Indian Journal of Science and Technology, *Vol 8(33), DOI: 10.17485/ijst/2015/v8i33/76080, December 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

FPGA based High Speed Memory BIST Controller for Embedded Applications

Mohammed Altaf Ahmed[1]*, D. Elizabeth Rani[1] and Syed Abdul Sattar[2]

[1]GITAM Institute of Technology, GITAM University, Visakhapatnam – 531173, Andhra Pradesh, India; altaface1@gmail.com, kvelizabeth@rediffmail.com; hod_ei@gitam.edu
[2]Royal Institute of Technology and Science, Chevella – 501503, Andhra Pradesh, India; syedabdulsattar1965@gmail.com

Abstract

In the current high speed, low power VLSI Technology design, Built in Self Test (BIST) is emerging as the most essential part of System on Chip (SoC). The industries are flooded with diverse algorithms to test memories for faults. The March based algorithms are become popular so quickly for locating faults in memories. This research study attempt to design the memory BIST controller for March 17N as selected algorithm. It tests various memories for faults. A simple architecture is implemented in Verilog Hardware Description Language (HDL), which can be easily integrate with SoC and is able to locate the fault location in the semiconductor memories. Integration of memory BIST controller in SoC design improves chip yield. The design has achieved 497.47MHzof maximum frequency by use of only 158 slice LUTs on Virtex-7 Field Programming Gate Array (FPGA) device. The proposed memory BIST controller is suitable for SoC integration to test various memories at high speed with very low area overhead.

Keywords: Low Area, Memory BIST, SoC Integration, Test Memories, Yield Improvement

1. Introduction

Embedded memories in Integrated circuit and System on Chip (SoC) play a vital role. In 1981, an industry leader was rumored to have said, "Nobody will need more than 640 KB of RAM"[1]. But it would never be enough and even after more than three decades nobody can assure that the requirement of memories in IC is limited. The popularity and requirement of embedded memories are drastically increases since 1970s to present and expecting to continue in the future.

Now a days, semiconductor memories are extensively used to store huge volume of data in almost all digital systems. for example, latest smart cell phones, weather forecasting wrist watches are uses large amount of data. According to a survey conducter by a well known Semiconductor Industry Association (SIA). In 1999, 20 percent area of SoC are occupied by memories while in 2005 it was 71 percent. They had expected that it will be reached to 94 percent till 2014. The International Technology Roadmap for Semiconductors (ITRS) report is available for public[2,3] to track the survey. This suvey report is graphically represented in Figure 1.

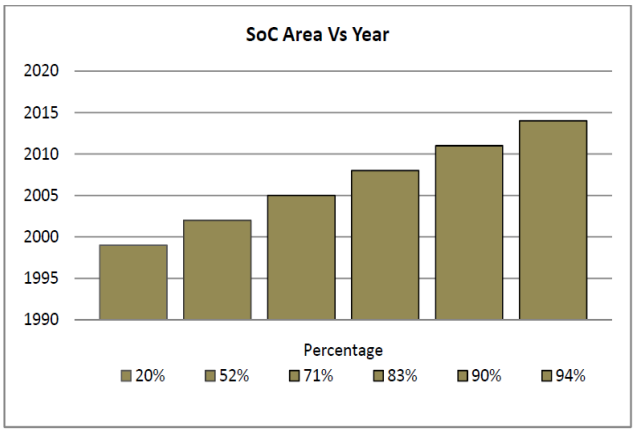

Figure 1. Area occupied by memories in SoC.

*Author for correspondence

The density in the IC increases, it results in the complex systems. The complexity in the system introduces physical and mechanical defects and increases the test complexity and cost of SoC. Test cost of embedded memory is increases with an exponential increase in density. Shrinking in feature size of the components introduces the sensitivity to fault deeply. Faults become more complex and thus testing cost of memories become larger than manufacturing cost of memories and System-on-Chip (SoC) behaves as a memory dominant chip.

Testing/diagnosing of embedded memories in SoC quickly becomes a real challenge and start affecting directly to the chip yield[4]. In respond to this challenge, the embedded memory self-test designs have introduced, and many Built-in-Self-Test algorithms[5] have developed to improve the total chip yield. As compare to the conventional method, uses Automatic Test Equipment (ATE) for testing memories this method is more cost effective and efficient. Therefore self testing algorithms are widely used in embedded memory testing.

Thus, Effective memory diagnostics and failure analysis methodologies help in enhancing the yield of SoC products, especially with fast revolution in new product development and advanced process technologies[4]. Generally memory BIST designs are fixed and cannot be changed once committed to silicon. Therefore, it must be ensure that the test algorithm used should provide high level of fault coverage. This will insure catching of all possible faults before committing to silicon.

This paper focuses to the algorithms used for testing memories in SoC and faults coverage through them. March based algorithms[6] have a wonderful solution for this scenario. As per comparison of various March algorithms[7], on the basis of fault coverage and error free memory assurance, the March 17N algorithm[8] has chosen in this research. This paper implemented March 17N algorithm for high speed and low area performance. The memory BIST controller architecture has designed and tested on Field Programmable Gate Array (FPGA) device[9]. The results obtained by selecting the device family Virtex 7 and Target Device xc7vx330t-3-ffg1157, as the strategy adopted in paper[10].

2. MBIST Principle

MBIST principal[11] is shown in Figure 2. This block diagram consists of memory BIST controller (BIST pattern generator) and memory model. The BIST controller has two output shows the test accomplishment signals (tst_done) and test pass/fail signal (fail_h). After test ends, tst_done become '1'. And If the test done without any error(pass) fail_h is set '0', otherwise '1'(fail). With this MBIST controller principle, the memories can be tested externally.

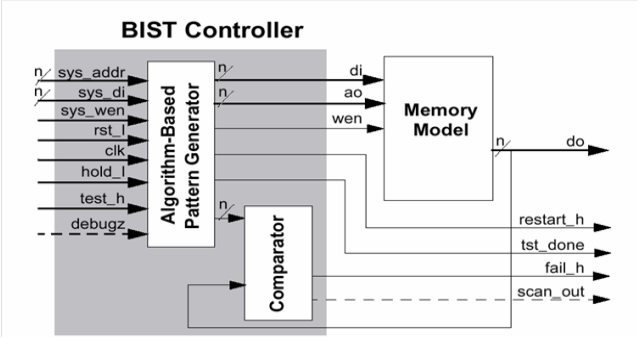

Figure 2. Memory BIST Principle.

3. Algorithm

To implement the Memory Built-in- self-test controller the algorithm is chosen to test the memories in SoC is March 17N algorithm[8] illustrated in Figure 3.

⇑ (w0); ⇑ (r0,w1,r1); ⇑ (r1,w0,r0); ⇑ (r0,w1);

⇓ (r1,w0,r0); ⇑ (r0); ⇓ (r0,w1,r1); ⇑ (r1);

Figure 3. March 17N Memory BIST algorithm.

The notations used in Figure 3 are explain in Table 1. The March algorithm consists of several March elements, separated by semicolon. The up-arrow stand for ascending order of the address sequence, down-arrow stands for descending order. Inside the parenthesis is the specification of read writes operation and the corresponding data background. These read-write operations are to be applied to each address, one by one, following the address order in front of the parenthesis.

Table 1. Notation of March17N Algorithms

Operation	Description
⇑	address 0 to address max
⇓	address max to address 0
w0	write 0
w1	write 1
r0	read a cell whose value should be 0
r1	read a cell whose value should be 1

All operations inside the parenthesis have to perform before we proceed to the next address. We use the March signature to represent the results from all operations in the test algorithm, which are either correct (represented by 0) or incorrect (represented by 1).

We assume here that only the read operation can detect the failure. For some special memories, however write-through and write-verify operations can also detect the error.

4. Architecture

The March 17N memory bist controller architecture consists of three different blocks, memory bist registers block, memory bist state machine block and external memory block shown in Figure 4. Memory bist registers block consist of two registers internally, BIST kickoff and bist status. Each of these blocks is designed separately using an HDL code and then verified individually. After the satisfactory implementation of all the blocks, these blocks are integrated to perform the memory diagnosis operation.

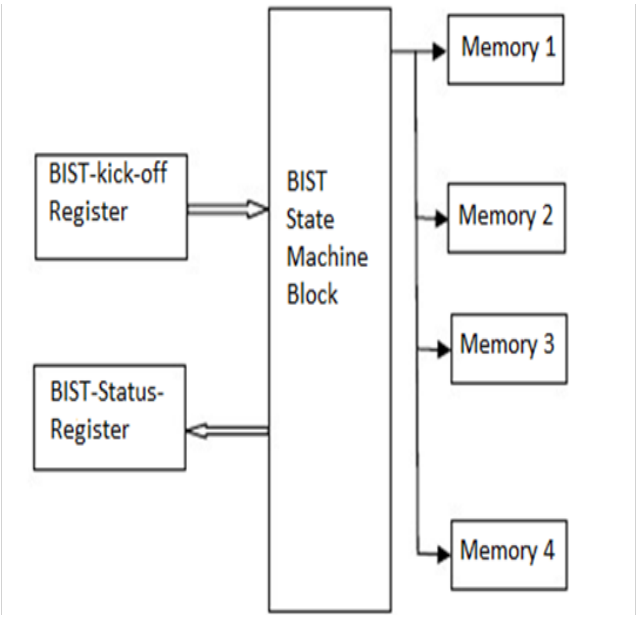

Figure 4. Memory BIST Controller Architecture.

4.1 Register Block

Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register". Registers are now usually implemented as a register file, but they have also been implemented using individual flip-flops, high-speed core memory, thin film memory, and other ways in various machines.

4.1.1 BIST Kickoff Register

It is a 32 bit register with address 32`h0; here it is used as an input register under read/write mode. The register bits are assigned shown below in the Table 2.

Table 2. BIST Kickoff Register

Register bit	Bit position	Mode	Description
BIST_MEMORY_ID	[31:30]	Read / Writable	Selection of memory for performing the operation.
	[29:25]		*Reserved*
BIST_BROADCAST	[24]	Read/ Writable	Performs operations to all the memories.
	[23:17]		*Reserved*
BIST_RESET	16	Read / Writable	Reset bist state machine
	[15:7]		*Reserved*
FORCE_BIST_ERROR	8	Read / Writable	Force bist error inserts errors while performing write operations in memory bist state machine.
	[7:4]		*Reserved*
BIST_HALT_ON_ERROR	3	Read / Writable	If it programmed at failoccured, memory bist state machine will halt.
BIST_RESUME	2	Read / Writable	Resume memory bist state machine, if a fail occurs.
BIST_STOP	1	Read / Writable	Stop memory bist state machine
BIST_START	0	Read / Writable	Kick off memory bist State the machine.

4.1.2 BIST Status Register

It is a 32 bit register with address 32`h1, here it is used as an output register under read only mode. The register bits are assign as given below in the Table 3.

Table 3. BIST Status Register

Register bit	Bit position	Mode	Description
BIST_FAIL_ MEMORY_ID	[31:30]	Read -only	Selection of Memory for Performing Memory BIST Operation.
	[29:28]		*reserved*
BIST_FAIL_ ADDR	[27:16]	Read –only	If fail occurred it display fail address.
	[15:12]		*Reserved*
BIST_FAIL_ ERR_POS	[11:7]	Read-only	It display the error position at fail address.
	[6:3]		*Reserved*
BIST_FAIL	2	Read -only	Fail MBIST State machine.
BIST_PASS	1	Read -only	Pass Memory BIST State Machine
BIST_DONE	0	Read only	Finish the operations of MBIST State Machine.

4.2 Memory BIST State Machine

To kick off the state machine, bist_start bitis programmed in BIST_KICKOFF Register. It will trigger the state machine, and it starts performing March 17N Memory BIST algorithm. It starts performing Write zero, read zero, write one, and read one operation as described in algorithm. The operation of the state machine is explained by Algorithmic State Machine (ASM) chart shown in Figure 5. Controller select memory by reading memory id signal and starts MBIST operations. Display all information in the status register if fail occurs else jump to the next location and complete operation. Each of the set of operations in the parenthesis is perform in a state. There are eight different kind of read and write operations required to perform as part of the Memory bist algorithm.

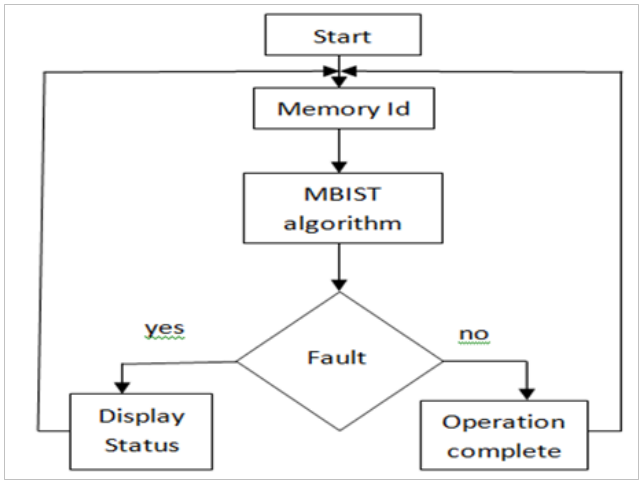

Figure 5. Memory BIST Algorithmic State Machine (ASM) Chart.

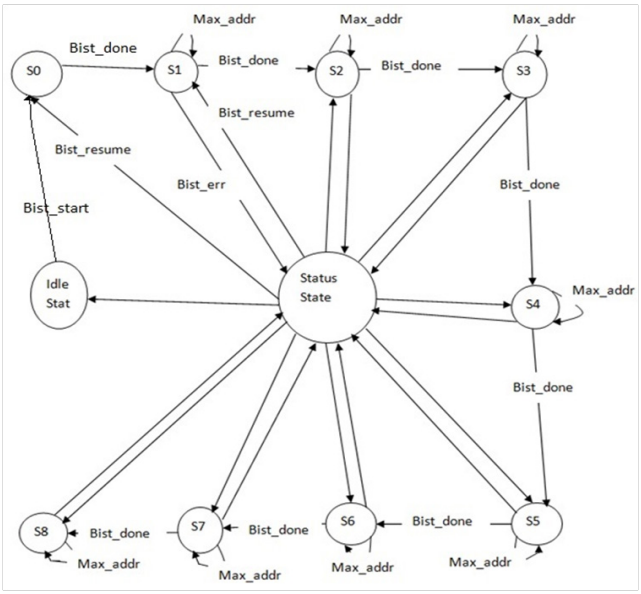

Figure 6. Memory BIST State Diagram.

The state Machine shown in Figure 6 consists of following states:

1. ST_IDLE
2. ST_1_W0_A
3. ST_2_R0W1R1_A
4. ST_3_R1W0R0_A
5. ST_4_R0W1_A
6. ST_5_R1W0R0_D
7. ST_6_R0_A
8. ST_7_R0W1R1_D
9. ST_8_R1_A
10. ST_STATUS

The Memory BIST State machine waits in ST_IDLE State until bist_reset deactivate, waiting for bist_start to programmed in Bist_Kickoff_Register. As soon as bist_start is programm, the state machine jumps to ST_1_W0_A state and starts writing zeros from address 0 to max address in ascending order. It will be in this state till

it performs the write operation to the overall memory. Then it jumps to the next state ST_2_R0W1R1_A and starts reading zeros, writing ones, and reading zeros operation at each address location. After performing read it compares against the expected values if comparison mismatches, it jumps to ST_STATUS state. While jumping to this state it also provides information like bist_comp_fail, bist_fail_address, bist_fail_mem_id, bist_fail_err_pos and the appropriate state information (state_info).

If bist_stop is programm anytime during the running of Memory BIST operation. It jumps to ST_STATUS State, asserts bist_done on the next clock cycle, checks the bist_fail_counter and asserts bist_pass or bist_fail along with error information in Bist_Status_Register.

In ST_STATUS, it latches the information (like bist_comp_fail, address, err_pos and mem_id) it got from the earlier state into the Bist_Status_Register, and it also increments a bist_fail_counter in this state. If bist_halt_on_error programmed the time when memory BIST was kicked off by bist_start, it waits in this state for bist_resume or bist_stop. If it gets a bist_stop signal. It asserts bist_done on the next clock and jumps back to ST_IDLE state and wait for bist_start; If it gets a bist_resume in this state it clears all the error information in BIST_STATUS_REGISTER and jumps back to the state (state_info) from where it arrived and starts performing Memory BIST operation from the next address into the memory.

If bist_halt_on_error is not programm when memory BIST was kicked off, while doing memory BIST operation. Whenever it gets any failure, it jumps to ST_STATUS state, latches all the error information into the Bist_Status_Register, increments the bist_fail_counter and jumps back to the state from where it arrived (state_info), irrespective of whether it gets bist_resume or not.

After coming back to the earlier state. The state machine continues performing Memory BIST Operation to the rest of the memory and jumps to the next state, if it gets any error in the consecutive states; it jumps to ST_STATUS state and performs the same operation described earlier. When the state machine has done all the operations in the last Memory BIST state (ST_8_R1_A), it jumps to ST_STATUS State and asserts bist_done to indicate completion of Memory BIST Operation. Here in the state if bist_fail_counter is non-zero it asserts bist_fail and appropriate last error information in Bist_Status_Register along with bist_done, to signify Memory BIST Operation failed. If bist_fail_counter is zero then, it asserts bist_pass and bist_done in Bist_Status_Register to signify Successful Completion of Memory BIST.

The Memory BIST Operation is perform onto the memory whose id is programmed in bist_mem_id in Bist_Kickoff_Register. While programming bist_start, if bist_broadcast is programm, the state machine will trigger memory BIST for each memory one after the other in a sequential fashion. When bist broadcast is programm, then after completion of memory BIST for each memory (bist_done), the status gets updated in Bist_Status_Register.

In Non-broadcast mode, irrespective of bist_halt_on_error, bist_fail_counter signifies the number of errors encountered in that appropriate memory while performing Memory BIST. But in broadcast mode. The only indication we get is the appropriate memory id's, Memory BIST Passed/failed, we won't get any information on no of failures (bist_fail_counter) occurred during memory BIST for each memory.

Memory BIST Kick off can only be possible for any of the single memory within the group of memory, or in broadcast mode, all the memories are selected for Memory BIST one after the other. There is no option of selecting any intermediate sequence or any in between number of memories for Memory BIST in the current Implementation of Memory BIST.

Similarly, when bist_force_error is program while kicking off Memory BIST, it forces the state machine to insert errors while writing data into the memory. The consecutive read operation will land into an error. This bit is used for negative testing of Memory BIST State machine. All the signals of MBIST controller those are shown in Figure 7 are explain in the Table 4.

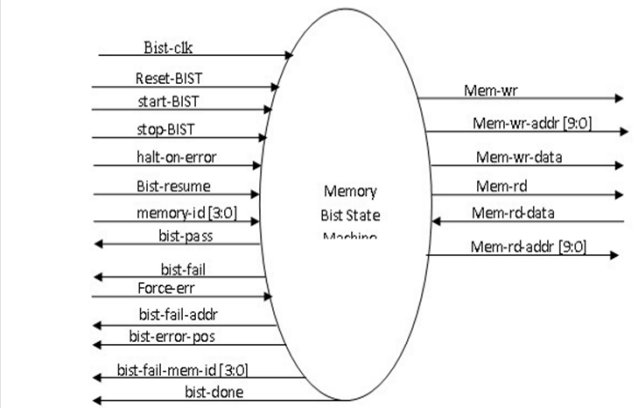

Figure 7. March 17N memory BIST signal diagram for fault diagnosis.

Table 4. Memory BIST controller

Signal	Direction	Width (bits)	Description
bist_clk	In	1	This is the Memory Bist clock input. All the internal state machines and hardware will get clocked with this clock.
bist_reset	In	1	Asynchronous reset; will reset the memory Bist state machine, and internal hardware.
bist_start	In	1	It will trigger the memory bist state machine. It will be reset by the state machine when bist_done is asserted.
bist_stop	In	1	It is required to stop memory BIST state machine synchronously and display the status Pass/Fail. When it activate SM assert bist_done on the next clock.
bist_halt_on_error	In	1	When this bit is programmed, Memory BIST state machine will halt itself in case of BIST failure and latch the error information (bist error addr and bist_error_pos) onto the appropriate output pins.
bist_resume	In	1	This Input is required to resume BIST state machine, in case when halt_on_error is programmed.
bist_memory_id	In	2	The input bit [1:0] selects the memory from a group of 4 memories for performing memory BIST operation.
bist_broadcast	In	1	This input selects all the memories one by one for performing Memory BIST Operation.
bist_mem_rd_data	In	32 (parameterized)	Memory Read data bus. The widths can be variable across the memories, this input has to be parameterized depending upon the width of the memories.
bist_force_error	In	1	Insert error during Memory BIST Write Operation.
bist_pass	Out	1	This output displays the successful operation of Memory BIST Operation.
bist_fail	Out	1	This output signifies memory BIST failure. If halt on error is programmed Bist state machine will halt on failure and will stop if bist_stop is programmed and continue if bist_resume is programmed.
bist_fail_addr	Out	12 (parameterized)	This fail address bus hold valid only when bist_fail is asserted. This fail address bus signifies the address of the memory where a comparison failure has occurred.
bist_error_pos	Out	5	The value present on this four bit error_pos bus is valid only when bist_fail is asserted by the bist state machine. This 4 bit signifies the error bit position within the byte read.
bist_fail_mem_id	Out	2	The value present on this four bit mem_id bus is valid only when bist_fail is asserted by the bist state machine. This 2 bit signifies the failed memory among a group of 4 memories.
bist_done	Out	1	This bit signifies the completion of Memory BIST Operation. Memory BIST pass/fail will be declared only when bist_done is high.
bist_mem_wr	Out	1	This is memory write enable signal. All the writes to the memory are only valid when this signal is high.
bist_mem_wr_addr	Out	12 (parameterized)	This is memory write address.
bist_mem_wr	Out	1	This is memory write enable signal. All the writes to the memory are only valid when this signal is high.
bist_mem_rd	Out	1	This is memory read enable signal. Data on the read data bus is valid one clock cycle later after bist_mem_rd is asserted.
bist_mem_rd_addr	Out	12 (parameterized)	This is memory read address.TD Data red from this address. Data read from this address.

4.3 Memory Block Diagram

To test the design memory block is designed by writing an HDL code in Verilog, the schematic shown in Figure 8 is the considerable block of memory connected externally. In which wr_en is a write enable signal to the memory to starts the writing process in the memory, wr_addr is a signal vector of 10 bits that can locate up to 2^10 memory locations:

Once the memory module is designed it is simulated using simulation tool [MODELSIM] to generate the waveform that shows in results. After thorough simulation, the memory module is synthesized using synthesis tool [Xilinx ISE] which generates the synthesis report.

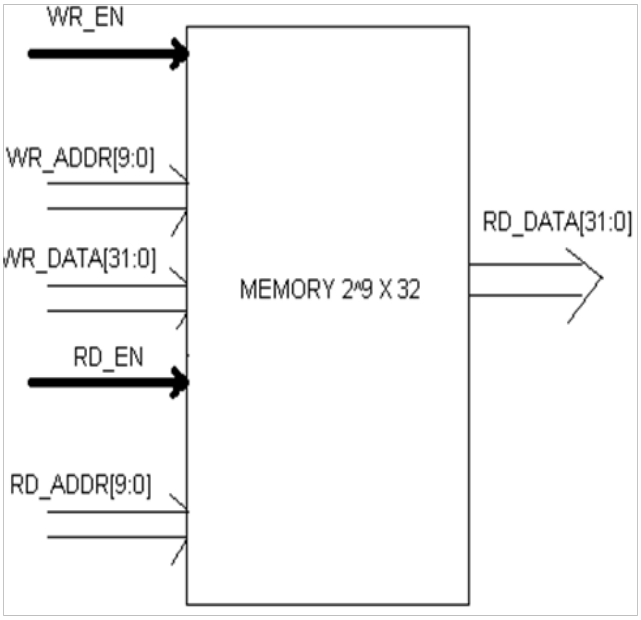

Figure 8. Memory Block.

5. Experimental Results

MBSIT verification mainly targets the functional correctness of the mbist controller. The simulation is based on March 17N algorithm.

5.1 MBIST Controller Working Mode Simulation

MBSIT controller simulation is carried out by mentor graphics tool ModelSimsimulator. Figure 9 is the simulation waveform of module mbist_top. The kick_off register is used as an input register under read/write mode. bist_start signal is used to kick off memory BIST controller. bist_resume is used to resume MBIST Controller,if fail occurs. And bist_stop is used to stop memory BIST operation. The bist_status register is used to store the result of the test, where bist_pass indicate MBIST test pass, bist_fail indicate MBIST test fail. bist_done indicate completion of the test. In this simulation writing /reading in various location of memory is shown in Figure 6. Four memory blocks with same data depth are tested by MBIST controller and test is serial. After the bist_done is set '1' test ends with fail/pass status, in the status register.

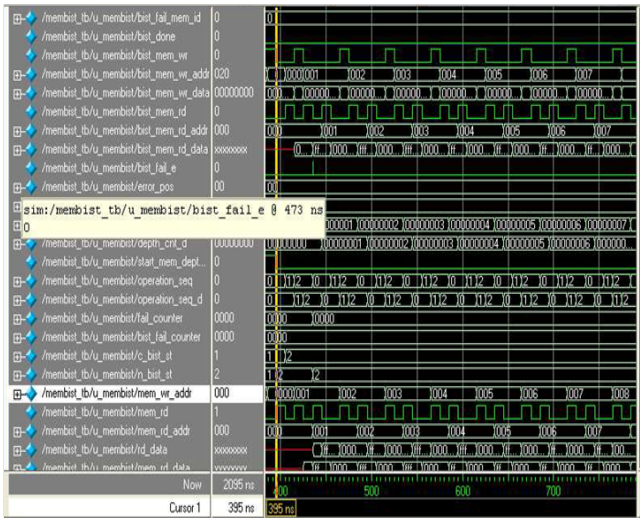

Figure 9. The Simulation waveform of module mbist_top.

5.2 Mbist Testing Period Analysis

Equation (1) gives the test period of March 17N
Cycle=17 x word (i)

5.3 MBIST Circuit Performance Analysis

Performance analysis based on the synthesis result. MBIST consumes up to 268940 kilobytes. This may add into total chip area[2] and then percentage of added area will be calculated. This leads up increase 1 or 2 percent area to the total chip area.

Synthesis of the design is carried out on Xilinx synthesis tool Xilinx 13.2 ISE[12]. The device family is selected for synthesis is Virtex 7 and Target Device is xc7vx330t-3-ffg1157. Results obtained by synthesis are tabulated in Table 5. It is shown in the table that design achieved 497.475 MHz of maximum frequency by utilizing only 158 Slice LUTs on 2.010ns minimum time period and Maximum combinational path delay is 1.175ns. Advanced synthesis report generated, which shows the digital logic

blocks generated in implementation of memory BIST Controller. The synthesis tool also generate net list. The logic blocks are shown in Figure 10 below indicates top level of design that internally consists of state machine and memory model.

Table 5. Experimental Results

Design	Area Slice LUTs	Time	Max Frequency	Maximum Combinational path delay
MBIST Controller	158	2.010ns	497.475 MHz	1.175ns

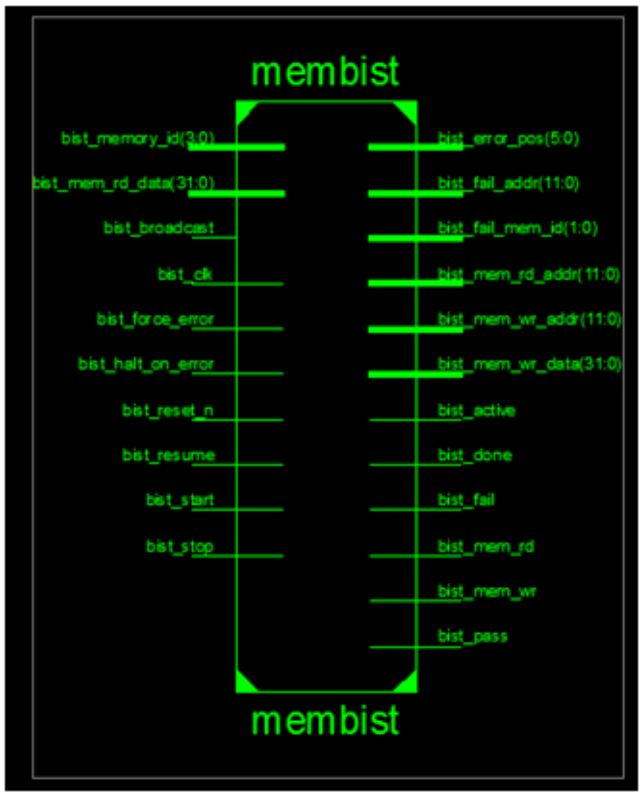

Figure 10. Synthesis Logic Block for MBIST Controller.

6. Conclusion and Future work

This paper implemented MBIST controller for SoC integration using Verilog HDL. The design includes MBIST principle, choice of algorithm, MBIST controller integration. By MBIST Controller reuse, the area of the chip saved. Working mode simulation, test time analysis is done. Simulation has done on Mentor graphics tool ModelSim simulator and synthesis is done on Xilinx tool Xilinx 13.2 ISE synthesizer. The design have achieved fast MBIST controller core for easy integration into SoC product.

As per the above discussed points the design can use to improve the chip yield. In the future, we can enhance the March algorithm for detecting the static and dynamic faults with very low area overhead. And we can add the fault repair strategy white integrating with SoC.

7. References

1. Acosta R, Nascentric EET. Verification Challenges of embedded memory devices; 2006 Aug 14. p. 1.
2. Bosio A, Dilillo L, Girard P, Pravossoudovitch S, Virazel A. Springer book- Advanced Test Methods for SRAMs. Effective Solutions for Dynamic Fault Detection in Nanoscaled Technologies; 2010. p. 1–19.
3. Sematech H. Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS); Taiwan; 2009 Dec. p. 70–6.
4. Zorian Y. Embedded infrastructure IP for SOC yield improvement. Virage Logic Fremont; USA. IEEE International Test Conference (ITC); 2002. p 709–12.
5. Stroud C. A Designer's Guide to Built-In Self-Test. Kluwer Academic Publishers. 2002; 1:429–38.
6. Li JF, Cheng KL, Wu CW. March-based RAM diagnosis algorithms for stuck-at and coupling faults. IEEE Test Conference Proceedings International; USA Baltimore: MD; 2001. p. 758–67.
7. Acharya GP, Rani MA. Survey of Test Strategies for System-On Chip and It's Embedded Memories. IEEE Recent Advances in Intelligent Computational Systems (RAICS); 2013 Dec 19-21. p. 199–204.
8. Wang CW, Cheng KL, Lee JN, Chou YF, Huang CT, Wu CW. Fault Pattern Oriented Defect Diagnosis for Memories. ITC International Test Conference IEEE; USA; 2003. p. 29–38.
9. Elavarasi R, SenthilKumar PK. An FPGA Based Regenerative Braking System of Electric Vehicle Driven by BLDC Motor. 2014 Nov; 7(7):173.
10. Aljumah A, Ahmed MA. Design of High speed Data Transfer Direct Memory Access Controller for System on Chip based Embedded Products. Journal of Applied Sciences. 2015 Jan; 15(3):576–81.
11. Cheng WT, Hill CJ, Kebichi O. MBIST Architect Process Guide. Mentor Graphics; Software Version; 2010 May. p. 1–19.
12. Xilin X. FPGAs: A Technical Overview for the First- Time User. Application Note: XAPP 097; (Version 1.3). Available from: http://www.xilinx.com