International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 6 (June 2014), PP.77-83

Hardware Sharing Design for Programmable Memory Built-In Self Test

[1]K. Abdul Munaf, [2]K.Venkata Siva Reddy, [3]M.V.Ramana Reddy
[1]Assistant Professor, Department of ECE, RCEW, Kurnool, Andhrapradesh, India
[2] Assistant Professor, Department of ECE, RCEW, Kurnool, Andhrapradesh, India
[3] Assistant Professor, Department of ECE, RCEW, Kurnool, Andhrapradesh, India

Abstract:- As the progress of deep submicron technology, embedded memory grows greatly in the System-on-Chip design. An efficient test method with relatively low cost is required for mass production process. Programmable Built-In Self-Test (P-MBIST) solution provides a certain degree of flexibility with reasonable hardware cost, based on the customized controller/processor. In this work, we propose a hardware sharing architecture for P-MBIST design. Through sharing the common address generator and controller, the area overhead of P-MBIST circuit can be significantly reduced. Higher testing speed can be achieved by inserting two pipeline stages. Finally, the proposed P-MBIST circuit can be automatically generated from the user-defined configuration file.

Keywords:- Memory, Instructions, Controller, Pipelining, P-MBIST, etc.,

I. INTRODUCTION

With increasing design complexity in modern SOC design, many memory instances with different sizes and types would be included. To test all of the memory with relatively low cost becomes an important issue. Providing user-defined pattern for screening out various manufacturing defects is also a major demand. To ease the tradeoff between the hardware cost and test flexibility, Programmable Built-In Self-Test (P-MBIST) method is an opening approach to complete the memory testing under these circumstances.

Many researchers have been focused on P-MBIST design [1-5]. Processor-based architecture provides high test flexibility, but it increases the test development costs while applying to various processor family [1-2]. To lower the design cost, a customized processor and instruction have been developed [3]. It uses program memory to store the test program. To further reduce the hardware cost, the instruction can be serially input and saved in one internal register by adopting simple controller [4-5]. However, the issue for multi-instance testing in parallel is not discussed [6].

In this work, we propose a hardware sharing architecture to test the memory with same type in parallelism. The proposed method uses only one address counter to generate the required address for March-based algorithm, including row scan and column scan. The controller can be applied to different memory types with the same read/write cycle. Higher testing speed can be achieved by inserting two pipeline stages. Finally, the proposed P-MBIST engine can be automatically generated from the user-defined configuration file. The paper is organized as follows. Section 2 gives a brief concept of the March-based algorithm. In Section 3, the proposed architecture and the detailed component design will be addressed. The pipeline design issue is also discussed. Section 4 describes the automation flow of the overall P-MBIST engine. The implementation result and the relative comparison are presented in Section 5. Finally, a brief summary is given in Section 6.

II. SUPPORTED TESTING ALGORITHMS

Many efficient testing algorithms have been proposed to detect different fault models [7-8]. However, to implement various testing algorithms in the same P-BIST design would require high area cost. Thus, the selection of test algorithm families should be carefully considered. March-based algorithm is an important class to detect a large variety of faults [8]. A March test consists of a finite sequence of March elements. Each element performs a series of "reading" and "writing" operations to all memory cells. The addressing order of each element is executed in ascending (), descending (), or either () way. As a test example described in Fig. 1, it contains three March elements. The first element performs writing 0 to each memory cell in either addressing order. The second element performs two operations to each address in ascending order. Finally, the third element reads all the address in descending order. The complexity order is noted to 4N, where N is the number of memory addresses.

$$\text{Test sequence (4N): } \{\Updownarrow (w0),\ \Uparrow (r0, w1),\ \Downarrow (r1)\}$$

Fig. 1 An example of March-based test sequence

Beside, based on the March-based test sequence, the dynamic faults can be tested by repeatly performing the same operation in the same memory cell [9].To efficiently implement March-based algorithm, we define the corresponding instruction for P-MBIST architecture. The advantage is that one can program the instruction to modify the testing algorithm during run time. In our shared architecture, only one address generator is required for testing all memory. Also, different memory types with the same read/write cycle can share the same controller. In the following section, we will discuss the overall architecture and its detailed component design. The pipeline design issue will be also addressed.

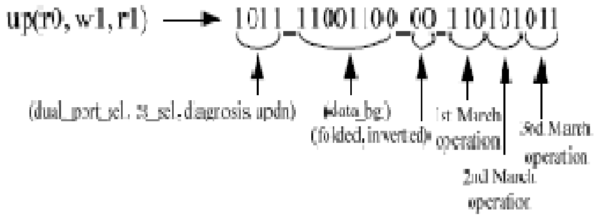

Fig. 2 An instruction with 3 March operations and 8-bit data background

III. PROPOSED PMBIST ARCHITECTURE

To support flexible March elements and diagnostic procedures, the proposed P-MBIST provides the following features :(1)different data background,(2) programmable March-based element,(3) column-scan addressing mode,(4) capturing the address, syndrome and state of failures,(5) parallel and sequential test, (6)pipeline mode for higher test speed. TABLE I lists the detailed programmable items of instructions. The folded and inverted bits can invert the data_bg while the switching of the specified address bit. By setting the data_bg to 0 and configuring both folded and inverted bits to 1, the checkerboard pattern can be generated. Column-scan addressing can be controlled by cs_sel. Finally each March operation description uses three bits, EOC, R/W, and polarity, to define the operation. Fig 2 gives an instruction example of performing three operations in a March element. By setting up the diagnosis bit to 1, the failure information will be dumped out while detecting, them is matching memory output.

TABLE I. Description of each instruction bit
(* Each March operation is represented by three bits)

Bit Name	Description
dual_port_sel	Valid for dual_port group: 0: port_A 1: port_B
cs_sel	0: Row-scan 1: Column-scan Address is accumulated by the numbers of column Mux
diagnosis	0: No dump 1: Dump fail information
updn	0: Address-down counting 1: Address-up counting
data_bg	Programmable data background
folded	1: data_bg inverts itself when the new row starts
inverted	1: data_bg inverts itself at each address
*EOC	0: The last command 1: Not the last command
*R/W	Write: 0 Read: 1
*Polarity	0: No inversion for data_bg 1: Inversion for data_bg

Fig 3 shows the block diagram of the proposed P-MBIST design, firstly the instruction is serially shifted into instruction_read module. To reduce the hardware costs, the address counter is shared to test all memory instances. Moreover the decoder block can support different memory type testing without increasing any state (i. e each memory has the same read/write cycle). The memory in the same type can be grouped and tested in either sequential or parallel way. The supported memory types include single/dual two-port SRAM, one/two-port register file and ROM. Finally according to the user-define configuration file the proposed P-MBIST engine can be automatically generated from the developed software program.

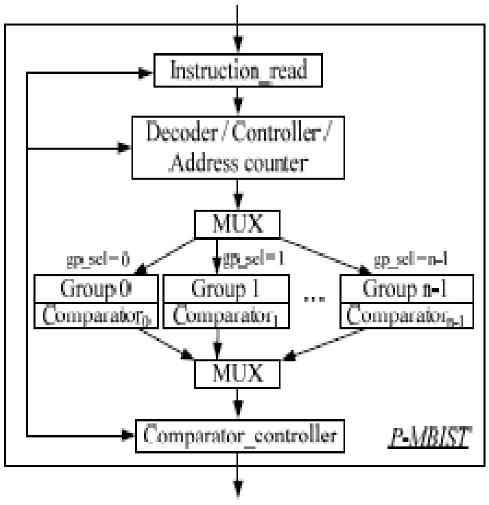

Fig. 3 Block diagram of P-MBIST design

A. Address Generator

To share the common address generator, it is a simple way to choose an address counter with the largest address bit among all the under-tested memory instances. As shown in Fig. 4, the low-bit of address counter can be used to indicate the low-address memory instance, by properly controlling the chip-select signal for each memory instance.

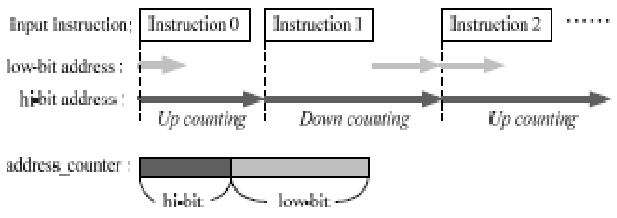

Fig. 4 Timing diagram of shared address counter

As for the column-scan mode, the address counter is not just ascending by 1. TABLE II provides an example, which defines the column Mux to 4 and the maximum address to 11. In TABLE II, the 1st column shows the address counter output for row scan (i.e. ascending by 1). The 2nd column gives the address for column scan (i.e. ascending by 4). To share the row-scan and column-scan adder, we use the up-scramble to adjust the added bit for column-scan mode. As shown in the 3rd column, the low bit of A[1:0] is shifted to the position of most significant bit, which means the row adder can directly apply to the A[2] bit. Similarly, after the process of down-scrambler, the column-scan address for ascending by 4 can be generated. By adding two scramble blocks, the row adder can support the column addressing. Fig. 5 shows the block diagram of address generator for row-scan and column-scan modes.

TABLE II. Scrambler function for column-scan mode

Ascending by 1 (row-scan)	Ascending by 4 (column-scan)	Up-scrambler [A[1:0], A[3:2]]	Down-scrambler
0000	0000	0000	0000
0001	0100	0001	0100
0010	1000	0010	1000
0100	0001	0100	0001
0101	0101	0101	0101
0110	1001	0110	1001
1000	0010	1000	0010
1001	0110	1001	0110
1010	1010	1010	1010
1100	0011	1100	0011
1101	0111	1101	0111
1110	1011	1110	1011

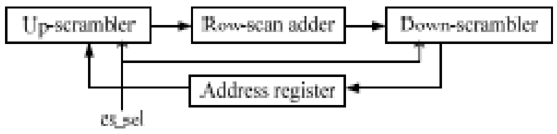

Fig. 5 Column-scan mode supported by row-scan adder

To continuously perform the operation to each address in column-scan way, it encounters two special cases due to overflow of address-up/down counting. TABLE III lists an example for 8-bit addressing, where the column Mux is 4 and the maximal address is 199. In the up-counting case, if the column-scan addressing exceeds the maximum address, the word-line address is reset to 0 and the bit-line address is switched to the next column cell. For the down-counting case, the maximum word-line address should be st to hi-bit part to deal with the overflow case. The overall block diagram of address generator is shown in Fig. 6. The path of column scan and row scan are highlighted. Finally , by the control signals, fail_h and EOC, address counter is held while dumping failure data (i.e. fail_h) and executed continuous operations in the same March element (i.e. EOC). A built-in look-up table is required to store the maximal address and column MUX number of all under-tested memories.

TABLE III. Overflow and normal cases for column-scan mode

Condition	inc_A_cs
(case 1) Overflow/up	{8'b00000000,inc_A[9:8]+1}
(case 2) Overflow/down	{max_addr[9:2],inc_A[9:8]}
(case 3)Normal/don't care	{inc_A[7:0],inc_A[9:8]}
Overflow condition: inc_A[7:0],inc_A[9:8]} > max_addr max_addr = 8'b11000111	

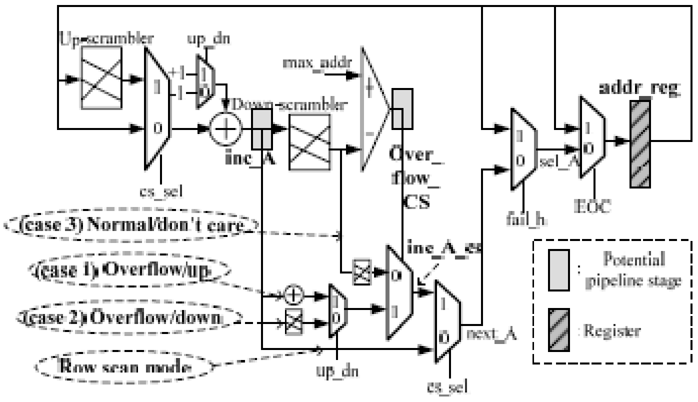

Fig. 6 Architecture of the proposed address generator

B. Timing diagram and controller design

Fig. 7 shows the general input and output timing diagram. Firstly, users should define the testing group (i.e. gp_sel) and specify the parallel or sequential test (i.e. ps_sel) for the chosen group. Then, the instruction is serially input from the signal, scan_in, while ins_se is enabled. After executing the current instruction, tst_done will go high and the next instruction can be read and so on.

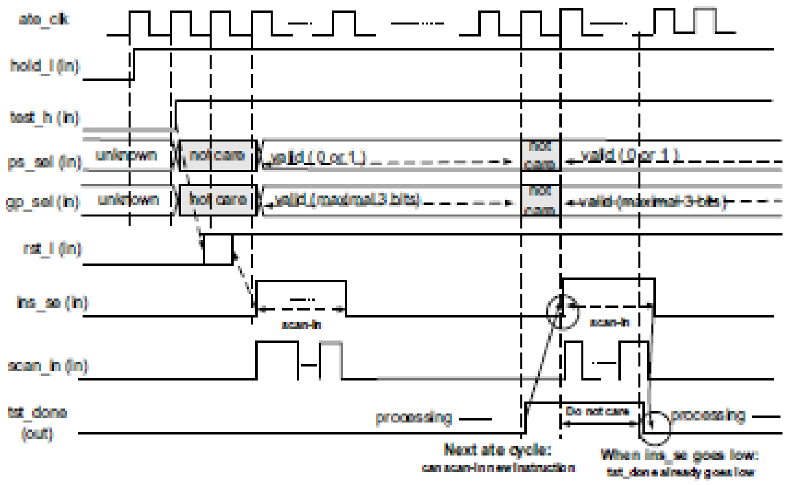

Fig. 7 General input and output timing diagram

Fig. 8 shows a simplified controller supporting 3 continuous March operations. Firstly, the external signal, rst_1, triggers the state and gets into wait state. Once the complete instruction has been input, it will go to March1 state. If the diagnosis mode is turned-on and the fault address is detected, the state will be held until all the failure information has been dumped out. Otherwise, there is no stall cycle while performing each March element.

C. Pipeline design issue

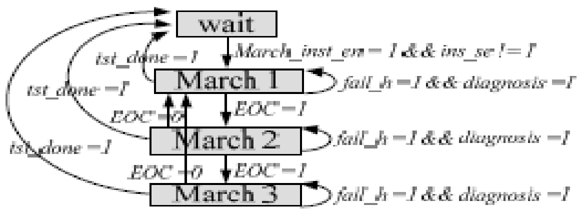

Fig. 8 Controller state for P-MBIST

For higher testing speed, address generator is a critical path because of the long delay of adder and comparator. To enhance the operating frequency, two pipeline stages are inserted to the address generator. The first stage is used to put to the bus of "inc_A" and the second stage is placed to the "Over_flow_CS" signal (i.e. refer to the potential pipeline stage in fir. 6). Fig. 9 illustrates the timing relation between pipelining address generator and two inserted pipeline states, pipe_state1 and pipe_state2. First, the register, inc_A_reg, is determined after one delay cycle of addr_reg. Afterward, Over_flow_CS_reg can be determined and the addr_reg can be further determined. Based on this design method, the operating frequency could achieve high testing-speed case for typical memory application (i.e. 0.13micro meter with 14-bit adder at 500MHz).

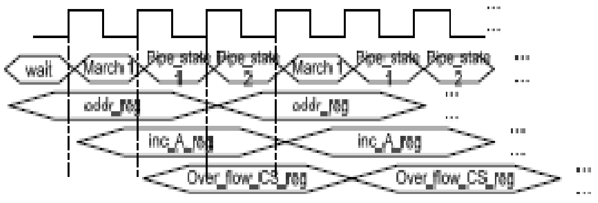

Fig. 9 Timing diagram of pipeline design

IV. AUTOMATION FLOW OF P-MBIST

For the automation flow, we use Tool Command Language (TCL) to develop a software program, which can generate the P-BIST design, according to the user-defined configuration file. As shown in TABLE IV, one can define the numbers of groups, numbers of March elements, pipeline mode, and numbers of memory instances in each group. The detailed memory information should be also provided. As described in TABLE V, it gives the detailed information of the first instance in group TP0. Based on this configuration file, 2-bit address counter and 8-bit data_bg are generated. Three March operations can be programmed. The address counter and controller can be shared to different groups, to reduce the hardware overhead.

TABLE IV. Configuration file for generating P-MBIST design

TCL command	P-MBIST Configurations
set_pmbist_mem_group	{TP0 ROM0}
set_max_march_set	3
set_pipeline	off
set_group_port_type TP0 two_port set_pmbist_group_mem TP0 {TP0_M0 TP0_M1} set_pmbist_group_mem_data TP0 TP0_M0 **0 10 1023 8 8 1** set_pmbist_group_mem_data TP0 TP0_M1 1 12 4095 8 3 1	
set_group_port_type ROM0 ROM set_pmbist_group_mem ROM0 {ROM0} set_pmbist_group_mem_data ROM0 ROM0 0 10 1023 1 32 1	

TABLE V Detailed descruption of given memory information

Memory index	Address bit	Maximal address	Column MUX	Data bit	Byte write
0	**10**	**1023**	**8**	**8**	**1**

V. COMPARISIONS

To evaluate the proposed P-MBIST architecture, we compare the proposed P-MBIST with Mentor's benchmark under the same condition. TABLE VI describes the cell area condition; it uses two individual controllers to support parallel and sequential testing. Moreover, the MBIST cannot support different memory types by using the same controller. Thus, it requires individual BIST circuit for each memory type.

TABLE VI. Several cases of Mentor's cell area for low-speed case (Memory size: 1024x8 bit, UMC 0.13 μm, Algorithms: March C+, March C-, and Checkerboard)

#of instance / type	Parallel	Sequential	**Par./Seq.**
2 / single-port	1440	1607	**3034**
2 / dual-port	1774	1925	3703
2 / two-port	1469	1636	3055
1 / ROM	N/A	999	999
2 /single-port, 2/two-port, 2/dual-port			**9790**
2 /single-port, 2/two-port, 2/dual-port + 1/ROM			**10788**

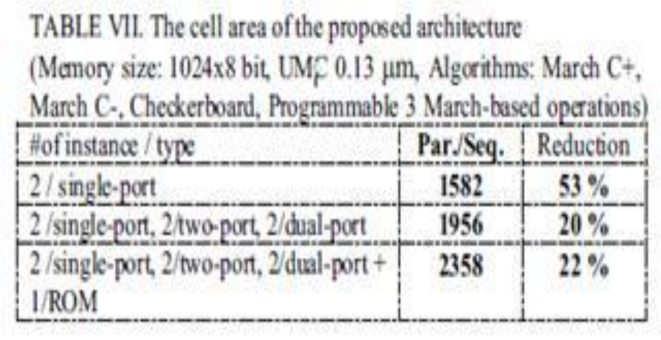

TABLE VII. The cell area of the proposed architecture
(Memory size: 1024x8 bit, UMC 0.13 µm, Algorithms: March C+, March C-, Checkerboard, Programmable 3 March-based operations)

#of instance / type	Par./Seq.	Reduction
2 / single-port	**1582**	**53 %**
2 /single-port, 2/two-port, 2/dual-port	**1956**	**20 %**
2 /single-port, 2/two-port, 2/dual-port + 1/ROM	**2358**	**22 %**

Compared with the proposed architecture, we use the same address counter for all memory instances and share the controller for each group. As shown in TABLE VII, the proposed P-MBIST circuit uses 1582 gates for testing two single-port instances. While extending to three groups with different memory types, it requires additional 374 gates. Furthermore, to support ROM testing with Multi Input Shift Register (MISR) algorithm, it increases 403 gate counts. By applying the proposed hardware sharing method, it can significantly reduce the hardware cost.

VI. CONCLUSIONS

In this work, we propose an efficient architecture for P-MBIST circuit. Based on the proposed hardware sharing method, we use the common address counter to provide all memory instances, including row and column scan addressing mode. Moreover, the controller can be extended to different memory types in the same read/write cycle condition, without increasing any state. Thus, the hardware cost can be greatly reduced. Finally the proposed P-MBIST circuit can be generated from the user defined configuration file.

REFERENCES

[1]. S.Boutabza, .M. Nicholaidis K.M. Lamara and A. Costa, "Programmable memory BIST," in Proc. ITC, pp: 45.2, Nov 2005.

[2]. I. Bayraktaroglu , 0. Caty and W. Yickkei, "Highly configurable programmable built-in self test architecture for high-speed memories" in proc. VLSI Test Symposium. pp: 21

[3]. D. Xiaogang. N. Mukherjee. C Wu-Tung S.M. Reddy. 'Full-speed field-programmable memory BIST architecture in Proc. ITC pp: 45.3. Nov. 2005.

[4]. D. Appello, V. Tancorre, P. Bernadi, M.Grosso, M. Rebaudengo and M.S. Reads, 'Embedded Memory Diagnosis: An Industrial Workflow: in Proc. ITC, pp: l -9, Oct. 2006.

[5]. S. Boutabza, .M. Nicholaidis K.M. Lamara and A. Costa, "A Transparent based Programmable Memory BIST in Proc. IEEE European Test Symposium. pp. 89-96. May 2006.

[6]. M. Miyazaki. T. Yoneda and H. Fujiwam. "A memory grouping method for sharing memory BIST logic,' in Proc. Asia and South Pacific Design Automation Conference. pp. 671-676. Jan. 2006.

[7]. Basic VLSI Design by Douglas A. Pucknell, Kamran Eshraghian, Prentice Hall, 1994

[8]. http://www.mentor.com