IMPLEMENTATION OF MBIST

Report by:
Bheema Lakshmi Pradeep (2021702013)
Bhartipudi Sahishnavi (2021702005)
Megha Saha (2021702010)

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. As these memories are large and densely packed, there is a high possibility of faults. 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). We are using March algorithms since March based tests are simple and possess good fault coverage. The primary purpose of this project is to implement MBIST using the MARCH algorithm on an 8x8 memory cell and detect stuck-at faults, transition faults & coupling faults.

I. INTRODUCTION:

There has been exponential growth in the digital VLSI world in recent times. Currently, the latest digital system consists of data paths, control paths, and memory devices. Memory devices have become an integral part of VLSI Circuits whose sole purpose is to store large chunks of data. Since these memories are large and densely packed, there is a high possibility of faults such as stuck-at faults, transition faults, coupling faults etc. As memories do not include logic gates and flip-flops, various fault models and test algorithms are required to test memories. There are many DFT techniques used to test the memories. But each method has its pros & cons. As embedded memories are dense and compact, BIST has been proven to be one of the most cost-effective and widely used solutions as 1) No external test equipment; 2) Reduced development efforts. 3) Tests can run at circuit speed to yield a more realistic test time.

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.

TYPE 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. The three main parts. 1) Address decoder logic 2) Memory cell array 3) Read/write logic, can each have flaws that cause the device to fail. Memory testing focuses on testing for these memory-specific faults such as

stuck-at faults, transition faults & coupling faults.

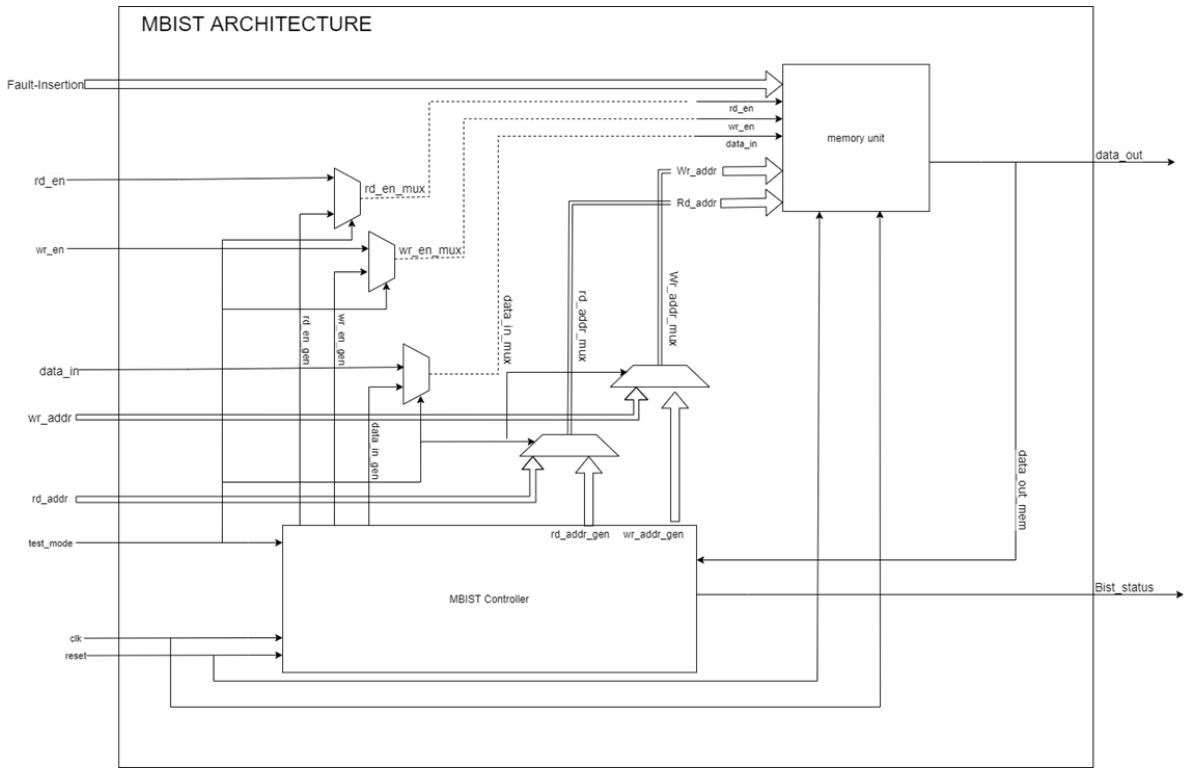
Stuck At Faults (SAF): A memory fails if one of its control signals or memory cells remains stuck at a particular value. Stuck-at faults model this behaviour, where a signal or cell appears to be tied to power (stuck-at-1) or ground (stuck-at-0). To detect stuck-at faults, you must place the value opposite to that of the stuck-at fault at the fault location. To detect all stuck-at-1 faults, you must place 0s at all fault locations. To detect all stuck-at-0 faults, you must place 1s at all fault locations.

Transition Faults (TF): A memory fails if one of its control signals or memory cells cannot make the transition from either 0 to 1 or 1 to 0. An up transition fault, the inability to change from 0 to 1, and a down transition fault, the inability to change from a 1 to a 0. The following figure shows a cell that might behave normally when a test writes and then reads a 1. It may even transition properly from 1 to 0. However, when undergoing a 0->1 transition, the cell could remain at 0—exhibiting stuck at-0 behaviour from that point on. However, a stuck-at-0 test might not detect this fault if the cell was at one originally.

Coupling Faults (CF): Memories also fail when a write operation in one cell influences the value in another cell. Coupling faults model this behaviour. Coupling faults fall into several categories: inversion, idempotent, bridging, and state.

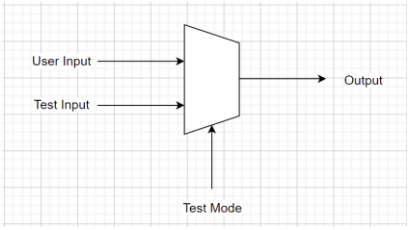
II. CONTROLLER ARCHITECTURE:

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 1) Test Collar, 2) MBIST Controller 3) Memory unit.



1) Test Collar:

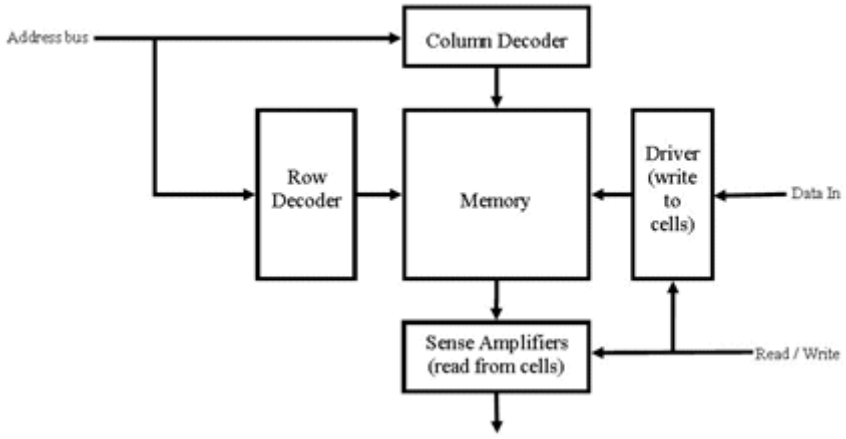
Test collar is a MUX based switch that switches between Normal Working Mode or Test Mode. When BIST is turned on, data from the BIST controller will be written into and read from memory. If BIST is turned off, data from the system such as the microprocessor will be sent to or retrieved from memory.



2) 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.



3) 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.

III. TEST 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 coverages of fault.

MARCH ALGORITHM:

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	Description
MATS	$\{ \Updownarrow (w0); \Updownarrow (r0, w1); \Updownarrow (r1) \}$
MATS+	$\{ \Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0) \}$
MATS++	$\{ \Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0, r0) \}$
MARCH X	$\{ \Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0) \}$
MARCH C–	$\{ \Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0) \}$
MARCH A	$\{ \Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0) \}$
MARCH Y	$\{ \Updownarrow (w0); \Uparrow (r0, w1, r1); \Downarrow (r1, w0, r0); \Updownarrow (r0) \}$
MARCH B	$\{ \Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0) \}$

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

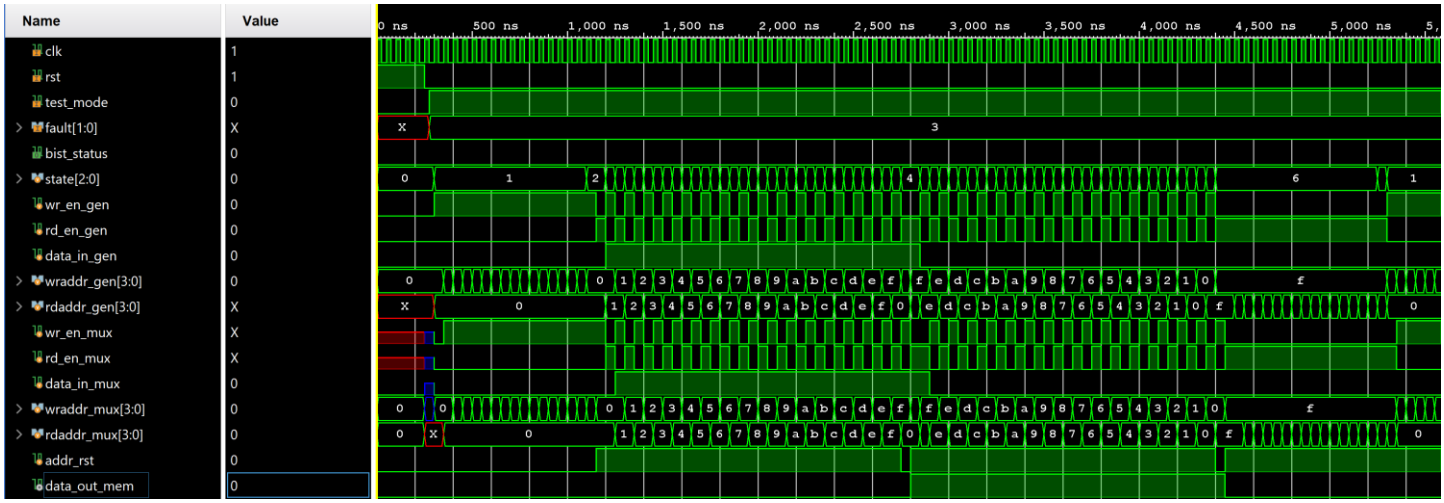
Step3: read 1 and write 0 with Decreasing addressing order

Step4: read 0 with Decreasing addressing order

IV. SIMULATION RESULTS:

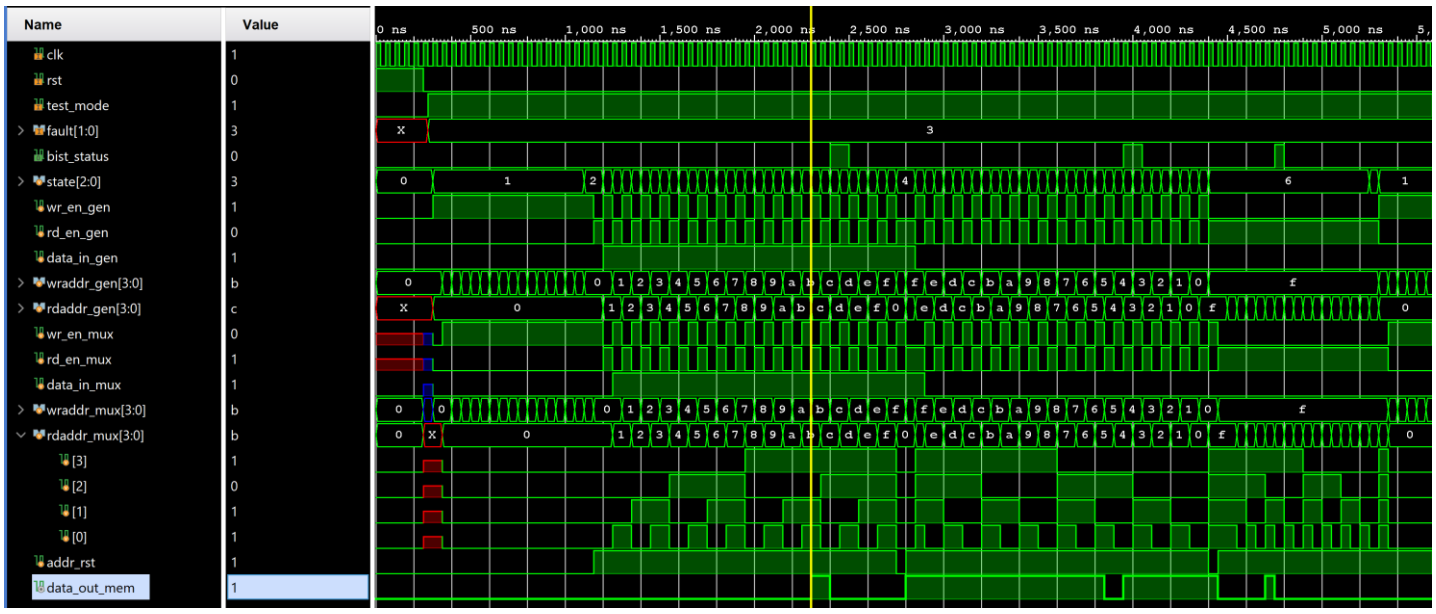
- **Fault Free condition:**

No Faults are introduced in the memory unit.

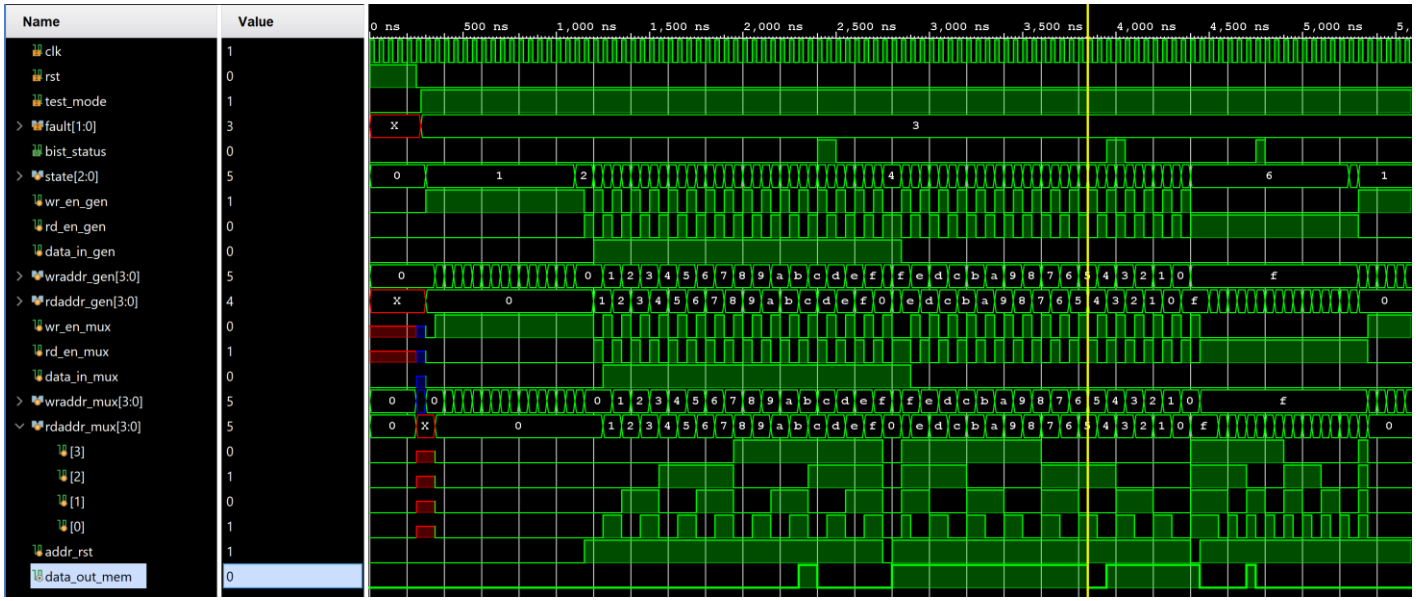


- **Stuck-at-Faults:**

During the “READ-0” operation, 1 is detected at (2,3) due to S-A-1 fault.

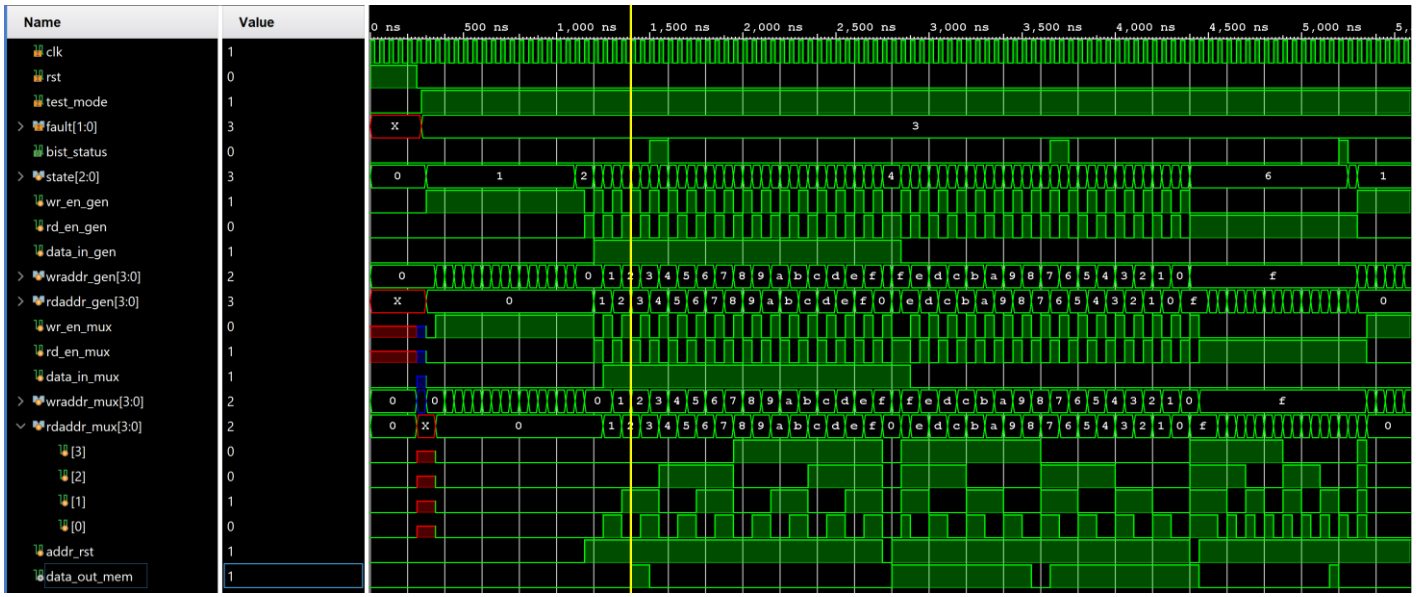


During the ‘READ-1’ operation, 0 is detected at (1,1) due to the S-A-0 fault.

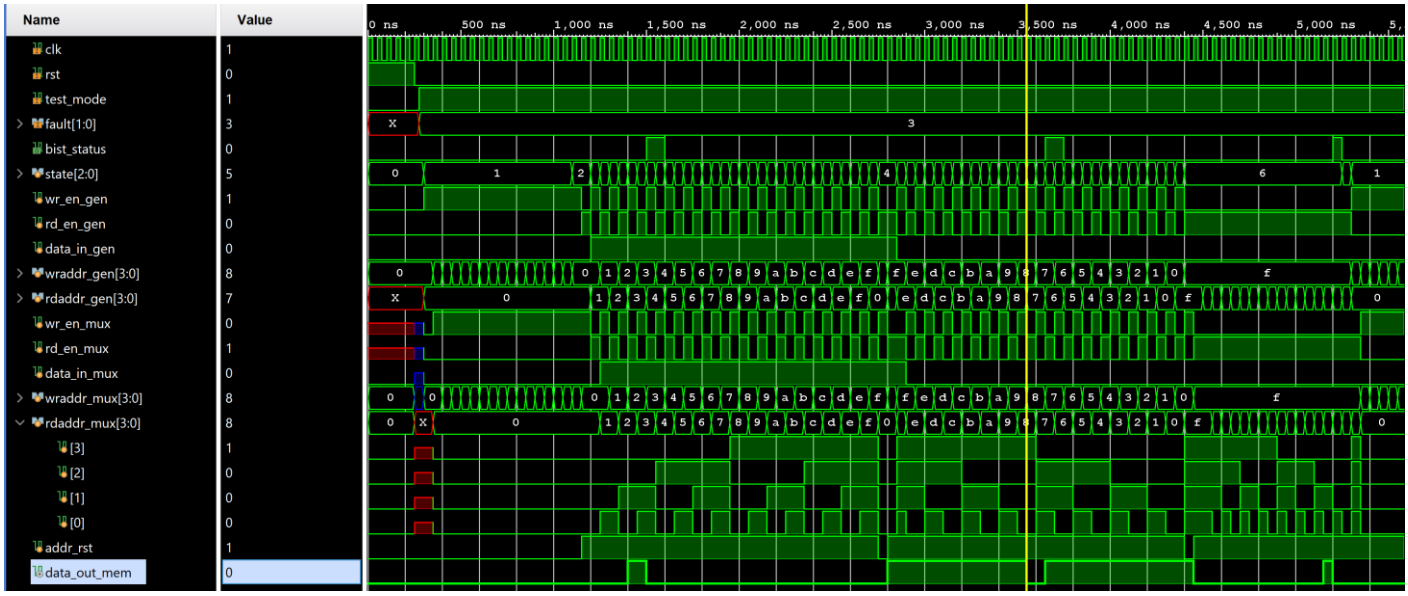


- **Transition Faults:**

During the “READ-1” operation, 0 → 1 is detected at (2,0) due to transition fault.

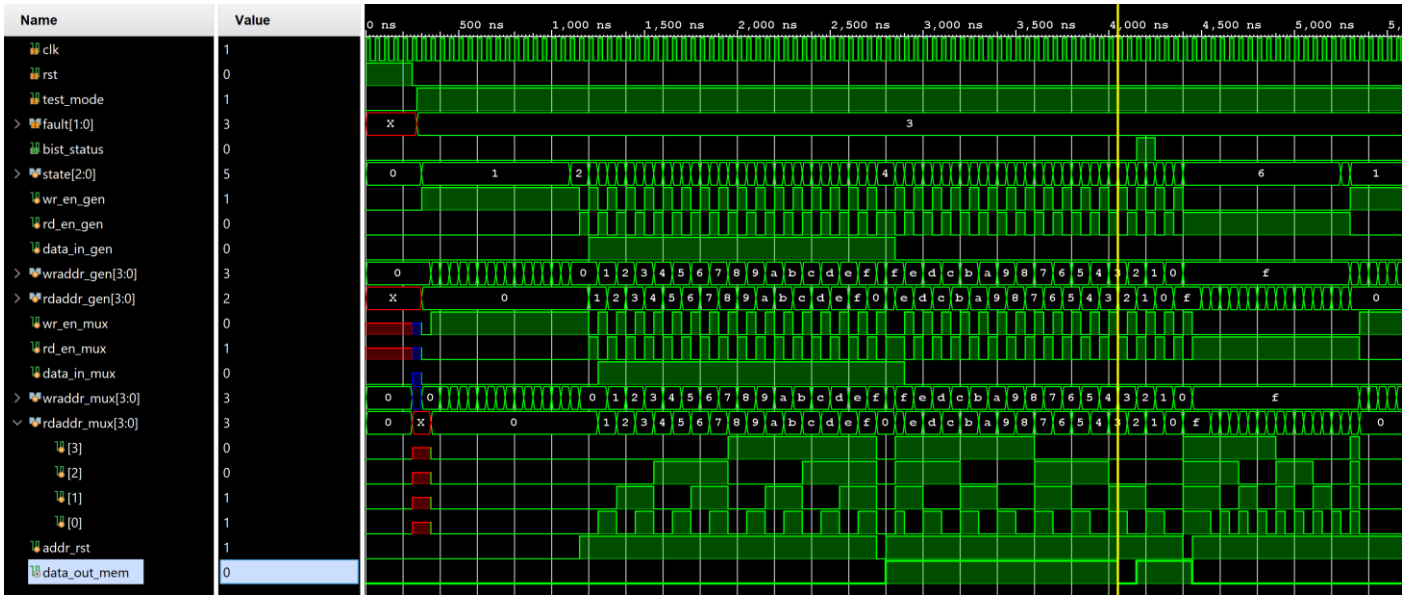


During the second “READ-0” operation, 1→ 0 is detected at (0,2) due to the S-A-0 fault.



- **Coupling Faults:**

During the “WRITE-1” operation, (1,3) changes from 0→ 1. Thus, (0,3) location toggles indicate the coupling faults. This fault is observed in the “READ-1” state.



V.CONCLUSION:

We have successfully implemented the MBIST using March March X algorithm and tested its functionality by adding a few faults into the memory and detecting it. We have inserted the following faults and successfully detected them.

Fault Model	Fault type	Fault Location
Struck at faults	**struck at 0**	**(1,1)**
	struck at 1	**(2,3)**
Transition Faults	0->1	(2,0)
	1->0	(0,2)
Coupling Faults	when (1,3) changes to 0->1 (0,3) toggles	(0,3)

VI. 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.

VII. REFERENCES:

1. *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*
2. *PATH DELAY TEST THROUGH MEMORY ARRAYS A Thesis by PUNJ POKHAREL.*
3. *Implementation of Finite State Machine (FSM) Based March SS Algorithm for Testing of Memory * Yogendra Yadav ** Neha Sharma *** Shraddha Bansal*
4. *FPGA based High Speed Memory BIST Controller for Embedded Applications Mohammed Altaf Ahmed1*, D. Elizabeth Rani1 and Syed Abdul Sattar2 1GITAM Institute of Technology, GITAM University, Visakhapatnam – 531173, Andhra Pradesh, India; altaface1@gmail.com, kvelizabeth@rediffmail.com; hod_ei@gitam.edu 2Royal Institute of Technology and Science, Chevella – 501503, Andhra Pradesh, India; syedabdulsattar1965@gmail.com*
5. *Modeling and Simulation of Finite State Machine Memory Built-in Self Test Architecture for Embedded Memories Nor Zaidi Haron1 , Siti Aisah Mat Junos @Yunus2 , Abdul Hadi Abdul Razak3 and Mohd. Yamani Idna Idris4 1, 2Faculty of Electronics and Computer Engineering, Universiti Teknikal Malaysia Melaka, Locked Bag 1200, Hang Tuah Jaya, 75450, Ayer Keroh, Melaka, Malaysia. 3 Faculty of Electrical Engineering, Universiti Teknologi MARA, 40450, Shah Alam, Selangor, Malaysia. 4 Faculty of Computer Science & Information Technology, University of Malaya, 50603, Kuala Lumpur, Malaysia. zaidi@utem.edu.my1 , aisah@utem.edu.my2 , adi3443@salam.uitm.edu.my3 , yamani@um.edu.my4 .*
6. *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*

7. *Designing, Modeling and Implementation of Memory Built In Self Test (BIST) Controller Jayashri Patil1 , Shashank Pujari2,, Dr. A.D. Shaligram3 1 ,2International Institute of Information Technology, Pune, India Email: jayashripatil3@yahoo.com shashankp@isquareit.ac.in 3 Name Pune University, Pune, India Email: adshaligram@gmail.com*
8. *Memory Interface Unit for Microcode Based Memory BIST Controller [1] Vadde SeethaRama Rao, [2] Chilumula.RamBabu [1][2] Assistant Professor in ECE Department [1][2] Sreenidhi Institute of Science and Technology,Hyd*
9. *BIST Memory Design and Testing 1Ms. S.DIVYA, 2 S.Tamilselvan, 3A. Selvakumar, 4V. Sureshkumar 1Assistant Professor, 2,3,4B.E. Students Department of ECE, SREC Coimbatore, India*
10. *https://ieeexplore.ieee.org*
11. *GitHub: Where the world builds software · GitHub*