FSM Based Memory BIST using Verilog-HDL

Devanshi Jariwala
Electronics and Communications Engineering
Nirma University
Ahmedabad, India
21bec050@nirmauni.ac.in

Parin Garg
Electronics and Communications Engineering
Nirma University
Ahmedabad, India
21bec082@nirmauni.ac.in

***Abstract*—Due to the growing demands of faster performance, less power consumption, & less area consumption, embedded memories used in the VLSI industry are compactly designed to operate at higher frequencies and store massive data. Dense design creates a wide variety of faults leading to the failures in the memory functions. These Memories are non-scan storage elements, i.e., they cannot be tested by simply replacing the memory cell using scan-cells in scan-based design.**

To test semiconductor memories, one of the most widely used techniques is MBIST (Memory Built-in Self-Test). BIST mechanism is a promising method to test and diagnose embedded memories like RAMs. The advantages include reduced test cycle duration and fewer complexes. March tests are widely used in production test thanks to their low time complexity and high fault coverage. The primary purpose of this project is to implement MBIST using the MARCH algorithm on an 4x4 memory cell and detect stuck-at faults, transition faults & coupling faults. The proposed BIST architecture focuses on minimizing power consumption during the testing phase while maintaining high fault coverage. It leverages the capabilities of the Verilog hardware description language to model and simulate the design. The implementations are carried out by using Verilog Hardware description language and ModelSim for waveform generation.

I. Introduction

Memory devices have become an integral part of VLSI Circuits whose sole purpose is to store large chunks of data. Dense packing of the cores creates faults in the cells, which give rise to the improper functioning of the memory. Causes of the failures depend on factors such as component density, manufacturing method, layout of the circuit, etc. So the detection of the fault is essential. Fault models has been developed to capture the effects of physical failures in RAMs. They are modeled after fault in memories so that tests to detect these faults can be used. Modeling helps to clarify, simplify and generalize the testing approach of the memory. Stuck-at fault model, Coupling fault model, Transitional fault model, etc were the important fault models relevant for the functional testing of RAMs.

As memories do not include logic gates and flip-flops, various fault models and test algorithms are required to test memories. In such conditions, design for testability (DFT) facilitates easy testing. Among the various DFT techniques BIST (Built In Self Test) is considered as the best solution for testing embedded memories within SoCs. The basic philosophy behind the BIST is "let the hardware test itself". BIST can easily be extended to provide the diagnosis data for monitoring, debugging and fault location for repairing. Memory Diagnosis is very important because memory cores are the most sensitive to process defects. Hence BIST is considered as a viable solution for both testing and diagnosing of embedded RAMs.

BIST circuit is designed to generate March tests which are widely used to detect RAM functional faults. Stuck-at fault dominate the majority of defects that occur in embedded RAMs. Dynamic Faults have been shown to be an important class of faults in the RAMs implemented with nano-scale technologies. March based algorithms were capable of locating and identifying the fault types which can help in catching errors during design and manufacturing. The March elements is marching through the memory, hence the name March Algorithm.

There are mainly two techniques for MBIST (Memory Built In Self Test) controller design, FSM based MBIST and Microcode based MBIST. FSM based MBIST controller is a hardware realization of a single testing algorithm in the form of FSM. Due to its large area overhead, use of Microcode based MBIST is not suited for SOCs. Thus, it is desirable to have an area efficient programmable MBIST for SOCs which can be achieved by introducing programmability in FSM based approach.

The report is organised as follows: Section 2 describes memory faults, which discusses all faults in memory. Section 3 discusses the BIST controller architecture, which uses march algorithms to detect memory faults. Section 4 describes the MARCH algorithm, followed by Section 5 with the simulation results. Finally, section 6 concludes the paper.

II. Types of Faults

Generally, the memory faults are due to Open connections and Shorted connections. Storage cells stuck at 0/1 Memories fail in several different ways. Memory testing focuses on testing for these memory-specific faults such as stuck-at faults, transition faults & coupling faults.

- Stuck at faults: The logic value of a cell always remains one or zero irrespective of the value written into it. There are two types of stuck at faults: stuck at zero faults and stuck at one fault. These are the most commonly occurring faults. The State Transition Graph (STG) for stuck at one fault is shown in the Fig. 1.

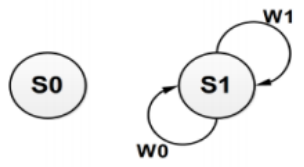


Fig. 1. FSM for Stuck-at faults

- Transition faults: A cell fails to undergo a transition from one logic state to the other. There are two types: rising transition fault and falling transition fault. Rising transition fault is one where a cell fails to make a transition from zero to one whereas falling transition fault is one where a cell fails to make a transition from one to zero. The state diagram for rising transition fault is shown in Fig. 2.

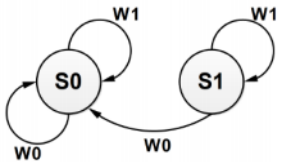


Fig. 2. FSM for Transition Faults

- Coupling faults: The logic value of one cell is affected by the operation or the content in the other cell. There are three types of coupling faults:
  - Inversion: The value in the victim cell is inverted if the aggressor cell has a transition.
  - Idempotent: The value in the victim cell is forced either to 1 or 0 when the aggressor cell has a transition.
  - State: The victim cell is forced to 1 or 0 if the aggressor cell is in a certain state.

  The state diagram for (rising inversion) coupling fault in shown in Fig. 3.

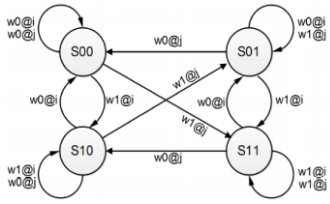


Fig. 3. FSM for Coupling faults

III. Controller Architecture

A general BIST architecture comprises a pattern controller, address generator, address limiter, data generator, read/write generator, and comparator. Whereas our architecture comprises three main components

- Test Collar
- MBIST Controller
- Memory Unit

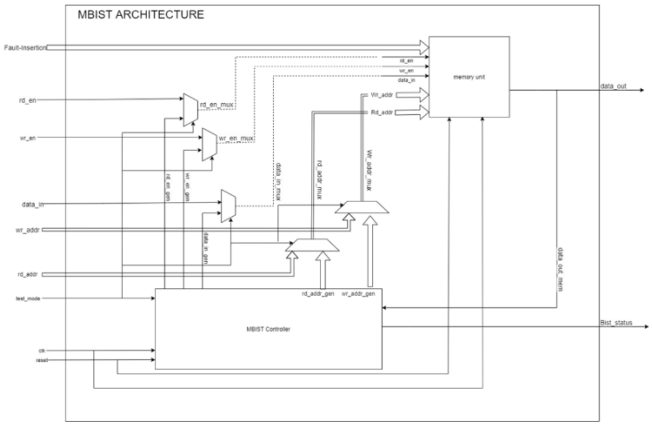
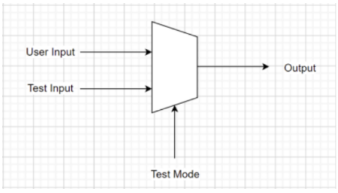


Fig. 4. MBIST Architecture

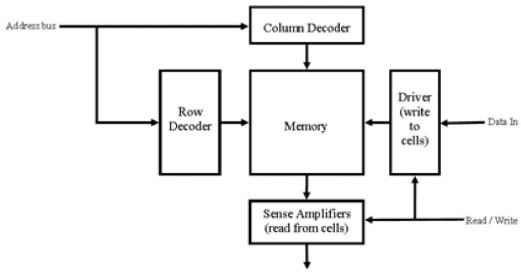
A. Test Collar

Test collar is a MUX based switch that switches between Normal Working Mode or Test Mode. When BIST is turned on, data from the BIST controller will be written into and read from memory. If BIST is turned off, data from the system such as the microprocessor will be sent to or retrieved from memory.



B. Memory Unit

A typical memory model consists of memory cells connected in a two-dimensional array, and hence the memory cell performance must be analysed in the context of the array structure. Row decoders and column decoders are used to split the given address into its respective row and column address and access the memory location. Here, we are designing a 4*4 RAM which consists of 1-bit data input, 1-bit data output, and a few read/write control signals along with 2 bits giving us the row locations and 2 bits giving us the column locations. The row and column decoders are present within the controller itself.



C. MBIST Controller

MBIST controller is the heart of the architecture, responsible for implementing the March Algorithm for fault detection. This MBIST controller is implemented in an FSM based architecture, where the controller interacts with various states of the algorithm with the help of a few control signals. The state machine defines control signals and determines when the system proceeds from one stage (state) to the next. Each state has its sub-states detailing its operation. Here we are using the March X algorithm for testing our memories.

IV. MARCH ALGORITHM

Memories are tested with special algorithms which detect the faults occurring in memories. Several different algorithms can be used to test RAMs and ROMs like:

- 0/1 algorithm
- Checker-board algorithm
- MARCH algorithm etc.

These algorithms can detect multiple failures in memory with a minimum number of test steps and test time. March tests have proved to be simpler and faster among the different algorithms proposed to test RAMs and have emerged as the most popular ones for memory testing. There are various types of March tests with varying coverage of fault.

The simplest (linear-time) tests that detect SAFs, TFs, and CFs are part of a family of tests called the Marches (i.e., the March tests). A March test consists of a finite sequence of March elements. A March element is a finite sequence of operations applied to every cell in the memory array before proceeding to the next cell. An operation can consist of writing a 0 into a cell (w0), writing a one into a cell (w1), reading an expected 0 from a cell (r0), and reading an expected one from a cell (r1). There are various types of MARCH algorithms used to detect faults in the memory. Some of which are tabulated below:

Algorithm	Definition
MATS+	{ ↕(w0); ↑(r0,w1); ↓(r1,w0); }
March X	{ ↕(w0); ↑(r0,w1); ↓(r1,w0); ↕(r0); }
March C-	{ ↕(w0); ↑(r0,w1); ↑(r1,w0); ↓(r0,w1); ↓(r1,w0); ↕(r0); }
March Y	{ ↕(w0); ↑(r0,w1,r1); ↓(r1,w0,r0); ↕(r0); }
March A	{ ↕(w0); ↑(r0,w1,w0,w1); ↑(r1,w0,w1); ↓(r1,w0,w1,w0); ↓(r0,w1,w0); }
March B	{ ↕(w0); ↑(r0,w1,r1,w0,r0,w1); ↑(r1,w0,w1); ↓(r1,w0,w1,w0); ↓(r0,w1,w0); }
March U	{ ↕(w0); ↑(r0,w1,r1,w0); ↑(r0,w1); ↓(r1,w0,r0,w1); ↕(r1,w0); }
March SS	{ ↕(w0); ↑(r0,r0,w0,r0,w1); ↑(r1,r1,w1,r1,w0); ↓(r0,r0,w0,r0,w1); ↓(r1,r1,w1,r1,w0); ↕(r0); }

In this project we have implemented MARCH X algorithm for fault detection:

March X Algorithm:

Step1: write 0 with Increasing addressing order

Step2: read 0 and write 1 with Increasing addressing order

Step3: read 1 and write 0 with Decreasing addressing order

Step4: read 0 with Decreasing addressing order

V. SIMULATION AND RESULTS

The MBIST implementation was subject to comprehensive smiluation testing to evaluate its effectiveness in setecting faults.

First, we run the algorithm in testing mode switched to high. It runs firsts in normal mode, with fault register set at 2'b11.

Next, it moves on to check stuck at faults, with fault register set to 00. No stuck at faults are detected in the below give case.

When fault register holds value 2'b01, the algorithm checks for transitional faults, separately evaluating the main module for presence of 0 to 1 and 1 to 0 transition faults. None are found in the below case.

Finally, the fault register holds value 2'b10, checking for coupling faults. When none are found, the algorithm reverts back to running in normal mode, with fault register again holding value 2'b11.

The simulation waveforms obtained are:

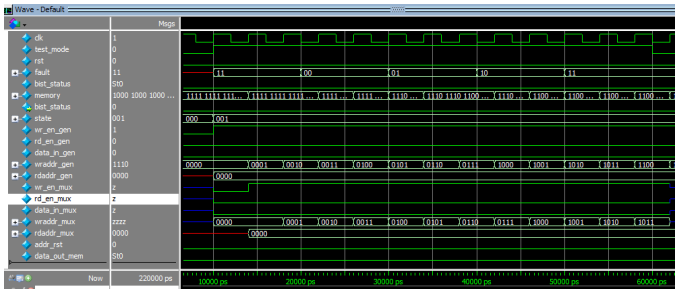


Fig. 5. Testing Mode Simulation

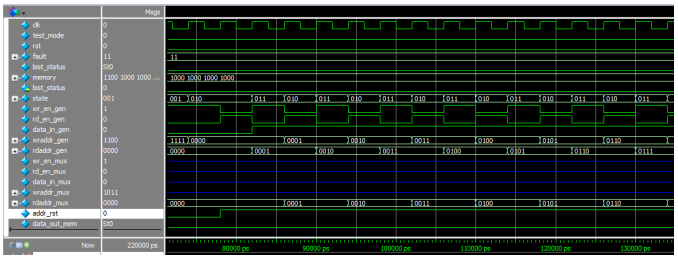


Fig. 6. Normal Mode Simulation

```
VSIM 8> run
# Normal Operation
run
# Struck-at-faults detection
run
# transition faults detection
# No transition faults detected
run
# No transition faults detected
# No transition faults detected
# Coupling detection
run
# Normal Operation
run
run
run
run
# No error
# No error
```

Fig. 7. ModelSim Transcript

VI. Conclusion

A programmable FSM based MBIST controller design is proposed in this paper. An up to 32- bit programmable Logic BIST is designed in Verilog and tested for a sample design. For a particular IC, the size is fixed and hence with just the clock signal and the test mode signals the particular IC can be tested. The output is also simplified with just the single signal indicating the pass or fail status after the signature analysis.

VII. Future Scope

We have implemented it on 8x8 RAM, but it can be implemented on RAM of any size. Also, we can implement a self-repair module with this architecture.

References

[1] N. Q. M. Noor, Y. Yusof and A. Saparon, ”Low area FSM-based memory BIST for synchronous SRAM,” Signal Processing & Its Applications,2009. CSPA 2009. 5th International Colloquium on, Kuala Lumpur, 2009, pp. 409-412.

[2] P. E. Joseph and P. R. Antony, ”VLSI design and comparative analysis of memory BIST controllers,” Computational Systems and Communications (ICCSC), 2014 First International Conference on, Trivandrum, 2014, pp. 372-376.

[3] P.C. Tsai, S.J. Wang, and F.M. Chang, “FSM-Based Programmable Memory BIST with Macro Command”, in Proc. Memory Technology, Design, and Testing, pp. 72-77, 2005.

[4] A NOVEL APPROACH IN MEMORY BIST USING MODIFIED MARCH C ALGORITHM IN SRAM BASED FPGA R.Karthick, Dr.V.Saminadan Research Scholar, Department of ECE, Pondicherry Engineering College, Puducherry. Professor, Department of ECE, Pondicherry Engineering College, Puducherry

[5] Comprehensive Study on Designing Memory BIST: Algorithms, Implementations and Trade-offs By: Allen C. Cheng Advanced Computer Architecture Lab Department of Electrical Engineering and Computer Science The University of Michigan Ann Arbor, MI 48109-2122 accheng@eecs.umich.edu

[6] Memory Interface Unit for Microcode Based Memory BIST Controller Vadde SeethaRama Rao, Chilumula.RamBabu Assistant Professor in ECE Department Sreenidhi Institute of Science and Technology,Hyd