International Journal of Computer Applications Technology and Research
Volume 3– Issue 4, 216 - 220, 2014, ISSN: 2319–8656

Implementation of FSM-MBIST and Design of Hybrid MBIST for Memory cluster in Asynchronous SoC

Lakshmi H R
Dept. of Electronics and Communication
BNM Institute of technology
Bangalore, India

Varchaswini R
Dept. of Electronics and Communication
BNM Institute of technology
Bangalore, India

Yasha Jyothi M Shirur
Dept. of Electronics and Communication
BNM Institute of Technology
Bangalore, India

Abstract: In current scenario, power efficient MPSoC's are of great demand. The power efficient asynchronous MPSoC's with multiple memories are thought-off to replace clocked synchronous SoC, in which clock consumes more than 40% of the total power. It is right time to develop the test compliant asynchronous MpSoC. In this paper, Traditional MBIST and FSM based MBIST schemes are designed and applied to single port RAM. The results are discussed based on the synthesis reports obtained from *RTL Complier from Cadence*. FSM based MBIST is power and area efficient method for single memory testing. It consumes 40% less power when compared with traditional MBIST. But, in case of multiple memory scenarios, separate MBIST controllers are required to test each individual memories. Thus this scheme consumes huge area and becomes inefficient. A novel technique for testing different memories which are working at different frequencies is in need. Therefore, an area efficient Hybrid MBIST is proposed with single MBIST controller to test multiple memories in an Asynchronous SoC. It also includes multiple test algorithms to detect various faults. An Asynchronous SoC with DWT processor and multiple memories is discussed in this paper, which will used as Design under Test [DUT] and Hybrid MBIST is built around it to test the heterogeneous memories. The design is coded in Verilog and Validated in Spartan-3e FPGA kit.

Keywords: FSM MBIST, Hybrid MBIST, Asynchronous SoC, low area, flexible, MARCH Algorithms

1. INTRODUCTION

Today's SoC's are memory dominant. More than 90% of physical area is dominated by memory according to the ITRS [International Technology Roadmap for Semiconductors] [3]. As memories become denser, they are more prone to defects and faults are more complex. Using external Automatic Test Equipment (ATE) will become expensive for dense memories due to pin inductance and tester pins cost [12]. Also, at-speed testing is not possible. Test time increases as the number of memories increase in a chip. An on-chip at-speed testing technique is the call of the hour. BIST is a Design for Test (DFT) technique [2] in which part of the circuit is used to test the circuit itself. MBIST is the most widely used technique for testing memories. MBIST technique integrates the functionality of automatic test equipment on the same die as embedded memories. It provides high speed testing and also testing can be done during operation and maintenance stage even outside the electrical testing environment.

There are two widely used techniques for MBIST. FSM based MBIST and microcode based MBIST. In FSM based MBIST, control signals for BIST controller operations are defined as state machines [8]. It is hardwired. Microcode based MBIST has a Programmable Memory BIST (P-MBIST) controller. It has flexibility of programming new test algorithms into the controller as and when needed, but suffers from area overhead [8].

The paper is organised as follows. Section 2 describes FSM based MBIST and introduces a hybrid MBIST scheme called *FSM-based Programmable MBIST*. Section 3 describes various MBIST algorithms. Section 4 describes Design Under Test (DUT) for hybrid MBIST and the proposed MBIST Controller.

2. FSM BASED BIST

FSM based BIST has a number of states. This technique uses 1 test algorithm for test operation. In the proposed paper, MARCH C- algorithm is used. This algorithm is chosen because it is simple and yet can detect many faults like ***stuck-at faults, address decoder faults, transition faults and some coupling faults*** [9].

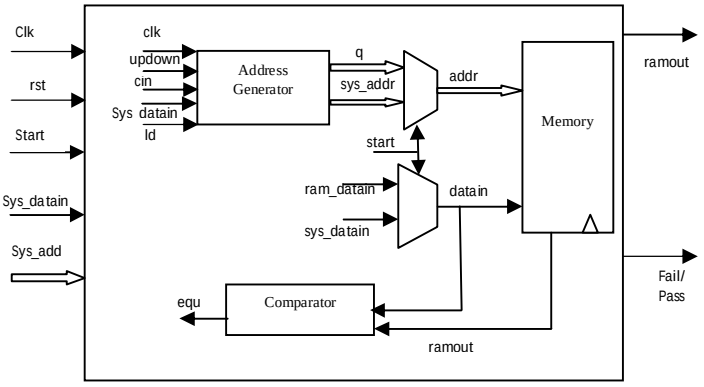

Figure. 1 Block Diagram for FSM MBIST

Figure 1 Shows block diagram for FSM based MBIST Architecture. In the block diagram ***Start, rst, Clk, Sys_datain, Sys_addr*** are input signals, ***ramout and Fail/Pass*** are the output signals. The blocks in Figure 1 are ***Address generator, Single -port Memory, Comparator and Multiplexers***. The Address generator is a counter which generates address for the memory to be tested. It can count up or down to provide address in ascending or descending order. Comparator compares the datain and ramout, if they are not equal ***Fail*** signal is raised, else, ***Pass*** signal is raised. The multiplexer

chooses between ***normal mode*** and ***test mode***. When start=1, test mode is selected.

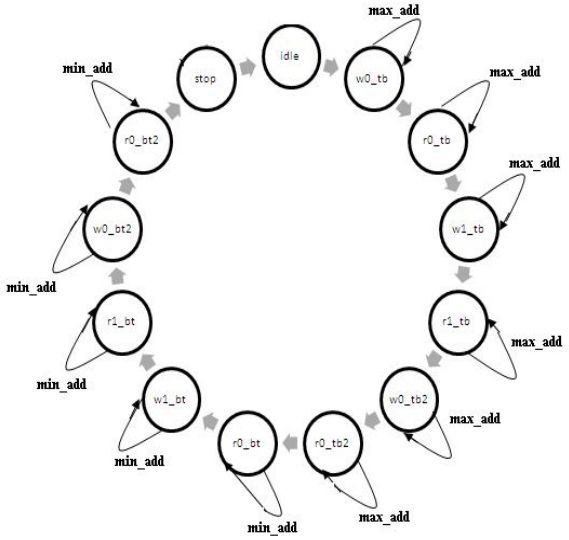

Figure. 2 State Diagram of FSM MBIST

Figure 2 shows the State Diagram for implemented FSM MBIST. The faults covered by this architecture are Address Decoder Faults, Transition Faults and Stuck-at Faults [2] The FSM MBIST has low area and good fault coverage. The percentage of fault coverage depends on the MARCH algorithm used. But since FSM MBIST is hard-wired, it is not flexible.

Presently, due to the need for low power and low area, asynchronous System on Chip (SoC)s are evolving. An asynchronous SoC contains memories of different types and different sizes operating at different frequencies. Hence a MBIST controller should be capable of testing heterogeneous memories at their respective frequency of operation. A single algorithm is not sufficient to test all the memories. Hence FSM MBIST will not be efficient in case of multiple memories. A microcode MBIST scheme can be used, but it results in larger area. To compensate for lack of flexibility in FSM MBIST and area overhead in microcode MBIST, a hybrid scheme is proposed- ***FSM based programmable MBIST***. The test algorithms and the MARCH elements are represented by codes and clusters of micro codes. Read/write operations, the applied test data and the addressing orders in an element are still controlled by the FSM.

3. MARCH ALGORITHMS

Patterns are the key elements in memory testing. These patterns look for weakness in the analog circuitry and for interaction between two or more neighboring structures [2]. A test algorithm is a finite sequence of test elements. A test element contains memory operations, data for write and read operations and address specified for the read and write operations to the memory [4].

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. March based tests are simple and they bear good fault coverage. Hence they are widely accepted and implemented in most modern memory BIST schemes [5].

The manner in which an operation of March algorithm progresses from cell- to-cell is determined by the addressing order, which can be an ascending order (addresses moving up from cell 0 to cell n- 1), denoted by the '↑' symbol, or a descending order, denoted by the '↓' symbol (addresses moving down from cell n-1 to cell 0). For some March elements the address order can be chosen arbitrarily, ie either ascending or descending. This will be indicated by the '↕' symbol. operations applicable to cells can be a 'w0', a 'wl', an 'r0' or an 'rl' operation. A complete march test is contained within the '{...}' bracket pair; while a March element is contained within the '(...)' bracket pair [13]. The codes of all test algorithms used in the proposed work are listed in the Table I. 8 MARCH algorithms are used.

Table 1. Test Algorithms with March Element Codes

Sl No.	Code	Algorithm	March Element Code
1	000	Mats Plus	{↕(w0); ↑(r0,w1); ↓(r1,w0)}
2	001	March X	{↕(w0); ↑(r0,w1); ↓(r1,w0);↕(r0)}
3	010	March C Minus	{↕(w0); ↑(r0,w1); ↑(r1,w0); ↓(r0,w1); ↓(r1,w0); ↕(r0)}
4	011	March LR	{↕(w0);↓(r0,w1);↑(r1,w0,r0,w1); ↑(r1,w0); ↑(r0,w1,r1,w0); ↑(r0)}
5	100	March A	{↕(w0);↑(r0,w1,w0,w1) ;↑(r1,w0,w1); ↓(r1,w0,w1,w0); ↓(r0,w1,w0);}
6	101	March U	{↕(w0);↑(r0,w1,r1,w0); ↑(r0,w1); ↓(r1,w0,r0,w1); ↓(r1,w0)}
7	110	March B	{↕(w0);↑(r0,w1,r1,w0,r0,w1); ↑(r1,w0,w1);↓(r1,w0,w1,w0); ↓(r0,w1,w0)}
8	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)}

4. DUT AND HYBRID MBIST TOP MODULE

Due to the recent trend in the use of asynchronous SoC's, a prototype has been designed to replicate the scenario. The asynchronous SoC in the proposed paper has a Discrete Wavelet Transform (DWT) processor, a Dual port RAM (Memory-1) and two Single port RAMs (Memory-2, Memory-3). All the blocks interact via asynchronous handshaking signals.

An input image is stored in Memory-1. The DWT processor requests the image and converts it to ***high frequency and low frequency components***. On request by Memory-2 and Memory-3 the high and low frequency components are stored in these two memories respectively. All the 3 memories are operating at different frequencies. The proposed BIST controller tests these heterogeneous memories at their respective frequency of operation. The design operates in two modes **– *normal mode*** and ***test mode.***

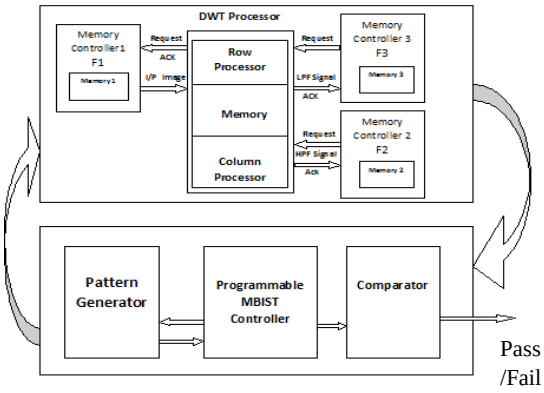

Figure. 3 DUT and Top Level of Hybrid MBIST

The top level of MBIST consists of a pattern generator, PMBIST controller and a comparator. The pattern generator generates the necessary MARCH patterns which are written to and read from the memory under test (MUT). The PMBIST controller controls the read and write operations. The comparator compares the pattern read from the memory with the pattern given by pattern generator and issues Pass/Fail signal.

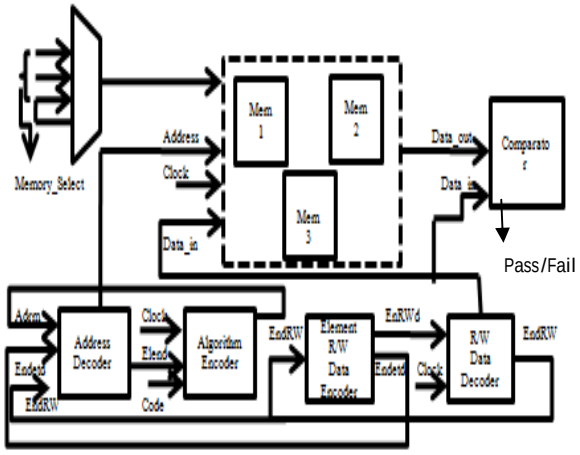

Figure. 4 Hybrid MBIST Controller

Figure 4 shows the block diagram for implementation of the proposed MBIST scheme. The MBIST controller shown if Figure 4 comprises of 4 blocks; ***Address Decoder***, ***Algorithm Encoder***, ***Element R/W Data Encoder*** and ***Data Decoder***. The Algorithm Encoder sends the MARCH elements' code and the address microcode to the Element R/W Data Encoder and the Address Decoder. The Element R/W Data Encoder receives the MARCH element's code and encodes the R/W operation as 0 for Read and 1 for Write. For example, element 2, r1w0 is encoded as 0110. It produces the ***Endetd*** signal to indicate the end of the encoding for the element under test. The address decoder is controlled by ***Endetd*** and ***Adrm*** (which represents address sequence microcode). It generates the ***Elend*** signal which triggers the Algorithm Encoder. The active ***Elend*** signal means the MARCH element under test has reached the last address and is ready for the next MARCH element. Finally the R/W-Data Decoder receives the encoded test data in form of microcode instructions, decodes it and writes to the memory. It also creates the ***EndRW*** signal to signify the end of RW operation for that cluster. The comparator compares the data read from memory, Data_out and data written to memory, Data_in. if both data are same, Pass signal is asserted, else Fail signal is asserted. This block shares the same clock as the memory under test. In this paper a multiplexer is used to select one of the three memories, for test.

Table 2. March Element Clusters

4 bits MARCH Element Code	R/W Operation in the Element
0000	w0
0001	r0
0010	w1
0011	r1
0100	r1,w0
0101	r0,w1
0110	r1,w0,w1
0111	r0,w1,wo
1000	r0,w0
1001	r1,w1
1010	r0,w1,r1,w0

Table 1 shows the 8 MARCH algorithms used in the proposed hybrid MBIST scheme. Based on the code, Algorithm Encoder selects one of the 8 algorithms and encodes it in form of clusters of microcode. Clusters are shown in Table 2. As the algorithms use common clusters, area is reduced. The first algorithm is selected and if no faults are detected next algorithm is selected and so on. If no faults are detected in all 8 algorithms, the memory is defect free.

5. RESULTS AND DISCUSSIONS

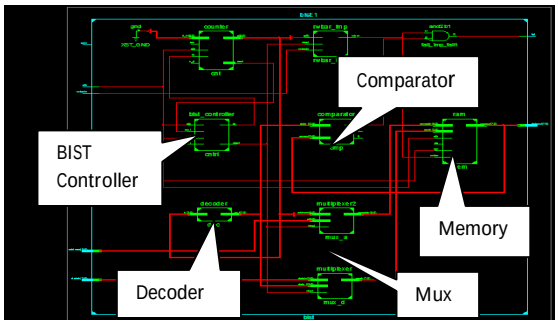

Figure. 5 RTL Schematic of Traditional MBIST

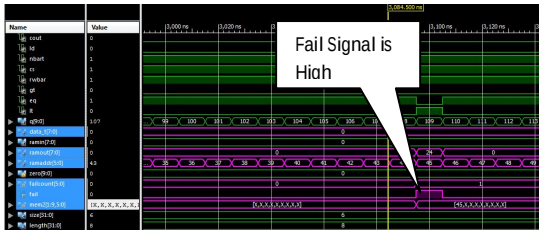

Figure. 6 Simulation result of Traditional MBIST with fault at address 45

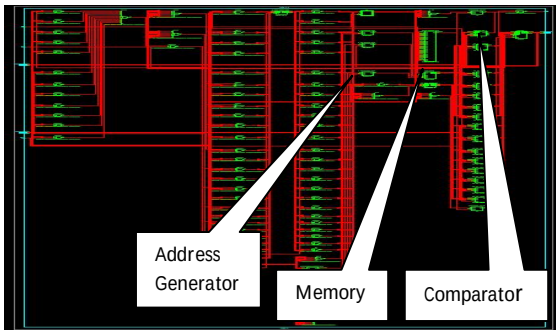

Figure. 7 RTL Schematic of FSM MBIST

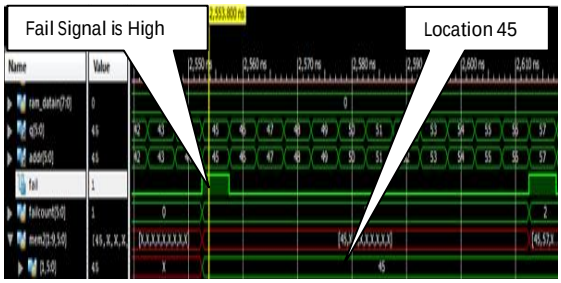

Figure. 8 Simulation result of FSM MBIST with fault at address 45

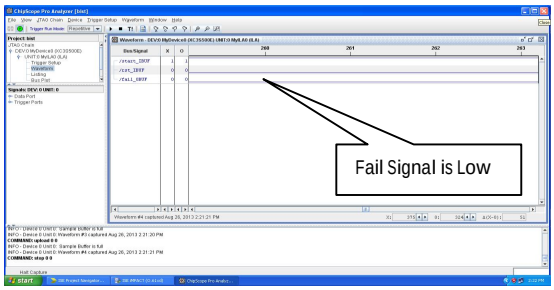

Figure. 9 Chip Scope Window showing Fail Signal

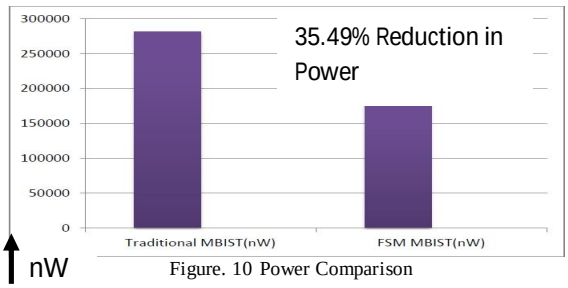

Figure. 10 Power Comparison

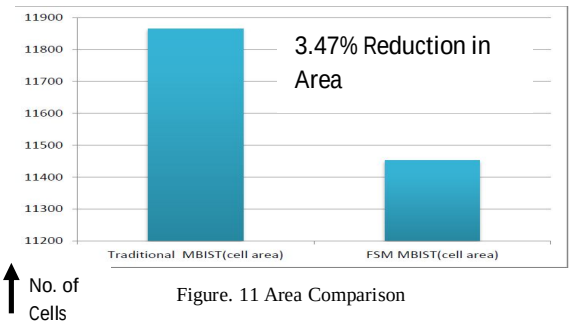

Figure. 11 Area Comparison

The Traditional MBIST was developed and the results obtained has been compared with FSM based MBIST. Figure 5 shows the RTL schematic for traditional MBIST. Figure 6 shows the simulation where Fail signal is asserted at location 45. Similarly Figure 7 shows the RTL schematic for FSM MBIST. Figure 8 shows the Fail signal being asserted in FSM MBIST. The fault is at address location 45 and it is stored in a register. FSM MBIST was successfully downloaded on the Spartan 3e kit and results were verified on chip scope as shown in Figure 9, where a fail signal is shown low for faultless architecture. The synthesis reports obtained from RTL Compiler from Cadence are used for comparison. The power taken by FSM MBIST is about 36% less when compared with the traditional one. The area remains almost same as that of traditional. The comparison graphs for power and area are shown in Figure 10. The fault coverage of the proposed hybrid MBIST is expected to be higher than that for FSM MBIST.

6. CONCLUSION

MBIST is the most widely used technique for testing memories. Traditional MBIST and FSM based MBIST can be applied to single memory at any given instant of time. Both these techniques are coded in Verilog and validated in Spartan-3e FPGA kit. The synthesis reports obtained from *RTL Complier from Cadence* are used to arrive at conclusion. FSM based MBIST is power and area efficient method for memory testing. It consumes nearly 40% less power when compared with traditional MBIST. But, in case of multiple memory scenarios, separate MBIST controllers are required to test individual memory. This method of testing memories consumes huge area and becomes inefficient. Therefore, an area efficient Hybrid MBIST is proposed with single MBIST controller to test multiple memories in an Asynchronous SoC. It also includes multiple test algorithms to detect various faults.

7. ACKNOWLEDGEMENTS

The authors wish to acknowledge BNMIT Management for supporting this work.

8. REFERENCES

[1] K.Zarrineh, and S.J. Upadhyaya, 1999. On programmable memory built in-self test architecture. In Proceedings of IEEE Design, Automation and Test in Europe Conference.

[2] R. Dean Adam. 2003. High Performance Memory Testing. Kluwer Academic Publisher.

[3] Paul McEvoy and Ronan Farrell. 2004. Built-In Self Test Engine For Memory Test. In proceedings of IEEE IC Test Workshop.

[4] A.J. Van de Goor. 1991. Testing semiconductor memories: Theory and Practice. John Wiley and Sons, Inc.

[5] Gayathri CV, Kayalvizhi N, and Malligadevi M. 2009. Generation of New March Tests with Low Test Power and High Fault Coverage By Test Sequence Reordering Using Genetic Algorithm. In proceedings of International Conference on Advances in Recent Technologies in Communication and Computing.

[6] Po-Chang Tsai, Sying-Jyan Wang and Feng-Ming Chang. 2005. FSM- Based Programmable Memory BIST with Macro Command. In proceedings of IEEE International Workshop on Memory Technology, Design, and Testing (MTDT).

[7] Sonal Sharma, Vishal Moyal. February 2013. Programmable FSM based MBIST Architecture. International Journal of Digital Application & Contemporary research, Volume 1, Issue 7.

[8] NurQamarina Mohd Noor, Azilah Saparon,Yusrina Yusof. 2010. Programmable MBIST Merging FSM and Microcode Techniques Using Macro Commands. In proceedings of 25th International Symposium on Defect and Fault Tolerance in VLSI Systems.

[9] Said Hamdioui1, Ad J. van de Goor, Mike Rodgers. 2002. March SS: A Test for All Static Simple RAM Faults. Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing.

[10] Nor Zaidi Haron, Siti Aisah Mat Junos, Abdul Hadi Abdul Razak and Mohd. Yamani Idna Idris. 2007. Modelling and Simulation of Finite State Machine Memory Built-in Self Test Architecture for Embedded memories. In proceedings of Dec. 2007 Asia-Pacific Conference on Applied Electromagnetics. pp. 1-5.

[11] http://en.wikipedia.org/wiki/Discrete_wavelet_transform

[12] www.eng.auburn.edu

[13] Van de Goor, A. J. Zorian, Yervant. 1993. EffectiveMarch Algorithms for Testing Single Order Addressed Memories. In proceedings of the European Conference on Design Automation with the European Event in ASIC Design