Ijesm
Consulting, help, relaxation
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES
&
MANAGEMENT

FSM BASED BIST ARCHITECTURE

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

Abstract

This paper presents a FSM based Programmable Built-In Self-Test (BIST) approach for testing memory modules in SOC(system on chip) In general, there are a variety of heterogeneous memory modules in SOC, and it is not possible to test all of them with a single algorithm. Thus, the proposed BIST is flexible and selectable. The selection command is used to select a test algorithm from a set of predetermined set of algorithms that are built in memory BIST. By using various memory test algorithm to test different memory module in SOC is scheme greatly simplifies the testing process. We also develop a programmable memory BIST generator that automatically produces RTL model of the proposed BIST architecture for a given set of test algorithms.

Keywords— FSM, BIST, SOC

Introduction

Digital systems are composed of memory subsystems defects in this memory arrays are generally due to shorts and opens in memory cells, address decoder and read/write logic. These defects can be modeled as single and multicell memory faults. These defects can be tested using Built-in Self-Test (BIST).

Built-In Self-Test (BIST) is a promising methodology for the aforementioned test problems. In the memory BIST (MBIST) technology, a dedicated BIST controller is used to implement a specific memory test algorithm when the chip under test (CUT) is in test mode [1]-[2]. We do not use this approach because the requirement for test algorithm may change during the life cycle of a given memory device and SOC usually contains different types of memory blocks, and each type many need a distinct test algorithm. Many programmable MBIST [3]-[8] have been developed, and they can be classified into two categories: one is based on the finite state machine (FSM) model [3],[4], and the other is based on the microcode controller [3]-[8]. In general, the microcode-based approach is more popular since it is more flexible because design of such a BIST increases the complexity, and the area overhead is usually higher. Here test algorithm is selected by applying some predefined instructions. We have to create a test instruction in each step of the test algorithm, and these external instructions are transferred to the internal registers one by one. In this way, we can freely select the applied test algorithm. The advantage of this approach is the flexibility to select test algorithms, but both complexity of the control mechanism and area overhead increase as well. If the memories are embedded in SOC, the automatic test equipment (ATE) needs to send many test commands to execute a test algorithm, and this process also increases the test time. In this paper, we propose a new FSM-based BIST architecture.

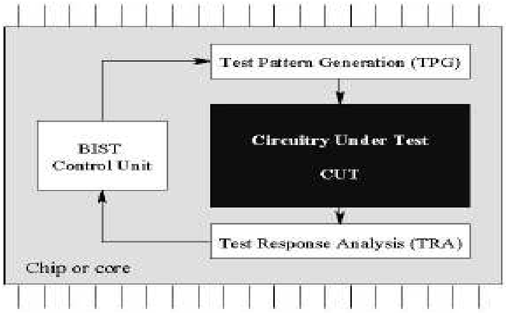

Figure 1.1: FSM based BIST Architecture

The proposed is more efficient in terms of circuit size and test data to be applied, and it requires less time to configure the BIST. We also develop a programmable memory BIST generator that automatically produces RTL model of the proposed BIST architecture for a given set of test algorithms.

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

Proposed Test Pattern Generator

For test a given memory cell is good, it required 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.

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

Table 1.1: Notations of operations

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 Table 1.2, 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.

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);

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)}

Table 1.2

We optimized a simple small test pattern generator for use in the memory BIST, utilizing both of the aforementioned characteristics of the five march algorithms.

Matsplus algorithm FSM

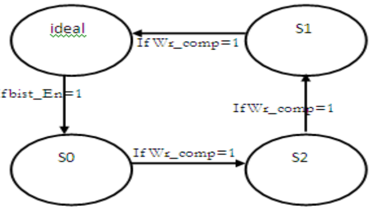

Figure 1.2: Matsplus algorithm Architecture

CMinus algorithm FSM

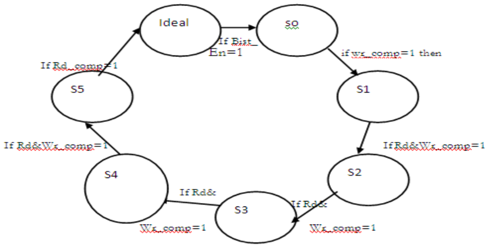

Figure 1.3: CMinus algorithm Architecture

March LR

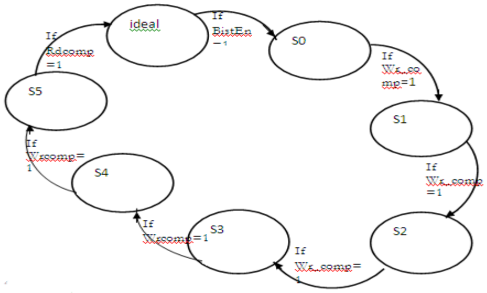

Figure 1.4: March LR Architecture

March-X

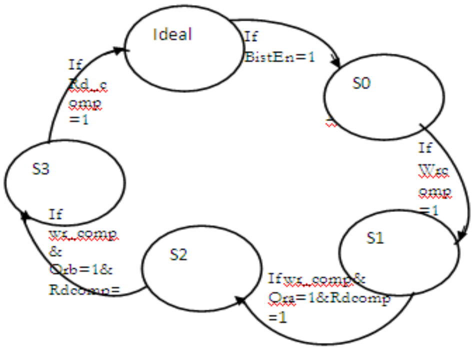

Figure 1.5: March-X Architecture

March U

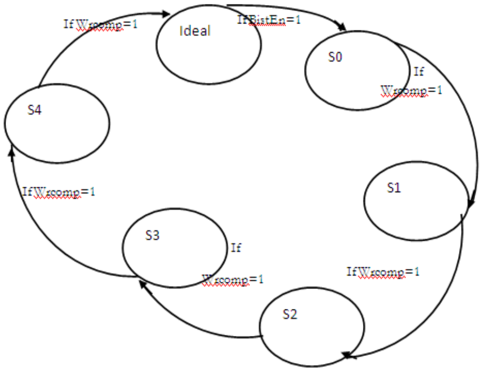

Figure 1.6: March U Architecture

Simulation Result

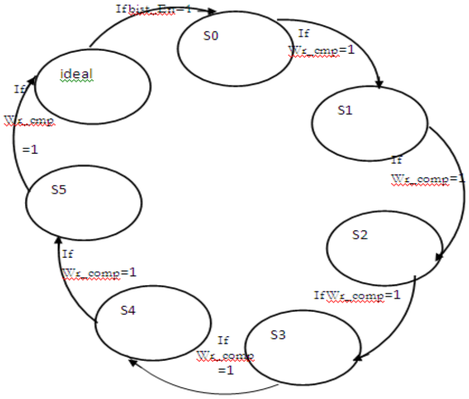

Figure1.7: MrchSS Algorithm

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, a table 1.3 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

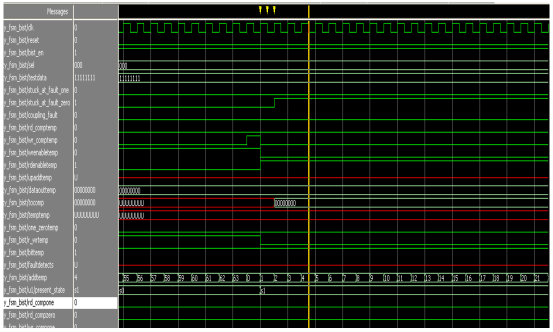
Messages
y_fsm_bist/clk 0
y_fsm_bist/reset 0
y_fsm_bist/bist_en 1
y_fsm_bist/sel 000
y_fsm_bist/testdata 11111111
y_fsm_bist/stuck_at_fault_one 0
y_fsm_bist/stuck_at_fault_zero 1
y_fsm_bist/coupling_fault 0
y_fsm_bist/rd_comptemp 0
y_fsm_bist/wr_comptemp 0
y_fsm_bist/wrenabletemp 0
y_fsm_bist/rdenabletemp 1
y_fsm_bist/upaddtemp U
y_fsm_bist/dataouttemp 00000000
y_fsm_bist/tocomp 00000000
y_fsm_bist/temptemp UUUUUUUU
y_fsm_bist/one_zerotemp 0
y_fsm_bist/r_wrtemp 0
y_fsm_bist/bittemp 1
y_fsm_bist/faultdetects U
y_fsm_bist/addtemp 4
y_fsm_bist/u1/present_state s1
y_fsm_bist/rd_compone 0
y_fsm_bist/rd_compzero 0
000
11111111
00000000
UUUUUUUU
55
56
s0

Figure1.8: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

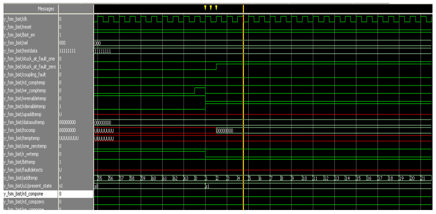

Figure 1.9: Final Implementation

Conclusion

In this paper we design a FSM based BIST architecture it provide us with the ability of select test algorithms on-line The proposed method will be very useful in SOC testing, since many different memory core modules (e.g., DRAM, SRAM and ROM) may be employed in SOC and they require different test algorithms.

References

1. J. van de Goor and A. Offerman, "Towards a uniform notation for memoty tests," in Proc. European Design and Test Conf., 1996, pp. 420-427.
2. V. G. Mikitjuk, V.n. Yarmolik, and A.J. van de Goor, "RAM testing algorithms for detecting multiple linked faults," in *Proc. European Design and Test Conf.*, 1996, pp. 435-439.
3. K. Zarrineh, and S. J. Upadhyaya, "On programmable memory built-in self test architectures," in Proc. IEEE Design, Automation and Test in Europe Conf., pp. 708-713, Mar. 1999.
4. K. Zarrineh, and S. J. Upadhyaya, "Programmable memory BIST and a new synthesis framework," in Proc. Fault-Tolerant Computing Symp, pp. 352-355, 1999.
5. 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.
6. 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.
7. P. 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.
8. W.L. Wang, K.J. Lee, and J.F. Wang, "An on-chip march pattern generator for testing embedded Memory Cores", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 9, No. 5, pp. 730-735, Oct. 2001.