MODELING AND SIMULATION OF FINITE STATE MACHINE MEMORY BUILT-IN SELF TEST ARCHITECTURE FOR EMBEDDED MEMORIES

Nor Zaidi Haron [1], Siti Aisah Mat Junos @Yunus [1], Abdul Hadi Abdul Razak [2]

[1]Faculty of Electronics and Computer Engineering, Universiti Teknikal Malaysia Melaka (UTeM)

[2]Faculty of Electrical Engineering, Universiti Teknologi Mara

zaidi@utem.edu.my, aisah@utem.edu.my, adi3443@salam.uitm.edu.my

Abstract

Memory Built-in Self Test (MBIST) or as some refer to it array as built-in self-test is an amazing piece of logic. Without any (direct) connection to the outside world, a very complex embedded memory can be tested efficiently, easily and at lower cost. Modeling and simulation of Finite State Machine (FSM) MBIST is presented in this paper. The design architecture is written using Very High Speed Integrated Circuit Hardware Description Language (VHDL) code using Xilinx ISE tools. The architecture is modeled and synthesized using register transfer level (RTL) abstraction. Verification of this architecture is carried out by testing stuck-at-faults SRAM. Two BIST algorithms are implemented, i.e., MATS and March C- to test the faulty SRAM.

***Keywords**: Memories, Built-in Self Test, Finite State Machine, Very High Speed Integrated Circuit Design Language.*

I. INTRODUCTION

There are several Designs for Testability (DfT) techniques for embedded memories. Each of the methods has advantages and disadvantages. As embedded memories are becoming dense, their controllability and observability are difficult to predict. This is where built-in self test (BIST) technique plays its part. BIST technique integrates the functionality of an automatic test system onto the same die as embedded memories. Therefore, test pattern generation and response can be performed automatically.

The generic memory BIST architecture is illustrated in Figure 1. The architecture consists of three main blocks: controller, pattern generator and signature analysis register. The controller is used to govern the overall sequence of events. For example, if the address counter should be counting up or down or if the data is being generated should be a marching 0 or marching 1 pattern.
The test pattern generation block contains the circuitry to produce the address, data, and control values necessary to create each test stimulus that is to be applied to the memory. This block typically contains an up/down counter for generating the address sequences needed by most memory test algorithms.

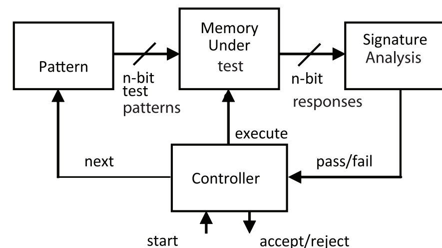

Figure 1: Generic MBIST architecture

The signature analysis register compares the values read out of the memory with

expected values generated by the signal generation block. The result of each comparison is accumulated into a status flip-flop in order to provide an accept/reject result at the end of the test.

In this paper, a finite state machine-based architecture controller is presented. Previous papers [3]-[6] have discussed this architecture but using either assembly or C language. These languages cannot be implemented in hardware such as Field Programmable Logic Array (FPGA) or Application Specific Integrated Circuit (ASIC). A hardware description language (HDL)-based MBIST would be interesting for educational purposes. Many universities are migrating to HDL such as VHDL and Verilog in teaching electronics subjects such as embedded systems and digital communication. Furthermore, designs written using HDLs can be implemented into FPGA or ASIC for further investigation.

The rest of the paper is organized as follows. Section II describes memory faults. Section III describes design methodology. Section IV reports simulation results whilst Section V concludes and suggests future works.

II. MEMORY FAULTS

There are numerous fault models applicable to memories. As new memory circuit topologies are designed, new models are required. This is to ensure that the memories are free from faults. The fault model can be summarized as follows [1]-[3]:

a. Stuck-at-faults (SAF)

These faults imply that either a cell or line is stuck to logical '1' or '0' which is called Stuck-at-1 and Stuck-at-0 respectively.

b. Transition faults (TF)

Transition faults seem to be similar to Stuck-at-faults. However, in this case once the memory cells are written to one state it is impossible to transition back.

c. Coupling faults (CF)

Coupling faults involve coupling between two adjacent cells. When a 0 to 1 (or 1 to 0) is written to a cell, it caused the neighbor cell to change its desired value. These faults can be classified into inversion or idempotent.

d. Neighborhood pattern sensitive faults (NPSF)

These faults involve a base memory cell in the centre which is surrounded by eight neighboring cells. It is possible for base memory cell to transit to a certain value influenced by its neighborhood cells.

e. Data retention faults (DF)

These faults occur in memory cells that are unable to retain their value some time after a write or read operation.

f. Address decoder faults (AF)

These faults occur due to no cell can be accessed with a certain address, multiple cells are accessed simultaneously or a certain cell can be accessed with multiple addresses.

Some of the faults that may occur in SRAM cell are pictured in Figure 2 below.

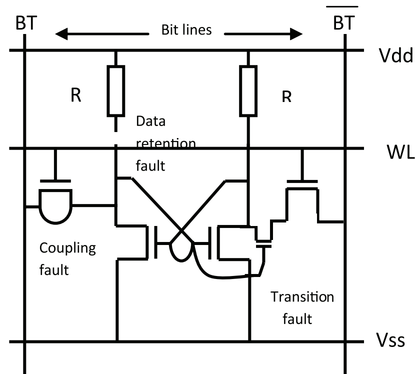

Figure 2: Possible faults in SRAM [3].

III. METHODOLOGY

The methodology used in accomplished the simulation and modeling is presented in this section. First, the concept of finite sate machine (FSM)-based memory built-in self test (MBIST) is presented. Next, FSM test pattern based on March C-algorithm is given. After that, the design entry which uses VHDL is described, and finally the behavioral simulation set up is explained.

a. Finite state machine (FSM)-based MBIST

Figure 3 shows the block diagram of the designed FSM-based MBIST architecture. It consists of a pattern controller, address generator, address limiter, data generator, read/write generator and comparator.

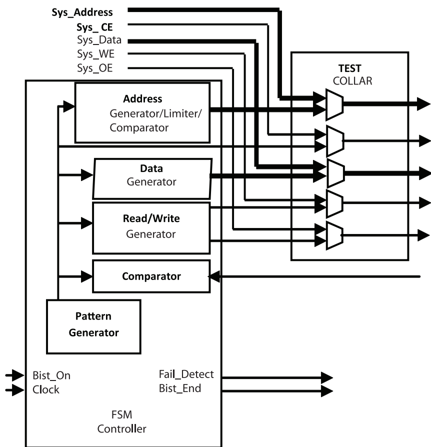

Figure 3: Block diagram of FSM-based MBIST controller

The pattern controller is an important component of this MBIST architecture. The controller interacts with other components as a finite state machine would. The state machine defines control signals and determines when the system proceeds from one stage (state) to the next. Each state has its own sub states detailing its particular operation.

The pattern controller interacts with the read/write controller to allow the correct series of "reads" and "writes" for each unique pattern. It also interacts with the data generator and address counter to provide the correct data and address stimuli to the memory.

The address counter interacts with the address limiter and address comparator for identifying the proper start and stop address points. The address comparator interacts with the pattern controller to help identify when one pattern is finished. The write/read controller determines how many cycles a given address is maintained before the address counter is incremented.

The data generator interacts with the address counter, the read/write controller and the pattern controller. These interactions allow the data generator to provide the memory with the correct data corresponding to the particular element of the test pattern.

All the portions work together to provide stimuli to the memory under test. Then the memory under test provides the output to the comparator at the memory output. A pass or fail indication is determined by comparator where the data from memory output and data generator must be equally same. The comparator also works with the remainder of the BIST sections to form a complete test cycle by cycle.

A test collar is designed to provide signal channeling either from BIST controller or from system. When BIST is turned on, data from BIST controller will be write to and read from memory. If BIST is turned off, data from system such as microprocessor will be sent to or retrieved from memory.

b. FSM test pattern

The key element for any state machine MBIST component is a counter. For example, an address generator that works in up/down sequences can be designed using counter. The counter determines which address to be inserted with appropriate data. The maximum address control signal is generated to indicate that

the entire memory addresses have been tested and hence enabling the process to proceed to next pattern.

In [4]-[5], the example of March C data generation is implemented using state machine sequence. The operation starts upon receiving data generation control signal from the controller. Each state will be executed until maximum address is reached. 'Finish' state indicates that all March C elements have been applied to entire memory.

A state diagram describing the first two operations of read/write generator of March C- (⇕ (w0), ⇑(r0, w1) are illustrated in Figure 4 [2]. The operation starts upon receiving read/write control signal from controller. Note that when the first element (write 0) has reached maximum address, the next element of read '0 and write '1' is executed. The execution will stop when 'Finish' state is reached.

c. Design entry

The various components of MBIST controller design entry is implemented using Very High Speed Integrated Circuit Hardware Description Language (VHDL) code. This versatile design language allows users to describe circuits in many levels of abstraction. The design is written in behavioral VHDL description for compatibility reason (register transfer level (RTL) modeling and synthesis). The RTL abstraction produces detailed clock-driven design at the level of registers, memories, and latches. The synthesized design is capable in meeting the functional specifications when implemented in FPGA.

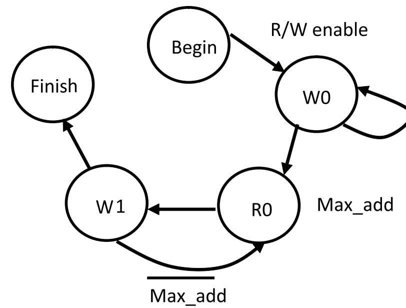

Figure 4: First two March C- algorithm state diagram [2]

Each of the components is combined using structural description. Schematic design entry can also be used for this purpose. The different is structural description is code-based (similar to behavioral description) whilst schematic is graphical symbol design technique. Although schematic seems easier to deal with than the code-based writing, schematic symbol needs to be created for each components before it can be used. Figure 5 shows the Xilinx ISE Webpack design tools, which is used to perform this step.

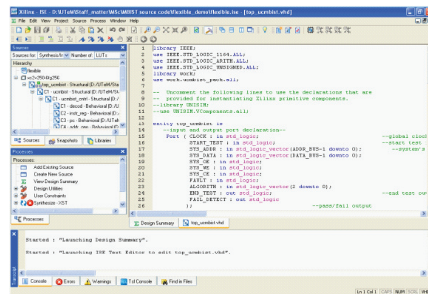

Figure 5: Xilinx ISE interface [7]

d. Behavioral Simulation

Several tools provide simulator to execute this step in order to test the design prior to the actual construction. This step is done through the use of a testbench. A testbench is a set of test stimuli with timing corresponding to circuit design. The response of the circuit under test is then read out in term of waveform or set of predetermined output dialog associated with certain errors.

The testbench file is responsible to provide input stimuli to the memory under test (MUT) such as clock and various test control signals. Then the MUT will produce a response in term of the detected failure signals that will be displayed as waveform. Others internal signals can also be displayed by selecting the simulator option. A part of waveform, output dialogues that are predetermined with expected errors can also be set. This project uses Modelsim XE III simulator provided in Xilinx ISE 8.2i design tool package from website [7].

IV. EXPERIMENTAL RESULTS

The simulation waveform on a fault-free SRAM is a reference for a website [8] is depicted in Figure 6. There are four groups of waveform namely; global clock, system signals, bist signals and memory signal. Global clock is the clock used by the architecture operated at 20 MHz.

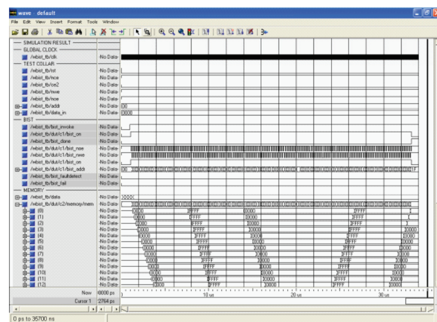

Figure 6: Fault-free memory

System signals consist of signals operated in normal cycle that is during *start_test* signal is deactivated. At this period, system signals are selected by test collar and passed to memory. Upon the activation of *start_test* signal, *bist_on* is also active which marks the start of test cycle. BIST signals are multiplexed and pass to the memory until the completion of testing phase. This can be observed when *bist_end* or *test_end* transits to logic high. Testing time is taken at this point by moving the cursor and time is automatically displayed at the bottom of the cursor.

The SRAM model is also amended to be in defective state by inserting stuck at 1 fault failure. In Figure 7, *fail_detect* signal indicates the failures detected by microcode MBIST. When tested data are dissimilar with the expected data generated by data generator, comparator will invoke this signal.

There are four *fail_detect* pulses displayed in Figure 7 but the failure is actually detected only at address 3 and 10. The stuck at fault failures set in the SRAM model can be easily detected by March C- elements. Each *rw* (read, write) element in up or down direction is able to detect these faults. For an example, bit 0 in memory address 3 and bit 15 in memory address 10 are stucked at logic 0 and logic 1 respectively. The testing starts by writing sixteen bits of zeroes into memory. When second element takes place, the read data is compared with expected all zeroes data. The conflicting data in address 10 activates the fault signal. The third element writes sixteen bits of ones into memory. Then, the read data is compared with expected all zeroes data again. At this point, data in memory 3 invokes fault signal.

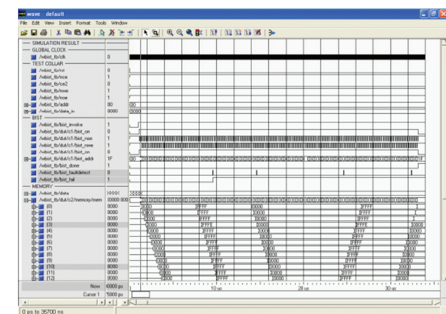

Figure7: Faulty memory

V. CONCLUSION

The simulated waveforms have shown that FSM-based MBIST architecture is an effective testing method to test embedded memories as it offers good fault coverage and low cost of testing method. Redesign

work needs to be performed only on pattern controller when new testing pattern algorithm is applied to the controller. Therefore design time and verification process can be reduced. Both two testing algorithms implemented are able to detect stuck at fault set in the SRAM under test.

This BIST circuitry can be embedded in SRAM where automatic testing can be done after manufacturing (or even by the users). By doing this, memory manufacturer can alleviate to use automatic testing equipments (test machines) which are very expensive and time consuming to test all of their products (memories). Further works will concentrate on the improvement of area overhead and testing time

VI. REFERENCES

R. Rajsuman. 2001. Design and Test of Large Embedded Memories: An Overview, in Proceedings of IEEE Design and Testing of Computers, , pp. 16-27.

R. D. Adams, 2003. High Memory Performance Memory Testing: Design Principles, Fault Modeling and Self-Test. Kluwer Academic Publishers.

C. H. Tsai and C. W. Wu. 2001. Processor-Programmable Memory BIST for Bus-Connected Embedded Memories, in Proceedings of Asia and South Pacific Design Automation Conference, pp. 325-330.

M.H. Tehranipour, Z. Navabi, and S.M. Fakhraie. 2001. An Efficient BIST Method for Testing of Embedded SRAMs, in Proceedings of International Symposium on Circuits and Systems, pp. 73-76.

S, Y. Huang and D. M. Kwai. 1998. A High-Speed Built-In-Self-Test Design for DRAMs, in Proceedings of IEEE/ACM Design Automation Conference, pp. 632-637.

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

Xilinx Inc., Xilinx ISE Webpack v8.21. http://download.xilinx.com.

A. Klindworth, "A generic VHDL entity for a typical SRAM with complete timing parameters". http://tech-www.informatik.unihamburg.de/vhdl /models/sram/sram .vhd.