The Dissertation Committee for Geewhun Seok
certifies that this is the approved version of the following dissertation:

Testability Considerations for Implementing an Embedded Memory Subsystem

Committee:

Nur A. Touba, Supervisor

Baxter F. Womack

Tony Ambler

Earl Swartzlander

Gary Hallock

Testability Considerations for Implementing an Embedded Memory Subsystem

by

Geewhun Seok, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2011

Dedicated to my family

Acknowledgments

I thank God for letting me finish the dissertation. The dissertation is a result of continuous discussion with Jesus. Jesus said "I am the vine; you are the branches. If a man remains in me and I in him, he will bear much fruit; apart from me you can do nothing (John 15:5)." I am grateful to my parents and wife for their love and support. I greatly appreciate their prayers throughout this endeavor.

I would like to thank to my Ph.D. committee members - Prof. Womack, Prof. Ambler, Prof. Swartzlander and Prof. Hallock for their guidance, understanding and support throughout the course of this research. Working with them has been a great experience and great fun. Thanks to my committee members for their ideas and invaluable feedback. I would also like to thank Professor Nur A. Touba specially for his guidance and support. Thanks to the Electrical and Computer Engineering staff for their assistance. Being a part-time Ph.D. student and a full-time working employee is an adventure that pushes a man to his creativity and production limits. I would like to express my gratitude to my friends for their encouragement and assistance. Special thanks to Hong Kim, Paul Policke and Mohammed Baker. Also, special thanks to my colleagues and managers, Paul Basset at my employer for their understanding and encouragement.

Testability Considerations for Implementing an Embedded Memory Subsystem

Publication No. _____

Geewhun Seok, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Nur A. Touba

There are a number of testability considerations for VLSI design, but test coverage, test time, accuracy of test patterns and correctness of design information for DFD (Design for debug) are the most important ones in design with embedded memories. The goal of DFT (Design-for-Test) is to achieve zero defects. When it comes to the memory subsystem in SOCs (system on chips), many flavors of memory BIST (built-in self test) are able to get high test coverage in a memory, but often, no proper attention is given to the memory interface logic (shadow logic). Functional testing and BIST are the most prevalent tests for this logic, but functional testing is impractical for complicated SOC designs. As a result, industry has widely used at-speed scan testing to detect delay induced defects. Compared with functional testing, scan-based testing for delay faults reduces overall pattern generation complexity and cost

by enhancing both controllability and observability of flip-flops. However, without proper modeling of memory, Xs are generated from memories. Also, when the design has chip compression logic, the number of ATPG patterns is increased significantly due to Xs from memories. In this dissertation, a register based testing method and X prevention logic are presented to tackle these problems.

An important design stage for scan based testing with memory subsystems is the step to create a gate level model and verify with this model. The flow needs to provide a robust ATPG netlist model. Most industry standard CAD tools used to analyze fault coverage and generate test vectors require gate level models. However, custom embedded memories are typically designed using a transistor-level flow, there is a need for an abstraction step to generate the gate models, which must be equivalent to the actual design (transistor level). The contribution of the research is a framework to verify that the gate level representation of custom designs is equivalent to the transistor-level design.

Compared to basic stuck-at fault testing, the number of patterns for at-speed testing is much larger than for basic stuck-at fault testing. So reducing test and data volume are important. In this desertion, a new scan reordering method is introduced to reduce test data with an optimal routing solution. With in depth understanding of embedded memories and flows developed during the study of custom memory DFT, a custom embedded memory Bit Mapping method using a symbolic simulator is presented in the last chap-

ter to achieve high yield for memories.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The role of DFT (Design-for-test) is to introduce logic which provides for or enhances the testability of the functional logic and then to use that logic for manufacturing test of the chip. Having said that, the main goal is to achieve maximum test coverage with minimum test time so that minimum DPPM (Defective Parts per Million) level with low cost can be ensured while shipping the part to the customer. This exhaustive testing becomes more and more important if the application of the chip is reliability critical (automotive, communication, and medical applications). Hence, it is required to ensure that we test every bit of the logic present on the chip and at the same time this testing should be correct such that it addresses the faults which can come up during the functional operation of the chip. A general System On Chip (SOC) normally has lots of digital logic and some analog components, the major ones being memories. High-performance embedded memory is a key component in VLSI Design. The benefit to using embedded memory is improved performance, multi-port memories, reduced power consumption, and dedicated architecture [1].

This chapter provides background on the issues related to testing of

embedded memory interface logic using scan-based testing. It also presents gate modeling of memory subsystems and Bit Mapping of embedded memories. Section 1.1 describes current testing strategies for memories interface logic. Section 1.2 describes scan-based at-speed testing for memories interface logic. Section 1.3 shows custom memory subsystem design modeling for ATPG. Section 1.3 describes the issues related to test time and data volume. Section 1.4 introduces Bit Mapping of embedded memories. The contributions and organization of this dissertation are provided in Section 1.5.

1.1 Testing Around Memories

As memories are often the largest occupants of silicon real estate, it becomes crucial to properly address the interfacing logic around them (shadow logic). In this section, several testing strategies currently being deployed across the industry to test this shadow logic will be presented, and detailed analysis of each methods will be provided. There are two main method applied for testing the shadow logic around the memories.

1. Memory bypass during scan-based testing - Black box model of memory
2. Memory write through feature during ATPG - Fully functional model of memory

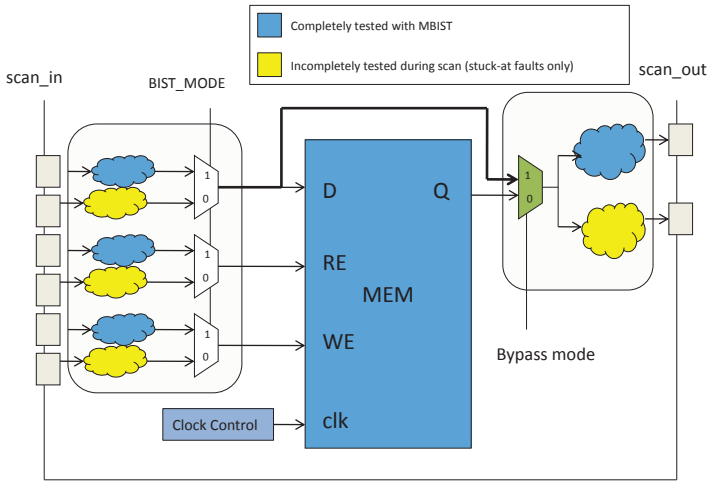


Figure 1.1: Scan-based Testing with Memory Bypass Feature

1.1.1 Memory Bypass During Scan-Based Testing

This bypass mechanism allows the observation of values driven by the input interface logic and at the same time it also gives controllability over the values driven onto the output interface logic. Figure 1.1 shows bypass wires which are implemented around the memory. The data values come from the input interface logic to the output interface logic during scan-based testing and thus bypass the memory completely. It can achieve high stuck-at coverage on the interface logic, but, since the bypass path is not the actual functional path, obtaining delay fault coverage on this logic would be incorrect. There is wire overhead associated with this strategy. This bypass logic also degrades the timing at the output side of the memory due to presence of an additional multiplexer. These paths are generally the most timing critical paths in an

SOC and hence, the additional multiplexer can actually limit the maximum operational frequency of the SOC.

In summary, the surrounding interface logic (shadow logic) is incompletely tested. Only some of stuck-at faults get tested, whereas, the delay faults remain untested. The logic that remains untested (or incompletely tested) is represented in Figure 1.1 in yellow.

1.1.2 Memory Write Through During Scan-based Testing

Most of the available industry standard ATPG tools [2] are capable of reading in the full functional model of memories (RAM/ROM) - either verilog or tool specific formats. This capability can help design to achieve complete test coverage around the memories. Using the functional model of memory helps to observe the values driven by input interface logic by actually doing a WRITE action onto the memory. At the same time, the output interface logic can be controlled by doing a READ action onto the memory. This "observability" and "controllability" of the interface logic allows the complete testing (both "stuck-at" and "delay" faults) of this logic. Also, the delay fault testing makes more sense here, because real functional paths which is from memory input to interface logic are used. Compared with memory bypass method, as can be seen from Figure 1.2, the yellow area is complectly tested.

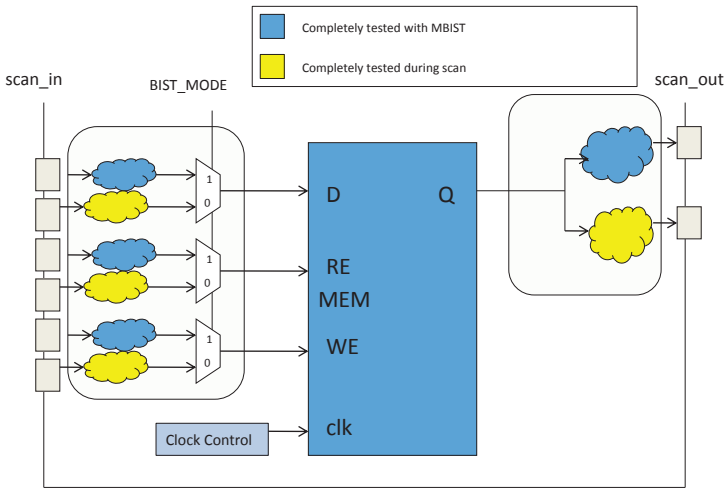


Figure 1.2: Scan-based Testing with Memory Write Through Feature

1.2 Scan-based At-Speed Testing for Memory Interface Logic

At-speed testing applies a test at the CUT (Circuit Under Testing) with functional speed to detect defects and slow paths due to process variation. If the CUT is run at high speed, it is possible to have many timing-related defects. It is mainly due to manufacturing problems which prevents the device to operate at-speed.

Figure 1.3 shows a breakdown of failed parts based on logic tests only. there were no unique stuck-at failures or slow functional failure. At-speed scan test can detect the majority of defective parts. At-speed functional test [3] also can screen out many defects but the cost of development and test

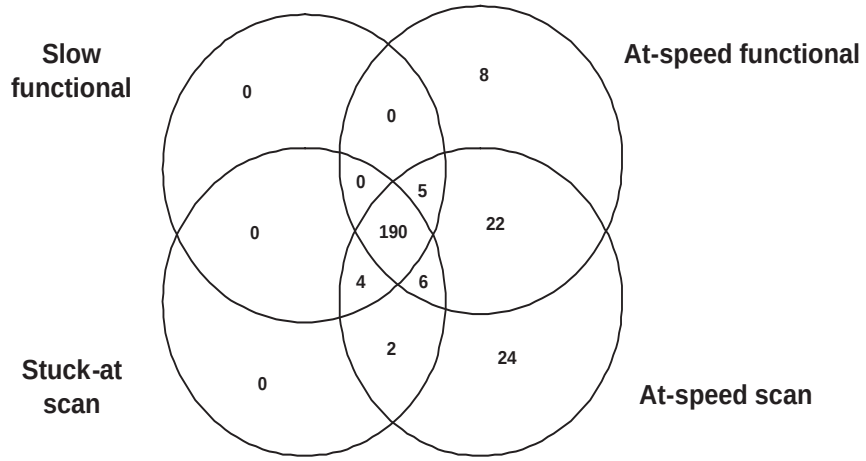


Figure 1.3: At-speed Scan Testing

time is high. At-speed scan and at-speed Build-In-Self-Test (BIST)[4] can solve these problems. At-speed scan can cover enough critical paths which BIST can not sensitize. There are major problems which make at-speed test difficult. First, the gate model netlist of custom circuit are often simplified with a behavior model like a black box model into memories. The paths of custom circuits are critical paths and these paths must be tested with delay testing. If the gate model does not have these paths then ATPG is unable to generate the patterns. Using an extracted gate model netlist from a schematic is the solution which can be replaced with the behavior model. However, an extracted model is an improper model that ATPG cannot understand. Manual jobs are required and often this injects human error. Second, current ATPG simulations are based on a zero delay simulation and require another timing simulation. Detecting timing related errors is difficult in zero delay simulation,

for example it is hard to detect race conditions in zero delay simulation.

1.3 Custom Design Modeling For Test Pattern Generation

Memories are good candidates for custom design. Custom memory design is an important part of embedded processor design to get high performance. The custom design flow is different from standard design and requires a separate design hierarchy. The RTL and transistor level models are developed separately. The critical thing is tht the transistor level model needs to have same functionality which is described in RTL. ATPG tools use the gate-level netlist for test generation vector simulation [2]. As can seen from 1.1.2, the memories need to be modeled with a full functional model to be enable at-speed testing. Translating the transistor-level netlist into the ATPG tool's model is an important step to get the expected output from silicon debug. Ensuring the equivalency checking between the ATPG tool's model and the schematic of the actual design is critical.

1.4 Test Time and Data Volume

Minimizing the number of patterns is important for scan-based at-speed test. Traditionally only stuck-at test patterns were used. However, the increase in the number of vectors required for Scan-Based At-Speed Test to get complete coverage is about 2.5X or more [5]. Popular methods to reduce test time are using multiple scan chains and scan compression logic.

1.5 Contributions and Organization of the Dissertation

The focus of this dissertation is a detailed study of testability considerations in the design of embedded memory subsystems. This study includes scan-based testing, new DFT logic, ATPG modeling, verification of models, and a new Bit Mapping Methodology for custom embedded memories. The main reason to tackle the design of embedded memories is the industry trend to at-speed scan test which is able to reduce cost compared to functional testing. Due to the increase in the number of patterns from scan-based at-speed test, reducing test data volume is important. A new scan reordering method is introduced to reduce test data with an optimal routing solution. With in depth understanding of embedded memories and the methodology developed during the study of custom memory DFT, a custom embedded memory Bit Mapping method using symbolic simulation is presented in the last chapter. The main contributions are the following.

1. A register based testing method with X masking logic is proposed to prevent Xs from memories, increase compressor mode test coverage, and reduce the ATPG pattern count. The benefit of doing read and write at the same cycle in memories is addressed. 6T-cell based memories are modified to be able to do this behavior during test mode.

2. A novel ATPG pattern generation scheme is proposed to prevent Xs from memories. The behavior of reading before writing into memories is prevented using virtual memories information during ATPG run-time.

3. The detailed design flow to verify the validity of all the generated vectors make it ensure that a custom macro design's ATPG model are built to correctly emulate the actual design. Using an RC verilog switch level simulator reduces simulation times dramatically.

4. To prove the practicality of the approach, the design and methodology were implemented during Qualcomm DSP(Digital Signal Processor) core design project.

5. A Novel scan chain partitioning scheme is proposed which test set and test time reduction along with the optimal routing inside each partition. With the increased demand for delay testing, reducing test data is needed to save the test costs.

6. Within a limited design time, delivering accurate logical to physical Bit Mapping information is an essential process to get high yield from an embedded memory. A novel Bit Mapping method using symbolic simulation is introduced in capture 5. This method can be applied to reduce simulation time for large memories. The overall automated flow is also presented to handle custom embedded memory design.

Chapter 2

Write-through Method for Embedded Memory with Compression Scan-based Testing

Demands for low defects per million (DPM) rates are increasing as process technology scaling is able to increase transistor density and add more functionality to the integrated circuits. For stuck at fault and delay testing, Scan-based testing in conjunction with ATPG is the preferred approach to reduces DPM compared to functional testing. However embedded memories have been a challenge to ATPG gate level simulation due to limitation of gate level generation method and the additional logic needed to prevent unknowns (X's) to be propagated from memory during ATPG testing, this X-propagation becomes more of an issue when the design has a test compressor. This chapter examines the challenges of ATPG memory write through method on the design with chip test compression logic and proposes new design strategy and ATPG pattern generation method. The proposed design will make the memory look like a one dimensional set of registers and ATPG pattern generation method will support write through mode without Xs propagation.

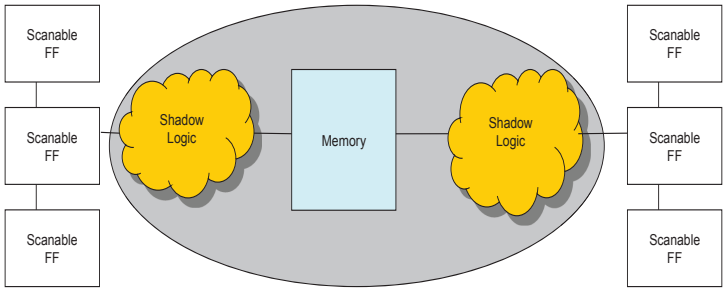


Figure 2.1: Shadow logic around memory

2.1 Introduction

One of the most common design-for-test strategies for SOCs is scan design which structures a general design to behave as combinational logic gates. These gates models have been understood by state-of the art ATPG tool and successfully detected faults with patterns which are generated from ATPG. However most processors and SOCs have a mixture of large and small memories which are based on SRAMs, CAMs, ROMs, register-files, FIFOS, and many other regular structures Even though the memory structure itself may be covered using other testing techniques like Built In Self Test (BIST), there is a lot of logic between the regular structure and the rest of the design (shadow logic) as shown in Figure 2.1.

Several ATPG vendors have provided the capability to model memory structures with a behavioral model [6]. In [7], Sitram and Sanjay presented impact and cost of handling memory models for ATPG in which they showed improvement of the stuck-at and transition fault test coverage with memory

modeling methodology.

Another major trend for SOC is to use on chip test compression [8]. A design with a compression methodology can reduce test data volume and testing time. However one of the major problems with an on-chip test compressor is that, if there is feedback logic like memory in the scan chain whose contents cannot be uniquely determined to be known values during simulation (also called X's or unknown logic values), then the entire signature will get corrupted and will be of no use. This will result in masking many test patterns that will affect coverage. Sources of X's include uninitialized and uncontrollable memory blocks, bus contention, floating buses, multiple clock domains, and inaccurate simulation models [9].

To remove unknown logic values, a significant amount of engineering effort is required through each design stage. There is a possibility that an ATPG memory model generates Xs, because it is difficult for the ATPG tool to comprehend all the memory operation and control signals for the memory. Moreover, if an SOC has in addition to compiler memory a lot of custom memories, making the commercial ATPG understand these memories may be difficult. This chapter proposes a new design and testing methodology to apply a write through method which will improve test coverage and reduce scan overhead for hard macros. The effect of using memory model with on-chip test compression will be described. It provides a solution to prevent Xs which are generated from memory when ATPG does not have controllability of read and write operation of the memory. This chapter is organized as follows:

Section 2.2 will introduce the method of modeling memory for ATPG in the custom design flow and how the memory model fits into the flow, Section 2.3 addresses current memory scan testing methods, Section 2.4 details the register based testing strategy and Section 2.5 proposes virtual Xs prevention logic for ATPG. Section 2.6 report the experimental results from an industry embedded processor with mixed memory and this chapters conclude with Section 2.7.

2.2 Memory Modeling for ATPG

A memory model is for describing generic memory to represent array behavior. A typical gate netlist can be generated from a Synthesis flow (Figure 2.2-(1)) and most of the logic in a custom design also can be translated into gate level netlist(Figure 2.2-(2)). However structures like memory which has bi-stable logic cannot be automatically translated and even if generated, a memory netlist can not be understood by ATPG tools [10]. From a test perspective, memory has been often modeled as a black-box and the logic surrounding the memory has not been covered due to propagation of X values from the output of black boxed memory. Figure 2.1 shows the uncovered logic by ATPG if a memory model was not used. Fault coverage between the scan-frontier and the memory boundary will be lost due to unknown status.

A method called write through memory, which uses a memory model, can increase the test coverage. In other words, the surrounding shadow logic can be tested by the write through method. The write through method is also important to do delay testing because the many paths through memory

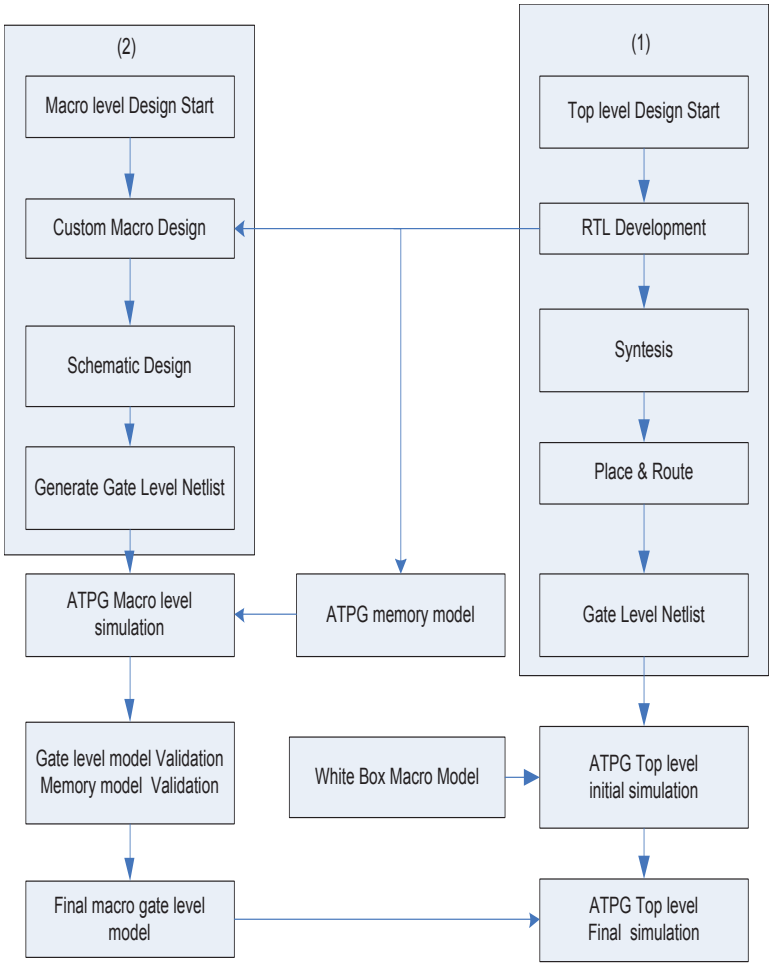


Figure 2.2: High Level Overview of Design Flow with Custom Embedded Memory Showing The ATPG Flow and The Memory Model

do essential roles of justification on many delay paths. It is good to know that the memory model is not for testing memory itself. Instead, embedded memory can be tested by BIST, functional test or memory scan.

2.3 Current Memory Scan Testing Methods

In this section, two design approaches to test the shadow logic through scan will be addressed. The first is the bypass method. It is most common and simple method. It uses wires to bypass memory from the data input to data output and locates the MUXs to output of the memory. It has a MUX delay at output of the memory. Due to this MUX delay this design is often improper for a critical path. This method has issues with at-speed testing coverage because the memory is modeled as a black-box.

The second method for memory during ATPG is to use a write through mode, it uses a memory model and requires the control of read and write enable signals and the memory clock. This method needs sequential ATPG to test the shadow logic. Sequential ATPG patterns have more than two pulses during capturing time. This allows input values to propagate to the output of the memory like one cycle for writing and another cycle for reading into memory. This technique is attractive because it can provide at-speed testing coverage. It can cover up macro to macro paths which are often timing critical paths. However, Currently the write through method has the following drawbacks:

1) Xs propagation; the ATPG memory model can generate Xs if the memory accesses a location that was not written to. It is difficult for the ATPG

tool to understand all the combinations of read, write, data and address. This may result in reading memory contents that have not been written to already through previous patterns. Due to the limitation in tool capability for write through mode, the number of patterns is increased to cover up to desired test coverage and if the design has a scan compressor, the generated Xs give more impact to pattern count. Table 2.1 shows how Xs from memories can impact to static ATPG test coverage. The test coverage came from the same design with different setup mode, uncompress and compress mode. If there is no Xs propagation, test coverage of compress mode is equal to or greater than uncompress mode, but due to Xs from memories, it suffered with Xs in scan chains. Because the outputs of the scan chains must correspond exactly to the expected simulated states, any don't-care bits will corrupt other scan chains. Don't-care states from memories need to be eliminated if XOR compression is used. By doing Xs source analysis the percentage of Xs due to memory is found to be about 34 percent of the total Xs. Unfortunately, even some Xs can be eliminated from the design, uncontrolled memories affect the contents of scan chains during write-through mode due to improper control of write and read enable signal. To avoid further testability transgressions during automated

Table 2.1: The Effect of X Propagation in Test Compression

Test Mode	Test Coverage	Pattern Count
Uncompress mode	98.57%	1247
Compress mode	93.24%	1743

test pattern generation, the ATPG tool requires paying special attention to the support needed for memories. However, it is difficult for an ATPG algorithm to understand each custom macro design and have proper control of the read and write enable signal control. In order for the ATPG algorithm to create patterns which pass data through memory, one of the requirements is that the write control lines need to be available from the top level module, but it is impossible to add an additional top level pin just for ATPG.

2) Improper memory model and verification time; Many custom memories can not be described as simple behavior codes. For example, memory can have multi-port read/write access. Even after these modeling, it requires a lot of time to verify ATPG model vs. real circuit behavior.

2.4 The Register Based Test Strategy for Embedded Memory

The basic philosophy behind the register based test strategy is to make the memory look like a single register during testing mode and force writing to occur before any read operation in this single register. This modification is done during the design phase itself. The only requirement is to use a small number of gates to constrain the address and X masking logic. The next two sections will show the proposed methodology for two types of memories: multi-port single cycle, and traditional SRAM 6T cell based memory.

2.4.1 Memory that can Write and Read at the Same Cycle

Like high performance multi-port register files, some memories can write the data at the rising edge and read the data at the falling edge. The benefit from this behavior with constrained address values is that scan patterns require only two capture pulses for stuck-at-fault detection during write through, The first pulse is for write and read into the memory. The second one is for capture from scanable flip-flops. The register file is handling high timing critical path which is not allowed to use the bypass method (Section 2.3) which requires MUXs in the memory output path. Usually accessing memory data through a read operation is a timing critical. It is important not to insert additional delay into this path. Figure 2.3 shows a multi-ported register file which has 4 write and 8 read ports. During ATPG mode, all write and read address ports are constrained and allow to write and read into memory at the same cycle. To allow accessing memory locations per wordline simultaneously, different write addresses (wAddr0, wAddr1, wAddr2, wAddr3) and read addresses (rAddr0, rAddr1, rAddr2, rAddr3, rAddr4, rAddr5, rAddr6, rAddr7) are applied to each memory during ATPG mode. 32 bit data memory cells are modeled as 32 flip flops. It is worth to mention that the address paths are tested by MBIST and the primary concern is the data path.

The modeling og memory as flip-flops has the following advantages:

1. For transition delay testing, it requires only three cycles for capturing. For example, Write “0”, Read“0” and capture (W0R0, W1R1, Capture)

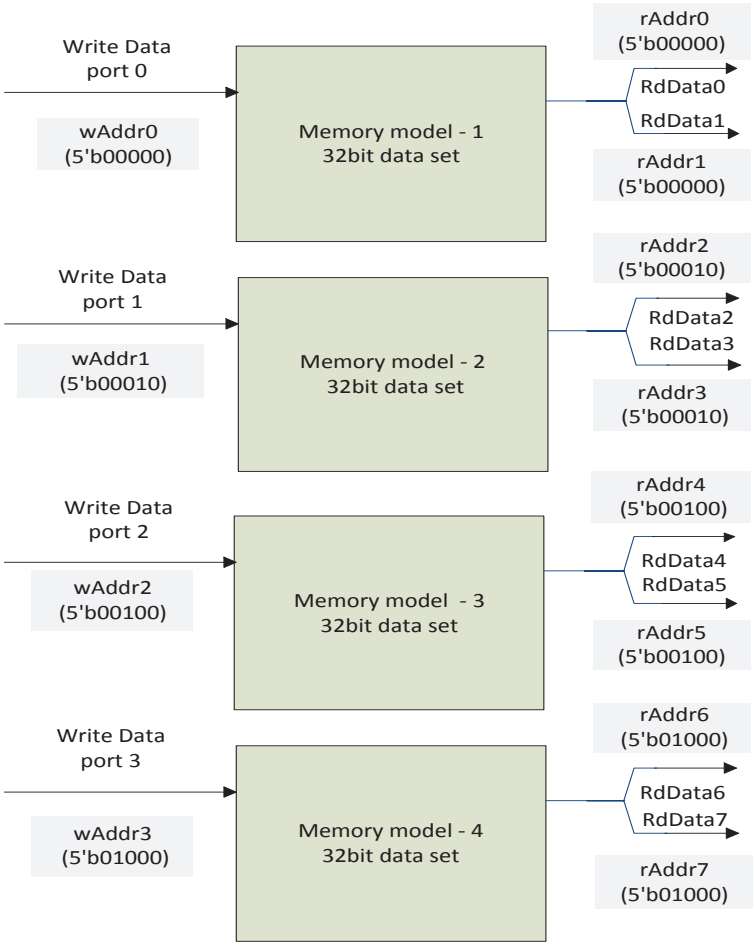


Figure 2.3: The Register Based Testing for a Multi-ported Register File

are needed rather than five cycles for capturing (W0A1, W1A2, R0A1, R1A2, Capture). this reduces the sequential depth for ATPG.

2. It does not need complicated memory modeling and RAM-sequential ATPG.

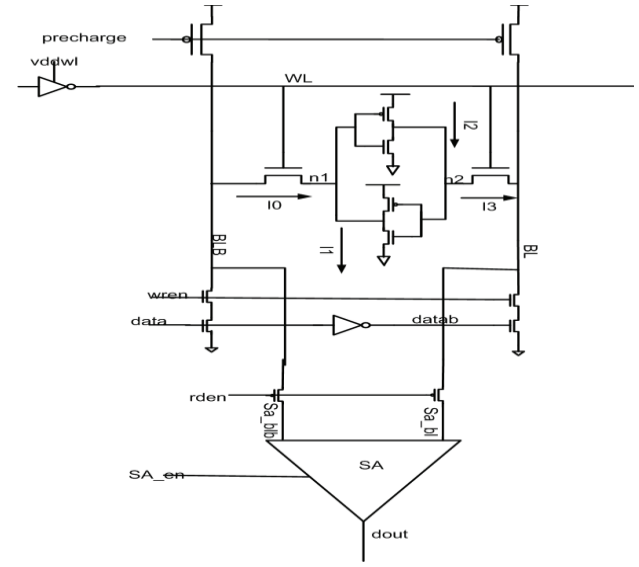
3. It does not need time-consuming fault coverage management to know which paths are not covered by MBIST because the proposed method is the scan-based single solution.

2.4.2 Register Based-ATPG for 6T Cell Based Memory Testing

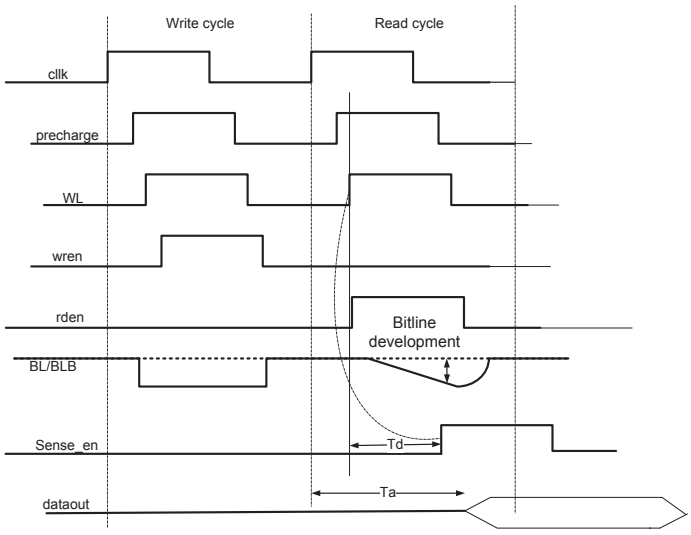
Typical 6T cell based memory array don't support read and write operation at the same cycle during functional mode, but with minimal modification, a register based test strategy can be applied with changing of sensing data on the sense amplifier during ATPG test mode.

Figure 2.4(a) and 2.4(b) show a typical memory cell and waveform for major signals on a 6T-cell based memory. The access starts by asserting the word line based on the row address and then the read or write signals assert write enable or read enable. In the case of a write operation one side of the column will be pulled low and the other side stays high (logic 1) based on the write data. During normal read operation when the cell word line asserted a current flows from BL or BLB based on the stored data and a voltage difference get developed between BL and BLB.

The sense amplifier signal (sense_en) is asserted to evaluate the read



(a)



(b)

Figure 2.4: (a) 6T Based Memory Array (b) Major Signal Waveforms

data. It is desirable to do read and write in the same cycle so there will be no issue in doing so if the read and write uses the same address as the bitlines are shared and the write operation already is pulling one side of the bitline low so the sense amplifier will be able to detect the data to be written. Even though the cell is not used to read, for testing purposes there is full controllability and observability of all the signals (output data bus) coming out of the array.

2.4.3 X Masking Logic for Scanable Flip-Flops to Output Data Paths

As pointed out in Section 2.3, Xs from memory can propagate to the test compressor. Even with a register based test strategy, Xs can be generated from the register which is a non-scanable cell. This happens when the first writing occurrs. To prevent the Xs during the first capture cycle (C1), Figure 2.5 shows how proposed X masking logic can prevent Xs. The final output (clock) will be input clock of scanable the flip-flops which are located on output data paths. Note that only the first cycle of capture is masked and other capture cycles are allowed to observe data from memory. ATPG_mode is asserted during ATPG. Expected clock behavior (clock) is shown in the waveform. Because scan able FFs are controlled through this conditional clock and does not require any gate in the data path, timing impact to critical path is minimal. Note that this approach is compatible with any commercial ATPG tool.

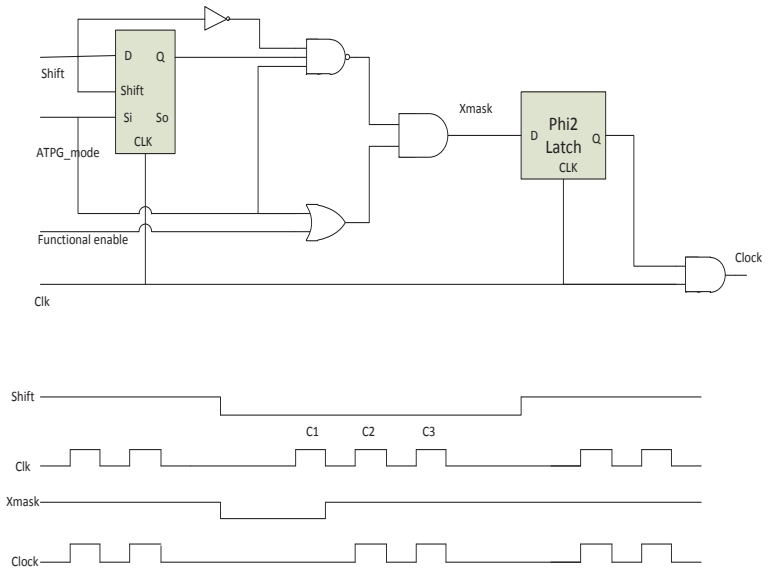


Figure 2.5: X Masking Logic

2.5 X Prevention ATPG Method for Custom Embedded Memory

In this section, an ATPG method is introduced to avoid generating Xs from the memory. This method can be applied when the memory is customized and the ATPG tool has difficulty to understand the write and read enable of memory. This method may be used when the memory can do write and read in the same cycle.

2.5.1 Virtual X Prevention Logic for ATPG

Figure 2.6 provides an overview of how X prevention logic will produce non-X patterns using ATPG.

The virtual memory is used to keep the status of each memory cell.

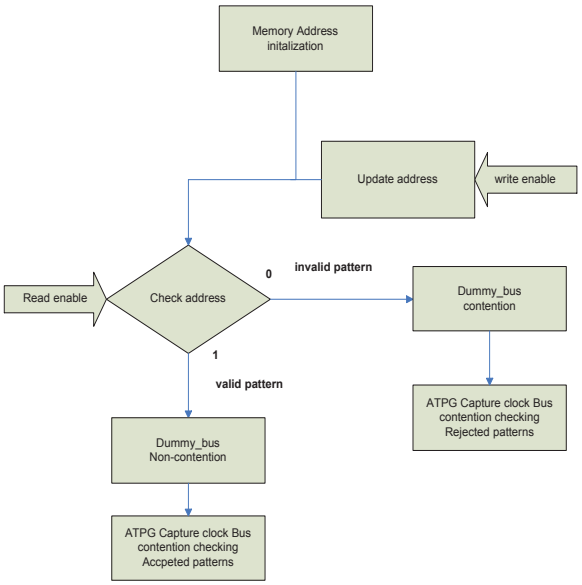


Figure 2.6: Block Diagram of X Prevention Logic Behavior

Figure 2.7 shows how virtual memory addresses are initialized and updated. The virtual memory has same address size as the actual memory. During the ATPG pattern generation step, if a real memory address location is written in first the matched virtual address is updated as "1". If the memory address was read without being written, initialized bit value "0" on the virtual memory address indicates that the address has not been written and it is expected to generate Xs from the memory. Dummy bus contention logic generates the contention based on checked address. ATPG BUS contention detection algorithm will reject the pattern and regenerate the new pattern.

Bus contention checking can be done through clock-on cycle, which occurs at the instant the clock is asserted, but prior to the changes which would occur from propagating sequential gate inputs to their outputs. Figure

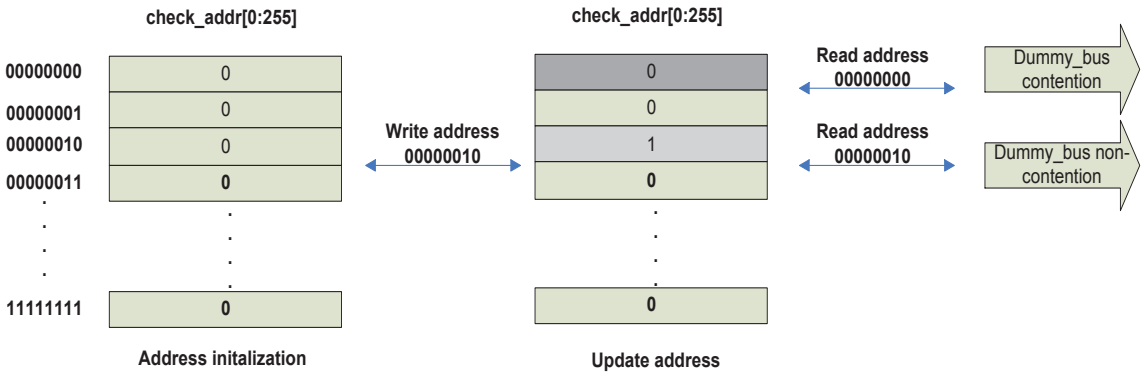


Figure 2.7: Address Initialization and Updated Address

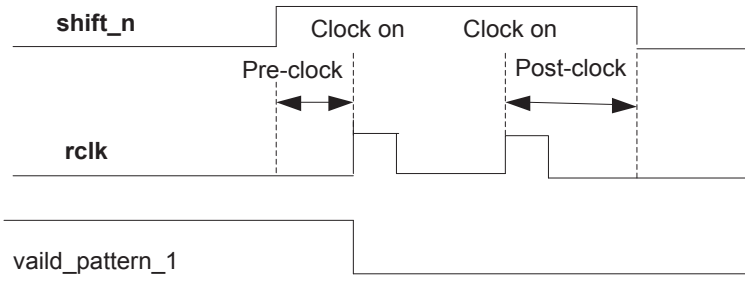


Figure 2.8: Contention Checking Method

2.8 shows the different portions of a typical capture clock cycle and where bus contention checking occurs. Clock-on contention checking identifies invalid read enable signals. If any pattern has this type of contention, it is discarded and other patterns are attempted. Bus contention checking can be done through two ways which are contention checking after pattern generation and contention checking before pattern generation (ATPG methods). Use of ATPG methods is more CPU intensive [2]. Figure 2.8 shows valid_pattern_1 which is deasserted when an invalid read enable signal is detected. The equations for valid_pattern_1 is described below. WE is a writing event and RE is a reading event into the memory location.

valid_pattern_1 = valid1 [addr] & RE

valid1 [addr] = $\sim$ reset & WE

A bus with three-state buffers is used to generate dummy bus contention. The Verilog code to implement this logic is shown in Figure 2.9.

2.5.2 Two-Step ATPG Pattern Generation

To reduce the pattern generation complexity two modes (regular ATPG mode and write through mode) were applied one by one. The regular mode usually means scan-based memory bypass mode. It is noted that the bypass method is still useful in cases where the memory surrounding logic is well covered by the bypass. If the path through memories has room to put Bypass MUXs on data path this mode can be still be applied, but the write through mode needs to be applied if the design has uncovered faults in the surrounding

```
module check_pattern_valid (wclk, rclk, a, scan_n);
  input wclk;
  input rclk;
  input [7:0] a;
  input scan_n;
  reg   valid_pattern_1;
  wire  [1:0] dummy_bus;
  reg   check_addr [0:255] ;
    event WRITE_OP;
always @(wclk or a) if (wclk) begin
 check_addr [a] = wclk;
 #0; ->WRITE_OP;
 end
always @(rclk or a or WRITE_OP)
 if (rclk)  begin
   valid_pattern_1 = check_addr[a];
 end
  wire iclk, iwclk, wi_clk, i_clk;
  wire [1:0] drv;
  wire [1:0] hitor;
  and scan_rclk (iclk, rclk, scan_n);
  and ud0(drv[0], iclk, 1'b1);
  and ud1(drv[1], iclk, iwclk);
  and ud2(iwclk, ~valid_pattern, ~wclk);
  or  uor_0(hitor[0], drv[1], drv[0]);
  or  uor_1(hitor[1], drv[0], drv[1]);
 bufif1 ub0_0(dummy_bus[0], 1'b1, hitor[0]);
 bufif1 ub0_1(dummy_bus[0], 1'b1, drv[0]);
 bufif1 ub1_0(dummy_bus[1], 1'b1, hitor[1]);
 bufif1 ub1_1(dummy_bus[1], 1'b0, drv[1]);
  initial $readmemh ("./ram1.data",check_addr);
endmodule
```

Figure 2.9: Verilog Code for 256X8 Memory of X Prevention Logic

logics. Below is a sample of the steps of generating ATPG patterns with embedded memory:

1. Set to regular mode (Bypass mode)
2. Run regular ATPG - Patterns set 1
3. Write fault list
4. Remove regular mode constraints and apply write through mode constraints
5. Read fault list which comes from step 3
6. Run sequential ATPG - Patterns set 2

2.5.3 Verification of X Prevention Logic

X prevention logic is no-faulted during ATPG pattern generation and does not affect functional and other test mode. ATPG models with X prevention logic must be robustly and effectively verified that the patterns generated by the ATPG tools based on the gate level models do indeed represent the schematic therefore the silicon. The scan equivalency between these gate and transistor level models were verified using industry standard RC Verilog switch level simulator [11].

Table 2.2: The Effect of Register Based Testing and X Prevention ATPG Method

(A)

Design	Regular Mode	Regular Mode + Write through Mode (2 steps)		
		Coverage	% of Xs	Pattern Count - Basic(Seq)
A	49.63%	91.59%	34.71%	17(316)
B	67.57%	93.11%	36.83%	10(173)
C	97.80%	97.85%	34%	1091(345)

(B)

Design	Register Based Testing		
	Coverage	% of Xs	Pattern Count - Basic(Seq)
A	97.21%	0%	16(299)
B	98.02%	0%	9(156)
C	98.51%	0%	1063(142)

2.6 Experimental Result

The proposed methods were implemented in a 28nm Qualcomm DSP core design project. Table 2.2-(A) shows the coverage, percentage of Xs from scan chain and pattern count of design before applying proposed schemes. Table 2.2-(B) shows effectiveness of proposed schemes. The test coverage is increased compared to regular mode and write through mode. The number of patterns is reduced compared to write through mode. Design A is a register file which has multi-port read and write ports. Design B is L2 data which uses 6T Cells. Design C contains designs A, B and more custom embedded memories. The regular mode means bypass mode (Design B) or non-bypass mode (Design A). The reason with non-bypass mode of design A is due to

a timing critical path through the memory. So it has constrained memory data outputs for regular test mode. Original designs A and B are modified to apply register based testing. Bypass MUXs (2 gate delays) are removed from Design B, and AND/OR gates (1 gate delay) are applied to constrain the address ports for register based on testing. X masking logic is also introduced to design A and B for each data output scan cell's clock. For other memories (except design A and B) X prevention ATPG method is used to eliminate Xs when memory address locations are read in first before being written. Design C is simulated with the test compressor. Non-X values on the scan chains make the compressor more effective.

2.7 Conclusion

This chapter addressed the issue of Xs propagating to the test compressor. When an SOC design uses custom embedded memories, the ATPG tool could not control write and enable ports properly during the write through mode. A register based test strategy and X prevention memory model is presented for ATPG, which increases the overall test coverage. The test compressor logic was efficiently utilized by preventing Xs from memory. Experiments with industry applications were performed for performance evaluation and verification of the proposed logic.

Chapter 3

Verification of Gate Level Model for Custom Design in Scan Mode

In this chapter, a flow to validate scan equivalency between the gate level netlist of a custom circuit design and the schematic using delay aware industry standard tools will be described. General ATPG tools have not been tested with delay and results predicted based on 0-delay netlist environment. Industry has struggled with this 0-delay environments and required thorough verification. Custom designs are designed using transistor level models and tools. The transistor level model needs to be translated to the gate level to be used by the DFT tools. It is essential to have a robust and accurate flow to verify the gate level netlist with delay. To consider the delay effect, the flow uses an industry standard RC Verilog switch level simulator to effectively and thoroughly verify the scan equivalency between these gate and transistor level models. This is the first time that RC Verilog switch level simulator has been used to verify the scan equivalency.

3.1 Introduction

One important view of the custom macro is a gate level model to be used for the Automatic Test Pattern Generation (ATPG) tools like TetraMax [2]. These ATPG patterns are used to screen for manufacture stuck at fault and transition tests for these custom circuit designs used on the chip. The patterns generated by the ATPG tools which take in gate level model are used to verify the silicon. If the gate level model and schematic are not equivalent then the patterns that been generated by ATPG models do not represent the actual silicon. This adds more challenges to the silicon debug because when patterns fail one has to determine whether it is pattern issue or real failure. Debugging failing ATPG patterns can be extremely difficult and time consuming. It is essential to verify that the gate level model and the schematic of the actual design are equivalent [12]. Figure 3.1 shows where the gate level model view fits in the design flow of the custom macro. The Register Transfer Language (RTL) used to describe the custom macro often doesn't include scan in it and if it does, the scan is added through a manual process that needs to be checked against the schematic for equivalency. The equivalency checking between the RTL model and the SPICE model which is done on box 4 of Figure 3.1 is often targeted only for functional mode and does not cover scan mode because the full details of the scan mode behavior may not be included in the RTL model. The two models (RTL and schematic) could be equivalent during functional mode but different during scan. The flow to generate gate level model (box 6 of Figure 3.1) from schematic differs from one project to another but due to

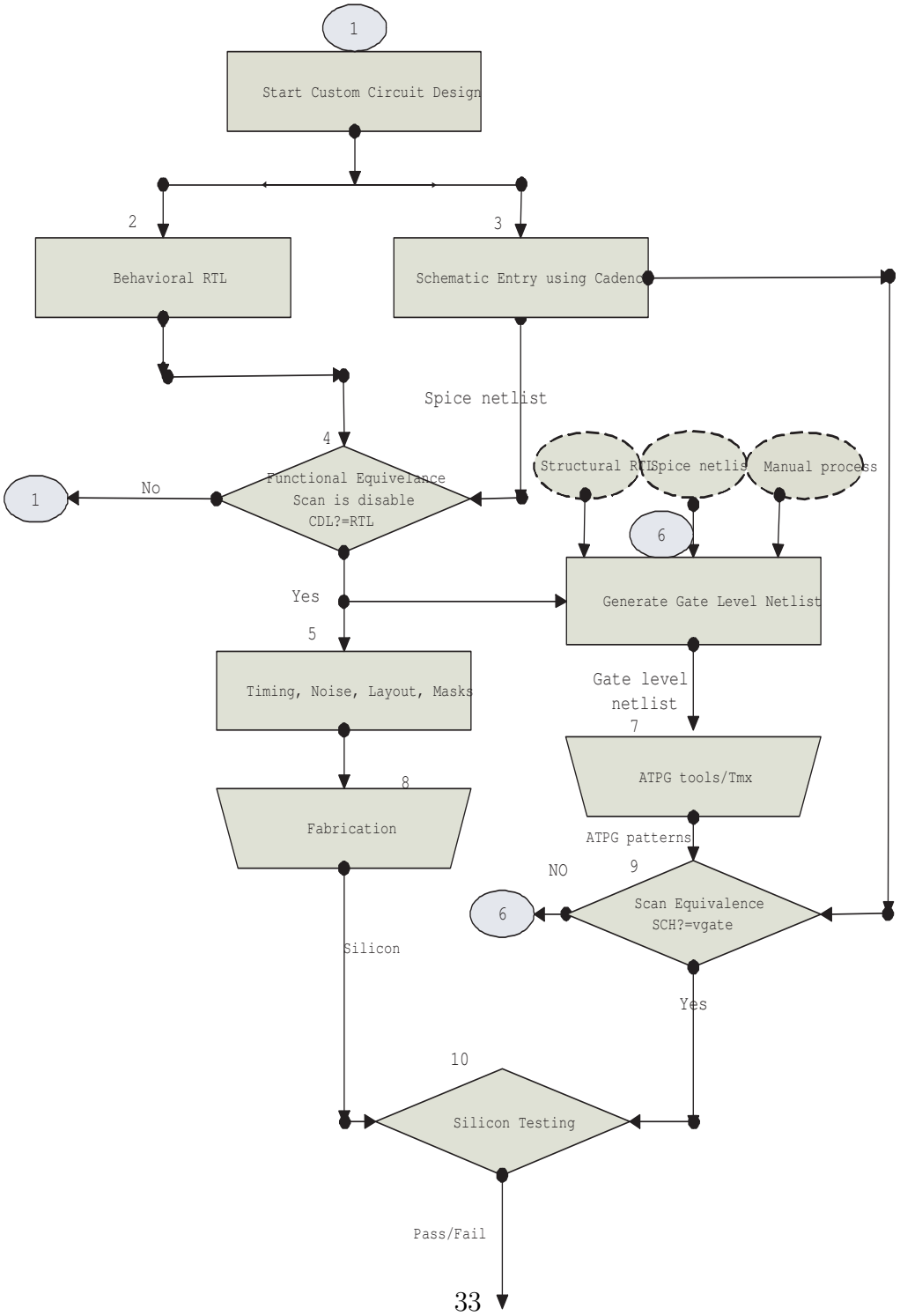


Figure 3.1: Custom Design Flow

tool limitations and design complexity, the process in many instances requires manual edits of the netlist or additional constraints/assertions to guide the tool, this makes the model creation process very error-prone.

This chapter presents a flow to robustly and effectively verify that the patterns generated by the ATPG tools based on the gate level models do indeed represent the schematic and therefore the silicon. This chapter is organized as follows: Section 3.2 introduces a high level view of custom design flow and how the gate level netlist fits into the flow, Section 3.3 details how the gate level model is validated and its equivalency with schematic is verified, Section 3.4 shows the experimental results and the chapter concludes with Section 3.5.

3.2 Design Flow

High performance and low power SOC design requires a comprehensive strategy and multi-level optimization from software to hardware. For the hardware design, all different design styles need to be exploited. One of the critical decisions that must be made when designing a chip is what portions of the logic is going to be implemented using a design and what part is synthesized. The decision whether to select custom design versus synthesis is based on complex trade off between achieving high density, better timing, lower power versus added complexity, resources and schedule. Memories (SRAM/CAM), TLB's, and register files are good candidate for custom design. Once it is decided to real a part of the logic using custom design, that logic will be separated out and put into a new design hierarchy. For custom semiconductor chip designs,

the RTL and transistor-level models are developed and verified using separate CAD tool suites, for most of the design, but are intended to model the same function. Once complete, the RTL level and transistor-level models must then checked to ensure that they represent the same Boolean function [13]. One way to do that is by translating the transistor-level netlist into a gate-level model through model abstraction and then using Verilog equivalency checking tools like Verplex [14] to verify functionality. Another way to verify equivalency between the RTL and the schematic SPICE netlist is by utilizing the switch level simulator like ESPCV [15]. Both approaches have advantages and typically a flow will use multiple approaches to verifying correctness.

An illustrative custom design flow is shown in Figure 3.1 where it starts with high level description of the intended functionality and the different spec such as timing, area, and power. The next step is to generate a schematic for the logic using schematic capture CAD tools like Cadence Virtues custom design platform. The schematic can be a mix of standard library cells and custom cells that are built up from the transistor level typically with a complex hierarchical structure in the design. After generating completion of the optimize design that implements the required functionality described in the RTL, functionality of the custom design can be verified against the behavioral RTL using the ESPCV tool from Synopsys. ESPCV is switch level simulator that can read in a design in both a behavioral RTL format and a transistor level netlist format and attempts to perform a symbolic, formal verification of their equivalency. Since quite often the RTL does not fully model the details

of the DFT features built into the design, these features must be disabled and no verification is done on them. Many tools in the design flow do not deal well with transistor-level designs so once the design is complete the transistor level netlist can be translated into a gate level netlist for these tools. Most of the logic in a typical custom design can be automatically translated into logic gate logic abstraction tools like the Verplex tools from Cadence. However structures with more complex behaviors, like SRAM cells, sense amps, and complex latch structures can not be automatically translated and must be manually modeled by the designer. There is the potential for errors to be introduced due to the manual modeling steps - and even the abstraction tools are not error free - so it is desirable to have an efficient gate level model validation flow.

3.3 Gate Level Model and Schematics Validation for ATPG

The gate level model and schematics validation process consists of three steps: 1) Run through the ATPG tool to generate patterns 2) Use HDL Verilog simulation to validate the patterns against the gate level model 3) Validate through ESPCV with the RC Switch level model generated from the SPICE netlist. Figure 3.2 is an overview for gate level model validation flow. This flow is described from the standpoint of verifying ATPG patterns and DFT functionality but the same principles can also be applied to functional mode verification as well.

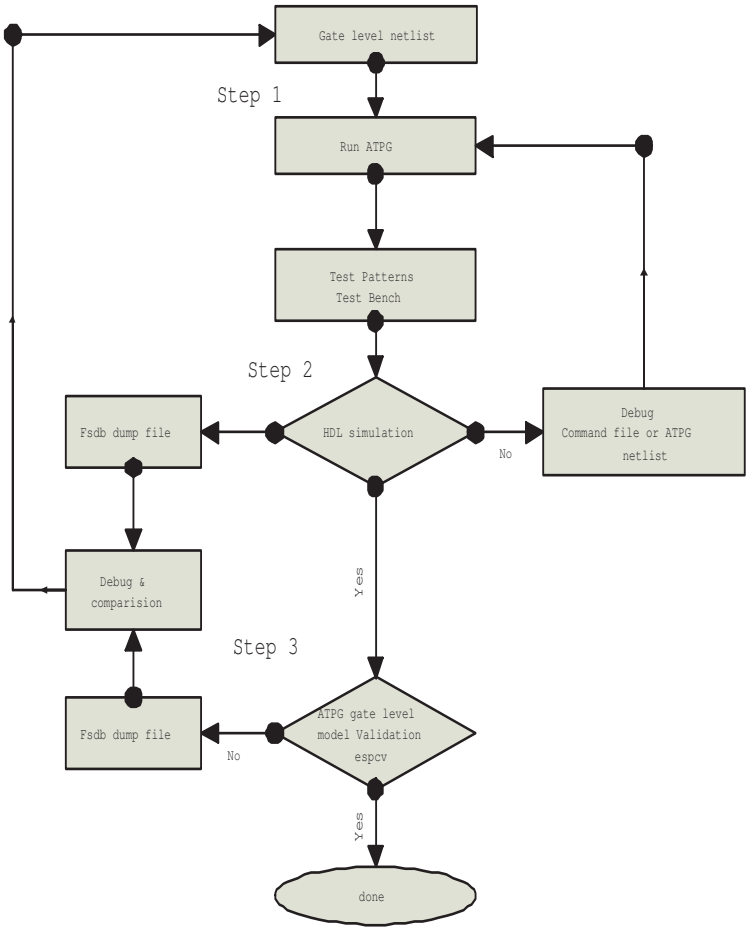


Figure 3.2: Gate Level Model Validation Framework

3.3.1 Step 1. Run Through ATPG Tool

Before attempting to generate ATPG patterns, the ATPG Tool first does a thorough validation of the gate level model from a DFT-compatibility standpoint. The main goal of this step is to insure the gate level model passes a series of scan design rule checks. After the DFT DRC stage ATPG patterns are generated with the goal of achieving 100% fault coverage. In a custom macro there are often circuits that are difficult to control and/or observe so the coverage is likely to be well below 100%. To achieve accurate fault coverage there are times when non-standard gates need to be changed to APTG friendly standard gates - for example bit line keepers must be modeled in a way that the tool understands they just preserve a node's state but don't actively drive it. Many of the clocking and control strategies used in custom designs may confuse the tools so that during the DRC checks and pattern generation the ATPG tool may believe there are errors causing broken scan chains and invalid input control. These false errors can prevent successful pattern generation until the offending circuits are re-modeled in a tool friendly manner. Finally, the actual memory cell array can be modeled using built-in memory primitive models which have advantages over using a detailed cell-level model in terms of simulation time and complexity. This memory model enables the tool to test shadow logic outside the memory model. Once patterns have been successfully created the tool can create a test generation pattern output file and Verilog test bench that will be used in last 2 steps.

3.3.2 Step 2. Validation Through HDL Simulation

The second step in the verification flow is to use an HDL simulator like VCS or Modelsim to validate that the ATPG tool was correctly interpreting the gate level model by simulating the application of the ATPG patterns to the design. Problems in the gate level model, invalid ATPG input constraints and other problems can result in ATPG patterns fail to produce the expected output results. Failures in the ATPG pattern validation can be debugged using standard RTL simulation debug tools by creating VCD or FSDB waveform files for viewing by Novas nWave. The FSDB dump file will also be used in step 3 if the ESPCV find mismatches in the transistor level verification.

3.3.3 Step 3. Validation with a Golden Model

Even with verification of the ATPG patterns against the gate level model RTL simulation there may be failures on actual silicon tests due to:

1. ATPG results predicted based on 0-delay RTL environment
2. Imperfect gate level model creation flow.

In this flow, ESPCV is applied to the problem of verifying that the gate level model correctly reflects the transistor level design.

ESPCV is a symbolic simulator that has been tailored to perform custom circuit equivalence checking. It is designed to provide functional verification coverage of a Verilog reference design against a SPICE netlist or Verilog switch-level design [15].

ESPCV provides two modes which are binary and symbolic mode. The tool is actually primarily intended to be used as a symbolic simulator to verify the very complex functional modes of the block under all possible input stimulus. For ATPG pattern verification the flow uses the binary mode of ESPCV to quickly simulate the application of the ATPG patterns to the design. ESPCV binary mode is much faster than transistor level simulators like HSIM and NanoSim which have also been used for this sort of verification. The flow to generate the golden model is shown in Figure 3.3. First the ESPS2V [15] utility translates the SPICE netlist to a golden RC verilog switch-level netlist using a configuration file which has port information. This netlist is annotated with transistor widths and lengths and process information. This simple step makes it possible to run ESPCV's RC mode algorithm that dynamically resolves strength issues and automatically calculates net delays to correctly resolve the behavior of things like SRAM cell write operations and timing delay chains. Compared to traditional transistor simulators ESPCV can provide both functional accuracy and simulation speed. This makes it possible simulate many more patterns and gain much higher confidence in the equivalence of the two models. For most design EPSCV can get close to 100% confident by running all patterns. By using the same Verilog test bench for the VCS gate level verification and the ESPCV simulations debugging failures is also simplified.

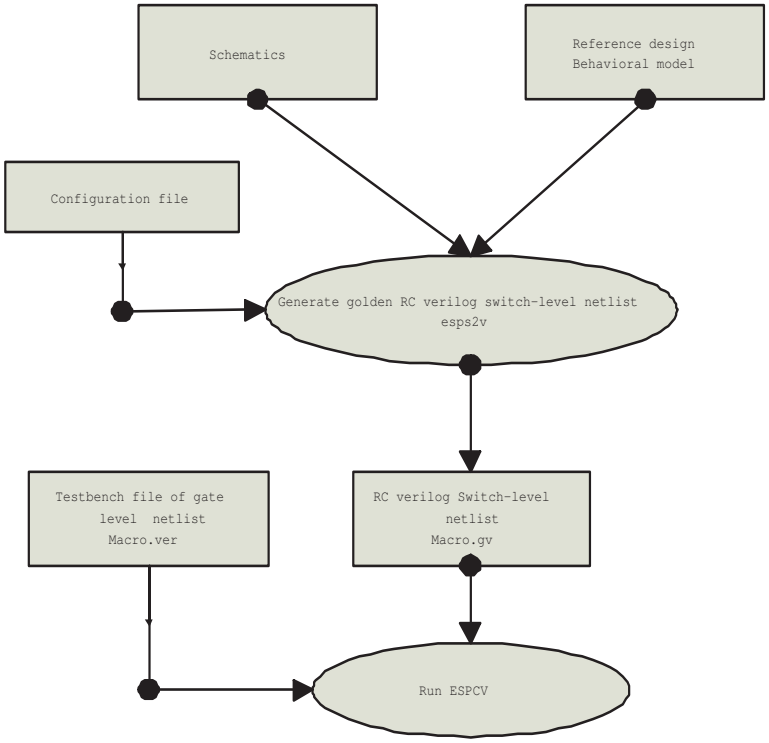


Figure 3.3: The Flow to Generate the Golden Model

Table 3.1: Simulation Time

Circuit	Number of Patterns	Transistor Level Simulator Simulation Time (mins)	RC verilog switch Level Simulator Simulation Time (mins)
Circuit A	7	1442	1.83
Circuit B	167	40324	271
Circuit C	411	Not testable	762

```
... Starting ESP simulation:
//                  0.00 ns : Begin test_setup
esp Info:   DC initialization complete.
//                 30.00 ns : Begin patterns, first pattern = 0
//                 30.00 ns : ...begin scan load for pattern 0
// *** ERROR during scan pattern 1 (detected during load of pattern 2)
1        87        6        (exp=0,        got=1)        //        pin IU_S_sout[64], scan cell 6, T=          6204.00 ns
1        87        10       (exp=1,        got=0)        //        pin IU_S_sout[64], scan cell 10, T=          6244.00 ns
```

Figure 3.4: Test Result

3.4 Experimental Result

This section provides details of experimental result on custom designed circuits taken from Qualcomm QDSP6. Table 3.1 shows the run time compared to transistor level simulation. Even for large design ESPCV can verify gate model with many patterns. It also can simulate with normal verilog file which has delay parameter. It is fast enough to verify the custom macro gate model. These were several simulation runs where the gate model was not identical to the expected values. Figure 3.4 is the snapshot of one of the test results. It shows the pattern number, expected output, output pin name and time of failure. From this result and internal dump values, users can easily find the location of incorrect model.

3.5 Conclusion

Gate level model generation for test has traditionally been a complex, manual process subject to the test engineer's skill and the designer's intelligence. This chapter describes a validation method between gate level models and schematics for custom macro designs. The gate level model does not contain delay information. Often this results in patterns mismatches. The proposed framework undergoes a three-step validation process to ensure its correct functionality. The flow accepts a SPICE netlist from schematics to generate golden RC Verilog switched level netlists. The test patterns generated using the ATPG tools are simulated with the delay aware netlists. With simple set up environment the flow effectively verifies the scan equivalency between these two models, gate level netlist and schematic.

Chapter 4

An Efficient Scan Chain Partitioning Scheme with Reduction of Test Data under Routing Constraint

The drawback of adding a delay test is that the test pattern volume required to keep quality high is expanding many-fold [5]. This is compounded by the fact that the designs themselves keep getting larger and hence there is more circuitry to test. Over the last few years, a number of test compression solutions have been developed. It is important when evaluating compression techniques to consider the compression in both test time and in test data volume. Test time is the amount of time the test takes to run on the tester, while the test data volume refers to the amount of memory needed on those testers to hold the test pattern information. In this chapter, a proposed scan chain partitioning scheme considers reduction of test set and test time, and the optimal routing inside each partitioned scan chain. First, two compatible scan cells are searched in a input test set. One group of compatible scan cells is included in one partitioned scan chain, while the other group is in the other scan chain. In finding these compatible scan cells, the group-based approach is employed since it provides a more optimal routing solution among the compatible scan cells in each of these two scan chains. After these two scan

chains are filled with compatible scan cells, they will be able to share one of two compatible columns in the input test set only during the shift-in process. Therefore, one of two compatible columns can be omitted from the input test set and the scan operation. Results with ISCAS'89 benchmark circuits show that the proposed method could reduce test data volume by 25-33% compared with a normal multiple scan design.

4.1 Introduction

Test time increases dramatically since a larger test pattern is required to maintain the desired fault coverage as an integrated chip (IC) becomes more complicated. In order to reduce the test time, the multiple scan chain architecture is popularly adopted in industry. The multiple scan chain architecture partitions a long scan chain into multiple shorter scan chains so that each partitioned scan chain has the almost same length as each other to maximize reduction of test time. However, the multiple scan architecture demands a higher number of I/O pins and still has to deal with a very large test data set. An IC with a large test data set can only be tested with expensive test equipment having large pin memories to store the test data set. For this reason, reducing the test data volume is very important in reducing the cost of testing.

Several methods have been proposed for the above test volume issue in scan testing. Test compression schemes are typically used to reduce the test volume [16][17][18]. In these methods, the test data can be reduced by a

encoding method. An on-chip decompressor changes the encoded test data to the original test data. However, multiple scan chains require either a separate decompressor or a serialization circuit. Thus, decreases in test data volume are achieved through higher DFT area.

As other test volume reduction methods, K. Lee, *et al.* in [19] argued that the volume of the test set can be reduced by generating common test patterns applicable to all CUTs by using the compatibility of test patterns. I. Hamzaoglu, *et al.* [20] proposed the Illinois scan architecture, which has two scan modes, a serial scan mode and a broadcast scan mode. This may alleviate correlation problems. A further enhancement used in [21] is combining the Illinois scan architecture and the dynamic scan architecture for reducing test data. As an alternative approach, K. Miyase, *et al.* [22] [23] described a scan tree architecture, where some of the scan cells can take in the same test data when the trees are mutually compatible.

This chapter proposes a scan chain data reduction method using a scan chain reordering. Actually scan chain reordering solutions [24] [25] [26] [27] are optimization technique to minimize the routing area overhead of scan chains. This technique was combined with the window based grouping method to reduce congestion.

The technique for designing reducing data volume provides such advantages. It does not require additional DFT logic like the encoding method. It is applicable to any scan design. Initially, ATPG generates a test set and the proposed method works on a given test set. Fault coverage does not change

from the original test set.

The rest of the chapter is organized as follows, In Section 4.2 the preliminary definitions are introduced. Section 4.3 describes a proposed algorithm to determine compatible columns and assign these columns to scan chains. Section 4.4 presents experimental results on benchmark circuits. Finally Section 4.5 concludes the chapter.

4.2 Preliminaries

This section introduces some definitions that are frequently used in this chapter. The first step is to find as many compatible columns as possible from an input test set. A column is compatible with another column in an input test set when the bit value at each position in the column is the same as the bit value in the corresponding positions of the other column. Generally, there can be more than one column compatible with another column in the input test set.

> **Definition 1:** A ***compatible column*** of a column in the input test set is a column each bit of which in any given position is the same as the bit in the same position in the other column, where don't-care bits can be interpreted as either 0 or 1.

Figure 4.1 shows an example of columns that are and are not compatible with a particular column in an input test set. As already mentioned, there usually exist more than one compatible column with respect to a particular

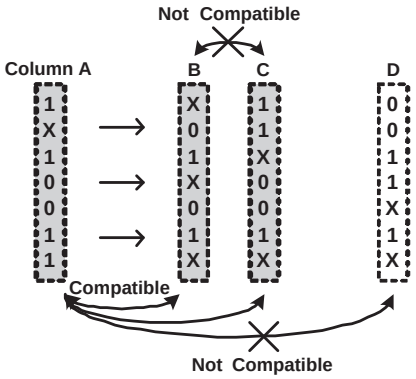


Figure 4.1: Compatible and Not Compatible Column of a Column in Input Test Set

column in the input test sets. In Figure 4.1, Columns B and C are compatible with Column A, because the bits in the rows of Column A are compatible with those in the corresponding rows of Column B or of Column C, where don't-cares are converted as 0 or 1. Column D, however, is not compatible to Column A, because the first and fourth rows have different bits. Note that even though Columns B and C are compatible with Column A, they are not necessarily compatible with each other. In this case, Columns B and C have different bits in their second rows.

> **Definition 2:** A ***compatible scan cell*** of a particular scan cell is defined as a scan cell whose corresponding column in an input test set is compatible with the column that is equivalent to the particular scan cell.

These two definitions will be used throughout the chapter. The next section discusses the algorithm for finding the compatible columns of each

column in the input test sets and then assigns them to it.

4.3 Scan Chain Partitioning Algorithm

The proposed algorithm starts with the physical information about the placement of the scan cells. The scan synthesis provides the optimal order of stitched scan cells to create a short length of scan routing with the aim of having less congestion. The proposed algorithm uses only the placement information of the scan cells in partitioning a scan chain into multiple ones, and then the optimal re-routing process is performed within each partitioned scan chain.

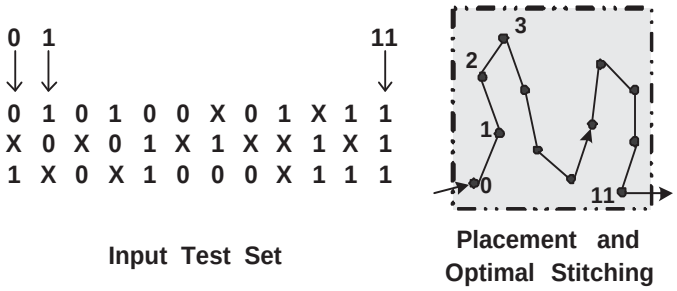


Figure 4.2: An Input Test Set and its Placement and its Optimal Scan Stitching

Given the optimal order of the scan cells in Figure 4.2, their optimal order is ignored, and only their placement information is kept and employed. Based on the physical information about the scan cells, the division is performed to separate the scan cells into two groups. The scan cells in one group come from one half of the placements, whereas those in the other group come

from the other half. Figure 4.3 shows only a vertical division; however, it could have been divided into two in a different way, for example, horizontally. The way that scan cells are divided into two groups is basically the user's decision, but the two groups must have almost the same number of scan cells. After the division, the scan reordering of a test set is conducted. First, the scan cells in one half of the placement are placed, and then those in the other half follow in the remaining order of the test set. During this reordering process, the scan

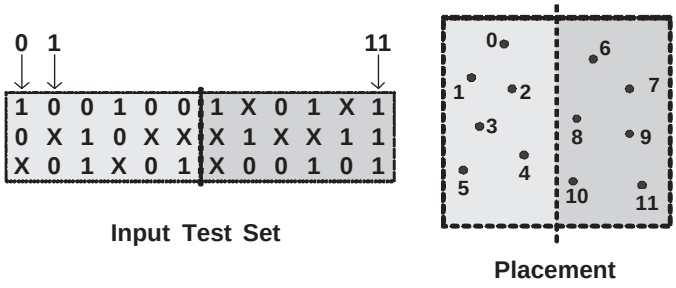


Figure 4.3: Reordering and Renumbering After the Division

cells that are physically close to each other in each half are placed together in the test set. In order to accomplish this systematically, the cluster-based approach and the near-neighbor approach are used to find the new order of scan cells in each half of the placement. Now, the columns in the first half of the input test set physically belong to one half of the placement. On the other hand, the columns in the second half of the input test set belong to the other half of the placement. Figure 4.3 illustrates this process.

With the newly reordered input test set, a list of compatible column(s) (scan cell(s)) of every scan cell in the first half of input test set is created. A

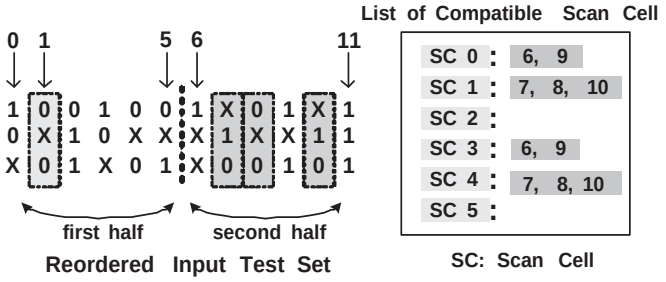


Figure 4.4: List of the Scan Cells Compatible with the Scan Cells in the First Half

column(s) (scan cell(s)) compatible to a column in the first half of input test set is (are) found by checking the column's compatibility to every column in the second half. This creates a list of columns compatible with the columns in the first half, as shown in Figure 4.4. For example, Column 1 in the first half has three compatible columns (Columns 7, 8, and 10, in the second half, as shaded in Figure 4.4), because all rows of Column 1 and those of the compatible columns are compatible in value. After the list of compatible columns is generated, the next move is to select one eligible compatible column among the many compatible columns of each column in the first half of input test sets. The chosen compatible columns will surely give the optimal routing solution because the selection of the compatible scan cells in the second half is performed while considering the physical proximity among the scan cells in the second half. For instance, when the compatible scan cell for scan cell 0 in the first half shown in Figure 4.4 is found, scan cell 6 is assumed to be picked as its compatible scan cell. Next, if the compatible scan cell for scan cell 1 in the first half is found, scan cell 7 would be a good candidate because it has

a smaller distance to scan cell 6 than any other scan cells in the second half. The scan cells 6 and 7 are consecutively numbered because of their closeness. This selection and connection obviously provides a shorter length.

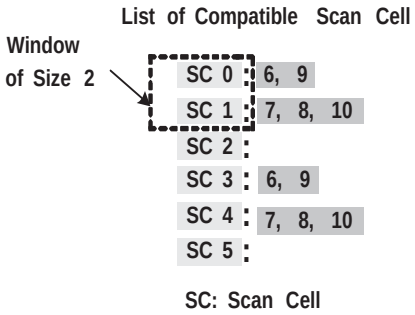


Figure 4.5: The Definition of a Window

To accommodate this selection process, the term window is introduced in Figure 4.5. A window is a group of columns in the first half of the input test sets that is selected and tested to determine its corresponding compatible columns in the second half. With the input test set in Figure 4.4, the size of the window is initially set as two scan cells, which means the two columns in the first half are to be selected and tested together. A window size of 1 means that selection and testing are no longer performed on a group basis. Thus, the size of the window must be equal to or greater than 2. The window procedure is applied to the selection process of the compatible scan cells. The window for selecting the compatible columns should be the same size as the window in the first half. Figure 4.5 depicts the concept of the window.

The size of the window depends basically on the user's decision. A reasonable size of the window can be found by a thorough investigation of the

list of compatible scan cells and the computational complexity. In addition, it is not difficult to determine a reasonable size for the window. If the selected size of the window fails to find a sufficient number of compatible scan cells, reducing the size will be an alternative solution, because a reduced size is more likely to find a sufficient number of compatible scan cells. Before the details of the procedure for choosing a group of compatible scan cells are explained, note that the scan architecture that will have a group of two compatible scan cells is two scan chains with a shared scan-in, as shown in Figure 4.6. If a three-scan-chain architecture is adopted with an input test set and a list of compatible scan cells, two of the scan chains share the same scan-in input, because their input test data has the same compatibility. Basically, the number of scan-ins in a multiple-scan-chain architecture is determined by the available number of input pins and other factors. The current study, however, assumes that there is no restriction on the available number of input pins. Figure 4.6(a) depicts one long scan chain with one scan-in and one scan-out, and Figure 4.6(b) shows a proposed scan architecture that has three scan chains with two scan-ins. Generally, in partitioning a scan chain, the difference in the number of scan cells in each scan chain should be minimized to make test time shorter and to make the control circuit simpler.

Now, the detailed procedure for selecting a group of chosen compatible columns is explained step by step with the help of the following example.

Step 1 (Figure 4.7(a)): A window of size 2 is applied to the first two scan cells, that is, scan cells 0 and 1 in the first half of the input test set. The

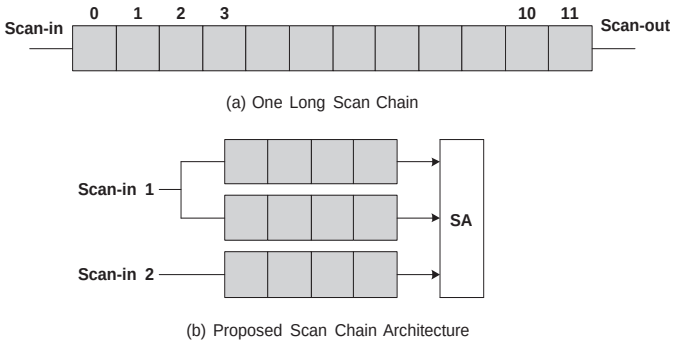
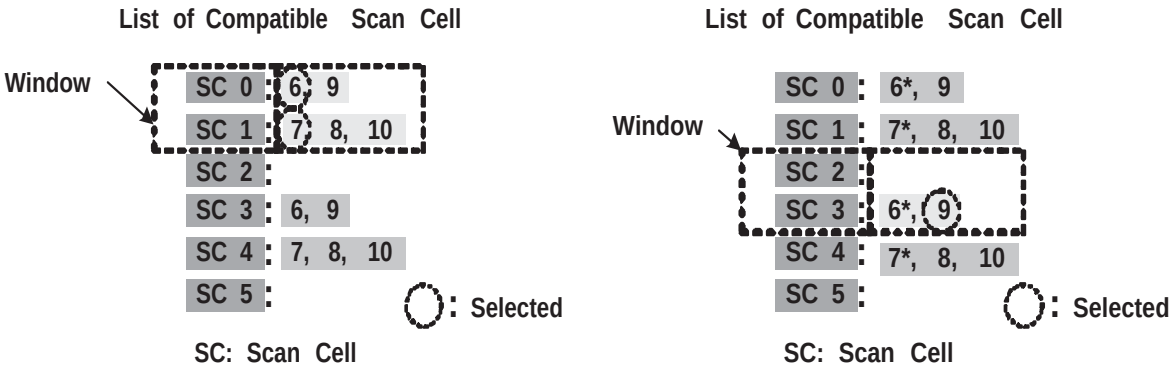


Figure 4.6: Proposed Scan Chain Architecture



(a) Step 1 (b) Step 2

Figure 4.7: Process of Selecting Compatible Scan Cells

goal in this step is to find two compatible scan cells in the list of compatible scan cells for scan cells 0 and 1. First, set scan cell 6 in the list of compatible scan cells as the compatible scan cell for scan cell 0 in the window. Next, scan cell 7 or 8 in the list of compatible cells is a candidate as a compatible scan cell for Scan Cell 1 in the window. A scan cell 7 is a better choice than scan cell 8 because the former is physically closer to scan cell 6. While a group of two compatible scan cells for two scan cells in the window are being found, no more processing is necessary in this stage. Now, the two scan cells in the window and their two corresponding compatible scan cells should not be selected in further processing because they have already been chosen for each other. Thus, they are marked with a "star" symbol, which means they are no longer available. When the window finds all compatible scan cells for every scan cell in the window, it shifts by the size of the window.

Step 2 (Figure 4.7(b)): The window moves to the next two available scan cells in the first half, that is, to scan cells 2 and 3. However, there are no compatible scan cells for scan cell 2 in the window, so Step 2 fails to select. If the window fails, it shifts down by 1 scan cell.

Step 3 (Figure 4.8): The window shifts by one scan cell if the window fails to find a compatible scan cell. Now, the window includes scan cells 3 and 4. Scan cell 9 in the list of compatible scan cells is selected for the scan cell 3 in the window because scan cell 6 was already selected in Step 1. The selecting step moves to the scan cell 4 in the window. Because scan cell 7 was already selected in Step 1, it is excluded for scan cell 4. Now, scan cells 8

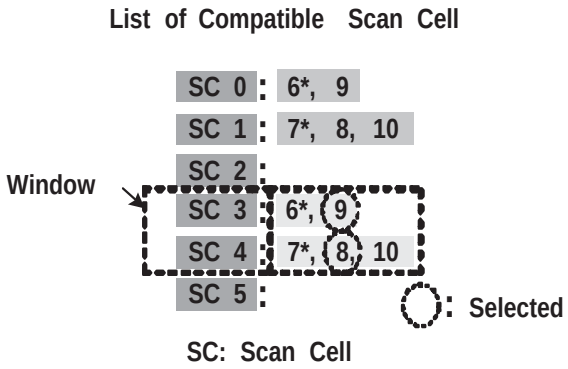


Figure 4.8: Step 3 - Process of selecting compatible scan cells

and 10 compete for the compatible scan cell. Scan cell 8 is picked here because it comes first and is closer to the two previously selected scan cells, that is, Scan cells 6 and 7. Scan cell 10, however, would also be a good pick. Because this step finds all compatible scan cells to fill the scan chain that shares the same scan-in with the other, no more processing is required. In this example, four compatible scan cells and their corresponding scan cells total eight, where eight is the number of scan cells in the two scan chains that share the same scan-in. Consequently, the process of selection stops here.

Figure 4.9 shows the last state of the list of compatible scan cells, where the selected compatible scan cells are marked with the star symbol. The four chosen scan cells and their partner scan cells in the first half will be assigned to two scan chains that have the same scan-in, whereas the rest of the scan cells will be assigned to the other scan chain, as shown in Figure 4.10.

Note that a scan cell in the window and its chosen compatible scan cell should go to the same position of two scan chains, because they must always

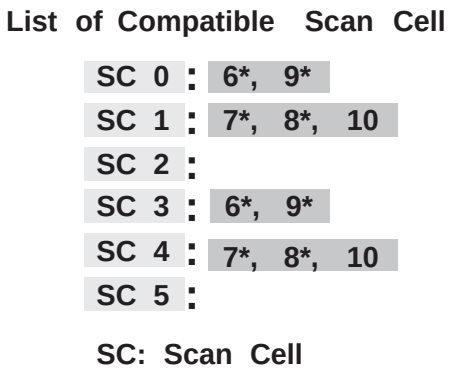


Figure 4.9: The Last state of the List of Compatible Scan Cells

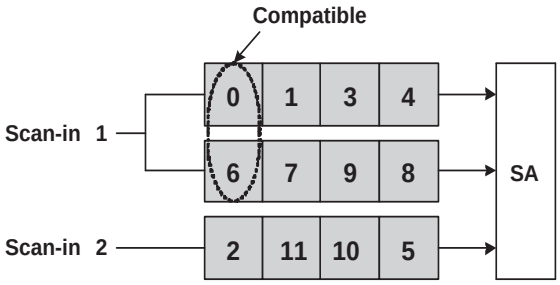
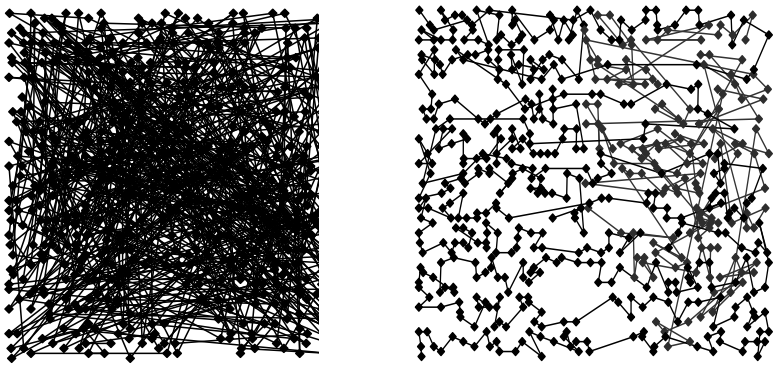


Figure 4.10: Optimal Assignment of Scan Cells to Three Scan Chains

have the compatible bit. For instance, if scan cell 0 goes to the first position of one of the two scan chains, its corresponding compatible scan cell 6 has to go to the first position of the other scan chain. Figure 4.10 depicts the optimal assignment of all scan cells to two scan chains and to the last scan chain. Based on this assignment, the routing among scan cells in each scan chain can be made. To observe the efficiency of the proposed scan chain partitioning scheme, it can be compared with the conventional three-scan-chain architecture in the next section. The conventional three-scan-chain architecture is assumed to assign its scan cells into scan chains based on physical proximity.



(a) (b)

Figure 4.11: (a) Routing of s15850 Obtained Without a Routing Constraint and (b) Routing of s13207 Obtained by the Scan Chain Partitioning Scheme with Three Scan Chains

4.4 Experimental Results

The proposed scan chain partitioning was performed with ISCAS'89 benchmark circuits with a 0.16um digital CMOS standard cell library used to produce the placement and scan routing. To compare the resultant scan length of the proposed partitioning scheme with that of Silicon Ensemble [28], Figure 4.11(a) depicts the scan length of the three-scan-chain scheme of ISCAS'89 s15850 by Silicon Ensemble [28]. For convenience, three scan chains are connected into one scan chain, because one global connection is basically similar to three individual connections. On the other hand, Figure 4.11(b) illustrates the length of the scan chain of the proposed scan chain partitioning scheme. The depiction shows the scan routing with few cross-overs and little congestion. The scan routing in Figure 4.11(b) provides a 33% reduction in test data. If this much cross-over and congestion are allowed in routing the

scan path, it is acceptable. If not allowed, compromises should be considered. One compromise is between the length or congestion of the scan routing and the reduction rate of the test data. Because most of the cross-over and congestion come from one of two scan chains sharing the same scan-in, the degree of congestion can be relieved by reducing the reduction rate of the test data. The approach to controlling the reduction rate is to increase or decrease the number of partitioned scan chains. Increasing the number of scan chains may reduce the degree of congestion because there is a possibility of avoiding the connection of any two scan cells that might create congestion.

Another compromise is between congestion and the size of the window. As the size of the window increases, it relieves the congestion of scan routing, because a large window size can provide shorter scan lengths between two scan cells without congestion than a small window size can. This approach can provide a short total scan routing; however, if the size of the window increases, it is highly likely to achieve a smaller reduction rate of test data, because it is difficult to find all the compatible scan cells that satisfy the scan cells in the window. This compromise should be taken into account also.

Table 4.1 shows the results of the proposed scan chain partitioning scheme in terms of the scan length and the reduction of test data. The third column indicates the scan routing that is optimally obtained by Silicon Ensemble. The fourth column is the number of partitioned scan chains that might give the best possible result for a benchmark circuit.

When the number of partitioned scan chains is three, two of the three

Table 4.1: Results of the Proposed Scan Chain Partitioning Scheme

Circuit	# scan flip-flops	Optimized wire length	# chains	Resulting wire length	Reduction rate of test data
s13207	669	1.15879e6	3	2.8650e6	33.3%
			4	2.3790e6	25 %
s15850	597	1.14044e6	3	2.9095e6	33.3%
			4	2.5147e6	25 %
s38417	1636	2.9967e6	4	5.9448e6	25%
s38584	1452	2.79421e6	4	8.43766e6	25%

scan chains share the same scan-in, and 33.3% of the columns in test sets are not needed. That reduction is indicated in the last column of Table 4.1. With the reduction, the length of scan routing is increased by the proposed scan partitioning scheme; however, the increase in the length of the scan routing is not significant in that it maintains the same order of magnitude. The one largest benchmark has an increase of one order of magnitude greater than that of Silicon Ensemble. That increase, however, ranges from threefold to fivefold, as with other benchmarks. That increase is not remarkable in routing the scan path. Thus the results here are considerable and significant.

As for computational complexity, the proposed scheme is not time-consuming. First, the identification of compatible scan cells for the scan cells in the first half of input test set requires a computational complexity of $O(n^2)$, where n is the number of scan cells. Because n is not large, the computational complexity is not large. In fact, actual complexity is less than $O(n^2)$, because only half of the scan cells find their compatible scan cells. The computational

complexity of picking one eligible compatible scan cell is $O(n^2 * m)$, where n is the number of scan cells and m is the number of iterations in seeking the number of partitioned scan chains. The number of scan cells is actually large compared with the number of iterations; therefore, the number of scan cells is the dominant factor in determining computational complexity. The number of scan cells, however, is small, and for that reason the total complexity in finding the compatible scan cells is not great. Other computational factors, like numbering the scan cells based on their physical proximity, employ the idea of the near-neighbor algorithm [25]. Their computation does need a little time to number scan cells when finding the compatible scan cells. Overall, however, computational complexity is small.

4.5 Conclusion

This chapter presented a method for finding compatible scan cells which are physically close to each other. With the compatible scan cells, designers can provide a reduction of test data and optimal routing solution. in addition, the reduction in pin counts and test data volume enables utilization of low cost testers. Experimental results for the ISCAS'89 benchmark circuits showed that this test method is indeed very simple and useful to achieve test data reductions.

Chapter 5

Automated Bit Mapping Generation Methodology for Custom Embedded Memory

One of the important steps in custom memory design for silicon validation is the generating the logical to physical bit mapping information. This bit mapping information is an essential factor to get zero defects and therefore high yield. Test engineers used this information to debug failures in the memory blocks on the tester. Historically generating the bit mapping information required the designer to manually figure out this mapping and either create a diagram by hand or write a custom script to describe the mapping. This manual process is error prone and hard to validate. This chapter will show a novel bit mapping flow using symbolic simulation. By simulating a memory symbolically, simulation time can be dramatically reduced. Also an automated flow is presented for generating the final mapping information.

5.1 Introduction

CMOS memory blocks are generally arranged in an array structure comprised of rows and columns. The data can be written and read from one (or more) row at a time. An address is supplied to select which row to read or

write to. The address is usually encoded and a row decoder is used to select the desired row.

Figure 5.1 shows a simple memory block comprised of three parts :

1. Memory Array: This is the actual memory storage component of the design containing m columns (bitlines) by n rows (wordlines) of memory cells.

2. Row Decoder: This is used to decode an x bit wide encoded address into an n bit wide decoded address which in turn is used to select the corresponding wordline to enable for read/write.

3. Data Path: This contains read and write circuitry for each bitline. Typically this would be a bit line driver for write operations and a sense amp/output driver for read operations.

For the simple memory case, the lookup table can be written as shown in Table 5.1. This maps data+address combinations to exact x,y memory locations. It is assumed that the origin of the array is at (0μ,0μ) and the width and height of each memory cell is 1μ .

In its simplest implementation the physical array structure can be made to match the logical intent. However often this is not the case. In some cases the logical aspect ratio of the memory can not be implemented physically because of electrical or physical limitations, in which case the array can be folded to accommodate a different physical aspect ratio. A tall and skinny

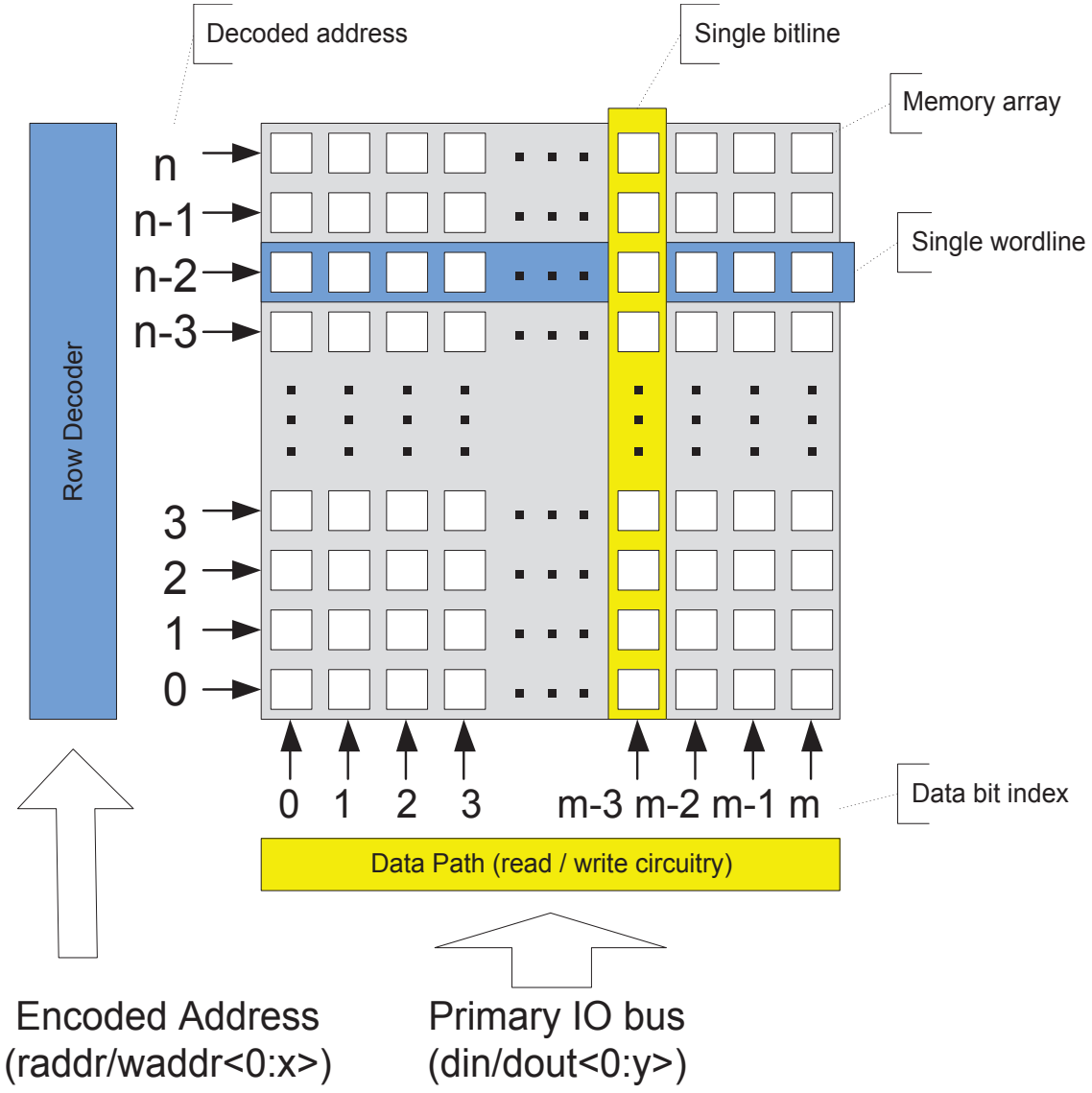


Figure 5.1: Simple Memory Array Structure

Table 5.1: Simple Bitmapping Table

Data	Address	X	Y
1000..	0000	0	0
0100..	0000	1	0
0010..	0000	2	0
.	.	.	.
.	.	.	.
1000..	0001	0	1
0100..	0001	1	1
0010..	0001	2	1

array may cause the read operation to take longer since the bitlines are too long. To fix this the array can be folded into a more desirable aspect ratio. For example a logical array of 8 entries with 4 bits in each entry can be built as a physical array of 4 rows with 8 bits in each row (see Figures 5.2 and 5.3). This is accomplished by multiplexing the data where a single physical row holds two logical entries. A column multiplexer is then used to select which half of the physical row the desired data is read or written to. One or more address bits (Addr$\langle 2 \rangle$ in this example) can be used to select between the different physical bitlines holding the logical column data.

Furthermore, in addition to column multiplexing, the data columns and rows can be interleaved, flipped or scrambled for various reasons. This does not pose any functional problems for the design since the data is always read and written to the same physical memory cell. However this poses additional complexity in generating the bit mapping information. In addition to the

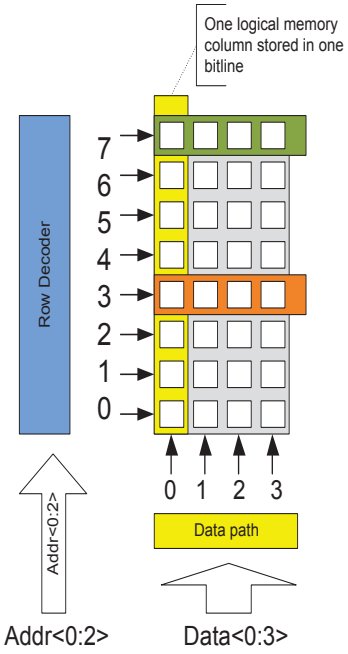


Figure 5.2: 8 entry x 4 bit Array Implemented Without Folding

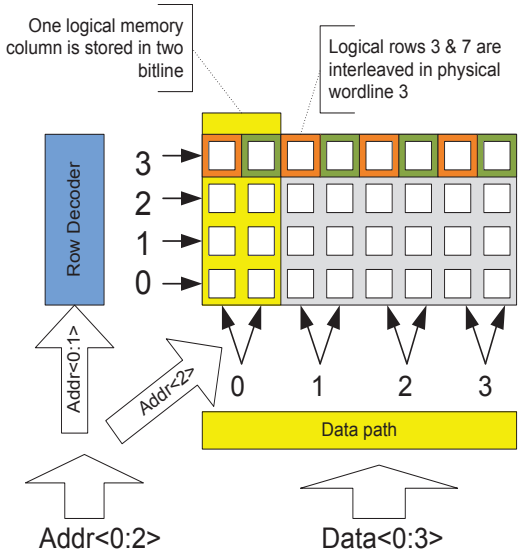


Figure 5.3: 8 entry x 4 bit Array Implemented with Folding

above complexities, the spacing between memory cells can vary because of banking and gaps for read or compare circuitry in the case of CAMs. This makes it harder to predict the exact coordinates of the memory cells with a simple script.

With the System on Chip (SoC) trend, there is an increasing demand to use embedded memories. Diagnosing these built-in memories has become more important to get high yield [29] [30] [31]. MFAT (Memory Failure Analysis Tool) is a combination effort with online ATE (Automated Test Equipment) code which logs memory failure information and offline scripts which map that failure information to physical bitcell coordinates. The failure information captured during ATE testing and then the offline processing script converts the captured logical BIST information to the corresponding physical memory instance and bitcell coordinates.

In order to perform the mapping, three sources of information are required:

1. The chip layout files to know the type, origin, orientation and size of each memory instance.

2. The mapping between the BIST partitioning (memory collars, address range and data bit range) and each memory instance.

3. The location and size of the bitcells corresponding to the macro address and data bits.

The layout files in #1 above are generated in the process of chip tape out. The mapping in #2 above is known for most memories during BIST insertion: the majority of arrays are compiled memories where a single memory instance is tested by a single BIST collar and there is 1:1 mapping between the BIST and memory address and data busses. For custom macros this mapping can be fairly complicated: due to performance considerations the BIST insertion might be done several hierarchal levels above the actual memory. This introduces the potential for address and data bus scrambling and results in cases of multiple BIST collars testing a single macro instance, a single BIST collar testing multiple macro instances, or a combination of the two where subsets of a memory's address range are tested by several BIST collars. The bitcell location in #3 is delivered as an "FA (Failure Analysis) file" for each memory type. The FA file is in the format of a lookup table with a single line per combination of memory address and data bit. The bit mapping information is available from this FA file.

5.1.1 Objective of This Work

The objective of this work is to create an automated bit mapping flow and reduce the simulation time. More specifically, the flow can handle custom macro designs. Historically generating the bit mapping document for custom macro design required the designer to do manual creation of bit mapping. This creates two layers of potential user error as shown below. It is desirable to prevent those errors:

5.1.1.1 First Order Errors

First order errors occur when the designer doesn't really know the actual mapping, this can happen because the circuit designer in charge of the memory block is different from the mask designer building the actual layout. The mask designer will often build the memory using hierarchical sub blocks that get flipped and mirrored several times through out the hierarchy. At the end, the design passes logical to physical checks and can be extracted and simulated, so the circuit designer is none the wiser.

5.1.1.2 Second Order Errors

In this case the designer does know the correct mapping pattern. There is possibility that the designer makes mistakes in the custom script created to generate the bit mapping file. Or in the case of the diagram, the details are not conveyed in a comprehensive enough way and the mapping information is misinterpreted by the test engineer. These errors are hard to detect by existing methods. The size of the bit mapping file (which can be hundreds of thousands of lines) makes it impossible for the designer to comprehensively debug it without a visual representation of the data.

5.2 Bit Mapping Flow Overview

This section presents overall flow of bit mapping that can automatically generate the bit mapping information directly from the physical memory layout and transistor level logic simulations. The flow uses several commercial tools

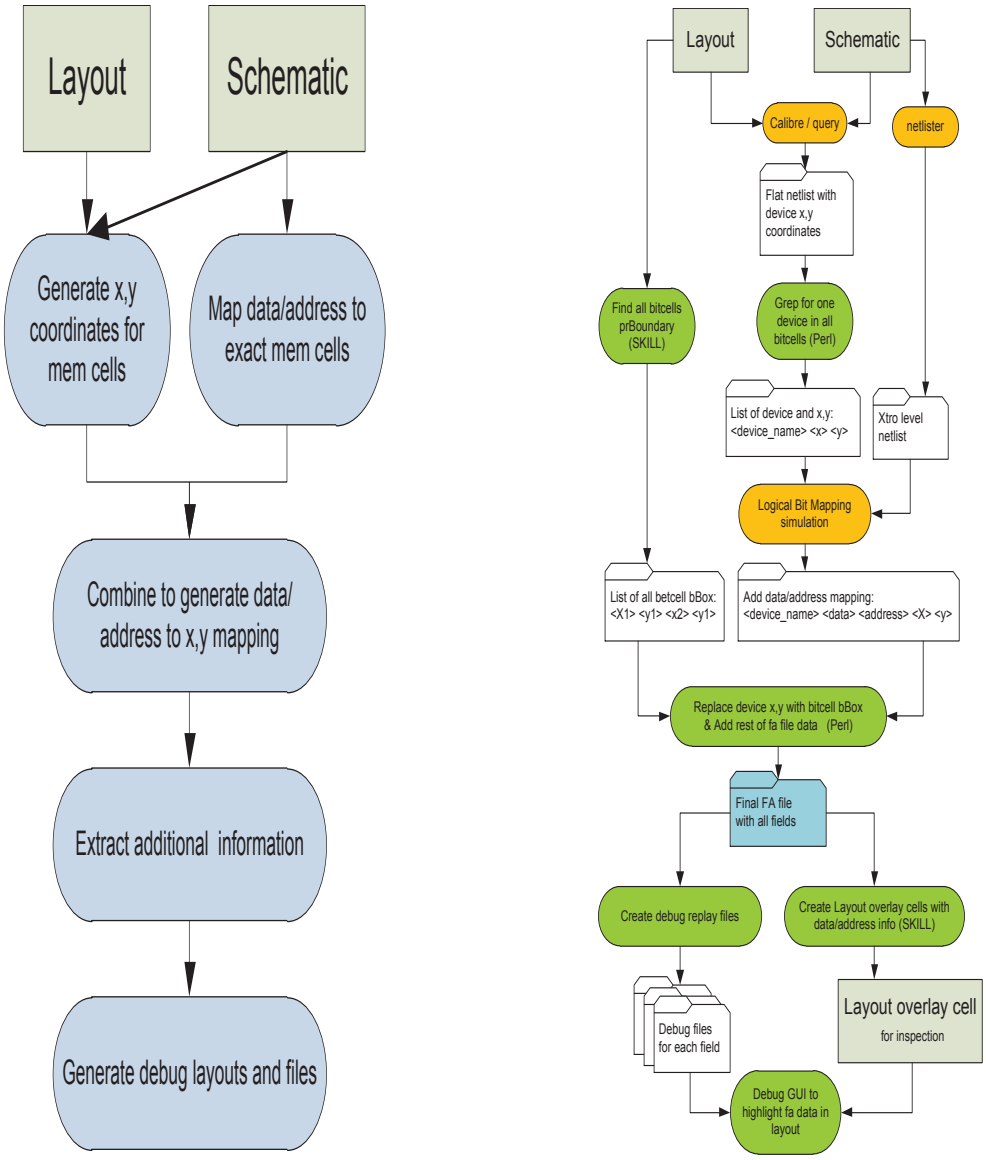


Figure 5.4: Bit Mapping Flow Overview (Left) Flow Details (Right)

with some custom code and user interface layers developed in-house. The flow can be broken down into the following areas (see Figure 5.4)

1. Physical coordination information - Generate a mapping between all memory cells in the design and their physical x,y coordinates

2. Logical Simulation for Bit Mapping - Simulate to exercise all data/address combinations for each memory

3. Combine the two mapping tables to generate data/address to x,y mapping

4. Extract additional design information like banking and indexing needed to generate the full FA format

5. Additional debug tools and GUI interface to visually debug the generated bit mapping

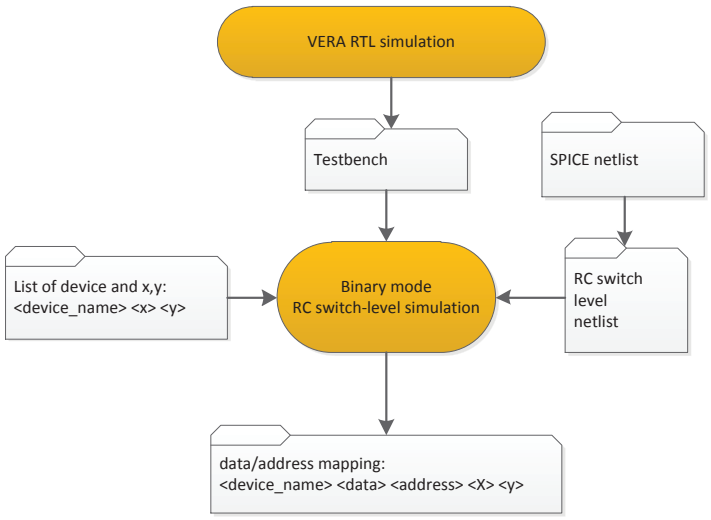
5.3 Physical Coordination Information

To find memory cells' x,y coordinates, the first step of the flow is to get a full list of the memory cells in the design. To accomplish this, Mentor Graphics' Calibre LVS (Layout vs. Schematic) [32] is used, which is normally used to validate the final layout against the designer created schematics. Calibre has a query utility that allows generating a flat transistor level netlist with x,y coordinates for each transistor in the design. Next the netlist is parsed and searched for a specific transistor naming pattern representing one of the transistors in the memory cell. The user has to specify this pattern since it is design specific. This gives a list of the occurrences of specific transistor in all of the memory cells in the design.

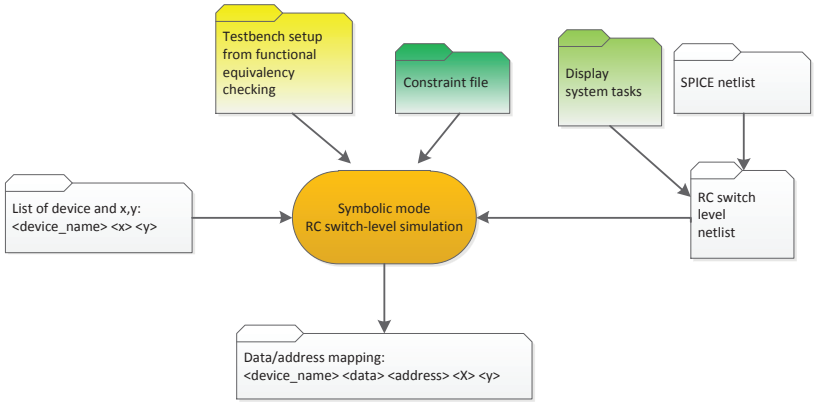
5.4 Logical Simulation for Bit Mapping

The relationship between a memory data+address combination and an actual bitcell location can be found by a simulation. The address and data space are swept though all ranges and each memory cell location in the netlist is probed to see where the data+address combination got written. an RC switch-level simulator is used for this purpose [15]. This simulator has been tailored to perform custom equivalence checking on transistor level netlists. It is designed to provide functional verification coverage, Verilog vs. SPICE netlist equivalency checking or Verilog switch-level design simulation, but this simulator is used for a different purpose to find out the memory cell location. It is effective due to its event-driven nature and efficiency of simplified switch-level model, The binary mode of this simulator provides a good mechanism to match between the physical location of memory cells and the logical cell information. Figure 5.5(a) shows the list of inputs to run this simulation. The RTL simulation with VERA [33] is used to generate a testbench for the RC switch-level simulator. The SPICE netlist is translated into a switch-level Verilog netlist using esps2v [15]. When the data is written to a specific memory cell, the log file includes that memory cell instance name along with the data/address and x,y information.

However, if such binary mode RC Verilog simulation is used there is one drawback: the simulation time can be too long for large memory blocks. For example, a 2KB x 16 sub array with 1 simulator license can take many days to finish the simulation. The run time can be reduced by partitioning



(a)



(b)

Figure 5.5: RC switch-level simulation flow (a) binary bit mapping (b) symbolic bit mapping

the testbench or matching the single sub array to different locations, but these options require a lot of manual work. To address this issue more efficiently the logical bitmap flow using symbolic simulation was developed. It uses fact

that when the simulator encounters a symbolic input, it propagates a Boolean expression. These symbolic expressions can be searched for when probing the memory nodes instead of looking for a simple 0 or 1 value. Using just 1 symbolic cycle, all memory cells are inspected which gives the same results as binary mode simulations in a fraction of the time. Figure 5.5(b) shows this flow. An initial simulation setup comes out from the functional equivalency checking and additional constraints were used to propagate only address and data symbolic input to memory cells. Based on the cell type of memory, display system tasks are added into RC switch level netlist to print out the hierarchical names of memory cell. The simulation run time in this case is a few minutes instead of hundreds of days. Table 5.2 shows the simulation time to compare with the binary bit mapping method.

5.5 Generating Memcell Box Data

So far there are single x,y coordinates in our mapping tables corresponding to the origin of a single transistor inside each memory cell. The

Table 5.2: Simulation Time of Binary and Symbolic Bit Mapping

Design Name	# of bitcells	Binary Bit Mapping	Symbolic Bit Mapping
A	2176	12 sec	0.47 sec
B	11264	60 sec	0.87 sec
C	262144	Not testable	28 mins

complete Bit Mapping information however requires two sets of coordinates: x1,y1 representing the lower left corner of the memory cell and x2,y2 representing the upper right corner of the cell. To get this information, a custom Cadence SKILL [34] procedure is used that loops through all the cells in the block hierarchy and locates all the memory cells. It then looks for a prBoundary shape in the layout which represents the extents of the cell. The result of this step is a file containing a list of all x1,y1,x2,y2 bounding box coordinates of all the cells in the design translated to top level memory coordinates. This file will be parsed later in the flow and each x,y coordinate of the original mapping will be matched with the corresponding x1,y1,x2,y2 coordinate from this file.

5.6 Generating the Final FA File

Along with the single table which contains the x1,y1,x2,y2 coordinates and data + address combinations from the previous step, additional information is required. The values of those fields are extracted automatically as follows:

1. Row & Column: After sorting the data by x,y coordinates the Row & Column information is organized by the order of the data. Cells with matching y and increasing x belong to a single row. Similarly cells with matching x and increasing y belong to a single column. The rows and columns are numbered by increasing order from the lower left corner of the design.

2. Bank: Looking at the x,y sorted data cells with touching boundaries are expected to belong to the same bank. When cells where x2 of the first cells does not equal x1 of the second cells, there is a horizontal gap between them. Similarly vertical gaps are found by looking at the y2 and y1 coordinates of neighboring cells. Since not all gaps correspond to different banks (some gaps correspond to read or compare circuits within the same bank) the designer can specify a gap threshold to filter those out.

3. Index: This corresponds to the portion of the address used for column muxing. The rest of the address bits are used for row decoding. Row decoding and column muxing is distinguished by the direction the data moves in. For example, if the address space is swept and certain bits of the address correspond to cells arranged vertically then these bits are used for row decoding. if other bits correspond to cells arranged horizontally then they are used for column muxing.

4. Wordline: Now that index and row decode portions of the address space are identified, the row decode bits can be swept to figure out the corresponding memory rows exercised by them. These rows are then numbered in increasing order matching the row decode address they correspond to.

5. Bitline: Similarly to the wordlines, the bitlines are determined by sweeping through the index portion of the address space in combination with

the data bus bits.

5.7 Debugging the FA File

Given the massive amount of data in a FA file (some blocks have 256,000 lines with 9 data fields in each line) it's humanly impossible for a designer to manually inspect the data in the file. To get around this, a couple of visualization tools have been developed:

5.7.1 Layout Overlay Cells

Using a custom SKILL procedure, the data is read from the FA file to generate static overlay cells that can be placed on top of the real layout. These overlay cells have informational shapes and labels that mark the boundary of each cell in the FA file and annotate the values of the different fields. This allows the designer to visually inspect the data on top of the real layout (see Figure 5.6(a))

5.7.2 Interactive Debug GUI

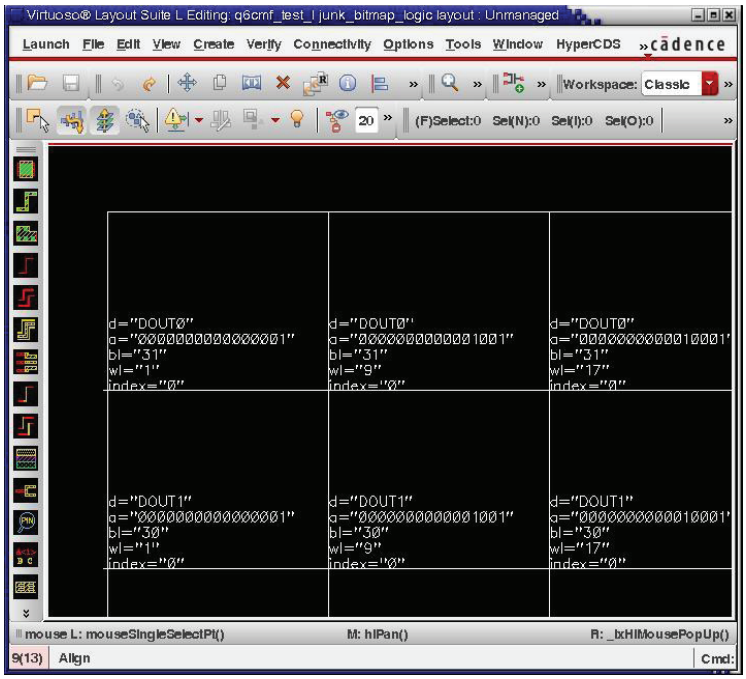
The layout overlay cells are great for making sure the x,y coordinates in the FA file fit perfectly on top of the real layout cells. They are also helpful in debugging smaller designs, but the static nature of the overlay cell makes it hard to debug large designs. To address this issue, a Debug GUI (Figure 5.6(b)) was created that allows the designer to sweep through any combination of the FA fields and have the corresponding cells highlighted in

the layout. There is also a replay feature that allows the designer to create files with specific data and or address sweeps and have the tool sweep through them and play an animation of the corresponding cells in the layout.

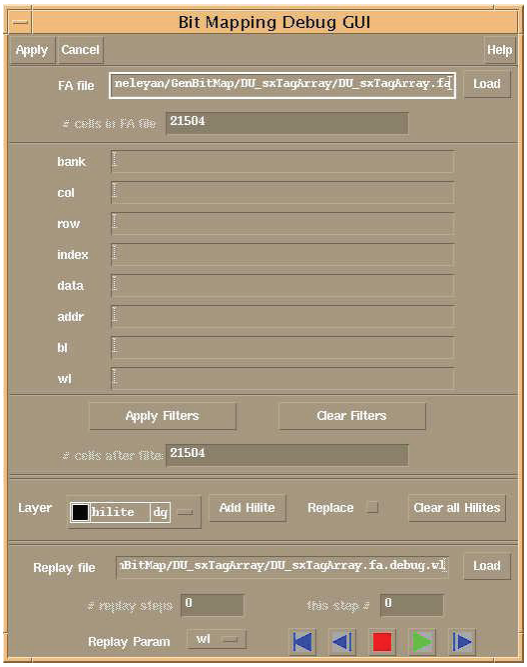
Both the Layout overlay cells and the debug GUI are not restricted to debugging FA data generated by this flow. Even if a designer has an FA file generated by other means, these debug tools can be used to validate the accuracy of their data.

5.8 Conclusion

The automatic flow for bit map generation described in this dissertation has been successfully used in several Qualcomm DSP(Digital Signal Processor) core design projects across several technologies (65nm, 45nm and 28nm). It has eliminated much of the inherent user error in the FA files delivered to the silicon validation team and improved confidence in the testing results. To deliver accurate information within a limited time through design tape-out, the automated flow helped quickly to generate bit mapping information for complicated custom macro design. Using the symbolic simulator dramatically reduced simulation time and the developed GUI debug tool gave high confidence that the data is correct.



(a)



(b)

Figure 5.6: (a) Layout Overlay for Debug (b) MFAT Debug GUI

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This dissertation addresses four testability considerations for design with embedded memory namely,

Test coverage for memory interface logic

Gate model verification with delay aware netlist

Delay testing cost in terms of test time and data volume

Custom embedded memory bit mapping.

- The memory interface logic has been less of a consideration, but with the demands of delay testing, the paths through memories are critical ones and it is required that they be covered by scan based testing to reduce cost. The issue of X generation from memories was addressed. To resolve this issue, register based testing with X prevention logic and an X prevention ATPG method were proposed. These methods showed increased test coverage without an increase in the number of patterns.

- Gate model generation for test is an important task to do with scan-based testing without any mismatches. There have been many challenges to

do this job correctly. The methodology developed in the dissertation accepts a SPICE netlist from a schematic to generate a golden RC verilog switched level netlist. The test patterns generated using the ATPG are simulated with a delay aware netlist. Because this delay aware netlist comes from the schematic after the layout designer has finished the placement, it represents the exact delay from the design point of view. Using the new methodology, the designer can also improve the design to increase the test coverage and avoid mismatches.

- The cost of delay test has been increased due to volume demand for additional delay patterns which can require testing for more than stuck-at fault patterns. The third method proposed a way for finding compatible scan cells which are physically close to each other. Using this compatible scan, test data, and test time can be reduced with optimal routing.

- A custom embedded memory often comes with address scrambling which causes errors in the bit mapping information. Generating this mapping table is an essential stage to find memory faults during post-silicon debugging. The suggested methodology was the first time symbolic bit mapping was used which saves simulation time. A newly developed automated flow were presented.

6.2 Future Work

For the future work, the goal is to establish a complete flow to do delay testing which includes path delay fault testing.

Path delay fault testing is becoming a very important test method to detect small delay defects, but without proper handing of embedded memories scan-based at-speed testing often ignores critical paths in the design. Therefore, the current work is an important step to do correct things for path delay testing.

Bibliography

[1] E. J. Marinissen, B. Prince, D. Keltel-Schulz, and Y. Zorian, "Challenges in Embedded Memory Design and Test," in *Proc. of Design, Automation and Test in Europe*, Mar 2005, pp. 722 – 727, Vol. 2.

[2] *Synopsys Inc.*, "TetraMAX," Version E-2010.12-SP2, 2010.

[3] J. Saxena, K. M. Butler, J. Gatt, R. Raghuraman, S. P. Kumar, D. J. Campbell S. Basu, and J. Berech, "Scan-Based Transition Testing - Implementation and Low Cost Test Challenges," in *Proc. of International Test Conference*, Oct. 2002, pp. 1120–1129.

[4] M. P. Kusko, B. J. Robbins, T. J. Koprowski, and W. V. Huott, "99% AC Test Coverage Using Only LBIST on the GHz IBM S/390 zseries 900 Microprocessor," in *Proc. of International Test Conference*, Oct. 2001, pp. 586–592.

[5] P. Gillis, K. McCauley, F. Woytowich, and A. Ferko, "Low Overhead Delay Testing of ASICs," in *Proc. of International Test Conference*, 2004, pp. 534–542.

[6] P. Wohl and J. Waicukauski, "Using Verilog Simulation Libraries for ATPG," in *Proc. of International Test Conference*, 1999, pp. 1011–1020.

[7] S. Yadavalli and S. Sengupta, "Impact and Cost of Modeling Memories for ATPG for Partial Scan Designs," in *Proc. of International Test Conference*, 1997, pp. 274–278.

[8] N. A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design Test of Computers*, vol. 23, no. 4, pp. 294 –303, Apr 2006.

[9] S. Mitra, S.S. Lumetta, and M. Mitzenmacher, "X-tolerant Signature Analysis," in *Proc. of International Test Conference*, Oct. 2004, pp. 432 – 441.

[10] E. K. Vida-Torku and G. Joos, "Designing for Scan Test of High Performance Embedded Memories," in *Proc. of International Test Conference*, Oct 1998, pp. 101 –108.

[11] G. Seok, B. Mohammad, H. Kim, and P. Bassett, "Verification of gate level model for custom design in scan mode," in *Microprocessor Test and Verification*, 2007.

[12] S. Kundu, "Gate Maker: A Transistor to Gate Level Model Extractor for Simulation, Automatic Test Pattern Generation and Verification," in *Proc. of International Test Conference*, 1998, pp. 372–381.

[13] K. Zarrineh, T. A. Ziaja, and A. Majumdar, "Automatic Generation and Validation of Memory Test models for High Performance Microprocessors," in *International Conference on Computer Design: VLSI in Computers & Processors*, 2001, pp. 526–529.

[14] *Cadence Inc.*, "Verple," Version 6.1, 2006.

[15] *Synopsys Inc.*, "ESPCV," Version 2010.06-SP1, 2010.

[16] B. Koneman, "LFSR-Coded Test Patterns for Scan. Designs," in *Proc. of European Test Conf*, 1993, pp. 237–242.

[17] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan Vector Compression/Decompression Using Statistical Coding," in *Proc. of VLSI Test Symposium*, 1999, pp. 114–120.

[18] A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data Compression," in *Proc. of VLSI Test Symposium*, 2001, pp. 42–47.

[19] K. J. Lee, J. J. Chen, and C. H. Huang, "Using a Single Input to Support Multiple Scan Chains," in *Proc. of International Conference on Computer Aided Design*, 1998, pp. 74–78.

[20] I. Hamzaoglu and J. H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," in *Proc. of International Symposium on Fault Tolerant Computing*, 1999, pp. 260–267.

[21] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T.W. Williams, "A Reconfigurable Shared Scan-in Architecture," in *Proc. of VLSI Test Symposium*, 2003, pp. 9–14.

[22] K. Miyase and S. Kajihara, "Optimal Scan Tree Construction with Test Vector Modification for Test Compression," in *Proc. of Asian Test Symposium*, 2003, pp. 136–141.

[23] K. Miyase, S. Kajihara, and S. M. Reddy, "Multiple Scan Tree Design with Test Vector Modification," in *Proc. of Asian Test Symposium*, 2004, pp. 76–81.

[24] M. Hirech, J. Beausang, and Xinli Gu, "A New Approach to Scan Chain Reordering Using Physical Design Information," in *Proc. of International Test Conference*, 1998, pp. 348–355.

[25] S. Makar, "A Layout-Based Approach for Ordering Scan Chain Flip-Flops," in *Proc. of International Test Conference*, 1998, pp. 341–347.

[26] D. Berthelot, S. Chaudhuri, and H. Savoj, "An Efficient Linear Time Algorithm for Scan Chain Optimization and Repartitioning," in *Proc. of International Test Conference*, 2002, pp. 781–787.

[27] L. Guiller, F. Neuveux, S. Duggirala, R. Chandramouli, and R. Kapur, "Integrating DFT in the Physical Synthesis Flow," in *Proc. of International Test Conference*, 2002, pp. 788–795.

[28] *Cadence Inc.*, "Silicon Ensemble," Version 4.5, 2005.

[29] C. H. Stapper, F.M. Armstrong, and K. Saji, "Integrated Circuit Yield Statistics," *Proceedings of the IEEE*, vol. 71, no. 4, pp. 453 – 470, Apr 1983.

[30] D. Y. Lepejian, J. M. Caywood, A. Kablanian, F. J. Ferguson, and A. Jee, "An Automated Failure Analysis (AFA) Methodology for Repeated Structures," in *Proc. of VLSI Test Symposium*, Apr 1994, pp. 319 –324.

[31] J. D. Segal, T. Ho, B. Hodgkins, P. Misic, J. Lin, and M. Yegnashankaran, "Predicting failing bitmap signatures for memory arrays with critical area analysis," in *Advanced Semiconductor Manufacturing Conference and Workshop*, 1999, pp. 178–182.

[32] *Mentor Inc.*, "ESPCV," Version 2010.03, 2010.

[33] *Synopsys Inc.*, "VERA User Guide," Version X-2005.12-7, 2005.

[34] T. J. Barnes, "SKILL: a CAD System Extension Language," in *Proc. of Design Automation Conference*, Jun 1990, pp. 266 –271.

[35] A. K. Majhi, G. Gronthoud, C. Hora, M. Lousberg, P. Valer, and S. Eichenberger, "Improving Diagnostic Resolution of Delay Faults using Path Delay Fault Model," in *Proc. of VLSI Test Symposium*, 2003, pp. 345–350.

[36] M. A. Breuer and S. K Gupta, "New Validation and Test Problems for High Performance Deep Sub-Micron VLSI Circuits," in *Proc. of VLSI Test Symposium*, 2000, p. 8.

[37] V. R. Devanathan, A. Hales, S. Kale, and D. Sonkar, "Towards Effective and Compression-Friendly Test of Memory Interface Logic," in *Proc. of International Test Conference*, Nov. 2010, pp. 1 –10.

[38] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations," in *Proc. of Custom Intergranted Circuit Conference*, May 2001, pp. 223–228.

[39] Z. Barzilai and B.K. Rosen, "Comparison of AC Self-Testing Procedures," in *Proc. of International Test Conference*, 1983, pp. 89–94.

[40] J. L. Carter, V.S. Iyengar, and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults," in *Proc. of International Test Conference*, Sept. 1987, pp. 418–427.

[41] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. of International Test Conference*, Oct. 1985, pp. 342–349.

[42] K. Heragu, V. D. Agrawal, and M. L. Bushnell, "Statistical Methods for Delay Fault Coverage Analysis," in *VLSI Design Conference*, Jan. 1995, pp. 166–170.

[43] J. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition Fault Simulation," in *IEEE Design & Test of Computers*, Apr. 1987, pp. 32–38.

[44] K. T. Cheng, S. Devadas, and K. Keutzer, "Delay-Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 8, pp. 1217 –1231, Aug 1993.

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

[46] K. T. Cheng, "Test Generation for Delay Faults in Non-Scan and Partial Scan Sequential Circuits," in *Proc. of International Conference on Computer-Aided Design*, Nov. 1992, pp. 554–559.

[47] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," in *Proc. of Asian Test Symposium*, Nov. 2001, pp. 325–330.

[48] S. Mitra and K. S. Kim, "X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction," in *Proc. of International Test Conference*, Oct. 2002, pp. 311–320.

[49] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K. H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide, and J. Qian, "Embedded Deterministic Test for Low Cost Manufacturing Test," in *Proc. of International Test Conference*, Oct. 2002, pp. 301–310.

[50] W. Qiu, X. Lu, J. Wang, Z. Li, D. M. H. Walker, and W. Shi, "A Statistical Fault Coverage Metric for Realistic Path Delay Faults," in *Proc. of VLSI Test Symposium*, April. 2004, pp. 37–42.

[51] A. Devgan and C. Kashyap, "Block-Based Static Timing Analysis with Uncertainty," in *Proc. of International Conference on Computer-Aided*

Design, Nov. 2003, pp. 607–614.

[52] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test Application Time and Volume Compression Through Seed Overlapping," in *in Proc. of Design Automation Conference*, 2003, pp. 732–737.

[53] A. R. Pandey and J. H. Patel, "Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs ," in *Proc. of VLSI Test Symposium*, 2002, pp. 9–15.

[54] I. Hamzaoglu and J. H. Patel, "Compact Two-Pattern Test Set Generation for Combinational and Full Scan Circuits," in *Proc. of International Test Conference*, 1998, pp. 944–953.

[55] A. Würtenberger, C. S Tautermann, and S. Hellebrand, "A Hybrid Coding Strategy For Optimized Test Data Compression," in *Proc. of International Test Conference*, 2003, pp. 451–459.

[56] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," in *Proc. of Design Automation Conference*, 1993, pp. 102–106.

[57] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," in *Proc. of International Conference on Computer-Aided Design*, 1998, pp. 283–289.

[58] J. D. Segal, T. Ho, B. Hodgkins, P. Misic, J. Lin, and M. Yegnashankaran, "Predicting Failing Bitmap Signatures for Memory Arrays with Critical Area Analysis," in *Advanced Semiconductor Manufacturing Conference and Workshop*, 1999, pp. 178–182.

[59] Y. Kukimoto, M. Fujita, and M. Tanaka, "Symbolic Verification of CMOS Synchronous Circuits Using Characteristic Functions," in *Proc. of Custom Integrated Circuits Conference*, May 1991, pp. 11.5/1 –11.5/4.

[60] M. Merino, C. Mateos, and K. Terryll, "Achieving Good Correlation Results between Bitmap and TENCOR Data," in *Proc. of Advanced Semiconductor Manufacturing Conference*, 2001, pp. 131 –134.

Index

Vita

Geewhun Seok was born in Suwon, Korea on September 7, 1974, the son of Hobong Seok, and Jongrae Lee. He graduated from Daesung High School, Daejon, Korea in 1993 and received the B.S. degree in electronic engineering from Korea University, Seoul, Korea in 2002, M.S. degrees in electrical engineering and computer science from KAIST, Daejon, Korea in 2003, and is currently pursuing his Ph.D. degree in electrical and computer engineering from the University of Texas at Austin.

He is currently with Qualcomm DSP core design team in Austin, TX. His research interests are in the areas of testing and design automation with emphasis on design for testability, delay fault testing, built-in self test.

Permanent address: 16803 Willow Oak Ln Round Rock, TX 78681

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.