INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & MANAGEMENT

AREA OPTIMIZED FSM BASED BIST

Sonal Sharma*[1], Vishal Moyal[2]
*[1,2]SSITM, Bhilai (C.G.)

Abstract
This paper proposed the structured design methodology to construct FSM based programmable memory BIST approach for testing memory modules in SOC (system on chip). The BIST architecture could be used to test memories in different stage of their fabrication and therefore result in lower overall memory test logic overhead. The proposed scheme supports various memory test algorithms which are used to test different types of memory modules in SOC. We show that proposed FSM based BIST architecture achieves a good flexibility with smaller circuit size compared with previous methods.

Keywords — BIST, SOC, Programmable memory.

Introduction
In present scenario semiconductor memory testing plays a vital role in computer system design and diagnosis. There are few very good reasons why memory testing deserves special attention. First, memory is essential to electronic products. Almost all system chips contain some type of embedded memory, such as ROM, SRAM, DRAM, and flash memory. Second, dynamic increment in circuit complexity and device density of memory chips alike all other digital circuits which makes memory testing more and more complicated due to appearance of the new defect mechanisms in memory devices and constraints of fault coverage and the time spent on the test procedure. An FSM based programmable memory BIST controller architecture, was proposed in earlier. It gives users the selection of test algorithms on-line. When proposed method compared with previous programmable BIST designs, it achieves roughly the same level of flexibility, detects more faults and high frequency. And also it reduces area (gate count) without disturbing the speed The proposed method will be very useful in SOC testing, since many different memory core modules (e.g., DRAM,S since many different memory core modules (e.g., DRAM,SRAM and ROM) may be employed in SOC and they require different test algorithms.

*** Corresponding Author**
E. mail: sonal.sharma30@gmail.com

This paper presents algorithms for different test patterns, surveys of current memory BIST architecture, and discussion of various implementation issues. The paper is organized as follows: Section 2 describes the BIST algorithms of various memory tests. Section 3 present novel design aspects in memory BIST and respective simulation results. The conclusions are offered in Section 4.

BIST algorithm for various memory tests
There are many efficient testing algorithms have been proposed to detect different fault models [2]. Though implementing various testing algorithms in a single P-MBIST design would require high area cost. In our work, we maintained low area without disturbing the speed Traditionally P-MBIST architecture consists of following hardware units: MUX -- the set of multiplexers or another wrappers, which are used to isolate RAM module under test from external devices; TAP(WSP)-controller provides the serial communication between P-MBIST hardware and external devices and ATE (usually IEEE 1149.1 or P1500) interfaces are used) [3], [4]; FSM -- is a central core of PMBIST hardware, which controls all main units and executes the predefined memory test algorithm; MM -- micro program memory unit, which stores the test in binary format; RI - additional unit.

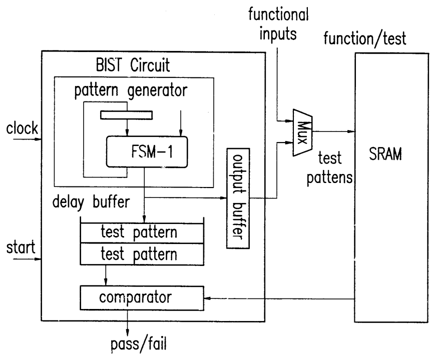


Figure1: Block diagram of BIST operation

Memory test is a set of basic operations performed on a Memory module to determine functionality. There is wide range of functional memory tests [5]. One type of tests that proven to be practically effective in time and complexity is the March test [6]. Any March test for bit oriented RAM can be defined by the set of primitives of MTL language:

1) The set of basic operations *r*0, *r*1, *w*0, *w*1, where *r* means read operation and *w* - write operation of predefined values 0(1) for the memory cell with current value of address *a*;
2) March element (test phase) - the concrete finite sequence of basic operations applied for current memory cell :(*r0*, *w*1, *r*1);
3) Each march element has addressing order, which denotes the direction of address space transmission: symbol ⇑denotes addressing order from 0 to 2m −1 , symbol ⇓denotes backward addressing order from 2m −1 to 0 and symbolic is used when the addressing order is irrelevant; for example the first march element of all tests looks like c(*w*0) ;
4) The finite set of different march elements forms complete march test; for instance, march test *MATS*++ can be written as {c (*w*0); ⇑(*r*0, *w*1); ⇓(*r*1, *w*0, *r*0)} .
In order to verify whether a given memory cell is good, it is necessary to conduct a sequence of write and read operations to the cell. The actual number of read/write operations and the order of the operations depend on the target fault model. Most commonly used memory test algorithms are March tests, in which there are finite sequences of *March elements*. A March element is a finite sequence of read (r) or writes (w) operations applied to a cell in memory before processing the next cell. The address of the next cell can be in either ascending or descending address order. The notations are summarized in the table shown below:

Table1: Notations of operations

r	A Read Operation
w	A Write Operation
⇧	Up addressing order
⇩	Down addressing order
⇳	Any addressing order

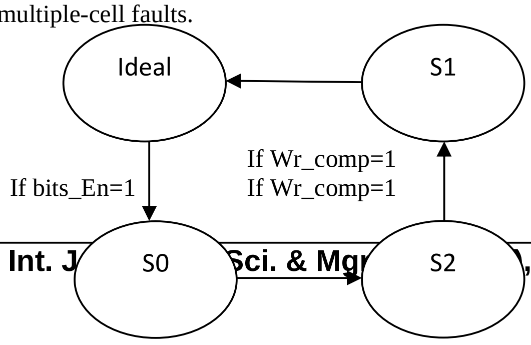
When an algorithm reads a cell response will be either 0 or 1 and they are denoted as r0 and r1 respectively. similarly write 0(1) into a cell is denoted as w1(w0) .we show commonly used test algorithm in table with above notation For example, the MATS+ algorithm first writes a 0 to each cell in any order ((w0)). In the second March element, it first verifies if the content in a given cell is 0, and then writes a 1 into the same cell. The process is conducted from address 0 up to the last memory cell ((r0, w1)). In the last March element, the algorithm verifies if the content of a cell is 1 and then write 0 back to the cell, for all cells starting from the last one down to address 0 ((r1, w0)).
From Table2, we can see that different test algorithms may have the same march elements, and thus we can design a simple and flexible BIST controller with shared components. In table2, the number of the first column indicates an algorithm, which is selected and sent by ATE. The proposed BIST supports 8 algorithms. Therefore, the selection number has 3 bits and indicates each supportable algorithm. Each algorithm consists of some codes. The code means a March element and has five-bits. The most significant bit of March element code means address order.

If the bit is set "0", address is generated in decreasing order. Else address is made in increasing order. Rest bits of March element code indicate read/write operation. For example, a code "1000" is reading "0"from memory, writing "1" and writing "0"to memory in regular sequence. There are total 35 march elements. However, we can express all algorithms using only 14 codes. Because same March elements share same code.

Table2: Memory test algorithms

No	Algorithm	March Elements Code
000	MATS+	{_(w0); _(r0,w1); _(r1,w0)}
001	March X	{_(w0); _(r0,w1); _(r1,w0); _(r0)}
010	March C-	{_(w0); _(r0,w1); _(r1,w0); _(r0,w1); _(r1,w0); _(r0)}
011	March A	{ (w0); (r0,w1,w0,w1); (r1,w0,w1);
100	March B	{_(w0); _(r0,w1,r1,w0,r0,w1); _(r1,w0,w1); _(r1,w0,w1,w0); _(r0,w1,w0)}
101	March U	{_(w0); _(r0,w1,r1,w0); _(r0,w1); _(r1,w0,r0,w1); _(r1,w0)}
110	March LR	{_(w0); _(r0,w1); _(r1,w0,r0,w1); _(r1,w0); _(r0,w1,r1,w0); _(r0)}
111	March SS	{_(w0);_(r0,r0,w0,r0,w1);_(r1,r1,w1,r1,w0); _(r0,r0,w0,r0,w1);_(r1,r1,w1,r1,w0); _(r0)}

Let us determine the basic operations in terms of notation. It is necessary to note that each read operation *r* (*dt*∈{0,1}) consists of two micro operations: reading he value *d* from selected memory cell and comparison the value *d* with the reference value *dt*:{1, *X* , *a*, *X* ,1}⇒{*d*}, *CMP* (*d* , *dt*) . (5)The basic write operation *w* (*dt*∈{0,1}) *dt* will be written as{1,0 →1, *a*, *dt*,0}⇒{*Z*} . (6)Each basic operation belongs to specified march element. All operations from current march element are performed sequentially for selected memory cell. When last operation from March element will be completed then the current value of address will be changed according to the specified addressing order. For this reason we denote two markers for all basic operations: *AO* - Address Order and *LO* – Last Operation. Also we add these markers to the notation of basic operations 1:[*AO*, *LO*]{*Inputs* }⇒{*Output* _ *Data* } . (7)Classical March tests are designed to detect different types of faults. All March tests are able to detect single-cell faults of different multiplicity, but not all of them are able to detect single faults, which affect more than one cell [6].At first case only two types of addressing order can be used: ⇑and ⇓. All three types of addressing order are used by multi run March tests to detect multiple-cell faults.



If Wr_comp=1

Figure2: Mats+ Algorithm

The first state of this algorithm is “Idle” state ,indicating that ,there is not any BIST operation is performed Mats+ algorithm will be in “Idle” state unless BIST_EN signal is remain Equal to “0” .The BIST operation start as soon as BIST_EN signal is made equal to “1” then in this algorithm the first operation is “ W0 “ which means that there is write 0” operation is to be performed hence “S0” is the first state in which write “0” operation when “write_complite” signal equal to “1” then FSM entered in new state “S1” . In “S1” state there is two element One is r0 and another W1 .When write one operation is performed then “write complete” signal become equal to “1” then FSM will switch to “S2” State .where in two operation required to be performed Read one and write Zero .Thus when the last operation is completed then FSM will switch to idle state.

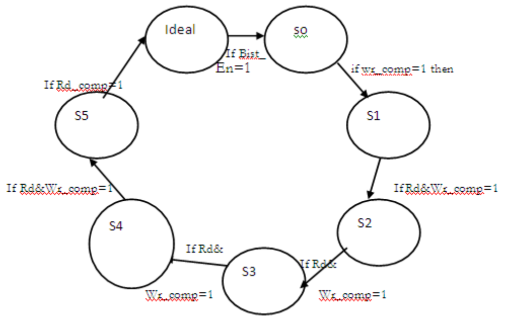


Figure3: Marchc minus Algorithm

In the fig FSM of March C Minus algorithm has been shown In which the first state “idle” which shows that this memory BIST is not working .whenever BIST_En is made Equal to “1” then the first operation of this algorithm is performed in form of “Write 0” operation this operation is defined by “the S0” state when write complete signal is made Equals to “1” then FSM switch to the “S1” state this state is having two elements so two operation is needed to be performed so as soon as second operation of this state of this state is performed then “Write complete 1” signal gets Equal to “1” results in FSM switches to the third state hence all operation of this state are performed in same manner a. finally idle state is achieved when last operation of this state is performed.

Simulation Results

For the efficiency of the proposed method, we implement proposed BIST. And we construct the RTL programmable MBIST (PMBIST) model for those algorithms.

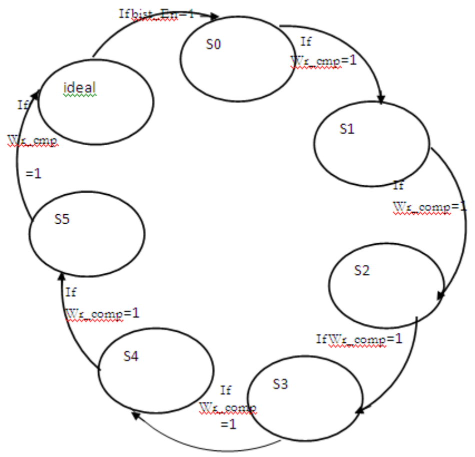


Figure4: MrchSS Algorithm

In the fig FSM of March SS algorithm has been shown In which the first state “idle” which shows that this memory BIST is not working .whenever BIST_En is made Equal to “1” then the first operation of this algorithm is performed in form of “Write 0” operation this operation is defined by “the S0” state when “write complete 0” signal is made Equal to “1” then FSM then switch to the “S1” state this state is having 4 elements so four operation is needed to be performed when is the first operation of this Algorithm is performed the counter ,along with this operation is increased by one to define that first of two identical operation is performed (the counter is provided as there are two identical operation consecutively given)so as soon as fifth operation of this state of this state is performed then “Write complete 1” signal gets Equal to “1” results in FSM switches to the third state hence all operation of this state are performed in same manner a. finally idle state is achieved when last operation of this state is performed
In the first experiment, we synthesize the PMBIST with Xilinx ISE 8.1i. To prove the better performance and area utilization, Table 3 has been shown.
Here in simulation we are showing the Faulty FSM BIST in which test input of 8 bit has been given to FSM BIST also “001” has been shown by “sel” signal to the MATS+ in which 1 test vector “0” needs to be written and “0” all eight location is written for 64 counts after which algorithm will be changed to next state.

Table 3: Result

Algorithm	**Gate Count of Previous method**	**Gate Count of Proposed method**
MATS+	730	216
March X	768	241
March C-	762	281
March B	1,038	1,215
March Lr	NI	905
March U	NI	885
March SS	NI	651

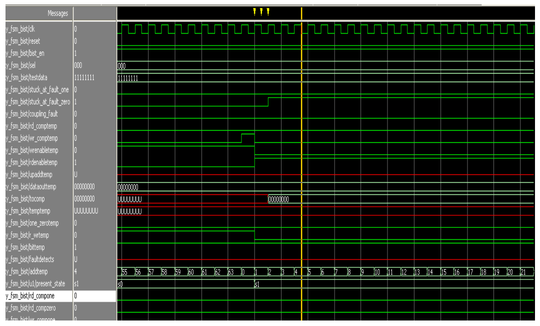


Figure5:Simulation of faulty FSM BIST

In this simulation ‘sel’ line is 000 is given to select the Mats+ algorithm in which S1 state is showing the read 0 operation during which output from RAM is transferred to comparator in test data to comparator is given as “1” hence stuck at 0 fault is being shown in this simulation

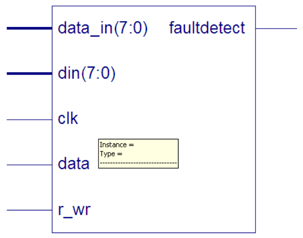


Figure6: RTL view of fault detect

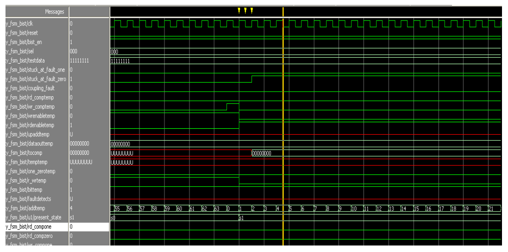


Figure 7: FINAL IMPLEMENTATION

Conclusion

We have implemented the FSM BIST by Finite state machine. We also compare the two FSM BIST in terms of gate count and power dissipation. The next thing we can do is to implement the Logic BIST.

References

1. P. H. Bardell, W. H. McAnney, and J. Savir. Built-In Test for VLSI: Pseudorandom Techniques. Wiley Interscience,
2. V. D. Agrawal, C. R. Kime, and K. K. Saluja, “A tutorial on built –test Principles,” IEEE Design & Test of Computers, Vol. 10, No. 2, pp. 69 77, March 1993.
3. J. van de Goor, Testing Semiconductor Memories: Theory and Practice, John Wiley and Sons, U.S.A., 1991. Proceedings.
4. A. J. van de Goor and A. Offerman, “Towards a uniform notation for memory tests,” in Proc. European Design and Test Conf., pp. 420-427,1996.
5. V. G. Mikitjuk, V. N. Yarmolik and A. J. van de Goor, “RAM testing algorithms for detecting multiple linked faults,” in Proc. EuropeanDesign and Test Conf., pp. 435-439, 1996.
6. H. Bardell and W. H. McAnney, “Built-in test for RAMs,” IEEE Design & Test of Computers, Vol. 5, No. 4, pp. 29-36, Aug 1988
7. WonGi Hong, Jung Dai Choi, Hoon Chang, “A Programmable Memory BIST” for Embedded 2008 International SoC Design Conference.
8. Balwinder singh , Sukhleen Bindra Narang, and Arun Khosla on “Address Counter / Generators for Low Power Memory BIST” IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011.
9. R. Nair, S. Thatte, and J. Abraham. E_cient algorithms for Random Access Memories. In IEEE Trans. on Computers, pages 572{576, June 1978.