PSEDUO FUNCTIONAL PATH DELAY TEST THROUGH EMBEDDED MEMORIES

A Thesis

by

YUKUN GAO

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

MASTER OF SCIENCE

Chair of Committee,	Duncan M. Walker
Committee Members,	Sunil Khatri
	Paul Gratz
Head of Department,	Dilma DaSilva

May2015

Major Subject: Computer Engineering

ABSTRACT

On-chip memory arrays are widely used in systems-on-chip. Prior research has shown that timing critical paths often go through these memories. Embedded memories are typically tested using memory built-in self-test and macro test. However, these techniques have relatively low small delay fault coverage, so functional test must be used to accurately determine maximum operating frequency. In this work we achieve high delay fault coverage by testing the timing critical paths in and out of embedded memories, including the paths in the surrounding logic.

We use our prior work on pseudo functional K longest path per gate test generation and extend it to handle memory test. In pseudo functional test, low-speed preamble cycles are used to stabilize the supply voltage before the at-speed launch and capture cycles. Since the memory cells are non-scan, a value that is captured in the memory must be moved to a scan cell using low-speed coda cycles. This approach tests any path through a non-scan latch. Our approach eliminates the coverage "shadows" around embedded memories and non-scan latches.

We have established a flow to test industrial circuits with embedded memory. Industrial circuits with different size memory arrays are used to justify the efficiency of the flow. Our results demonstrate that we can effectively generate patterns that cover paths in and out of the memories.

DEDICATION

To those who love me

ACKNOWLEDGEMENTS

I would like to thank my advisor and committee chair, Dr. Duncan M. (Hank) Walker for his advice and support throughout my M.S. studies at Texas A&M University. His insights in this particular research area, his technical guidance and spiritual support were invaluable to this work. This dissertation would never have been completed without his advice and encouragement. I owe him lots of gratitude for making my research life enjoyable and rewarding. What I learned from him will benefit my future career.

I am grateful to my committee members, Dr. Sunil Khatriand Dr. Paul Gratz for their valuable suggestions and personal encouragement. I want to thank Dr. Rabi Mahapatra for his advising in my final defense. I would also like to thank Weizhong Chen, Yujia Liu and Tengteng Zhang for their accompany during my life at office.

At the end, thanks for my family for what they have given to me.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1 Delay Testing

During fabrication, physical defects may happen, which will affect circuit functional performance. These defects can be detected with traditional test methods [1][2][3]. However, manufacturing defects that could not cause a functional failure in the circuit, may only affect the timing performance of the circuit. In order to ensure that the manufactured chips work within the specific timing requirement, those small manufacturing defects should be detected by delay tests. It becomes essential to apply delay tests since fabrication processes are becoming much more complex and the system frequency is raised. Delay faults can be divided into global delay fault, which are result from the variations of global process and local delay faults, which are result from the variations of local process. In order to automatically generate test patterns (ATPG) [4][5] and estimation the fault coverage [6] for delay faults, delay fault models [7][8] have been introduced as the abstraction of delay defects. In section 1.2, different kinds of delay fault models are discussed. Among them, the path delay fault (PDF) is efficiency to cover Small Delay Defects (SDDs). In section 1.3, design for test structures and approaches for testing delay defects are described. Section 1.4 and 1.5 described pseudo-functional testing and the idea of K Longest Path Per Gate (KLPG) delay test method.

1.2 Delay Fault Models

Delay fault models are the abstraction of the behaviors of the circuits which have delay defects. In the following section, popular delay fault models such as transition fault model, gate delay fault model, line delay fault model, path delay path model are introduced.

1.2.1 Transition Fault Model

Among different delay fault models, the transition fault(TF) model [7] is the most popular one. It assumes that the delay fault only affects one place in the whole circuit. Two transition faults, a slow-to-rise and a slow-to-fall delay fault, are related to each gate. Thus, the number of transition faults is linear in gate counts of the circuit. Also, it assumes the transition fault effect can be observed through any path to any primary output or pseudo primary output, regardless of the path’s length. With such assumption, transition fault tests don’t need to consider the circuit timing.

In order to generate tests for transition faults [2], stuck-at fault test generation tools could be easily adopted and extended. One vector is firstly applied to initialize the circuit, and another one sensitizes and propagates the fault response to the primary outputs (PO) or pseudo primary outputs(PPO). Thus the test vector of a transition fault test is a vector pair.

Because of the “lazy” nature of transition fault test generator, short paths are chosen to sensitize and propagate the fault response. Thus small delay defects and accumulated delay defects along the path, may be neglected when performing the TF test [9][10]. Also, glitches generated from the fault sites [11] could degrade the test quality.

1.2.2 Gate Delay Fault Model

The gate delay fault model [4][5][6][12] is a quantitative model for delay faults. It assumes the characteristics, such as size and location, and the delays through logic gates, are known. Also, it assumes the defect is lumped on the input or output of the gate. The main advantage of gate delay fault model is that it could model all gates. However, in this model, distributed defects are not considered, also the exact defect size may not be available.

1.2.3 Line Delay Fault Model

The line delay fault model [13][14] is a combination of the transition delay fault model and path delay fault model. It requires generating a rising or falling transition through target line. Thus, the number of fault sites is twice the number of lines in the circuit. One main advantage is that, the distributed and accumulated small delay defects are considered. However, lines on some shorter paths which have hard constrains may not be tested.

1.2.4 Path Delay Fault Model

Compared with fault model listed before, the path delay fault model [8] can cover small delay defects(SDD) and accumulated delay on a path. This model could deal with paths with any size of delay. Typically, a circuit is considered faulty with a path delay fault if any one path is slow for a rising or falling transition. Thus tests for the path delay fault model can be used to detect Small Delay Defects (SDDs) in the circuit. The main constraint of this fault model is that the number of paths in the circuit can be exponentially proportional to the number of gates in the circuit. Thus, it is not practical to cover all the paths in the circuit.

1.3 Scan Based Delay Test

Design-For-Test (DFT) techniques are used to improve the testability of a circuit [15], among which the SCAN design is most popular. In scan design, storage elements such as latches and flip-flops in the circuit are replaced by scan cells. Then scan cells are “stitched” together to form one or more scan chains. In this way, internal storage elements can be accessed in the circuit which are not directly observable without DFT structure. Thus the testability of the circuit is improved. Also the complexity of test generation is significantly reduced.

The idea of scan design is illustrated in Figure 1.

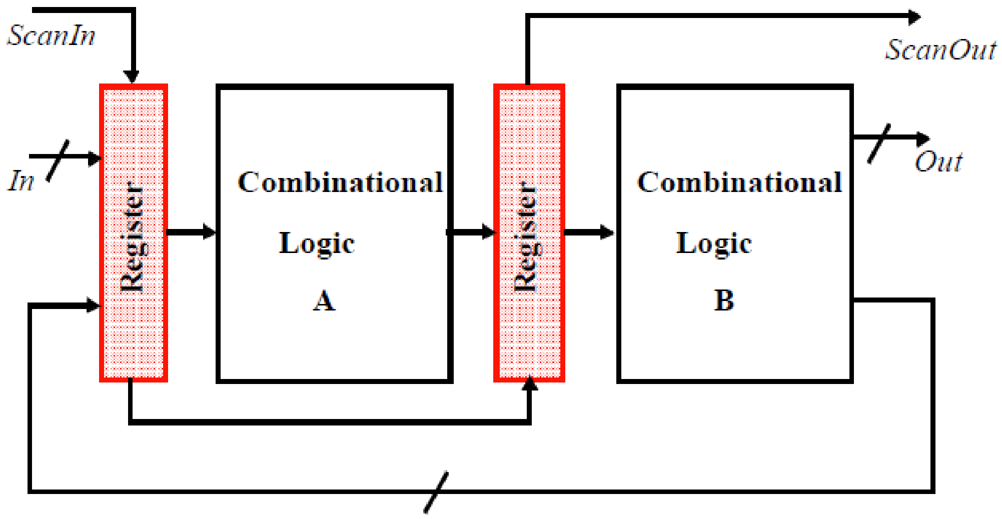

Figure 1. Structure of scan design

The procedure of a typical scan based testing is shown below:

a) The Circuit-Under-Test(CUT) is set to scan mode and test vectors are shifted in to scan cells serially.

b) The Circuit-Under-Test (CUT) switches to functional mode and the test vectors are applied to the combinational logic.

c) The test results are captured into the scan cells at the next clock cycle.

d) The CUT is then switched to scan mode and the test results are serially shifted out and compared with the expected responses. And the next test vector can be shifted in at the same time.

1.3.1 Scan Cell Types

1.3.1.A Muxed-D Scan

Muxed-D scancell, shown in Figure 2, is composed of a multiplexer and a standard D flip-flop (FF).The scan enable (SE) signal is used to select between the data input (D) and scan input (SI). Clock signal (CP) is used to clock the flip-flop in both normal and test modes.

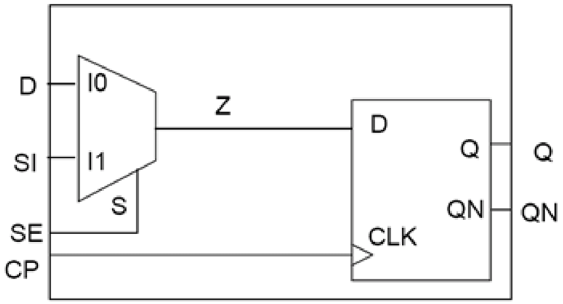

Figure 2. Muxed-D scan cell

1.3.1.B LSSD Scan

Level sensitive scan design (LSSD) [16][17] scan cell contains a pair of latches in “master-slave” model. As shown in Figure 3, a master two-port D latch L1 is followed by a slave D latch L2. Three clocks A, B, and C are used to select between the data input D and the scan input I. LSSD is then clocked appropriately during test mode. Based on different clock schemes, LSSD can be implemented by a single-latch design [16] or a double-latch design [18] based on different clock schemes.

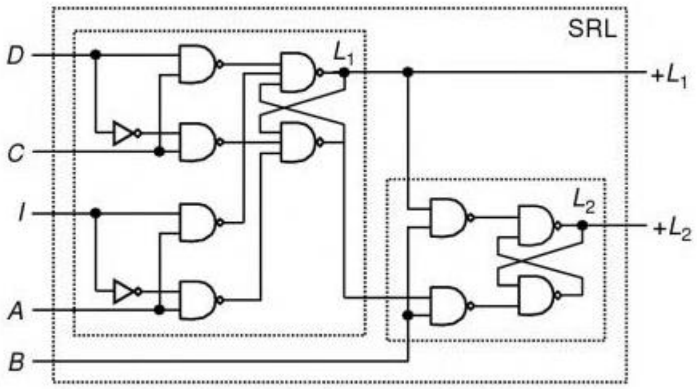

Figure 3. LSSD scan cell

1.3.1.C Enhanced Scan

Enhanced scan [19][20], shown in Figure 4, are introduced to achieve higher fault coverage, and reduce the requirement of timing constrain of control signal. It could be achieved by adding an extra holding latch to the output of each flip-flop. In this way, it could store two bits at the same time and enable to apply an arbitrary pair of test vectors. Obviously, the extra holding latch will introduce extra area consumption, timing and power consumption.

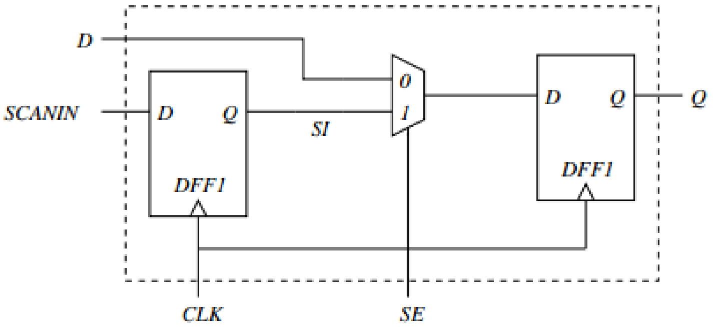

Figure 4. Example of enhanced scan cell

1.3.2 Scan Based Delay Testing

During delay testing, transitions are launched into the circuit, thus a vector pair {$V_{1,}$ V_2} are needed. The first vector V_1initializes the circuit and the second vectorV_2 launches the transitions. Based on the relationship between V_1 and V_2, delay tests can be mainly divided into two types: Launch-On-Shift (LOS) [21][22] and Launch-On-Capture (LOC). [23]. Figure 5 gives an example of both LOS and LOC.

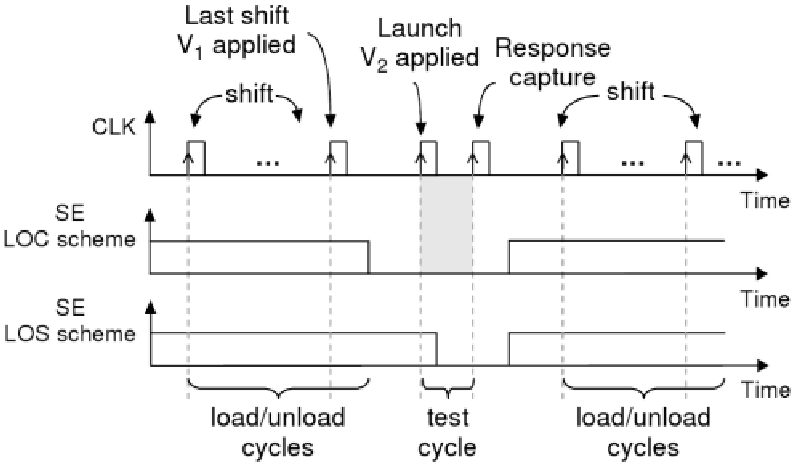

Figure 5. Clock diagram for scan based delay testing

1.3.2.A Launch-On-Shift

In the Launch-On-Shift (LOS) scheme, V_2 is a one-bit shift of the first vector V_1. After the last shift, one at-speed capture clock cycle is then applied to capture the test response. Thus, it’s necessary to at-speed switch between scan mode and functional mode, which needs an at-speed Scan Enable (SE) signal. The at-speed SE signal need to be routed as clock tree, which is not practical in high-speed designs.

1.3.2.B Launch-On-Capture

In the Launch-On-Capture (LOC) scheme, the second vector V_2 is the combinational circuit’s response to the first vector V_1. In the first capture clock cycle,

$V_2$is captured into the scan cells. In the second capture clock cycle, transitions are launched into the circuit, and are also used to capture the test response.

In LOC design, the speed of SE signal is relaxed, since the mode could be switched slowly. Thus LOC is widely used in high-speed designs.

1.4 Memory Model in Scan-based Delay Test

1.4.1 Black Box

As shown in Figure 6,the memory has input and output ports without internal description, which is called a “black box”. Any transitions captured into memory are lost and also no transitions are launched. Only unknown (X) values can be obtained from the black box. This technique creates “shadow” regions around the memory which cannot be tested and reduce the fault coverage.

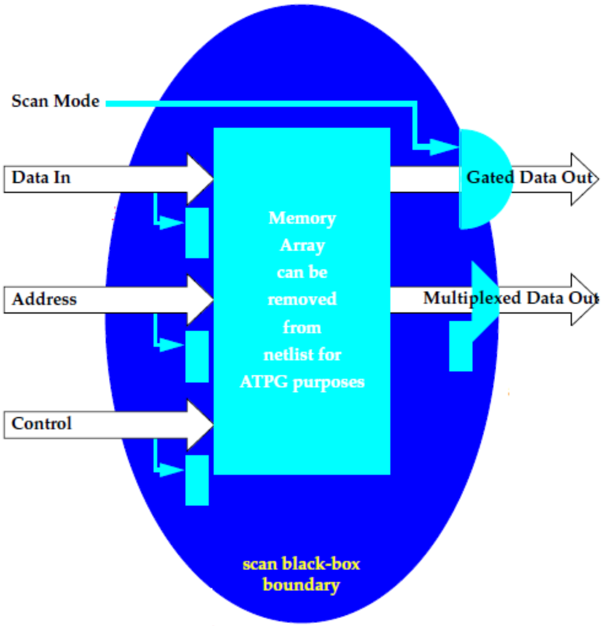

Figure 6. Memory black box model

1.4.2 Memory Bypassing

Memory bypassing mode, as shown in Figure 7, is introduced to alleviate coverage loss result from X values. In this mode, the output of the memory is fixed to a certain value or the input into the memory is transferred to the output. Thus, “X” values are reduced. However, the power supply noise (PSN) is not similar to functional memory operation, since in bypass mode, only the address decoder related to a special bypass word in the memory could be activated.

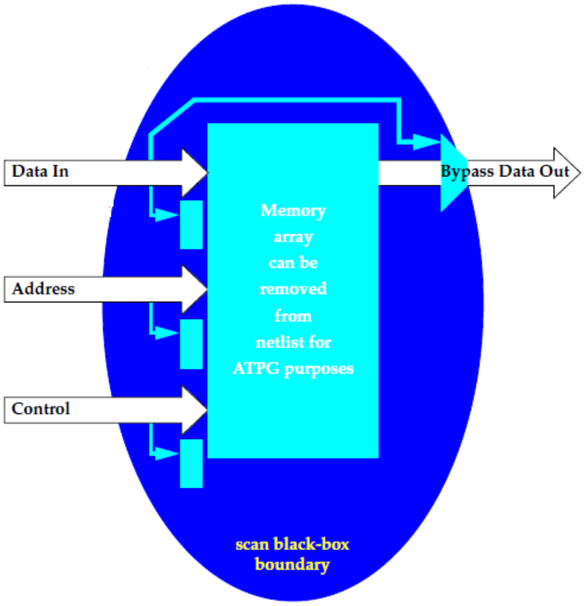

Figure 7. Memory bypassing model

1.5 KLPG Algorithm and CodGen

From previous works [24][25], the *K Longest Path Per Gate* (KLPG) algorithm has been proposed to generating vectors efficiently for delay test. In KLPG based path delay test generation, each gate have K longest rising and falling paths going through it. The reason why we want to find K longest paths is, there exits process variations [26] in the fabricated chips. As illustrated in Figure 8, because of process variation, each path can be the longest path with certain probability. In order to achieve high small delay, defect fault coverage, those paths need to be covered.

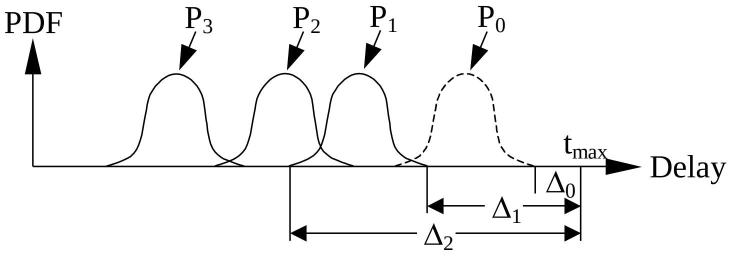

Figure 8. Probabilistic distribution of path lengths

1.6 Pseudo Functional Test

In traditional delay tests, the slow scan-in cycles are followed by the at-speed cycles. One problem result from this clocking scheme is that, the circuit will suffer DI/dt noise and IR-drops. Thus the voltage level of the circuit will act like an inductive ringing, as shown in Figure 9. However, the voltage level of the power grid in the circuit is important to the accuracy of delay test results [27] [28]. If the power supply voltage is lower than functional mode during the path delay test, the chip will run more slowly than normal and fail the test. This is termed *overkill*.

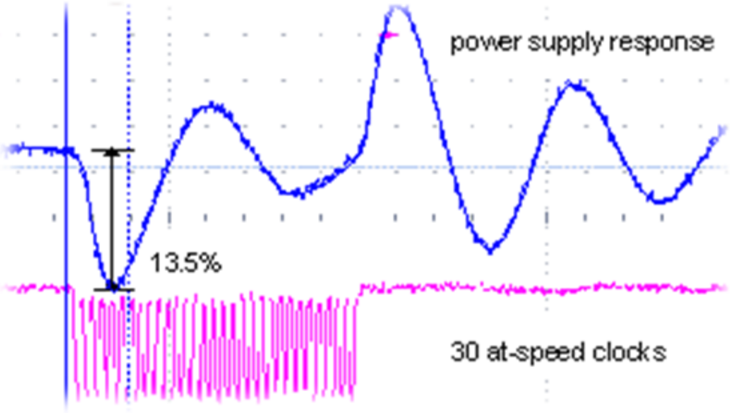

Figure 9. Delay test induces drop of power supply voltage [27]

One solution to address the mismatch of power supply voltage between test mode and normal mode, is a Pseudo Functional Test (PFT). In the PFT, there exist several clock cycles between slow scan-in clock cycles and at-speed test cycles, we call it *preamble cycles*. After those preamble cycles, the power supply voltage will be stabilized, and the chip is in the pseudo functional mode. The clock scheme is shown in Figure 10.

However, in PFT, ATPG will need more effort to generate feasible vectors. Instead of back-trace through two at-speed preamble cycle in the traditional delay test scheme, the ATPG now need to back-trace extra preamble cycles.

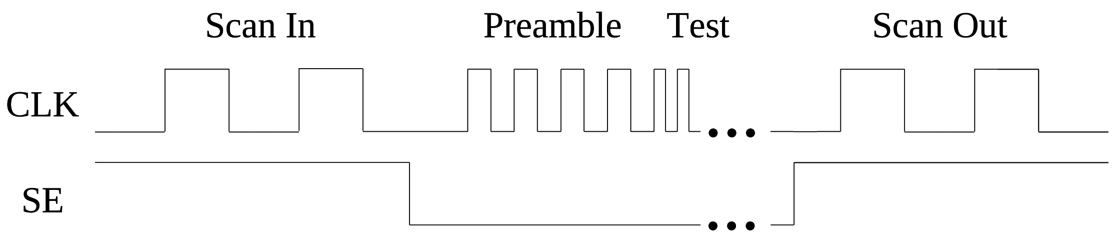

Figure 10. Clock diagram of pseudo functional test

1.7 Structure of This Dissertation

The rest of this dissertation is structured as follows: In Section 2, the proposed flow is presented. In this section, we will also discuss the memory model and test method. Section 3 discusses the modification and implementation of the current PFT KLPG (PKLPG) tool in order to support the memory model. We then discuss experimental results on benchmark and industrial circuits. In Section 4, we discuss algorithm performance and demonstrate speedup using Boolean Satisfiability (SAT). Section 5 concludes with directions for future work.

2. PSEUDO FUNCTIONAL PATH DELAY TEST THROUGH EMBEDDED MEMORY

2.1 Motivation

On-chip memory arrays are widely used in Systems-on-Chip (SoCs). Typically, embedded memory arrays are tested using functional March patterns [29][30]. Those patterns could be generated on chip with memory built-in self-test (MBIST) [31][32][33][34]. Similar to MBIST, an Embedded Micro-Tester can be used to test the components of a source at speed [35]. Macrotest [36] embeds memory tests in scan tests. With the above technologies, latch-based embedded arrays can be tested using scan test[37]. Non-scan cells can be tested using at-speed functional patterns [38]. Although the logic surrounding the memory can be tested using scan [39], the memory is still tested using BIST/functional patterns.

The above techniques provide good fault coverage of the Memory arrays themselves, and can thoroughly cover the delay defaults within the memory when supported by appropriate Design-for-Test (DFT) structures. However, they do not target small delay defects on the paths into and out of memory cells, non-scan flip-flops and latches. Prior research has shown that timing critical paths often pass through embedded memory arrays. As shown in Figure 11, the paths through embedded memory arrays consist of a write path and a read path. These paths must be tested to achieve good correlation between the maximum operating frequency (FMAX) of functional and structural delay test [29][30], and to significantly reduce defect levels.

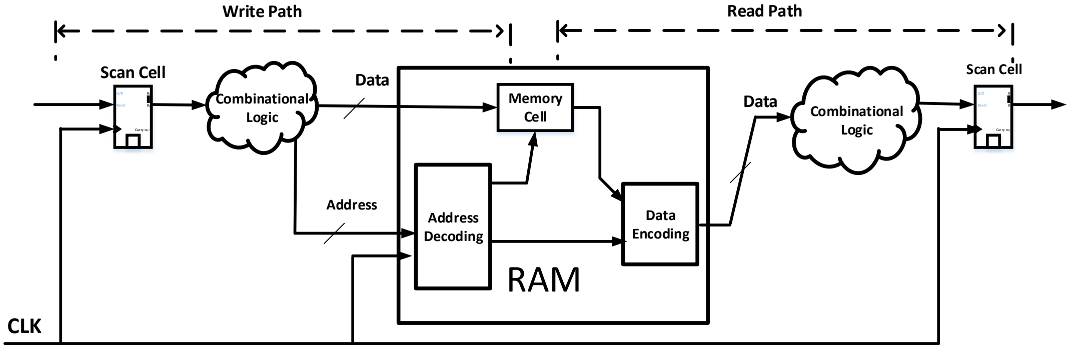

Figure 11. Paths into and out of memory

The motivation of this research is to establish an effective flow that can perform pseudo functional path delay test through embedded memory arrays.

2.2 Proposed PFT through Embedded Memory Test Generation Flow

In order to perform path delay test through embedded memory, we need to propose a standard flow that can handle a circuit with embedded memory arrays. The first step is to model the memories for ATPG usage. Since memory cells are non-scanned cells, we must model memory with this feature. Then, we need to combine the memory model and original circuit to get the circuit with the memory as a “white box” structural model. This white box model can then go through flattening, leveling and partitioning, and test generation, using the existing in-house KLPG tool *Codgen*. The flow is shown in Figure 12.

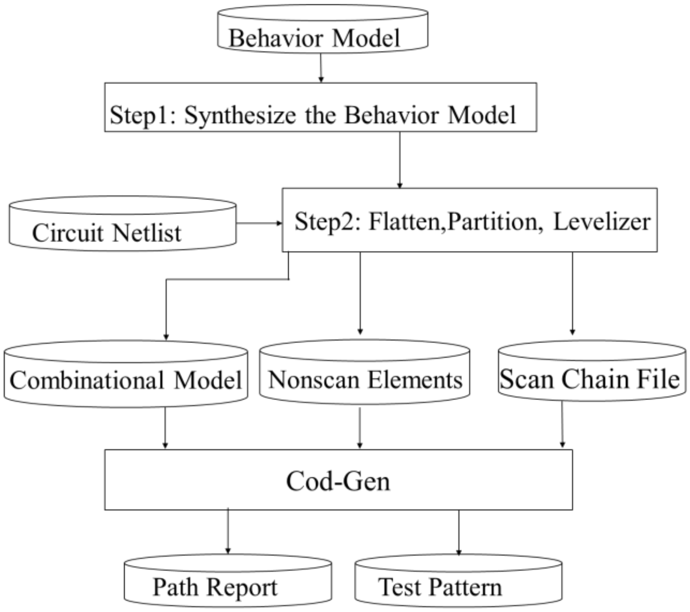

Figure 12. Proposed flow of PFT KLPG

2.3 Memory Model

There are many choices for mapping a memory to a logic circuit. A common example is the CRAM (combinational SRAM) model available in many commercial ATPG systems. In our work, we have modeled the memory array with asynchronous read and synchronous write (using a global clock). In this model, data is written to the memory arrays during the clock edge, when the correct address is decoded and write and chip enable signals are active low.

Typically, when a value is written in a memory cell it might not be read for a long time because such memory cells are enabled by the address and the same address may not be decoded frequently. However, industry feedback suggests that the value written in cells will usually be read within the next few clock cycles. So we immediately read from the memory after writing the value into the memory. This permits the scan tests for different memory words to be independent. Furthermore, the memory is modeled as a uniform delay model such that all the delays are lumped at interface gates, with the same delay to and from every memory cell. Therefore, every path to and from each memory cell must be tested, to ensure that the longest paths have been tested.

Large memories, such as L2 and L3 cache arrays, are not a uniform array, but a hierarchy of memory arrays and interconnection networks. In this case, we assume that the network is already described and we just synthesize the memory sub-arrays.

2.4 Synthesis

If a structural implementation of the memory is available, we use that directly. Otherwise, we need to synthesize the behavioral description of the memory array to get the structural model. We assume the memory behavioral model is available in Verilog, as depicted in Figure 13.

In legacy test models where a logic simulation model is not available, the memory arrays may be "black boxes", which has no internal description. In this situation, we need to write a behavior Verilog description.

The structural model of memory is achieved by synthesis the behavioral model achieved before, with commercial logic synthesis tool or memory compiler. These tools synthesize the behavioral description to a gate level implementation. During the synthesis, memory cells are mapped to flip-flops with multiplexers on their inputs to select between the data input of the memory or the outputs of the flip-flop. Also, the multiplexer is controlled by the address decoder. Figure 14 shows the general logic model of the memory arrays, which contains memory cells and feedback multiplexers. If the correct address is decoded, and write enable and chip enable are active low, new data is written to the memory on the clock edge, otherwise the same data recirculates. The data can be read any time once the address is decoded.

```
`define addressBusSize  A
`define dataBusSize  B
module memoryArray AxB (
  input [`addressBusSize-1:0] A, //Address
  input CLK,
  input [`dataBusSize-1:0] D, // Data In
  input EZ, // Enable chip, active low
  input WZ,//Write enable , active low
  output [`dataBusSize-1:0] Q // Data out
);
  //total depth = 2^addressBusSize
  parameter depth=1<<`addressBusSize;
  reg [`dataBusSize-1:0]reg_file[depth-1:0]; //total storage
  //read operation. When chip enable is low
  assign Q=(EZ==0)?reg_file[A]:`dataBusSize'bx;
  //write operation. when both write and chip enable are low
  always @ (negedge CLK)
  begin
   if (WZ==0 && EZ==0)
     reg_file[A]<=D;
  end
endmodule
```

Figure 13. Verilog template for memory behavior model

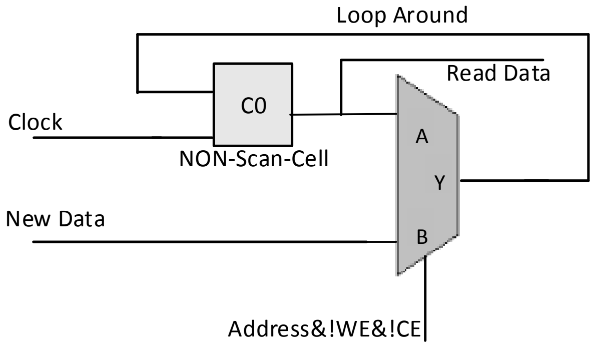

Figure 14. General structure of a memory array

At first glance, the synthesized logic model is not close to a real six-transistor SRAM cell with bit line pre-charge and sense amplifier logic. However, this structural model is acceptable for ATPG purposes. In real design, small memory arrays are implemented as latch or flip-flop arrays in standard cell flows. Also the reading and writing operations mimic the real SRAM operation. A more realistic structure can be created using a template with address-decoded enables feeding latches, which in turn feed tristate output bit lines. However, compared with the general synthesis flow, this fixed template cannot handle diverse memory types.

Although optimization is not necessary for the ATPG usage, the synthesis tool limits the fan-in of multiplexer and creates multiplexer tree for larger memories. Also, it takes advantage of AND-OR-INVERTER (AOI) and OR-AND-INVERTERS (OAI) cells in the library for optimization purpose.

Figure 15 shows a synthesized 4x3 memory array, which is composed of 12 non-scan cells, 4 bit width address and 3 data bits. So each address selects 4 non-scan cells. Layers of gates are generated for writing and reading the data. For reading the data, the first logic layer contains a series of AND-NOR gates given by U53, U52, U56, U55, U61 and U58, and gives a total of 6 possible data bit values out of 12 non-scan cells. Depends on the decoded address, U53 and U52 selects one output data out of the first four possible non-scan cells. The memory array can be visualized as a 2D array of memory[4][3], where the first index is the memory address and the second index is the data bit. Since the memory gives a 3-bit output for any address, U53 and U52 gives the value of memory[X][0] where X can be from 0 to 3.

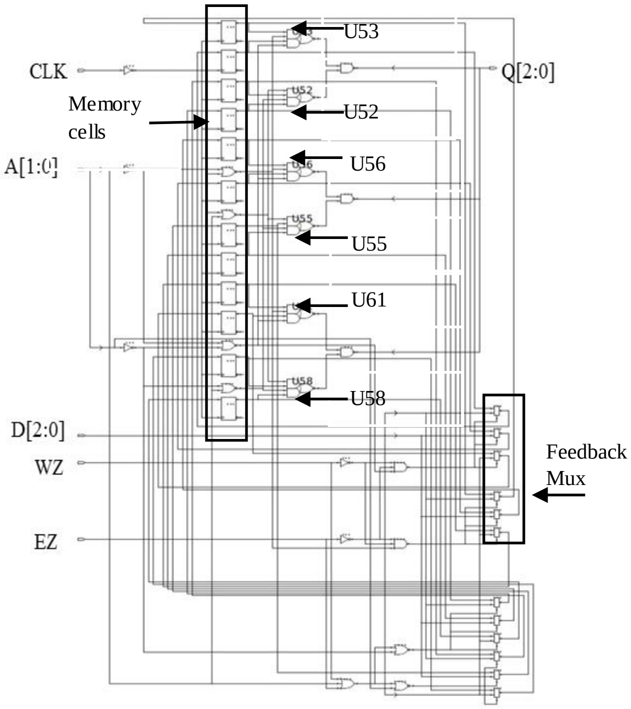

Figure 15. Logical model of 4x3 memory array

The decoding logic adds more logic, if the number of address bits increases. For each additional address bit, one level of decoding logic is added. For example, if we have an 8x3 memory array, one extra level will be needed, compared with the 4x3 memory array shown in Figure 15, to select each data bit from the 8 addresses.

The output decoding of one-bit slice of a 256x8 memory array is shown in Figure 16. The first data bit (0^{th}) is decoded out of 256 possible addresses. Once the decoded address X is sent to the treelike combinational logic, the 0^{th} data from the memory cell of this decoded address is then obtained from the output of the 2:1 multiplexer. The decoding logic depends upon the design library of synthesis tool and the optimization used. For example, minimum power or minimum area consumption can be applied during the synthesis which will result in a different decoding structure. We use the default setting of medium power and minimum area in our synthesis.

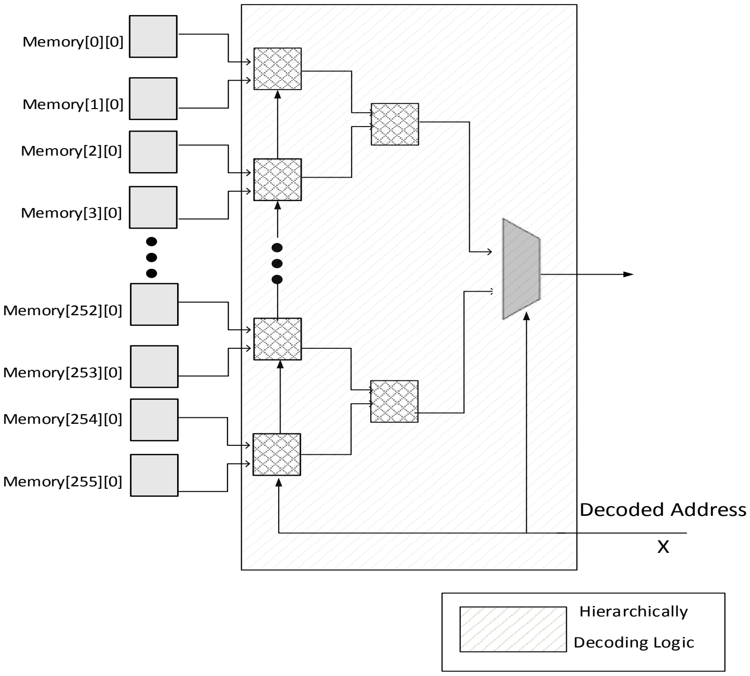

Figure 16. Output decoding structure in a 256x8 memory

The address decoding for memory writing is obtained by similar combinational logic. If the address bits increase, the logic levels of decoding logic will also increase. These levels are similar to the treelike combinational logic for reading the data.

2.5 Longest Paths Into and Out of the Embedded Memory Array

Longest paths through memory arrays could be divided into two types: longest paths into memory arrays and longest paths out of memory arrays.

2.5.1 Longest paths into the memory

The longest paths into a memory or non-scan cell are tested during at-speed launch and capture cycles as the typical delay test. However, the Boolean values of this path are captured in a non-scan memory cell, which is not observable directly. Those values are needed to be observed so that the paths are kept. To address this problem, we add one or more(if needed) clock cycles to the original clock scheme and further extend *Codgen* to support this feature. The untimed(and so use a low-speed clock) extra cycles are termed *coda* cycles.

In the coda cycles, any paths could be used for propagating the captured Boolean value. The details of choosing paths in coda cycle will be discussed in the later section. Once the coda path is selected, then the path is justified. Since the at-speed path might be a sequential false path, it can be more efficient to first justify the at-speed path before finding paths in the coda cycles.

If there exist pipelined memories in the design, such as registers in a classic 5-stage processor data pipeline, it may take several coda cycles to propagate the captured Boolean value to primary outputs or pseudo primary outputs, as shown in Figure 17. During the test, a fixed number of coda cycles are assumed. Thus to cover all targeted path, a test set may have tests with several different coda cycles may be needed.

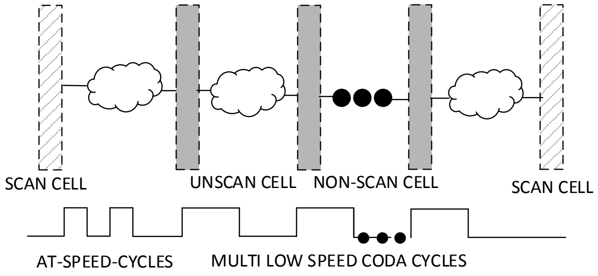

Figure 17. Multiple coda cycles

2.5.2 Longest Paths Out of the Memory

It requires launching a transition at the memory output to test the longest paths out of the memory, as the read path in Figure 11. If the memory models are “black box”, no transitions could be launched. Also, with the typical memory bypassing where outputs are assigned a fixed value, no transitions could be launched neither. Enhanced scan enable transition generation at the memory output, but at an area and delay cost. In our work, we proposed a way to launch the transition by using the existing preamble cycles to initialize the memory array with values that can then be read to launch the transition.

Suppose we have a 2x1 memory array with two cells, C0 and C1, as shown in Figure 18. We have one data output. The output is multiplexed between C0 and C1, as controlled by the address line.

If C0 holds 0 and C1 holds 1, if we want to launch a rising transition from the memory, then the address line switches from 0 to 1. Prior to the at-speed address transition, we must write C0 and C1, during the preamble cycles.

To show the launch of a falling output transition F, the multiplexer in Figure 18 is decomposed to primitive gates in Figure 19. The rising transition R, starts from the select line of multiplexer, which is actually the address line of the memory array. The path will grow through NOT gate and the AND1 gate.

In the two at-speed time frames, the A need to be {1,1} for robust test. Those values are obtained from the non-scan cell C0, which are written in preamble cycle. The path will further add the OR gate in the partial path along with all the necessary assignments.

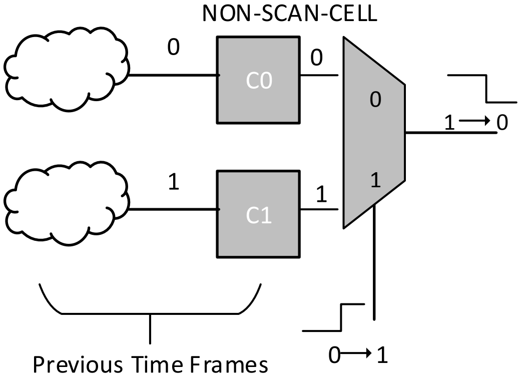

Figure 18.Launching a transition by toggling the address

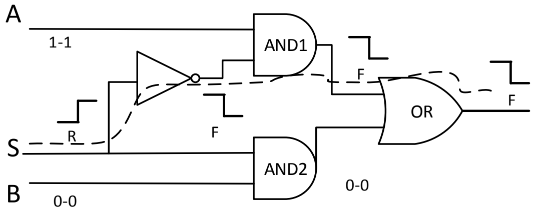

Figure 19. Necessary assignments inside a multiplexer to propagate a transition

During the preamble cycles, the necessary assignments (NAs) are written to the non-scan cells. To set all the necessary values, a sufficient number of preamble cycles are needed. Assuming only one word can be written at a time, at least two preamble cycles are needed to generate transitions by toggling between two bits. More than two preamble cycles can be used to reduce the dI/dt noise and IR drop before the at-speed test.

Figure 20 shows the clocking scheme of a PKLPG test with coda cycles. In this example, 4 preamble cycles, 2 at-speed test cycles and 4 coda cycles are used, along with scan in and out cycles.

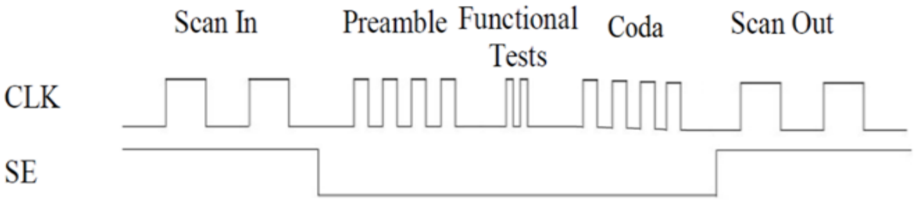

Figure 20. Coda cycles in PFT KLPG

3. MODIFICATION OF CODGEN

This research is built on top of prior work with *CodGen*, which is a KLPG based path delay test generator supporting Pseudo Functional Test (PFT) [40] and dynamic compaction [41].

3.1 Description of CodGen

The basic *CodGen* algorithm is shown in Figure 20. In *CodGen*, path delay tests are generated in three steps:

- Path Search: The KLPG algorithm will generate K longest rising and falling paths through each target fault site. In order to sensitize the searched path, a set of Necessary Assignments(NAs) is generated along with the searched path. For example, in order to propagate the value from the input to the output, a NA is logic one for one input of an AND gate.

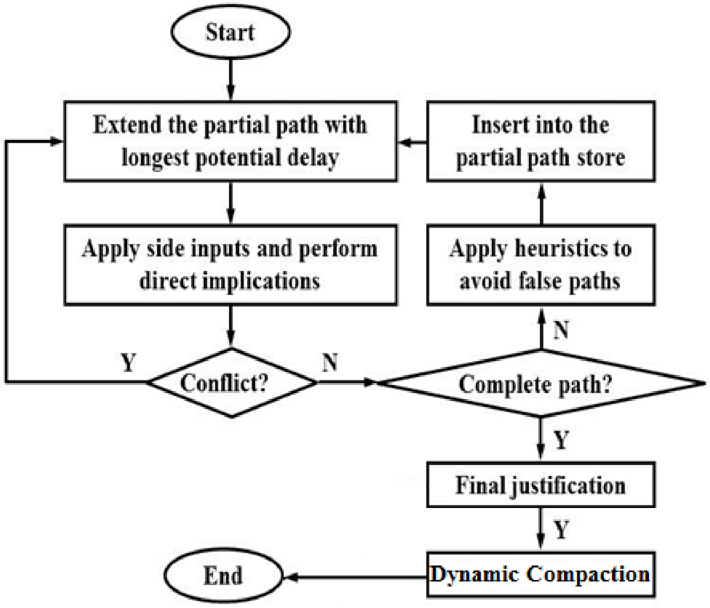
Start
Extend the partial path with longest potential delay
Insert into the partial path store
Apply side inputs and perform direct implications
Apply heuristics to avoid false paths
Y
Conflict?
N
N
Complete path?
Y
Final justification
Y
End
Dynamic Compaction

Figure 21. Basic *CodGen* algorithm

- Path Justification: After path search, the founded path need to be justified. The reason is that, some paths which pass the direct implication and false paths prune methods, may still be unsensitizable. Path justification will find a set of value for primary inputs(PIs) and pseudo primary inputs(PPIs) to guarantee the necessary assignments found in the previous stage. As shown in Figure 21, to sensitize the path from *b* to *g*, *a=0* and *c=1* are necessary assignments(NAs), and *{X,0}*, *{1,1}* are the values in two vectors on *a* and *c* to generate necessary assignments.

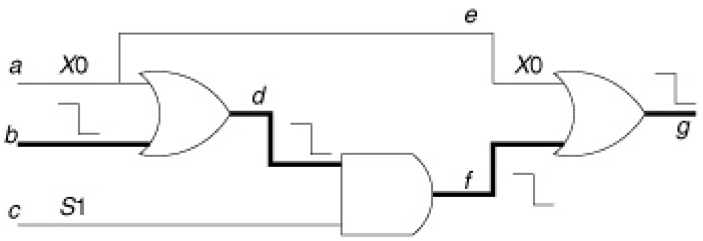

Figure 22. Example of sensitizable path

- Dynamic Compaction: In order to reduce the final vector count, vector for each justified path can be compact together. If the compaction is done after all the vectors are generated, the compaction is called static compaction. Another compaction method is dynamic compaction which compacts the vectors during the path finding. Compared with static compaction, this method will improve the compaction ratio while take more CPU time.

3.2 Non-Scan Cells in CodGen

Non-scan cell behavior differs in testing the path into memory and the path out of the memory.

3.2.1 Testing Longest Paths into the Memory

When testing the longest path into the memory, the transition will be captured in a non-scan cell. Once the value is captured in a non-scan cell, we need to start path finding in the coda cycle. The ATPG uses SCOAP (Sandia Controllability Observability Analysis Program) metrics to first select the easiest (most observable, with the fewest necessary assignments) path to a Primary Output or a Pseudo-Primary Output, which is the data input of a scan cell. If that fails, it tries the next path, and repeat until successfully finding a path. The search space is shown in Figure 23.

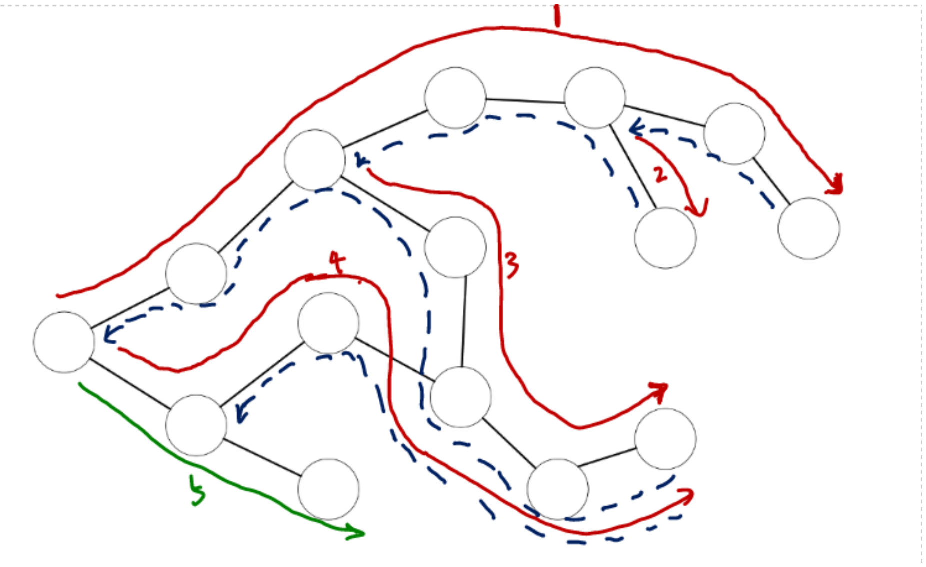

Figure 23. Path finding in the Coda Cycle

In the above search space, assume the feasible path is 5th path and the 1st path is the most observable path at the beginning. If the 1st path fails the final justification, the tool simply backtracks to the latest decision made point and tries the next path, 2nd path in this case. It will repeat backtracking until the 5th path is tried.

After finishing propagation, the complete path is put into the path pool for final justification.

3.2.2 Testing Longest Paths Out of the Memory

When testing longest paths out of the memory, the transition is launched through the address line. The contents of memory cells are initialized during the preamble cycles. Since the memory cells are functional during the preamble cycles, we need to justify them during these cycles. If the constraint is too tight to initialize memory cells, the paths related to these cells are not sensitizable.

3.3 Coda Cycle in Codgen

In order to support coda cycles in Codgen, extra time frames are added. If we denote the total number of cycles as #FRAMSC and that of unmodified cycles as #FRAMS, the number of coda cycles is #FRAMSC-#FRAMS. As described previously, if there exists a pipelined memory structure, more than one coda cycle will be needed to propagate the captured response to a scan cell.

3.4 Experimental Results

To test the proposed ATPG flow, experiments are performed on a 3.4 GHz Intel Core i7 processor with 8 GB of memory. Robust tests are used to test the paths in and out of the memory. A total of 2 preamble cycles are used in all tests. Since in our experiments either we placed a scan chain on the memory output or we assume there are scan cells between pipelined memory arrays, only 1 coda cycle is used. Dynamic compaction is used to compact the tests together. This compaction does not exploit the word-oriented nature of the memory.

Different memory arrays were synthesized with Synopsys Design Compiler. The memory size is given as *A*x*B* where *A* denotes the total number of words and *B* denotes the number of output bits.

3.4.1 Standalone Memory

In this experiment, the memory is directly controlled by the Primary Inputs (PIs) and observed by the Primary Outputs (POs), as shown in Figure 24.

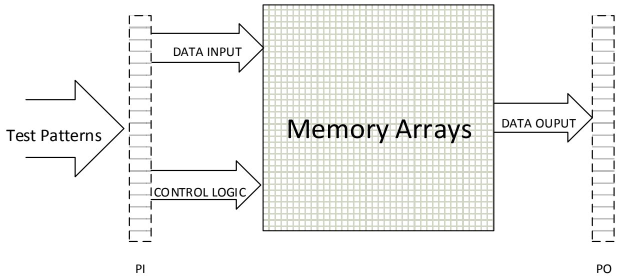

Figure 24. Standalone memory test

Table 1 shows the number of paths (rising and falling paths are counted separately), compacted patterns, CPU time, and gate count of the synthesized test model. The patterns generated are sufficient to test all of the paths in the memory arrays. In the 8x256 memory, the paths can be tested by writing eight words of 0s and eight words of

1s and reading can be tested by reading those eight words of 0s and eight words of 1s. So, 32 patterns are necessary to test the memory. *CodGen* generates a total of 33 patterns. A 256x8 memory can be tested by 1024 patterns of 8-bits each, while dynamic compaction achieves 1114 patterns. The dynamic compaction algorithm is greedy, so may not find the minimum test set. A second reason is that a limited pool of paths is kept in memory, which may cause the algorithm to write a pattern to memory, when there are still paths that might compact into the pattern. Note that despite the presence of non-functional paths, the pattern count is close to the functional pattern count. For example, the non-functional flip-flop feedback path cannot have transitions on it when the flip-flop is holding its state.

Table 1. Results of PFT KLPG standalone memory arrays

Memory Size	Paths	Patterns	Time	Gate count
4x3	58	16	0:00:01	106
4x8	232	16	0:00:02	320
8x4	227	33	0:00:02	297
16x8	1022	73	0:00:12	1310
8x16	1044	34	0:00:09	1310
64x8	4161	305	0:01:54	5217
8x64	4116	35	0:01:28	5150
128x8	8428	608	0:09:10	10462
8x128	8212	34	0:05:30	10270
256x8	16661	1114	0:52:58	20793
8x256	16404	33	0:25:28	20510

The paths, pattern and gate count increase approximately linearly with memory size. The number of paths is slightly more than linear and the number of gates slightly less than linear, due to the select logic overhead. However, CPU time increases approximately quadratically with circuit size. The CPU time for the 8x16 memory array is 9 seconds and for the 8x64 memory array is 1 minute 28 seconds. The reason for this behavior is that each time a path is extended by one gate during the path search algorithm, direct implication is performed to screen out false paths. The structure of the address decoder causes direct implications to propagate throughout the memory circuit. Essentially, a 1 on one decoder output implies a 0 on every other output. Most of these implications are not useful for screening false paths, significantly increasing CPU time.

3.4.2 Memory Array with Combinational Logic Inputs

We next test a memory array with its inputs fed through a combinational logic circuit, as shown in Figure 24. The combinational circuit used is a full adder. The outputs of the adder are connected to the address lines only, with the data and enable lines taken directly from the PIs.

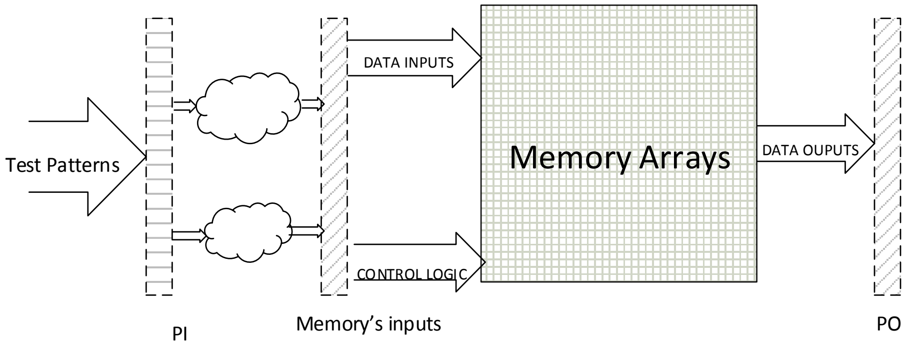

Figure 25. Memory fed through combinational logic

The results obtained from PKLPG are shown in Table 2. Fewer paths are generated due to the input constraints. Depending on the memory, the number of patterns and CPU time may go up or down.

Table 2. Results of memory PFT KLPG with adder in front

Memory Size	Paths	Patterns	Time	Gate count
4x3	50	17	0:00:01	112
4x8	182	17	0:00:01	326
8x4	195	39	0:00:01	310
16x8	831	76	0:00:10	1323
8x16	810	43	0:00:11	1323
64x8	3355	320	0:02:00	5230
8x64	3163	43	0:01:32	5163
128x8	6782	595	0:09:47	10475
8x128	6298	44	0:06:58	10283
256x8	12528	996	0:48:29	20806
8x256	12566	39	0:30:46	20523

3.4.3 Memory Array in Industrial Design

To test our method on an industrial design, we consider a streaming audio controller chip (Chip1). This controller has 4 memory arrays modeled as black boxes in the original test model, whose sizes are 256x8, 2048x8, 8192x8, and 4096x8. The total gate count of the chip is 41,263, excluding the memory arrays. We wrote the behavioral model for the 256x8 memory and synthesized it to get the structural model. Then we replaced its black box model with the structural model. The gate count of the chip with the structural model increases to 56,206. We configure the KLPG algorithm to apply 2 preamble cycles, 2 at-speed launch and capture cycles, followed by 1 coda cycle. We also chose the robust test and enabled dynamic compaction. The results are shown in Table 3.

The results show a 60% increase in tested paths once the shadows around the 256x8 memory are removed. The longest testable path shrinks slightly. This will be explained below. The average path length increases by 6% due to paths entering and exiting the memory. The pattern count increases faster than the path count, due to longer paths with more necessary assignments. The CPU time falls significantly. The reason is that the ATPG no longer has to attempt to work its way around the memory shadows to test a path.

Table 3. Results of PFT KLPG with controller

Circuit Name	# Paths	Longest Testable Path Length	Average Path Length	# Patterns	CPU Time
Original	5145	42	9.245	474	03:29:56
Chip1 with 256x8	8161	41	9.797	878	01:44:52

However, when testing Chip1 with the 2048x8 memory replaced with a structural model, the ATPG ran for more than 6 CPU days, with the poor results shown in Table 4. The path count, longest testable path, and average path length all fell considerably, despite the high CPU time.

Table 4. Results of controller with 2048x8 memory as white box

Circuit Name	# Paths	Longest Testable Path Length	Average Path Length
Chip1 with 2048x8	3882	35	7.17435

The reason for the poor results in Table 4 and the reduction in the longest path in Table 3 lies in the use of the PODEM algorithm for justification and compaction. The details will be discussed in Section 4.

4. TECHNIQUES TO IMPROVE SAT EFFICIENCY

Since the PODEM-based justification method failed to test circuits with larger structural memory models, we will discuss the reason and solution in this section.

4.1 Limitation of PODEM Justification Method

In KLPG, for a particular target fault site, we use three main bounds to limit the CPU time:

- Partial Path Store Size
- Number of Max Try
- FAN_limit: Number of FANs

These bounds are set based on experience with prior designs, so that true paths are identified, but CPU time is not wasted on false paths.

One of the most important features of Chip1 with the 2048x8 memory model is that some gates in the memory model have very high fan-outs, up to 10,000memory. In *CodGen*, each time we want to extend the path, we duplicate the original path for each fan-out and store these copies in the partial path store. These paths are then available if the search process must consider another fan-out branch from that gate. Storing these paths may lead to loss of sensitizable paths, for the reasons listed:

1. The new added paths have *Esperance* that is greater than other paths in the partial path store, so they are stored at the top of the sorted store. If storing these fan-out

paths causes the partial path store to exceed its size, paths of lower *Esperance* are discarded, even though they may be sensitizable.

2. Each successful path extension will add one to the counter of number of tries, thus the high fan-out would consume many of the total tries for the fault site. It is very likely that the path will exhaust its maximum number of tries, even though the path can continue to be extended.

3. If those high fan-out paths are not able to pass final justification, they will take large fraction of the allowable FAN searches. The remaining FAN attempts may not be sufficient to identify the sensitizable paths.

The original *CodGen*code uses PODEM as its justification algorithm. The built-in backtrack limit in PODEM may cause the algorithm to give up some delay tests even though they could be justified using a more efficient method. Combined with the three limits above, this problem reduces the fault coverage of the delay tests when high fan-out nodes are present.

4.2 Using SAT as Justification Method

Recently the performance of Boolean Satisfiability (SAT) solvers has significantly improved [42][43] and SAT-based ATPG has been successfully implemented to generate test patterns for stuck-at faults, transition delay faults and path delay faults [44][45]. Therefore, it is natural for us to replace the PODEM-based approach with a SAT-based approach, in order to improve both the efficiency of path delay test generation and the fault coverage of the delay tests.

Experiments were conducted to test the performance of *CodGen* with the *MiniSat*SAT solver, and the results compared with those of the original *CodGen* using PODEM. All experiments were performed for K=1 where K is the number of rising and falling paths to be tested through each line in the circuit. Robust path constraints and Launch-On-Capture (LOC) tests were applied. Two preamble cycles, two at-speed cycles and one coda cycle were used in the experiments.

The results of *CodGen* with PODEM and SAT are compared in Tables 5 and 6. In Table 5, we can see that for the smaller memory benchmarks, PODEM and SAT have the same results, but SAT usually takes more CPU time. For the larger circuits, SAT finds more paths and compacts them into fewer patterns, using similar or less CPU time. In Table 5, SAT finds many more paths of longer length, using about 3x less CPU time in the largest case. The average path length is lower in SAT with the structural models, due to the large numbers of short paths that are found, which were previously not found.

Table 5. Comparison of PODEM and SAT with small circuit

Circuit Name	# Paths		#Patterns		#Gate		CPU Time (h:m:s)	
	PODEM	SAT	PODEM	SAT	PODEM	SAT	PODEM	SAT
8x16	810	810	43	43	1323	1323	0:00:20	0:00:09
8x64	3163	3163	43	43	5163	5163	0:00:58	0:01:07
8x128	6298	6298	44	44	10283	10283	0:02:31	0:04:35
8x256	12566	12566	39	39	20523	20523	0:09:55	0:15:14
16x8	839	839	77	77	1323	1323	0:00:22	0:00:09
64x8	3387	3387	309	302	5230	5230	0:01:31	0:01:33
128x8	6831	6847	597	549	10475	10475	0:04:57	0:07:08
256x8	12688	13528	1081	1053	20806	20806	0:24:27	0:18:12

Table 6. Comparison of PODEM and SAT with industrial circuit

Circuit Name	# Paths		Length of Longest Testable Path		Average Path Length		CPU Time (d:h:m:s)	
	PODEM	SAT	PODEM	SAT	PODEM	SAT	PODEM	SAT
Original	5145	14788	42	48	9.245	12.385	03:29:56	06:42:53
Chip1with 256x8	8161	13623	41	48	9.797	10.844	01:44:52	05:26:22
Chip1with 2048x8	3882	25206	35	48	7.174	28.138	>6:00:00:00	2:07:16:18

4.3 Further improvements in SAT Solver

When doing dynamic compaction, the SAT solver uses a C++ STL map to maintain the gate index and corresponding assignment information. The map in C++ STL is realized by a red-black tree. Each insertion and retrieval takes O(logN) time. The STL destruction method is not called properly. In order to improve efficiency and reduce memory consumption, we replace the STL map with an array used as a static hash table. Its store and retrieval time complexity is O(1). It also has a simple access mechanism.

Table 7. Result of SAT with different implemented map on small circuit

Circuit Name	CPU Time (h:m:s)	
	SAT with STL Map	SAT with Static Hash
8x16	0:00:09	0:00:08
8x64	0:01:07	0:00:55
8x128	0:04:35	0:04:05
8x256	0:15:14	0:15:18
16x8	0:00:09	0:00:08
64x8	0:01:33	0:01:24
128x8	0:07:08	0:06:19
256x8	0:18:12	0:30:32

Table 8. Result of SAT with different implemented map on industrial circuit

Circuit Name	CPU Time (d:h:m:s)	
	SAT with STL Map	SAT with Static Hash
Original	0:06:42:53	0:06:39:53
Chip1with 256x8	0:05:26:22	0:01:40:00
Chip1with 2048x8	2:07:16:18	0:16:53:14

The experimental results comparing STL Map and Static Hash are shown in Tables 7 and 8. When running SAT with STL MAP on Chip1with the 2048x8 structural memory, it would consume >60GB memory. Using Static Hash, the memory consumption is<3GB. In the benchmark memories, the CPU time is reduced for most circuits. In Chip1, the CPU time is reduced by a factor of 3 when the structural memory models are used, while achieving the same test results.

5. CONCLUSIONS AND FUTURE WORK

This thesis focuses on performing pseudo function path delay test through embedded memory arrays. The experimental results with different benchmarks demonstrate the practicality of the proposed test flow. A series of techniques to improve the accuracy and efficiency of test generation are investigated. By replacing the original PODEM justification algorithm with Boolean Satisfiability (SAT), we significantly reduced the runtime of test generation while covering more paths on the industrial circuit.

Future work includes increasing the performance of the ATPG on large memory arrays, and demonstrating cases where the memory is deeply embedded in the logic. We will limit the direct implication search space to the parts of the circuit where they are useful. We will also consider models where the delays match those of the supplied behavioral model. Eventually we would like to consider logic models that have switching activity closer to that of the memory array and improve the memory delay model according to the vendor’s memory structure.

REFERENCES

[1] R. D. Eldred, “Test Routines Based on Symbolic Logical Statements," *Journal of the ACM*, vol. 6, pp. 33-36, 1959.

[2] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, “Transition Fault Simulation," *IEEE Design and Test of Computers*, vol. 4, no. 2, pp. 32-38, 1987.

[3] K. -T. Cheng, “Transition Fault Testing for Sequential Circuits,” *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 12, no.12, pp. 1971-1983, Dec. 1993.

[4] Z. Barzilai and B. K. Rosen, “Comparison of AC Self-Testing Procedures,” in *Proceedings of IEEE International Test Conference*, pp. 89-94, Oct. 1983.

[5] G. L. Smith, “Model for Delay Faults Based Upon Paths”, *IEEE Int’l Test Conf.*, Philadelphia, PA, Oct. 1985, pp. 342-349.

[6] V. S. Iyengar, B. K. Rosen, and I. Spillinger, “Delay Test Generation 1 – Concepts and Coverage Metrics,” in *Proceedings of IEEE International Test Conference*, pp. 857-866, Sept. 1988.

[7] V. S. Iyengar, B. K. Rosen, and I. Spillinger, “Delay Test Generation 2 – Algebra and Algorithms,” in *Proceedings of IEEE International Test Conference*, pp. 867-876, Sept. 1988.

[8] J. Carter, V. Iyengar, and B. Rosen, “Efficient Test Coverage Determination for Delay Faults,” in *Proceedings of IEEE International Test Conference*, pp. 418-

427, Sept. 1987.

[9] E. S. Park, M. R. Mercer, and T. W. Williams, “Statistical Delay Fault Coverage and Defect Level for Delay Faults,” in *Proceedings of IEEE International Test Conference*, pp. 492-499, Sept. 1988.

[10] C. W. Tseng and E. J. McCluskey, “Multiple-Output Propagation Transition Fault Test,” in *Proceedings of IEEE International Test Conference*, pp. 358-366, Oct. 2001.

[11] X. Lin and J. Rajski, “Propagation Delay Fault: A New Model to Test Delay Faults,” in *Proceedings of IEEE Asian South Pacific Design Automation Conference*, pp. 178-183, 2005.

[12] A. K. Pramanick and S. M. Reddy, “On the Fault Coverage of Gate Delay Fault Detecting Tests,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 1, pp. 78-94, Jan. 1997.

[13] A. K. Majhi, J. Jacob, L. M. Patnaik, and V. D. Agrawal, “On Test Coverage of Path Delay Faults,” in *Proceedings of International Conference on VLSI Design*, pp. 418-421, Jan. 1996.

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

[15] L. H. Goldstein and E. L. Thigpen, “SCOAP: Sandia Controllability/Observability Analysis Program,” in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 190–196, Jun. 1980.

[16] E. B. Eichelberger and T. W. Williams, “A Logic Design Structure for LSI Testability,” in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 462-468, June 1977.

[17] F. Motika, N. Tendolkar, C. Beh, W. Heller, C.Radke, and P. Nigh, “A Logic Chip Delay-test Method Based on System Timing,” *IBM Journal of Research and Development*, Vol. 34, No.2/3, pp. 299-312, March/May 1990.

[18] S. DasGupta, P. Goel, R. G. Walther, and T. W. Williams, “A Variation of LSSD and Its Implications on Design and Test Pattern Generation in VLSI,” in *Proceedings of IEEE International Test Conference*, pp. 63-66, Nov. 1982.

[19] C. T. Glover and M. R. Mercer, “A Method of Delay Fault Test Generation,” in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 90-95, Jun. 1988.

[20] B. I. Dervisoglu and G. E. Strong, “Design for Testability: Using Scan Path Techniques for Path-delay Test and Measurement,” in *Proceedings of IEEE International Test Conference*, pp. 365-374, Oct. 1991.

[21] J. Savir, “Skewed-Load Transition Test: Part I, Calculus,” in *Proceedings of IEEE International Test Conference*, pp. 705-713, Sept. 1992.

[22] S. Patel and J. Savir, “Skewed-Load Transition Test: Part II, Coverage,” in *Proceedings of IEEE International Test Conference*, pp. 714-722, Sept. 1992.

[23] J. Savir and S. Patel, “Broad-Side Delay Test,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1057-1064, Aug. 1994.

[24] W. Qiu and D. M. H. Walker, “An Efficient Algorithm for Finding the K Longest

Testable Paths Through Each Gate in a Combinational Circuit", *IEEE Int'l Test Conf.*, Charlotte, NC, Sep. 2003, pp. 592-601.

[25] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, "K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits", *IEEE Int'l Test Conf.*, Charlotte, NC, pp.223-231, Oct. 2004.

[26] D. M. H. Walker, "Tolerance of Delay Faults," *IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, 1992, pp. 207-216.

[27] P. Pant and J. Zelman, "Understanding Power Supply Droop During At-Speed Scan Testing", *IEEE VLSI Test Symp.*, Santa Cruz, CA, May 2009, pp.227-232.

[28] B. Nadearu-Dostie, K. Takeshita and J.-F. Cote, "Power-Aware At-Speed Scan Test Methodology for Circuits with Synchronous Clocks", in *Proceedings of IEEE International Test Conference*, pp.1-10, Oct. 2008.

[29] Belete, D. and Razdan, A. and Schwarz, W. and Raina, R. and Hawkins, C. and Morehead, J., "Use of DFT techniques in speed grading a 1 GHz+ microprocessor," in *Test Conference, 2002. Proceedings. International*, 2002, pp. 1111-1119.

[30] Zeng, J. and Abadir, M and Vandling, G. and Wang, L and Kolhatkar, A. and Abraham, J., "On correlating structural tests with functional tests for speed binning of high performance design," in Proc. IEEE International Conference, *2004*, pp. 31-37.

[31] Abadir, Magdy S., and Hassan K. Reghbati, "Functional testing of semiconductor random access memories.," in *ACM Computing Surveys (CSUR)15.3*, 1983, pp. 175-198.

[32] Zarrineh, Kamran, Shambhu J. Upadhyaya, and Sreejit Chakravarty, "A new framework for generating optimal march tests for memory arrays.," in *Test Conference, 1998. Proceedings., International. IEEE*, 1998.

[33] R. Rajsuman, "Design and test of large embedded memories: An overview," *Design Test of Computers, IEEE,* vol. 18, no. 3, pp. 16-27, May 2001.

[34] Zarrineh, K. and Upadhyaya, S.J., "On programmable memory built-in self test architectures," in *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, 1999, pp. 708-713.

[35] Crouch, Alfred Larry, Jennifer Lynn McKeown, and Clark Gilson Shepard., "Multiple BIST controllers for testing multiple embedded memory arrays.". U.S. Patent 5,995,731., 30 November 1999.

[36] Tuna, M. and Benabdenbi, M. and Greiner, A., "At-Speed Testing of Core-Based System-on-Chip Using an Embedded Micro-Tester," in *VLSI Test Symposium, 2007. 25th IEEE*, 2007, pp. 447-454.

[37] Ross, D.E. and Wood, T. and Giles, G., "Conversion of small functional test sets of nonscan blocks to scan patterns," in *Test Conference, 2000. Proceedings. International*, 2000, pp. 691-700.

[38] Fan Yang and Chakravarty, S., "Testing of latch based embedded arrays using scan tests," in *Test Conference (ITC), 2010 IEEE International*, 2010, pp. 1-10.

[39] Banga, M. and Rahagude, N. and Hsiao, M.S., "Design-for-test methodology for non-scan at-speed testing," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1-6.

[40] S. Lahiri, D. M. H. Walker and K. Bian, "KLPG based Pseudo-functional Test with Dynamic Compaction", *SRC TECHCON,* Austin, TX, Sep. 2011.

[41] Z. Wang and D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test", *IEEE VLSI Test Symp.*, San Diego, CA, Apr-May, 2008, pp.243-248.

[42] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an efficient SAT solver", *IEEE/ACM Design Automation Conf.*, San Francisco, Jun. 2001, pp. 530-535.

[43] N. Eén and N. Sörensson, "An extensible SAT solver", *Int'l Conf. on Theory and Applications of Satisfiability Testing*, Vancouver, BC, May 2004, pp. 502-518.

[44] R. Drechsler et al., "Test Pattern Generation using Boolean Proof Engines", Ch. 10, Springer, Berlin, Germany, 2009.

[45] R. Drechsler, S. Eggersglüβ, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel and D. Tille, "On Acceleration of SAT-based ATPG for Industrial Designs", *IEEE Trans. on Computer-Aided Design*, vol. 27, no. 7. pp. 1329-1333, Jul. 2008.