A Flexible Programmable Memory BIST for Embedded Single-Port Memory and Dual-Port Memory

Youngkyu Park, Hong-Sik Kim, Inhyuk Choi, and Sungho Kang

Programmable memory built-in self-test (PMBIST) is an attractive approach for testing embedded memory. However, the main difficulties of the previous works are the large area overhead and low flexibility. To overcome these problems, a new flexible PMBIST (FPMBIST) architecture that can test both single-port memory and dual-port memory using various test algorithms is proposed. In the FPMBIST, a new instruction set is developed to minimize the FPMBIST area overhead and to maximize the flexibility. In addition, FPMBIST includes a diagnostic scheme that can improve the yield by supporting three types of diagnostic methods for repair and diagnosis. The experiment results show that the proposed FPMBIST has small area overhead despite the fact that it supports various test algorithms, thus having high flexibility.

Keywords: Built-in self-test, BIST, programmable memory BIST, PMBIST, flexible PMBIST, single-port memory, dual-port memory.

Manuscript received Oct. 20, 2012; revised Mar. 6, 2013; accepted Mar. 21, 2013.

This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the CITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-1005) supervised by the NIPA (National IT Industry Promotion Agency).

Youngkyu Park (phone: +82 2 2123 2775, yk76.park@samsung.com), Inhyuk Choi (ihchoi@soc.yonsei.ac.kr), and Sungho Kang (corresponding author, shkang@yonsei.ac.kr) are with the Department of Electrical & Electronic Engineering, Yonsei University, Seoul, Rep. of Korea.

Hong-Sik Kim (hongsik1.kim@sk.com) is with the Advanced Design Group, R&D, SK Hynix Inc., Icheon, Rep. of Korea.

http://dx.doi.org/10.4218/etrij.13.0112.0717

I. Introduction

Advances in semiconductor technology and reuse design methodologies have enabled many intellectual property cores, such as processor cores, large capacity embedded memory, and mixed signal analog cores, to be integrated on a single chip. Such integration is referred to as the system on chip (SoC) design methodology. In current SoC products, embedded memory cores occupy most of the chip area and thus have become a key factor in the reliability of SoC devices. In addition, the limited bandwidth of external automated test equipment (ATE) makes the testing of embedded memory cores more difficult. Therefore, embedded memory core testing in the SoC design environment has become a very important and time-consuming task [1]-[3].

Single-port memory and dual-port memory are widely used as embedded memory in SoC products. In single-port memory, the data transactions are conducted through a single IO port, whereas in dual-port memory, two different data transactions can be simultaneously performed through two independent IO ports. As such, the fault models and test algorithms of single-port memory are different from those of dual-port memory. However, most of the research in the field of embedded memory testing has been devoted to single-port memory [4]-[6].

The memory built-in self-test (BIST) technique is widely regarded as an optimal embedded memory test in the SoC design environment since it can enable an at-speed test with reduced IO channel test bandwidth [7], [8]. However, previously developed memory BIST methods only support a limited test based on the March algorithm and the restrictive non-March algorithm, and, as a result, the flexibility and fault coverage are restricted. Programmable memory BIST (PMBIST) methodologies have been proposed to enhance the flexibility of conventional memory BIST techniques and to test diverse fault models [9]-[11]. Flexibility in PMBIST is defined as how various test algorithms can be supported and is one of the main concerns for PMBIST. PMBIST techniques can improve the flexibility of conventional memory BIST techniques to accommodate various test algorithms. However, these techniques increase the area overhead significantly.

The majority of research in the field of embedded memory testing has focused on single-port memory. However, research regarding PMBIST architecture for dual-port memory is also important. PMBIST has tradeoffs between its fault coverage and area overhead. This is because supporting various test algorithms requires a large area overhead, and, if the number of supported test algorithms is limited, the area overhead shrinks but the fault coverage also decreases. Thus, a PMBIST architecture for both single-port memory and dual-port memory is needed. The PMBIST should be able to support various test algorithms with a small area overhead. To improve the memory yield, a diagnostic test is also needed. The diagnostic data is generated by the BIST circuit.

In this work, a new microcode-based PMBIST architecture is proposed to test both single-port memory and dual-port memory. The proposed flexible PMBIST (FPMBIST) supports March-based algorithms, non-March algorithms, and dual-port memory test algorithms. That is, all fault models for single-port memory and dual-port memory can be addressed with a single BIST architecture. Such a scheme allows for the grouping of single-port memory and dual-port memory into the same BIST domain so that the total BIST area can be reduced and so that simple test scheduling is possible. An optimized instruction set is also proposed in this study to reduce the area overhead of the programmable BIST circuitry. Furthermore, the FPMBIST design supports three types of diagnostic methods. With this diagnostic scheme, the FPMBIST design can increase the memory yield.

II. Previous Works on PMBIST Architecture

Generally, PMBIST can be classified into one of two types: finite state machine (FSM)-based and microcode-based. FSM-based PMBIST allows the ATE to choose a test algorithm from several predetermined FSM components of test algorithms. This method usually has low test complexity and small area overhead but has relatively low flexibility and fault coverage. On the other hand, the microcode-based PMBIST is the most widely used approach. The microcode-based PMBIST saves the test algorithm in the form of instructions in internal storage and implements the test algorithm using the saved instructions. This method of PMBIST has high flexibility and fault coverage, but the area overhead increases significantly.

A variety of architectures have been proposed for PMBIST design. The architecture in [12] can support five groups of test algorithms: March, Galloping/Walking, the test for the address decoder open, butterfly, and sliding diagonal. The base loop and local loop are also used to support non-March algorithms, but only a two-level loop is supported. As a result, the programming flexibility is restricted. The scheme in [13] upgrades the programmable BIST architecture of [12] by changing the instruction set to support multi-loop performance, and the architecture also supports a diagnostics scheme. However, the area overhead is significantly increased.

The nonlinear PMBIST (NPMBIST) architecture in [14] was proposed to effectively test single-port memory and supports both March and non-March algorithms. The NPMBIST uses an optimized instruction set to minimize the area overhead and supports multi-loop performance to ensure the effective implementation of complicated test algorithms. However, NPMBIST does not allow algorithm downloading from external ATE and cannot support such non-March algorithms as data retention tests, butterfly, and so on. NPMBIST can only generate limited complicated addresses using the fast diagonal method.

The architecture in [15] is a microcode-based programmable BIST scheme for dual-port memory testing. The test program set consists of 4-bit test primitives with which the microprogram for the target test algorithm is coded. The microprogram is stored in the μprogram memory. The ROM or an In-System Programmable module is used for the μprogram memory. This scheme can partially support multi-loop performance. The scheme of [15] can only access the same cell through two different ports. As such, only one port fault can be tested.

The architecture in [16] is a BIST scheme for dual-port memory using the same interface as the standard 1149.1 test access port. This scheme tests the dual-port memory by generating test patterns with transformed test algorithms from a March-based test algorithm, such as March C– or March A. In this case, only the March-based test algorithm can be materialized without multi-loop support, so the programming flexibility is limited. Furthermore, it cannot detect 2PF2av faults [17] because it cannot write and read simultaneously at two different cells through two ports.

III. FPMBIST

In previous FSM-based MBIST schemes, fixed test patterns based on test algorithms (such as the March algorithm) were supported such that the fault coverage and flexibility were restricted, even with a smaller hardware area. In addition, different BIST circuitries were required for single-port memory and dual-port memory. Consequently, the total memory BIST circuit area was increased.

In this work, a new microcode-based FPMBIST architecture is proposed that supports various test algorithms and memory port types. The target test algorithm for the FPMBIST is programmed with microcodes based on the proposed instruction set and is stored in the internal memory. By decoding the microcode sequence, test patterns based on the target test algorithm are generated using the BIST logic. The FPMBIST can test both single-port memory and dual-port memory, so memory of the same IO-type can be grouped together into the same test domain. The total BIST area overhead can thus be reduced compared to conventional BIST schemes, and the embedded memory cores can be scheduled in a more flexible manner.

Figure 1 shows the operating sequence of the FPMBIST for single-port memory and dual-port memory. Initially, the FPMBIST is in the Start state. In the Load_CMD state, the commands are loaded from the outside by the Command Load (CMDL) signals. In the Shift_CMD state, the commands are shifted to the decoder by the MTestH signals. The FPMBIST has two instruction sets to implement the algorithms. Therefore, the CMD decoder consists of two types of decoding logic. If Test Algorithm Select (TAS)=0, the FPMBIST will be in the Decode_LIS state, and the CMD decoder will decode commands with the decoding logic for the linear instruction set. If TAS=1, the FPMBIST will be in the Decode_NLIS state, and the CMD decoder will decode commands with the decoding logic for the nonlinear instruction set. In the Pattern_Generation state, the test patterns are generated by controlling the test pattern generator (TPG) by using signals received from the CMD decoder. In the Target_Memory_Test state, the target memory is determined by the Test Memory Select (TMS) signals, and is then tested. In the Diagnostic state, the fault information detected by the memory test is transmitted to the external ATE. This state is operated in two modes by the Diagnostic data Mode Select (DMS) signals. The purpose of the Bypass mode is to support the faulty information. The purpose of the Fault Information for Repair (FIR) mode is to support the fault syndrome and addresses of faulty cells for the redundancy analysis, and the purpose of the Fault Information for Diagnosis (FID) mode is to support the diagnostic data for a fault analysis consisting of the fault syndrome, the addresses of faulty cells, and the pattern that detected the fault.

In previous dual-port memory BISTs, the testing was performed on the dual-port memory by applying the test

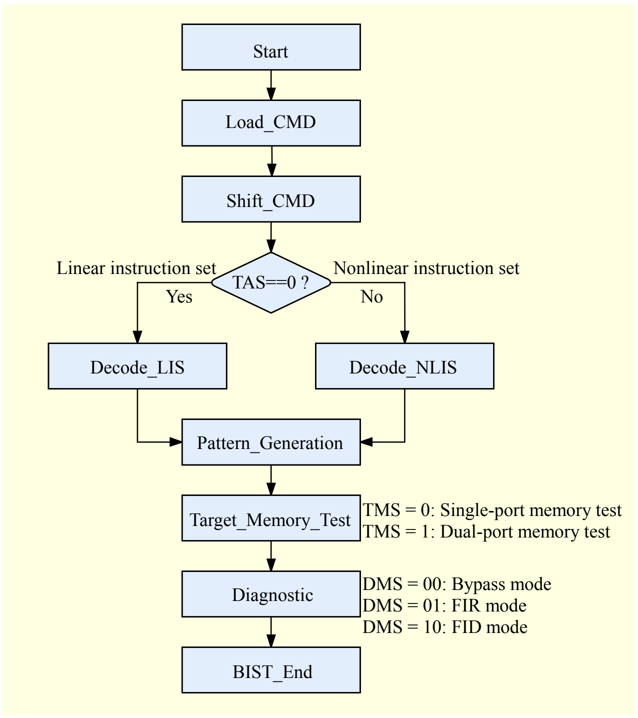


Fig. 1. Operating sequence of FPMBIST.

stimuli through one port and receiving the test responses through the other port. In addition, one cell could be simultaneously read and written through different ports. However, several fault models, such as 2PFav, cannot be tested using these schemes. The proposed FPMBIST supports conventional BIST operational modes and can simultaneously access two different cells through two different ports.

An instruction set is developed for the FPMBIST. A non-March algorithm, such as the Galloping and sliding diagonal algorithms, and a March-based algorithm can easily be microcoded by the instruction sets. The instructions support most of the memory test algorithms with little hardware complexity and allow the microcode size to be optimized. In addition, the FPMBIST supports a data retention test and improves memory yield with the diagnostic scheme.

1. Instruction Set Architecture

In the proposed FPMBIST, the target test algorithm is programmed into microcode by the instruction set, which is stored in the internal memory. The microcode sequence is decoded into the original test stimuli during the BIST test cycle. The proposed instruction set is divided into a linear instruction set and a nonlinear instruction set according to the target fault model. The nonlinear instruction set is used in the non-March-based test of one-port related fault models, and the linear instruction set is employed in the March-based test of one-port

Table 1. Nonlinear instruction set.

Instruction set	Inst[13:12]		Inst[11]		Inst[10]		Inst[9]		Inst[8]		Inst[7]		Inst[6]		Inst[5]		Inst[4:2]		Inst[1:0]	
Function	Instruction Control		Background True/Inversion		Memory Operation		Port Select		Column Output		Row Output		Column Counter		Row Counter		Counter Control		Operation Control	
Opcode	00	Increment	0	True	0	Read	0	A	0	CX	0	RX	0	CX	0	RX	000	NOP	00	NOP
	01	Branch	1	Inversion	1	Write	1	B	1	CY	1	RY	1	CY	1	RY	001	CNOP	01	Y←X
	10	RDBranch															010	^Row	10	X←Y
	11	Pause															011	^Column	11	Terminate
																	100	D		

and two-port fault models. The nonlinear instruction set can also support a simple dual-port memory test. The sizes of the linear and nonlinear instruction sets in this study are 9 bits and 14 bits, respectively. In addition, multi-loop performance is supported by the nonlinear instruction set to more easily implement the non-March algorithm. The architectural features of the nonlinear instruction set are shown in Table 1.

An Instruction Control field (Inst[13:12]) signifies the operational status so that the target test algorithm can be implemented. The Increment state allows the next instruction to be run, and the Branch state allows the program counter to jump to the destination instruction. The branch operation is performed through a branch register that also supports multi-loop performance. The Return Data inversion Branch (RDBranch) state is the command to rerun the test program through the inversion of the background data, and the Pause state is the command to hold the instruction during a set time to detect a retention fault. The Background True/Inversion field (Inst[11]) determines the background data type. If this bit is 0, the background data fields are filled with "0"; otherwise, they are filled with "1." The Memory Operation field (Inst[10]) defines the memory access type. If this field is "0," the read operation is performed; otherwise, the write operation is performed. The Port Select field (Inst[9]) selects the memory port to be accessed. This bit allows the non-March algorithm that is implemented by the nonlinear instruction set to be applied to the dual-port memory test. The Column Output field (Inst[8]) selects the memory test column address between column counter X and column counter Y. The Row Output field (Inst[7]) selects the memory test row address between row counter X and row counter Y. The Column Counter field (Inst[6]) activates the column address counter (incremented/ decremented) between column counter X and column counter Y for the generation of the memory test column address. The Row Counter field (Inst[5]) activates the row address counter for the generation of the memory test row address. The column address and the row address generate one address. The Inst[8],

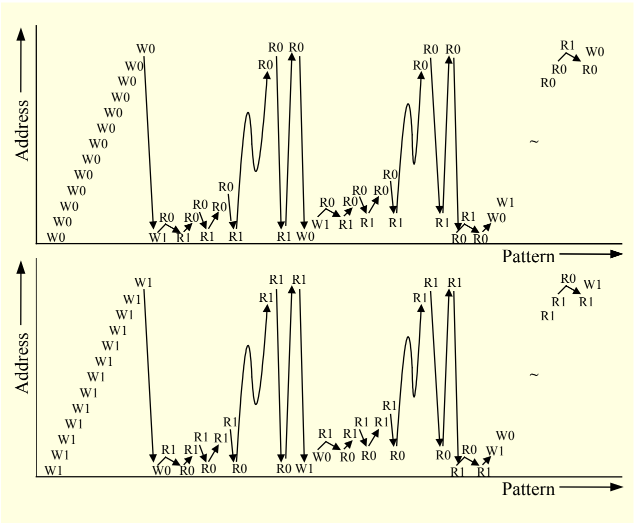
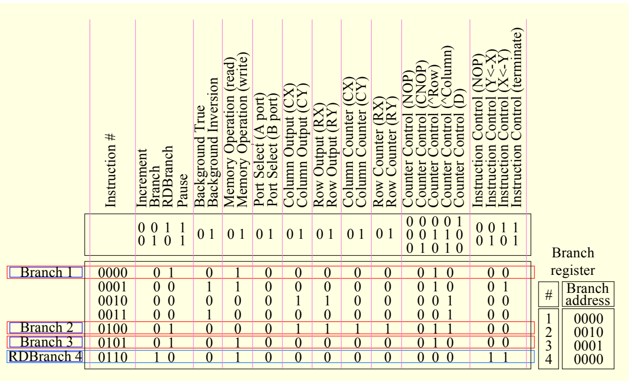


Fig. 2. Test patterns of Galloping algorithm.



	Instruction #	Increment / Branch / RDBranch / Pause	Background True / Background Inversion	Memory Operation (read) / Memory Operation (write)	Port Select (A port) / Port Select (B port)	Column Output (CX) / Column Output (CY)	Row Output (RX) / Row Output (RY)	Column Counter (CX) / Column Counter (CY)	Row Counter (RX) / Row Counter (RY)	Counter Control (NOP) / Counter Control (CNOP) / Counter Control (^Row) / Counter Control (^Column) / Counter Control (D)	Instruction Control (NOP) / Instruction Control (Y<-X) / Instruction Control (X<-Y) / Instruction Control (terminate)
		0 0 1 1 / 0 1 0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 0 0 0 1 / 0 0 1 1 0 / 0 1 0 1 0	0 0 1 1 / 0 1 0 1
Branch 1	0000	0 1	0	1	0	0	0	0	0	0 1 0	0 0
	0001	0 0	1	1	0	0	0	0	0	0 1 0	0 1
	0010	0 0	0	0	0	1	1	0	0	0 0 1	0 0
	0011	0 0	1	0	0	0	0	0	0	0 0 1	0 0
Branch 2	0100	0 1	0	0	0	1	1	1	1	0 1 1	0 0
Branch 3	0101	0 1	0	1	0	0	0	0	0	0 1 0	0 0
RDBranch 4	0110	1 0	0	1	0	0	0	0	0	0 0 0	1 1

Branch register

#	Branch address
1	0000
2	0010
3	0001
4	0000

Fig. 3. Materialization of Galloping algorithm using a nonlinear instruction set.

Inst[7], Inst[6], and Inst[5] sets are used for multi-loop performance so that a complex address sequence can be generated for the non-March algorithms. The Counter Control field (Inst[4:2]) designates the counter option. The NOP state means no operation. The CNOP state is an option for Inst[5] and Inst[4] to maintain the current value without counting the

Table 2. Linear instruction set.

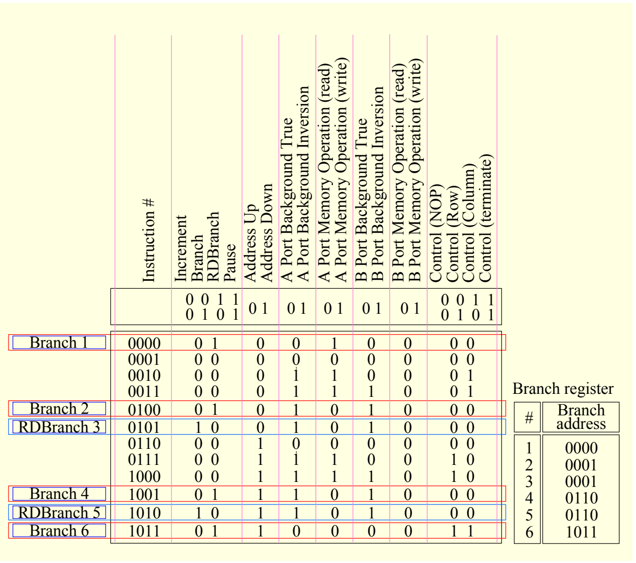
Instruction set	Inst[8:7]		Inst[6]		Inst[5]		Inst[4]		Inst[3]		Inst[2]		Inst[1:0]	
Function	Instruction Control		Address Up/Down		A Port Background True/Inversion		A Port Memory Operation		B Port Background True/Inversion		B Port Memory Operation		Operation Control	
Opcode	00	Increment	0	Up	0	True	0	Read	0	True	0	Read	00	NOP
	01	Branch	1	Down	1	Inversion	1	Write	1	Inversion	1	Write	01	Row
	10	RDBranch											10	Column
	11	Pause											11	Terminate

column address counter and the row address counter. The ^Row and ^Column options of Inst[4:2] are used to support the diagonal, fast row, and fast column methods. ^Row and ^Column are conditional arithmetic functions for fast diagonal addresses. The ^Row is a function for arithmetic operations that increases the row addresses infinitely. The ^Column is a function for arithmetic operations that increases the column addresses infinitely. The basic method of address generation is the fast diagonal method. The D states are the options to increase and decrease the address as much as the value saved at the D register when the address was generated. These options are used to materialize such algorithms as the butterfly algorithm. The Operation Control field (Inst[1:0]) designates the instruction option. The NOP state means no operation. The X←Y and Y←X states are the options to generate addresses by exchanging the values of the X counter and the Y counter.

By controlling the four counters, a complex address sequence based on the non-March algorithms can easily be generated. In addition, a simple dual-port memory test is supported through the port selection bit. Shown in Fig. 2 are test patterns based on the Galloping algorithm, a nonlinear March algorithm. A representative implementation of the Galloping algorithm via a nonlinear instruction set is shown in Fig. 3.

To implement the March-based test algorithms, the linear instruction set includes instructions with a fixed length of 9 bits. Dual-port memory tests based on the March algorithms are also supported through the linear instruction set. Therefore, various memory operations and features for each IO port can be implemented with the linear instruction set. The basic method that is used to generate the address of the linear instruction set is the fast diagonal method. The architecture and the features of the linear instruction set are shown in Table 2.

The Instruction Control field (Inst[8:7]) of the linear instruction set is equivalent to its counterpart (Inst[13:12]) in the nonlinear instruction set. The Address Up/Down field (Inst[6]) determines the direction of the address generation for the March and dual-port memory algorithms. The Background True/Inversion field (Inst[5]) determines the background data type for Port A. The A Port Memory Operation field (Inst[4]) defines the memory access type for Port A. If this field is “0,” a read operation is performed; otherwise, a write operation is performed. The Background True/Inversion field (Inst[3]) determines the background data type for Port B. The B Port Memory Operation field (Inst[2]) defines the memory access type for Port B. To test one-port faults, the Inst[3] and Inst[2] bits are fixed at a logic value of “0” so that the test generation for Port B can be deactivated. The Operation Control field (Inst[1:0]) signifies the instruction option that is generally used for the testing of dual-port memory. During the dual-port memory test, write operations are performed on a cell through one port, while read operations are performed on a neighboring cell in the same column (or the same row) through the other port. As an example, for Inst[1:0]=“10,” the port address for the read operation is generated in the adjacent direction of the column.



	Instruction #	Increment / Branch / RDBranch / Pause (00 / 01 / 10 / 11)	Address Up / Address Down (0 1)	A Port Background True / A Port Background Inversion (0 1)	A Port Memory Operation (read) / A Port Memory Operation (write) (0 1)	B Port Background True / B Port Background Inversion (0 1)	B Port Memory Operation (read) / B Port Memory Operation (write) (0 1)	Control (NOP) / Control (Row) / Control (Column) / Control (terminate) (00 / 01 / 10 / 11)
Branch 1	0000	0 1	0	0	1	0	0	0 0
	0001	0 0	0	0	0	0	0	0 0
	0010	0 0	0	1	1	0	0	0 1
	0011	0 0	0	1	1	1	0	0 1
Branch 2	0100	0 1	0	1	0	1	0	0 0
RDBranch 3	0101	1 0	0	1	0	1	0	0 0
	0110	0 0	1	0	0	0	0	0 0
	0111	0 0	1	1	1	0	0	1 0
	1000	0 0	1	1	1	1	0	1 0
Branch 4	1001	0 1	1	1	0	1	0	0 0
RDBranch 5	1010	1 0	1	1	0	1	0	0 0
Branch 6	1011	0 1	1	0	0	0	0	1 1

Branch register

#	Branch address
1	0000
2	0001
3	0001
4	0110
5	0110
6	1011

Fig. 4. Materialization of March A2PF algorithm using linear instruction set.

The linear instruction set uses a smaller number of bits than the previous instruction sets when implementing March-based algorithms. For example, in a March stage such as ↑(r1, w0, r0); or ↑(r0, w1, r1);, six March elements (r1, w0, r0, r0, w1, and r1) are implemented by six instructions of the previous instruction set, but the linear instruction set is implemented in four instructions. This is possible because of the RDBranch state in Inst[8:7] of the linear instruction set. Therefore, the March A2PF algorithm of 18*N* [18] is implemented with 12 instructions, involving a reduction of six instructions. Figure 4 shows the materialization of the March A2PF algorithm with the linear instruction set.

2. FPMBIST Architecture

In this study, a new advanced PMBIST architecture and test algorithm are proposed to test both single-port memory and dual-port memory. The FPMBIST supports algorithm downloading from the external ATE so that single-port memory and dual-port memory can be tested via TMS and TAS. A block diagram of the proposed FPMBIST design is shown in Fig. 5.

The FPMBIST architecture consists of the BIST controller (BCTR), TPG, and diagnostic data processing module (DPM) blocks. The BCTR block reads instruction codes from the instruction memory and conveys them to the TPG block. In addition, the BCTR controls the FPMBIST via the input ports, such as CMDL, TMS, TAS, DPS, Pause Time Select (PTS), and MTestH. The TPG block generates the test address, test data, and test control patterns based on the instruction codes and control signals from the BCTR block. The DPM block transforms the fault information that was obtained by conducting the memory tests into diagnostic data and provides it to the external ATE for repair or diagnosis.

The input signals of the FPMBIST include BIST Reset Signal (BRS), MTestH, TMS, CMDL, TAS, DMS, PTS, and CMD. The MTestH signal forces the FPMBIST to enter test mode. The CMDL signal forces the FPMBIST to enter program mode, where the instruction code sequence is stored in the internal instruction memory from the external ATE. The TMS signal conveys the memory type to be tested. If TMS is "0," the target is single-port memory; otherwise, the target is dual-port memory. The TAS signal determines the instruction set type of the current instructions. If TAS is "0," then the instructions belong to the linear instruction set for the March-based and dual-port memory test algorithm; otherwise, the instructions belong to the nonlinear instruction set for the non-March algorithm. Through the use of both TMS and TAS, the target memory port type and the test algorithm are determined. BRS is the reset signal for FPMBIST, while the CMD signal receives instruction codes from the ATE. The DMS signal is used to select the type of diagnostic data to be provided to the external ATE. The PTS signal is entered from the ATE for data retention tests. The BIST Finish Signal (BFS) is an output pin that is used to indicate that the BIST operation is finished. BIST Fault Out (BFO) is an output that is used to send the diagnostic data to the ATE.

As shown in Fig. 5, the BCTR block controls the FPMBIST operations and the flow of instructions. The BCTR block consists of a CMD block, CMD controller, and FPMBIST controller. The FPMBIST controller controls the FPMBIST based on the input signals from the external ATE and controls the ability of the CMD block to read and store the instructions from the ATE to the internal instruction memory during the program mode. The FPMBIST controller also controls each

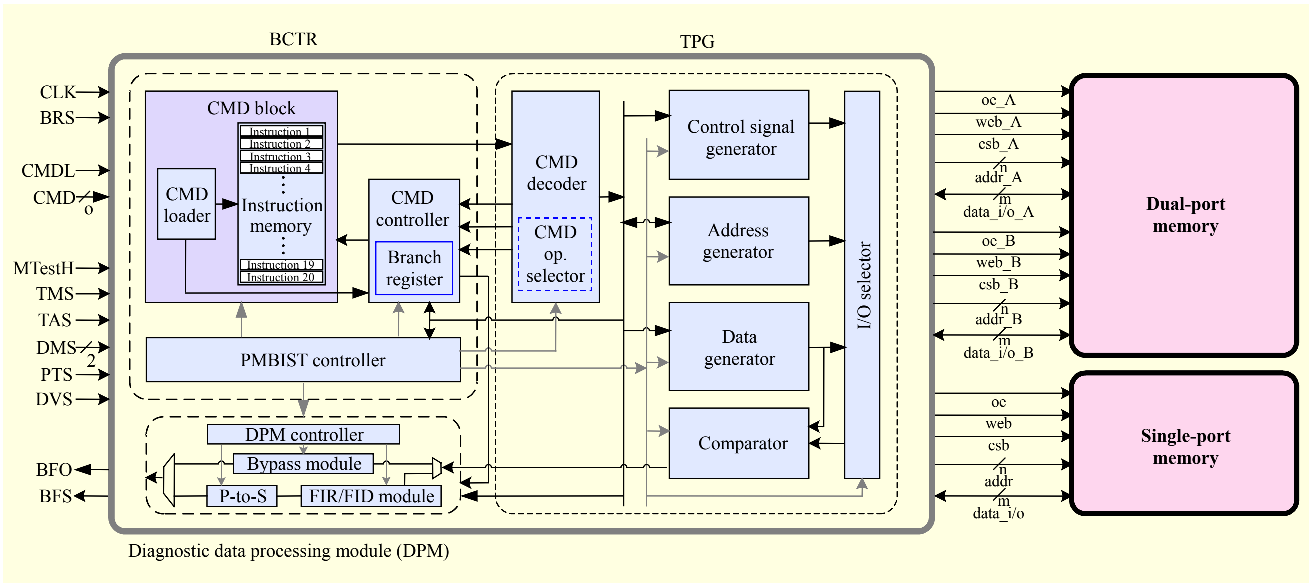


Fig. 5. FPMBIST architecture.

internal block so that test patterns can be generated during the test mode. The CMD block loads and stores the instruction codes from the external ATE to the internal instruction memory. The CMD controller reads and sends the instruction sequence from the instruction memory to the CMD decoder in the TPG block. The CMD controller has a branch register that stores branch addresses to effectively control the branches of the instructions. The instruction sets generate complex patterns by simply exploiting the multi-loops through the branch register.

The test patterns are generated in the TPG block and are applied to the large-capacity memory. A pass/fail test is then performed and the results are determined. The TPG block consists of a CMD decoder, control signal generator, address generator, data generator, comparator, and IO selector. The CMD decoder interprets the instructions from the instruction memory and generates the test pattern information for the test generators (control signal generator, address generator, and data generator). The instructions from the CMD block are received and interpreted by the CMD decoder, which includes a CMD operation selector circuit to determine the instruction type (linear and nonlinear instruction sets). The CMD decoder then interprets the instructions according to the corresponding instruction type. The control signal generator produces memory control signals, while the data generator generates memory test data based on the instruction decoding results. The address generator produces a test address, and the comparator determines the test results by comparing the test response data from the target memory with the test reference data from the data generator. The control signal, address, and data generators can generate both single-port memory and dual-port memory test patterns. The IO selector determines the output ports of the test patterns according to the target memory.

The TPG of the proposed FPMBIST includes an address generator to support diverse methods (fast diagonal, fast row, and fast column) for the complex address sequence generation of diverse algorithms. The address generator consists of two counters to support a two-level loop, as shown in Fig. 6. The address increment and decrement sequences are sufficient for most March algorithms. However, for non-March algorithms, the address sequences are generated through diverse loops; consequently, a single counter cannot support address generation. Therefore, the proposed address generator independently exploits two counters (the X and Y counters) to easily generate a complex address sequence. Each counter consists of a column counter and a row counter, and it includes a D register that can count to the specified value. The D register designates the values at the BIST controller. In addition, various counting offsets can be set by the ATE to generate diverse address sequences.

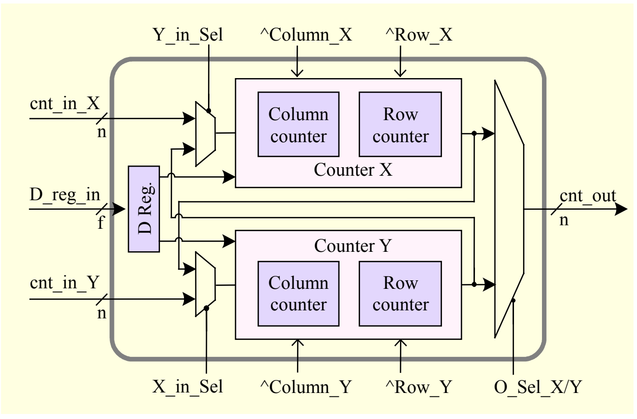


Fig. 6. Address generator.

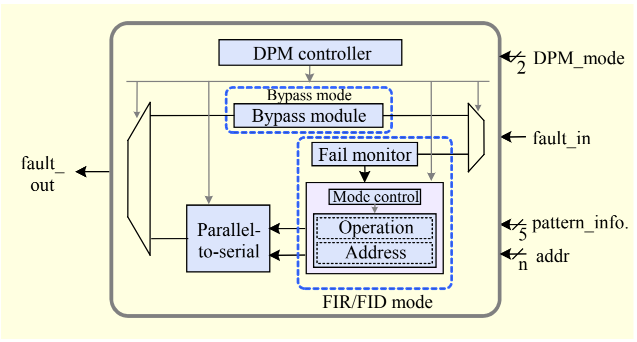


Fig. 7. Diagnostic data processing module.

Figure 7 shows the simplified block diagram of the DPM design. The DPM block generates the diagnostic data using the memory test results received from the TPG. The DPM block supports three types of diagnostic methods: bypass, FIR, and FID. The bypass method is a mode to support the fault information of 1 bit whether the memory is faulty or not. If a fault is detected, a “1” will be shown; if no fault is detected, a “0” will be shown. FIR is a mode that supports fault address information for redundancy analysis. FIR supports the fault syndrome and the address information for faulty memory cells. FID is a mode to support detailed diagnostic information for a fault analysis. FID’s diagnostic data consists of the fault syndrome, the fault cell address, and the pattern that detected the fault. Since the background data is selected by the ATE, background data information is not necessary. The DPM modifies the diagnostic data created in the FIR and FID modes into serial data by using a parallel-to-serial circuit and then serially outputs the data to the external ATE.

Figure 8 shows the diagnostic data formats of the FIR and FID. The FIR format consists of two fields: the fault syndrome and the fault address. The FID format consists of three fields: the fault detecting pattern, the fault syndrome, and the fault address. The fault syndrome is the fault data information of faulty cells. The fault address is the address of the faulty cell.

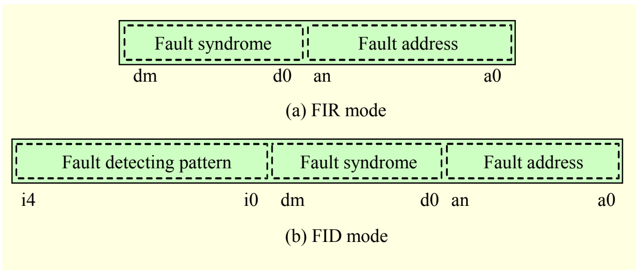


Fig. 8. Diagnostic data format of FIR and FID.

The fault detecting pattern reports the read operation in the test algorithm that detected the fault.

IV. Implementation and Verification

To evaluate the proposed FPMBIST, the FPMBIST architecture is implemented using the Samsung embedded single-port SRAM (16.3 K×16) and dual-port SRAM (8.2 K ×32) [19], which can be used for SoC designs. The target single-port memory is an SRAM with a 7-bit row address, a 7-bit column address, and a 16-bit data word. The target dual-port memory is an SRAM with a 9-bit row address, a 4-bit column address, and a 32-bit data word. In addition, various memory test algorithms for single-port memory testing (March Y, March C+, March SS, Walking, and Galloping) and dual-port memory testing (March s2PF, March A2PF) are microprogrammed via the proposed instruction set. The proposed FPMBIST scheme is designed with Verilog HDL and is synthesized using a Synopsys Design Compiler with TSMC 0.13-μm CMOS technology.

The instruction bit size of the proposed instruction set that is used to generate the test patterns is shown in Table 3. The instruction set does not require as many instructions as the number of March elements when it implements March-based algorithms. For example, the March C+ algorithm of 14*N* requires 10 instructions to implement algorithms with linear instruction sets. This is possible because of the RDBranch function. Accordingly, the March C+ algorithm requires 90 bits (10×9 bits) of instruction memory. The March s2PF and March SS algorithms [20] are implemented with nine instructions and fourteen instructions, respectively, from the linear instruction set. The Galloping algorithm was implemented with only seven instructions from the nonlinear instruction set. The nonlinear instruction set generates complex patterns by simply exploiting the two-level loop of the address generator and the multiple loops through the branch register.

The FPMBIST generates test patterns using the instruction codes stored in the internal instruction memory. The sizes of the codes determine the total BIST area overhead. Since most of the conventional memory test algorithms and the proposed dual-port memory test algorithms are encoded with fewer than 30 instructions, the instruction memory size is 70 bytes (14 bits×40). According to the synthesis results, the hardware area (or gate count) of the proposed FPMBIST with a 70-byte instruction memory is estimated to be 7,942 gates, and the maximum operating frequency is calculated to be 400 MHz.

A performance comparison of the proposed FPMBIST and previous microcode-based PMBIST schemes is also shown in Table 4. The architectures from [12]-[14] are programmable memory BIST schemes for single-port memory testing, whereas the schemes from [15] and [16] are intended for dual-port memory testing.

The area overhead of each internal block of the PBIST is shown in Fig. 9. The CMD block, address generator, and DPM occupy about 67.5% of the total hardware area. The address generator occupies a large area because it requires a complex counter architecture. The diagnostic module also occupies

Table 3. Instruction bit sizes for test algorithm.

Algorithm	Instruction set	Instruction set size	Instruction bit
Galloping	Nonlinear instruction set	14 bits	98 bits
Butterfly	Nonlinear instruction set	14 bits	140 bits
March Y (8*N*)	Linear instruction set	9 bits	54 bits
March C+ (14*N*)	Linear instruction set	9 bits	90 bits
March SS (22*N*)	Linear instruction set	9 bits	126 bits
March s2PF (14*N*)	Linear instruction set	9 bits	81 bits
March A2PF (18*N*)	Linear instruction set	9 bits	108 bits

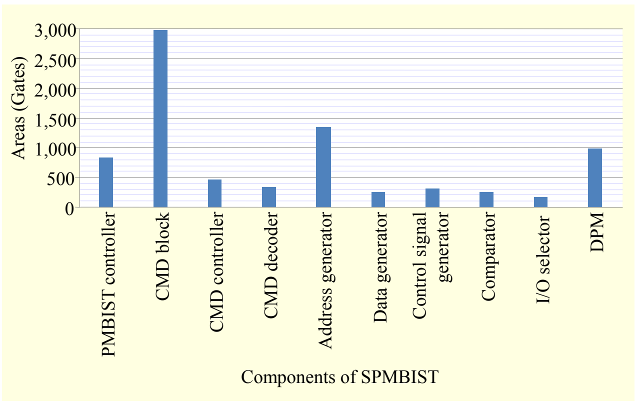


Fig. 9. Areas of different components of FPMBIST architecture.

Table 4. Performance comparison.

			[12]	[13]	[14]	[15]	[16]	FPMBIST
Target memory	Single-port memory		Y	Y	Y	Y	Y	Y
	Dual-port memory		N	N	N	Y	Y	Y
Target test algorithm	March-based algorithm		Y	Y	Y	Y	Y	Y
	Non-March algorithm	Galloping, sliding diagonal	Y	Y	Y	N	N	Y
		Butterfly	N	Y	N	N	N	Y
	Dual-port memory test algorithm		N	N	N	Y	Y	Y
Data retention fault test			N	N	N	N	N	Y
Address generation methods	Fast diagonal		Y	Y	Y	Y	Y	Y
	Fast row		N	N	N	N	N	Y
	Fast column		N	N	N	N	N	Y
Flexibility			Medium	High	High	Medium	Medium	Very high
Multi-loop			N	Y	Y	P	N	Y
Maximum frequency			NA	NA	300 MHz	NA	NA	400 MHz
Diagnostic scheme	Fault information		Y	Y	N	N	N	Y
	Efficiency		Medium	Medium	NA	NA	NA	High
Instruction set architecture	Linear instruction set size (bit)		9	8	9	4	NA	9
	Nonlinear instruction set size (bit)							14
Materialization of test algorithm	Instruction bits of March Y (bit)		90	120	72	68	NA	54
	Instruction bits of March C+ (bit)		144	216	126	108	NA	90
	Instruction bits of March SS (bit)		216	344	198	140	NA	117
	Instruction bits of March s2PF (bit)		NA	NA	NA	216	NA	81
	Instruction bits of March A2PF (bit)		NA	NA	NA	NA	NA	99
	Instruction bits of Walking (bit)		NA	NA	90	NA	NA	84
	Instruction bits of Galloping (bit)		NA	NA	108	NA	NA	98
Area overhead (gate)			8.9 K	13.6 K	6.4 K	9.9 K	NA	7.9 K

Y: support, N: not support, P: partially support, NA: not available

about 12.6% of the area overhead.

Table 4 shows a comparison of the proposed scheme and previous programmable BIST schemes in terms of the testable memory types, supportable test algorithm, instruction size, and the instruction memory requirements. In Table 4, March C+ is used for the single-port memory test, whereas March s2PF is employed for the dual-port memory test. The hardware areas of the proposed scheme and of the previous architectures are also displayed in Table 4.

The architectures in [12] and [13] can support March and non-March algorithms; however, the scheme in [12] cannot encode all non-March algorithms since multi-loop is not supported. The scheme in [13] can support multi-loop performance, but the hardware area is too large. The architecture in [14] supports both March and some non-March algorithms. The scheme in [14], which supports the Galloping algorithm, requires 108 bits of instruction memory. It also supports multi-loop performance and reduces the area overhead. The schemes in [12] and [13] include diagnostic logic with gate counts of 0.3 K and 1.2 K, respectively. However, the diagnostic logic supports only the failing memory address and the fail-map data as fault information, and thus an accurate fault analysis is not possible. Also, the PMBIST schemes in [12]-[14] can only test single-port memory, not dual-port memory.

The architectures in [15] and [16] can encode all March algorithms and portions of the dual-port memory test algorithms. The schemes of [15] and [16] cannot encode the A2PF algorithm and can only access the same cell through two different ports. Therefore, it is not possible to detect all faults in dual-port memory. In addition, non-March algorithms for single-port memory cannot be programmed; thus, the fault coverage is constrained.

The proposed FPMBIST can encode all March and non-March algorithms for both single-port memory and dual-port memory. In addition, since multi-loop performance is supported, complex algorithms can be efficiently programmed. To encode the March C+ (14*N*), March s2PF (16*N*), and Galloping algorithms, 90-bit, 81-bit, and 98-bit instruction memory is required, respectively, through the linear and nonlinear instruction sets. The nonlinear instruction set (14 bits) has five more bits than the instruction set (9 bits) proposed in [14]. However, it uses 10 fewer bits to implement the Galloping algorithm. Furthermore, it can effectively implement diverse non-March algorithms and it can support data retention tests and so on. As a result, the memory sizes are considerably smaller than those in previous PMBIST schemes. The implementation results reveal that the hardware area of the proposed scheme is the smallest. The proposed FPMBIST can test both single-port memory and dual-port memory and can support various test algorithms in a small hardware area. In addition, since memory of the same port type can be grouped into one BIST test domain with the proposed FPMBIST scheme, various embedded memory cores can easily be tested with relatively smaller area overhead and shorter test application times. Furthermore, the FPMBIST has an efficient diagnostic module that supports the bypass, FID, and FIR methods for repair and diagnosis. The diagnostic module improves memory yield by effectively supporting diverse diagnostic data to the external ATE.

V. Conclusion

A new microcode-based flexible programmable BIST scheme was proposed to encode various algorithms for memory of different port types. The linear and nonlinear instruction sets were also developed to support all March-based algorithms, dual-port memory test algorithms, and non-March-based algorithms for both single-port memory and dual-port memory. In addition, the FPMBIST supports the data retention test. Due to the efficient address generator architectures and the use of diverse address generation methods (fast diagonal, fast row, and fast column), the instruction sets can reduce the instruction memory requirement and reduce the BIST hardware area. The diagnostic module improves memory yield by effectively supporting diverse fault information for repair and diagnosis. According to the experiment results, the proposed FPMBIST scheme can ensure more flexible programmability for various test algorithms and a smaller instruction memory requirement and is able to support memory of diverse port types in a small hardware area.

References

[1] Y. Zorian, E.J. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," *IEEE Comput.*, vol. 32, issue 6, 1999, pp. 52-60.

[2] 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. Syst.*, vol. 9, issue 5, 2001, pp. 730-735.

[3] A. van de Goor et al., "Low-Cost, Customized and Flexible SRAM MBIST Engine," *Proc. Int. Symp. Design Diagnostics Electron. Circuits Syst.*, 2010, pp. 382-387.

[4] K. Zarrineh, S.J. Upadhyaya, and S. Chakravarty, "A New Framework for Generating Optimal March Tests for Memory Arrays," *Proc. Int. Test Conf.*, Oct. 1998, pp. 73-82.

[5] S. Hamdioui, Z. Al-Ars, and A.J. van de Goor, "Testing Static and Dynamic Faults in Random Access Memories," *Proc. VLSI Test Symp.*, Apr. 2002, pp. 395-400.

[6] A. van de Goor et al., "Generic, Orthogonal and Low-cost March Element Based Memory BIST," *Proc. IEEE Int. Test Conf.*, 2011, pp. 1-10.

[7] C. Cheng et al., "BRAINS: A BIST Compiler for Embedded Memories," *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Oct. 2000, pp. 299-307.

[8] K. Yamasaki et al., "External Memory BIST for System-in-Package," *Proc. Int. Test Conf.*, Nov. 2005, pp. 1145-1154.

[9] A.W. Hakmi et al., "Programmable Deterministic Built-In Self-Test," *Proc. IEEE Int. Test Conf.*, Oct. 2007, pp. 1-9.

[10] N.Q. Mohd Noor, A. Saparon, and Y. Yusof, "An Overview of Microcode-Based and FSM-Based Programmable Memory Built-in Self-Test (MBIST) Controller for Coupling Fault Detection," *Proc. IEEE Symp, Ind. Electron. Appl.*, Oct. 2009, pp. 469-472.

[11] H.C. Lu and J.F. Li, "A Programmable Online/Off-line Built-in Self-Test Scheme for RAMs with ECC," *Proc. Int. Symp. Circuits Syst.*, May 2009, pp. 1997-2000.

[12] X. Du et al., "Full-Speed Field-Programmable Memory BIST Architecture," *Proc. IEEE Int. Test Conf.*, Nov. 2005, pp. 1165-1173.

[13] X. Du et al., "A Field Programmable Memory BIST Architecture Supporting Algorithms with Multiple Nested Loops," *Proc. IEEE Asian Test Symp.*, Nov. 2006, pp. 287-292.

[14] Y. Park et al., "An Effective Programmable Memory BIST for Embedded Memory," *IEICE Trans. Inf. Syst.*, vol. E92-D, no. 12, Dec. 2009, pp. 2508-2511.

[15] A. Benso et al., "A Programmable BIST Architecture for Clusters of Multiple-port SRAMs," *Proc. IEEE Int. Test Conf.*, Oct. 2000, pp. 557-566.

[16] M. Karunaratne and B. Oomman, "Optimized BIST for Embedded Dual-Port RAMs," *Proc. IEEE Midwest Symp.*

Circuits Syst., Aug. 2010, pp. 125-128.

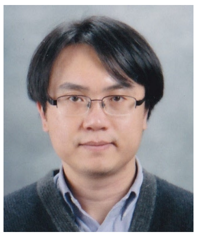
[17] S. Hamdioui and A.J. van de Goor, "Efficient Tests for Realistic Faults in Dual-Port SRAMs," *IEEE Trans. Comput.*, vol. 51, issue 5, 2002, pp. 460-473.

[18] Y. Park et al., "An Effective Test and Diagnosis Algorithm for Dual-Port Memories," *ETRI J.*, vol. 30, no. 4, Aug. 2008, pp. 555-564.

[19] *Samsung 0.13 μm Generic Process Compiled Memory* (*STD150E*), Data Book of Samsung Electronics, May 2005.

[20] S. Hamdioui, A.J. van de Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," *Proc. IEEE Int. Workshop Memory Technol.*, *Design*, *Testing*, July 2002, pp. 95-100.

Youngkyu Park received his B.S. degree in electronics engineering from Hoseo University, Rep. of Korea, in 2004 and his M.S. and Ph.D. degrees in electrical and electronics engineering from Yonsei University, Seoul, Rep. of Korea, in 2007 and 2013, respectively. From 2013, he was a senior engineer with Memory Division, Samsung Electronics Company, Rep. of Korea. His current research interests include VLSI design and testing, memory BIST, and design for testability.



Hong-Sik Kim received his BS, MS, and PhD degrees in electrical and electronics engineering from Yonsei University, Seoul, Rep. of Korea, in 1997, 1999, and 2004, respectively. In 2005, he was a postdoctoral fellow with the Virginia Institute of Technology, Blacksburg, VA, USA. In 2006, he was a senior engineer with System LSI Group, Samsung Electronics Company, Rep. of Korea. From 2007 to 2010, he was a research professor with Yonsei University. Since 2010, he has been a senior engineer with SK Hynix Semiconductor. His current research interests include design for testability, lossless data compression, 3-D graphics rendering hardware design, and new memory/storage subsystems.

Inhyuk Choi received his BS in electrical and electronics engineering from Yonsei University, Seoul, Rep. of Korea, in 2009. Currently, he is working toward a combined MS/PhD in electrical and electronics engineering at Yonsei University. His research interests include SoC design, design for testability, and system-level test and validation.

Sungho Kang received his BS in control and instrumentation engineering from Seoul National University, Seoul, Rep. of Korea, and his MS and PhD in electrical and computer engineering from the University of Texas at Austin, Austin, TX, USA, in 1992. He was a research scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc., and a senior staff engineer with Semiconductor Systems Design Technology, Motorola Inc. Since 1994, he has been a professor in the Department of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea. His main research interests include VLSI/SOC design and testing, design for testability, and design for manufacturability.