On Programmable Memory Built-In Self Test Architectures

Kamran Zarrineh and Shambhu J. Upadhyaya
Department of Electrical and Computer Engineering
SUNY at Buffalo, Buffalo, NY 14260

Abstract

The design and architectures of a microcode-based *memory BIST and* programmable FSM-based *memory BIST unit are presented. The proposed microcode-based memory BIST unit is more efficient and flexible than existing architectures. Test logic overhead of the proposed programmable versus non-programmable memory BIST architectures is evaluated. The proposed programmable memory BIST architectures could be used to test memories in different stages of their fabrication and therefore result in lower overall memory test logic overhead. We show that the proposed microcode-based memory BIST architecture has better extendibility and flexibility while having less test logic overhead than the programmable FSM-based memory BIST architecture.*

1. Introduction

Digital systems are composed of data paths, control paths and memories. The low cost of memory and high memory demand of high-speed DSP circuits and generic microprocessors have made the memory subsystem an important focus of the design.

Defects in 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 [6, 10]. Memories are more likely to fail than random logic and therefore three classes of memory tests have been proposed to detect the memory faults [10, 1, 2]. Application of test sequences to embedded memories using off-chip testers results in a high test time and test cost due to the large size of embedded memories. To overcome this problem, the computed test sequences are generated on-chip using a memory Built-In Self Test (BIST) unit.

A memory BIST unit consists of a controller to control the flow of test sequences and other components to generate the necessary test control and data. A memory BIST controller could be designed as a Finite State Machine (FSM)-based or microcode-based controller. The FSM-based controllers are the hardware realization of a selected memory test algorithm. This type of memory BIST architecture has optimum logic overhead, however, lacks the flexibility to accommodate any changes in the selected memory test algorithm. This results in re-design and re-implementation of the FSM-based controller for any minor changes in the selected memory test algorithm. In microcode-based controllers, a selected memory test algorithm is written in terms of a set of supported instructions and loaded in the memory BIST controller. This type of memory BIST architecture allows changes in the selected test algorithm with no impact on the hardware of the controller. This flexibility might result in higher logic overhead for the memory BIST controller [8]. However, the additional logic overhead, if kept in reasonable levels, could be justified by using the memory BIST to reduce the cost of diagnostics.

Memories undergo different type of testing during the course of their design and fabrication. Tests necessary for embedded memories and diagnostics have a set of requirements on the memory BIST controller. To satisfy these requirements additional hardware might have to be added to the memory BIST unit. Comparison of logic overhead of non-programmable versus programmable memory BIST architectures considering only the logic overhead of only one type of memory test or diagnostics might not truly reveal the overhead of one architecture over another. Therefore, to compare and evaluate different memory BIST architectures, the overall testing and diagnostics requirements of memories and the flexibilities of each memory BIST architecture must be considered.

Programmable memory BIST controllers that could accommodate testing requirements of embedded memories were proposed in [4, 3, 9]. The drawback of the method described in [4] is that only memory tests where each component does not have more than two instructions could be coded efficiently. Furthermore, it is not clear if their architecture has the flexibility to accommodate the test requirements of the memories in different stages of their fabrication. The programmable ar-

chitecture described in [3] considers dividing a test algorithm into smaller sub-tests and loading the necessary microcodes through multiple loads. This is time consuming and might not always be feasible. Also, this architecture does not support optimum representation of the symmetric test algorithms and test algorithms that consist of several smaller loops such as march test algorithms [5]. The architecture described in [9] is mainly designed for diagnostics and process monitoring. In this approach, the test algorithm is loaded in a 32×14 SRAM using an initialization sequence. This SRAM has 3 to 4 times larger size than the previous programmable methods. Also, testing the SRAM itself adds to the complexity of the overall testing of the design.

In Section 2 of this paper, we present the details of two programmable memory BIST architectures that are more efficient and do not suffer from the drawbacks of the existing methods. The test logic overhead and extendibility of the proposed programmable methods versus a set of non-programmable architectures are evaluated in Section 3 and Section 4 concludes the paper.

2. Proposed Memory BIST Architectures

The memory BIST controller asserts and de-asserts a set of controlling signals for the memory array and other components of the memory BIST unit based on a selected memory test algorithm. Two programmable memory BIST controller architectures *microcode-based* and *programmable FSM-based* have been developed and are described in this section.

2.1. Microcode-Based BIST Controller

In microcode-based memory BIST architecture the controller consist of 1) storage unit, 2) instruction counter, 3) instruction selector, 4) branch register, 5) instruction decoder and 6) reference register as shown in Fig. 1.

The storage unit is a two dimensional $Z \times Y$ buffer and stores the necessary microcodes of a memory test algorithm. A 2-bit initialization signal would initialize the storage unit to the default microcodes or a set of custom microcodes. The supported microcode is 10-bits wide and consist of a 2-bit field for address generation, 2-bit for data generation, 1-bit for compare, 2-bits for read/write and a 3-bit field to control the flow of the microcode. A description of different fields and an example of memory test algorithm (March C) are shown in Fig. 2.

The instruction counter is a $log_2(Z)+1$ bit binary counter that specifies which instruction should be executed next. The *Reset* signal would initialize the instruction counter to 0 in the beginning of the memory test.

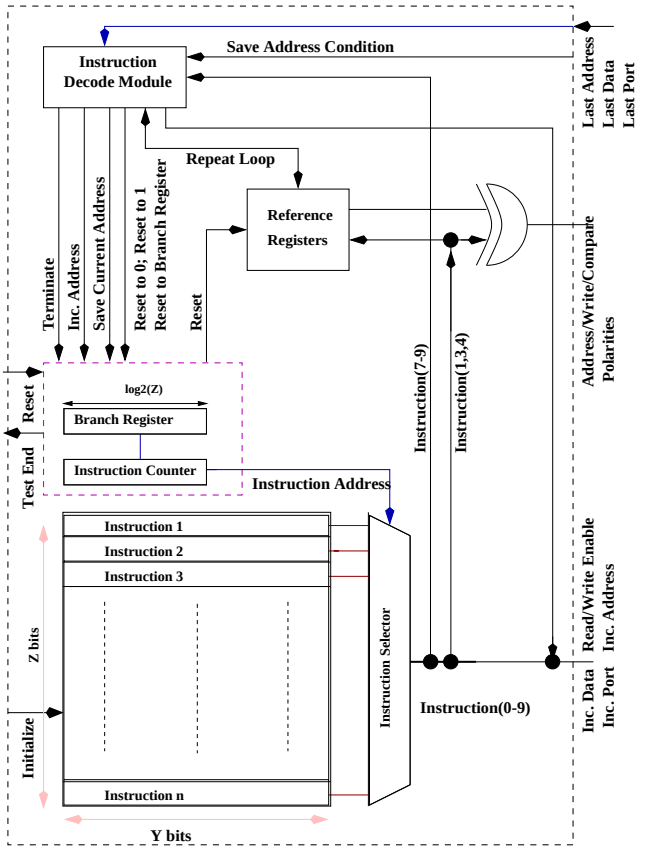

Figure 1. Microcode-Based BIST Controller

The last bit of the instruction counter specifies the end of the test. This bit could be specified by exhausting the allowed instruction addresses or by asserting the *Terminate* signal. Instruction selector is a Z×Y:Y selector with $log_2(Z)$ control lines. The output of the instruction counter is the control signals of the instruction selector. The branch register is a $log_2(Z)$ bit register and holds the address of the branch-to instruction. If *Save Current Address* signal is asserted, the value of the instruction counter is copied to the branch register and similarly if *Reset to Branch Register* signal is asserted, the value of branch register is copied back to the instruction register in the case of a branch.

The instruction decoder interprets the 3-bit conditional field (last field) of the microcode instruction and controls the flow of execution of the microcodes in the storage unit. The *Inc. Address* signal would hold/increment the instruction counter while *Reset 0*, *Reset 1* and *Reset Branch Register* reset the instruction counter to the first, second or the instruction specified by the branch register. A combination of *Last Port*, *Last Data*, *Last Address* and *Repeat Loop* is used as the necessary conditions for the 3-bit conditional instruction. The

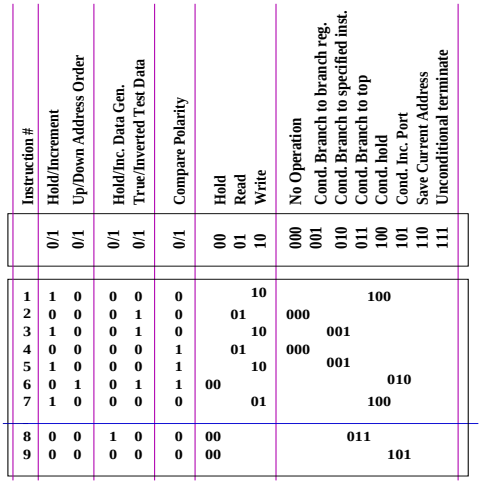

Figure 2. Microcode-Based Instruction Definition

signal *Save Address Condition* allows saving the contents of the instruction counter in the branch register automatically if a condition occurs. For example, if the last address is specified as the save address condition, then every time the *Last Address* signal is asserted, the contents of the instruction counter is copied in the branch register. This enables an automatic looping capability for a set of operations on a given cell. The *Inc. Port* signal is asserted by the instruction decoder module if the next port has to be activated in the case of multiport embedded memories.

Furthermore, to allow optimum coding of symmetric test algorithms, ie. March C or March A [10], a 4-bit reference register is designed to hold the repeat loop bit, auxiliary address order, auxiliary data and auxiliary compare. The repeat loop bit is asserted by execution of *Repeat* instruction and specifies that the microcode in the storage unit is being repeated. If the repeat loop bit is de-asserted and *Repeat* instruction is being executed the address order, data and compare polarity fields of that instruction are stored in their corresponding reference register bits. This allows re-execution of a set of microcodes with different address order, test data and compare polarities. The auxiliary address order, data and compare values stored in reference register are XORed with the address order, test data and compare specified in each instruction. The *Reset* signal initializes the contents of the reference register to 0. The combination of *Repeat* and reference registers enables optimal coding of symmetric memory test algorithms.

To further illustrate the instruction set and capability of the proposed architecture consider the March C [10] test algorithm shown mathematically in Eq. 1 and illustrated in microcodes in Fig. 2.

$$MarchC = \{\Updownarrow(w_0) \Uparrow(r_0w_1) \Uparrow(r_1w_0) \Downarrow(r_0w_1) \Downarrow(r_1w_0) \Updownarrow(r_0)\} \quad (1)$$

In Eq. 1, the address order=$\{\Uparrow, \Downarrow, \Updownarrow\}$: refers to traversing the address space of a memory from 0 to *n-1* (up address order), from *n-1* to 0 (down address order) or either (don't care). The operations w_d and r_d refer to write or read logic value d=$\{0,1\}$. The first instruction in Fig. 2 writes 0 to each memory cell in up address order. The second and third instructions indicate that read memory cell i for expected 0 and write 1 while incrementing in each address in up address order respectively. Similarly, the fourth and fifth instructions indicate the same sequence of operations as the third and fourth instructions while expecting 1 and writing 0. The address order, test data and compare polarity fields in the sixth instruction, set the auxiliary address order, test data and compare values in the reference register and therefore in this case the second through fifth instructions are repeated with complemented address order, test data and compare polarities. Since the repeat bit is set after the first execution of the fifth instruction it will be considered as a no operation in the second execution, however, the repeat bit is reset to 0. The seventh instruction reads each memory cell expecting 0 in an up address order. The eight and ninth instructions are needed if supporting word-oriented and multiport memories. If the last data pattern has not been reached the eight instruction would increment the data generation unit and reset the instruction counter else it would reset the data generation unit and execute the next instruction. Similarly, if the last port has not been reached the ninth instruction increments the port and reset the instruction counter to 0 else it terminates the test.

2.2. FSM-Based Memory BIST Controller

Given a set of march test components as shown in Eq. 2 then most march test algorithms could be produced by using a combination of these test components with different address order, test data and compare values. For example, by selecting march test component SM_0 with up address order and data value of 0, followed by SM_1 with up address order, data and compare vlaues of 1 and 0 the first two components of March C are created. Similarly, the remaining test components of March C could be generated. Our proposed programmable FSM-based memory BIST unit consist of a lower parametrized FSM-based controller and an upper 2-dimensional circular buffer as shown in Fig. 3.

$$SM_0 = \langle \Updownarrow (w_d) \rangle; SM_1 = \langle \Updownarrow (r_{\overline{d}} w_d) \rangle; SM_2 = \langle \Updownarrow (r_{\overline{d}} w_d r_d w_{\overline{d}}) \rangle$$

$$SM_3 = \langle \Updownarrow (r_{\overline{d}} w_d w_{\overline{d}}) \rangle; SM_4 = \langle \Updownarrow (r_{\overline{d}} r_{\overline{d}} r_{\overline{d}}) \rangle; SM_5 = \langle \Updownarrow (r_{\overline{d}}) \rangle;$$

$$SM_6 = \langle \Updownarrow (r_{\overline{d}} w_d w_{\overline{d}} w_d) \rangle; SM_7 = \langle \Updownarrow (r_{\overline{d}} w_d r_d) \rangle \quad (2)$$

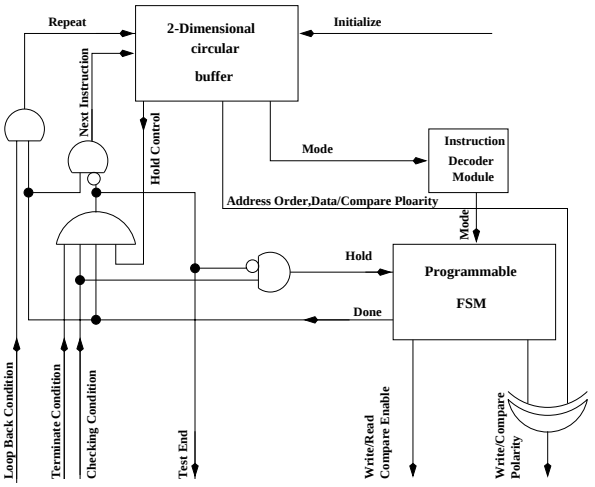

Figure 3. Programmable FSM-Based Memory BIST

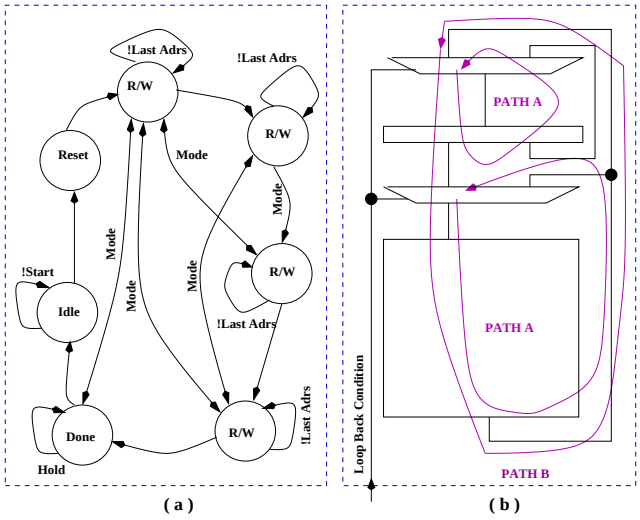

Figure 4. (a) Lower and (b) Upper Level Controllers

The lower level controller is a parameter driven 7-state FSM as shown in Fig. 4(a). and is designed to realize the march test components described in Eq. 2. In state *Idle* and *Done* the lower level controller is de-active. In state *Reset* appropriate memory BIST components, ie. Address Generation, etc., are reset. Depending on the value of *Mode* signal, the next 4 R/W states perform read or write operations on the memory under test. The signal *Last Address* is the terminating signal and the entering condition to state *Done*. If the input signal *Hold* is asserted the FSM would stay in *Done* otherwise would move to idle state.

Instruction #	Cond. Test End	Up/Down Adrs Order	Hold/Inc. Data Gen. True/Invtd. Test Data	Hold/Inc. Port	SM0 SM1 SM2 SM3 SM4 SM5 SM6 SM7
	0/1	0/1	0/1 0/1	0/1	000 001 010 011 100 101 110 111
1	0	0	0 0	0	000
2	0	0	0 1	0	001
3	0	0	0 0	0	001
4	0	1	0 1	0	001
5	0	1	0 0	0	001
6	1	1	0 1	0	101
7	1	1	1 0	0	xxx
8	1	1	0 1	1	xxx

Figure 5. FSM-Based Instruction Definition

The upper level controller is a 2-dimensional circular buffer, as shown in Fig. 4(b), that holds the parameters necessary for the low level controller to realize a march test. The input signal *Initialize* sets the upper level controller to a default test algorithm or a set of custom instructions could be loaded to the upper level controller if necessary. These instructions contain the necessary information to set the parameters of the low level controller and are divided into 5 fields: the first 1-bit field is one of the conditions necessary to hold the low level controller in its *Done* state. The next 1-bit field contains the reference address order, the third 2-bit field contains the control signals for the data generation unit. The fourth 1-bit field contains the polarity of the expected data. These signals are considered as the reference or base values and the FSM might generate an address order, write and compare values that would be XORed with these signals. The last 3-bit field specifies the mode of the operation of the low level controller. An example memory test algorithm, March C, is shown in Fig. 5. The signal *Next Instruction* allows the upper level controller to executes the next instruction. To support memories with different characteristics, ie. word-oriented and multiport memories, the capability to repeat execu-

tion of different instructions based on a given condition is necessary. Therefore, for a word-oriented and multiport memory, for each port and for each data background pattern the entire test algorithm is repeated. This translates into having a *Loop Back Condition* signal that allows repeating the test algorithm for each port by taking path A in Fig. 4(b). At the end of the data background, *Loop Back Condition* signal changes the flow of execution of the instruction to path B which allows the port to be incremented. The input signal *Checking Condition*, if asserted, indicates the valid time for checking the *Termination Condition* signal.

For example, consider the instruction set for March C as shown in Fig. 5. The first instruction would write 0 in each memory cell in up address order. The second and fourth instructions would read a cell and expect 0 followed by write 1 in each memory cell in up and down address orders respectively. Similarly, third and fifth instructions read a memory cell and expect 1 and write 0 to each memory cell in up and down address orders respectively. The sixth instruction reads each memory cell in up address order. The seventh instruction increment the data generation unit and loops back to the first instruction until the last data pattern has been reached. The eight instruction activates the next port in the multiport memories. These two instruction are to support word-oriented and multiport memories. The test is terminated once the last port has been processed.

3. Experimental Results

The proposed programmable memory BIST controller architectures have been designed and their logic overhead have been evaluated against a set of non-programmable memory BIST architectures.

In the first set of experiments, we assume that the memory under test is bit-oriented and single port. The test algorithms are assumed to be symmetric with the number of operations comparable to March C and March A [10]. A set of non-programmable FSM-based controllers which are logic realization of March C and March A and their deviations were designed and used to evaluate the logic overhead of the proposed programmable methods. Deviations of March C and A have the added capability to detect data retention faults. These algorithms are referred to as March C^+, March C^{++}, March A^+ and March A^{++}. In March C^+, two march test components $Hold \Uparrow(r_{\overline{d}}\ w_d\ r_d)$ $Hold \Uparrow (r_{\overline{d}})$ are added to the end of the March C algorithm to allow detection of data retention faults. In March C^{++} each read operation is replaced by three read operations to excite and detect disconnected pull-up/down devices in the memory cells. Similarly, March A^+ and March A^{++} are enhanced version of March A test algorithm. The results are summarized in Table 1. The first column specify the memory BIST architecture, the second column is the the degree of flexibility that the architecture provides for realizing different types of test algorithms. The third column is the internal area size(2×2-input NAND gates). The fourth column is the size of the controller using IBM CMOS5S (0.35 micron technology).

Table 1. Size of the Memory BIST Methodology For Bit-Oriented and Single port memories

Method	*Flex.*	*Int. Area*	*Size* μm^2
Microcode-Based	HIGH	923	0.191
Prog. FSM-Based	MEDIUM	890	0.185
March C	LOW	264	0.0547
March C^+	LOW	277	0.0574
March C^{++}	LOW	388	0.0805
March A	LOW	246	0.0510
March A^+	LOW	282	0.0585
March A^{++}	LOW	370	0.0767

In addition, to study the extendibility and changes in the area overhead of the memory BIST architectures, we have modified the memory BIST units to support word-oriented and multiport memories. The results are summarized in Table 2.

Table 2. Size of the Memory BIST Methodology For Word-Oriented and Multiport Memories

Method	*Word-Oriented*		*Multiport*	
	Int.A.	Size μm^2	Int.A.	Size μm^2
Microcode-Based	1007	0.209	1143	0.237
Prog. FSM-Based	979	0.203	1104	0.229
March C	284	0.059	299	0.060
March C^+	295	0.061	309	0.065
March C^{++}	413	0.086	433	0.090
March A	258	0.054	276	0.057
March A^+	296	0.061	314	0.065
March A^{++}	391	0.081	403	0.084

Preliminary experimental results show that any reduction in the area of the storage units of the proposed programmable memory BIST architectures has the largest effect on the area of programmable memory BIST units. For microcode-based architecture the storage unit holds the instructions of the test algorithm with no dependence on the functional clock and therefore the storage cells of the storage unit could be designed much slower, smaller and less expensive. Particularly, IBM ASICs provides

a set of smaller and slower scan-only storage cells which could be readily used. These cells are approximately 4 to 5 times smaller than regular full scan registers and operate in about 1/8 or 1/16 of functional clock rate. In programmable FSM-based architecture the cells in storage unit shift for each march test component and therefore have to satisfy the functional clock rate.

The storage unit of the microcode-based architecture is designed as scan-only registers. This technique reduces the testing problem of the storage unit as well. Testing a scannable register file, storage unit of the FSM-based architecture, is simpler than testing a small SRAM or ROM. However, testing the data paths of the scannable registers using random logic BIST might require additional test points or in the case of stored pattern test methodology requires more test pattens and more complex ATPG tool. The scan-path of the scan-only registers is easily tested via the scan-in ports and could be used as a set of stimulus test points to test the entire memory BIST unit. The area overhead of the enhanced storage unit is summarized in Table 3. From the presented experimental results the following observations could be made.

Table 3. Adjusted Size of Microcode-Based Controller

Method	*Adj. Int. Area*	*Adj. Size* μm^2
Bit-Oriented	405	0.0840
Word-Oriented	423	0.0879
Multiport	475	0.0985

- Re-design of the microcode-based memory BIST controller result in approximately 60% reduction in the size of the controller (see entries in Table 1-3).
- The microcode-based memory BIST controller provides better flexibility with lower logic overhead than programmable FSM-based memory BIST controller.
- By increasing the fault model and enhancing the test algorithm the area overhead of the non-programmable memory BIST unit increases.
- The area difference between the microcode-based and programmable memory BIST controllers decreases as the capability of the non-programmable memory BIST unit is enhanced.

4. Conclusion

In this paper, the design of two programmable memory BIST architectures was presented. The proposed memory BIST architectures could accommodate changes in the test algorithm with no impact on the hardware. Different types of memory test algorithms could be realized using the proposed memory BIST architectures and therefore memories with different characteristics and test requirements could use the same memory BIST architecture. In addition, the proposed microcode-based architecture does not suffer from shortcomings of other published programmable memory BIST architectures.

The re-design of the storage unit of the microcode-based architecture resulted in an optimized microcode-based memory BIST architecture with comparable logic overhead with non-programmable methods. The reduction in the area overhead combined with the flexibilities of the microcode-based expand its application from diagnostics [9] to on-line testing [7] and makes the additional logic overhead readily justified.

Acknowledgment

This reasearch has been supported in part by TDA organization in IBM. Authors would like to thank Mr W. Huott and Mr P. Shephard III for helpful discussions.

References

[1] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Test for VLSI: Pseudorandom Techniques*. Wiley Interscience, 1987.

[2] B. F. Cockburn and Y.-F. N. Sat. Synthesized Transparent BIST for Detecting Scrambled Pattern Sensitive Faults in RAM. In *Proc. Int. Test Conf.*, pages 23–32, 1995.

[3] P. G. S. III, W. V. Huott, P. R. Turgeon, R. W. B. Jr., G. Yasar, F. J. Cox, P. Patel, and J. B. H. III. Programmable built-in self test method and controller for arrays. *U. S. Patent No. 5,633,877*, May 1997.

[4] H. Koike, T. Takeshima, and M. Takada. A BIST Scheme Using Microprogram ROM for Large Capacity Memories. *Proc. Int. Test Conf.*, pages 815–822, 1990.

[5] M. Marinescu. Simple and efficient algorithms for functional RAM testing. In *Proc. Int. Test Conf.*, pages 572–576, Oct. 1982.

[6] R. Nair, S. Thatte, and J. Abraham. Efficient algorithms for Random Access Memories. In *IEEE Trans. on Computers*, pages 572–576, June 1978.

[7] M. Nicolaidis. Transparent BIST for RAMs. In *Proc. Int. Test Conf.*, pages 598–607, 1992.

[8] K. K. Saluja, S. H. Sng, and K. Kinoshita. Built-In Self-Testing RAM: A Practical Alternative. *IEEE Design and Test*, pages 42–51, February 1987.

[9] I. Schanstra, D. Lukita, A. van de Goor, K. Veelenturf, and P. J. van Wijnen. Semiconductor Manufacturing Process Monitoring Using BIST for Embedded Memories. In *Proc. Int. Test Conf.*, pages 872–881, 1998.

[10] A. van de Goor. *Testing Semiconductor Memories: Theory and Practice*. John Wiley and Sons, U.S.A, 1991.