLK_3. We assume CLK_1 and CLK_3 domains are integer multiple faster than the CLK_1 domain. Three kinds of clock domain crossing are modeled in this example. CLK_1 → CKK_2, CLK_1 → CLK_3 and CLK3 → CLK_2.

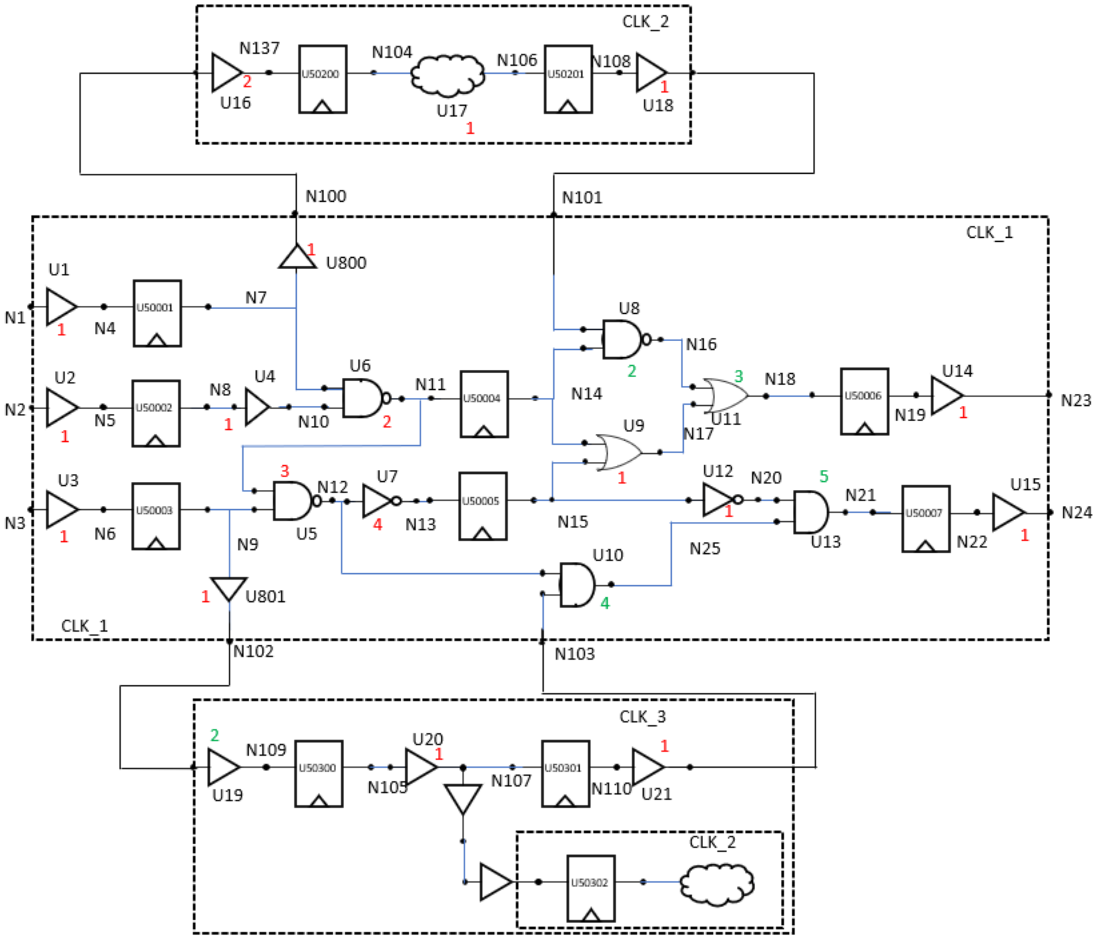


**Figure 3.11 Path Generation in CDC Scenario**

The clock tree paths for these domains are depicted in Figure 3.12. The AND gates U100, U101 and U102 are the ICG cells. The clock-pin for the ICG cells is connected to the clock grid of the clock tree. On-chip PLLs and clock dividers can generate these clocks. The enable-pin of the ICG cells is driven from the clock-gating logic of these individual clock domains. The ↯ mark is the figure signifies the clock justification issue elaborated below.

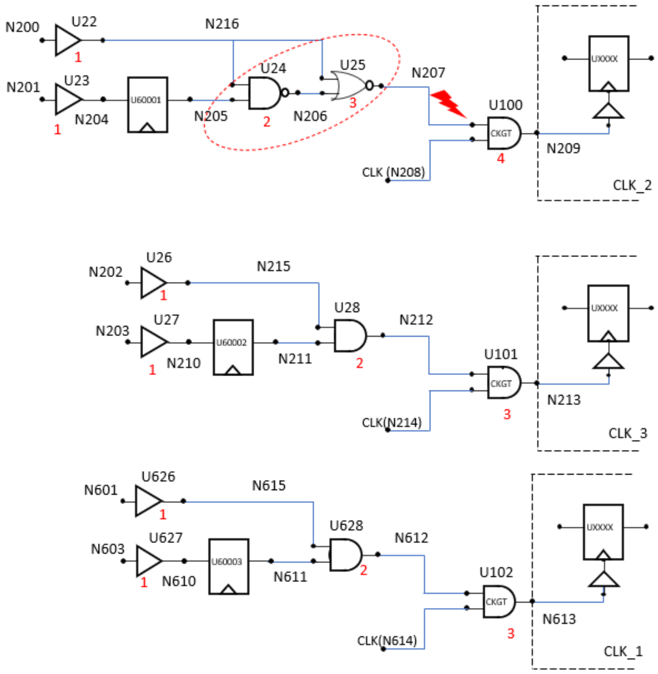


**Figure 3.12 Clock Path Justification**

The clock path justification is run first to determine if a path can be launched or captured in a clock domain. We perform a logic topology check on the fan-in cone of the ICG cells to determine the justifiability of the clock domain controlled by the ICG cell. In Figure 3.12, clock domain CLK_1 and CLK_3 can be justified. U60002 and U60003 drive the clock-enable (CE) signals for these two domains respectively. Assuming that U28 and U628 can be sensitized to propagate the test mode CE signal to the ICG inputs, these two clock domains can be enabled in test mode. For clock domain CLK_2, clock justifiability fails due to the presence of the logical conflict in the CE logic cone. The circuit topology encircled with the dashed red line prohibits the propagation of the CE signal driven by U60001 through the gates U24 and U25 to the ICG input. This logic cone is an oversimplified depiction of real-life scenarios. A multi-clock domain circuit may be designed with two mutually exclusive clock-domains in the functional mode. Any CDC between such domains would be functionally invalid. Hence, generating cross-domain paths in the test mode between those clock domains must be prohibited. The clock path justification would indicate the mutual exclusivity between the two domains. The path generation procedure that follows the clock justification step can be guided with this information.

An additional set of SFF conversion heuristics is developed to handle the CDC scenario. We do not use synchronizers in our example. The conversion heuristics ensure that the first SFF reachable in a clock domain is never converted into a DFF. This ensures that the cross-domain data is properly captured in the receiving domain to maintain intended functionality of the circuit. In the example circuit, U50200 and U50300 will be protected from conversion. When the SFF conversion procedure is run individually in each of the domains, U50200 and U50300 get marked as launch-only FFs. In industrial circuits, more complex fan-in logic may exist for these SFFs. If these SFFs are not marked as the launch-only FFs, the intra-domain conversion process may convert them into DFFs. It is essential to maintain the correct sequence of the SFF conversion in the CDC scenario. We assign higher priority to the CDC mode in the multi-round SFF conversion step. Even if the intra-domain conversion would have converted any of those SFFs, the reconversion steps run at the end of the SFF conversion process would convert these DFFs back to SFFs. This priority can be controlled using configuration settings in the ATPG engine to allow various design intents. Although the implementation can handle any number of clock domains in a design, we have verified the implementation with three clock domains in our experiments.

The SFF conversion heuristics can be configured independently for the above-mentioned scenarios. It helps configure the ATPG engine in a flexible manner.

# 4. EXPERIMENTAL RESULTS

The KLPG algorithm and the SFF conversion heuristics are implemented in C++. Experiments have been run on the ISCAS89 and ITC99 benchmark circuits on an Intel Core i7 machine. Multiple values of K and number of *coda* cycles (#CYC) have been used in the experiments to characterize the implementation. We assume that the PIs are held constant and the value on the POs are ignored in our experiments, as in a low-cost production tester. The values driven by the DFFs are assumed to be unknown. This is conservative, since simulation of the test scan-in cycles would set most DFFs to known values. Robust path sensitization criterion is used in all the experiments. We used the Skywater 130nm open source PDK and cell library to analyze the leakage power benefit of SFF conversion in the benchmark circuits. We developed synthetic benchmark circuits to validate our memory array test and CDC scenarios.

## 4.1. Experimental Results on the Topological Heuristics

We use the ISCAS89 benchmark circuit s5378 to illustrate the findings in the topological analysis of the benchmark circuits. Figure 4.1 illustrates the path profile in this circuit.

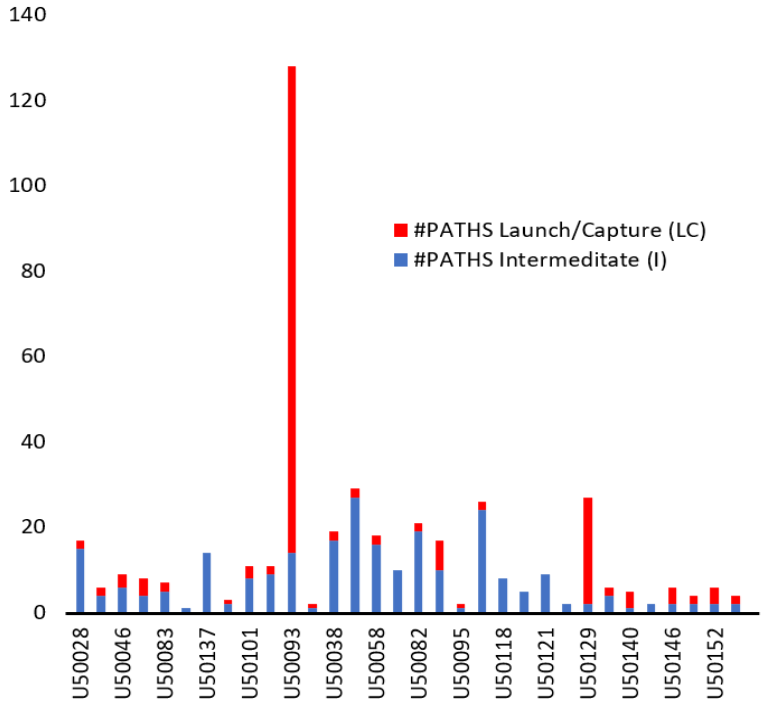


**Figure 4.1 Path Profile in s5378 Benchmark Circuit**

The red bars represent the number of paths either launched or captured by an SFF (LC). The blue bars represent the number of paths where an SFF is present as an intermediate SFF in a multi-cycle path (I) when we apply one or more *coda* cycles. The value of LC is used to rank the SFFs in the design to determine the impact of converting the SFFs on the test coverage. The ratio of the two parameters ($H_R$=LC/I) acts as a guiding factor in the conversion process that we explain later in this section. It can be observed that only a small number of the SFFs have a high LC value. A similar trend is observed in the other benchmark circuits.

We observe different topological trend on different SFFs in the circuit. U50093 has a very large $H_R$ value. Converting this SFF into DFF would result in a large coverage loss. The number of paths captured by this SFF as the final SFF in a path drops to one in one *coda* cycle and to zero in two *coda* cycles. For all the higher number of *coda* cycles scheme, this SFF always appears as an intermediate SFF. Hence, the coverage loss resulting from converting this SFF into DFF can be fully regained by using the OBS driven path extension. U50048 shows a different trend. This SFF has a low value of $H_R$. The OBS driven path extension would not yield a lot of benefit if this SFF is converted into a DFF.

The graph in Figure 4.2 shows the coverage loss as a function of the percentage of the DFFs in the design for s5378. Two subsets of the FFs in the design are used to derive the graph. The FF subsets are derived from the ranking of the SFFs in the design. The rank is defined as the number of at-speed paths captured by the SFF in the at-speed cycle. "FF Set 1" contains the SFFs with fewest at-speed paths captured while "FF Set 2" contains the SFFs with the most at-speed paths captured.

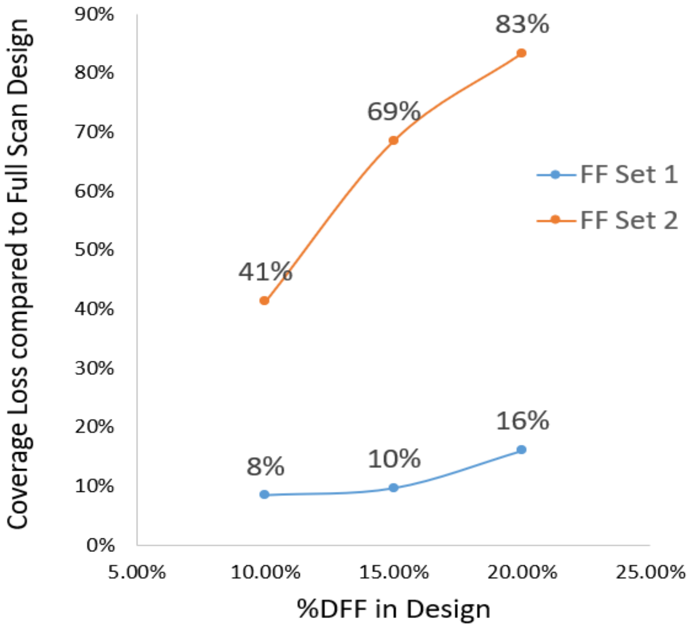


**Figure 4.2 Coverage Loss with Varying DFF%**

Coverage loss is defined as the percentage of the testable paths becoming *false* paths as a result of the SFF to DFF conversion. For both SFF sets, the coverage loss increases with an increasing percentage of DFFs in the design. The relative difference in the coverage loss between these two sets is quite significant. A similar trend is observed in the other benchmark circuits. The sensitivity of the coverage loss varies across all the benchmark circuits because it is topology dependent.

The three data points on the X axis represents 10%, 15% and 20% DFFs in the design. The coverage loss varies between 41% and 83%. In practical circuits, only a very small amount of coverage loss is acceptable. If the sensitivity of the curve is quite high, a small increase in the SFF conversion percentage can greatly impact the coverage loss. A flatter curve on the other hand would result in incremental coverage loss for a much larger additional SFF conversion percentage, which can yield much better PPA benefit. It should be noted that OBS driven path extension needs to be used to bring down the coverage loss in each of these scenarios for the SFF conversion result to be viable. Since the efficacy of the OBS driven path extension depends on the circuit topology, the attainable PPA benefit also is tied to the circuit topology.

Figure 4.3 illustrates the dependency of the attainable SFF conversion percentage on the heuristic threshold used during the conversion process.

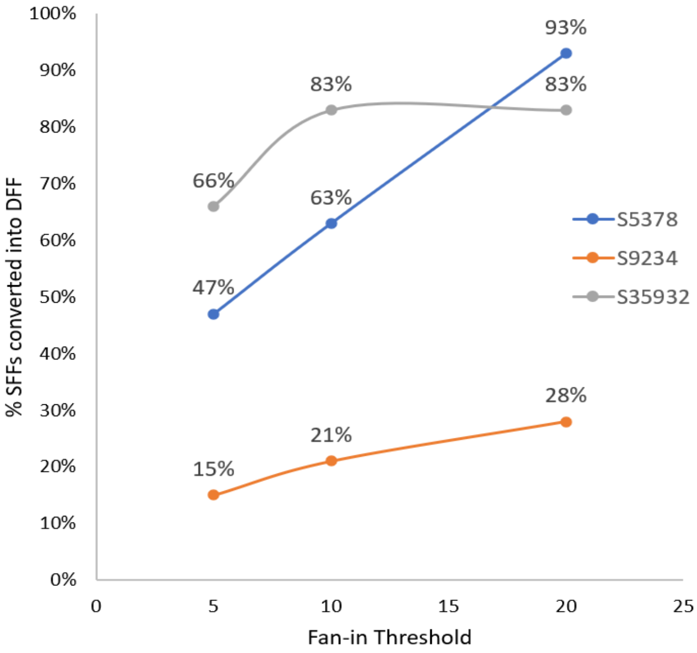


**Figure 4.3 SFF Conversion Statistics**

The percentage of the converted SFFs increases with larger fan-in threshold for all the benchmark circuits in this figure. A similar trend is observed on other benchmark circuits. The sensitivity of the SFF conversion percentage is much higher for s5378 compared to s9234 and s35932. The absolute percentage of the conversion also varies quite drastically among these circuits. The graph for s9234 is quite flat. Only 28% of the SFFs could be converted with the fan-in threshold of 20. For the same threshold, we could convert close to 90% of the SFFs in the other two circuits. It should be noted that no SFF exclusions are performed to derive this sensitivity data. And the threshold for the acceptable coverage loss is set to an arbitrarily large value. In practical application, the upper bound of the SFF conversion percentage would most likely be clamped by the coverage loss threshold before the upper bound of the fan-in threshold is reached.

Figure 4.4 shows the result of multi-round conversion process on  s5378 by incorporating the SFF exclusions described earlier in chapter 3.

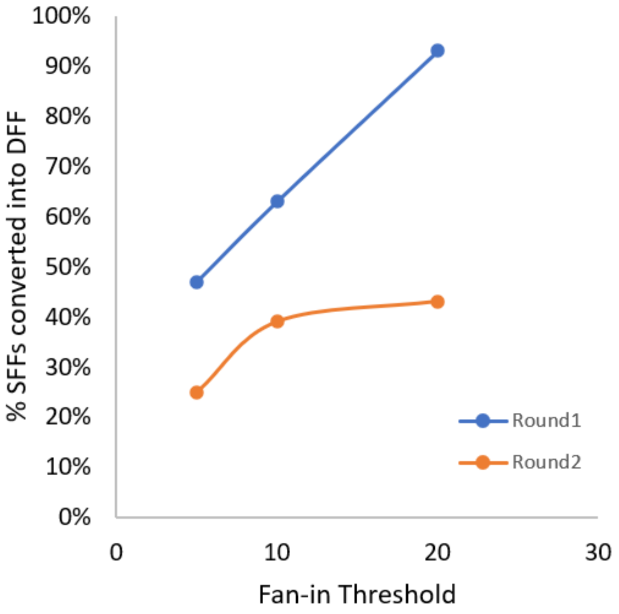
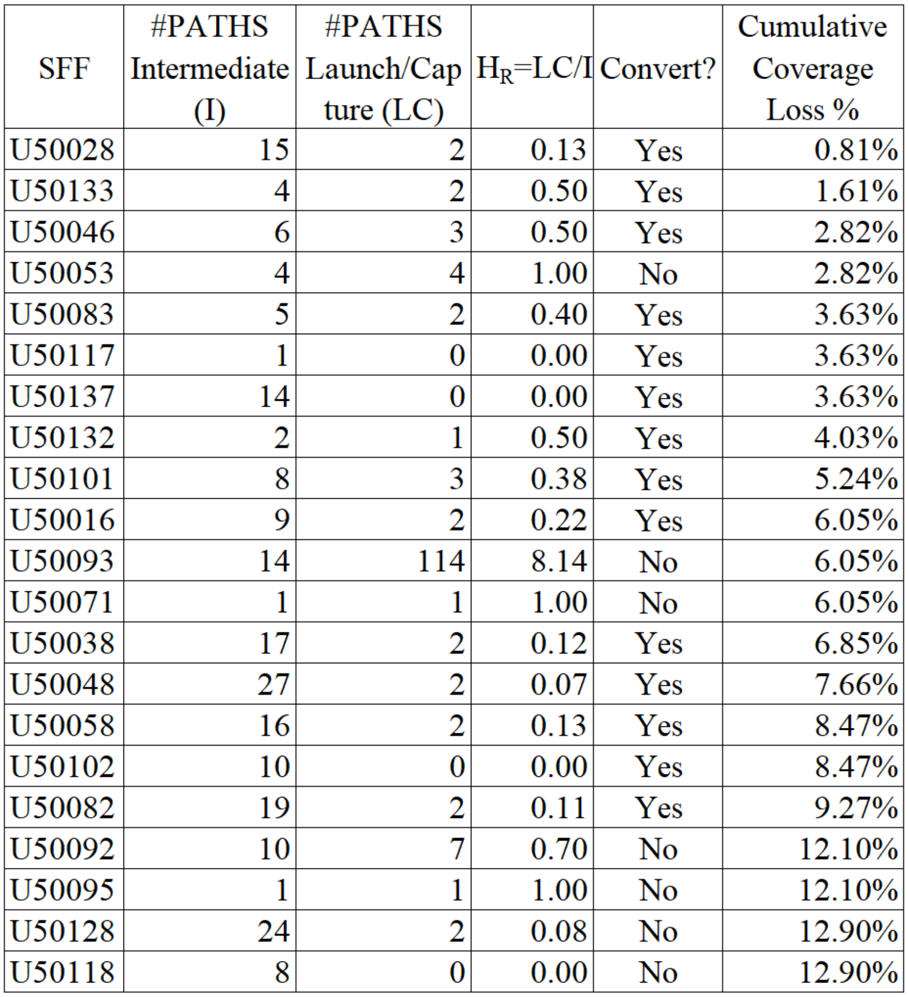


**Figure 4.4 Multi-Round Conversion in s5378**

The round 1 conversion trend is the same as the one shown in Figure 4.3. When we run the SFF reconversion step in round 2, the overall SFF conversion percentage comes down compared to round 1 and saturates over the multiple fan-in threshold values.

We now elaborate on the PBA analysis. We use the data from s5378 captured in Table 4.1 for the explanation. A subset of the SFFs in this circuit is captured in this table for illustration purpose. Column 2 lists the number of paths on which an SFF is present as the intermediate FF when *coda* cycles are used (I) . Column 3 lists the number paths an SFF is present as a launch or capture FF (LC) . Column 4 lists the heuristic parameter $H_R$ which is defined as the ratio of LC and I. Column 5 captures the decision of the conversion on the SFF based on the value of $H_R$. The final column lists the cumulative coverage loss as a result of the SFF conversion up to the SFF listed in a row.

**Table 4.1 SFF Conversion Matrix with One Coda Cycle Clocking Method**

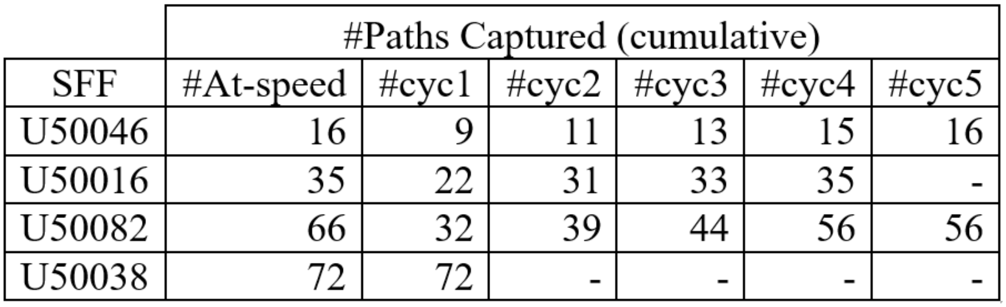


| SFF | #PATHS Intermediate (I) | #PATHS Launch/Capture (LC) | $H_R$=LC/I | Convert? | Cumulative Coverage Loss % |
|---|---|---|---|---|---|
| U50028 | 15 | 2 | 0.13 | Yes | 0.81% |
| U50133 | 4 | 2 | 0.50 | Yes | 1.61% |
| U50046 | 6 | 3 | 0.50 | Yes | 2.82% |
| U50053 | 4 | 4 | 1.00 | No | 2.82% |
| U50083 | 5 | 2 | 0.40 | Yes | 3.63% |
| U50117 | 1 | 0 | 0.00 | Yes | 3.63% |
| U50137 | 14 | 0 | 0.00 | Yes | 3.63% |
| U50132 | 2 | 1 | 0.50 | Yes | 4.03% |
| U50101 | 8 | 3 | 0.38 | Yes | 5.24% |
| U50016 | 9 | 2 | 0.22 | Yes | 6.05% |
| U50093 | 14 | 114 | 8.14 | No | 6.05% |
| U50071 | 1 | 1 | 1.00 | No | 6.05% |
| U50038 | 17 | 2 | 0.12 | Yes | 6.85% |
| U50048 | 27 | 2 | 0.07 | Yes | 7.66% |
| U50058 | 16 | 2 | 0.13 | Yes | 8.47% |
| U50102 | 10 | 0 | 0.00 | Yes | 8.47% |
| U50082 | 19 | 2 | 0.11 | Yes | 9.27% |
| U50092 | 10 | 7 | 0.70 | No | 12.10% |
| U50095 | 1 | 1 | 1.00 | No | 12.10% |
| U50128 | 24 | 2 | 0.08 | No | 12.90% |
| U50118 | 8 | 0 | 0.00 | No | 12.90% |

The SFF conversion heuristic uses the ratio ($H_R$=LC/I) as threshold to determine whether an SFF should be converted into a DFF. The value of $H_R$ acts as a guide to anticipate the potential coverage loss. A threshold of 0.5 is used to generate this table. The first three SFFs in the table have an $H_R$ value that is not greater than the threshold. As a result, these three SFFs are marked for conversion. The cumulative coverage loss at this point stands at 2.82%. The value of $H_R$ for U50053 is 1.0 which is greater than the threshold. As  a result, this SFF is not marked for conversion. The value of the cumulative coverage loss remains unchanged. We use a threshold of 10% for the cumulative coverage loss for the purpose of illustration. The cumulative coverage loss grows to 9.27% after U50082 is marked. The cumulative coverage loss grows beyond 10% if any other SFFs are converted into DFFs. As a result, when U50092 is evaluated, although its $H_R$ value is less than the threshold of 0.5, it is not marked. None of the subsequent SFFs are marked since the cumulative coverage loss has already surpassed the threshold. The conversion scheme can be further optimized if the SFFs are ranked according to their individual impact to the test coverage. A higher percentage of ranked SFFs can be converted.

Combining the PBA scheme with the SCOAP based metrics and using the multi-cycle path extension can maximize the SFF conversion percentage while limiting coverage loss. Table 4.2 shows the cell capture statistics for a set of SFFs in the s5378 circuit. These SFFs were chosen for representation purposes only. The second column shows the number of at-speed paths that are captured by the corresponding SFFs in a full-scan design. The SFFs in the first column are then converted into DFFs.

**Table 4.2 Path Capture Statistics in s5378**



| | #Paths Captured (cumulative) | | | | | |
|---|---|---|---|---|---|---|
| SFF | #At-speed | #cyc1 | #cyc2 | #cyc3 | #cyc4 | #cyc5 |
| U50046 | 16 | 9 | 11 | 13 | 15 | 16 |
| U50016 | 35 | 22 | 31 | 33 | 35 | - |
| U50082 | 66 | 32 | 39 | 44 | 56 | 56 |
| U50038 | 72 | 72 | - | - | - | - |

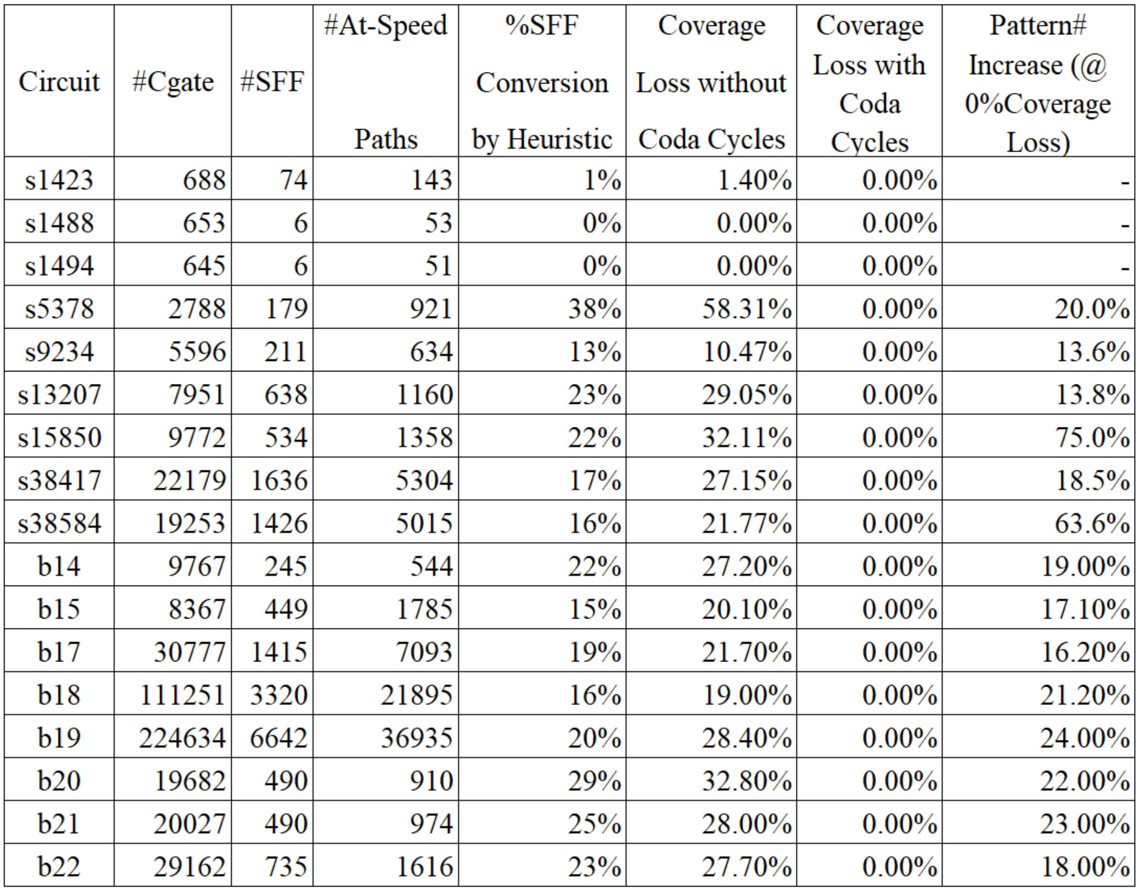
The at-speed paths become false paths resulting in coverage loss. We then apply different numbers of coda cycles to reclaim the coverage. The subsequent columns show the cumulative number of paths that could be extended over the number of coda cycles from the corresponding DFFs to SFFs. We see varied results across the FFs. For U50038, a single coda cycle was sufficient to reclaim all the lost coverage. For U50046, five coda cycles were required to reclaim the coverage. For U50082, 10 paths could not be extended to SFFs even over five coda cycles. More coda cycles for this FF did not yield a better result.

The data in Table 4.2 was generated with K=2 during path generation. A total of 921 paths could be generated in the at-speed cycle of the full-scan design. We could convert 47% of the SFFs in the design with a fan-in threshold setting of 5. The PBA analysis prevented conversion of 24 SFFs into DFFs because the multi-cycle clocking method could not recover the coverage that would be lost if those SFFs were converted. After the multi-round conversion, we were able to achieve 34% SFF conversion at no coverage loss.

As alluded before, the efficiency of the SFF conversion scheme is dependent on the circuit topology. There exists a tradeoff between the SFF conversion percentage yielding to PPA benefit and the allowable coverage loss in a circuit. A higher value of allowable coverage loss can yield a much better PPA benefit. The optimal point in this tradeoff can be quite different from one circuit to another.

Table 4.3 summarizes the results on ISCAS89 and ITC99 benchmark circuits. This table illustrates the topological dependency of the conversion efficiency and corresponding coverage impact on the reported circuits.

**Table 4.3 Tradeoff between SFF Conversion and Test Coverage Loss**



| Circuit | #Cgate | #SFF | #At-Speed Paths | %SFF Conversion by Heuristic | Coverage Loss without Coda Cycles | Coverage Loss with Coda Cycles | Pattern# Increase (@ 0%Coverage Loss) |
|---|---|---|---|---|---|---|---|
| s1423 | 688 | 74 | 143 | 1% | 1.40% | 0.00% | - |
| s1488 | 653 | 6 | 53 | 0% | 0.00% | 0.00% | - |
| s1494 | 645 | 6 | 51 | 0% | 0.00% | 0.00% | - |
| s5378 | 2788 | 179 | 921 | 38% | 58.31% | 0.00% | 20.0% |
| s9234 | 5596 | 211 | 634 | 13% | 10.47% | 0.00% | 13.6% |
| s13207 | 7951 | 638 | 1160 | 23% | 29.05% | 0.00% | 13.8% |
| s15850 | 9772 | 534 | 1358 | 22% | 32.11% | 0.00% | 75.0% |
| s38417 | 22179 | 1636 | 5304 | 17% | 27.15% | 0.00% | 18.5% |
| s38584 | 19253 | 1426 | 5015 | 16% | 21.77% | 0.00% | 63.6% |
| b14 | 9767 | 245 | 544 | 22% | 27.20% | 0.00% | 19.00% |
| b15 | 8367 | 449 | 1785 | 15% | 20.10% | 0.00% | 17.10% |
| b17 | 30777 | 1415 | 7093 | 19% | 21.70% | 0.00% | 16.20% |
| b18 | 111251 | 3320 | 21895 | 16% | 19.00% | 0.00% | 21.20% |
| b19 | 224634 | 6642 | 36935 | 20% | 28.40% | 0.00% | 24.00% |
| b20 | 19682 | 490 | 910 | 29% | 32.80% | 0.00% | 22.00% |
| b21 | 20027 | 490 | 974 | 25% | 28.00% | 0.00% | 23.00% |
| b22 | 29162 | 735 | 1616 | 23% | 27.70% | 0.00% | 18.00% |

Column 1 lists the benchmark circuit. Column 2 lists the number of combinational gates. Column 3 lists the original number of SFFs. Column 4 lists the number of at-speed paths generated with K=2, for full-scan circuits. Column 5 lists the percentage of SFFs that could be converted to DFF with the fan-in threshold set to 5. Feedback loops are protected from conversion in these experiments. SFF conversions that result in paths that cannot be recovered by any of the clocking methods are also excluded as a part of the PBA step. Column 6 lists the coverage loss resulting from these SFF conversions, assuming no coda cycles. The coverage loss is defined as the percentage of the paths in column 4 that cannot be observed as a result of the SFF to DFF conversion. Column 7 shows the coverage loss after we apply the coda cycles.

The percentage of the SFFs that could be converted varies between 13% and 38% amongst these benchmark circuits except for the benchmark circuits s1423, s1488 and s1494. These circuits have very few SFFs. The circuit topology of s1423 prohibits any SFF conversion. These three circuits are excluded from all subsequent analysis.

The coverage loss as a result of the SFF conversion varies between 10.47% and 58.31%. We could recover the entire coverage loss on all of these benchmark circuits by applying anywhere between 1 to 3 *coda* cycles. Hence, the 13-38% SFFs could be converted into DFFs without any coverage impact. Column 8 shows the increase in the pattern count for the 0% coverage impact. The increase in the pattern count is dictated by the unknown values driven by the DFFs in the circuit that prohibits the ATPG from compacting multiple path delay tests into one test pattern. As previously noted, assuming unknown values is very conservative.

The CPU time to run the FF conversions is only a few seconds for even the largest benchmark circuits.

Figures 4.5 and 4.6 show the pattern count increase as a function of the percentage of the SFFs that could be converted in a benchmark circuit.

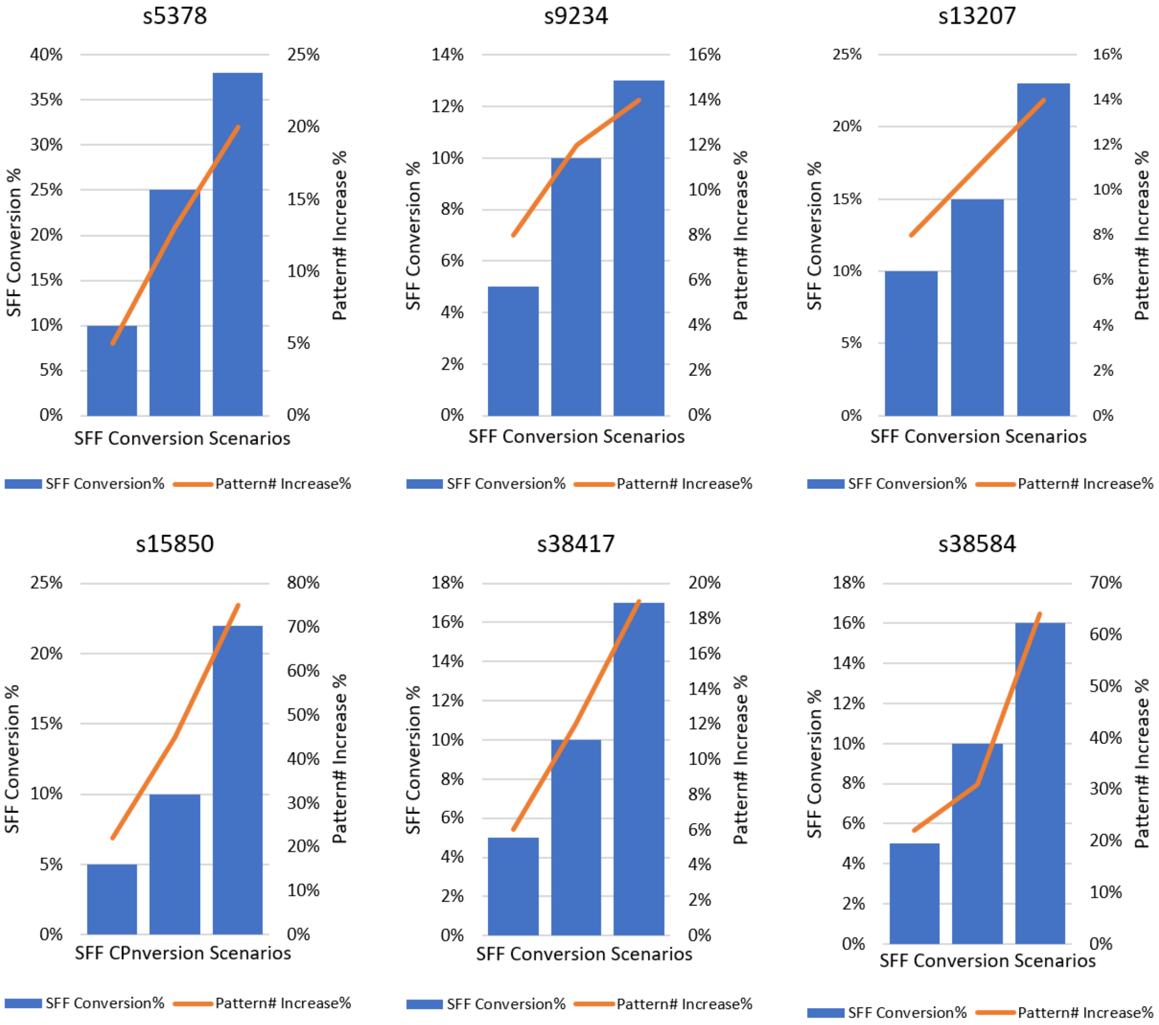


**Figure 4.5 Tradeoff between SFF Conversion% and Pattern# Increase in ISCAS89 Benchmark Circuits**

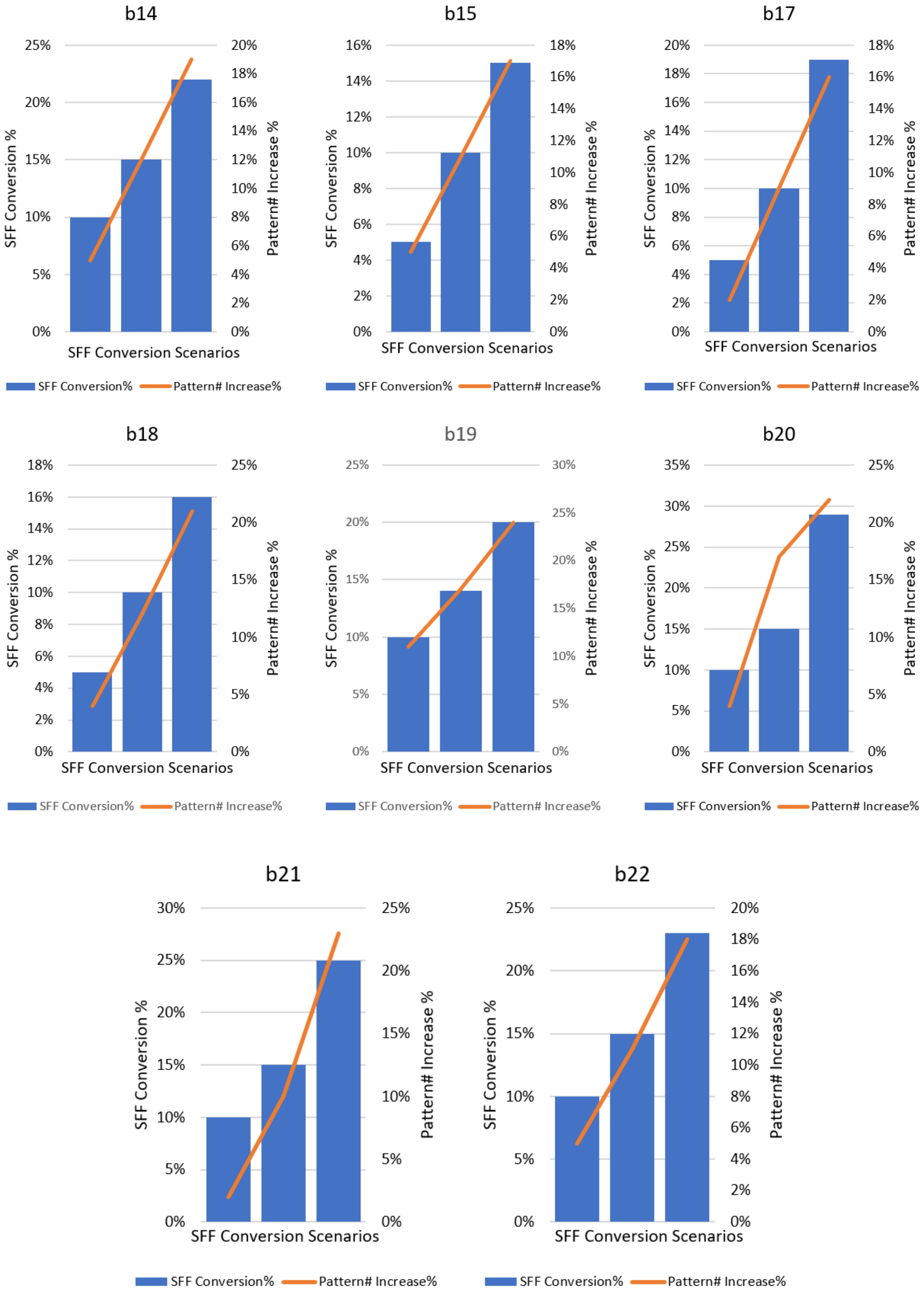


**Figure 4.6 Tradeoff between SFF Conversion% and Pattern# Increase in ITC99 Benchmark Circuits**

All the benchmark circuits show a similar trend in these two figures. The pattern count increases in s15850 and s38584 are higher compared to the other circuits. This deviation can be attributed to the topological dependency of the test pattern compaction process. Due to the unique fan-out configuration in a circuit, a larger pool of unique test patterns may be required to test all the multi-cycle paths.

The pattern count increase reported in these figures is for maintaining 100% test coverage. A tradeoff exists between the pattern count increase and the SFF conversion percentage. The pattern count increase can be limited by constraining the SFF conversion target in a circuit.

For comparison, the work in [8] reports test coverage loss as a result of the SFF cell conversion on multiple designs. A design with 35% DFFs is reported to have transition delay test coverage of 90.25%. The associated pattern count increase is also reported to be almost double the pattern count used in the full scan design.

Results presented in this research shows that the multi cycle path generation mechanism was able to maintain 100% test coverage for the path delay tests for all the benchmark circuits with an average pattern count increase of 26%. A maximum of 38% SFF conversion could be achieved at a pattern count increase of 20% in our experiments.

The SFF conversion percentage scales well over all the benchmark circuits. The topological heuristics introduced in this work analyze the gates present in one sequential pipeline stage, which allows it to scale well across all the benchmark circuits with different gate counts.

We now present leakage power improvement from the SFF conversion for the larger benchmark circuits in Figure 4.7. The selected circuits cover the SFF conversion range described in Table 4.3. We used the Skywater 130nm open source PDK and cell library to analyze the leakage power benefit of converting the SFFs into DFFs. The leakage power benefit reported in this study results from swapping the SFFs with the DFFs. We do not perform re-placement and re-routing of the design after the SFF conversion step in a Place-and-Route (PNR) tool. DFF cells have smaller foot-print and smaller stage delay. The timing slack on the data-path increases when an SFF on a data-path is replaced with a DFF. This additional slack can be traded for lowering the power of the circuit by using higher-VT transistors on the data-path or by using smaller drive strength cells to achieve lower dynamic power.

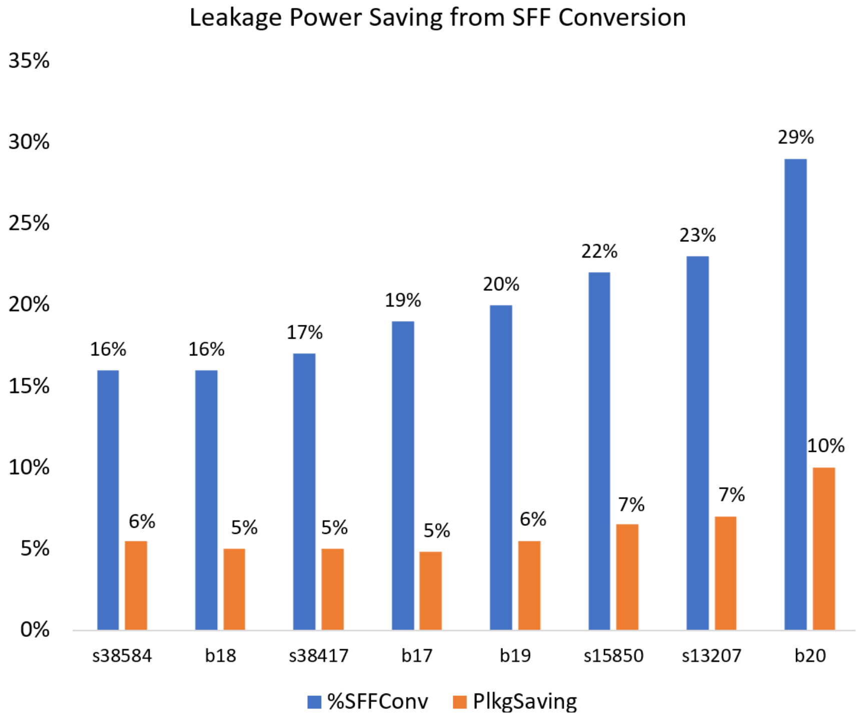


**Figure 4.7 Leakage Power Saving Statistics**

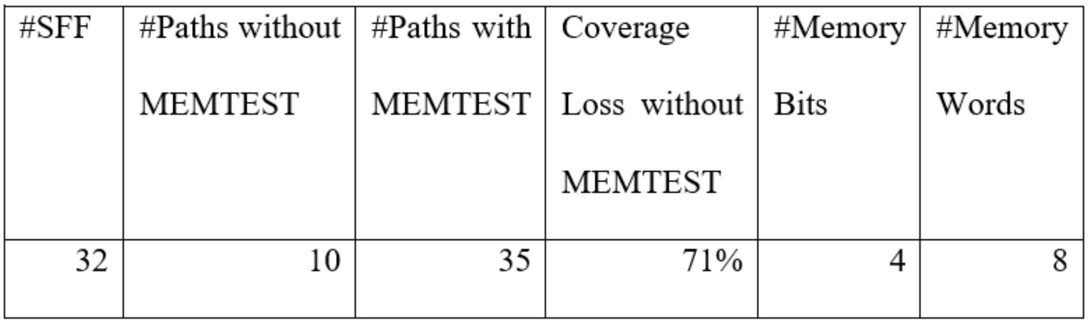
We were able to achieve 5%-10% reduction in leakage power saving from the SFF conversion on the benchmark circuits, which is quite significant. It should be reiterated that this power savings comes at no cost to the test coverage in these circuits. Hence, we are able to maintain the quality of the test for these circuits during the SFF conversion process.

The leakage power saving is directly proportional to the percentage of the leakage power contributed by the SFFs in these circuits. As a result, the leakage power saving can be identical across multiple circuits with varied amount of SFF conversion. We achieved 5% leakage power improvement on b17, b18 and s38417 while the SFF conversion percentage was between 16% and 19% for these circuits. On the other hand, we could achieve much higher percentage of leakage power saving with comparable or higher value of the total number of gates per SFF (GC/SFF) in a circuit. The sizes (total number of gates) of b20 and s38417 are comparable. The leakage power of s38417 is however much larger compared to b20. This is due to the logic gate composition in these circuits and the relative leakage power positioning of the gates in the PDK. As a result, the SFF leakage power contribution is much higher in b20 than s38417. We achieved 10% leakage power reduction for b20 compared to 5% in s38417, even with a higher value of GC/SFF for b20.

### 4.2. Memory Test Results

Table 4.4 summarizes the results obtained on the synthetic benchmark circuit incorporating a memory instance depicted in Figure 3.10.

**Table 4.4 Memory Test Statistics**



| #SFF | #Paths without MEMTEST | #Paths with MEMTEST | Coverage Loss without MEMTEST | #Memory Bits | #Memory Words |
|---|---|---|---|---|---|
| 32 | 10 | 35 | 71% | 4 | 8 |

A majority of the SFFs in this circuit are present in the fan-in cone of the memory array. As a result, the coverage gain obtained by the memory test capability is quite high. It should be noted that the lost coverage in this circuit could be completely recovered with the help of 3 *coda* cycles. A different circuit topology may require a different clocking scheme to recover coverage loss.

### 4.3. Experimental Results on Clock Domain Crossing

We developed synthetic benchmark circuits to validate the CDC implementation in our code. Figure 4.8 depicts the synthetic benchmark that is created by stitching three instances of the s27 benchmark circuit. This circuit consists of three clock domains marked in the figure. The logic clouds in the figure are used to provide connectivity between the output of clock domain 1 to the inputs of the clock domains 2 and 3 respectively. This circuit has CDC paths between clock domains 1, 2 and 1, 3.  This oversimplified circuit is used for representation purpose only. Other experiments have been run on synthetic benchmark circuits that have been created by stitching multiple instances of s1423 and s5378 respectively.

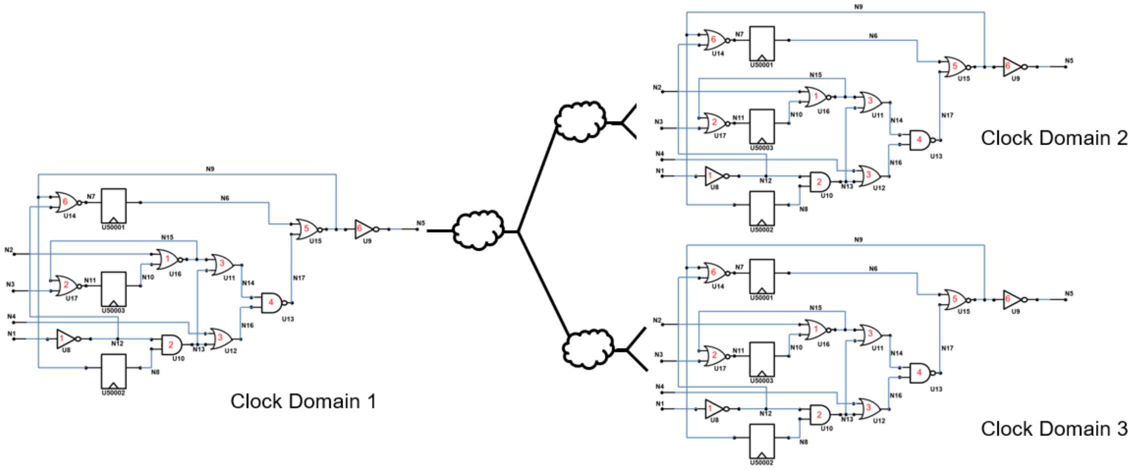


**Figure 4.8 Synthetic Benchmark Circuit with Multiple Clock Domains**

We developed verbose logging for the CDC test mode to prove the correctness of the implementation. We verified the intended behavior of the CDC mode by reviewing the converted list of the SFFs against the constraints of the CDC mode conversion described earlier in this work.

# 5. CONCLUSIONS AND FUTURE WORK

We presented a topological approach to convert SFFs into DFFs in a full-scan circuit. The SFF conversion results in a PPA benefit. We quantified the leakage power saving obtained by the SFF conversion process on the ISCAS89 and ITC99 benchmark circuits. The percentage of SFFs that can be converted in a circuit with a specified level of coverage loss is circuit topology dependent. We present SFF conversion percentage on all the benchmark circuits at zero coverage loss. Observability driven multi-cycle path extension was used to reclaim the lost coverage on the paths that terminate at a DFF. The goal of the observability driven path extension is to make sure that the most observable branch of a fan-out cone can be extended to an SFF. The primary benefit of using observability is that it minimizes the sensitization requirements of the selected path in the coda cycle from interfering with the sensitization of the at-speed test. The results showed that 100% test coverage could be maintained by using one to three *coda* cycles. We observe that some paths remain untestable, due to the inability to generate a path delay test, due to X values in the DFFs, rather than the inability to find a coda path compatible with the path delay test. We will explore whether test initialization sequences can be applied during *preamble* cycles or scan shift to minimize the number of X values in the DFFs, and to determine whether selective DFF set or reset could recover most of the fault coverage.

As part of the future work, both long and short paths can be considered by heuristics during the SFF ranking and conversion process. More properties of the SFF can also be considered while developing the conversion heuristics. In the current work, we focused on the leakage power aspect of the PPA optimization. Incorporating the SFF conversion step upstream in the synthesis, place and route flow can yield more benefits. Using a vendor PNR tool to quantity the PPA benefit in the future can provide more insight into the tradeoffs of the SFF conversion process. The memory array test and the CDC solutions should be run on more standard benchmark circuits as a part of the future work.

# REFERENCES

[1] Intel Corp., “Over 50 Years of Moore’s Law,” https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html. Retrieved February 2022.

[2] 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.

[3] Chakraborty, Avijit (2015). Observability Driven Path Generation for Delay Test. Master's thesis, Texas A & M University.

[4] A. Chakraborty and D. M. H. Walker, "Observability Driven Path Generation for Delay Test Coverage Improvement in Scan Limited Circuits," *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020, pp. 1-4, doi: 10.1109/DFT50435.2020.9250797.

[5] Laung-Terng Wang, Charles Stroud, Nur Touba “System-on-Chip Test Architectures, 1st Edition Nanometer Design for Testability”, Elsevier.

[6] Paul Theo Gonciari, Bashir M. Al-Hashimi, Nicola Nicolici “Improving Compression Ratio, Area Overhead, and Test Application Time for System-on-Chip Test Data Compression/Decompression”, Springer Netherlands.

[7] T. McLaurin and I. P. Lawrence, “Improving Power, Performance and Area with Test: A Case Study,” IEEE International Test Conference, Phoenix, AZ, USA, 2018, pp. 1-10, doi: 10.1109/TEST.2018.8624752.

[8] Dong Xiang, Yi Xu and H. Fujiwara, “Nonscan Design for Testability for Synchronous Sequential Circuits Based on Conflict Resolution,” IEEE Transactions on Computers, vol. 52, no. 8, pp. 1063-1075, Aug. 2003, doi: 10.1109/TC.2003.1223640.

[9] T. Cheng, “Test Generation for Delay Faults in Non-Scan and Partial Scan Sequential Circuits,” IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, USA, 1992, pp. 554-559, doi: 10.1109/ICCAD.1992.279313.

[10] Skywater, “130nm Open Source PDK,” https://github.com/google/skywater-pdk, 2020.

[11] Tehranipoor, Mohammad, Peng, Ke, Chakrabarty, Krishnendu, "Test and Diagnosis for Small-Delay Defects", Springer Science+Business Media, LLC 2011.

[12] Pomeranz, I.; Reddy, S.M., "Transition Path Delay Faults: A New Path Delay Fault Model for Small and Large Delay Defects," in Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.16, no.1, pp.98-107, Jan. 2008.

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

[14] Laung-Terng Wang, Charles Stroud, Nur Touba "System-on-Chip Test Architectures, 1st Edition Nanometer Design for Testability", Morgan Kaufmann 2010.

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

[16] VLSI Test Principles and Architectures, Design for Testability, Elsevier, ISBN: 9780123705976.

[17] N. Eén, N. Sörensson, “The MiniSat Page, Introduction”. Retrieved from minisat.se on March 2015.

[18] M. L. Flottes, R. Pires, B. Rouzeyre and L. Volpe, "Scanning datapaths: a fast and effective partial scan selection technique," Proceedings Design, Automation and Test in Europe, Paris, France, 1998, pp. 921-922, doi: 10.1109/DATE.1998.655969.

[19] Kwang-Ting Cheng, "Partial scan designs without using a separate scan clock," Proceedings 13th IEEE VLSI Test Symposium, Princeton, NJ, USA, 1995, pp. 277-282, doi: 10.1109/VTEST.1995.512649.

[20] T. McLaurin and I. P. Lawrence, "Improving Power, Performance and Area with Test: A Case Study," 2018 IEEE International Test Conference (ITC), Phoenix, AZ, USA, 2018, pp. 1-10, doi: 10.1109/TEST.2018.8624752.

[21] T. McLaurin and I. P. Lawrence, “Improving Power, Performance and Area with Test: A Case Study,” IEEE International Test Conference, Phoenix, AZ, USA, 2018, pp. 1-10, doi: 10.1109/TEST.2018.8624752.

[22] Y. Gao, T. Zhang, S. Chakraborty and D. M. H. Walker, "Delay Test of Embedded Memories," *2014 IEEE 23rd North Atlantic Test Workshop*, 2014, pp. 65-68, doi: 10.1109/NATW.2014.22.

[23] A. Singh, D. Bose, and S. Darisala, “Software Based In-System Memory Test For Highly Available Systems,” Proceedings of the 2005 IEEE International Workshop on Memory Technology, Design, and Testing (MTDT’05), pp 89- 94, September 2005.

[24] P. Bernardi, L. Ciganda, M. Reorda, and S. Hamdioui, “SW-based transparent in-field memory testing,” 2015 16th Latin-American Test Symposium, LATS 2015, March 2005.

[25] C. Huang, J. Huang, and C. Wu, “A Programmable Built-In Self-Test Core for Embedded Memories,” Proceedings of the Asia and South Pacific Conference on Design Automation, February 2000.

[26] W. Hong, J. Choi, and H. Chang. “A Programmable Memory BIST for Embedded Memory,” SoC Design Conference, December 2008.

[27] Y. K. Malaiya and R. Narayanaswamy, Testing for timing faults in synchronous sequential integrated circuits, in Proc. Int. Test Conf., pp. 560–571, October 1983.

[28] N. Karimi, Z. Kong, K. Chakrabarty, P. Gupta and S. Patil, "Testing of Clock-Domain Crossing Faults in Multi-core System-on-Chip," *2011 Asian Test Symposium*, 2011, pp. 7-14, doi: 10.1109/ATS.2011.68.

[29] C. Leong *et al.*, "Built-in Clock Domain Crossing (CDC) test and diagnosis in GALS systems," *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 72-77, doi: 10.1109/DDECS.2010.5491815.