ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

# FPGA Implementation of Address Bit Generation Block & 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

***Abstract:*-- Memories are the most universal components today. Almost all system chips contain some type of embedded memory, such as ROM, SRAM, DRAM and flash memory. Testing of these memories is a challenging task as testing time, overhead area and cost of the test plays an important role during testing. The direct accessing of embedded memories is highly difficult. The testing of these memories done through a BIST (Built-in-self-test) Controller The Microcode based Memory BIST controller can be proven to be one of the best methods to test the memories. For this we are using March C/C+ algorithms, since March based tests are simple and possess good fault coverage. Microcode based Memory BIST consists of Program counter, Instruction decoder, Address generation block and Memory interface unit. This paper deals with the implementation of Address generation block, Memory interface unit for a Microcode based Memory BIST controller by using March C/C+ algorithms. Address generation block gives the address of the memory to the Memory interface unit and the Memory interface unit of MCMBIST controller acts as an interface between other blocks of MCMBST Controller and Memory under test. It has Address register, Data register, Control register, Multiplexers and Comparator. Address register takes the address from Address generation block, stores it for the fault diagnosis. Data register takes the data from Instruction decoder, processes and stores it. Control register takes read or write information from Instruction decoder and stores in it. The information from all these registers passed through the Multiplexers and to the memory under test for testing of all locations of memory. March C/C+ algorithms applied to Memory which under test. These blocks are designed by using Verilog HDL code, simulated by NC Verilog compiler and synthesized by RTL Compiler. These blocks are integrated with program counter and instruction decoder to form a Microcode based Memory BIST Controller.**

**Keywords: - BIST, March C/C++, Micro Code, Verilog, HDL.**

## 1. INTRODUCTION

Today, there are various urban areas developing all around the globe, with this development of urban areas, the Memories are the most universal component today. Almost all system chips contain some type of Embedded memory, such as ROM, SRAM, DRAM, and flash memory. In the Embedded domain, Embedded RAMs of the StrongArmSA110 occupy 90% of the total area. The paper ion is, by 2010, memory will represent more than 90% of the chip area in average SOC environment. With the advent of deep-submicron VLSI technology, the memory density and capacity is growing. The clock frequency is never higher. The dominant use of Embedded memory cores along with emerging new architectures and technologies make providing a low cost test solution for these on-chip memories a very challenging task. Built-in self-test (BIST) [7] has been proven to be one of the most cost-effective and widely used solutions for memory testing for the following reasons. 1)No external test equipment; 2. Reduced development efforts3. Tests can run at circuit speed to yield a more realistic test time 4. On-chip test pattern generation to provide higher controllability and observability 5. On-chip response analysis 6. Test can be on-line or off-line 7. Adaptability to engineering changes 8. Easier burn-in support.

The main aim of this paper is to implement the Address Generation Block and Memory Interface Unit for Microcode Based Memory BIST Controller and to integrate these blocks with other two blocks (Program Counter & Instruction Decoder) to form the Microcode Based MBIST Controller. A Microcode-based Memory BIST Controller features a set of predefined instructions, or Microcode, which is used to write the selected Test Algorithms. The written tests are loaded in the memory BIST controller. This Microcode-based type of memory BIST allows changes in the selected test algorithm with no impact on the hardware of the controller. Microcode Based Memory BIST Controller which generates the test patterns (data, address and control signals) by using the March C & March C+ algorithms. These test patterns are applied to memory, which is under test, through a memory interface unit. This Memory BIST Controller is inserted into the memory model, which describes the memory, which is under test. Finally this controller gives the pass/fail

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

information of the MUT (memory under test). The chapter I presents the motivation, objective of low Memory Design and BIST, Chapter II presents Basic Theory of BIST Chapter III Micro Code Based MBIST Controller Chapter IV represents Simulation results, Chapter V presents Conclusion and Future scope

## 2. BASIC THEORY OF BIST:

### *2.1 What is SoC?*

SoC (system-on-a-chip} is a complex VLSI device generally designed to perform a specific task and contains many component modules and a number of memory units besides a high performance Microprocessor. Testing these SoCs is becoming an increasing challenge as these devices are becoming more and more complex. A SoC design is typically built block by block; efficient testing is also best done block by block. Today, designers can install a specialized, pre-designed, configurable Embedded system to test and debug each block. Using such an Embedded system, a designer can specify the test speed, fault coverage, diagnostic options, and test length for testing any random logic block.

### *2.2 Embedded Memory:*

Memory units on SOC devices are Embedded memories. Embedded memory is any non-stand-alone memory. It is an integrated on-chip memory that supports the logic core to accomplish intended functions. High-performance Embedded memory [4] is a key component in VLSI because of its high-speed and wide bus-width capability, which eliminates inter-chip communication. Several advantages of using embedded memories are provided below. They include reduced number of chips, reduced pin count, multi-port memories, less board space requirements, faster response with memory Embedded on-chip, dedicated architecture, memory capacity specific for an application, reduced power consumption, and greater cost effectiveness at the system level. The main disadvantages of Embedded memories are that they are generally larger in size and are more complex to design and manufacture. Additionally, a trade-off must often be found between design and technology since the optimized technology for a memory cell is not the same as that for Embedded logic devices

### *2.3 Built in self Test. (MBIST):*

While embedded memories are useful for the SoCs the main difficulty with their use is their testing. It is very difficult to provide an external access to these for testing them. The most common method adopted for testing these units is to provide an on chip mechanism for their testing generally in the form of built in self test.

#### *2.3.1 Types of BISTs*:

Built-in self-test (BIST) is a design technique in which parts of a circuit are used to test the circuit itself. BIST represents a merger of the concepts of built-in test and self test, and has come to be synonymous with these terms [3]. There are two types of BIST 1)On-line test 2)Off-line test. On-line BIST has tests implemented on-chip. It has shorter test time but an area overhead of one to three percent. Off-line BIST, on the other hand has tests implemented off-chip. It has longer test time but no area overhead. On-line BIST can further be classified into three subgroups: Concurrent test on-Concurrent BIST Transparent BIST.

#### *2.3.1.1 Concurrent BIST:*

Concurrent BIST is a memory test mechanism where the memory can be tested concurrently with normal system operation. Thus, it has instant error detection and possible correction, but all faults will be detected within the restrictions of the method used. There is also certain hardware overhead associated with this scheme. For example, we need logic to write out, read in, and store redundant information generated during the test process. There will also incur certain performance penalty upon every memory access.

#### *2.3.1.2 Non-Concurrent BIST:*

Non-Concurrent BIST is test mechanism that requires interruption of the normal system function in order to perform tests; usually a special test mode is required. The advantage is there is no need to preserve the data yields certain space savings. The disadvantage is the circuit cannot detect faults that are not covered by the fault models used. Transparent BIST scheme is very similar to the Non-Concurrent scheme except the memory contents are preserved.

### *2.4 Types of fault models:*

Generally the memory faults [1] are due to Open connections and Shorted connections Storage cells stuck at 0/1 Memories fail in a number of different ways. The three main parts. 1) Address decoder logic2) Memory cell array 3) Read/write logic, can each have flaws that cause the device to fail? Memory testing, while similar to random logic testing, focuses on testing for these memory-specific failures. The types of memory faults Stuck at faults Transition faults coupling faults Neighborhood pattern sensitive faults

#### *2.4.1 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 behavior, where a signal or cell appears to be tied to power

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

(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 location

***2.4.2 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 behavior from that point on. However, a stuck-at-0 test might not detect this fault if the cell was at 1 originally.

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

***2.4.4 Inversion coupling Faults (CFins):***
Inversion coupling faults commonly referred to as CFins; occur when one cell's transition causes inversion of another cell's value. For example, a 0->1 transition in cell_n causes the value in cell_m to invert its state.

***2.4.5 Idempotent coupling faults (CFids)***
Idempotent coupling faults, commonly referred to as CFids, occur when one cell's transition forces a particular value onto another cell. For example, a 0 >1 transition in one cell causes the value of 2nd cell to change to 1 if the previous value was 0. However, if the previous value was 1, the cell remains a 1.These can occur when a short, or bridge, exists between two or more cells or signals. In this case, a particular logic value triggers the faulty behavior, rather than a transition. Bridging faults fall into either the AND bridging fault (ABF) or the OR bridging fault (OBF) subcategories. ABFs exhibit AND gate behavior; that is, the bridge has a 1 value only when all the connected cells or signals have a 1 value. OBFs exhibit OR gate behavior; that is, the bridge has a 1 value when any of the connected cells or signals have a 1 value.

***2.4.6 State coupling faults (SCF):***
State coupling faults, abbreviated as SCFs, occur when a certain state in one cell causes another specific state in another cell. For example, a 0 value in cell i cause a 1 value in cell j.

***2.4.7 Neighborhood pattern sensitive faults (NPSF):***
State coupling faults, abbreviated as SCFs, occur when a certain state in one cell causes another specific state in another cell. For example, a 0 value in cell i cause a 1 value in cell j.

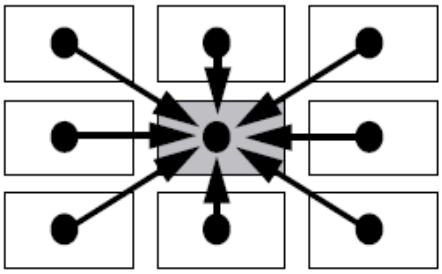


***Fig: State diagram of memory location with NPS faults***

An active fault occurs when, given a certain pattern of neighboring cells, one cell value change causes another cell value to change. A passive fault occurs when a certain pattern of neighboring cells cause one cell value to remain fixed. A static fault occurs when a certain pattern of neighboring cells forces another cell to a certain state. The complexity of neighborhood pattern sensitive faults requires a variety of different detection methods. The test algorithms available for detection of this fault type do not readily lend themselves to BIST, as they require significant area overhead and produce very long test sets. Currently, no single, commercially available tool supports algorithms to detect neighborhood pattern sensitive faults.

***2.4.8 Address decoder faults***
These address decoder faults are due to faulty cell access. There are two types of address decoder faults. One is inaccessible cell faults i.e. no cell will be accessed with the current address and other are multiple cell accesses i.e. more than one cell is selected with the current address.

***2.5 Test algorithms:***
A test algorithm is a test procedure which uses a finite sequence of test elements for testing the memory and identifying and locating defects [2]. A test element contains a number of memory operations (access commands), data pattern (background) specified for the read operation, address (sequence) specified for the read and write operations. These test algorithms can be divided into two groups1) Classical tests 2) March-based tests

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

***2.5.1 Classical Test Algorithms:***

Classical test algorithms are either simple, fast but have poor fault coverage, such as Zero-one, Checkerboard; or have good fault coverage but complex and slow, such as Walking, GALPAT, Sliding Diagonal, Butterfly, MOVI, and etc.. Due to these imbalanced conflicting traits, the popularity of these algorithms is decreasing.

***2.5.1.1 GALPAT and Walking 1/0:***

The Gal pat (Galloping pattern) and Walking 1/0 algorithms consist of similar operations. First, they fill the memory with 0s or 1s, except for the base cell, which contains a 1 or 0, respectively. During the test, the base cell walks through the memory. The difference between GALPAT and Walking1/0 is how they read the base cell. Walking 1/0 reads all cells after each step, with the base cell last. GALPAT also reads all cells, but reads the base cell after each one.

***2.5.1.2 Checkerboard:***

The checkerboard algorithm detects stuck-at faults for memory cells and adjacent cell shorts, providing previous tests prove the address decoding circuitry is fault free. The algorithm divides the cells into two groups (cells_1 and cells_2), such that every neighboring cell is in a different group.

Below figure shows how this process forms the memory cells into a checkerboard pattern. The algorithm then writes (and reads) 0s into all cells in the cells_1 group and 1s into all cells in the cells_2 group. The algorithm repeats this process by writing (reading) 1s into cells_1 cells and 0s into cells_2 cells.

***2.5.2 March-based Test Algorithms***

A March-based test algorithm is a finite sequence of March elements. A March element is specified by an address order and a number of reads and writes. Since March-based tests are all simple and possess good fault coverage, they are the dominant test algorithms implemented in most modern memory BIST.

The algorithms in most common use are the March tests. March tests generate patterns that “march” up and down the memory addresses; writing values to and reading values from know locations. These algorithms can retrieve the proper parameters from the memory model, automatically determining the memory size and word length. Examples of some March-based tests are MATS, MATS+, Marching 1/0, March C-, March Y, March A, March B, and etc..

***2.5.2.1 ATS and Modified ATS (MATS):***

These algorithms detect unlinked stuck-at faults. Knaizuk in 1977 developed the Algorithm Test Sequence (ATS), Nair in 1979 improved it and renamed it MATS. The MATS algorithm provides the shortest march test for unlinked stuck-at faults. Abadir in 1983 developed The MATS+ algorithm, which enhances the MATS algorithm by making no assumptions on the memory technology in use.

***2.5.2.2 March A and March B:***

The march A and march B algorithms cover some linked faults, such as idempotent linked faults, transition faults linked with idempotent coupling faults, and inverting faults coupled with idempotent coupling faults.

***2.5.2.3 March C:***

The March C algorithm and its modifications are now the most popular algorithms for memory testing. This algorithm, which consists of 11 operations (11n), writes and reads words of 0s, followed by writing/reading words of 1s, in both descending and ascending address spaces. Specifically, the algorithm consists of the following steps: Write 0s to all locations starting at the lowest address (initialization),Read 0 at lowest address, write 1 at lowest address, repeating this series of operations until reaching the highest address, Read 1 at lowest address, write 0 at lowest address, repeating this series of operations until reaching the highest address , Read 0 from the lowest address to the highest address, Read 0 at highest address, write 1 at highest address, repeating this series of operations until reaching the lowest address, Read 1 at highest address, write 0 at highest address, repeating this series of operations until reaching the lowest address. The March C Algorithm detects the faults such as stuck-at , transition, coupling - unlinked idempotent and inversion, and other coupling faults o bit-oriented address.,

| *S.No* | *Addressing order* | *Operations* |
|---|---|---|
| 1. | Incrementing | Write 0s to all locations(Initialization) |
| 2. | Incrementing | Read 0s and write 1s to all locations |
| 3. | Incrementing | Read 1s and write 0s to all locations |
| 4. | Decrementing | Read 0s and write 1s |
| 5. | Decrementing | Read 1s and write 0s |
| 6. | Decrementing | Read 0s |

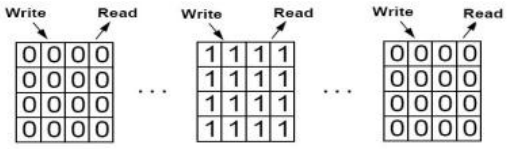
***Table: March C algorithm***

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

***2.5.2.5 March C+/March2***

The March C+ algorithm is derived from the modified March C algorithm described by Rob Dekker and Frans Beenker in 1990. The following table shows the modified March C algorithm, which modifies the original March C algorithm by adding an extra read operation after each stage of the march. While increasing the algorithm from 10n to 13n, this extra read allows additional fault detection, most notably, stuck-open faults for all types of RAM




***Fi g: March C+ algorithm operations***

***2.5.2.4 March C:***

The following figure shows the March C algorithm. The March C- algorithm modifies the March C algorithm by eliminating the redundant Read 0 operation between the ascending and descending address operations. Removing this operation reduces the algorithm from 11n to 10n, without sacrificing any fault coverage. The March C- algorithm detects the same faults as March C.1. Write 0s to all locations starting from the lowest address (Initialization)2. Read 0 at lowest address, write 1 at lowest address, repeating this series of operations until reaching the highest address.3. Read 1 at lowest address, write 0 at lowest address, repeating this series of operations until reaching the highest address.4. Read 0 at highest address, write 1 at highest address, repeating this series of until Operations reaching the highest address. 5. Read 1 at highest address, write 0 at highest address, repeating this series of operation operations reaching the highest address. 6. Read 0s from high address to low.

| *S.No* | *Addressing order* | *Operations* |
|---|---|---|
| 1. | Incrementing | Write 0s to all locations(Initialization) |
| 2. | Incrementing | Read 0s,write 1s and read 1s to all locations |
| 3. | Incrementing | Read 1s,write 0s and read 0s to all locations |
| 4. | Decrementing | Read 0s,write 1s and read 1s to all |
| | | locations |
| 5. | Decrementing | Read 1s,write 0s and read 0s to all locations |

***Table: Modified March C Algorithm:***

Table shows the March C+ (or March2) algorithm which further modifies the modified March C algorithm by adding one more read operation at the end of the final stage. This increases the algorithm from 13n to 14n.

***Table : March C+ algorithm***

| **S.No** | **Addressing order** | **Operations** |
|---|---|---|
| 1. | Incrementing | Write 0s to all locations(Initialization) |
| 2. | Incrementing | Read 0s,write 1s and read 1s to all locations |
| 3. | Incrementing | Read 1s,write 0s and read 0s to all locations |
| 4. | Decrementing | Read 0s,write 1s and read 1s to all locations |
| 5. | Decrementing | Read 1s,write 0s and read 0s to all locations |

The March C+ algorithm is detects the same faults as March C and also stuck open faults, timing faults, if the test is performed at speed.

***2.5.2.6 Varied Data background:***

The above mentioned algorithms use 0s and 1s as background data. We can also use 0101… or 1010… as background data.

The varied background data for March C+ algorithm is as follows.

1. Write 0101 to all locations starting at address 0 up to address 3.
2. Read 0101 at address 0, write 1010 at address 0, read 1010 at address 0, repeating this series of operations in addresses 1, 2, and 3.
3. Read 1010 at address 0, write 0101 at address 0, read 0101 at address 0, repeating this series of operations in addresses 1, 2, and 3.
4. Repeat steps 2 and 3, but this time begin at address 3 working down to address 0.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

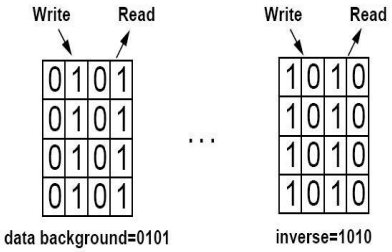



*Fig : Varied background data operation*

### *2.6 Why March C & March C+ algorithms?*

Among the all March based algorithms March C has 11 operations (11n) & March C+ has 14 operations (14n) and more defect coverage. Due to less no of operations it requires less testing time to test the memory, under test. And it detects more no of faults because of its more defect coverage. That's why we have selected the March C & March C+ algorithms in the implementation of Microcode -based memory BIST controller.

### *2. 2.7 various types of memory BIST Controllers*

To test on board memories on an SOC, for which a direct external access is not available, special circuitry MBIST (Memory Built-in Self Test) is provided on chip along with the memory. This circuitry tests the Embedded memory and gives the information that whether all memory locations within the memory are working correctly or not.

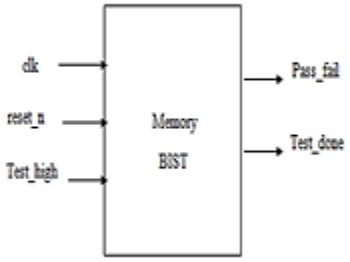



*Fig: Block Diagram of Memory of BIST*

The inputs for the memory BIST circuit are clk, test_high, reset_n and the outputs are pass-fail, test_done. Initially the power to the circuit is switched on. By applying the reset_n signal to the program counter, instruction decoder and address generation block. Then all the registers, program counter and address generation block are initialized. When the signal test high is asserted then the BIST controller automatically selects the March C or March C+ algorithm and the circuit enters into test mode. And passl_fail signal will be asserted if the corresponding memory location is perfect otherwise the signal will be de-asserted. After completion of March algorithm then test done signal will be asserted indicating that the test is completed.

This section describes different implementation schemes for Memory BIST. A memory BIST unit consists of a controller to control the flow of test sequences and other components to generate the necessary test control and data. Based on the type of Controller used the memory BIST classified into three different types. These are 1. Microcode-based MBIST 2. Processor-based MBIST3. Hardwired-based MBIST

### *2.7.1 Microcode-based MBIST Controller:*

A Microcode-based memory BIST Controller features a set of predefined instructions, or microcode, which is used to write the selected test algorithms. The written tests are loaded in the memory BIST controller. This microcode-based type of memory BIST allows changes in the selected test algorithm with no impact on the hardware of the controller. This flexibility, however, may come with the cost of higher logic overhead for the controller.

The below illustrated a Microcode-based memory BIST Controller which consists of program counter, instruction decoder, address generation block, memory interface unit.

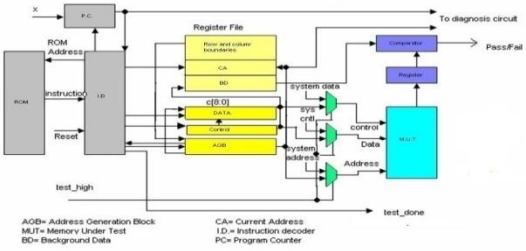



*Fig: Microcode-based MBIST controller*

### *2.7.2 Processor-based MBIST Controller:*

With the introduction of deep-submicron VLSI technology, core-based SOC design is attracting an increasing attention. On an SOC, memories are the most universal cores; almost all system chips contain some type of Embedded memory, such as ROM, SRAM, DRAM, and flash memory. To provide a low cost test solution for these on-chip memory cores is a challenging task.

Conventional hardwired-based memory BIST approach is one possible solution and its advantage are short test time and small area overhead. However, sometimes it is not feasible to have one BIST circuit for each memory core. For instances, a typical ASIC or SOC may have tens of SRAM cores with

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

different sizes and configurations. If each memory core on chip requires a BIST circuit, then the area and test pin overhead will be unacceptably high. Therefore, a new type of memory BIST scheme which utilizes an on-chip microprocessor to test the memory cores is processor-based BIST. The processor-based memory BIST is done by executing an assembly-language program in the on-chip microprocessor to generate test patterns including the address sequence, data patterns, and control signals. The memory outputs are then compared with the expected correct data. The advantage of such a scheme is that it is highly flexible because various test algorithms can be realized by simply modifying the assembly programs run on the micro processor and it is also easy to support testing of multiple memory cores. However, the drawback is a much longer test time.
A modified processor-based scheme was shown in below figure. It utilizes a processor programmable BIST circuit to realize a test algorithm using pre-defined test elements. The approach is a combination of on-chip processor-based BIST and the hardwired-based BIST. The proposed scheme has the following advantages.

- The test time is short due to dedicated BIST core.
- The flexibility of processor-based BIST is maintained.
- Multiple memory cores can be supported without multiple BIST cores, multiple sets of external test pins, or complicated routing.

BIST core is inserted between the CPU core and the on chip bus, which also connects the memory cores. In normal operation mode, the CPU transparently accesses the system bus with slight time overhead introduced by the multiplexers. The overhead can be minimized by careful design of the multiplexers which can be integrated with the bus drivers. In memory BIST mode, the BIST circuitry takes over the control of the on-chip bus. It executes certain test algorithm programmed by the CPU and generates the addresses, input data, and control signals of the memory core. It also compares the memory output response with the expected correct data. In order to allow these two different modes, several multiplexers are used to multiplex the address bus, data input bus, data output bus, and control bus between the CPU core and the BIST circuitry.

### *2.7.3 Hardwired-based MBIST Controller:*

A hardwired-based controller is a hardware realization of a selected memory test algorithm, usually in the form of a Finite State Machine (FSM). This type of memory BIST architecture has optimum logic overhead, however, lacks the flexibility to accommodate any changes in the selected memory test algorithm. This results in re-design and re-implementation of the hardwired-based memory BIST for any minor changes in the selected memory test algorithm. Although it is the oldest memory BIST scheme amongst the three, hardwired-based BIST is still much in use and techniques have been kept developing. The block diagram of hardwired based BIST controller is shown in below figure.
It consists of three major blocks.

- BIST controller unit.
- Access unit.
- Memory interfacing unit.

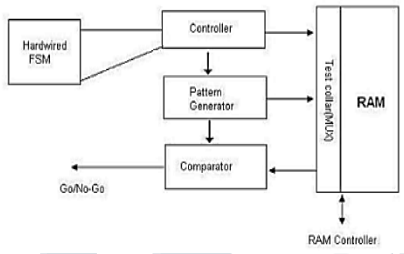



***Fig: Hardwired BIST block diagram***

### *2.8 Trade-offs:*

The below table gives the various design trade-offs among the three implementation schemes.

***Table 2.6: Tradeoffs among the three controllers***

| Scheme | Test Time | Area OH | Routing OH | Flexibility |
|---|---|---|---|---|
| Hardwired | Short | Low | High | Zero |
| Micro Code | Average | High | Low | L:ow |
| Processor | Long | Zero | Zero | High |

The four evaluation metrics used are:1)Test time2)Area 3)overhead 4)Routing overhead Flexibility. The flexibility is expressed in terms of programmability of algorithms and adaptability to engineering changes. As shown in the table, the Microcode based MBIST Controller is compact, low routing overhead. On the opposite end of spectrum, the Processor-based MBIST is the most flexible, zero area or routing overhead, but incur longer test time. And for Hardwired MBIST Flexibility is less, Shorter test time

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

## 3. MICROCODE BASED MBIST CONTROLLER:

A Microcode-based MBIST Controller [4] features a set of predefined instructions, or microcode, which is used to write the selected test algorithms the written tests are loaded into the Memory BIST controller. This type of controller allows the changes in the selected test algorithm with no impact on the hardware of the controller. This flexibility however costs higher logic overhead. BIST controller is commonly used for Embedded memory testing. Rapidly increasing circuit complexity along with product quality has made BIST a viable test solution for memory processors. Most SOC implementations use a variety of Embedded cores and memory arrays which exist in current microprocessors. With limited accessibility of Embedded cores & advances in memory technology & shrinking geometries causing memory testing to be a constant challenge. Memory BIST uses March algorithms to test all addresses, data locations.

### *3.1 Specifications for the Microcode based MBIST controller:*

The specifications that have been provided for the Microcode based MBIST controller is as follow 1.Clock frequency - 340 KHz2.Size of the memory 256x8 bits.3.Number of memories connected to the BIST controller -1.To implement these specifications the following block diagram as been arrived at.

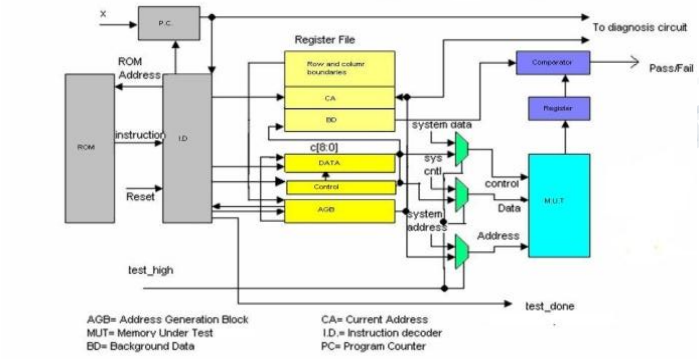


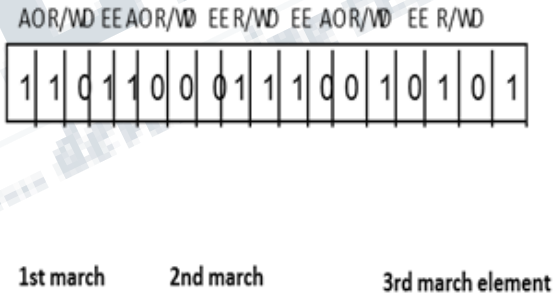

***Fig: Microcode based MBIST controller***

#### *3.1.2.1 Description of data flow diagram for Microcode based MBIST controller:*

When the test high signal is enabled then MBIST architecture will enter into test mode otherwise it will perform normal operation In the test mode, first we set the reset pin to zero then all registers and program counter values are initialized. Row and column boundaries are loaded into the register file. Now the reset pin will be changed to '1'. Then PC value is incremented to the address of first March element in an algorithm. Instruction decoder will send the request signal to program counter and fetches the address of a March element in ROM. This instruction decoder (ID) will fetch the microcode of that March element and decodes it. Depending on the microcode ID generates addressing order to Address Generation Block (AGB), Control and Data signals. Depending on the addressing order Address Generation Block will generate the address of memory to which the data has to write. The above generated address and data are stored in register file. Memory operation decoder (data and control blocks) and address generator are synchronized so that the data signal is sent to memory after generation of address. The data stored in the register file is consists of 9 bits where the MSB bit indicates whether the data is supplied to comparator or not. If the MSB is '1', it means write operation is going on , so o need to supply data to comparator, if it is '0' then it supplies the data to comparator. A register is used between memory and comparator to store the read data from the memory. Comparator will compare the data to state whether the memory is perfect or not

#### *3.1.3 Format of microcode:*

The following microcode is an example of MATS+ algorithm (w0);↑(r0,w1);↓(r1,w0) ↕;




Generally a March element consists of 4 bits. In that first bit represents Addressing order,2nd and 3rd bit represents memory operations and last bit represents End of Element(EE). If EE =0, it means there some more operations left in that March element. Those operations don't require addressing order again.

***Table: Addressing order***

| AO | Addressing Order |
|---|---|
| 1 | Incrementing Order |
| 0 | Decrementing Order |

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

***Table: Memory Operations***

| Operation | Representation |
|---|---|
| Read 0 | 00 |
| Read1 | 01 |
| Write0 | 10 |
| Write1 | 11 |

***3.2 Microcode Generation***:

In Microcode based MBIST controller, we are using both March C & March C+ algorithms. March C algorithm consists of seven March elements and March C+ algorithm consists of six March elements.

***3.2.1 March C algorithm:***

{↕(w0) ;↑(r0,w1); ↑(r1,w0) ;↕(r0); ↓(r0,w1);↓ (w1,r0); ↕(r0)}

| S no | Addressing order | Operations |
|---|---|---|
| 1 | Incrementing | Write 0s to all locations(Initialization) |
| 2 | Incrementing | Read 0s and write 1s to all locations |
| 3 | Incrementing | Read 1s and write 0s to all locations |
| 4 | Incrementing | Read 0s |
| 5 | Decrementing | Read 0s and write 1s |
| 6 | Decrementing | Read 1s and write 0s |
| 7 | Decrementing | Read 0s |

***3.2.2 March C+ algorithm:***

{↕(w0);↑(r0,w1,r1);↓(r1,w0,r0);↓(r0,w1,r1);↓(r1,w0,r0);↓(r0) ;}

Microcode [0] = Address order

Microcode [1] = Control Signal (Read/Write

Microcode [2] = Data register (0/1)

Microcode [3] = End of element

Microcode [4] = Control Signal (Read/Write)

Microcode [5] = Data register (0/1)

Microcode [6] = End of element

Microcode [7] = Control Signal(Read/Write)

Microcode [8] = Data register (0/1)

Microcode [9] = End of Element

| S no | Addressing order | Operations |
|---|---|---|
| 1 | Incrementing | Write 0s to all locations(Initialization) |
| 2 | Incrementing | Read 0s and write 1s to all locations |
| 3 | Incrementing | Read 1s and write 0s to all locations |
| 4 | Incrementing | Read 0s |
| 5 | Decrementing | Read 0s and write 1s |
| 6 | Decrementing | Read 1s and write 0s |
| 7 | Decrementing | Read 0s |

Depending upon format of microcode (which is explained in 3.1.3), the microcode generated for March C & March C+ elements are as follows. The microcode will have the address order, control signal(R/W Signal), Data register, End of element

***3.2.3 Microcode generation for March C & March C+ elements:***

The Microcode for the March C & March C+ elements as shown and corresponding march element address given to the March elements.

| Address[3:0] | March Element | Microcode[9:0] |
|---|---|---|
| 0000 | (↑W0 | 10'b0000001011 |
| 0001 | (↑R0,W1 | 10'b0001110001 |
| 0010 | (↑R1,W0 | 10'b0001010101 |
| 0011 | (↑R0 | 10'b0000001001 |
| 0100 | (↓R0,W1 | 10'b0001110000 |

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

| | | |
|---|---|---|
| 0101 | (↧R1,W0 | 10'b0001010100 |
| 0110 | (↧R0 | 10'b0000001000 |
| 0111 | (↥R0,W1,R1 | 10'b1100110001 |
| 1000 | (↥R1,W0,R0 | 10'b1000010101 |
| 1001 | (↧R0,W1,R1 | 10'b1100110000 |
| 1010 | (↧R1,W0,R0 | 10'b1000010100 |

The Microcode based MBIST controller consists of four blocks. They are:

- Program counter
- Address generation block
- Memory Interface Unit
- Instruction decoder

## 4. SIMULATION RESULTS:

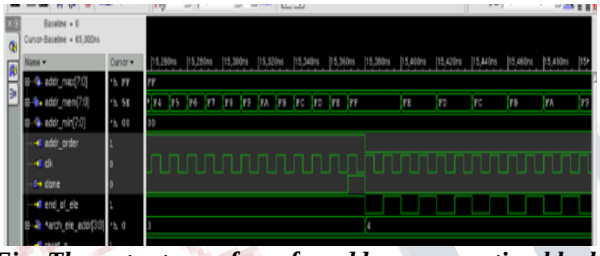


***Fig: The output waveform for address generation block (March C algorithm)***

***Explanation of output waveform for address generation block (March C):***

These Simulation result for the Address generation block for March C algorithm is shown above.Addr_max, addr_min , address order are the inputs to the address generation block. Addr_max, addr_min gives the address boundaries for the memory. Addr_mem selects the particular address location. In that location march elemental operations are done.

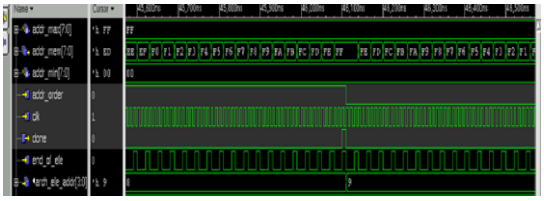


***Fig: The output waveform for address generation block (March C+)***

***Explanation of output waveform for address generation block (March C+)***

The simulation result for the Address generation block for March C+ algorithm is shown above. Addr_mem, addr_min , address order are the inputs to the address generation block. Addr_mem, addr_min gives the address boundaries for the memory. Addr_mem selects the particular address location. In that location march elemental operations are done.

***Fig : The output waveform for address registers***

***Explanation of output waveform for register (MIU):***

The simulation result for memory interface unit registers is shown above. The inputs to these MIU registers are Clock reset,address_mem,data register , read write signals and the outputs are test_addr, test_cntrl, test_data. The address is coming from the address generation block, data and control signals are coming from the instruction decoder block. The register outputs are given as inputs to the comparator.

***Fig: The output waveform for muxes (MIU)***

***Explanation of output waveform for muxes (MIU):***

The simulation result for memory interface unit muxes is shown above. The inputs to these MIU muxes are test_addr, test_cntrl, test_data , test_high and the outputs are addr_out, data_out, cntrl_out. When test_high='1' the addr_out selects the test_addr, data_out selects the test_data, cntrl_out selects the test_cntrl signals respectively.

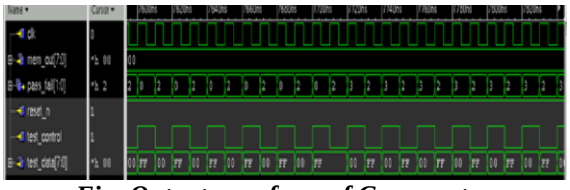


***Fig: Output waveform of Comparator***

Explanation of output waveform for comparator (MIU):

The simulation result for memory interface unit comparator is shown above. The inputs to these MIU comparator are

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

Clock reset, mem_out,, test_cntrl, test_data. When test_cntrl='1',the pass/fail signal<=2.when test_cntrl=0 and test_data not equal to mem_out then the pass/ fail =3. When test_cntrl=0 and test_data equal to mem_out then the pass/ fail =0.

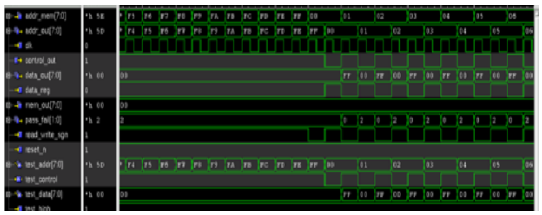


***Fig: output waveform of Memory interface unit***

***Explanation of output waveform for Memory interface unit:***

Memory interface unit has addr_mem, read_write, data, clock, reset and mem_out as inputs. The outputs to this unit are addr_out, data_out, cntrl_out and pass-fail. According to the addr_mem, read_write, data signals given at the input, in the next clock cycles these given inputs comes as outputs. According to this, output is compared, pass-fail information is generated.

***Explanation of output waveform for program counter***

The inputs to the program counter are clock, reset, algorithm and instruction decoder signal and the output is March element address. When the reset pin is zero the program counter will be initialized to zero. When the reset pin becomes one it checks for the algorithm (March C or March C+). For algorithm='0' program counter value is incremented to the address of the first March element in March C algorithm. Now it checks for the ID signal. If it is '1' then program counter value is sent to the instruction decoder. Later instruction decoder signal becomes zero. Similarly For algorithm='1' program counter value is incremented to the address of the first March element in March C+ algorithm. Now it checks for the ID signal. If it is '1' then program counter value is sent to the instruction decoder. Later instruction decoder signal becomes zero.

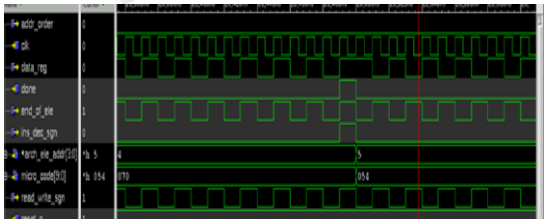


***Fig: The output waveform for instruction decoder (March C algorithm)***

***Explanation of output waveform for instruction decoder (March C algorithm):***

The simulation result for the instruction decoder for March c algorithm is shown above. The inputs to the instruction decoder are clock,reset,microcode,march element address and done signals and the outputs are address order, control signal, data register and end of element. The results shown here are for March element addresses ('0100' & '0101') which consists of

March element address

0100 ( ↑R0, W1)

0101 ( ↓R1, W0)

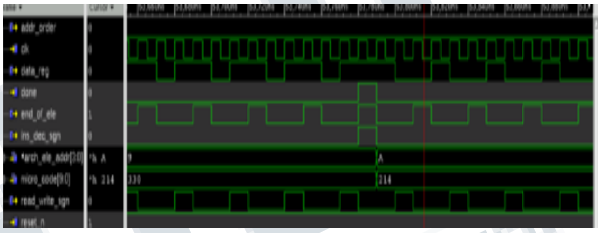


***Fig: The output waveform for instruction decoder (March C+ algorithm)***

***Explanation of output waveform for instruction decoder (March C+ algorithm):***

The simulation result for the instruction decoder for March c+ algorithm is shown above. The inputs to the instruction decoder are clock,reset,microcode,march element address and done signals and the outputs are address order, control signal, data register and end of element. The results shown here are for March element addresses ('1001' & '1010') which consists of March element address

1001 ( ↓ R0, W1, R1)

1010 ( ↓ R1, W0, R0)

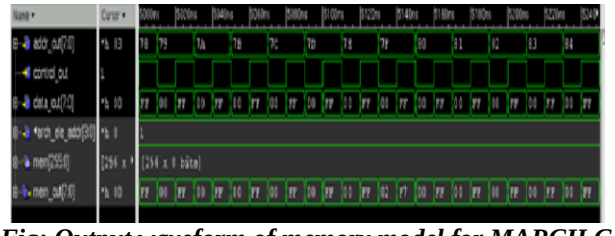


***Fig: Output waveform of memory model for MARCH C Algorithm***

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

***Explanation of output waveform for memory model (MARCH C Algorithm)***

Memory model is a unit which is used to test the memory In this memory model we are applying inputs from the multiplexers such as control out (read/write signal), address out, data out. And this model gives output as memory out In this unit we are forcing some wrong data in particular location for getting the fail signal. Now the memory model output is given to the comparator for comparing the back ground data (original data) and memory out .if both are matching it gives the pass signal otherwise it gives fail signal

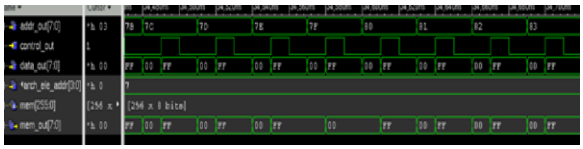


***Fig: Output waveform of memory model for MARCH C+ Algorithm***

***Explanation of output waveform for memory model (MARCH C+ Algorithm):***

Memory model is a unit which is used to test the memory In this memory model we are applying inputs from the multiplexers such as control out (read/write signal), address out, data out. And this model gives output as memory out In this unit we are forcing some wrong data in particular location for getting the fail signal. Now the memory model output is given to the comparator for comparing the back ground data (original data) and memory out .if both are matching it gives the pass signal otherwise it gives fail signal

***Fig: output waveform of MCMBIST Controller for MARCH C Algorithm***

***Explanation of output waveform for MCMBIST controller (March C) :***

The output waveforms of total MBIST controller for March elements w0, (r0w1) are shown above. The inputs signals to this block are clock, reset, test high, and algorithm. If there is no memory, for testing cases mem_out has to be supplied by user. If there is memory under test, output of this memory is given to total MBIST controller as input which taken internally input to the comparator for comparison of memory data with the background data. This provides pass/fail information. Addr_out, data_out, cntrl_out are the outputs of total MBIST controller

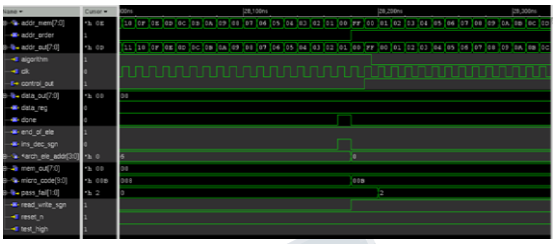


***Fig: output waveform of MCMBIST controller for MARCH C-MARCH C+ Algorithm***

***Explanation of output waveform for MCMBIST controller (March C-March C+) :***

The output waveforms of total MBIST controller when the algorithm is changing from March C to March C+ are shown above. The inputs signals to this block are clock, reset, test_high, and algorithm. If there is no memory, for testing cases mem_out has to be supplied by user. If there is memory under test, output of this memory is given to total MBIST controller as input which taken internally input to the comparator for comparison of memory data with the background data. This provides pass/fail information. Addr_out, data_out, cntrl_out are the outputs of total MBIST controller

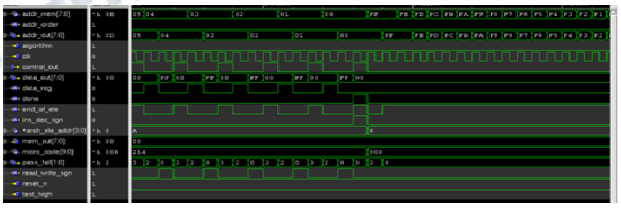


***Fig: output waveform of MCMBIST controller for MARCH C+ Algorithm***

***Explanation of output waveform for MCMBIST controller (March C+)***

The output waveforms of total MBIST controller for March elements (r1w0r0),r0 are shown above. The inputs signals to this block are clock, reset, test_high, and algorithm. If there is no memory, for testing cases mem_out has to be supplied by user. If there is memory under test, output of this memory is given to total MBIST controller as input which taken internally input to the comparator for comparison of memory data with the background data. This provides pass/fail information. Addr_out, data_out, cntrl_out are the outputs of total MBIST controller.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 10, October 2017

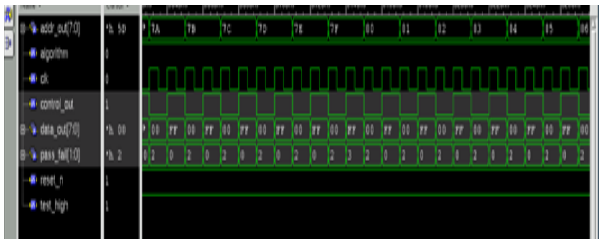


***Fig: output waveform of MCMBIST TEST BENCH for MARCH C Algorithm***

Explanation of top module test bench output waveform for MCMBIST controller (March C):

The output waveforms of total MBIST controller are shown above. The inputs signals to this block are clock, reset, test high, and algorithm. If there is no memory, for testing cases mem_out has to be supplied by user. If there is memory under test, output of this memory is given to total MBIST controller as input which taken internally input to the comparator for comparison of memory data with the background data. This provides pass/fail information. Addr_out, data_out, cntrl_out are the outputs of total MBIST controller

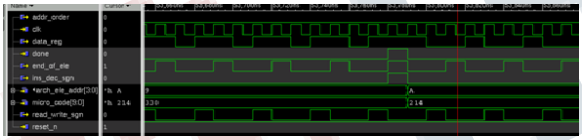


***Fig: Output waveform of MCMBIST Controller test bench for MARCH C+ Algorithm***

Explanation of top module test bench output waveform for MCMBIST controller (March C+):

The output waveforms of total MBIST are shown above. The inputs signals to this block are clock, reset, test high, and algorithm. If there is no memory, for testing cases mem_out has to be supplied by user. If there is memory under test, output of this memory is given to total MBIST controller as input which taken internally input to the comparator for comparison of memory data with the background data. This provides pass/fail information. Addr_out, data_out, cntrl_out are the outputs of total MBIST controller.

## 5. CONCLUSION & FUTURE SCOPE

### *5.1Conclusions:*

Memory testing is very important but challenging. Memory BIST is considered the most possible best solution due to various engineering and economic reasons. March tests are the popular algorithms currently implemented in BIST microcode. Various implementation schemes for memory BISTs are presented and their trade-offs are discussed:

A Microcode-based MBIST Controller has highly compacted Memory test algorithms. It has increased flexibility to alter the algorithms and reduced storage requirements. Different proposed innovations are also surveyed. Using Defect Coverage not Fault Coverage as our measure for test quality is revolutionary. Integrating diagnostic capabilities into BIST improves overall system robustness and chip yield. Automatic generation eases design efforts for test integration and help satisfying time-to-market requirements. Self-reparability is the key to fault-tolerant and reliable circuit. In conclusion, the future Memory BIST designs should be fast, small, efficient, robust, and flexible.

In this paper the address generation block and memory interface unit are designed and its logic is verified by performing the simulation and synthesis. Finally these blocks are integrated with program counter and instruction decoder, to form the Microcode based memory BIST controller.

The designed Microcode based memory BIST controller is simulated with the NC Verilog compiler at the clock frequency of 340 KHz, and net list is generated by using RTL compiler. The size of the MUT (memory under test) is 256x8 bits. Means, it has 256 row address locations with a data width of 8 bits. At the end of the test the memory BIST controller gives the pass/fail information of the memory under test. If the controller gives the pass signal it indicates that the memory is fault free; otherwise it indicates that some faults are present in the memory.

### *5.2Future Scope:*

The future Memory BIST designs should [8] be fast, small, efficient, robust, and flexible. The above challenges achieved in Embedded SoC memory testing will be driven by the following.

- Fault modeling: New fault models should be established in order to deal with new defects introduced by today and future (deep-submicron) technologies.
- Test algorithm design: Optimal test/diagnosis algorithms to guarantee high defect coverage for the new memory technologies en reduce the DPM level.
- BISR: Combining BIST with efficient and low cost repair schemes in order to improve the yield and system reliability.

- The configuration of the Microcode based MBIST controller is carried out by using more than one memory model.

## REFERENCE:

[1]. Mentor Graphics, "MBIST Architect-Reference Manual", Software Version V 8.6_4.
[2]. Mentor Graphics, "Built-In Self Test Process Guide", Software Version V 8.6_4.
[3]. Allen C. Cheng, "Comprehensive Study on Designing Memory BIST", Dec 2006.
[4]. Paul McEvoy and Ronan Farrel, "Built-in test engine for memory test", IEEE IC Test Workshop, Sep 2004.
[5]. Samir Palnitkar, " Verilog HDL", Low Price Edition,2006.
[6]. Micheal D.Cieletti, "Advanced Digital Design with the Verilog HDL", Prentice Hall of India, 2003.
[7]. Arthur D. Friedman, Melvin A. Breuer and Miron Abramovci, "Digital Systems Testing And Testable Design", A Jaico Book, 2006.
[8]. T. Chen and G. Sunada, "Design Self-Testing and Self-Repairing Structure for High Hierarchical Ultra-Large Capacity Memory Chips", IEEE Trans. On VLSI, Vo. 1, No. 2, pp. 88-97, June 1993.

## AUTHORS'BIOGRAPHY

**Vadde SeethaRama Rao** is presently working as an Assistant Professor in the Department of Electronics and Communication Engineering, SNIST Hyderabad, Telangana, India. He is having 4Years of teaching experience. His areas of interest are VLSI and Analog IC Design.

**CH. Ram Babu** is presently working as an Assistant Professor in the Department of Electronics and Communication Engineering, SNIST Hyderabad, Telangana, India. He is having 8Years of teaching experience. His areas of interest are Low Power VLSI, Analog IC Design, and Micro Electronics.