92093

IMPLEMENTING GENERIC BUILT-IN SELF TEST FOR TESTING KILO-BIT MEMORIES

A Thesis
Submitted in partial fulfillment of the requirement for the award of degree of

Master of Technology
in
VLSI Design and CAD

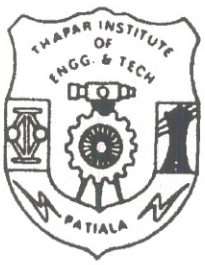
By
Mr. Anuj Gupta
Regn. No-6030402
Batch 2003-2005

Work carried out at

India Design Centre (IDC)
Freescale Semiconductors Ltd.
Noida, U.P.

Supervisors

Mr. Sanjay Gupta **Mr. Amardeep Singh**



Electronics & Communication Engineering Department
Thapar Institute Of Engineering And Technology
(Deemed University) Patiala (Punjab)-147004 (INDIA)

May, 2005

CANDIDATE DECLARATION

This is to certify that the thesis titled, "**Implementing Generic Built-in Self Test for Testing kilo-Bit Memories**" submitted by Anuj Gupta, Regn No. 6030402, in partial fulfillment of the requirement for the award of the degree of Master of Technology in VLSI Design & CAD at Thapar Institute of Engineering & Technology (Deemed University), Patiala (Pb.), is a bonafide work carried by him under our supervision and guidance.

For Amit Sharma
[AMIT SHARMA]

Mr. Sanjay Gupta
Design Lead
NCSD, India Design Center,
Freescale Semiconductors Ltd.
Noida (U.P.), India

Asingh.

Mr. Amardeep Singh
Sr. Lecturer, C.S.E.D,
Thapar Institute of Engg. & Tech.
Patiala (Pb.), India

Dr. R.S Kaler
H.O.D., E.C.E.D,
Thapar Institute of Engg. & Tech.
Patiala (Pb.), India

Dr. D.S. Bawa
Dean of Academic Affairs
Thapar Institute of Engg. & Tech.
Patiala (Pb.), India

ABSTRACT

IMPLEMENTING GENERIC BUILT-IN SELF TEST FOR TESTING KILO-BIT MEMORIES

by Anuj Gupta

The report presents the implementation of Generic Built-in Self Test (GBIST) engine for at-speed testing of semiconductor memories embedded in high performance application specific System on Chips (SoCs).

Embedded memory continues to represent a larger portion of today's SoCs. Because of this trend, and the nature of a memory's small geometries, implementing a sound memory testing strategy is one of the most significant design decisions.

Application of test patterns using off-chip testers results in a high test time due to the large sizes of embedded memories. Also the limited resources on the external tester (ports, memory, etc) have forced the system designers to introduce self testing logic within the system itself, more popularly known as Built-in Self Test (BIST). The computed test sequences to test faults in embedded memories are generated on-chip using a memory Built-In Self Test (BIST) unit.

The fine geometries typical of an embedded memory make it susceptible to subtle defects. Testing it requires a thorough set of patterns strategically chosen to expose manufacturing defects. GBIST engine provides test circuitry that applies, reads, and compares test patterns to expose these defects. It is designed to save chip area and achieve high fault coverage with minimal test time and implements the industry-standard memory test algorithms. These algorithms include the common *MARCH C* and *Checkerboard* algorithms, along with varied pattern backgrounds and many others.

In this thesis we focus on the implementation of testing algorithms for Static Random Access Memories (SRAMs) in the GBIST engine. During the course of the study it was found that the Checker-Board algorithms, implemented in the GBIST, was affected by the physical layout of the memory bits. The algorithm's read/write sequence and pattern was modified to take into account the scrambling of the memory cells in the SRAM array. The scrambling of the memory cells arises due to the muxed layout of

large memories (having a mux factor that is greater than unity), which are optimized to balance the load, the parasitics and delays on its internal wires and transistors.

The GBIST test logic can be applied to an unlimited number of memories, with varying sizes and configurations. The three key areas the GBIST efficiently addresses to ensure all embedded SRAMs and ROMs are thoroughly tested are high test quality, easy application, and versatility.

ACKNOWLEDGMENTS

I express my sincere gratitude to my supervisors Mr. Sanjay Gupta, Mr. Amit Sharma as well as all other members of our NCSG (Networks & Computing System Group) Design team at Freescale Semiconductors India ltd. and especially Mr. Amardeep Singh under whose inspiration, encouragement and guidance I had the opportunity to do the work for my thesis. They all allowed me to work on my thesis in complete freedom while strongly supporting my academic endeavors, no matter where they took me. I would like to thank them for introducing me to the problem and providing invaluable advice throughout the course of my thesis.

It was a pleasure working at India Design Center (IDC), Freescale Semiconductors ltd., Noida and T.I.E.T., Patiala (Pb.). This is mostly due to the wonderful people who have sojourned there over the past years.

This work could never have been accomplished without the inspiration, guidance and support of my innumerable friends, colleagues and my family members.

Anuj Gupta
Regn. No. (6030402)

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Table Number Page

TABLE OF CONTENTS

GLOSSARY

IC	Integrated Circuit
SoC	System-on a Chip
BIST	Built-in Self Test
BISR	Built-In Self-Repair
ECCs	Error Correcting Codes
DSM	Deep Sub-Micron. Design having a minimum feature size in the nanometer (10^{-9}m) range or below
DFT	Design For Testability
RAM	Random Access Memory. Generally used to refer memories that can read and write
ROM	Read Only Memory
SRAM	Static Random Access Memory
Flip-Flop	A bistable device used for storage of binary information
CUT	Circuit Under Test
ATPG	Automatic Test Pattern Generation
GBIST	Generic Built-in Self Test

CHAPTER 1

INTRODUCTION

In this modern era, electronic equipments and products have become part and parcel of our daily life. The key components of an electronic product are integrated circuits (ICs). In today's semiconductor world, integration technology is improving and refining dramatically, with zero failure, high reliability and longevity being the major business issues as well as customer expectation for the electronic goods. In many applications, accuracy and high reliability are essential and even life critical as in medical and aerospace applications. Dramatic improvement of integration technology in IC manufacturing is rapidly leading to exceedingly complex, multi-million transistor chips. All the functionalities of an electronic system are being integrated on a single chip in less than 1 cm^2 silicon area, popularly known as *System-on-a Chip (SoC)* which is the future of the IC technology. This growth is expected to continue in full force for the future years. However to make its production practical and cost effective, the semiconductor industry roadmaps identify test and diagnosis [1] as two of the most important hurdles to overcome.

With the continuous increase of integration densities and complexities, the problem of integrated circuit (IC) testing has become much more acute. IC testing is now no more a back-end issue, rather it has become a front-end burning issue, which needs an economic solution with reliable performance. Otherwise all the benefits of semiconductor technology would be meaningless.

Testing of integrated circuits (ICs) is of crucial importance to ensure a high level of quality in product functionality in both commercially and privately produced products. The impact of testing affects areas of manufacturing as well as those involved in design. Given this range of design involvement, how to go about best

achieving a high level of confidence in IC operation is a major concern. This desire to attain a high quality level must be tempered with the cost and time involved in this process. Also with the Semiconductor industry moving towards *Deep Sub-Micron (DSM)* design in the nanometer range there is a need to rethink about the manufacturing test process. With initial yield that can be significantly lower than those achieved at larger process technologies and the emergence of new fault types, manufacturing test will play a more important role in ensuring product quality.

During IC manufacturing, various physical defects may occur during the numerous physical, chemical and thermal processes [2]. Common defects are particles (small bits of materials that bridge two lines), incorrect spacing (wide or narrow variations in line spacing that may short a circuit), incorrect implant value (due to machine error or blockages), misalignment (misplacing one layer with respect to the previous layer), holes (exposed area that is unexpectedly etched), weak oxides and contamination (unwanted foreign material) [3]. Accuracy is very important in IC design and manufacturing because even a single error in the final layout can make a chip useless.

Traditionally, test technology was focused on the logic portion of the design. However, the *International Technology Roadmap for Semiconductors 2000* [16] has shown that it is common for today's design to consist of more than 50% embedded memory, and is expected to increase with years to come, clearly suggesting towards a need to develop a high quality memory test strategy for achieving a comprehensive SoC test.

With the rapid increase in the design complexity, *Built-In-Self Test (BIST)* has become a major design consideration in *Design-For-Testability (DFT)* methods and is becoming increasingly important in today's state of the art SoCs. Keeping the fault coverage high while maintaining an acceptable design overhead is of utmost importance. Pseudorandom Test Pattern Generation (TPG) result in low-overhead but decent (perhaps uncertain) fault coverage model, while deterministic TPG have high overhead and complete-fault coverage model.

BIST is beneficial in many ways. First, it can reduce dependency on external *Automatic Test Equipment (ATE)*. This aspect impacts the cost/time constraint because the ATE will be utilized less by the current design and can be used elsewhere or on other devices. In addition, BIST can provide At-Speed, in system testing of the *Circuit-Under-Test (CUT)* [4]. In addition, BIST can overcome pin limitations due to packaging, make efficient use of available extra chip area, and provide more detailed information about the faults present. All these benefits are plentiful motivations for BIST. A generic approach to BIST is shown in Figure 1. On a very basic level, BIST needs a stimulus (the Test Pattern Generator (TPG) in this case), a circuit to be tested, a way to analyze the results, and a way to compress those results for simplicity and handling. However, BIST designs can differ in many ways. Each method has its own set of tradeoffs and design considerations. If the BIST design is not appropriate for the IC it is testing, then it can actually be a detriment to the design.

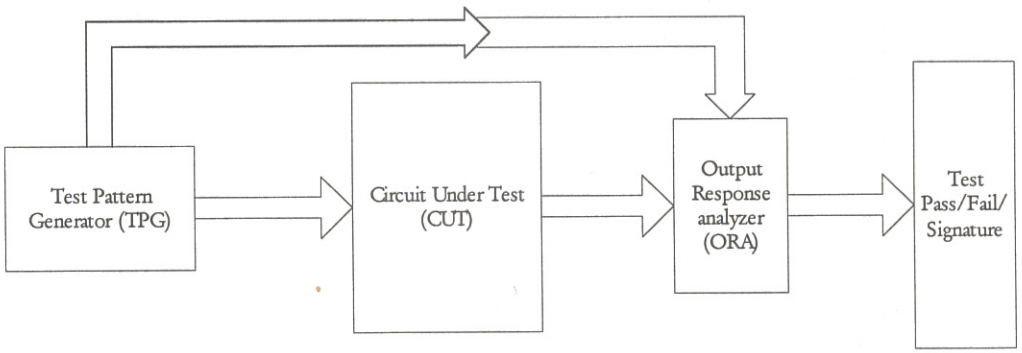


Figure 1 Basic BIST Architecture Block Diagram

CHAPTER 2

Semiconductor Memories

Memory is one of the technology drivers in the integrated circuit business because of the highly repetitive nature of memory arrays. Relatively small improvements in the design of a memory bit multiplied by the large number of bits on a chip can make a big difference in chip cost and performance. Gordon Moore, one of the founders of Intel Corporation, stated that memory size doubles approximately every two years. The generalized version of **Moore's Law** is that chip complexity doubles approximately every two years. As the dimensions of the smallest features on a chip are reduced by $1/n$, the area required for a gate is reduced by $(1/n)^2$.

2.1 Semiconductor Memories Classification

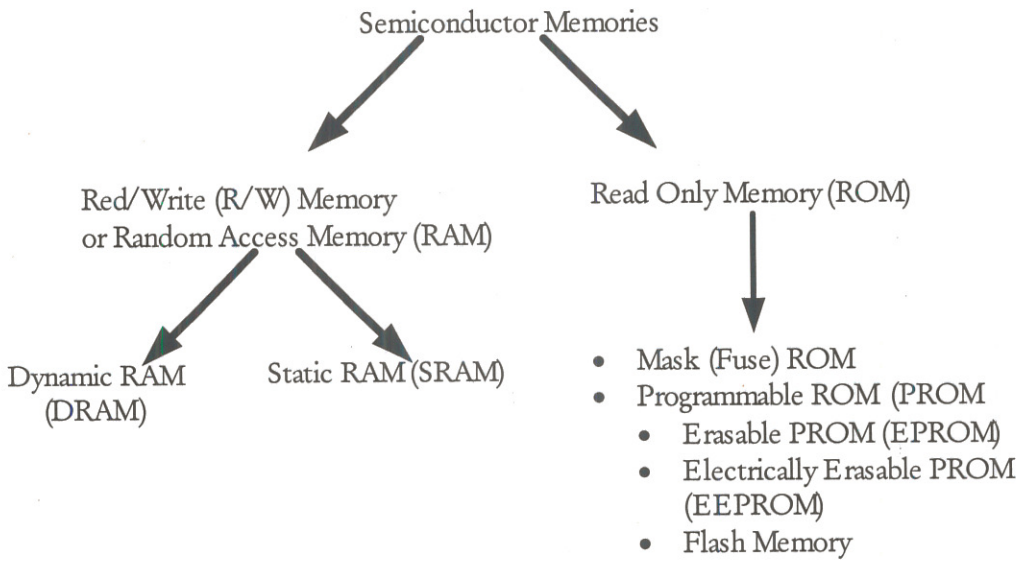


Figure 2 Semiconductor Memories

Semiconductor memory devices are generally categorized as volatile or nonvolatile random access memories (RAMs) based on data storage technique and the type of data access mechanism into the following two main groups:

2.1.1 Non-volatile Memory (NVM)

Also known as *Read-Only Memory (ROM)*, they retain information even when the power supply voltage is switched off. There are several variants of ROM, all of which differ in the way in which data is stored in them in the first place and in the way that data may be altered. Common uses for ROM include tables of unchanging data (such as character-generator tables or tables of sines), Boot ROMs, microprograms for running a processor, and permanent configuration information. With respect to the data storage mechanism NVM are divided into the following groups:

- *Mask ROM:* - The data is stored at the factory at the time of manufacture. The term *mask* comes from the use of photographic masks used to create the semiconductors used in this type of memory. The cost to create the mask is significant but if large number of units are required, the per-unit costs is very low.

- *Programmable ROM (PROM):* - The user is responsible for putting the contents into the memory. This can be done only once and usually entails burning fuses selectively in the device. Programming a PROM requires application of a high voltage to one input of the device to put it into write mode.

- *Erasable PROM (EPROM):* - Data is stored as a charge on an isolated gate capacitor ("floating gate"). Data is removed by exposing the PROM to the ultraviolet light.

- *Electrically Erasable PROM (EEPROM):* - The user is responsible for storing the data. However, individual address locations in the device can be erased electrically and rewritten. These memories are very attractive for

prototyping and for storing information (such as configuration choices) which it is desirable not to lose when power is removed. Unfortunately, the number of erasures is finite and not very large (on the order of 100,000 at the time of writing, adequate for many purposes.) As with all ROMs whose contents can be modified at all, writing the data is substantially slower than reading it. Often, though, an EEPROM can be reprogrammed in the same circuit it is used in. Programming an EEPROM requires application of a high voltage to one input of the device to put it into write mode.

- *Flash Memory:* - It is similar to EEPROM except that individual addresses cannot be erased and rewritten. Rather, entire blocks (also called sectors) of the memory are erased at one time. In fact, some flash memories can only be rewritten in their entirety. The fact that individual addresses cannot be selectively altered makes flash memories less useful for storing varying data during program execution than for program memories. As with all ROMs whose contents can be modified at all, writing the data is substantially slower than reading it is. Often, though, a flash memory can be reprogrammed in the same circuit it is used in.

2.1.2 Volatile Read/Write (R/W) memory

Also known as *Random Access Memory (RAM),* they lose their stored contents upon the removal of power. RAM refers to memories which can be written to or modified and which can be accessed randomly. Random access implies that Access time is independent of the address. The term RAM is something of a misnomer inasmuch as ROM and other types of memory also can be accessed randomly. From the point of view of the data storage mechanism RAMs are divided into two main groups:

- *Static RAM (SRAM)* where data is retained as long as the power supply is on. Data is stored on cross-coupled inverters, which represent binary 'one' or 'zero'.

- *Dynamic RAM (DRAM)* uses capacitors to store the binary information. The stored information may vanish, even with continuous power available, unless their contents are refreshed frequently. Unappealing as this trait is, the fact that DRAM can achieve much higher density than SRAM makes it attractive in large-memory systems. However, it generally is also slower than SRAM.

2.2 Random Access Memories

In RAMs, the information is stored either by setting of a bistable flip-flop circuit or through the charging of a capacitor. In either of these methods, the information stored is destroyed if the power is interrupted. Such memories are therefore referred to as volatile memories. If data is stored (i.e. written into the memory) by setting the state of the flip-flop, it will be retained as long as the power is applied and no other write signals are received. The RAMs fabricated with such cells are known as static RAMs, or SRAMs. When a capacitor is used to store data in a semiconductor RAM, the charge needs to be periodically refreshed to prevent it from being drained through leakage currents. Hence volatile memories based on this capacitor-based mechanism are known as the dynamic RAMs, or DRAMs.

SRAM densities have generally lagged behind those for DRAMs (1:4) [14], mainly because of the greater number of transistors in a static RAM cell. However, static RAMs are being widely used in systems today because of their low power dissipation and fast data access time. Early static RAMs were developed in three separate technologies: bipolar, NMOS and CMOS. By the middle of 1980s, the vast majority of SRAMs were made in CMOS technology.

SRAM speed has been usually enhanced by scaling of the MOS process since shorter gate channel length, L(eff) translates quite effectively into faster access time [14]. This scaling of the process from first generation to the second generation SRAMs has

resulted in support of higher density products. The following section will discuss in brief the SRAM cell structures.

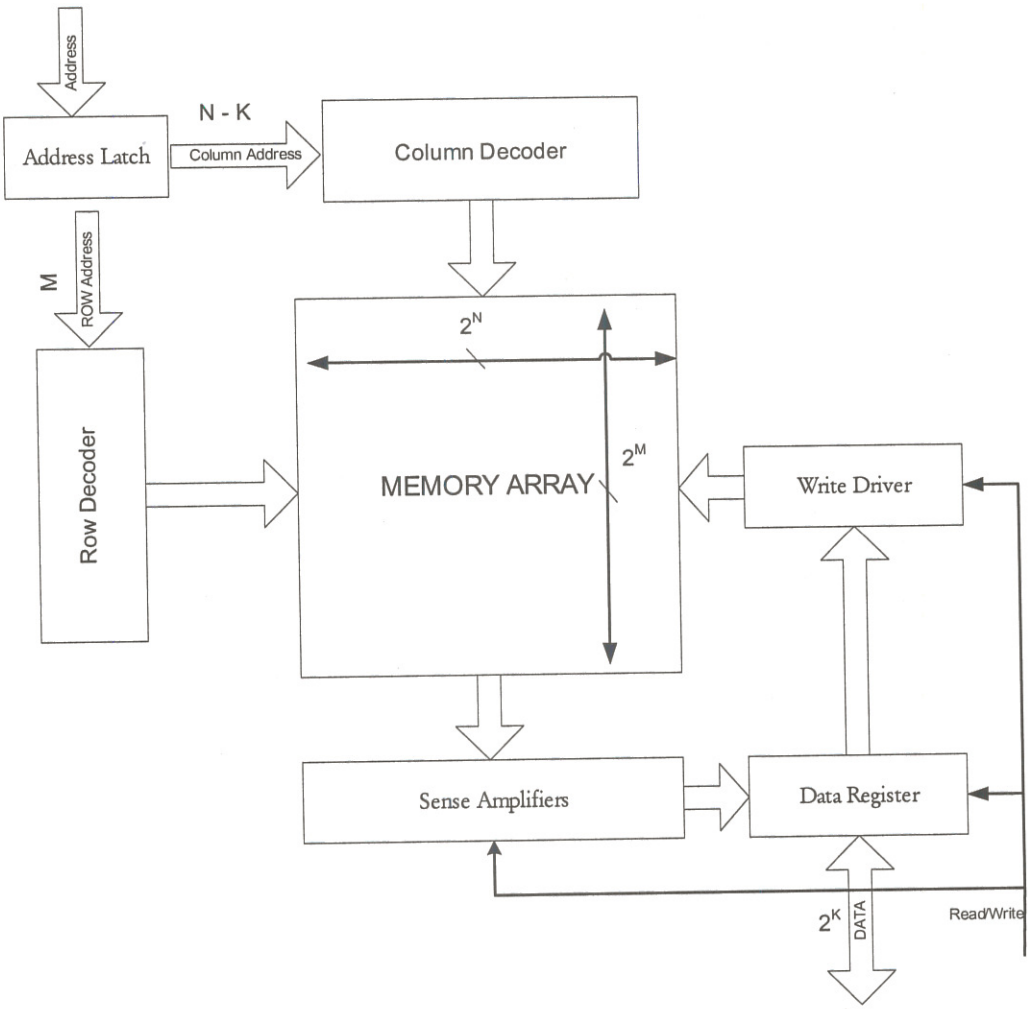


Figure 3 Functional Model of a Basic SRAM Chip

Figure 3 shows a general model for a SRAM chip, for a DRAM chip additional refresh logic would be added. The Address latch contains the address. The high-order address bits (proportional to the number of rows) are connected to the row decoder, which selects a row in the memory array. The low-order address goes to the column decoder.

Addressing of the cell is done using a two-dimensional addressing scheme consisting of row and a column address. The row decoder selects only one row of cells to be selected at a time by activating the *word line* of that particular row. The *word line* is connected to all gates of the pass transistors of all the cells in a row, and only one word line is activated at a time. The selection of a particular cell/word in a row is done under the control of column decoder which selects the set of complementary bits for that particular cell/word that is to be either read or written.

When the read/write line indicates a read operation, the contents of the cell (or word) in the memory array, selected by the row decoder and column decoder, are amplified by the sense amplifier and loaded into the Data register, and presented on the data output lines. During a write operation the data on the data input lines is loaded in the Data register and then into the memory cell (or word) in the memory array, selected by the row decoder and column decoder, through the Write driver. Usually the data input and output lines are combined to form a bi-directional data bus, thus reducing the number of input and output lines on the memory chip periphery.

The internal organization of the memory chip may vary from what it is visible externally. A 1Mb chip may *logically* (as visible to the external environment) be organized as 1Mb addresses of words which are one bit wide. *Physically* (actual layout inside the chip), the memory cells are organized as a matrix or a number of matrices [12]. For example, the physical layout could be a 1K X 1K (1K rows and 1K columns). In word oriented memories the Mux factor of the memory decides the number of columns in the physical layout of the memory. For example, a 1K X 32 (word size of 32 bits) with Mux factor of '1' would have 1K rows and 1 column (32 bit wide), but with a Mux factor of '4' the number of rows would reduce to 256 (1024/4) and the number of columns would increase to 4 (32X4) i.e. each row would be now 128 bit wide instead of 32 bits.

Such organizations of the memory help circuit layout designers in reducing the parasitics in the long word and bit lines of the memory array as well as in the better

utilization (or even minimal) of the silicon area as the memory layout can be modified in order to fit in available area.

2.2.1 Static Random Access Memories

SRAM is classified as a volatile memory because it depends upon the application of continuous power to maintain the stored data. If the power is interrupted, the memory contents are destroyed unless backup battery storage system is maintained. A SRAM is a matrix of static, volatile memory cells, and address decoding functions integrated on-chip to allow access to each cell for read/write functions. The semiconductor memory cell use active element feedback in the form of cross coupled inverters to store a bit of information as a logic "one" or a "zero" state. The memory cells are arranged in parallel so that all the data can be received or retrieved simultaneously. An address multiplexing scheme is used to reduce the number of input and output pins. As SRAMs have evolved, they have undergone a dramatic increase in there density. Most of this has been due to scaling down to the smaller geometries. Also, scaling of feature size reduces the chip area, allowing higher density SRAMs to be made more cost effective.

2.2.2 SRAM Cell Structures

The basic SRAM cell made up of cross-coupled inverters has several variations, a few of them are shown in Figure 5. The early NMOS static RAM cell consisted of six transistor design, four consisted of enhancement mode and two depletion mode pull-up transistors. A significant improvement in cost and power was achieved by substituting ion-implanted polysilicon load resistors for the two pull-up transistors [14]. These were called R-load NMOS, and a successor to these is called the "mixed-MOS" or "resistor load CMOS". These SRAMs consisted of an NMOS transistor matrix with high ohmic resistor load and CMOS peripheral circuits which allowed the benefit of lower standby power consumption while retaining the smaller chip are of NMOS SRAMs. The resistors are made from polysilicon with high resistivity (values

in the range of 100 GΩ). The resistor in the load device of the SRAM cell where replaced by PMOS enhancement mode transistors to further reduce the area and power requirements of the memory cell. In the low voltage standby mode, the standby current for the mixed MOS part is typically in the micro-amp (μA) range while for the CMOS in the nano-amp (nA) range. This low power dissipation in the standby mode opened the potential for high-density battery back-up systems.

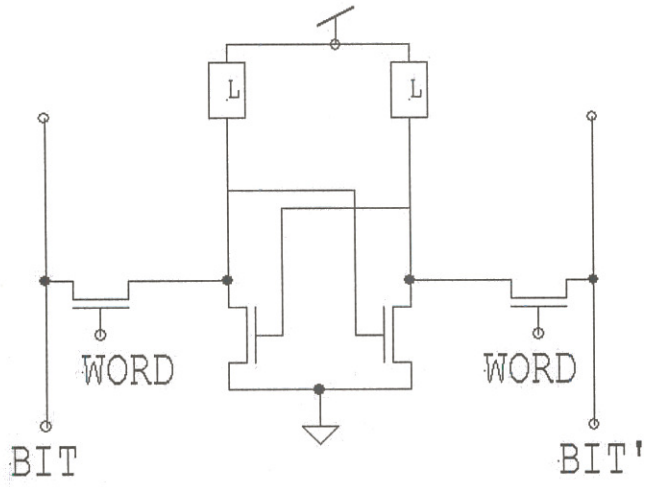


Figure 4 Basic SRAM Memory cell architecture

Figure 4 shows an NMOS SRAM cell with load devices which maybe an enhancement or depletion mode transistors as in the NMOS cell, PMOS transistors in CMOS cell or a load resistor in a mixed MOS or an R-load cell. The access and storage transistors are enhancement mode NMOS. The purpose of the load device is to offset the leakage current at the drain terminal of the select and storage transistor. In full CMOS cell load PMOS load transistors result in essentially zero current flow except during the case of switching of the value. However depletion load and resistive load cell have low value of leakage current flowing through them, thus have some stand by power dissipation.

There can be various structures of the basic SRAM cell depending on the application and the area of use. Figure 5 shows some of the configurations of the basic SRAM. Figure 5(a) shows the basic SRAM memory cell array with four enhancement mode NMOS transistors and two depletion mode load devices. Figure 5(b) shows the use of polysilicon resistive load devices instead of the deletion load transistors. Figure 5 (c) shows the basic six transistor full CMOS SRAM cell consisting of two cells and two load devices in a cross coupled inverter configuration, with two select transistors used for selecting the particular memory cell and Figure 5(d) shows an eight transistor dual port SRAM cell. Useful in cache architecture, particularly embedded memory in microprocessor chip, the memory can be accesses simultaneously from both the ports.

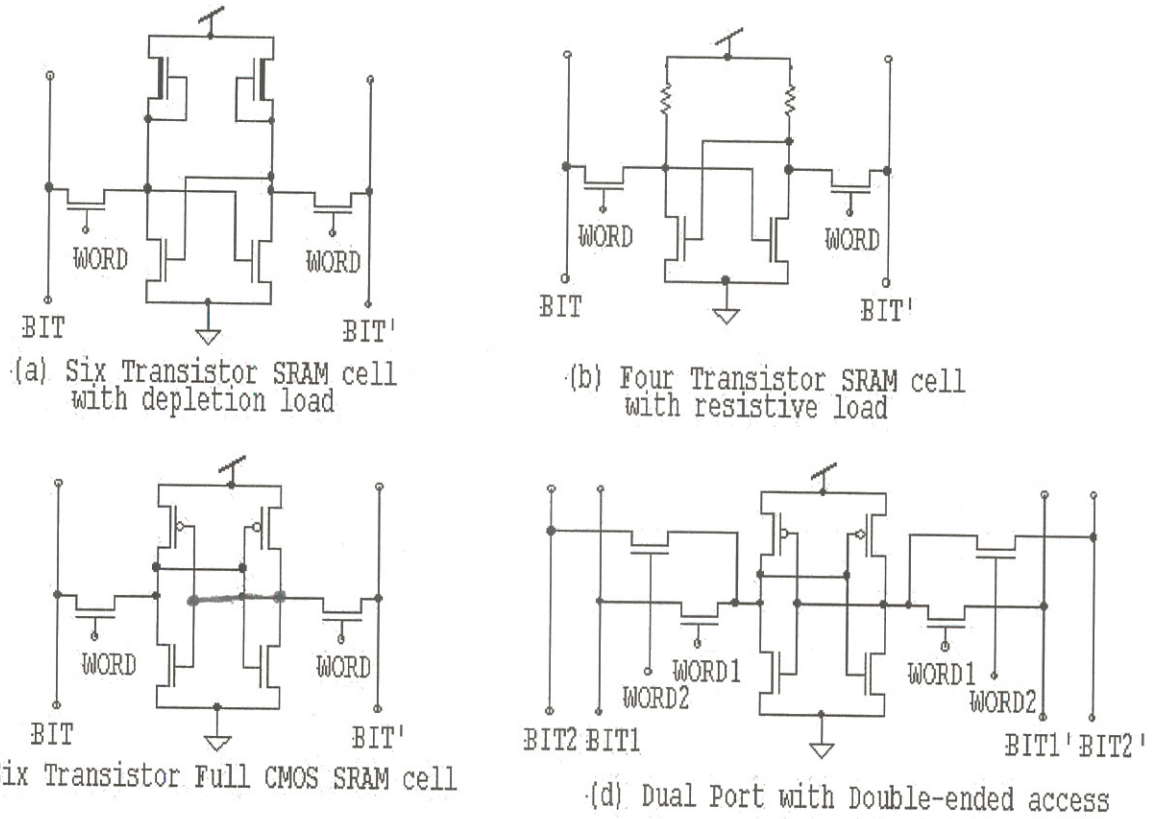


Figure 5 Various Configurations of SRAM cells [12],[14]

2.2.3 Basic CMOS SRAM cell operation

A basic CMOS SRAM cell schematic is shown in Figure 6. The bit information is stored in the form of voltage levels in the cross-coupled inverters. The circuit has two stable states "logic 1" and "logic 0". When in logic state 1, the node N5 is high and N6 is low, therefore switching the transistors T1 and T4 off and T2 and T3 on. The logic 0 state would do exactly the opposite. During a read/write task the row address is decoded by the row decoder which selects the appropriate word line of the address row. Upon selection the T5 and T6 transistors of the selected cell (or entire row for byte oriented memories) are switched on. The column decoder decodes the column address and connects the bit line (BIT) and inverse bit line (BIT') of all the cells in the selected column.

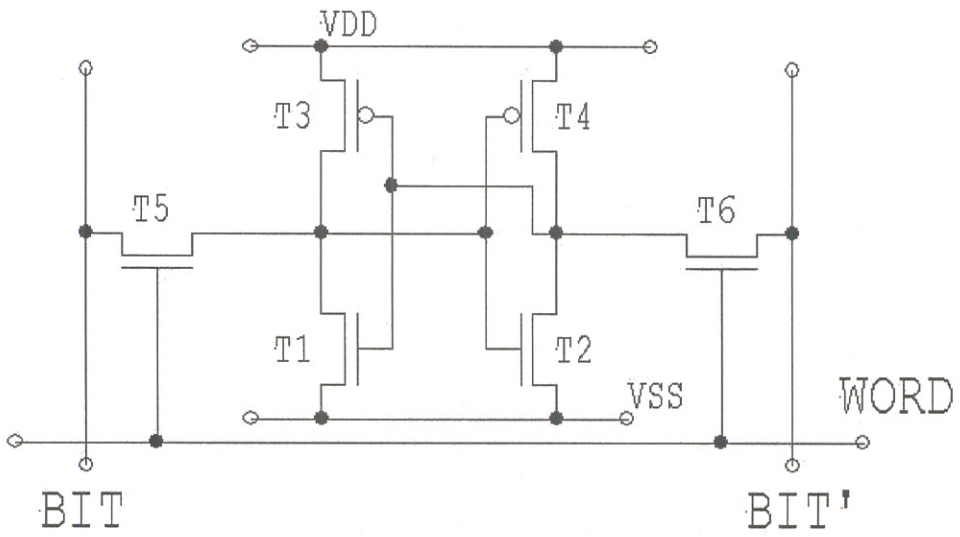


Figure 6 Schematic diagram of six-transistor CMOS SRAM cell

A Read operation starts with both the BIT and BIT' lines high and selecting the desired word line. The data in the cell will pull one of the bit lines low (through T1 if cell is in logic state "1" or T2 if the cell is in logic state "0"). This differential signal on

the BIT and BIT' lines is detected by the sense amplifier and after amplification is read out through the output buffer. The READ task is nondestructive and the contents of the cell remain intact even after several readouts from the logic cell.

During a WRITE task data (DATA) is placed on the bit line while the inverted data (DATA') is placed in the BIT' line. When the desired word is activated by selecting the word line for that row, the selected cells are forced into the state presented by the bit lines. Hence the new data is stored in the cross coupled inverters. To store a logic 1 in the cell BIT line is set to logic 1 and the BIT' line is set to logic 0 followed by activating the word line, causing the cell to change state to the desired value.

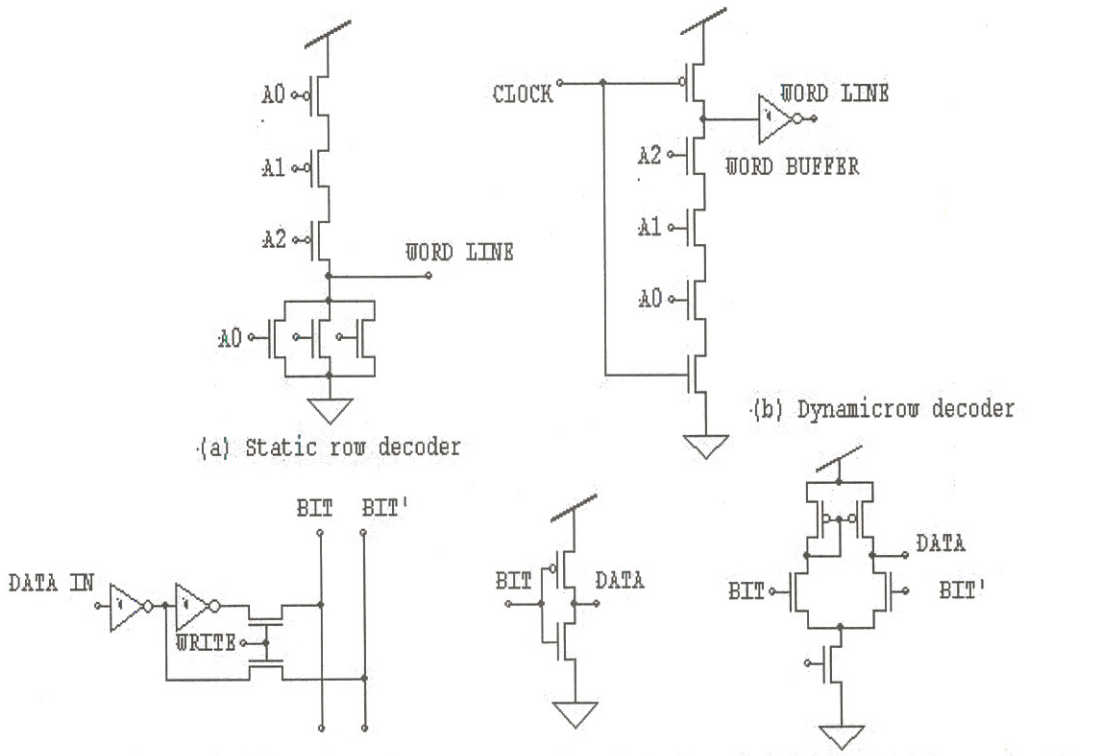


Figure 7 Various SRAM circuit elements

SRAM memory cell array's peripheral circuits consist of an address decoder (row and column) and the read/write sense amplifier. A typical write circuitry consists of

inverters on the input buffer and a pass transistor with a write control input signal to control the BIT and BIT' lines. Read circuitry generally involved the use of single indeed differential sense amplifiers to read the low-level signals from the cells.

2.3 Address and Data Scrambling in Embedded Memories

Scrambling means that the *logical* structure, as seen by the user from the outside of the chip, differs from the *physical* or *topological* internal structure of the chip. The consequence is that logically adjacent addresses may not be physically adjacent (this is called *address scrambling*) and that logically adjacent data bits are not physically adjacent (this is called *data scrambling*) [7].

2.3.1 Data Scrambling

For a memory of a given size, the *memory cell array* (*MCA*) area has to be laid out such that it approaches a square in order to balance the lengths (and therefore the capacitances and delays) of the bit lines and the word lines. However, this requires the number of words and the number of bits per word (referred to as B) to be about the same, which will not always be the case. For example, consider the 64X4 logical organization (64 rows each containing 4 bits), the topology of such a memory would not be acceptable, because the bit lines would be too long. A much better topology would be 16X16; i.e., 16 rows, each containing 16 bits. The translation of the logical 64X4 organization to the topological 16X16 organization (indicating 16 rows with 16 bits each) is called *folding*. Several folding schemes exist, depending on the layout of the bits in a row.

In the *adjacent* folding scheme, the B bits of the logical word are still adjacent topologically in the row; i.e., the row is filled with a set of W words, W = (Number of bits in a row)/B. For example in a 16X16 memory it would means that each row

contains W = 4 logical B = 4 bit words, whereby the bits of each word are physically adjacent; Figure 8 (a).

In the *distributed* folding scheme the B bits of the logical word are distributed over the entire row in such a way that the W bits numbered '0' of the W logical words in a row are physically adjacent, etc. The result is that two logically adjacent bits are separated by W-1 bits of the other words in that row, Figure 8(b).

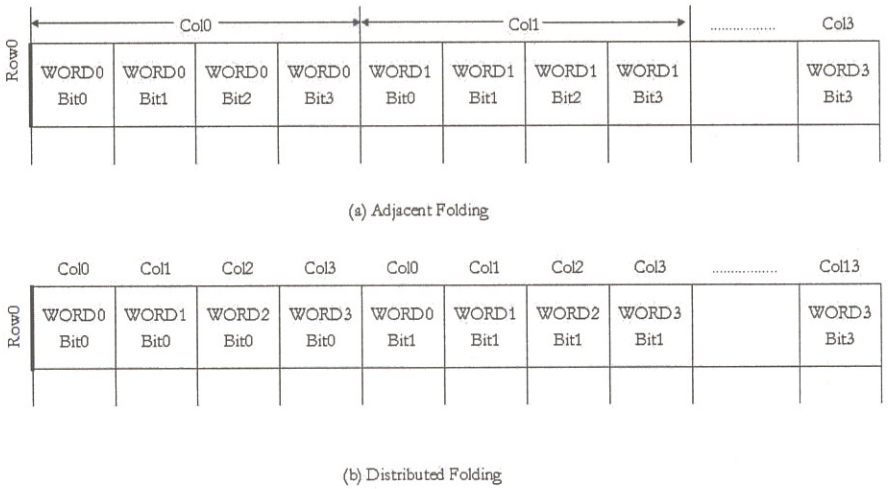


Figure 8 Distributed and Adjacent Folding

2.3.2 Addressing Scrambling

The RAM array can be partitioned into Blocks, Rows and Columns. The number of columns in the memory decides its Mux factor. For a Kilobit memory the address bus is made with the Block bits in the MSB position, Row bits after them, and then the Column bits in the LSB position, as explained in the examples below:

Example 1: A 512X32 RAM arranged into a single block with 64 rows and a column Mux factor of 8 (64 Rows and 8 columns of 32 bit each) would have address bus 9 bits wide

Address Bus: a8 a7 a6 a5 a4 a3 - a2 a1 a0
Row Addr Col Addr

Example 2: A 512X32 RAM arranged into 2 block with 16 rows and a column Mux factor of 16 (2 blocks each having 16 Rows and 16 columns of 32 bit each) would have address bus 9 bits wide

Address Bus: a8 a7 a6 a5 a4 - a3 a2 a1 a0
Blk Addr Row Addr Col Addr

By default if the address, generated by the system (or a state machine as in a BIST engine), is directly applied to the RAMs address port, then we will be accessing the RAMs *Column fast mode*, i.e. all the words of the Columns in a Row will be accessed before moving onto the next Row Figure 9 (a). To access the RAMs in *Row fast mode*, i.e. accessing all the words of the Rows in a Column before switching to the next column in a block Figure 9 (b), the address bus is shifted to the *left*, the number of shifts is the same as the number of bits reserved for the column addressing. This number can also be obtained from the memory Mux factor (m). 'm' is always in power of 2 (2^n), then the normal address must be shifted by n bit to the left to implement Row fast addressing, 'n' is also equal to the number bits reserved for the column addressing. While shifting the address bit to obtain Row fast addressing the Block bits (if any) must not be shifted and they must remain at the MSB position, as explained below

Example1: Address Bus: a5 a4 a3 a2 a1 a0 - a8 a7 a6
Row Addr Col Addr

Example 2: Address Bus: a8 a3 a2 a1 a0 - a7 a6 a5 a4
Blk Addr Row Addr Col Addr

In Column fast addressing mode the address increases with a step of unity (i.e. 1, 2, 3, 4 ....), while in a Row fast addressing mode the incremental step is greater than unity and is equal to the number of columns in a row (i.e. 0, 4, 8... for a RAM with Mux

factor of 4). It must however be noted that the number Columns in a Row are always in power of 2 (i.e. 2, 4, 8 ...).

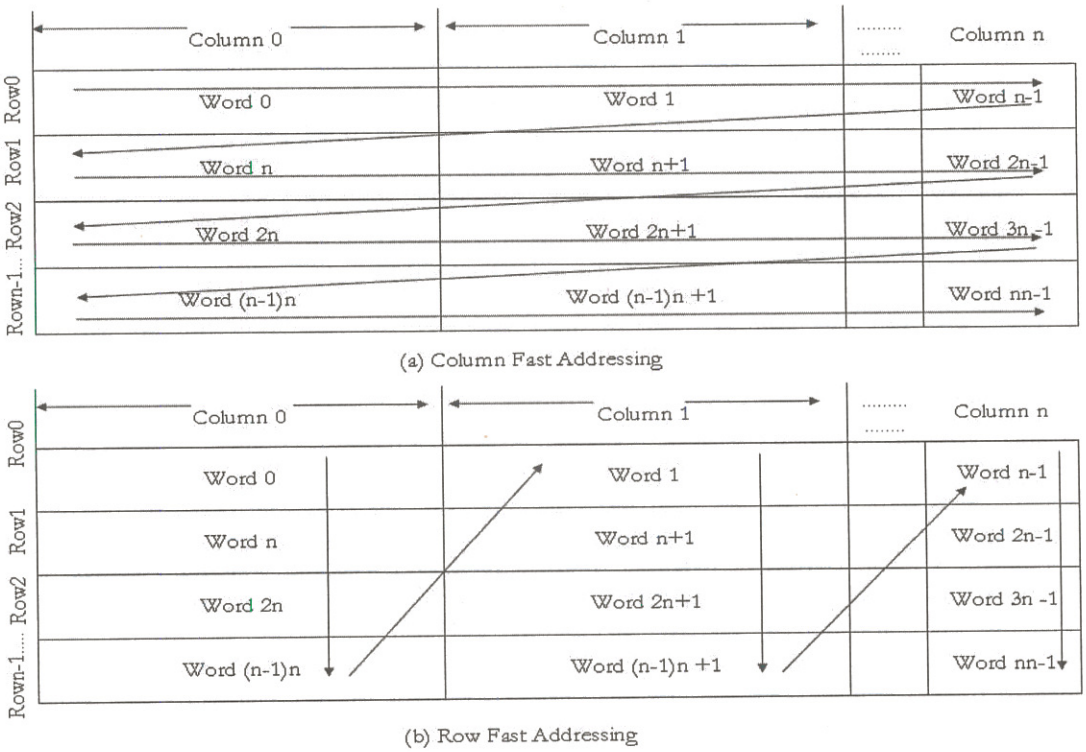


Figure 9 Column Fast vs Row Fast Addressing

CHAPTER 3

Testing Semiconductor Memories

During the fabrication of a chip, some errors may occur which may induce the faults in the circuit devices. This results in the inaccurate functioning of the chip. Hence the chips need to be tested for the presence of any faults else it may result in the inaccurate functioning of the chip. Since the number of circuits on the chip are increasing exponentially, testing of such multi million transistor chip causes a serious problem, and this introduces the concept of *Design For Testability (DFT)*. Design for testability (DFT) refers to including test considerations into the design specifications. DFT includes using design rules that forbid the use of certain hard-to-test circuit forms and/or require using inherently testable forms. It attempts to ensure that internal nodes are sufficiently controllable and observable. Specific DFT techniques include providing parallel test modes to test multiple arrays simultaneously, *Built-In Self-Test (BIST)*, *Built-In Self-Repair (BISR)*, and *Error Correcting Codes (ECCs)*. One important consideration with ECC is that it must be possible to disable the ECC during testing so that bonafide errors are not missed due to the correction.

In recent years, embedded memories are the fastest growing segment of SoCs. They therefore have major impact on the overall Defects per Million (DPM). According to 2001 International Technology Roadmap for Semiconductors (ITRS 2001), today's SoCs are moving from logic dominant chips to memory dominant chips, since future application will require lot of memory. The memory share on the chip is expected to be about 94% in 2014 [22].

Embedded memories are essential components in SoC design. These have increased in size dramatically during the past few years. SoCs generally contain large number of memories, like *SRAM (Static Random Access Memory)*, *DRAM (Dynamic Random Access*

Memory), ROM (Read Only Memory), EEPROM (Electrically Erasable Programmable Read Only Memory), etc, some of which may be even megabits in sizes. These embedded memories can be used to implement register files, FIFO (First in First Out), data cache, instruction cache, transmit/receive buffers, storage for audio/video, etc.

Testing of embedded memories is one of the major problems in today's embedded core-based SoCs as well as in complex microprocessors. In a significant number of cases, these memories are tested using *Built-in Self Test* methods. However, a number of methods are used in SoC design to test these memories.

3.1 Fault Modeling for Embedded Memories

It is important to make the distinction between a physical defect in a RAM and a memory failure. A physical defect is anything within the physical structure that deviates from what was intended, such as the presence of unwanted material, the absence of desired material, and imperfections in the lattice of the substrate. A physical defect may or may not lead to a failure, a situation in which the device behaves in such a way that violates its specifications. Physical defects that are not serious enough to immediately cause failure are known as latent defects. Latent defects may worsen with time and eventually reach the point where they do cause failure.

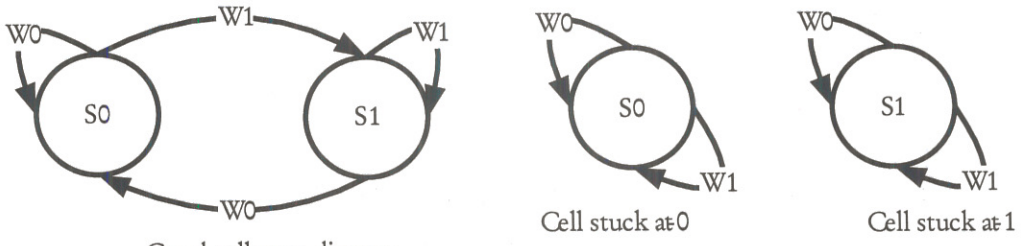
Incorrect behavior is described at some convenient level of abstraction as a fault. Faults with similar behaviors are grouped into fault types, and a set of faults that supposedly describes all types of faulty behaviors is known as a fault model.

Memory fault models are significantly different than the fault models used for digital logic. Line stuck-at faults, bridging, opens, and transistor stuck-on/off fault models work fairly well for digital logic [18], however, these fault models are insufficient for determining the functional correctness of memories. In addition to line stuck-at,

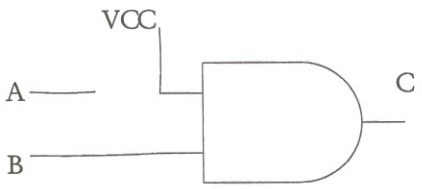
bridging and open faults, memories also include transition, cell coupling and bit pattern faults.

Fault model that is used to target embedded memories consists of the following fault classes [12], [14], [15],[18],[21]

a) *Line stuck-at faults*: Includes single and/or multiple lines (input, output, address, or bit line) in a memory cell *stuck-at* 0 or at 1. The behavior of *stuck-at* is shown in state diagram Figure 10 (a) below. Here the cell appears to be tied to power or *(stuck-at-1)* or ground *(stuck-at-0)*. To detect a *stuck-at* fault a signal value opposite to that of *stuck-at* fault must placed at the fault location. To detect a *stuck-at-1* a "0" is placed at the fault location and to detect a *stuck-at-0* fault a "1" is placed at the fault location. Shown in the Figure 10 (b) is an AND gate with one of its input *stuck-at-1*.



(a) Stuck-at fault state Diagram



(b) AND gate 'A' input Stuck at-1

Figure 10 Stuck-at fault

b) *Cell stuck-at faults*: Refers to a memory cell that maybe *stuck-at* 0 or at 1. A memory cell is said to be *stuck-at* if the logic value of the cell cannot be changed by any action on the cell or by influences from other cells. Its behavior is similar to the line stuck-at fault but here we consider the memory cell as a whole that is stuck-at 0 or 1. A test that has to detect and locate all stuck-at faults, must satisfy the following requirement:

For each cell, a 0 and a 1 must be read.

c) *State transition fault:* A special case of Stuck-at fault is the transition fault. It refers to transition in a RAM memory cell from 1 → 0 and 0 → 1. A memory cell with a *transition* fault fails to undergo at least one of the transitions 0 → 1 (up transition fault) or 1 → 0 (down transition fault).

Good cell state diagram

Cell with 0 → 1 Transition Fault

Figure 11 Transition Fault state Diagram

A test that has to detect and locate a transition fault must satisfy the following requirements:

Each cell must undergo a 0 → 1 transition, and a 1 → 0 transition, and be read after each transition before undergoing any further transition.

d) *Cell coupling faults*: these faults mean that a specified memory location is affected (either data or transition at a location) because of other location (by

data or transition at other locations), i.e. a write operation that causes a 0 → 1 or 1 → 0 transition in one cell changes the content of the other cell. Cell coupling can be of the following type:

- *Inversion* coupling fault (*Cfin*) (contents of cell inverts): A 0 → 1 or 1 → 0 transition in one cell inverts the content of the second cell. If a test has to detect all CFins it should satisfy the following requirements:

For all cells that are coupled, each cell should be read after a series of possible CFins may have occurred (by writing into the cells with the condition that the number of transitions in the couple d cells is odd i.e. the CFins do not mask each other)

- *Idempotent* type (*CFids*) (contents of cell change only if cell has a specified data): a 0 → 1 or 1 → 0 transition in one cell forces he content of the second cell to a certain value, 0 or 1. If a test has to detect all CFids it should satisfy the following requirements

For all cells that are coupled, each cell should be read after a possible CFids may have occurred (by writing into the coupling cells in such a wa y that the sensitized CFids do not mask each other)

- *State coupling* (contents of cell changes only by specified data at other locations). A memory cell, say cell *i*, is said to be *state coupled* to another memory cell, say *j*, if cell *i* is fixed at a certain value x($x \in \{0, 1\}$) only if cell j is in one defined state *y* ($y \in \{0, 1\}$).

e) *Neighborhood Pattern-Sensitive faults (NPSF):* This is a special case of state coupling faults, which refers to that in the presence of some specific data in one part of memory, data in some other part of the memory may be affected. Pattern-sensitive faults may be dynamic (due to change in data) or static (due to fixed data). A test that has to detect and locate NPSF must satisfy the following requirements:

Each cell must be written and read in state 0 and in state 1, for all possible changes in the neighborhood patterns.

f) *Stuck-Open faults:* A memory cell is said to be *stuck-open* if it is not possible to *access* the cell by any action on the cell. The stuck-open fault is popular for use in CMOS circuits. When a defect occurs that breaks a line internal to a CMOS gate, it can result in a "memory effect." For example, in this circuit, if the output is driven high with a 00 input, and then a 10 input is applied, then the output will stay high (its previous value). However, if the output is driven low with a 01 input, and then the 10 input is applied, then the output will stay low (its previous value) which appears OK.

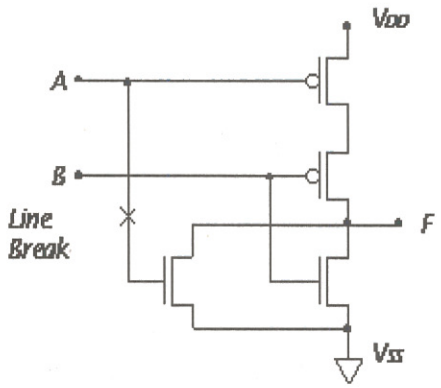


Figure 12 A' Input to NMOS is broken (stuck-open) resulting a "memory effect"

g) *Bridging faults*: sometimes called a short, it consists of a connection (called bridge) between two (or more) cells or lines. Bridging may occur among input lines, output lines, address lines, and bit lines. These include 'AND' bridging faults and 'OR' bridging faults.

h) *Addressing faults*: A row or column decoder may access the addressed cell or a non-addressed cell, or it may access multiple cells. There maybe a *multiple access fault* from one memory cell to a memory cell at another address

i) *Data retention fault:* This refers to a fault that occurs because the memory cell could not hold data for a specified period of time. There is said to be a *data retention fault* in a cell if the cell fails to retain its logical value after some units of time. Data retention faults are very important for RAMs and sometimes for programmable ROMs and Flash memories.

j) *Bit pattern faults*: In programmable ROMs (PROM/EEPROM) and Flash Memories, programming faults may cause an error, For example, a fusible-link based ROM may have an un-blown (or partially blown) fuse at an address location as well as blown fuse at an unintended location. This results in a '0' instead of a '1' and vice versa.

The important point to get is that accurately modeling all of the possible faults and devising tests that can detect all of them is not a simple, straightforward procedure. For instance, a common type of fault is a stuck-at fault, where a node is assumed to be stuck-at either a 0 or a 1 value. Detecting whether any of the bits in a memory array are stuck at 0 or 1 is fairly straightforward. Write all 0's, then read all of the values back and check that they are all 0; next write all 1's, and read all of the values back and check that they are all 1. But this would not detect a fault where the contents of one cell affected another cell nearby (possibly through a short circuit or by capacitive coupling). Detecting such pattern sensitive faults can be much more difficult. Some other examples of faults are access time failures (caused by signal transitions in the address decoders that are too slow or too fast), transition faults (where a cell can be written from 0 to 1 but not from 1 to 0, or the other way around), and data retention faults (when a cell loses its value after some amount of time less than the amount required by the RAM refresh).

CHAPTER 4

GBIST for Test and Diagnosis of Embedded Memories

Embedded memories are the densest components within a system-on-chip (SoC), accounting for up to 90% of its real estate [9]. Memories also are the most sensitive to process defects, making it essential to thoroughly test them in the SoCs. Hence complexity of today's IC design and component density demands that embedded memory testing be taken further than traditional pass/fail testing. As the geometries of ICs become increasingly concentrated, new techniques such as Built-in-Self Test (BIST), diagnostic testing and Built-in Self-Repair (BISR) must be implemented into the process.

Characteristics of today's SOC designs include the following [10]:

- Typically more than 30 embedded memories on a chip.
- Memories scattered around the device rather than concentrated in one location.
- Different types and sizes of memories.
- Memories doubly embedded inside embedded cores.
- Test access to these memories from only a few chip I/O pins.

Built-in Self Test (BIST) is a *Design for Test (DFT)* methodology of choice for testing embedded memories at-speed within a SoCs. It offers a simple and low-cost means to test for failures of embedded memories without significantly impacting device performance.

While it has been used primarily for production pass/fail testing, BIST can be extended to provide the diagnostic data required for process monitoring and repair. Although the

area overhead required by the BIST circuitry is increased, designing the diagnostic circuitry into the BIST provides many advantages in terms of time for both setup and test.

4.1 Generic BIST (GBIST)

Generic Built-in Self Test (GBIST) is a BIST engine that is used to test memories in the kilobit range. It is a plug and play solution that is easy to generate and integrate into any SoC. It can test any number of RAMs and ROMs simultaneously at there functional frequency i.e. at-speed testing. The time required to test memories simultaneously is proportional to the size of the largest memory connected to the GBIST engine.

In the basic BIST architecture, each memory is tested by a BIST block (memory interface) that supplies a series of patterns to the memory, usually *MARCH* patterns, and then compares the outputs against a set of expected responses. Because the patterns are highly regular, the outputs from the memories can be compared directly to the reference data internally in the BIST itself or externally on the tester and appropriate flags/signals can be set to indicate the pass fail result. This ensures that an incorrect response from the memory will be immediately flagged as a test failure. A pass/fail signal is provided per memory under test in addition to the global pass/fail signal, which makes the diagnosis of faulty behavior easy.

Due to the very regular structure of a RAM, most memory test algorithms use repetitive steps and do not require complex logic to implement in hardware. They could be implemented based on either random logic or micro-programmed control. An advantage of the micro-programmed based approach is that it is flexible and can be used to add additional tests and test algorithms to achieve better fault coverage with minimal area overhead. The overhead for implementing BIST on very large RAMs is estimated to be between 0.1 - 1%.

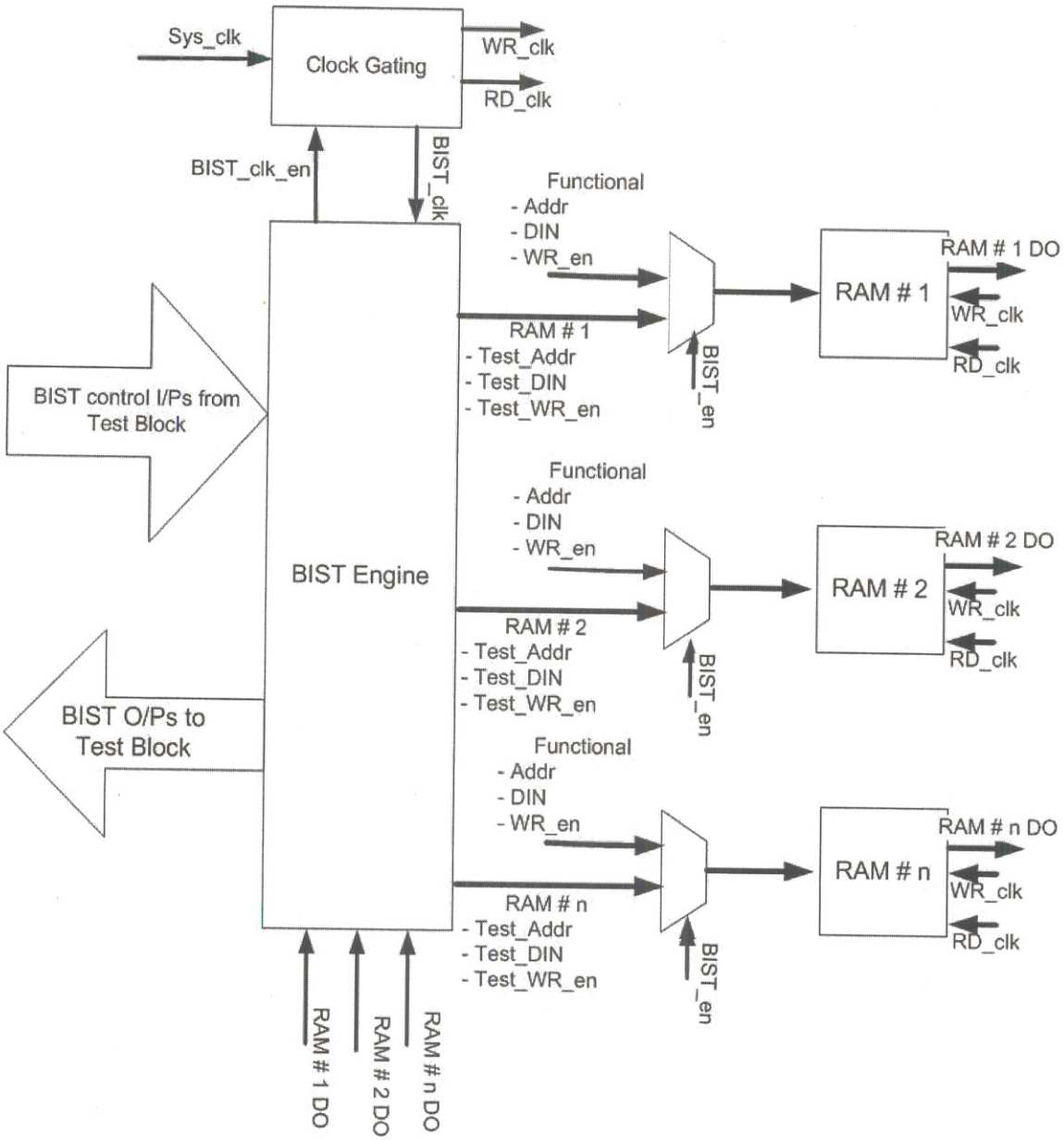


Figure 13 Generic BIST (GBIST) Engine Block Diagram

Even if an algorithm is applied using BIST, the test could still take a long time on a very large RAM. Thus in order to reduce the test time for testing memories during production

the GBIST engine is design to test all the RAMs in parallel. Moreover, all the GBIST engines in the system can be triggered at the same time, thus significantly reducing the test time for memories. There are other advantages of implementing the test in BIST, even if it does take a while. Whatever is tested using GBIST can proceed in parallel with other checks. One of the many advantages of using the GBIST engine is that the apart from the normal operation of memory testing the same engine can be utilized to obtain the failure data, which is very useful during failure analysis. A complete memory map can be obtained without adding much area overhead.

4.2 GBIST Algorithms

The reliability of the embedded memory depends on the fault coverage guaranteed by the testing algorithm implemented used during the self-testing of the memory. A number of test algorithms have been proposed, like *MARCH C, Checkered Board, and Diagonal* each covering a different set of fault type. By default MARCH C 13N algorithm is implemented along with Checker-Board algorithm. But if here one or more memory arrays that implemented late write buffers then MARCH C 24N algorithm is implemented instead of MARCH C 13N.

4.2.1 MARCH C Algorithm

A *MARCH* element is a finite sequence of read or/and write operations applied to every cell in the memory array, either in increasing or decreasing address order as shown in Figure 14.

The MARCH algorithm was originally designed for testing memory bit oriented memories, i.e. every word in the memory is one bit wide. However read or write for a word oriented RAMs involves reading or writing to an entire word of data. So the test algorithm needs to be redefined for the word oriented memories [17]:

W0 : Write pattern (<Data Background>)

R0 : Read pattern (<Data Background>)

W1 : Write inverted pattern (<Inverted Data Background>)

R1 : Read inverted pattern (<Inverted Data Background>)

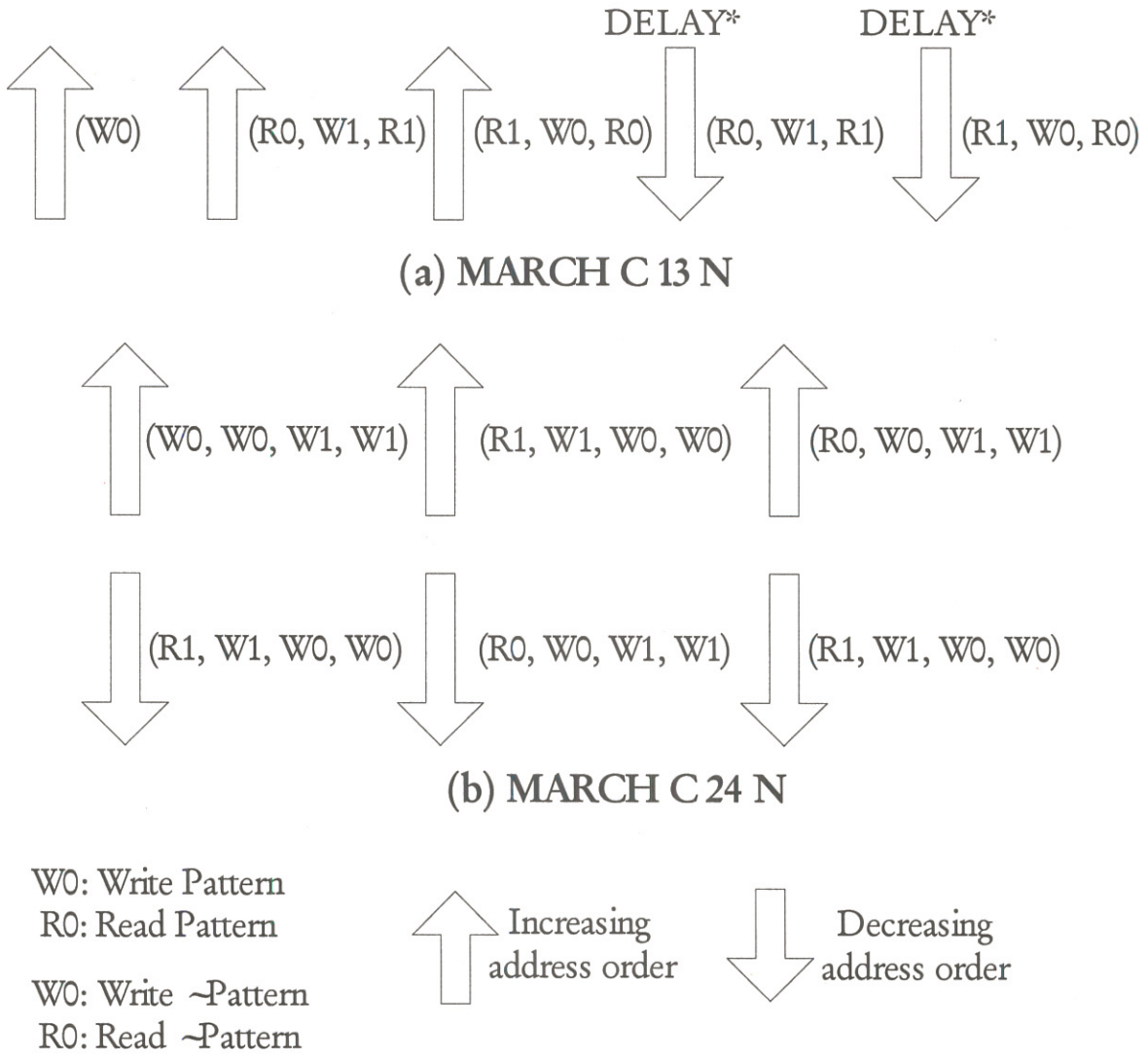
Here the read and write operation is done on memory words instead of just single cell at a time, which required the pattern size to increase from 'one' to 'n' bits, known as the *Data Background (DB)*. The MARCH algorithm proceeds by performing sequence of read and write operations on a single memory word before moving on to the next address location in the sequence as shown in Figure 14. In total 13 read and write operations are done on every memory cell/word in MARCH C 13N while 24 read and write operations are done in case MARCH 24N algorithm is implemented in the GBIST engine.

Word oriented RAMs introduce the problem of state coupling faults between two cells at one address location. To detect these faults all states of two arbitrary cells at same address must be checked. This is only possible if several Data backgrounds are used[1]. The number of patterns (DBs) required to test the memory of given address location depends on its word size, i.e. the number of bits in a word. If a memory consists of 'm' bits per word then a minimum of K data backgrounds will be needed to test the memory [15], where,

$$K = \lceil \log_2 m \rceil + 1,$$
$$\lceil x \rceil := \min\{n \in Z \mid n \geq x\} \qquad \text{Eqn. 1}$$

[1] Detection of all coupling faults between two cells at the same address is not ensured by the 9N, 13N or any other test algorithm with one data background. The proof of detection of state coupling faults, only holds for cell at different addresses[17]

Data Backgrounds for m= 256 are given in Table 2.



DELAY*: MARCH algorithm is paused only if Retention testing is enabled in the BIST Engine This sequence will be repeated for every pattern selected

Figure 14 MARCH C Algorithm

Table 1 Number of Patterns required for testing RAM upto 256 bits

Max RAM Data width	Number of Patterns	pattern #0	pattern #1	pattern #2	pattern #3	pattern #4	pattern #5	pattern #6	pattern #7	pattern #8
Upto 4 bits	3	X	X	X						
Upto 8 bits	4	X	X	X	X					
Upto 16 bits	5	X	X	X	X	X				
Upto 32 bits	6	X	X	X	X	X	X			
Upto 64 bits	7	X	X	X	X	X	X	X		
Upto 128 bits	8	X	X	X	X	X	X	X	X	
Upto 256 bits	9	X	X	X	X	X	X	X	X	X

Table 2 MARCH Patterns

	Pattern Applied
March pattern 0	256'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000
March pattern 1	256'h5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555_5555
March pattern 2	256'h3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333_3333
March pattern 3	256'h0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F_0F0F
March pattern 4	256'h00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF_00FF
March pattern 5	256'h0000_FFFF_0000_FFFF_0000_FFFF_0000_FFFF_0000_FFFF_0000_FFFF_0000_FFFF_0000_FFFF
March pattern 6	256'h0000_0000_FFFF_FFFF_0000_0000_FFFF_FFFF_0000_0000_FFFF_FFFF_0000_0000_FFFF_FFFF
March pattern 7	256'h0000_0000_0000_0000_FFFF_FFFF_FFFF_FFFF_0000_0000_0000_0000_FFFF_FFFF_FFFF_FFFF
March pattern 8	256'h0000_0000_0000_0000_0000_0000_0000_0000_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF

These patterns area applied to the memory cell array in succession to each other, with no one pattern having a priority over the other. Hence a complete test for word oriented RAM will thus proceed by first running the test algorithm with data background 1, then run it again with data background 2 and so on, until the all required number (K) of pattern have been applied.

The MARCH C 13N is the most wide used algorithm that is to test embedded RAMs in a SoC. It has a test length of 13N, where N is the number of address locations in the memory. Since K patterns are applied to a memory during testing, using a single pattern every time the BIST is run, hence the total test length for MARCH C 13N algorithm is effectively 13N X K.

Figure 15 shows the simulation results for GBIST engine testing a 64X64 RAM having a mux factor of 2 using the MARCH C 13N algorithm. The GBIST engine is operating in the 1st MARCH state (↑ W0) and is writing the pattern 0 (Table 2) to the RAM cells in increasing order of address. Here it can be seen that the address are incremented in a row-fast mode.

RAMs designed with a late write buffer mechanism require multiple write through the buffer to the array. Writing to an address and immediately reading the contents stored in the array requires performing a write to a different memory location to flush the late write buffer. Any read to the address of the last write will access data stored in the late write buffer bypassing access to the array. A write operation to a new address flushes the buffer and enabled read from the array. MARCH 24N an extension of the IFA-13 algorithm makes all sequences the same number of operations per address forces writes through the late write buffer to the array as shown in Figure 14 (b),[24]. This algorithm incorporates write recovery where every bit cell is written with the true and complement data during every write sequence.

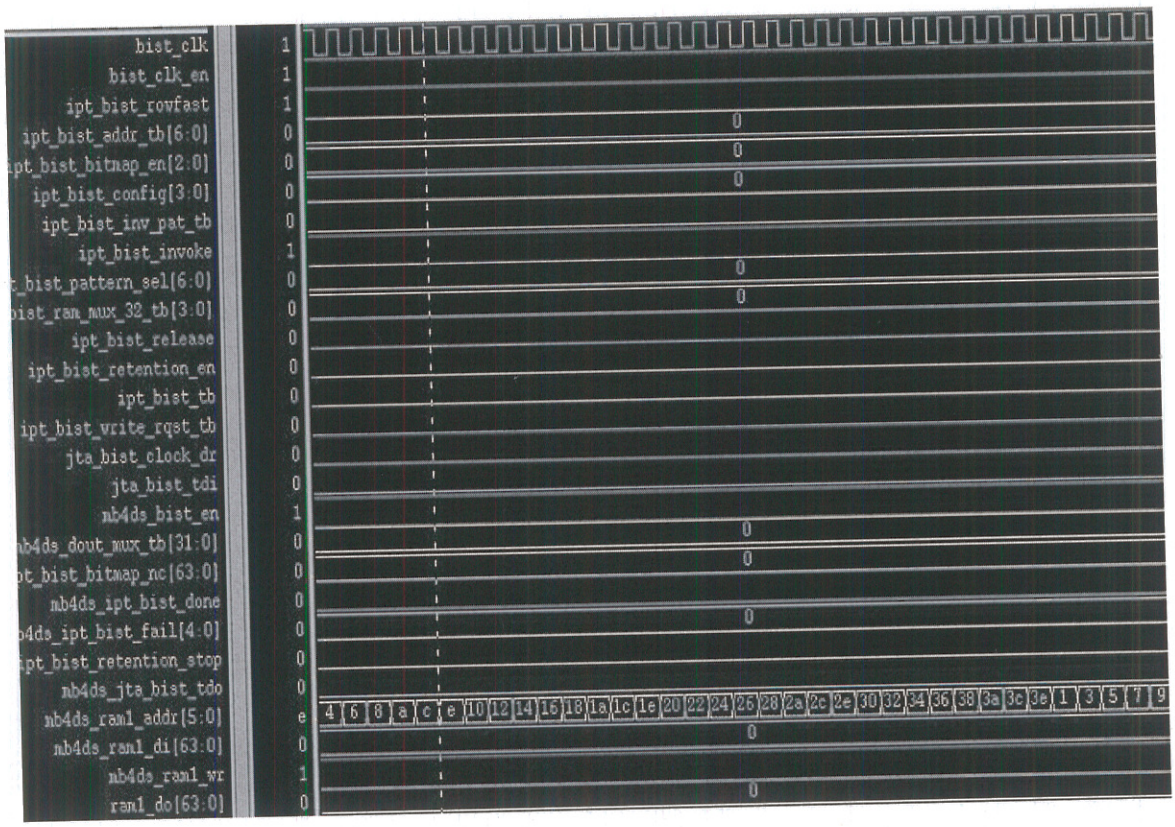


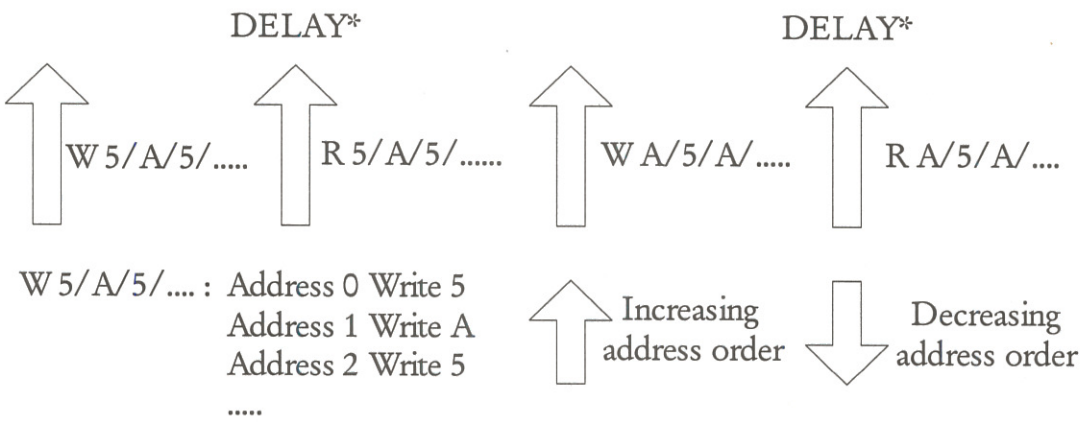
Figure 15 MARCH C 13N Algorithm in 'W0' (patterns 0) state, with Row fast Addressing for 64X64 RAM with Mux factor = '2'

MARCH C tests, because to their reasonable test time and simplicity, have been found to detect Stuck-at faults, Transition faults, addressing faults, and Coupling faults. The MARCH C 13N also covers Stuck-open faults and with introduction of two delay phases in the algorithm Data Retention faults are also be covered.

4.2.2 Checkered Board Algorithm

Checker Board algorithm consist of a sequence of read or/and write operation on each and every cell of the memory, either in increasing or decreasing order of memory address. The pattern is written to the memory such that a checker board is formed on the memory cells, i.e. every memory cell storing bit-1 has memory cells storing bit-0 (complement data) in its immediate surrounding and vice-versa, as shown in Figure 17.

Here 2 operations each of read and write are performed per memory word/cell hence the total test length is 4N, where N is the number of address locations in the memory. The Checker Board algorithm detects addressing faults, Stuck-at faults, a few Transition faults and coupling faults. With the addition of delay phases, as in the case of MARCH C algorithm, the algorithm can be utilized to test Data retentions faults as well.



DELAY*: Algorithm is paused only if Retention testing is enabled in the BIST Engine. This sequence will be repeated for every pattern selected

Figure 16 Checkered Board Algorithm

4.3 Effect of Address & Data Scrambling on Checker Board Pattern

Originally the Checker Board algorithm proposes a pattern sequence consisting of alternate 5 and A (values in Hex) i.e. 5/A/5/A/.. as shown in Figure 16 i.e. a writing '5' and 'A' alternatively on adjacent address locations followed by a read operation on successive addresses. This pattern creates a checker board on the entire memory cell array as shown in Figure 17.

	Bit0	Bit1	Bit2	Bit3
Row0	0	1	0	1
Row1	1	0	1	0
Row2	0	1	0	1
Row3	1	0	1	0
Row4	0	1	0	1
Row5	1	0	1	0
Row6	0	1	0	1
Row7	1	0	1	0
Row8	0	1	0	1
Row9	1	0	1	0
Row10	0	1	0	1
Row11	1	0	1	0
Row12	0	1	0	1
Row13	1	0	1	0
Row14	0	1	0	1
Row15	1	0	1	0

Figure 17 Checker Board Pattern (5/A/5/A/...) for 16X4 RAM

This pattern works well in case the memory words are not folded (either distributed or adjacently, Figure 8). But in case the memory words are folded in a distributed fashion as shown in Figure 8 (b), the Checker Board pattern needs to be modified so that a checker board pattern is obtained on the memory cell array.

Instead of using a pattern of alternating 5's and A's, a pattern constituting of alternating 0's and F's is used [23], incase the memory words are arranged in distributed fashion, otherwise the normal sequence of alternation 5's and A's is applied to the memory as shown in Figure 18.

Further in case of using the alternation 0's and F's pattern, the write and read pattern sequence of the BIST engine is modified according to the addressing mode implemented in the BIST engine:

1. Column-Fast Addressing

   Pattern that is to be written to the memory is inverted in every even numbered row (incase the memory words are folded). For example in case of a 16X4 RAM

(16 rows having 4 bits each) 0101/1010/0101/1010/..... (5/A/5/A/...) pattern is written to the memory Figure 17. But in case this memory is arranged as 8X8 (8 rows having 8 bits each) and each row has words arranged as shown in Figure 18 (c) so that a distributed folding is achieved, then 0000/1111/1111/0000/0000/1111/1111/0000/..... (0/F/F/0/0/F/F/0....) must be written to the memory to achieve checkerboard, which would not be achieved if the original pattern is used Figure 18 (a). During the read cycle similar corrections are made to the BIST memory interface so that correct data are matched with the contents read from the memory.

2. Row-Fast Addressing

Pattern that is to be written to the memory is inverted in every even numbered column (incase the memory words are folded). In this case for the same memory arrangement the pattern would now be 0000/1111/0000/1111/0000/1111/0000/1111/1111/0000/..... (0/F/0/F/0/F/0/F/F/0/F/....) must be written to the memory to achieve checkerboard as shown in Figure 18 (d), which would not be achieved if the original pattern is used Figure 18 (b). During the read cycle similar modifications are also done in the BIST memory interface to make sure correct data is compared with data read from the memory.

Here it must however be noted that in case the memory has a Mux factor of '1' (one) then the Row fast Address is the same as the Column fast Address.

	Bit0		Bit1		Bit2		Bit3	
	Word0	Word1	Word0	Word1	Word0	Word1	Word0	Word1
Row0	0	1	1	0	0	1	1	0
Row1	0	1	1	0	0	1	1	0
Row2	0	1	1	0	0	1	1	0
Row3	0	1	1	0	0	1	1	0
Row4	0	1	1	0	0	1	1	0
Row5	0	1	1	0	0	1	1	0
Row6	0	1	1	0	0	1	1	0
Row7	0	1	1	0	0	1	1	0

(a) Column-Fast Mode (5/A/5/A/....)

	Bit0		Bit1		Bit2		Bit3	
	Word0	Word1	Word0	Word1	Word0	Word1	Word0	Word1
Row0	0	0	1	1	0	0	1	1
Row1	1	1	0	0	1	1	0	0
Row2	0	0	1	1	0	0	1	1
Row3	1	1	0	0	1	1	0	0
Row4	0	0	1	1	0	0	1	1
Row5	1	1	0	0	1	1	0	0
Row6	0	0	1	1	0	0	1	1
Row7	1	1	0	0	1	1	0	0

(b) Row-Fast Mode (5/A/5/A/....)

Col0
Col1

	Bit0		Bit1		Bit2		Bit3	
	Word0	Word1	Word0	Word1	Word0	Word1	Word0	Word1
Row0	0	1	0	1	0	1	0	1
Row1	1	0	1	0	1	0	1	0
Row2	0	1	0	1	0	1	0	1
Row3	1	0	1	0	1	0	1	0
Row4	0	1	0	1	0	1	0	1
Row5	1	0	1	0	1	0	1	0
Row6	0	1	0	1	0	1	0	1
Row7	1	0	1	0	1	0	1	0

(c) Column-Fast Mode (0/F/0/f/...)

	Bit0		Bit1		Bit2		Bit3	
	Word0	Word1	Word0	Word1	Word0	Word1	Word0	Word1
Row0	0	1	0	1	0	1	0	1
Row1	1	0	1	0	1	0	1	0
Row2	0	1	0	1	0	1	0	1
Row3	1	0	1	0	1	0	1	0
Row4	0	1	0	1	0	1	0	1
Row5	1	0	1	0	1	0	1	0
Row6	0	1	0	1	0	1	0	1
Row7	1	0	1	0	1	0	1	0

(d) Row-Fast Mode (0/F/0/F/....)

Figure 18 Effect of Data Scrambling (Distributed folding) on Checker Board Pattern on a 16X4 RAM arranged as 8X8 RAM

It has been demonstrated experimentally [5][6] that the fault coverage of a test varies by about 35% by using different addressing sequences and/or by using different *Data Backgrounds (DBs)*, while it has become an industrial practice to apply a set of test algorithms several times, every time using a different pair of DBs in order to compensate for the effects of scrambling. Hence the GBIST uses K different patterns while testing embedded RAMs with MARCH algorithm, and also provides two modes of addressing, namely Row-fast and Column-fast addressing, as explained earlier. The Checker-Board algorithm, however, uses only one pattern (pattern 0 or pattern 1of Table 2, depending

of the memory muxing) during testing of RAMs, but it can also be operated in Row-fast and column-fast addressing modes.

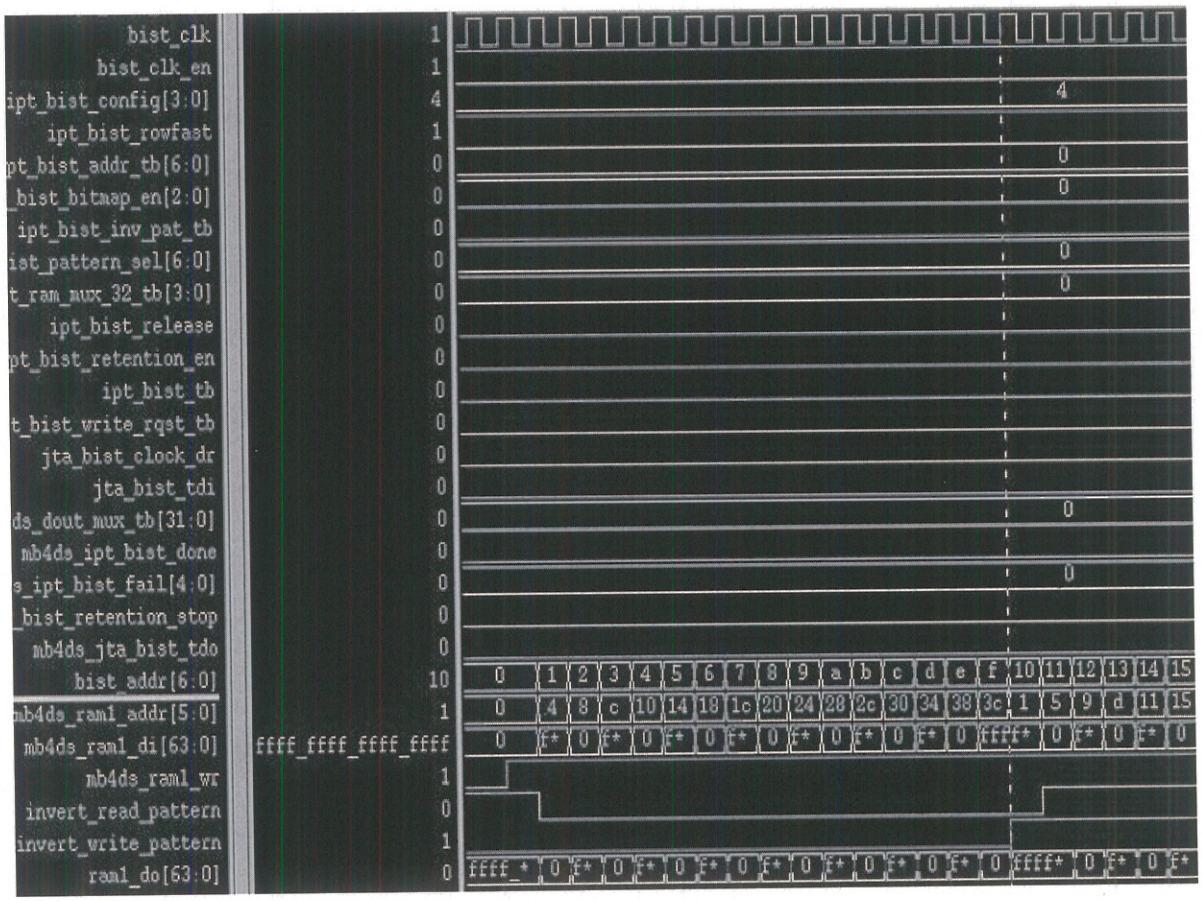


Figure 19 Checker-Board Algorithm W 0/F/0/F/... with Rowfast Addressing for 64X64 RAM with mux factor ='2'

To detect Data retention faults separate data retention enable input signal is provided to the GBIST engine which when set would cause the GBIST operation to pause after every 3rd (↑ R1, W0, R0) and 4th (↓ R0, W1, R1) MARCH step. The same is true for the Checker Board algorithm, with the pause sequence being set after every 1st (↑ W 5/A/5/..) and 3rd (↓ W A/5/A/...) Checker Board state

4.4 Testing Embedded ROMs

The Generic BIST engine (GBIST) can test any number of RAMs as well as ROMs simultaneously. It can also be used to test only embedded ROMs also. All the algorithms that are used for testing of RAMs are used testing of ROMs as well.

For testing ROMs a *MISR (Multiple input signature register)* based approach is utilized. Since ROMs cannot be written, as they are read only memories, the read sequences of the algorithms implemented in the GBIST (Checker-Board) are used to build the MISR for the ROM under test.

There are three methods to test ROMs [23]

4.4.1 Signature equivalence reference word technique

In this method the MISR that would be obtained from a correct (fault free) ROM is calculated and provided as a reference word to the GBIST ROM interface. When the final word is read from the ROM the MISR thus generated in the ROM interface is compared with this reference word Figure 21. If both these values are same then the fail bit for the ROM is not set otherwise the fail bit is set to 1. The advantage of this method is that the MISR compare register compare register would have a value zero if the generated MISR and the reference word are the same else its value would be different (which makes debugging easy), and only a fail bit needs to be monitored at chip level instead of the MISR itself for checking the ROM pass/fail status.

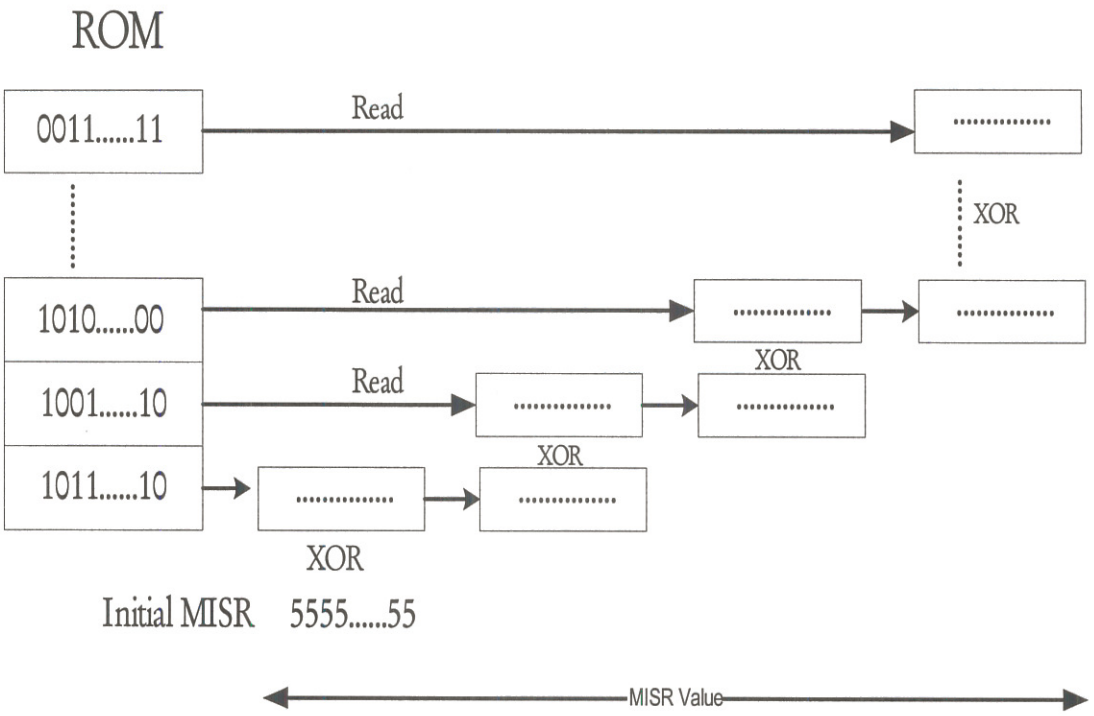


Figure 20 ROM MISR creation

4.4.2 Complementary last word technique

In this method we place the complementary MISR[2] value at the last address location of the ROM. Hence the final MISR thus obtained would constitute of all zeros. The initial MISR is generated from the 1st n-1 address locations of the ROM under test and the complementary MISR is placed at the last address location (n). This method has the advantage that it makes the pass/no-pass process easy on the tester. Since the final value of the MISR must be zero so any error in the ROM will reflect a number that is easy to detect during the testing process.

[2] Complementary MISR is the values which when placed in the last address location of the ROM would make the final value of the MISR "zero". It is not the complement of the MISR obtained from the 1st n-1 words.

4.4.3 Good old technique: Signature matching on the tester

It is called the good old technique because in this method the value of the MISR is generated and compared with the desired values at the tester itself. It increases the tester over head as it must generate the MISR from the data read from the ROM as well as store the required value of the MISR for every ROM under test.

Figure 21 shows the simulation results for the implementation of the "Signature equivalence reference word technique", described above, where the MISR generated from the ROM is compared with the reference MISR that is already provided to the GBIST unit.

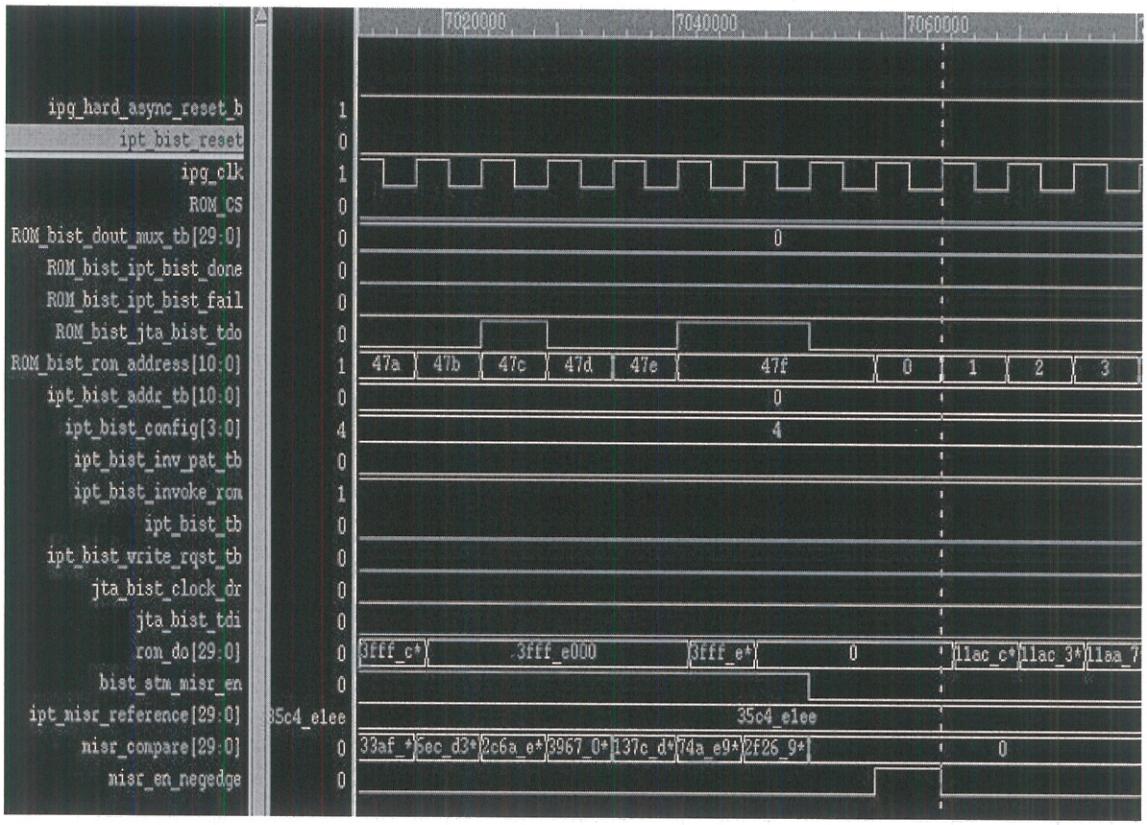


Figure 21 Testing 1184X30 ROM by comparing the generated MISR with the reference MISR value

4.5 Fail Map Extraction

Fail map extraction is required to output all the relevant data necessary to determine why a failure occurred within the memory. This data is post-processed using diagnostic software to isolate the defective location within the memory.

The problem associated with extracting the data from an embedded and self-tested memory is one of data access. The test data can be made available to the external port by using two schemes. Firstly the BIST identifies the failing memory locations and halt/suspends its operations to serially output the faulty data through the test data output port (tdo). In the second method we split the out data port into a 32 bit signal and selectively multiplex out the 32-bit data with the help of a control signal. By this way we can have the data available at the output in chunks of 32 bits each, thus speeding up the debugging process over the serial output data and yet reducing the number of output pins.

These schemes require only a few additional ports per BIST controller and an interrupt handling mechanism in the tester software to detect the failure and capture the fail data. And because only the failing data is extracted from the device, this method ensures that additional application time is minimized. The total testing time depends on the number of failures in the test and may not be identical from device to device. For devices with few defects such as those manufactured in mature processes, this method is an effective means of extracting fail data.

A yet another mechanism makes use of a small diagnostic data bus and repeats the BIST operation many times to view a different slice of the memory output for each run [11]. As with the earlier methods, the diagnostic mode is run only on the devices that fail the initial production test. When in the diagnostics mode, the BIST controller runs the entire test, and a slice of the memory output is multiplexed onto the diagnostic data bus. The test is repeated, with different slices being captured onto the diagnostic bus until all of the memory's outputs have been captured.

4.6 Increasing Manufacturing Yield and Product Reliability

Because memories are used as test vehicles for monitoring the silicon process and improving its yield, extracting additional diagnostic data to determine the causes of failures now is required in the testing strategy. In addition to diagnosis, many embedded memories are designed with built-in redundancy, which provides spare rows and columns that can replace failing locations. Redundancy enables the manufacturer to repair a number of otherwise defective devices to ensure maximum production yield.

Manufacturing yield can be improved by using RAMs having self-repair capability. These have spare rows and columns to replace those occupied by defective cells. The repairs are performed using current-blown fuses, laser-blown fuses, or laser-annealed resistor connections. Study into RAMs has shown that memory repair increased the yield from 1% to 51%.

Plotting the failure rate vs. time for integrated circuits (ICs) that operated properly after manufacturing yields the so-called bathtub curve. The bathtub curve has three main regions. Initially, the failure rate is relatively high, because many latent defects develop into actual failures. This high failure rate drops off to yield a long, flat period on the curve in which the failure rate is low. In this period the reliability of the device is high, e.g. 10 to 100 failures in 10^9 hours under normal operating conditions. The third period of the curve represents the end of the IC's lifetime. The failure rate rises as long-term, slowly developing failure mechanisms (such as electro-migration, corrosion, and mechanical stress) begin to be significant.

To compensate for the relatively high failure rate at the beginning of ICs lifetime, manufacturers typically utilize a burn-in period when the ICs are operated continuously for several hours at high temperature and possibly high voltage. This is designed to accelerate the mechanisms that cause early failures. The devices are tested following burn-in, and manufacturers therefore ship devices that are theoretically at the beginning of their high-reliability period.

4.7 GBIST - Overheads

The area overhead (gate count) of the Generic BIST is strongly linked to its environment, i.e. the number of RAMs and/or ROMs instantiated (tested by GBIST), data width and address length of these memories. Since each GBIST engine targets different number and types of memories (RAMs and/or ROMs), the gate count i.e. the number of gates, in the engine varies accordingly. Table 3 gives a few examples of rough estimated gate counts for some of the GBIST engines catering to a number of memories in various blocks of the SoC.

Table 3 Estimated gate count for various GBIST engines

Memories per GBIST Engine		GBIST	
RAMs (AddrXWrd_size)(Qty)	ROMs	Total Area (μm^2) (a)	Total Gate Count * ($\approx$ a/b)**
256X32 mux=8 (1)	--	11542.2080	4089
32X64 mux=1 (3), 48X64 mux=1 (1), 64X65 mux=1 (1)	--	31059.8105	11005
6144X32 mux=16 (1)	--	10811.9102	3831
544X32 mux=8 (3)	--	13690.0508	4850
2048X32 mux=8 (1)	--	10084.4355	3573
4096X32 mux=8 (4)	--	16619.7070	5889
--	1152X32 (1)	9047.2041	3206
2784X32 mux=8 (1), 2048X32 mux=8 (1)	--	13516.4736	4789
2048X32 mux=8 (2)	--	12328.2441	4368
64X64 mux=4 (4), 95X42 mux=1 (1)	--	28329.8398	10037
32X32 mux=1 (16)	--	27167.7168	9626
448X40 mux=8 (1)	--	10528.9648	3031
1024X72 mux=8 (1)	--	15033.5156	5327
512X144[#] mux=8 (1)	6144X128[##] (1)	39388.7070	13957

*equivalent to the same number of '2 input NAND' gates

** b= 2.8224 μm^2, '2 input NAND' gate equivalent area

[#] Two 512X72 mux=8 RAMs placed in parallel to form 512X144 configuration

[##] Two 6144X64 ROMs placed in parallel to form 6144X128 configuration

Table 4 Power estimates for GBIST

Memories per GBIST engine		GBIST		
RAMs (Addr X word_size)(Qty)	ROMs	Total Area (μm^2)	Power	
			Dynamic(μW)	Leakage (μW)
256X32 mux=8 (1)	--	11542.2080	1479.7040	26.0246
32X64 mux=1 (3), 48X64 mux=1 (1), 64X65 mux=1 (1)	--	31059.8105	2.7299 (mW)	80.6668
6144X32 mux=16 (1)	--	10811.9102	798.5840	21.8269
544X32 mux=8 (3)	--	13690.0508	846.4087	28.1875
2048X32 mux=8 (1)		10084.4355	777.7504	20.4568
4096X32 mux=8 (4)	--	16619.7070	995.6365	34.9554
--	1152X32 (1)	9047.2041	978.1149	18.3329
2784X32 mux=8 (1), 2048X32 mux=8 (1)	--	13516.4736	960.3312	29.0402
2048X32 mux=8 (2)	--	12328.2441	760.1550	26.4424
64X64 mux=4 (4), 95X42 mux=1 (1)	--	28329.8398	1468.1212	69.3350
32X32 mux=1 (16)	--	27167.7168	1781.0517	60.3835
448X40 mux=8 (1)	--	10528.9648	1.2599 (mW)	31.4896
1024X72 mux=8 (1)	--	15033.5156	1.6940 (mW)	65.1819
512X144[#] mux=8 (1)	6144X128[##] (1)	39388.7070	5.8495 (mW)	225.6561

[#] Two 512X72 mux=8 RAMs placed in parallel to form 512X144 configuration

[##] Two 6144X64 ROMs placed in parallel to form 6144X128 configuration

The synthesis of these GBIST engines has been done at the block level, hence the values mentioned in theTable 3 and Table 4 are rough estimates of the final values that would be obtained after final synthesis. The area penalty paid for implementing BIST on the memories in the system was seen to be roughly around 1.33% of the total chip area. The area and the power estimates of GBIST engines were done for the *cmos90gp (90 nm) 7m2t (7 metal 2 thick layer)* process, under worst case conditions of 0.9 V and 125°C using Synopsis Physical compiler.

CHAPTER 5

5.1 Results/Conclusions

Today's SoCs are moving from logic dominant systems to memory dominant systems. The share of semiconductor embedded memories is expected to touch 90% by 2014 [22]. Also with an ever-increasing cost and time of testing SoCs, it is essentially important to achieve a high degree of fault coverage in the minimum possible time while limiting the overheads.

Memories are fabricated much denser than random logic and therefore are more prone to defects and failures. Defects in memories are due to shorts and opens in memory cells, address decoder and read/write logic. These defects are modeled as single and multi-cell memory faults and different classes of memory test algorithms have been proposed to detect these memory faults [12]. The most popular and widely accepted deterministic test algorithms are the MARCH tests.

Application of test patterns using off-chip testers results in a high test time due to the large sizes of embedded memories. Also the limited resources (pins, memory, etc) on the external tester have forced the system designers to introduce self testing logic within the system itself more popularly known as Built-in Self Test (BIST). The computed test sequences to test faults in embedded memories are generated on-chip using a memory Built-In Self Test (BIST) unit. Generic Built in Self Test (GBIST) is a test methodology of choice that has proven its capability to achieve high fault coverage on semiconductor memories that are embedded in the heart of high density application specific SoCs, with acceptable performance and area overheads.

As the size of the embedded memories increases due to increase in address locations or word sizes, their layout needs to be modified to balance the load, parasitics and delays on its internal wires and transistors. The result is that the logical placement of

the bits in the memory cell array differs from that physical or the actual placement. Though these changes are invisible to the external user, they tend to modify the test sequence and patterns implemented inside the GBIST engines testing them.

As a result of this scrambling, the Checker-Board algorithm sequence and pattern was modified so that a perfect checker-board is formed on the memory array which is required by the algorithm. The algorithm's implementation was also affected by the type of addressing implemented by the GBIST engine, namely Row-fast and Column-fast address, which necessitated the deviation of the read and write sequences from the originally described flow of sequences in the algorithm. These changes are implemented for the memories having a muxed layout (the mux factor of the memory is greater than unity).

Built in test algorithms of the GBIST engine achieve 100% fault coverage for bit oriented and word oriented faults, including stuck-open, stuck-at, transition, address decoder, and coupling faults. The ease of generation and integration into any type of core also with added built in capability to provide data for diagnosis and fault analysis places GBIST far ahead of the rest of the similar solutions available in the industry.

5.2 The Next Step: Programmable BIST

The structure of a memory BIST unit depends on the characteristics of the memory-under-test, target fault model and the overall test strategy of the design. Memory BIST units could not be re-used from one design to the next unless they undergo a series of modifications and enhancements to take care of the new fault types which some times have even unknown behavior. This is time consuming and increases the design time-to-market.

With today's complex process nodes such as 90nm and below, failure mechanisms cannot be fully understood and planned for at the design stage. The ability to modify

the memory test program on-the-fly for both failure analysis and device screening is crucial. A memory BIST controller, that controls the flow of test sequences and other modules to generate the necessary test data and control signals, could be designed as non-programmable or programmable. The non-programmable controllers are the hardware realization of a selected memory test algorithm and cannot be modified without changing the hardware realization of the memory BIST controller.

Memories undergo different types of tests during their design and fabrication and therefore each memory BIST controller might have to generate different types of test patterns. Programmable memory BIST architectures could be used to generate different types of test patterns and might be an ideal solution to the problem of memory testing in memory-intensive designs.

[9] Bhavsar D. K., "An Algorithm for Row-Column Self Repair of RAMs and Its Implementation in the Alpha 21264," Proceedings of the International Test Conference, 1999.

[10] Crouch, A. L., Design-For-Test for Digital ICs and Embedded Core Systems, Prentice-Hall PBR, 1999, ISBN-0-13-084827-1.

[11] Liakot Ali, Roslina Sidek, Ishak Aris, Bambang Sunaryo Suparjo, Mohd. Alauddin Mohd. Ali, "Challenges and directions for testing IC", INTEGRATION, the VLSI journal 37, 2004, pp 17–28

[12] A. J. van de Goor, "Testing Semiconductor Memories: Theory and Practice", John Wiley and Sons, U. S. A, 1995

[13] Crouch, A.L., Mateja, M., McLaurin, T.L., Potter, J.C., Tran, D., "The Testability Features of the 3rd Generation Coldfire® Family of Microprocessors", Proceedings of the International Test Conference, 1999

[14] Ashok K. Sharma, "Semiconductor Memories Technology, Testing & Reliability", IEEE Press, 1997, (© 2005 IEEE, Reprinted with permission)

[15] Rob Dekker, Frans Beenker and Loek Thissen, "A Realistic Fault Model and Test Algorithm for Static Random Access Memories", IEEE Transactions on Computer Aided Design, Vol. 9, No. 6, June 1990, pp 567-572

[16] International Technology Roadmap for Semiconductors (ITRS), 2000

[17] Rob Dekker, Frans Beenker and Loek Thissen, "Fault Modeling and Test Algorithm Development for Static Random Access Memories", IEEE International Test Conference, 1998, pp 343-352

[18] R. Rajsuman, Digital Hardware testing, Norwood, MA: Artech house, 1992

[19] Miron Abromici, Melvin A. Breuer and Arthur D. Friedman, "Digital System Testing and Testable Design", Computer Science Press, New York, USA, 1990

[20] Sung-Mo Kang, Yusuf Leblebici, "CMOS Digital integrated Circuits : Analysis and Design", Third Edition, Tata McGraw-Hill, 2003

[21] "MBISTArchitect Process Guide", V8.2005_1, Mentor Graphics Corporation, Feb 2005 (© 2005 Mentor Graphics Corporation)

[22] "2001 International Technology Roadmap for Semiconductor", Computer, vol. 35, no. 1, pp. 42-53, 2002

[23] K. G. Varun, Achin Grover, Amit Sharma, "Next-Generation Freescalable DFT Architecture for high performance SoCs", 5th IDC Technical Symposium, Freescale Semiconductors Ltd, Mar 2005

[24] "Generic BIST Block Guide", Freescale Semiconductor Ltd., Revision 4.5, Dec 2003

[25] "Generic Memory BIST Design Specifications", Freescale Semiconductor Ltd., Revision 4.4, Oct 2003

[26] D. Appello, P. Bernardi, A. Fudoli, M. Rebaudengo, M. Sonza Reorda, V. Tancorre, M. Violante, "Exploiting Programmable BIST for the diagnosis of embedded memory cores", ITC International Test Confrence, 2003, pp 379-385