The Report Committee for William Hugh O'Donnell
Certifies that this is the approved version of the following report:

A Programmable MBIST with Address and NPSF Pattern Generators

APPROVED BY

SUPERVISING COMMITTEE:

Nur Touba, Supervisor

Jacob Abraham

A Programmable MBIST with Address and NPSF Pattern Generators

by

William Hugh O'Donnell, B.S.E.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2013

Dedicated to my family: Tanya, Willow and Eslee.

Acknowledgments

I would like to thank my supervisor Professor Nur Touba for all of his advice and encouragement. I would also like to thank Professor Jacob Abraham for agreeing to be the reader for this report. Both have made significant contributions to memory BIST designs and I am honored they are willing to work with me on this report.

This work would not have been possible without all the love and support I received from my family: my parents John and Lolita O'Donnell, my in-laws Peter and Belvia Ortega, my siblings Rosemary and John. Their support has carried me through graduate school. I cannot thank each of these people enough.

Finally, I would like to thank my classmates, especially Robert, Anil and Miguel, for their support, camaraderie and knowledge. Without them working late nights and long weekends with me on our various assignments, labs and projects, I would not be where I am today.

A Programmable MBIST with Address and NPSF Pattern Generators

William Hugh O'Donnell, M.S.E.
The University of Texas at Austin, 2013

Supervisor: Nur Touba

The movement to smart mobile connected devices which consolidate functions of traditionally separate devices is driving innovation in System-on-chips (SoCs). One of the innovations helping to meet the current needs of SoCs is the integration of larger memory with the processor, and with this, comes the challenge of testing all the memory cells. The programmable memory BIST offers a flexible approach to designers and testers because it allows the memory test algorithms to be updated when new memory fault models are discovered. But this flexibility comes at a trade-off to area as the BIST circuitry needs to be integrated next to the memory array. This report proposed some improvements to an existing design that will improve flexibility by enhancing the address generation schemes while simultaneously eliminating the need for an auxiliary memory in cases where a Type-1 NPSF background will be used. The areas of the base design is compared to the proposed design to show the address and data generation improvements can be achieved with only 1.8% increase in area with an 8KB memory.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The movement to smart mobile connected devices which consolidate functions of traditionally separate devices is driving innovation in System-on-chips (SoCs). One of the innovations helping to meet the current needs of SoCs is the integration of larger memory with the processor. As the need for memory has increased, SoCs have moved from logic dominant chips to memory dominant chips. In fact, [10] estimates 94% of the SoC will be dedicated to memory by 2014.

Because embedded memories are designed with aggressive design rules to increase the density of transistors per area, they are more prone to manufacturing defects that negatively affect the overall chip yield. Testing memory to identify faulty blocks is critical to increase and maintain acceptable yield [18]. A built-in self-test (BIST) has previously been used to test logical elements of the chip, but now have been adapted to test memory as well. The density of memory and difficulty in accessing all signals has driven the growth of memory built-in self-test (MBIST) as the only practical and cost-effect solution for embedded SoC memories [20].

As not all fault models can always be predicted for a new product, pro-

grammable MBIST (PMBIST) designs allow developers to address fault detection, create efficient algorithms, and overcome challenges presented by new technology. Conversely, a non-programmable MBIST requires the designer to correctly predict the fault models to be tested, but changes to the design or mask typically requires another tape-out and can be quite expensive. Because of this, the programmable MBIST has gained popularity amongst memory designers.

The most common memory test today is the March test because its structured and repetitive algorithms can be applied very easily to the symmetric nature of memory cells [21]. The design presented in [3] primarily uses the March test to verify the integrity of the memory. In general, there are three components of memory tests: the test algorithm or operations in each cell; the data; and the memory address sequence [9].

The PMBIST design presented in [3] will be used as the base design and is intended to provide an area efficient design with good flexibility on algorithm and test data programming. This report proposes improvements to the design by providing programmability to the address order and counting method. The previous design uses a linear counter and pseudo-random counter to sequence through memory addresses. This allows the BIST to run March algorithms, but another algorithm, Moving Inversion (MOVI) cannot be run because it requires 2^i address counting method. MOVI is popular industry for its ability to detect read and write disturbance while also allowing for determination of the best and worst case memory access times [6]. This project adds the 2^i

counting method along with Gray coding and Address Compliment to integrate common industry test addressing sequences into the BIST.

In addition, this paper will show a reduction of area can be achieved by introducing a dynamic pattern generation block to replace the auxiliary memory used to store neighborhood pattern-sensitive fault (NPSF) backgrounds. In the base design, the data pattern for auxiliary memory can initialized through a scan mechanism [**?**], but relying solely on scan to load data patterns can be time consuming for serial BIST schemes. Additionally, the MOVI fault model requires an NPSF data background of Type 1 to be effective. Because of this, this report integrates an NPSF data background generator to remove the scan delay. This provides faster access time and potentially removes the need for an auxiliary memory thus speeding up test time and saving area. In achieving these design goals, this report will show enhancements to the [3] design in two of the three memory test components.

This paper is divided into five chapters. Chapter 1 offers motivation for this project and supplies definitions for common terminology. Chapter 2 provides background information on the memory faults, current memory testing algorithms and designs to address some of the challenges of memory testing. Chapter 3 presents details of the design proposed in this report and chapter 4 follows with testing results and comparisons to other designs. Finally, chapter 5 concludes the report.

1.1 Memory BIST Terminology

BIST - Built-In Self-Test

MBIST - Memory BIST

PMBIST - Programmable MBIST

SoC - System-on-Chip

NPSF - Neighborhood Pattern-Sensitive Fault

ANPSF - Active NPSF

PNPSF - Passive NPSF

SAF - Stuck-At Fault

CF - Coupling Fault

TF - Transitional Fault

Chapter 2

Memory BIST Fundamentals

Memory testing has many parts, but in general there are three components: the test algorithm or operations performed in each cell; the data written or read from the cell; and, the memory address sequence or counting method [9]. The basic concept is to confirm that a value written to a memory address is still present in that memory address at any point in the future. Many factors can affect the ability of the memory to retain its proper value. Those factors manifest as one of the four standard memory faults: Stuck-At Faults, Transitional Faults, Coupling Faults and Neighborhood Pattern-Sensitive Faults [6]. All current BIST engines are able to detect these faults with varying degrees of effectiveness. The BIST engines use memory test algorithms to detect, and in some cases, repair these faults.

The address counting method and address order are major influences to a test algorithm's ability to detect these faults. A BIST needs an address generator that can produce the correct sequence of addresses without omitting any addresses.

2.1 Memory Faults

Precise fault modeling is the key to designing efficient fault tests. Fault models reflect real, specific defects in memory so high defect coverage and detection is strongly dependent on the quality of the fault model [10]. The following sections offer a brief description about the four classic types of faults [2] that models are designed to detect.

2.1.1 Stuck-At Faults

The most common type of memory fault occurs when the memory cell is locked into one state, either a 0 or 1. A defect free cell can be written to either state and, when read, will still contain the information previously written. The stuck-at fault occurs when a state is written to the cell, but the subsequent reads return the only one state regardless of the previously written value. Fig. 2 shows the state diagram for a stuck-at fault [**?**]. Another fault that can be classified as a stuck-at fault is the address decoder fault. It is like the data stuck-at fault, but occurs on the address lines and leads to accessing wrong addresses, no addresses, or multiple addresses [**?**].

2.1.2 Transitional Faults

Similar to a stuck-at fault, the transition fault also locks into a single state, but has the characteristic of being in either state prior to the write. The memory cell may contain 0 or 1 when powered on, but after a write, it cannot transition back. The characteristic behavior of this fault is the memory can

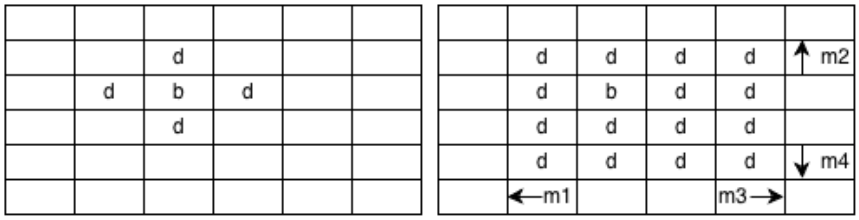
only written in one direction.

2.1.3 Coupling Faults

There are numerous types of coupling faults, but they can be simply expressed as a cell affecting its neighboring cells and causing the neighbor to falsely transition or change state. Coupling faults can be unidirectional or bi-directional. In unidirectional coupling faults, one cell (aggressor cell) couples into another (victim cell), but the opposite does not occur. A parasitic diode connection between the cells is a common cause of this behavior. The bi-directional coupling fault occurs when pairs of cells can affect each other. One way that this type of defect can occur is through bridging [2].

2.1.4 Neighborhood Pattern-Sensitive Fault (NPSF)

This fault model is in the class of coupling faults, but it is caused by particular patterns in neighboring cells rather than one specific cell. The neighborhoods are usually defined as Type-1 or Type-2 neighborhoods [4] as shown in Figure 2.1. Type-1 neighborhoods consist of five cells: one cell in center and four cells physically adjacent to the center cell. In this configuration, the center cell is the base cell and the four adjacent cells are called the deleted neighborhood. Type-2 neighborhoods consist of multiple cells where the base cell is in the center and deleted neighborhood is comprised of the cells within m_1 columns to the left, m_2 rows above, m_3 columns to the right and m_4 rows below the base cell. This report will focus on Type-1 neighborhoods.



(a) Type-1 Neighborhood (b) Type-2 Neighborhood

Figure 2.1: NPSF Neighborhoods
b: base cell
d: deleted neighborhood

Three classes of NPSF faults exist:

1. Active NPSF (ANPSF): the base cell's contents change due to changes in the pattern of the deleted neighborhood.

2. Passive NPSF (PNPSF): the base cell's contents cannot change due to specific pattern in the deleted neighborhood.

3. Static NPSF (SNPSF): the base cell's contents are forced to a specific value because of the contents of the deleted neighborhood

To test for all three classes of NPSF faults, all the combinations of values in the deleted cells and their transitions must be performed against the base cell. To do this, a Eulerian Sequence is employed to generate the appropriate sequence of values in the neighborhood. [12] offers a proof of the Eulerian Sequence as a testing mechanism. A 5-bit Eulerian Sequence is used as the Type-1 pattern.

To reduce the number of write operations to memory, multiple patterns can be written to memory simultaneously using either the tiling or two-group methods as shown in Figure 2.2.

Figure 2.2: Type-1 Neighborhood Tiling and Two-Group Method

In the tiling method, each tile is written to memory is such a way that none of the tiles overlap. The two-group method is comprised to two checkerboard patterns that are overlaid is such a way that a base cell is in one checkerboard pattern while the deleted neighborhood is within the other checkerboard pattern. This report will use the tiling method.

2.2 Memory Test Algorithms

The early tests before the 1980s were not based on fault models or proofs. Tests of this time, such as Scan, GALPAT and Walking, were considered Ad-Hoc and usually have very long test times of order O(n2) making

the non-ideal for larger memories [10]. The introduction of fault models in the early 1980s allowed the fault coverage to be mathematically proven. This paved the way for March tests to become the dominant type of tests since its fault coverage is proven mathematically and its test time is linear to the size of the memory [10].

2.2.1 GALPAT and variations

The GALPAT or Galloping 1s and 0s algorithm sequences through all memory addresses to test for faults. It starts by writing a background of zeros into all memory cells, then complements the first cell and compares it to every other cell, reading the base and test cell every time. The test continues testing every cell by complementing the current cell and comparing it to every other cell [**?**]. The test is effective for finding stuck-at faults, coupling faults and transition faults, but suffers from an extremely long test time compared to march tests [10].

2.2.2 Walking Algorithms

Walking algorithms are similar to GALPAT tests. They begin by initializing all memory to zeros and then complement the first test cell, or base cell. The test reads the base cell and then reads each of the memory cells to compare results without re-reading the base cell. The test effective for finding stuck-at faults, but also suffers from test times proportional to O(n2) [**?**].

2.2.3 March Test

The March test has become the most popular testing algorithm for memory. The March test algorithm is a finite sequence of March Elements (ME), each of which can be shared between other algorithms. A ME specifies the sequence of March operations applied to each cell before proceeding to the next cell [24]. A March operation (MO) is that action performed at the memory cell: write a 0 (w0), write a 1 (w1), read a zero (r0) and read a 1 (r1) [24].

The order in which the memory addresses are tested is the address order (AO), and the actual sequence of addresses is the counting method (CM). For example, for the range $1 \ldots 10$, the sequence 1,2,3,. . . ,9,10 would have an ascending AO and linear CM. Conversely, 10,9,. . . ,3,2,1 would have a descending AO and a linear CM. The AO uses the symbols $\Uparrow$ (ascending),$\Downarrow$ (descending),$\Updownarrow$ (AO irrelevant) as abbreviations in the ME definition [24].

The ME $\Updownarrow$(r0,w1), is interpreted at each memory address as read the memory with expected value of 0, the write a value of 1 and continue to the next address in ascending order [23].

2.3 Memory Address Counting Methods

For any number N, there are $N!$ ways to count to N. The memory address counting methods (CM) are important because they can directly affect a test's effectiveness at detecting faults. Running the test with $N!$ CM's is not

practical with today's memory sizes. The address generator in this report will focus on those that are common and important. Each of the CM's included has its own fault detection capability [7], [16], [22], [25], [11].

2.3.1 Linear Sequence

The linear sequence CM is the standard numerical sequence. Adjacent addresses differ by one numerical value. The up '⇑' sequence is 0, 1, 2, 3, ..., 2^N-1 while the down '⇓' sequence is 2^N-1, ..., 3, 2, 1, 0. Single-cell and coupling faults can be detected with this CM [26].

2.3.2 Address Complement

Address complement counting method (ACCM) specifies an address sequence where pairs of addresses are formed using the address and its one's complement. A four-bit address sequence would be: 0000, **1111**, 0001, **1110**, 0010, **1101**, etc [6]. In this series, the *even steps* form a linear sequence while the *odd steps* (in **bold**) are formed with the one's complement of its corresponding even pair. This CM forces all bits to change in a transition between pairs and causes large amounts of noise, large power surges and maximum delay; it is ideal for detecting speed-related faults.

2.3.3 Gray Code

Gray code is a binary numbering system where successive values differ by only one bit [6]. In the context of memory addressing, the address tran-

sitions will differ only in one bit (i.e., they have a *Hamming distance* of 1). This CM produces minimal noise, power and delay and is used for the minimal stress tests [26].

2.3.4 Worst Case Gate Delay

Speed related faults can be detected with the worst case gate delay counting method (WCGDCM). For every address, the WCGDCM creates N address-triplets consisting of the original address, the original address with a single bit inverted, and the original address again. The address-triplets in WCGDCM have a Hamming distance of 1 [25].

2.3.5 2^i Sequence

2^i counting method uses address pairs that differ in *one* bit. The i specifies the numerical difference between two pairs of numbers and also the bit that will be incremented or decremented. For example, when $i = 3$, $2^3 = 8$, so all address pairs in this sequence will vary by the bit 3, or by a value of 8. This is a popular CM used to detect speed-related faults, especially in the MOVing Inverions (MOVI) test [26].

2.3.6 Pseudo-Random

A pseudo-random CM generates a sequence of addresses that appear to be random, but are deterministic and can be reproduced. These sequences are commonly generated using LFSR's which implement at characteristic polyno-

mial function [6].

2.4 Challenges in Testing Embedded Memory

2.4.1 At-Speed

To effectively detect memory faults, the test must be run at least at the maximum operating clock frequency for the memory device. At-speed testing is important because timing for the chip may only be closed at limited test corners or external testing at high speed may be difficult or even impossible. Transition faults may not be detected if the chip is run below maximum operating frequency [5].

2.4.2 Back-to-Back (BtB) Access

Back-to-Back (BtB) accesses are necessary to detect faults that may occur when an action is repeated on a memory cell. BtB means the CPU uses the smallest number of clock cycles to access memory [23]. One fault that may only present during BtB testing is a cell suffering from read disturbance and losing its charge after 16 consecutive reads [15]. The faults are subtle enough that external testing may not be able to detect is the correct sequence or repetitions is not executed. Detecting these types of faults requires augmenting existing March test algorithms to effectively screen parts.

2.4.3 Area

The area cost of MBIST circuits is usually small compared to the memory under test, but for an SoC consisting of hundreds of memory cores [14] or programmable memory BIST with a large amount of instruction memory [19], the area overhead for the BIST circuits will be very high. To continue Moores law and maintain acceptable yields with the larger memory sizes, the BIST circuits will require innovation to reduce their footprint.

2.4.4 Power

Power dissipation during the MBIST is an important challenge because of the problems that may occur. High power during test along with the high switching frequency can cause excessive noise that can change the state of a circuit and cause a good die to fail the test and reduce yield [**?**], or cause circuit damage and destroy the chip [**?**], again reducing yield.

2.5 Designs to Address Challenges

MBIST is a hot-topic due to its increased importance in the new memory-dense SoCs. The challenges with the current MBIST topologies are well known. New innovations and design methods have been introduced to overcome these obstacles.

2.5.1 Topologies

There are three types of BIST topologies commonly used: finite state-machine (FSM)-based designs, micro-programmable based designs and microcode-based designs. Each has advantages and disadvantages that designers must consider to meet the requirements of their product.

2.5.1.1 FSM-Based Designs

In FSM-based designs, the control signals for the BIST are defined as state machines and are usually hardwired, or non-programmable. The FSM-based topology benefits from having a smaller area, but lacks flexibility to change its algorithm to support new fault models without redesign of the chip [19]. New FSM-based designs are beginning to incorporate some level of programmability, but at the cost of area [13] or clock frequency and test time [**?**].

2.5.1.2 Micro-Programmable-Based Designs

The micro-programmable-based designs use a microprocessor to generate the algorithm. These designs have high flexibility with their algorithms, but in-turn require a high area overhead for the processor [8]. Additionally, this type of design is not well suited for a stand-alone embedded RAM intellectual property (IP) core because of the processor requirement [3].

2.5.1.3 Microcode-Based Designs

Microcode-based topologies use written sets of instructions that are loaded into memory to execute memory test patterns. The controllers for microcode-based topologies are designed as programmable MBIST with the flexibility to select any instructions from supported set of test instructions. The test flexibility and ability to reprogram this design allows it to easily support new fault models should a test escape be discovered. The flexibility comes at a cost though in the storage area for the microcode instructions [19], [17]. The area overhead is a carefully evaluated intensively during the selection of the BIST design.

2.5.2 Existing Designs

The following section examines some of the state-of-the-art design methods and innovations currently being explored to address the MBIST implementation challenges. In general, at-speed challenges and to a lesser extent BtB have mostly been addressed by using MBIST to test memories, but area and power challenges still exist which and are being actively researched. The following section focus on BtB accesses, area reduction and power reduction in MBIST as they are the leading topics of research for MBIST improvements.

2.5.2.1 Design Methods for At-Speed and BtB Access Challenges

Even with the use of MBIST, BtB testing presents a particularly difficult problem because the architecture of the CPU and its interactions with the

memory controller must be well understood. Memory failures are now more frequently caused by speed-related faults rather than static faults. To screen for these faults, a test must be able to access memory as fast as the CPU. The design proposed in [23] addresses this issue by using the CPUs assembly language and core to test the memory.

In [23]s proposed solution, a small set of the assembly instructions for basic memory movement and logical functions are used to write March test algorithms. Memory interleaving, or folding, is also addressed in the new algorithms. The algorithms satisfy the BtB test requirements mainly by variations of loop-unrolling or by coding tight jump loops that still allow BtB memory access.

2.5.2.2 Design Methods for Area Challenges

Area is generally a concern for chips with multiple memory cores or with microcode-based designs that offer high flexibility. One potential solution [14] to reducing the area used by multiple memory cores is a pipelined MBIST for homogeneous RAMs. The paper identifies the test pattern generator as a significant portion of the BIST area proposes to use only one generator. It accomplishes this sending the pattern to the first memory, then uses the output of the first memory as the input to the second memory effectively pipelining the BIST and reducing the area overhead. The test controller still manages the memory selects, addressing and data verification, but only needs to control one pattern generator which also reducing wire routing area.

Another approach to reduce the MBIST footprint is to optimize the microcode itself [19]. This design takes advantage of the fact ME repeat for various algorithms and furthermore, usually show up in repeated clusters. These clusters are identified and defined as macros, and then the macros are encoded as new microcode instructions. So effectively, a group ME instructions has now become one microcode instruction. The solution is similar to creating a library function in software where it only physically resides in one memory location rather than at each invocation of the function.

2.5.2.3 Design Methods for Power Challenges

MBIST power is a concern because it is generally higher than normal operating power and it can have negative effects on yield. Novel techniques to optimize blocks of the MBIST help to reduce average and peak power usage. One technique [**?**] proposes a low power linear feedback shift register (LFSR) for the test pattern generator (TPG). This paper targets switching activity in the LFSR as one culprit for higher power - in particular, the non-correlated patterns cause higher power dissipation. The paper proposes using intermediate transitional vectors. In between two successive vectors, half of the first pattern is changed and output, then the second half is changed and output, then a few logic gates are added to generate a third output before finally outputting the new pattern. With this piecewise technique, five test patterns are generated while only using power equal to generating two vectors.

The previous proposal showed how optimizing the pattern generation

could reduce power. The next paper discusses ways to optimize the address generation block used for March tests by reducing switching activity. The following sequence is typically used for testing stuck-at faults: (w0), (r0), (w1), (r1) This sequence results in a large amount of switching activity in the address decoder. The paper proposes to instead use this pattern to reduce the activity:

(w0, r0, w1, r1)

The address decoder now only needs to change once per memory address while the stuck-at faults can still be detected. The paper further suggests alternate LFSR designs which offer better correlation between bit patterns. Patterns that are reduce the actual number of bits that change will reduce the switching activity. The Bit-Swapping LFSR (BS-LFSR) swaps values between neighboring bits and reduces switching activity by about 25% [1] while the Bipartite LFSR reduces switching activity by combining the first and second halves of the current vector with the next vector to create intermediate vectors [**?**].

Chapter 3

Proposed Design

The design in this report is based upon [3] which proposes a programmable memory BIST with auxiliary memory for NPSF. This design will modify the programmable memory BIST by introducing a pattern generator to dynamically generate the NPSF Type-1 neighborhood patterns. It will also replace the address counter with a programmable address generator to allow more flexibility in testing specific addresses sequences.

3.1 PMBIST Hardware Blocks

The proposed design is comprised of five major blocks: the cycle controller; the address generation block; the data generation block; the data comparison block; and the operator control register. The blocks and the connections are illustrated in 3.1.

3.1.1 Cycle Controller

The cycle controller determines which march operation should execute on the current address. When all march operations have completed on the current address, the cycle controller generates a signal that allows the address

counter to move to the next test address. The cycle controller then resets the march operation pointer to the first operation for the next address.

3.1.1.1 Control Mux

The control mux receives all the operation and polarity signals from the instruction register. Using the cycle counter's output as the mux control signal, the control mux selects the operation and polarity signal corresponding to the current cycle and outputs them to the operation formatting block and control register.

3.1.1.2 Cycle Counter

The cycle counter increments the count for the march operation to execute. The output of the cycle counter corresponds to the active march operation. The cycle counter is incremented by the clock signal and can be reset by a TS signal or when the current march sequence has completed for the current memory address.

3.1.1.3 Cycle Comparison

The cycle comparison unit compares the current cycle to the NO field of the instruction register. When the cycle counter matches the NO field, the comparison block generates an active high signal that is stored in the cycle controller's local flip-flop. The signal is also sent to the instruction register hold logic block.

3.1.2 Address Generation

The memory address to test and memory control signals are generated by the address and operation block. The address decode block is used to generate the instruction register hold signal. The hold signal maintains the instruction register's data until the current march sequence has completed for all memory address.

3.1.2.1 Address Counter

The address counter indicates the memory address to test. The direction of the address order can be programmed to increment or decrement through memory. The order of addresses is also programmable: linear up/down, pseudo-random sequence using LFSR, address complement, Gray coding, and 2^i counting methods.

3.1.2.2 Address Decode Logic

The address decode logic block determines if the address generated by the address counter is the last up (LU) or last down (LD) memory address for the current march sequence. The decode logic uses the signals from the address programmer block to determine whether the sequence direction is up or down.

3.1.3 Operation Control Block

In some designs, the memory algorithm operation signals from the instruction register do not necessarily need to match the memory's control signals. The operation formatting block can be used to translate the instruction register's operation code to the memory controller's signals such as write/read, enable and reset.

3.1.3.1 Operation Formatting

The Operation Formatting block converts the memory algorithm operation signal to explicit memory control signals such as read/write, enable and reset. The output of this block passes to the control register.

3.1.3.2 Operation Control Register

The control register interprets the operation formatted instruction and sends any internal control signal to other blocks of the PMBIST. In particular, this block sends the data mux select signal and controls the sequencing of the data generator. It also writes the memory control signals to the TCS bus.

3.1.4 Data Generator and Compare

Data can come from the instruction register or the data generator. A mux select signal is generated based on the current operation. For user data patterns, the data from the instruction register is selected and written to memory. For NPSF patterns, the data generator outputs the word to be

written to memory. The comparator checks if the data read from memory matches what is expected and generates the pass/fail signal.

3.1.4.1 Data Generator

The data generator block design is shown in 3.2. The gray counter is a 5-bit counter that generates gray code and repeats after 32 iterations. The gray code is passed to the transform block which converts the code to a Eulerian Sequence. A detailed description of the Euler Sequence can be found in [12]. Essentially, a 5-bit Euler Sequence used as the Type-1 NPSF background will transition through all the possible read and write combinations that could occur. This requires 161 different sequences which can be broken into five groupings as shown in 3.1. Except for the first column, each column is generated by a one-bit right shift and rotate followed by inverting the most and least significant bits [**?**].

Table 3.1: Partial 5-bit Euler Sequence

Iteration	Gray (E[0])	E[1]	E[2]	E[3]	E[4]
X0	00000	10001	01001	00101	00011
X1	00001	00001	00001	00001	00001
X2	00011	00000	10001	01001	00101
.....	...	...	...	...	...
X29	10011	01000	10101	01011	00100
X30	10001	01001	00101	00011	00000
X31	10000	11001	01101	00111	00010

The transformation block provides the 5-bit Euler Sequence to be used as the NPSF background. To write this value to memory using the Type-1

tiling method, the sequence must be spread over the adjacent rows to create the correct tiling background as shown in 3.3.

3.1.4.2 Data Comparator

Each read march element is checked with the data comparator. The comparator accepts as inputs the TDS bus and the output of the MUT. If the MUT output matches the TDS value, a pass signal is generated. If there is any discrepancy, the fail signal is generated.

3.1.4.3 Polarity and Data Register

The polarity signal from the current march operation is used to invert the data. If the polarity signal is false (0), the data is unmodified and stored to the data register. If the polarity signal is true (1), the data is inverted, then stored in the data register.

3.1.5 External Connections

Integration with the MUT and scan-path requires a few external connections. The scan-path writes data to the instruction register for the test. The MUT receives address, data and control signals from the memory BIST.

3.1.5.1 Scan-Path Connection

The scan-path receives data serially for the memory BIST. The scan-path signals correspond to the instruction register fields. When the scan-in

data has been clocked into place, the instruction register will latch the data and begin its test.

3.1.5.2 Memory Under Test Connections

The MUT connects to the MBIST through the test buses. The connections provides the data, control signals and address to the memory and are connected to the memory input pins.

Test Address Signals The test address signals (TAS) compose the memory address currently on interest to the test. They can point to the read address for a comparison or check or the write address to store new data.

Test Control Signals The test control signals (TCS) are generated from the instruction register's operation field. The signals are formatted to work with the MUT. The opcode is translated to the memory control signals such as read/write, reset and enable.

Test Data Signals The test data signals (TDS) contain the data to be stored in memory. These signals are connected to the MUT's data input pins and the data comparator's input pins. They are driven from the data register. When an auxiliary memory is used, the data pins are driven from the outputs of the auxiliary memory.

3.2 Modifications to Programmable MBIST

These modifications were made to the design.

3.2.1 Address Generator Expansion

The address generator originally supported linear and pseudo-random address counting methods. The design in this report will add Gray code, address complement and 2^i counting methods.

3.2.2 Programmable Address Generator

The original design did not make use of the address mode field and left it to the designer to decide on whether to implement the pseudo-random or linear counting methods. The only programmability for address was the direction which was set using the up/down field. This report adds the ability to select which address generation pattern to use. The instruction register was expanded with new fields for address generation modes to allow programmability, and the instruction field is decoded within the BIST to select the address counting method.

3.2.3 Dynamic Background Pattern Generation

The auxiliary memory is used to store NPSF background patterns in the base design. To reduce the area required by this design, the auxiliary memory is replaced by a dynamic background pattern generator. The generator creates the Type-1 neighborhood pattern and translates it to an 8-bit value that can

Table 3.2: PMBIST Address Modes

Name	Op Code (binary)	Description
Linear	0000	Standard numerical sequence
Pseudo-Random	0001	Repeatable random sequence
Address Complement	0010	Progresses linearly on even steps and inverts all bits on odd steps.
Reserved	01XX	Available for additional addressing modes
Gray Coding	0011	Changes only one bit per address transition
2^i	$1[j_{2:0}]$	Generates all address pairs with a *Hamming* distance of 1. $2^i = j_{2:0}$ where i is the address bit that determines address-pair for *Hamming* distance.

be written sequentially to memory to create the background pattern.

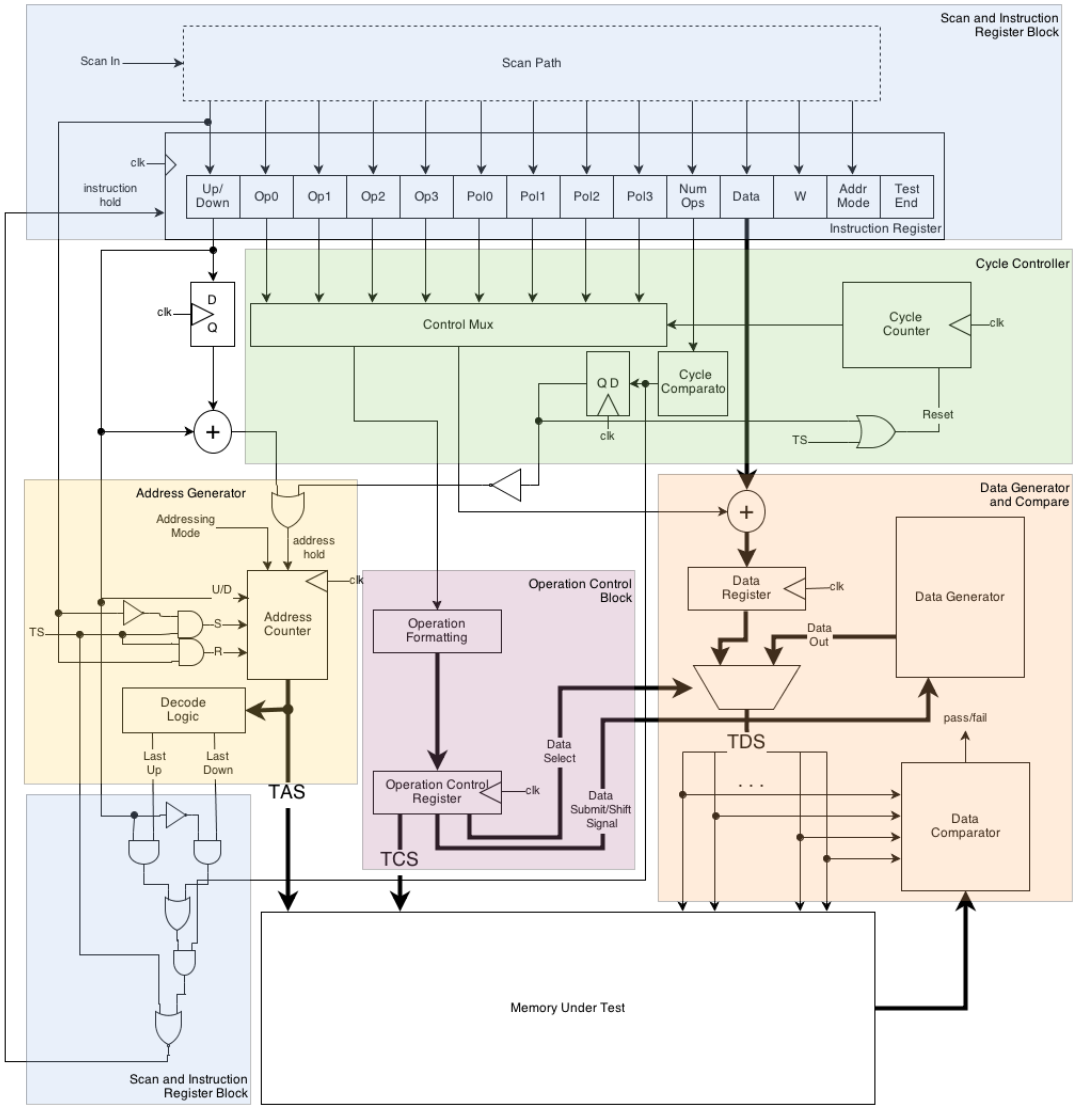


Figure 3.1: Major Block of the PMBIST Design

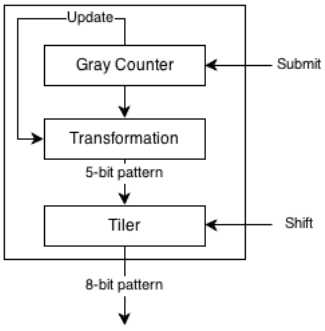


Figure 3.2: Data Generator Block Diagram

	Data Word							
0x8000	3	4	0	1	2	3	4	0
0x8001	0	1	2	3	4	0	1	2
0x8002	2	3	4	0	1	2	3	4
0x8003	4	0	1	2	3	4	0	1
0x8004	1	2	3	4	0	1	2	3

Figure 3.3: Type-1 Neighborhood Tiling Method for Design
Each 5-bit color block represents one Type-1 neighborhood.

Chapter 4

Results

The design results are presented in this section. The design was compiled using Synopsys Design Compiler with the Artisan 45nm typical libraries. The area measurements are taken from Design Compiler's built-in area report generator. The memory size information is obtained for Artisan Memory Compiler tools using the 180nm library. To show a more accurate comparison, the area from the Memory Compiler tool has been scaled to a more comparable size. This does not affect the analysis as the memory area is only used as a value to reference the original BIST and proposed BIST designs.

4.1 Area Comparisons

This report shows how a robust address and pattern generation scheme can be implemented with very little increase in area. The implementation does require some additional area over the base design. For large memory blocks, the address generator only requires 0.6% more area overhead than the base design. Similarly, the pattern generator adds 1.0% more area within the memory block. For designs using an auxiliary memory, the pattern generator reduces the overall memory area by 50%.

4.1.1 Address Generator Area

The original base design only offered binary and LFSR up/down counters for address counting methods. The proposed design improves on the base design by providing a more robust address counting method that industry uses to test memory during production. Table 4.1 below shows the area increases by 59.3% for the address counter block and 17.7% for the overall PMBIST area.

Table 4.1: Address Counter Area Comparison(um^2)

Component	Base Design	Proposed Design	Percentage Difference
address counter	1810	2884	59.3%
pmbist area	6057	7132	17.7%

It is important to note the overall area difference with respect to the full memory block. This can be interpreted as the area cost for the additional address counting methods that ease manufacturing tests and add more robustness to the PMBIST. Table 4.2 shows the percentage of the total memory block used by the original and proposed address counter designs. As the memory sizes increase, the area increase due to the address generator becomes insignificant when the benefits of the new addressing schemes are considered.

4.1.2 Pattern Generator Area

The pattern generator offers a built-in solution to replace auxiliary memories used for NPSF Type-1 tiling neighborhoods. Table 4.3 shows the estimated area for small memories that may be used for auxiliary memory to

Table 4.2: Address Counter Area within Memory Block

		Area Overhead Percentages		
Memory Size	Total Memory Area (um^2)	Base Design	Proposed Design	Difference
64x8	15780	27.7%	31.1%	3.4%
128x8	16884	26.4%	29.7%	3.3%
256x8	19333	23.9%	26.9%	3.1%
512x8	24108	20.1%	22.8%	2.7%
1024x8	34013	15.1%	17.3%	2.2%
2048x8	53276	10.2%	11.8%	1.6%
4096x8	92093	6.2%	7.2%	1.0%
8192x8	172577	3.4%	4.0%	0.6%

store patterns. This table shows the pattern generator uses 78.6-90.4% less area than the auxiliary memory. With such a large area reduction, additional circuits to generate other patterns could be added and still use less area than an auxiliary memory.

Table 4.3: Area of Pattern Generator Compared to Auxiliary Memory

Memory Size	Memory Area	Area Reduction
4x8	10289	78.6%
8x8	11393	80.6%
16x8	12497	82.4%
32x8	13601	83.8%
64x8	14706	85.0%
128x8	15809	86.0%
256x8	18258	87.9%
512x8	23033	90.4%

For designs using an auxiliary memory to store NPSF Type-1 tiling neighborhoods, the pattern generator will actually reduce PMBIST memory

overhead for memory. Table 4.4 shows the total memory block area can decrease by 5.0-35.2% when an 8x8 auxiliary memory is replaced by the pattern generator.

Table 4.4: Pattern Generator Memory Area Reduction

Memory Size	Base Design	with Aux. Mem	with PG	Area Reduction
64x8	14706	26099	16911	35.2%
128x8	15809	27202	18015	33.8%
256x8	18258	29651	20464	31.0%
512x8	23033	34426	25239	26.7%
1024x8	32939	44332	35144	20.7%
2048x8	52202	63595	54407	14.4%
4096x8	91019	102412	94224	9.0%
8192x8	171503	182896	173708	5.0%

4.1.3 Address and Pattern Generator within Memory Block

Combining the address and pattern generator will provide the most flexibility with the memory test algorithm while also minimizing the increase in the memory area to accommodate the new features. While the address generator adds to the total area, combining it with the pattern generator's area reduction actually achieves a better area than the original base design with a simple address counter and auxiliary memory. Figure 4.1 shows the area reduction achieved by combining both the address and pattern generator.

Compared to the original base design, the design implemented in this report achieves a more robust address generation scheme while using less area than the original design thanks to the use of a pattern generator in place of an

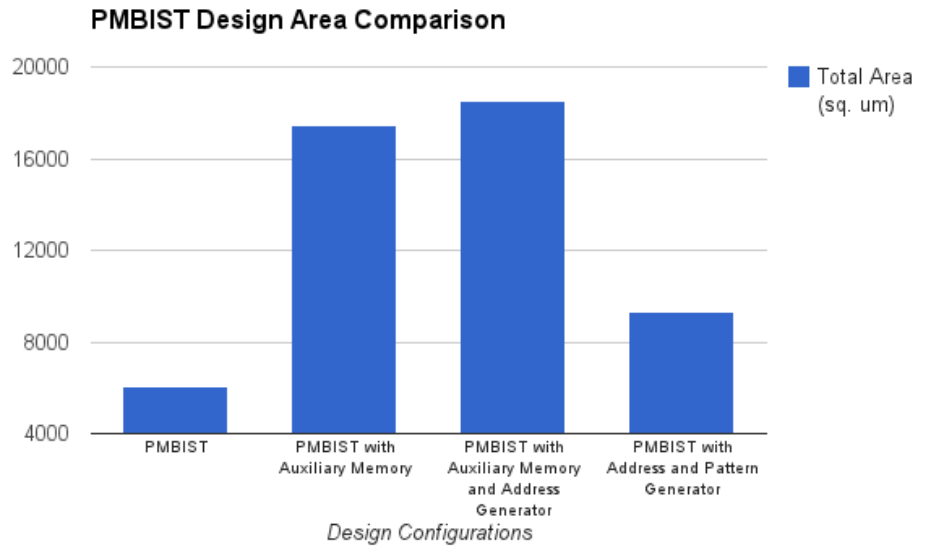


Figure 4.1: Area Comparison of Different PMBIST Configurations

auxiliary memory. As a final comparison the memory overheads for the various PMBIST designs discussed in this report are presented in Table 4.5. This table shows the percentage of the total memory IP block used by each of the PMBIST designs. The design with combined address and pattern generation only increases the memory overhead by 1.8-9.6% depending on memory size. By comparison, the designs using auxiliary memory add an additional 6.3-26.5% area overhead to the design.

Table 4.5: Area Overhead for each PMBIST Configuration

Memory Size	Base Design	Base with Aux. Mem.	AG with Aux. Mem.	AG with PG
64x8	29.2%	54.3%	55.7%	38.8%
128x8	27.7%	52.5%	54.0%	37.1%
256x8	24.9%	48.9%	50.4%	33.8%
512x8	20.8%	43.1%	44.6%	28.8%
1024x8	15.5%	34.6%	36.0%	22.1%
2048x8	10.4%	25.1%	26.2%	15.2%
4096x8	6.2%	16.9%	16.9%	9.3%
8192x8	3.4%	9.7%	9.7%	5.2%

Chapter 5

Conclusion

To meet current and future application requirements, SoCs must integrate memory. Designers have integrated to the point where over 90% of the footprint is allocated to memory. Because of the aggressive memory density and defects-per-million (DPM) level, the overall SoC yield is strongly dependent on the health of the memory.

Traditional circuit faults such as stuck-at and coupling must still be detected to maintain an acceptable yield. A variety of memory test algorithms are implemented to screen for these faults with the March Test becoming the most popular due to its mathematical backing and fast test time. But because of the size, speed and complexity of todays chips, running these tests through external memory testers is no longer a viable option and MBISTs have become the only practical solution for fault coverage.

This report shows that a programmable MBIST can achieve flexibility offered by external testers while also maintaining a low area overhead within the memory block. An address generator, which offers gray coding, 2^i, and address complement counting methods, combined with an NPSF Type-1 neighborhood background pattern generator can be integrated with only an

additional 5.2% area for memories 8KB memories - and the area overhead is further reduced as memories increase in size. This design will provide designers flexibility to adapt to new memory test models on the manufacturing line without sacrificing very much area.

5.1 Future Work

This report offered two changes to the PMBIST design proposed in [**?**], but there are many other innovations in research now that will help make PMBIST a more robust solution. Regarding the design offered in this report, there are a few other improvements that could help it to achieve better flexibility and more fault coverage.

5.1.1 3-D Memory Testing

This design proposed an NPSF pattern generator for 2-D memories where the memory layout is known. With the advent of multi-layer memories and even 3-D memories, the coupling affects from the memory layers above and below the cell will need to be considered as transistor technology reaches the sub-10nm threshold. The pattern generator could be improved upon by modifying the tile generator to output patterns above and below the base cell while still keeping the Type-1 neighborhood intact.

5.1.2 Generate Additional Patterns

The current design outputs the memory data values to generate a Type-1 NPSF tiling neighborhood in memory. Additional patterns can be added to the generator create Type-2 NPSF neighborhoods. Additionally, the two-group method could be employed for Type-1 neighborhoods instead of the tiling method to create a different way to test the memory cells.

5.1.3 Generate Additional Address Counting Methods

The original design contained a linear and pseudo-random counting method and the proposed design extended that by adding Gray coding, Address Complement, and 2^i address counting methods. There are still other counting methods such as Worst-Case Gate Delay that can be added to the address generator that would improve the coverage and offer more flexibility to designers and testers.

Bibliography

[1] A.S. Abu-Issa and S.F. Quigley. Bit-swapping lfsr for low-power bist. *Electronics Letters*, 44(6):401–402, 2008.

[2] R. Dean Adams. *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test*. Kluwer Academic Publisher, Massachuttes, USA, 2003.

[3] S. Boutobza, M. Nicolaidis, K.M. Lamara, and A. Costa. Programmable memory bist. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 10 pp.–1164, 2005.

[4] Kuo-Liang Cheng, Ming-Fu Tsai, and Cheng-Wen Wu. Neighborhood pattern-sensitive fault testing and diagnostics for random-access memories. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(11):1328–1336, 2002.

[5] Z. Conroy, G. Richmond, Xinli Gu, and B. Eklow. A practical perspective on reducing asic ntfs. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 7 pp.–349, 2005.

[6] A.J.van de Goor. *Testing Semiconductor Memories: Theory and Practice*. John Wiley & Sons Ltd., West Sussex, England, 1991.

[7] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan. Dynamic read destructive fault in embedded-srams: analysis and march test solution. In *Test Symposium, 2004. ETS 2004. Proceedings. Ninth IEEE European*, pages 140–145, 2004.

[8] J. Dreibelbis, J. Barth, H. Kalter, and R. Kho. Processor-based built-in self-test for embedded dram. *Solid-State Circuits, IEEE Journal of*, 33(11):1731–1740, 1998.

[9] A. Fradi, M. Nicolaidis, and L. Anghel. Memory bist with address programmability. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 79–85, 2011.

[10] S. Hamdioui, G. Gaydadjiev, and A. J. Van de Goor. The state-of-art and future trends in testing embedded memories. In *Memory Technology, Design and Testing, 2004. Records of the 2004 International Workshop on*, pages 54–59, 2004.

[11] S. Hamdioui, A. J. Van de Goor, J.D. Reyes, and M. Rodgers. Memory test experiment: industrial results and data. *Computers and Digital Techniques, IEE Proceedings* -, 153(1):1–8, 2006.

[12] J.P. Hayes. Testing memories for single-cell pattern-sensitive faults. *Computers, IEEE Transactions on*, C-29(3):249–254, 1980.

[13] WonGi Hong, JungDai Choi, and Hoon Chang. A programmable memory bist for embedded memory. In *SoC Design Conference, 2008. ISOCC*

'08. International, volume 02, pages II–195–II–198, 2008.

[14] Yu-Jen Huang and Jin-Fu Li. A low-cost pipelined bist scheme for homogeneous rams in multicore chips. In *Asian Test Symposium, 2008. ATS '08. 17th*, pages 357–362, 2008.

[15] J.B. Khare, A.B. Shah, A. Raman, and G. Rayas. Embedded memory field returns - trials and tribulations. In *Test Conference, 2006. ITC '06. IEEE International*, pages 1–6, 2006.

[16] M. Klaus and A. J. Van de Goor. Tests for resistive and capacitive defects in address decoders. In *Test Symposium, 2001. Proceedings. 10th Asian*, pages 31–36, 2001.

[17] H. Koike, T. Takeshima, and M. Takada. A bist scheme using microprogram rom for large capacity memories. In *Test Conference, 1990. Proceedings., International*, pages 815–822, 1990.

[18] E.J. Marinissen, B. Prince, D. Keltel-Schulz, and Y. Zorian. Challenges in embedded memory design and test. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 722–727 Vol. 2, 2005.

[19] N.-Q. MohdNoor, A. Saparon, Y. Yusof, and M. Adnan. New microcode's generation technique for programmable memory built-in self test. In *Test Symposium (ATS), 2010 19th IEEE Asian*, pages 407–412, 2010.

[20] N. Mukherjee, A. Pogiel, J. Rajski, and J. Tyszer. Bist-based fault diagnosis for read-only memories. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(7):1072–1085, 2011.

[21] R. Nair, S.M. Thatte, and J.A. Abraham. Efficient algorithms for testing semiconductor random-access memories. *Computers, IEEE Transactions on*, C-27(6):572–576, 1978.

[22] T. Powell, A. Kumar, J. Rayhawk, and N. Mukherjee. Chasing subtle embedded ram defects for nanometer technologies. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 9 pp.–850, 2005.

[23] A. Van De Goor, S. Hamdioui, and G. Gaydadjiev. Using a cisc microcontroller to test embedded memories. In *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*, pages 261–266, 2010.

[24] A. J. Van de Goor. Using march tests to test srams. *Design Test of Computers, IEEE*, 10(1):8–14, 1993.

[25] A. J. Van de Goor, S. Hamdioui, G.N. Gaydadjiev, and Z. Al-Ars. New algorithms for address decoder delay faults and bit line imbalance faults. In *Asian Test Symposium, 2009. ATS '09.*, pages 391–396, 2009.

[26] A. J. Van de Goor, H. Kukner, and S. Hamdioui. Optimizing memory bist address generator implementations. In *Design Technology of Integrated*

Systems in Nanoscale Era (DTIS), 2011 6th International Conference on, pages 1–6, 2011.