A New Approach to Programmable Memory Built-In Self Test Scheme

Kamran Zarrineh and **Shambhu J. Upadhyaya**
Department of Electrical and Computer Engineering
SUNY at Buffalo, Buffalo, NY 14240
contact phone: (607) 755-6892
email contact: zarrineh@eng.buffalo.edu

May 6, 1998

Abstract

The design and architecture of a reconfigurable memory BIST unit is presented. The proposed memory BIST unit could accommodate changes in the test algorithm with no impact to the hardware. Different types of march test algorithms could be realized using the proposed memory BIST unit and the proposed architecture allows addition and elimination of the memory BIST components. Therefore memories with different characteristics and test requirements could use the same memory BIST architecture. By integrating the diagnostics with this memory BIST unit, further reduction in the cost of test was achieved. Our experimental results show that the proposed memory BIST methodolgy leads to lower logic overhead than other programmable methods.

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 general microprocessors have made the memory subsystem an important focus of the design. For example, in a typical microprocessor, a large block of memory constitutes the data or the instruction cache which consists of a 512×10 memory block followed by some combinatorial logic (an XOR tree) and a $32\text{K}\times8$ block of memory. Microprocessors have a 256×100 block of memory for the TLB, two segments of 16×40 and 8×47 memories are needed for SLB and BAT respectively [1]. These embedded memories contain about 50% of the total transistors in the microprocessor. Furthermore, the flow of data usually includes the embedded memories and therefore a failure in embedded memories result in a failure in the entire design. A combination of off-line and on-line test procedures have been proposed to improve the reliability of designs with embedded memories [2]. In this paper, we only concentrate on off-line memory test issues.

Defects in memory arrays are generally due to shorts and opens in memory cells, address decoder and read/write logic. A memory fault model is derived from the determined defects of an embedded memory and a test sequence is computed to detect these memory faults. Memory arrays undergo different test procedures during their fabrication process to detect the memory defects early in the process. Therefore, different memory faults are modeled for each fabrication stage and the necessary test sequences are computed to detect the target memory faults. Application of test sequences to embedded memories using off-chip testers results in a high test time and test cost due to 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) scheme. This memory BIST unit is designed to generate the computed test sequences. Memory array defects depend on the design, layout, type, target technology and fabrication process of the embedded memories. These design parameters could change at any time during the design cycle which results in a change of memory defects and fault model. A change in the fault model could result in recomputation of test sequences and ultimately re-design and re-implementation of the memory BIST unit. This is labor intensive and increases the time-to-market for designs. To overcome this problem the memory BIST unit should accommodate changes in the memory test sequence with no impact to its hardware.

Memories with different sizes and different characteristics could co-exist in a design. For example, a design could have a set of multiport, word-oriented embedded memories with relatively shallow address space as well as a set of large, single port, bit-oriented embedded memories. Designing different memory BIST units to satisfy the characteristics and test requirements of the embedded memories is cumbersome and time consuming. A configurable memory BIST architecture where different components could be added or eliminated from the memory BIST unit to satisfy the test requirements of an embedded memory would facilitate the testing of the embedded memories.

In addition, the hardware overhead of using a memory BIST unit to test only one embedded memory with a shallow address space cannot be justified and therefore a memory BIST unit is shared among several embedded memories. Unfortunately, the high bandwidth of embedded memories with the propensity to be scattered in a design creates a high wiring overhead of this approach. To overcome this problem, a *scan-based* memory BIST methodology is used. In a scan-based memory BIST methodology, the memory test data is loaded into the scannable storage elements around the embedded memory via the scan path using the scan operation. During the memory test procedure these scannable storage elements isolate the embedded memory and are only active through their scan path [3]. An example of scan-based memory BIST methodology is shown in Figure 1 (a). For embedded memories with a deep address space, the scan-based approach results in an impractical test time and therefore a *dedicated* memory BIST approach is used. In this approach, the test data is loaded into the latches surrounding the embedded memory through their functional path in parallel as illustrated in Figure 1 (b). If necessary, to further reduce the test logic overhead a set of components of the memory BIST unit, ie. the data generation, address generation and response analyzer components, are integrated with the scannable storage elements surrounding the embedded memory. This memory BIST methodology is referred to as an *integrated* memory BIST approach.

The high number of transistors in the embedded memories results in an increasesed yield loss and embedded memories are attractive for collecting process defect data due to their regular structure which facilitate the task of physical failure analysis. Therefore, diagnostics becomes an important part of the memory test methodology and procedure. A memory BIST unit could be used for diagnostics as well as testing the embedded memories. Diagnostics requires the memory BIST unit to apply parts of the

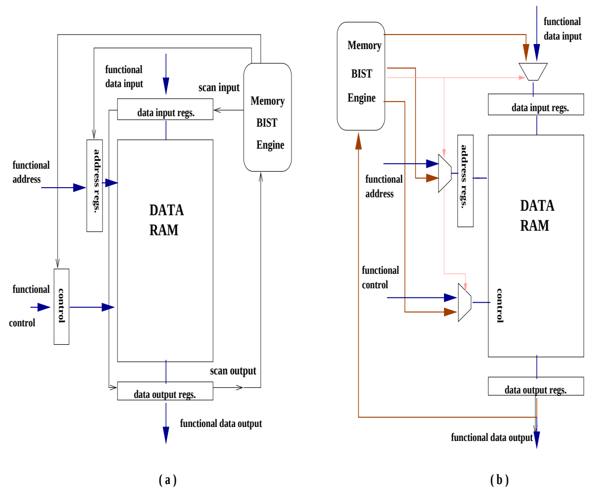

Figure 1: Testing an embedded memory using (a) scan-based and (b) dedicated memory BIST methodologies

computed test sequence independently and to make the test data and response of the embedded memory observable if necessary.

Programmable memory BIST controllers where the test sequence is described in terms of a set of supported microcodes was first proposed in [4, 5] with limited programmability and flexibility. In the programmable memory BIST unit proposed in [4], the memory BIST controller is efficient for test algorithms that have at most two operations in each test component, ie. March C [2]. The microcode based controller unit described in [5] is efficient only for walking 1/0 type algorithms where the test $data^{n+1}$ is one bit shifted version of test $data^{n}$. These architectures could result in a large controller for some memory test algorithms and do not have support for diagnostics.

In this paper we propose a reconfigurable building block memory BIST unit that could be reconfigured to accommodate the testing requirements of different types of embedded memories. The memory test sequences can be changed dynamically with no impact to the hardware and is capable of applying the test sequences necessary for testing the embedded memories in different stages of their fabrication. Furthermore, if available, the functional buffers surrounding the embedded memories are reconfigured and used as the test logic during testing and therefore further reduction in the logic overhead of the memory BIST unit is achieved.

This paper is organized as follows: Memory fault model and test algorithm are described in Section 2. The programmable memory BIST units is presented in Section 3. Section 4 illustrates the experimental results and experimental results and Section 5 concludes the paper.

2 Memory Fault Model and Test Algorithm

In this section the memory fault model and memory test algorithm used to satisfy different test procedures in embedded memories are discussed.

2.1 Functional Memory Fault Model

Opens and shorts between lines and transistors are two common defects in memory arrays. These defects can occur in memory cells, address decoder and read/write logic and can be modeled as single or multiple cell faults [2]. Address decoder faults can be modeled as memory cell array faults and are divided into 1) AF_1: faults preventing an address from selecting any cell or a cell from being accessed, 2) AF_2: fault that manifest itself as an address that accesses multiple cells and 3) AF_3: faults that cause multiple addresses to access the same cell [2, 6]. Depending on the design of a sense amplifier, its defects can modeled as memory array cell faults [7].

2.1.1 Single Cell Faults

Single cell faults are local to one memory cell. There are different kinds of single cell faults.

1. Stuck-at Faults (SAF): The state of a cell is stuck at a specific logic value regardless of the operation performed on that cell. We use $< \forall/0 >$ and $< \forall/1 >$ to denote that for any operation the logic value of the memory cell is 0 and 1 respectively.

2. Stuck Open Fault (SOF): A cell cannot be accessed due to an open in the word line. The notation $<\uparrow /0 >^{\star}$ or $<\uparrow /1 >^{\star}$ is used to represent this fault.

3. Transition Fault (TF): A cell fails to undergo a 0$\rightarrow$1 or a 1$\rightarrow$0 transition. We use $<\uparrow /0 >$ to denote a 0$\rightarrow$1 transition fault where 0 is the final value of the faulty cell. The notation $<\downarrow /1 >$ is for 1$\rightarrow$0 fault.

4. Data Retention Fault (DRF): A cell with such fault fails to retain its logic value after a time T. We use $< 0_T/ \uparrow>$ and $< 1_T/ \downarrow>$ to denote that the cell would fail to retain 0 and 1 value after time T.

2.1.2 Multiple Cell Faults

In multiple cell faults, more than one memory cell is affected by the memory fault. These types of faults can be modeled as coupling and neighborhood pattern sensitive faults. In this paper we concentrate only on coupling faults.

Coupling faults involve K cells (a coupling, a coupled and possibly one or more enable cell(s)). An operation on the coupling cell causes the logic value stored in the coupled cell to change to a specified logic value while enabling cell(s) have a specific pattern. Based on the type of operation on the coupling cell and the behavior of the coupled cell a number of coupling faults are defined as follows.

1. Inversion Coupling Fault (CF_{in}): an up ($\uparrow$) or down ($\downarrow$) transition of the coupling cell causes the value of the coupled cell to invert. Let $<\uparrow;\updownarrow>$ ($<\downarrow;\updownarrow>$) denotes that an up transition (down transition) of the coupling cell forces the value of the coupled cell to become the complement of its previous value.

2. Idempotent Coupling Fault (CF_{id}): this coupling fault consists of $<\uparrow;1>$, $<\uparrow;0>$, $<\downarrow;1>$ and $<\downarrow;0>$ memory faults. We use $<\uparrow;1>$ to denote an up transition in coupling cell forces the value of the coupled cell to 1. The others have similar behavior.

3. State Coupling Fault (CF_{st}): the 0 or 1 state of a coupling cell forces the coupled cell to 0 or 1 state. $< y;x >$ where y and $x \in \{0,1\}$ [8].

4. A Disturb Coupling Fault (CF_{dst}): an operation op on the coupling cell forces the value of the coupled cell to x. $< op; x >$ where: $op \in \{r_y, w_y\}$ and $x \in$ {0,1} [8].

Multiple faults are composed of single or multiple cell memory faults and are classified as unlinked or linked faults. In linked memory faults, coupling, coupled or both cells are shared among faults. A fault can be linked with its own or with different type of memory faults [2].

2.2 Memory Test Algorithms

Several types of tests are performed on embedded memories during their fabrication. The most common are burn-in, manufacturing and power-on reset memory tests. In a burn-in test methodology a checkerboard ($\overline{checkerboard}$) pattern is written into the embedded memory while the memory is stressed. The source of the stress is usually an increase in the temperature and a decrease in the operating voltage of the embedded memory which exposes data retention and marginal (weak) faults. In manufacturing testing, a series of patterns are applied to the embedded memory at-speed to detect stuck-at, transition and unlinked and/or linked coupling faults. Power-on reset test is applied anytime the memory is powered on and is used to detect stuck-at or transition faults. This test is also used to initialize the embedded memory to a known logic value.

Memory test algorithms could be classified as: 1) *deterministic*, 2) *pseudo random* [2] and 3) *pseudo exhaustive*. In a deterministic memory test algorithm, a set of patterns to detect the target faults is computed and applied to the embedded memory. A series of random memory test patterns are generated and used in pseudo random memory test. The number of patterns and the probability that a target fault goes undetected are inversely proportional in this scheme and therefore a large number of patterns might be necessary. In pseudo exhaustive memory test methodology, the memory is written with a set of background patterns in order to detect more hard-to-detect faults. In this paper we only consider deterministic test algorithms since the other two methods might result in a high test time and test logic overhead. The most popular and widely accepted deterministic test algorithm is the *march test*. A *march test* is a finite sequence of *march elements* applied to each memory cell in the memory array in either ascending or descending order before proceeding to the next memory cell [12]. March tests are popular because of their low temporal complexity, regular and symmetric structures and their ability to detect different types of memory faults. A march test contains at least one march test component. Each march test component consist of an address order=$\{\Uparrow, \Downarrow, \Updownarrow\}$ refers to the address order and consist of up/down, up and down address orders respectively; *operation*=$\{r_z^i,\ w_z^i \mid 0 \leq \text{i} \leq \text{n},\ \text{z} \in \{0,1\}\ \}$, where r_z^i and w_z^i refer to read and write z from/to cell i, which are methods to observe the value of a memory cell and/or create a transition on a memory cell. An example of a march test is March LR [13] as described in Eq. 1. This march test consist of eight components as identified with the superscript M_x where x={0,1,...,7}.

$$\begin{aligned} March_{LR} \;\; = \;\; & \langle \Updownarrow (w_0)^{M_0} \Downarrow (r_0 w_1)^{M_1} \Uparrow (r_1 w_0 r_0 w_1)^{M_2} \Uparrow (r_1 w_0)^{M_3} \\ & \Uparrow (r_0 w_1 r_1 w_0)^{M_4} \Updownarrow (r_0)^{M_5} Del \Updownarrow (r_0 w_1 r_1)^{M_6} Del \Updownarrow (r_1)^{M_7} \rangle \end{aligned} \tag{1}$$

In this paper march tests are used to generate the necessary test sequence to satisfy burn-in, manufacturing and power-on reset memory test procedures.

3 Proposed Programmable Building Block Memory BIST Unit

A memory BIST unit could be composed of: 1) controller, 2) address generation, 3) data generation, 4) response analyzer, 5) multiport/serial memory and 6) scan active/diagnostics. Depending on scan-based, dedicated, integrated and non-integrated memory BIST methodologies, there are differences in the design and implementation of the memory BIST unit. These differences are highlighted and described wherever applicable.

3.1 Memory BIST Controller Unit

The memory BIST controller unit 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. In the proposed approach the memory BIST controller is divided into: parametrized Finite State Machine (FSM) based lower level controller and microcode based top level controller.

3.1.1 Finite State Machine Memory BIST Controller Unit

The lower level controller is the realization of the march test components, ie. M_0, as shown by superscripts in Eq. 1. One parameter driven FSM is designed to realize up to eight different march test components as summarized in Eq. 2. An 8-bit input signal to the FSM indicates which march test component should be selected. The lower level controller is capable of generating any published march test algorithm as well a *hammer read test* to detect data retention faults. Non-integrated and integrated memory BIST methodologies have different timing requirements and therefore to support them two lower level memory BIST controllers are designed as shown in Figures 2 and 3 respectively.

$$\begin{aligned} SM_0 &= \langle \Updownarrow (w_d)\rangle; SM_1 = \langle \Updownarrow (r_{\overline{d}} w_d)\rangle; SM_2 = \langle \Updownarrow (r_{\overline{d}} w_d r_d w_{\overline{d}})\rangle; SM_3 = \langle \Updownarrow (r_{\overline{d}} w_d w_{\overline{d}})\rangle \\ SM_4 &= \langle \Updownarrow (r_{\overline{d}} r_{\overline{d}} r_{\overline{d}})\rangle; SM_5 = \langle \Updownarrow (r_{\overline{d}})\rangle; SM_6 = \langle \Updownarrow (r_{\overline{d}} w_d w_{\overline{d}} w_d)\rangle; SM_7 = \langle \Updownarrow (r_{\overline{d}} w_d r_d)\rangle \end{aligned} \quad (2)$$

The lower level controller is initialized by the top level controller. The states of the described lower level controllers could be described as follows: The *idle* state hold the memory test read and write signals at an off-state. State *ScanOp* is only applicable to non-integrated memory test methodology. This state disables the memory BIST unit and activates a scan operation to load/unload the test data to/from the latches around memories. The *scan_param[3]* consist of 3 enabling signals for the data_input, address and data_output scan chains. For non-integrated dedicated memory BIST methodology, this state is only one cycle long and the memory test data are loaded into the latches surrounding the embedded memory in parallel. For integrated memory BIST methodology the *compare* state is similar to the ScanOp state of non-integrated approach and enables the response analyzer unit. *Inc. Address* state increments the address generation unit while *Write Enable* and *Read Enable* states generate a write and read enable signals respectively. The output signals *XSL_Write* and *XSL_Read* are XORed with the input signal *Write_Data* and *Read_Data* signals. The *Done* state turns off the test read and write enable signals and turn on the done signal.

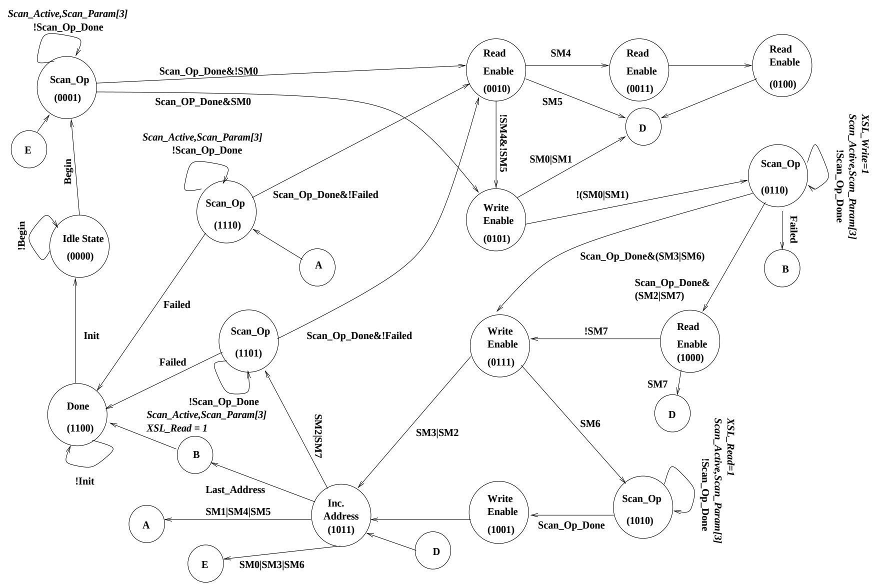

Figure 2: Non-integrated Lower Level Controller

3.1.2 Microcode Based Memory BIST Controller Unit

This microcode based memory BIST controller is a $Y \times Z$ scannable register file or ROM where Y is the width of the microcode and Z is the number of necessary instructions. In this paper the ROM realization of the top level controller is considered. A selector in front of the ROM would select the instruction that should be executed as shown in Figure 4. An instruction counter is a $log_2(Z)$ bit counter that points to the instruction that is being executed. The output of this counter is the control input of the instruction selector. The instruction counter has hold/increment and initialization input signals. The hold/increment signal enables the memory BIST to hold the instruction counter at the current instruction or increment it to the next instruction. The initialization signal, if enabled, sets the contents of the instruction counter to 0. The area overhead of the memory BIST controller depends on the size of the instruction ROM while the number of the necessary instructions depends on the instruction set and determines the size of the instruction ROM.

Two types of *regular* and *branch* instructions are supported. The regular instruction consist of 1-bit instruction type (regular instruction=0;branch instruction=1), 1-bit address order (down=0;up=1), 2-bit write data polarity (non-inverted=0;inverted=1) & (non-checkerboard=0,checkerboard=1)*, 2-bit read data polarity (non-inverted=0;inverted=1) & (non-checkerboard=0,checkerboard=1)*, a 3-bit march test component where each decimal number corresponds to a march test component in the lower level controller, ie. 000=SM_0, 001=SM_1, etc. and one bit to initialize the lower level controller if necessary. The memory test patterns created by the bit marked with an "*" could be merged with a set of data patterns for word-oriented memories and the two bits dropped from the microcode. The regular instruction is shown in Figure 5 (a). The last field determine the march test component that should

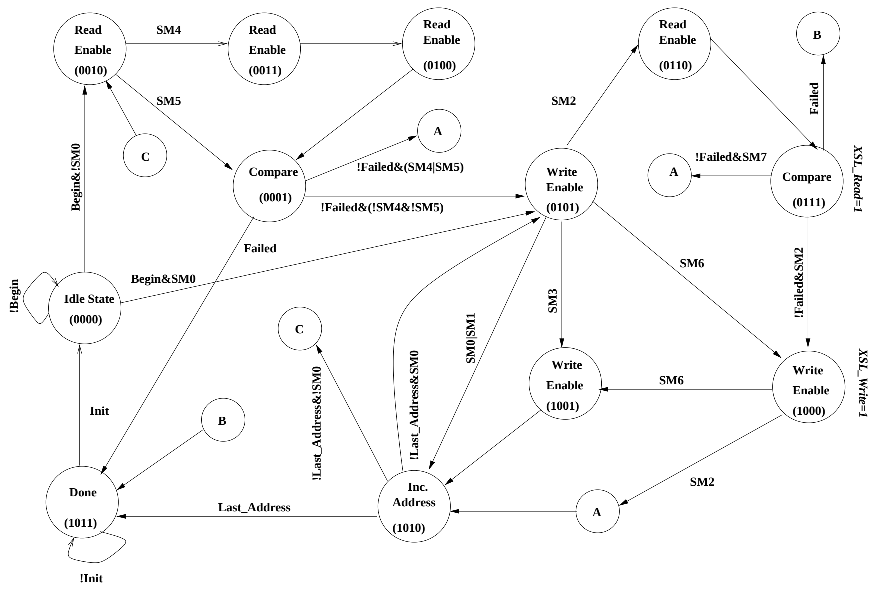

Figure 3: Integrated Lower Level Controller

be initiated by the lower level controller. The branch instruction is used to support multiport and/or word oriented memories and could be eliminated if not applicable. This instruction has instruction type, branch success action and branch condition fields as shown in Figure 5 (b). The grammar of the *branch instruction* could be summarized as:

```
if (< branch_condition >) then
    execute <branch_success_action>
else
    execute < branch_failed_action >
end if
branch_condition = {$fail, \overline{last\ port}, \overline{last\ datapattern}$}
branch_success_action = {$goto\ top$, $increment\ signal = ON$}
branch_failed_action = {$exit$}
```

To further demonstrate the top level controller burn-in, March LR (described in Eq. 1) and power-on reset memory tests are compiled to the supported microcode as shown in Tables 1, 2 and 3 respectively. To support word oriented memories an instruction, similar to $instruction^5$ of Table 1, to check the status and increment the data generation unit could be added to the example microcodes. For both regular and branch instruction, the first column of these Tables is the instruction number, second and last columns are instruction type and comments. For regular instruction the third column is the address order, the fourth column is the write inverted or checkerboard, the fifth column is read inverted and checkerboard fileds, the sixth column specifies the march test component and the seventh column initializes the lower level controller($\overline{initialize}$=0;initialize=1). In the case of branch instruction, the second and the first bit of the third column indicates the branch success fields, the second bit of third column and the fourth

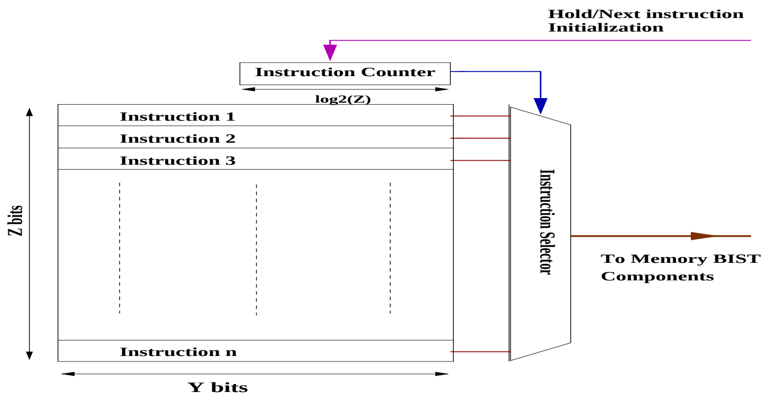

Figure 4: Controller of a Microcode Based Memory BIST Unit

Instruction Type	Address Order	Write Data Polarity	Read Data Polarity	March Test Component	Initialize

(a)

Instruction Type	Branch Success Action	Branch Condition	Branch Failed Action	Unused	Initialize

(b)

Figure 5: Description of (a) Regular Instruction, (b) Scan Active, (c) Branch Instruction

column refers to the branch condition. The first bit of the fifth column is the branch failed action and the sixth column initializes the lower level controller.

Table 1: Microcodes for Burn-in Test

1	0	0	01	00	000	1	SM0 ; $\Downarrow(w_{cb})$
2	0	0	10	01	100	1	SM4 ; $\Downarrow(r_{cb}\ r_{cb}\ r_{cb})$
3	0	1	11	11	000	1	SM0 ; $\Uparrow(w_{\overline{cb}})$
4	0	1	11	11	100	1	SM4 ; $\Uparrow(r_{\overline{cb}}\ r_{\overline{cb}}\ r_{\overline{cb}})$
5	1	1	10	01	100	0	if !last port then increment port selection and goto top else exit

3.2 Address Generation Unit

The address generation unit is responsible to generate the necessary memory test addresses for the memory array under test. The address generation unit can be constructed from Linear Feedback Shift Register (LFSR), binary counter, gray counter or maximum transition counter. The LFSR has a very simple structure, introduces a small logic overhead and the delay between the addresses is small. This solution is only applicable for symmetric march tests since they do not depend on the order of memory array addresses. The binary counter has more complex logic. This type of counter can be designed as carry look-ahead in order to reduce the delay necessary between any two adjacent address sequence. Binary counters are useful for test algorithms that have to be applied with a very specific address order,

Table 2: Microcodes for March LR Algorithm

1	0	0	00	00	000	1	SM0 ; $\Downarrow(w_0)$
2	0	0	10	00	001	1	SM1 ; $\Downarrow(r_0\ w_1)$
3	0	1	00	10	010	1	SM2 ; $\Uparrow(r_1\ w_0\ r_0\ w_1)$
4	0	1	00	10	001	1	SM1 ; $\Uparrow(r_1\ w_0)$
5	0	1	10	00	010	1	SM2 ; $\Uparrow(r_0\ w_1\ r_1\ w_0)$
6	0	0	00	00	101	1	SM5 ; $\Downarrow(r_0)$
7	1	1	10	01	100	0	if !last port then increment port selection and goto top else exit

Table 3: Microcodes for Power-ON Reset Test

1	0	0	00	00	000	1	SM0 ; $\Downarrow(w_0)$
2	0	0	10	00	001	1	SM1 ; $\Downarrow(r_0\ w_1)$
3	0	1	00	10	001	1	SM1 ; $\Uparrow(r_1\ w_0)$
4	0	0	00	00	101	1	SM5 ; $\Downarrow(r_0)$
5	1	1	00	01	010	0	if !last port then increment port selection and goto top else exit

eg. checkerboard test. Maximum transition address counters are the counters that create the maximum transition in the address decoder in order to test the speed that the address decoder can handle the transitions. This type of counters are primarily used to create transitions on the critical paths of the address decoder to detect its marginal defects. The specific design of the address decoder have to be known in advance to design an efficient maximum transition address counter.

Depending on the application, any type of address generation unit could be used in our proposed memory BIST unit. The address generation unit should accept a 3-bit input signal to perform initialization to 0, hold/increment the current address and the address order. The address generation unit should also indicate if the last address sequence has been reached. This unit can be designed as serial or parallel address generation units to support both scan-based and dedicated memory BIST methodologies. In serial address generation unit, the contents of the address generation is shifted out from the scan path similar to a shift register. The address output of this approach is only 1-bit wide. In parallel address generation unit, the test address is supplied in parallel via a test address BUS to the address ports of the memory under test. The output signals are connected to the address ports of the memory under test via a set of surrounding (isolation) latches. The width of the BUS is the address space of the memory array. A latch is used to indicate if the maximum allowed address space is reached. The output of the latch is sent back to the memory BIST controller. This latch is reset with the address generation unit.

3.3 Data Pattern Generation Unit

Data pattern generation unit generates the necessary patterns for the embedded memories. Depending on the test methodology and characteristics of the embedded memories, ie. word-oriented vs. bit-oriented, the generated memory test patterns could be a string of 0's (1's), checkerboard ($\overline{\textit{checkerboard}}$) or a more complex and involved set of data background patterns. The data background patterns are necessary for word-oriented embedded memories and the complexity of the data background pattern generation unit depends on their structure. *Static* or *dynamic* methods for generation of data patterns could be used. In the static approach the data pattern generation unit is a two dimensional circular buffer where the externally generated data patterns are loaded and stored in the unit. In the dynamic method the data pattern generation unit is a state machine that generates the necessary data patterns dynamically during the memory test. A set of data pattern generation units with different widths could be designed and stored in a library. The suitable data generation unit is selected and added to the proposed memory BIST unit.

To demonstrate the connection of a data generation unit to our memory BIST unit and avoid unnecessary complexities a bit-oriented embedded memory is assumed. A bit-oriented data pattern generation unit for scan-based memory BIST methodology is designed as shown in Figure 6 (a). The *init* signal initializes the register to 0, the *checkerboard* signal creates a checkerboard pattern and the *invert* signal inverts the out memory test data pattern. The register of the data generation unit is in the same scan chain as the surrounding latches. The data generation unit for dedicated and integrated memory BIST methodologies are constructed with an XOR gates as shown in Figure 6 (b). The *even bits* and *odd bits* signal fans out to the functional input ports of the surrounding latches or data input of the even and odd numbered bits respectively.

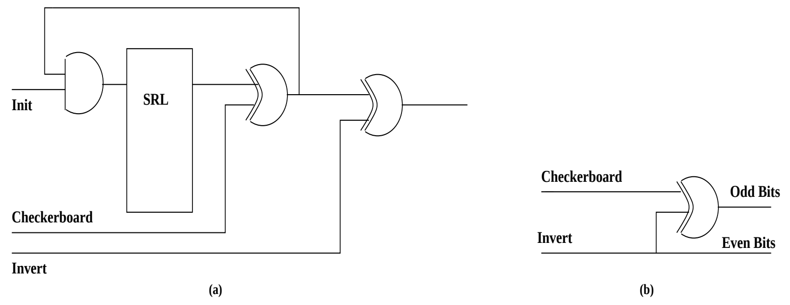

Figure 6: An Example of Data Generation Unit

3.4 Response Analyzer Unit

This units compares the actual and expected output response of the embedded memory and could be designed as a serial or parallel response analyzer to support both scan-based and dedicated memory BIST methodologies. The serial response analyzer compare the actual and expected output response of an embedded memory one bit at a time. The output of the comparator is ANDed with the multi-port/memory unit of that port/memory which allows the output of active ports to be analyzed. An example of this unit is shown in Figure 7 (a). The parallel response analyzer unit consists of an eXclusive OR (XOR) tree whose output is ORed and then ANDed with the port enable signal in the case of multiport embedded memories. The output of the AND gate is called *port Fail/Pass signal* and determines if a port has passed or failed the test. A port Fail/Pass signal exists for each port of the memory array. These signals are ORed together to create a *memory array Fail/Pass signal*, unless memory arrays are clustered in sharing serial configuration. In this case, the output of the OR gate is ANDed with the memory array enable signal and the output of the AND gate is called *memory Fail/Pass signal*. Each memory array has a memory Fail/Pass signal that is feed to a Fail/Pass latch. This latch enables the diagnostics to determine which memory array has failed the test. To reduce the test logic overhead, the memory Fail/Pass signals of each cluster could be ORed together and feed one Fail/Pass latch. The output of the multiple Fail/Pass latches are ORed together to create a cluster Fail/Pass signal and is fed back to the memory BIST control signal as shown in Figure 7 (b).

3.5 Multiport/Memory Enable Unit

The multiport/memory enable unit is only applicable if the embedded memories under test are

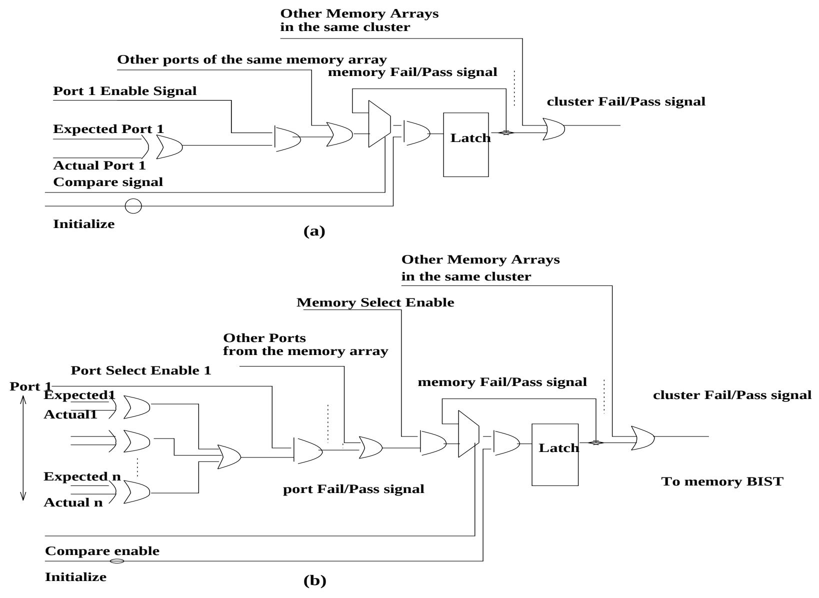

Figure 7: (a) Serial and (b) Parallel Response Analyzer Unit

multiport and/or are tested serially. Serial testing of embedded memories is necessary for the designs with power constraints or to support diagnostics. This unit could be eliminated from the memory BIST unit if not applicable. The multiport/memory enable unit consist of an W bit cyclic shift register with hold/increment and initialize input signals. Parameter W is the number of the ports in a multiport or number of memory arrays in sharing serial configuration under the test. The cyclic shift register is initialized with 10....0 and the next port/memory signal from the memory BIST controller would make the cyclic shift register to shift one bit down. An output signal indicates if the last port/memory has been processed. An example of this unit is shown in Figure 8.

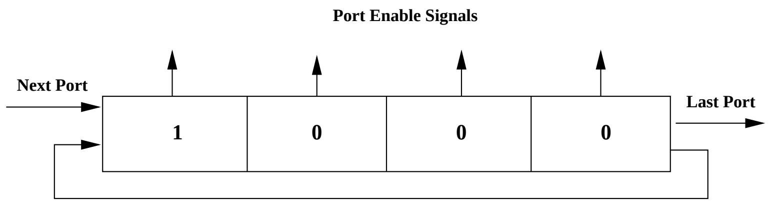

Figure 8: An Example of Port/Memory Select Unit

3.6 Scan Active/Diagnostics Unit

Scan active unit makes the memory BIST controller pauses for a pre-specified number of cycles to load/unload memory test data to/from memory BIST components and the latches surrounding the embedded memory. This unit consists of a decrementing counter and generates a pre-specified number of scan clocks (if applicable), eg. LSSD test methodology, and set the appropriate scan enable signals to prepare the participating scan chains in the scan operation. The rest of the memory BIST unit is in hold state and the read/write enable signals of the memory under test are also disabled. A scan active unit is shown in Figure 9.

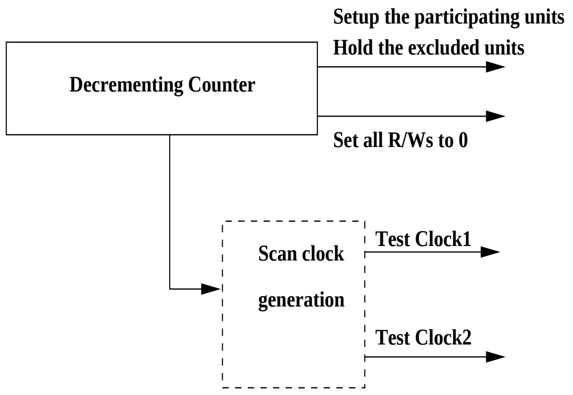

Figure 9: An Example of Scan Active Unit

Depending on the diagnostics algorithm, the diagnostics unit could be integrated with the scan active unit. In this case, a fail signal could also trigger the scan active unit to scan-out the memory test data. In this paper, this diagnostic methodology is assumed.

4 Experimental Results

To evaluate the overhead of the proposed memory BIST controller unit described in this paper, we have designed a set of programmable controllers using different approaches. The first controller is a programmable two level FSM based controller where both top and lower levels are designed and implemented using a finite state machine approach. The second controller is scan-based microcode based controller where two cycles are used to execute each instruction. The third controller is a scan-based microcode based controller where each instruction is executed by only one clock cycle. The fourth controller is a microcode based controller for a dedicated memory BIST methodology where each instruction is executed in one cycle. The comparison of logic overhead of other approaches vs. our reconfigurable memory BIST unit is summarized in Table 4. The first column represents the memory BIST methodology, the second column is the number of D-FFs. The source of D-FF could be the latches required for the FSM, branch register or latches to hold the control signals. The fourth and fifth columns are the instruction ROM and counter respectively. The sixth column is the number of 2:1 multiplexors in the memory BISt controller. The main source of these multiplexors are the branch registers. The seventh and eight columns are the AND and OR gates in the controller. The main source of these gates is the FSM. The last column is the size of the selector.

Based on the results summarized in Table 4, our proposed reconfigurable memory BIST methodology has lower logic overhead than the pure FSM based approach. In the pure microcode based approaches, our reconfigurable memory BIST scheme could generate all known march tests with a negligible increase in the size of the top level controller while the pure microcode based approach might result in a much larger controller. Furthermore, the microcode of our approach is very simple and therefore writing additional code is trivial. In the pure microcode based approach the microcode is very complex and generation of the necessary microcodes is cumbersome and error prone.

Table 4: Overhead of different memory BIST controllers

Method	*Sequential Elements*			*Combinational Logic*			
	D-FF	ROM	Prog. Counter (bit)	MUX	AND	OR	SELECTOR
FSM Based	**15**	**0**	**0**	**0**	**480**	**110**	**0**
Microcode Scan-Based Multi-cycle	**5**	**19x13**	**5**	**10**	**5**	**0**	**19x13:13**
Microcode Scan-Based Single-cycle	**5**	**28x11**	**5**	**10**	**5**	**0**	**28x11:11**
Microcode Dedicated Single-cycle	**5**	**25x11**	**5**	**10**	**5**	**0**	**25x11:11**
Reconfigurable Scan-Based & Dedicated Single-cycle Clock	**4**	**7x10**	**3**	**0**	**78**	**20**	**7x10:10**

To further evaluate the overhead of the proposed reconfigurable memory BIST, we have implemented a complete scan-based, non-integrated dedicated and an integrated dedicated memory BIST unit using a parametrized VHDL. The VHDL is first compiled to a Register Transfer Language (RTL) using HIS^{TM} (High Level Synthesis). For manufacturing testing, Level Sensitive Scan Design (LSSD) test methodology is used and the necessary test logic is inserted by using an early mode[14] Design For Test Synthesis (DFTS) tool. The isolation logic are automatically inserted [3] using DFTS tool. $BooleDozer^{TM}$ is used for logic optimization, timing correction and technology mapping. $TestBench^{TM}$ is used for behavioral simulation and verification. For scan-based memory BIST methodology two scan chains are created as described in Table 5.

Table 5: Scan Chain Configuration for Scan-Based Memory BIST Methodology

Scan Chain Name	*Components*
LSSD $CHAIN_1$	**Data-input, output and address isolation scannable registers, data generation unit, address generation unit, response analyzer unit**
LSSD $CHAIN_2$	**top controller, lower controller, multiport enable unit, last address register**

The size of different reconfigurable memory BIST methodologies has been measured in terms of a 2-input NAND gate and is summarized in Table 6. The first column of Table 6 represents the memory BIST methodology, the second through eight columns show the area of lower level controller, address generation unit, data generation unit response analyzer unit, multiport/serial enable memory, scan active unit and the isolation logic respectively. The last column represents the total size of each memory BIST methodology based on a 2-input NAND gate. The top controller was not included while the scan active and diagnostics units are integrated. For non-integrated memory BIST methodology, the embedded memory is assumed to be surrounded by scannable latches while in the integrated method only the multiplexors are used.

The experimental results shows that the logic overhead of integrated memory BIST methodology

Table 6: Size of the proposed memory BIST methodology

Memory BIST	*Components*							*Total Size*
	lower contr.	adrs gen.	data gen.	resp anal.	port select.	scan active.	isolation logic.	
Scan-based	182	175	11	11	16	28	421	844
Dedicated non-integrated	182	210	2	81	16	28	415	978
Dedicated integrated	124	210	2	81	16	28	148	609

produces the lowest logic overhead. This is because of a slightly smaller lower level controller and the absence of the isolation logic. The scan-based memory BIST methodology produced lower logic overhead than the non-integrated dedicated memory BIST unit. It is mainly due to smaller address generation and the fact that the isolation logic do not require multiplexors. The overall logic overhead of our proposed memory BIST methodology is comparable with other conventional methods while our methodology offers a great deal of flexibilities compare to the conventional methods.

5 Conclusion

In this paper the design and architecture of a reconfigurable memory BIST unit was presented. The proposed memory BIST unit could accommodate changes in the test algorithm with no impact on the hardware. Different types of march test algorithms could be realized using the proposed memory BIST unit and therefore memories with different characteristics and test requirements could use the same memory BIST architecture. By integrating the diagnostics with this memory BIST unit further reduction in the cost of test was achieved.

The experimental results showed that the logic overhead of our proposed memory BIST methodology resulted in lower logic overhead than other programmable methods. Also, implementation of scan-based, non-integrated and integrated dedicated memory BIST methodology illustrated the logic overhead of each method compare to each other and other conventional methods. The proposed memory BIST methodology has comparable logic overhead compare to other conventional methods while provide flexibilities. In addition, our building block approach allows the designers to reuse the proposed memory BIST unit for different designs.

References

[1] L. Ternullo. D. Adams. J. Connor. G. Koch. Deterministic self-test of a high-speed embedded memory and logic processor subsystem. *Proc. Int. Test Conf.*, 1995.

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

[3] S.J. Upadhyaya K. Zarrineh and P. Shephard III. Automatic insertion of scan structures to enhance testability of embedded memories, cores and chips. In *Proc. VLSI Test Symposium Conf.*, pages 98–104, 1998.

[4] T. Takeshima H. Koike and M. Takada. A BIST scheme using microprogram ROM for large capacity memories. In *Proc. Int. Test Conf.*, 1990.

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

[6] A.J. Van de Goor and G.N. Gaydadjiev. An analysis of (linked) address decoder faults. In *In Records of the IEEE. Int. Workshop on Memory Technology, Design and Testing*, pages 13–20, August 1997.

[7] Rob. Dekker. Fault modeling and test algorithm development for static access memories. In *Proc. Int. Test Conf.*, October 1988.

[8] A.J Van de Goor etal. March lr: A test for realistic linked faults. In *Proc. VLSI Test Symposium Conf.*, pages 272–280, March 1996.

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

[10] M. G. Karpovsky and Y. N. Yamolik. Transparent memory testing for pattern sensitive faults. In *Proc. Int. Test Conf.*, pages 860–868, October 1994.

[11] Bruce F. Cockburn and Y.-F. Nicole Sat. Synthesized transparent bist for detecting scrambled pattern sensitive faults in RAM. In *Proc. Int. Test Conf.*, pages 23–32, 1995.

[12] Marian Marinescu. Simple and efficient algorithms for functional ram testing. In *Proc. Int. Test Conf.*, October 1989.

[13] V.N. Yarmolik A.J Van de Goor, G.N. Gaydadjiev and V.G. Mikitjuk. March LR: A test for ralistic linked faults. In *Proc. VLSI Test Symposium Conf.*, pages 272–280, March 1996.

[14] V. Chickermane and K. Zarrineh. Addressing early deisgn-for-test synthesis in a production environmet. In *Proc. Int. Test Conf.*, pages 246–256, 1997.