IEICE TRANS. INF. & SYST., VOL.E92–D, NO.12 DECEMBER 2009

LETTER

An Effective Programmable Memory BIST for Embedded Memory

Youngkyu PARK[†], **Jaeseok PARK**[†], **Taewoo HAN**[†], ***Nonmembers***, ***and*** **Sungho KANG**[†a)], ***Member***

SUMMARY This paper proposes a micro-code based Programmable Memory BIST (PMBIST) architecture that can support various kinds of test algorithms. The proposed Non-linear PMBIST (NPMBIST) guarantees high flexibility and high fault coverage using not only March algorithms but also non-linear algorithms such as Walking and Galloping. This NPMBIST has an optimized hardware overhead, since algorithms can be implemented with the minimum bits by the optimized instructions. Finally, various and complex algorithms can be run thanks to its support of multi-loop.

key words: *Programmable BIST, test algorithm, multi-loop*

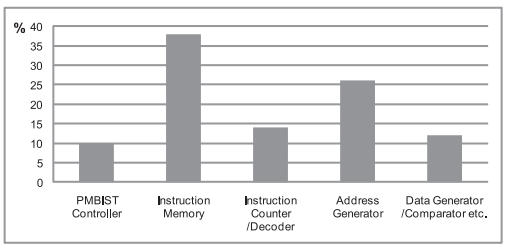

Fig. 1 Hardware overhead ratio of the PMBIST components.

1. Introduction

Diverse cores are being systemized on chips due to the recent technical development of semiconductor process and design. As the weight of embedded memory within aggregate Systems-On-Chips (SoC) gradually increases to 80–90% of the number of total transistors, the importance of testing embedded memory in SoC increases. The most prevalent method for testing embedded memory is the Memory Built-In Self-Test (BIST) [1].

The Memory BIST method has advantages in that the complexity of testing whole systems decreases remarkably because self-tests are performed for each module. Moreover, it can conduct tests quickly without utilizing expensive external test equipment. At the same time, however, the Memory BIST has disadvantages such as low flexibility and limited fault coverage, because it only supports limited test algorithms. Flexibility has improved somewhat in the Programmable Memory BIST (PMBIST) method because PMBIST supports numerous March test algorithms to complement the shortcomings of the Memory BIST method. However, PMBIST also has the disadvantages of limited fault coverage and high hardware overhead.

Figure 1 shows the hardware overhead ratio for each of the PMBIST components. It was found that the instruction memory and the address generator hold 68% of the entire hardware overhead of the PMBIST.

This paper proposes a Non-linear PMBIST (NPMBIST) architecture applicable to diverse test algorithms for the efficient testing of various embedded memories. It is difficult to test the memories embedded in highly integrated system semiconductors using only established March algorithms. Therefore, efficient testing can be implemented only if pertinent test algorithms are applicable according to the architecture design and memory usage. The proposed NPMBIST architecture can support March test algorithms, which are widely used, and non-linear test algorithms (Diagonal, Galloping, etc.). Furthermore, the proposed architecture can embed numerous test algorithms. Therefore, the flexibility, fault coverage, and credibility of embedded memories can be enhanced, because users can choose pertinent test algorithms and perform tests. On top of that, the most optimized instruction architecture is proposed to minimize the size of the instruction memory, which occupies 38% of the hardware overhead of the PMBIST. Furthermore, the hardware overhead is minimized by suggesting the address generator that enables multi-loop and variable counting.

2. Pattern Instruction Architecture

The proposed NPMBIST retains the BIST architecture of the micro-code that supports diverse test algorithms. The NPMBIST supports not only all the March test algorithms such as MATS+, March C−, March X, March A, March LR, and March SS, but also non-linear test algorithms such as Walking, Galloping, Diagonal, and Butterfly. This proposed NPMBIST suggests an effective architecture of pattern generation instruction that supports all kinds of algorithms.

The proposed NPMBIST saves test algorithms in the form of instructions in the internal instruction memory, and implements algorithms using saved pattern generation instructions. To implement algorithms, the size of the pattern generation instruction is defined as 9 bits. Table 1 shows each bit of the pattern generation instruction.

The pattern generation instruction has an architecture that effectively supports March and non-linear test algo-

Manuscript received May 25, 2009.
Manuscript revised August 12, 2009.

[†]The authors are with the Department of Electrical and Electronic Engineering, Yonsei University, Korea.
a) E-mail: shkang@yonsei.ac.kr

DOI: 10.1587/transinf.E92.D.2508

Table 1 Pattern generation instruction.

Instruction set	Inst[8:7]		Inst[6]		Inst[5]		Inst[4]		Inst[3]		Inst[2]		Inst[1:0]	
Function	Hold/ Increment		Address up/down		Background true/inv.		Memory operation		Counter operation		Counter output		Control	
Opcode	00	hold	0	up	0	true	0	read	0	A	0	A	00	NOP
	01	inc.	1	down	1	inv.	1	write	1	B	1	B	01	B ← A+1
	10	branch											10	A ← B+1
													11	terminate

rithms with the minimum number of bits. The pattern generation instruction supports multi-loop to implement non-linear algorithms. The brief architecture of the pattern generation instruction is as follows.

The "*Hold/Increment*" field of Inst[8:7] directs the operation phase of the pattern generation instruction to implement algorithms. '*hold*' lets the current pattern generation instruction maintain its phase, and '*increment*' tells it to perform the next pattern generation instruction. Next, '*branch*' tells the current pattern generation instruction to jump to the assigned pattern generation instruction. The branch operation of the pattern generation instruction takes the instruction address of the branch register. The "*Address Up/Down*" field of Inst[6] points to the direction of increasing or decreasing sequence addresses of March algorithms and non-linear algorithms. The "*Background True/Inv.*" field of Inst[5] decides whether to invert the background data. The "*Memory operation*" field of Inst[4] designates read/write operation commands within sequences of March algorithms and non-linear algorithms. The "*Counter operation*" field of Inst[3] chooses either counter A or counter B as the counter used by the address generator when generating addresses of March algorithms and non-linear algorithms. The "*Counter output*" field of Inst[2] assigns the appropriate counter to send out addresses when March algorithms and non-linear algorithms generate addresses. Inst[3] and Inst[2] support multi-loop. Finally, the "*Control*" field of Inst[1:0] designates options for the pattern generation instruction. For example, if Inst[1:0]=01, it assigns the option for the branch operation. 'B ← A+1' means to put the value of 'counter A plus 1' into 'counter B'.

Pattern generation instruction is composed of one instruction per each march element of the March algorithm. For instance, a March operation composed of three march elements like ⇑(r1,w0,r0) can be materialized as three pattern generation instructions. Therefore, the March C− algorithm can be materialized as ten pattern generation instructions, and the Galloping algorithm is materialized as twelve instructions.

3. NPMBIST Architecture

The proposed NPMBIST architecture supports March algorithms as well as non-linear algorithms. The NPMBIST can contain up to eight desirable algorithms, and its architecture makes tests with specific algorithms possible. Specifically, NPMBIST deals with diverse faults thanks to its numerous algorithms, and therefore has high fault coverage. Furthermore, users can minimize the time required for tests because they can choose the necessary algorithms.

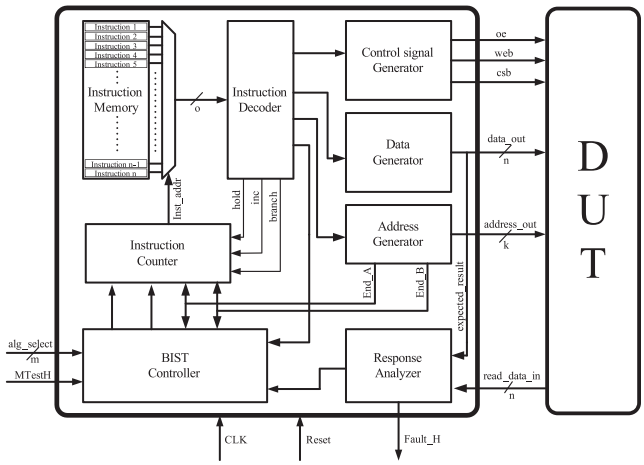

Fig. 2 NPMBIST architecture.

The proposed NPMBIST architecture consists of instruction memory, instruction counter, decoder, control signal generator, data generator, address generator, and response analyzer. Figure 2 shows the architecture of the proposed NPMBIST. To explain each block of the NPMBIST, the instruction memory saves the instruction of each test algorithm materialized as a proposed pattern generation instruction. The instruction counter directs the instruction memory to choose a saved pattern generation instruction, and the decoder decodes the instruction generated by the instruction memory. The control signal generator generates control signals put into the memory, and the data generator generates test patterns put into the memory. The address generator generates addresses put into the memory, and the response analyzer compares the results of tests and evaluates the existence of defects. Finally, the BIST controller receives test algorithms from users and controls the operation of the NPMBIST on the whole.

The NPMBIST supports 2-level nested loop to implement non-linear test algorithms. The address generator is composed of two counters to support 2-level nested loop. Since the address generator can generate complex patterns, it can generate various addresses. Like Butterfly algorithm, it effectively generates the pattern that tests victim cells, increasing the address by 1 or 2 to the up-down and the left-right side of the aggressor cell. Figure 3 shows the address generator that supports 2-level nested loop. The address generator of Fig. 3 is composed of two counters and

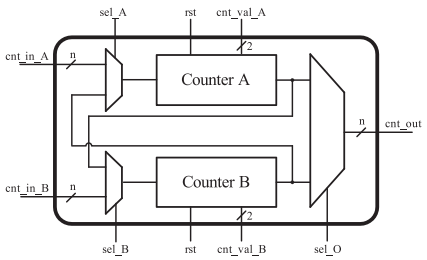

Fig. 3 Address generator.

three multiplexers, and counting is implemented by using the counter outputs of each. Furthermore, various address values can be generated because the unit of count varies from one to three. The proposed address generator can support various algorithms, so complex patterns can be generated.

The proposed NPMBIST deals with diverse faults thanks to its numerous algorithms, and therefore has high fault coverage and high flexibility. Furthermore, users can minimize the time required for testing because they can choose the necessary algorithms.

4. NPMBIST Implementation and Results

In this paper, the NPMBIST was designed to embed MATS+, March C−, March SS, and Galloping algorithms, in order to verify the architecture of NPMBIST. And this NPMBIST was fabricated in the form of a chip. The memory being tested has a 9-bit row address, a 4-bit column address, a 32-bit data word, and single port synchronous SRAM. We also performed a functional simulation using Modelsim by Mentor Graphics, and verified the synthesis of NPMBIST using Design Complier by Synopsys.

Figure 4 shows the materialization of the Galloping algorithm with the proposed pattern generation instruction. The operations are explained briefly as follows. Instruction no.1 operates 'write 0' on all cells of the memory. Instruction no.2 writes '1' at the first address of the memory. Each of the instructions from no.3 to no.5 reads '0', '1', and '0' in sequence, increasing the addresses of the memory in a specific pattern. Instructions from no.3 to no.5 repeat the 'read' operation until the last address of the memory. Then, the branch operation moving from instruction no.5 to no.3 refers to the branch register value. After that, instruction no.6 writes '0' on the cell in which '1' is written at the operation of instruction no.2. Then, instruction no.2 operates. The instructions from no.2 to no.6 are repeated until the last address of counters A and B is generated. Then the instructions from no.7 to no.12 repeat the operations in reverse.

Figure 5 shows a die photograph of the NPMBIST chip. The NPMBIST with the March algorithms and Galloping algorithm inside was fabricated in the form of a chip through the Samsung 0.35 μm process. The service voltage of NPMBIST ranges from −3.3 V to +3.3 V, and the maximum frequency is 300 MHz.

Table 2 is the synthesis results of NPMBIST using

Instruction #	Hold	Increment	Branch	Addr up	Addr down	Data true	Data invert	Mem opt(read)	Mem opt(write)	Counter opt(A counter)	Counter opt(B counter)	Counter out(A counter)	Counter out(B counter)	Control(NOP)	Control(A←B+1)	Control(B←A+1)	Control(terminate)
	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1
	0	1	0											0	1	0	1
1	0	0	0	0		0		1		0		0		0	0	0	0
2	1	0	0	0		1		1		1		0		0	1	0	0
3	1	0	0	0		0		0		1		1		0	0	0	0
4	1	0	0	0		1		0		0		0		0	0	0	0
5	1	0	0	0		0		0		1		1		0	0	0	0
6	1	0	1	0		0		1		0		0		0	0	0	0
7	0	0	0	0		1		1		0		0		0	0	0	0
8	1	0	0	0		0		1		1		0		0	1	0	0
9	1	0	0	0		1		0		1		1		0	0	0	0
10	1	0	0	0		0		0		0		0		0	0	0	0
11	1	0	0	0		1		0		1		1		0	0	0	0
12	1	0	1	0		0		1		0		0		1	0	0	1

Fig. 4 Pattern generation instruction of the Galloping algorithm.

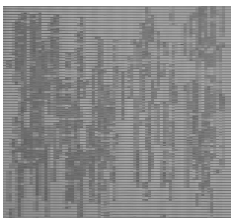

Fig. 5 Die photograph of NPMBIST.

Table 2 Results of synthesis.

	Instruction bits	Hardware overhead	Max operation speed
1 algorithm	126 bits	6,427 gates	300 MHz
4 algorithms	441 bits	11,462 gates	
8 algorithms	891 bits	20,530 gates	

a commercial synthesis tool (Design Complier by Synopsys). The process technology used for synthesis was the TSMC 0.13 μm library. The synthesis results of NPMBIST with four embedded algorithms (MATS+, March C−, March SS, and Galloping) indicates that the hardware overhead is 11,462 gates, and the maximum frequency is 300 MHz. The hardware overhead follows the standard of the 2-input nand gate. The synthesis result of NPMBIST that embeds only the March C+(14n) algorithm showed that the hardware overhead is 6,427 gates. The NPMBIST can embed a maximum of eight test algorithms, and the hardware overhead depends on the number of algorithms in the instruction memory.

5. Performance Evaluation

Table 3 shows a comparison between pre-proposed programmable memory with the BIST architecture and the proposed NPMBIST. The standards of comparison were supporting algorithms, instruction size, and instruction bits required to implement a traditional March 2(=March C+, 14n) algorithm. The area of the architecture were also compared. We have implemented the NPMBIST with the same March 2 algorithm, 16K × 16 memory size and 0.13 μm technology library for fair comparison with previous PMBISTs in Table 3.

The PMBIST in [2] and [3] only support March al-

Table 3 Comparison with previous PMBIST.

	[2]	[3]	[4]	[5]	NPMBIST
March algorithms	Y	Y	Y	Y	Y
Galloping /Walking	N	N	Y	Y	Y
Butterfly	N	N	Y	Y	Y
Diagonal	N	N	Y	Y	Y
Multi-loop	N	N	N	Y	Y
Instruction size (bits)	4	52	9	8	9
Instruction bits March 2 (bits)	160	312	144	216	126
Area (gates)	7.9K	NA	7.0K	12.4K	6.4K

gorithms and requires many instruction bits when implementing March 2 algorithm. The PMBIST in [4] can support March algorithms and non-linear algorithms. However it cannot support multi-loop. It can only use the "special multi-cycle instructions" in order to support multi-loop. And the PMBIST in [5] supports March algorithms, non-linear algorithms, and multi-loops, but requires lots of instruction bits when implementing March 2 algorithm, incurring defects of huge hardware overhead. This structure requires a lot of instruction bits, because it uses several instructions to support the multi-loop. The PMBIST in [4] supports only 2 loops but has small hardware overhead, while the PMBIST proposed in [5] supports multi-loops but has huge hardware overhead. The area overhead of the PMBIST structure in [4] is about 7.0K gates. And the PMBIST in [5] contain diagnostic logic which is about 1.2K gates. Therefore, Table 3 shows the comparison result of subtracting 1.2K gates from the area overhead of the PMBIST structure in [5]. Additionally, the PMBISTs in [4] and [5] have dynamic loading circuits that load the instruction to the instruction memory. The proposed NPMBIST also includes the dynamic loading circuits that load the instruction from ATE to the NPMBIST. In these circuits, the instruction memory occupies the most part of the hardware overhead, whereas other control circuits contain very small hardware overhead.

The proposed NPMBIST supports all March algorithms and non-linear algorithms. It retains address instructions that can implement all kinds of algorithms with a minimum number of bits and an address generator that supports multi-loop. Furthermore, the NPMBIST has an instruction size of 9 bits. The March 2 algorithm can be implemented with smaller instruction bits than previous architectures. Moreover, the hardware overhead of NPMBIST is 6.4K, which is small in comparison to that of all the previous PMBIST.

For the area overhead of PMBIST, the instruction memory and the address generator account for more than 38% and about 26%, respectively. The proposed NPMBIST minimized not only the size of instruction memory by the test algorithms through the optimum instruction structure, but also the area overhead by the architecture of the optimum address generator that supports multi-loop. Therefore, NPMBIST has the advantages of high flexibility, high fault coverage, and low area overhead.

6. Conclusions

The NPMBIST proposed in this paper supports March algorithms and non-linear algorithms, such as Walking and Galloping, and therefore has high flexibility and high fault coverage. This architecture has an optimum hardware overhead, since the implementation of algorithms is possible with a minimum of instruction bits. Also, through multi-loop, the NPMBIST architecture can support complex algorithms that contain unfixed data and address patterns.

Acknowledgments

This work was supported by "System IC 2010" project of Korea Ministry of Knowledge Economy.

References

[1] W.-L. Wang, K.-J. Lee, and J.-F. Wang, "An on-chip march pattern generator for testing embedded memory cores," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.9, no.5, pp.730–735, Oct. 2001.

[2] D. Appello, P. Bernardi, A. Fudoli, M. Rebaudengo, M.S. Reorda, V. Tancorre, and M. Violante, "Exploiting programmable BIST for the diagnosis of embedded memory cores," Proc. IEEE International Test Conference, pp.379–385, Oct. 2003.

[3] S. Boutobza, M. Nicolaidis, K.M. Lamara, and A. Costa, "Programmable memory BIST," Proc. IEEE International Test Conference, Paper 45.2, Nov. 2005.

[4] X. Du, N. Mukherjee, W.-T. Cheng, and S. Reddy, "Full-speed field-programmable memory BIST architecture," Proc. IEEE International Test Conference, Paper 45.3, Nov. 2005.

[5] X. Du, N. Mukherjee, C. Hill, W.-T. Cheng, and S. Reddy, "A field programmable memory BIST architecture supporting algorithms with multiple nested loops," Proc. IEEE Asian Test Symposium, pp.287–292, Nov. 2006.