Republic of Iraq
Ministry of Higher Education
and Scientific Research
University of Babylon
Faculty of Engineering
Electrical Department Engineering

Built -In Self-Test Approaches for a Static Random Access Memory Array

A Thesis Submitted to the College of Engineering at the University of Babylon in Partial Fulfillment of the Requirements for the Degree of Master of Engineering\ Electrical Engineering \ Industrial electronics

By
Fanar Abbas Abdullah Hassan

Supervised by
Prof.Dr.Qais Kareem Omaran

2022A.D. **1443A.H.**

Supervisor certification

I certify that the thesis entitled **"Built -In Self-Test Approaches for a Static Random Access Memory Array"** was prepared by **Fanar Abbas Abdulla** under my supervision at the department of electrical engineering, college of engineering, university of Babylon, as partial fulfillment of requirements for the degree of master in engineering /electrical engineering/electronics.

Supervisor

Signature:

Name :**Prof. Dr. Qais Kareem Omaran**

Date: / / 2022

In view of the above recommendation, I am forward this thesis for discussion by the Examination Committee.

Head of Electrical Engineering Department

Signature:

Name: Assist. Prof. Dr. Shamam Fadhil Alwash

Date**:** / / 2022

Examining Committee Certificate

We certify that we have read this thesis entitled **"Built -In Self-Test Approaches for a Static Random Access Memory Array "** and as an examining committee examined the student **Fanar Abbas Abdulla** in its content and that in our opinion it meets a standard of a thesis for the degree of master in engineering/electrical engineering/electronics.

Signature:
Name: **Prof.**
Dr. Osama Qasim Jumah
(Chairman)
Date: / /2022

Signature:
Name: **Asst.Prof.**
Dr. Hilal Abdul Hussein Al-Libawy
(Member)
Date: / /2022

Signature:
Name: **Asst.Pro.**
Moneer Ali Lilo
(Member)
Date: / /2022

Signature:
Name: **Prof.**
Dr. Qais Kareem Omaran
(Supervisor)
Date: / /2022

Signature:
Name: **Prof.**
Dr. Hatem Hadi Obeid
(Dean of College Of Engineering)
Date: / /2022

Signature:
Name: **Asst. Prof.**
Dr. Shamam Fadhil Alwash
(Head of Electrical Department)
Date: / /2022

Acknowledgement

First I want to thank **God** for helping me and giving me the strength, willingness patience to finish what I have started.

I would like to express my gratitude, faithful, appreciation and deep respect to my supervision **Prof.Dr. Qais Kareem Omaran** for his suggestion of the project , supervision , help, continuous advice and efforts for helping me in my work.

My special thank for all my friends who encourage me to finish my study.

Also, thanks go to all **staff** in the department of Electrical Engineering, College of Engineering, University of Babylon.

I owe a special thanks to my precious family, **My Father** and **My Mother** and **My Sister** and **My brother** who supporting me and helped me throughout my life and during this study.

Dedication

Every challenging work needs efforts as well as guidance elders especially those who were very close to our heart.

My humble effort I dedicate to my sweet and loving ***Father& Mother***

I dedicate this work to my beloved my husband ***Ammar*** whose unconditional encouragement & support made it possible for me to finish my thesis.

To follower of my life ***Elias***

To my **country** I dedicate this humble work .

Table of Contents

Table of Figures

List of Tables

List of Abbreviations

AF	Address Fault
ASIC	Australian Securities and Investments Commission
ATE	Automatic Test Equipment
ATPG	Automatically Test Patterns Generating
BAE	Pause-End-Export
BIST	Built In Self-Test
BL	Bit Line bar
BL	Bit Lines
CF	Couple Fault
CFid	Idempotent Coupling Faults

CFin	Inversion-Coupling Fault
CUT	Circuit Under Test
DFT	Design For testability
DUT	Device Under Test
DWD	Hierarchical Word Decoding
DWL	Divided Word Line
EEPROM	Electronically Parsed Programmable ROM
EPROM	Erasable Programmable ROM
FPGA	Field Programmable Gate-Array
FSM	Finite State Machine
HDL	Hardware Description Language
IOT	Internet Of Things
LFSR	linear Feedback Shift Register
MATS	Modified Algorithm Test Sequences
MBIST	Memory Built In Self-Test
MEB	March-Element-Based
MSCAN	Motorola Scalable Controller Area Network
NPSF	Neighborhood Pattern Sensitive Faults
NVRWM	Non-Volatile Reading Write Memories
O(X)	Time complexity of Testing
ORA	Output Response Analyzer
OTP	One-Time Programmable device
PCB	Printed Circuit Board
PD	Pull Down
PGT	Pass-Gate Transistor
PU	pull up
RAM	Random Access Memory
RDM	Random Dynamic Memory

RERAM	Resistive Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer level
RWM	Read-Write Memory
SAF	Stuck At Fault
SOC	System On Chip
SRAM	Static Random Access Memory
SS	Static Simple
TF	Transition Faults
TPG	Test Pattern Generator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integrated
WL	Word Line
β	beta ratio

Abstract

Built-In Self-Test (BIST) has become an important factor in the current manufacturing processes of memories to increase reliability and to improve the memory yield. The effectiveness of any test algorithm depends on the fault coverage and speed of detecting the fault. In this study , two BIST approaches for a Static Random Access Memory (SRAM) have been proposed. The first approach is Fast March algorithm while the second is Finite State Machine (FSM) algorithm. In the first approach, the traditional zero-one algorithm has been developed to Fast March algorithm utilizing word orientation method in which different steps of test pattern have been employed. In the second approach, the traditional March- C algorithm has been developed to Finite State Machine (FSM) algorithm where it has become dependent on the word orientation technique utilizing suitable test pattern.

The two approaches have been applied on two sizes of memory (16x8) bit word and (32x8) bit word where four types of faults have been assumed to verify the effectiveness of these algorithms. The assumed faults are Address Fault, Stuck -At Fault, Coupling Fault and Transition Fault.

The simulation results have proven that the Fast March is faster than FSM March algorithm three times (16 operations vs. 40 operations), but it can only detect one type of fault which is Stuck -at -fault. FSM March algorithm is slower than Fast March, but it is more reliable and can detect most of the memory faults. The simulation has been carried out by using Quartus II 13.0sp1 and ModelSim-Altera 6.5b software where VHDL language was employed in this work with processing time 100 MHz clock frequency.

CHAPTER 1

Chapter One

Introduction

1.1 General Introduction:

Built-in Self-Test (BIST) is a diagnostic system that is used to evaluate the circuit operation using part of the same circuit's components. The extra hardware expenditure would most compensate because of its advantages in terms of increased reliability and lowest servicing costs with a properly built BIST architecture. BIST lowers overall costs by reducing automatic test development attempts at all levels, reducing chip testing, improving unit level maintenance, and improving component replacement [1], [2]. the operational real-time clock is very useful, so that several circuits can be tracked with limited testing time. It reduces the length of the testing process [3]. Memory chips are used in all household appliances, computers, and laptops. As a result, BIST memory aids everyone in providing better service [4].

The primary motivation for testing is to confirm that the Device Under Test (DUT) with integrated circuits, Printed Circuit Boards (PCB), and system are capable of producing expected outcomes when applied with the specific inputs. Simple tests often result in “pass” or “fail” test results, while more advanced testing procedures additionally evaluate the DUT functionalities, discover errors, and display their locations. It is likely that the errors or faults will happen in a repeating pattern, which suggests a process adjustment is required [5]. A novel study that is expected to go a long way in developing VHDL (Very High Speed Integrated Circuit Hardware Description Language) and that has been blessed by BIST would thus allow standard RAM chips to be developed using these new hardware features [6].

1.2 Problem statement

The embedded memories, especially the embedded random access memories (RAM), are one of the most common components in electronic devices, as they are used to store the data necessary for the device’s work, and any defect in it leads to a defect in the device’s work or may lead to its permanent cessation of work. Therefore, it was necessary to test it and ensure that it is free of faults during each operation of the device. However, this test process is one

of the most complex processes as it requires sending a large amount of test patterns and processing these data after retrieval from memory to ensure their conformity. The solution for this problem is by adding a special circuit for testing embedded memory, which is called the Built-In Self-Test circuit (BIST).

Most memory self-test algorithms, such as Zero-One and March C, target bit-oriented memory, but current memories are word-oriented. Therefore these algorithms had to be developed for test word-oriented memories effectively.

1.3 Literature Survey

Memory Built-In Self-Test (MBIST) technique has been implemented for testing embedded memory. There has been considerable research on developing BIST algorithms, which known as March algorithms. The early March algorithms were based on bit-oriented memories[7]–[10], but they did not work effectively when testing the word-oriented memories [11], [12]. Van de Goor [13] has devised a method for translating March tests for bit-oriented memories into rests for word-oriented memories in a systematic way, by using a test patterns. The word-oriented March tests algorithms can be implemented using a Microcode [14]–[17], or a FSM [15], [18]–[22] for complicated March algorithm, and linear feedback shift register (LFSR) [23], [24], or counter [3], [15], [25], [26] for simple March algorithms.

Jain and Stroud 1986 [7] provided a way for evaluating embedded memories using a Built-In Self-Test (BIST), by proposing two algorithms. The first algorithm was based on pseudorandom data patterns and used to detecting stuck-at faults. The second algorithm was based on checkerboard patterns and used for detecting coupling faults. The testing performance of this method was suitable for small size and simple architecture (bit-oriented) memories which used at that time, but it could not be used to test the current large and complex word-oriented memories.

Van de Goor and Tlili 1998 [13] suggested a unique approach for word-oriented memory assessment that differentiates inter-word and intra-word faults and allows possible to convert bit-oriented memory tests from word-oriented mimic tests systematically. The inter word mistake test was combined with the intra word conversion test. This technique resulted in more effective testing covering all the problems being examined. Word-oriented memory

tests were crucial since most memories have an external data channel that is larger than one bit.

Venkatesh et al. 2002 [9] presented robust memory fault model to validate a BIST logic before fabrication, by proposing a novel fault modeling technique. Equations are used to express the memory architecture and the location of any faults that may occur in the cell array. The method used these equations to calculate a location where the fault may be modeled. However, this method was not generic and was targeting a specific type of memory (bit-oriented).

Bayraktaroglu et al. 2005 [14] presented a Microcode MBIST architecture based on dual-port memory testing algorithms proposed by Hamdioui and van de Goor [27]. The proposed architecture supports a wide range of memories with various test and debug support requirements. However, the implementation of this architecture was complicated and required special type of future generation spark microprocessor.

Haron et al. 2007 [20] used bit-oriented March C- method to model and simulate a Finite State Machine MBIST. Xilinx ISE(International Securities Exchange) tools were used to create the design architecture in VHDL code. Using register transfer level abstraction, the architecture was modeled and synthesized. Stuck-at-fault SRAM tests were used to validate this architecture. However, this method used traditional bit-oriented MARCH algorithms for testing word-oriented memories, so it can detect only simple Stack-at faults.

Haron et al. 2007 [11] suggested to find a stack-at-fault in the SRAM under test. Here a microcode MBIST modeling and simulation employing five BIST algorithms (MATS, MATS+, MARCH C, MARCH X, MARCH C-). Register Transfer Level (RTL) abstraction was used to model and synthesize the architecture, and it was written in VHDL code using the Xilinx ISE tool. However, this method used traditional bit-oriented MARCH algorithms for testing word-oriented memories, so it can detect only simple Stack-at faults.

Masnita et al. 2009 [8] presented the fault models and compared existing March algorithms for detecting and diagnosing Transition -Faults (TFs)and Stuck-At Faults (SAFs) .Modified Algorithm Test Sequences(MATS++) and March C- methods were described for detecting these errors. If just stuck-at-faults and transition -faults are examined, an technique with a 5N minimum test length was shown to be sufficient for increasing test

efficiency, according to this research. The usefulness of March CL and March C- diagnostic algorithms are also put to the test based on their fault syndromes.

Masnita et al. 2009 [12] reported the construction of a Memory Built-in Self-test (MBIST) information and read/write organizer using the March-based method suggested in [8]. The design employed a Finite State Machine (FSM)-based architecture, which was more credible given that the design was part of an engine that would be constructed only for the purpose of evaluating this method. It was a component of the MBIST motor that could be coupled to other parts to form a full MBIST motor.

Liyan et al. 2009 [16] presented the architectures design for Micro-coding and FSM-based controller a novel test technique which offered increased coverage of fault in single-cell failure detection and intra-word Coupling Fault (CF). The MBIST, which uses microcode's, utilized 86 cases but operated at a pace quicker than the FSM control, (77 instances), which were operating at a slower speed.

Noor et al. 2009 [22] presented FSM-based BIST design employing a unicounter memory BIST controller based on the FSM architecture of MARCH SAM. In a low-scale synchronous SRAM this BIST controller was capable of detecting all intra-word coupling failures (CF) and simplified SRAM synchronous testing. This was a simple and economical design since it reduced the amount of overhead at a pace while still functioning.

Sharma and Sood 2010 [17] offered a micro coded MBIST architecture that may be used to construct new March tests March Static simple (March SS) with a higher number of operations per element to meet the growing need for embedded memory testing. Because it provided a flexible methodology and higher fault coverage, this architecture was a useful testing method for embedded memories.

Husin et al. 2012 [3] presented a counter MBIST architecture using simple word-oriented March C- algorithm, which designed using VHDL. This MBIST architecture was fast and simple but could only detect Stack-at faults.

Chih-Sheng Hou et al. 2014 [19] suggested a March-Element-Based (MEB) compression method that compressed diagnostic information in a Pause-And-Export (PAE) form after all. March element operations were performed for an addressed word. Decrease of

duplicate fault diagnosis data, as a result. It is also possible to discover dynamic problems that are time-elated.

Reeja and Anusree 2014 [26] Proposed a built-in 8- bit SRAM test based on the March C- algorithm utilizing the VHDL coding approach. It can include host defects such as blocking failures, transition failures and many connection failures and decoder errors. However, this method applicable only on bit-oriented memories, and can't test word-oriented memories effectively.

Joseph and Antony 2014 [15] compared three MBIST controller designs based on March algorithms were on the grounds of size, power, and complexity: Microcode MBIST, FSM centered MBIST, and Counter centered MBIST. Counter-based MBIST controllers outperform microcode and FSM-based ones in terms of size, and complexity. Because System on Chip (SoC) use several memory modules and require a variety of testing approaches, the counter-based method is an excellent choice for memory module testing.

Ahmed et al. 2015 [18] presented a memory BIST controller design using the FSM approach for March 17N. It looked for flaws in various recollections. A simple architecture was created in Verilog Hardware Description Language (HDL), which could simply be integrated with SoC and could discover faults in semiconductor memories.

Koshy and Arun 2016 [25] presented a comparison between two proposed MBIST designs: BIST scheme with MEB Compression and BIST scheme with the fusion of March Algorithms. The BIST with MEB Compression scheme effectively reduced the exported diagnostic data at the time of redundancy. It compressed the diagnostic data in a PAE form when all the operations of a March element were completed for an addressed word. Consequently, the redundant diagnostic data of a fault could be minimized. A BIST scheme with the fusion of March Algorithms had been proposed for the various fault detection in RAMs. Design with Counter-based BIST architecture reduces the complexity. Detection of different faults in a RAM memory using a single counter based BIST design was advantageous.

Maneshinde et al. 2016 [21] proposed MBIST controller based on FSM , which had fewer number of situations comparing to the previous designs [15]. To attain slowered area and enhanced speed. However, this method applicable only on bit-oriented memories, and can't test word-oriented memories effectively.

John and Antony 2017 [23] suggests a method for testing multiple Memory cores in parallel using Configurable Linear Feedback Shift Register (CLFSR) structure based on bit-oriented March C- algorithm. A Configurable CLFSR structure was obtained by combining two CLFSR using additional control logic. This method was fast but can detect only Stack-at faults. Our proposed FSM March algorithm is better, because it can detect most of memory faults.

Tewary et al. 2020 [24] proposed a standard RAM chip with BIST enabled architecture based on a linear feedback shift register (LFSR) that revealed and supported with VHDL. The proposed BIST enabled RAM is designed using VHDL and implemented successfully into SPARTAN 6 Field Programmable Gate Array (FPGA) board. The method can be extended to test any number of memories in an IC in parallel. They concludeed that BIST enabled RAM tests itself to enhance safety and reliability. This method applicable only on bit-oriented memories and can't test word-oriented memories effectively.

Wei Chun et al. 2020 [10] offered a low-cost memory fault detection device that could be used to test for a variety of memory issues. The Automatic Test Equipment (ATE) memory tester's zero-one scan tests, and March algorithms were all modeled using the VHDL. However, this method was targeting only a specific type of memory (bit-oriented).

Sasikumar et al. 2021 [28] introduced a new enhanced MBIT unit that is scalable and features MISR signature analysis. The proposed MBIT architecture surpasses the conventional architecture as it enhances the fault detection resolution in memories. Through utilizing only 14 states for 7 March algorithms, the novice Algorithm was optimized, which resulted in better scalability of the MBIST with absence of area overhead. The introduced algorithm also has proved to have better resolution and accuracy due to the interface check MISR that helps in isolating the fault detection in memory interface and memory. MISR additionally allows pre-check of memory, resulting in rapid fault check.

All previous research focused on the development of MBIST systems. Some of them were interested in developing a MBIST systems that targeting a specific type of memories (bit-oriented) in order to make performance as best as possible by using simple algorithms such as pseudorandom and checkerboard and zero-one algorithms which applied using simple architectures such as counters or LFSRs, but it can't be used with other types of memories

and doesn't cover all types of memory faults. While others tried to develop a comprehensive MBIST system for all faults and memory types, by applying advanced algorithms such as March C- and MATS using complex architectures such as microprocessors and state machines programmed with hardware description languages (HDL), but the performance of this systems was weak, especially with modern and large memories.

In this work, two MBIST algorithms are developed and they are Fast March and FSM March that combine comprehensiveness and performance then implementing them using VHDL language.

1.4 Aims of the Study

The aims of this work is summarized as follows:

1. Design and implement a functional fault model for all SRAM faults using VHDL language.

2. Develop word-oriented memory testing algorithms (Fast March and FSM March) from classical Zero-one and March C- algorithms.

3. Detecting a wide range of SRAM faults by the proposed system.

1.5 Thesis outline

Chapter one: In this chapter, background about the BIST , literature review, aims of the study , and thesis outline were discussed in this chapter.

Chapter two: In this chapter, the Memory Testing Methodologies have been documented ,Why we use SRAM, Built-In Self-Test (BIST) ,March Algorithms are also discussed.

Chapter three: The Proposed methodologies for Fast March Algorithm and FSM March Algorithm are presented. In addition, VHDL design for Fast March and FSM march Algorithms are discussed.

Chapter four: The simulation results based on the algorithm of all case studies are presented and discussed in this chapter.

Chapter five: The conclusion and future works are drawn in this chapter

CHAPTER 2

Chapter Two

2. Memory Testing

2.1 Why Memory Built-in Self-test?

Nowadays, without semi-conductive memory any digital system is not possible, particularly in integrated circuit and System on Chips (SoC). They are extensively used for program storage, information configuration and other enormous quantities of data. The capacity and speed of the memory are directly connected to the performance of many applications. There are more built-in memories and the total integrated memory content of the SoC. Since memories are an essential element of the typical SoC, any advances in memory design and construction impact a range of features, including cost, performance and reliability of the complete SoC directly and substantially [29] .

Embedded memories occupy the main part of the chip in modern software and grow its ability, density and complexity linearly over the year. On the other side, fault types become increasingly complicated and diversified and can be avoided through testing. So the embedded memories testing devices like Automatic Test Equipment (ATE) become more complex and too costly (at least a million dollars per ATE tester). A promising solution to this dilemma is MBIST which adds low-cost test circuitry to the memory itself and provides an acceptable yield.

2.2 Types of Semiconductor Memory

Semiconductor memories can be classified according to the type of data storage and data access mechanisms in three main families (Figure 2.1): the Read–Write Memories (RWMs), which is most often called Random Access Memory (RAM), Read Only Memories (ROMs), and Nonvolatile Read–Write Memories (NVRWMs or NVRAMs)

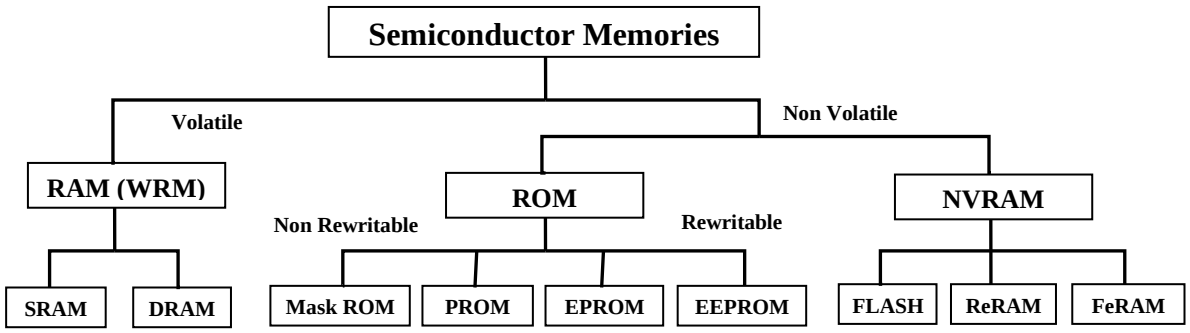


Figure 2.1Memory taxonomy [29]

RAM is a volatile memory type. This implies that when power is switched off, its content is gone. A RAM device may read or write data items for about the same length of time, regardless of where the data is located in the storage. Random Dynamic Memory (RDM) and SRAM are two key kinds of RAM. DRAM has transistor and condenser connected memory cells. The data must be updated often to maintain the data. SRAM must not be updated. For each memory cell, it employs many transistors but has no condenser. It stores the data till the power is Stopped.

Normally, SRAM is quicker than DRAM since no refresh cycles are available. The cost per cell in SRAM, however, is considerably greater than in DRAM. SRAM is mostly utilized for caching, whereas DRAM is better for primary storage. Many embedded systems nevertheless incorporate both RAM types—a tiny block of SRAM along the crucial data line and a somewhat bigger DRAM block for everything else [29].

The ROM is a non-volatile storage type. It is a persistent memory that keeps your saved data when the power is shut down eternally. ROM is normally read only and can't be exceeded by memory. Numerous types of ROMs can be reprogrammed, although (Erasable ROM, Programmable ROM, EPROM, etc.). By the methods used, it can be told how often fresh information can be rewritten (reprogrammed). This category is the progression of hardwired to programmable and delectable ROM devices. The fact that all of these devices are capable of storing data and programs indefinitely in the event of a power failure is a common feature of all of these devices [30]. Hardwired memories are still being utilized but are now recognized to stand out as "masked ROMs" from other types of ROMs [29].

A disguised ROM's principal benefit is its cheap cost of manufacturing. Sadly, costs are cheap only when huge amounts of the same ROM are needed. The PROM is identical to the

traditional ROM, with the exception that the consumer may program it. A specific device called the PROM programmer can be used to program PROM. All bits read under "1" are provided using a standard PROM [30].

A PROM programmer then employs high voltages to break connections in the chip permanently. Therefore, PROM is commonly referred to as a One-Time Programmable device (OTP). EPROM is a specific PROM form, which can be scheduled and erased. The programming can only be read. If exposed to a high UV source, EPROM can be damaged. It may be reprogrammed once it is deleted. The same programming is available in EPROMs as in PROMs.

EEPROM is the same internally as EPROMs, except electric deletion is carried out rather than UV radiation. Each bit of an EEPROM may be removed and rewritten. After writing new information, it remains on the device forever. This enhanced capacity is mostly at a disadvantage because to higher costs, but writing cycles are too much longer than on a RAM [30].

Nonvolatile random access memory (NVRAM) is a third family of memories that combines the main features of both RAMs and ROMs. It represents a large variety of memory technologies that can retain the data even when the power is cut off. This is in contrast to DRAM and SRAM, which both retain data only for as long as the power is on. The best-known form of NVRAM memory is flash memory [31]. It combines the best features of the memory devices described thus far. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable. These advantages are overwhelming and, as a direct result, the use of flash memory has increased dramatically in embedded systems. From a software viewpoint, flash and EEPROM technologies are very similar. The major difference is that flash devices can only be erased one sector at a time, not byte-by-byte. Typical sector sizes are in the range of 256 bytes to 16KB. Despite this disadvantage, flash is much more popular than EEPROM. Resistive random access memory (ReRAM) is regarded as one of the most promising alternative nonvolatile memory technologies for its advantages in very-high-storage density, simple structure, low power consumption, and long endurance as well as good compatibility with traditional complementary metal oxide- semiconductor (CMOS) technology [31]. It works by

changing the resistance across a dielectric solid-state material and shows the greatest potential in massive data storage.

2.3 Static Random Access Memory (SRAM):

Static Random Access Memory (SRAM) remains the workhorse of memory. There are a higher number of SRAM memories, especially when considering built-in memory globally. In many applications, SRAMs are quietly introduced. The earliest memories were made of SRAMs. SRAMs are quick and are used when the fastest speed memory, such as L1 microprocessor caches, are needed. They can be developed for applications with little power, such as IOT. Then their data will be maintained till the power is switched off or the data status is set to cell location. SRAM is the easiest to utilize of all semiconductor memory [32].

2.3.1 **Why SRAM?**

The origination of the concept of the Metal–Oxide–Semiconductor Field-Effect Transistor (MOSFET) based memory was first commercialized and perfected in the seventies. Robert Den nard of IBM investigated the dynamic memory cell using a single MOSFET and a capacitor in 1968 [33] . The first MOSFET based on dynamic random access memory- chip with 2k-bits was developed in 1971 with several process improvements in leakage control. However, DRAM performance has not kept the pace with the performance of the processors from the very beginning due to long access time and more power weak. The dynamic nature of DRAM requires that the memory must be refreshed periodically so as not to lose the content of the memory cells [34].

The growing gap between the processors and the DRAM performance has dictated the need of different levels of memory hierarchy in the processor architectures. The on-chip cache memories are often called L1, L2 and even L3. The different levels of cache memories are SRAMs and they dominate the memory hierarchy in performance but they are often integrated in a lesser capacity due to area limitations and the high cost per bit [34].

SRAMs continue to be critical component across a wide range of microelectronics applications from consumer wireless to high performance server processors, multimedia and System On Chip (SoC) applications. Modern high performance processors and SoC application demands more on-chip memory to meet the performance and throughput requirements [35].

2.3.2 SRAM Architecture:

An SRAM consists of an array of bi-stable memory bit cells along with peripheral circuitries, such as address (row and column) decoders, sense amplifiers, write drivers and bit line pre-charge circuits etc. Peripheral circuitries enable reading from and writing into the array. A classic SRAM memory architecture is shown in Figure. 2.2. The memory array consists of 2^n words of 2^m bits each. An SRAM array is composed of millions of identical bit cells [35].

A memory bit cell is a circuit capable of storing a single bit of information "1" or "0". They share a common word line (WL) in each row and a bit line pairs (BL, complement of BL) in each column of an SRAM array. The dimensions of each SRAM array are limited by its electrical characteristics such as capacitances and resistances of the bit lines and word lines that used to access bit cells at uniform delay in the array. Memory arrays are organized such that the horizontal and vertical dimensions are of the same order of magnitude. Therefore, large size memories may be folded into multiple blocks with limited number of rows and columns. After folding, in order to meet the bit and word line capacitance requirement each row of the memory contains 2^k words, so the array is physically organized as 2^n rows and 2^m columns. Every bit cell can be randomly addressed by selecting the appropriate word line (WL) and bit line pairs (BL, complement of BL), respectively, activated by the row and the column decoders [35].

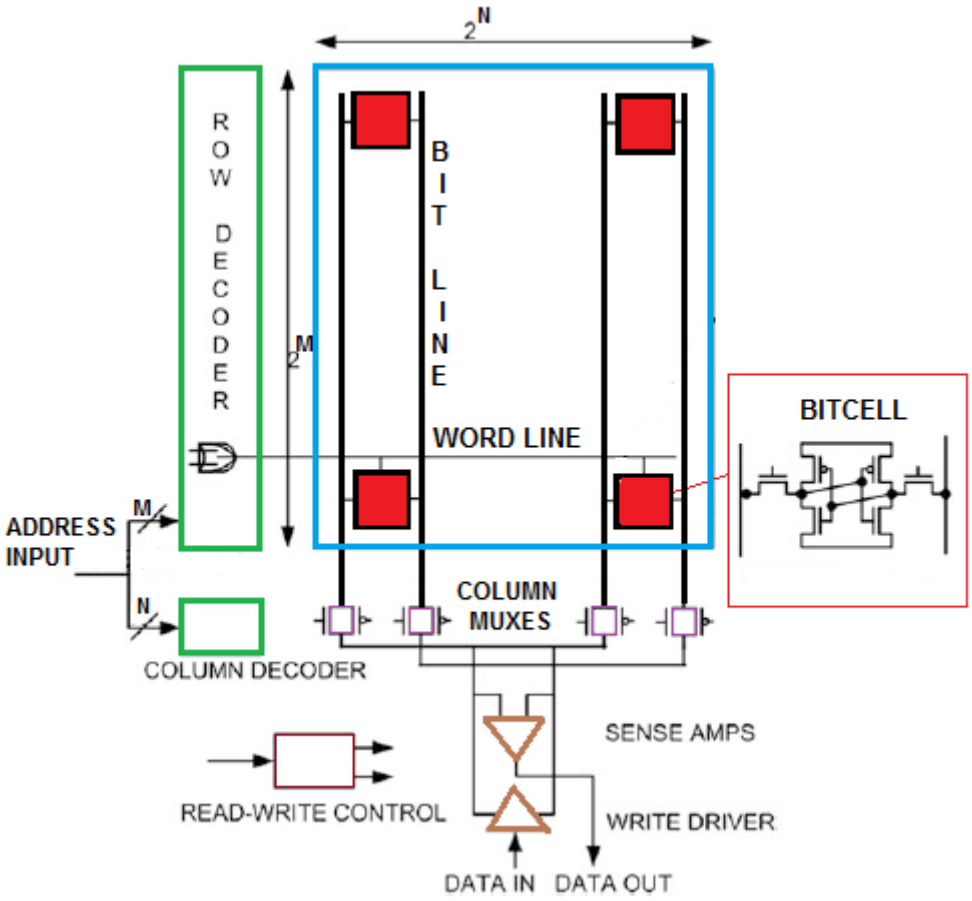


Figure 2.2: General SRAM array architecture [35]

2.3.3 SRAM Bit cell:

A SRAM bit cell, as depicted in Figure 2.2 is the fundamental building component of the SRAM array. Every piece of cell circuit can store a single bit of information. It offers a safe reading, writing and data storage operation, when the SRAM bit cell is powered up. SRAM Bit cell has 6T transistor which involves of four MOS transistors (NMOS) and two p MOS transistors (PMOS) as shown in Figure 2.3 At first the series PMOS and NMOS consists of a CMOS inverter (notice, in Figure 2.4a and Figure 2.4b, that a Complementary Metal Oxide Semiconductor (CMOS) circuit is displayed), have 6T SRAM cell [36].

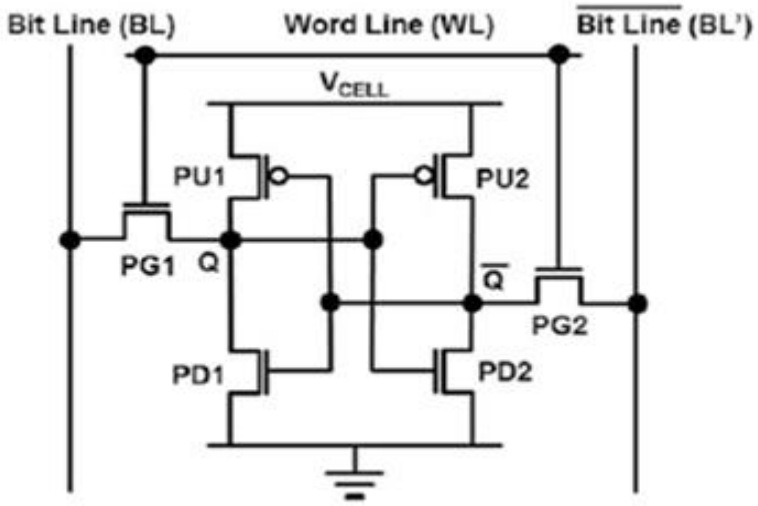
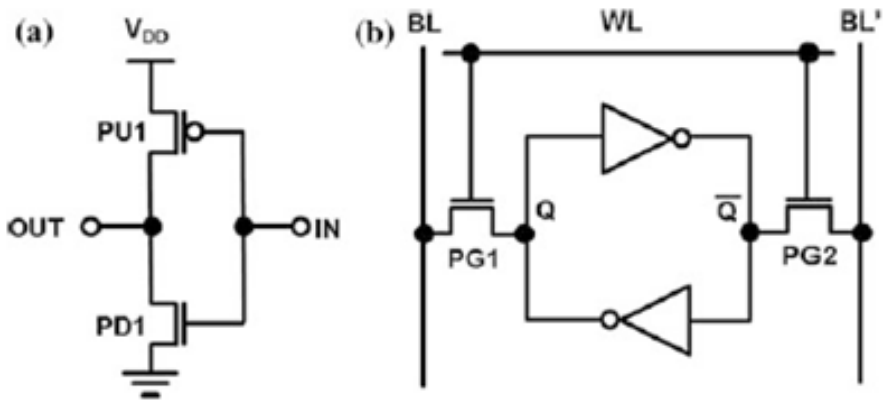


Figure 2.3: Circuit diagram of 6T SRAM [36]

For analyzing the operation of the SRAM bit cell in more detail, the basic operations of the SRAM cell can be divided into three: The first operation is the “read operation.” In the read operation, the data stored in the SRAM bit cell is accessed through the bit lines. However, the data stored in the SRAM bit cell must not be disturbed (i.e., the data stored in SRAM bit cell cannot be changed from 0 to 1 during the read operation). If this data is disturbed during the read operation, it is called a destructive read operation [36].

The second is the "writing process." During the written process, the data in the bit cell is reversed (or altered) to a new value from the previous value. It is vital that the data-fluting procedure is quick and dependable in this operation. "Data retention procedure" is the last operation. As a pair of transistors in the SRAM bit (i.e., pass-gate transistors (PG1) and PG2) divide the bit lines (e.g., bit line (BL) &bit line bar (BL')) from the SRAM, the SRAM bit cell operates as a coupling inverter latch [37].



(a) **(b)**

Figure 2.4: inverter circuit CMOS and" (b) "The 6T SRAM cell [37]

2.3.4 SRAM Bit cell Read Operation :

The 6T SRAM cell's reading and writing processes are nearly identical. BL and BL' are loaded first [38] . The gates are then triggered through the word line by providing voltages to the electrodes of the gate passing by (WL). The V_{CELL} line constant lies the linked inverter lock (where, usually, V_{DD} is biased). The biasing conditions on the bit lines fluctuate greatly between read and write operations [37].

The bit lines (BL and BL') of a 6T SRAM cell are pre-charged to V_{DD} (or V_{HI}), which is considered as the high-power or '1' bit, in order to enter data saved in the cross-coupled latch without reversing the recorded information. In the SRAM circuit, the parasitic capacitance is not taken into consideration. Generally speaking, the parasite capacitance of the bit line may be broken into three categories of parasites: (i) the bit line capacitance of the metal-wire, (ii) the overlapping capacitance of the gate-to-drain, and (iii) the source and pass gate transistor drain capability [39]. The metal-wire capacitance of the bit lines is a combination of (1) the capacitances between the BL and BL' metal wires, (2) the capacitance between the bit line and the other bit lines of neighboring SRAM cells, and (3) the capacitance between the bit line and the ground line (or GND, in short) [40].

the 6T SRAM cell is turning the pass gates on, by applying a voltage through the word line. Consider the case where, the bit ‘0’ is stored in a 6T SRAM bit cell.

Then, the voltage at nodes Q and $\overline{Q}$ is V_{DD} and GND, respectively. Right after the bit lines are pre-charged and the pass gates are turned on, the voltage at each node will be as shown in Fig. 2.5. Since the voltage at the node Q is GND, the PD2 is turned off. Similarly,

the PU1 is turned off because the voltage at node $\overline{Q}$ is V_{DD} . Then, we can figure out that the current will flow through BL–PG1–PD1, and thereby, the pre-charged BL will be discharged (shown as a green line in Fig. 2.5). At last, the voltage difference between BL and BL' is detected by a sense amplifier located between the bit lines. As a result, we can read the 6T SRAM bit cell data as '0' because the voltage at BL is discharged while BL' remains unchanged. Note that there is no current flow through PG2 while PG2 is turned on. This is because the voltage difference between the drain and the source of PG2 is zero (shown in Fig. 2.5) [37].

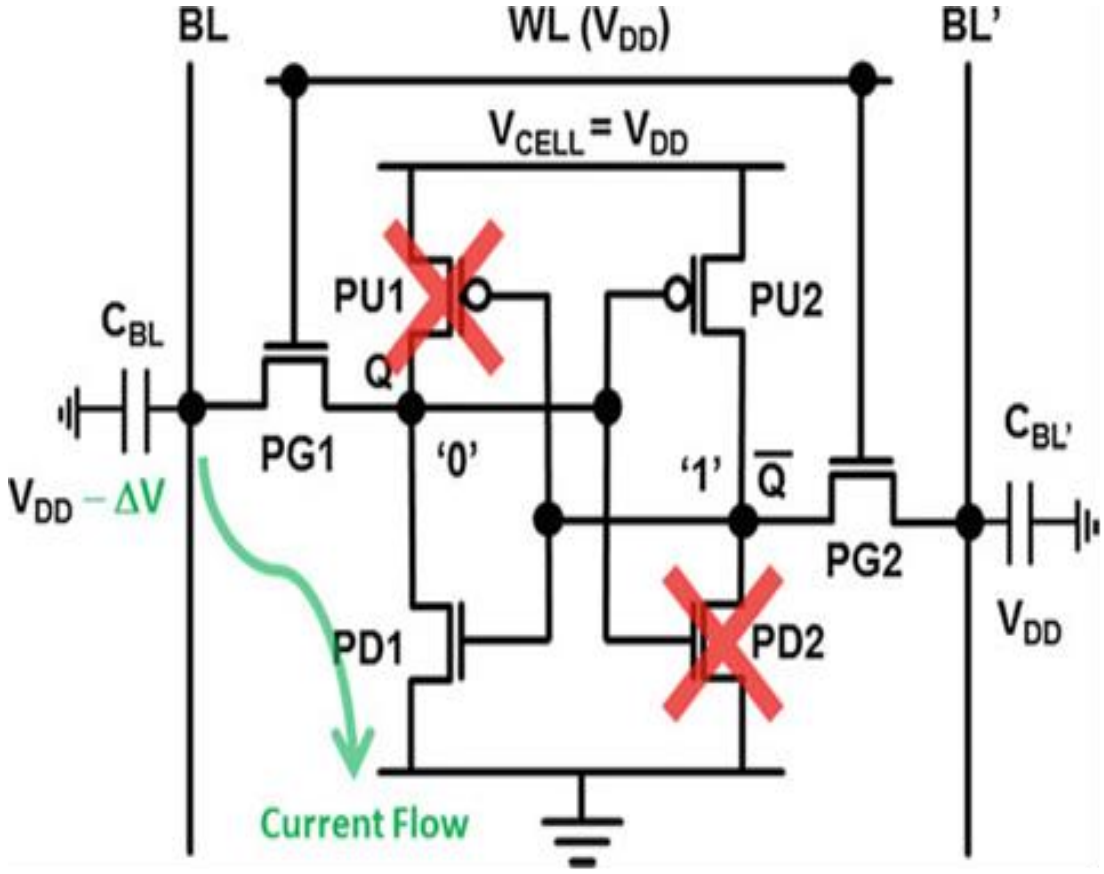


Figure 2.5: At each node during the read process, current flow and voltage levels [37]

BL is known when data from the 6T SRAM is downloaded. As a result of the current flowing via BL–PG1–PD1, the voltage at node *Q* is significantly raised.

When the voltage in node *Q* is sufficient, the PU2 may be switched on and off in response to the data in the 6T SRAM bit cell .Thus, PD1 has to be stronger than PG1. The readings must thus be somewhat closer to PD1 than to PG1.

The steadiness of a read process is therefore a struggle among the 6T SRAM. The ratio of the read steadiness can displaying related to 6T SRAM. The PD: PG ratio, which is called beta (β) ratio [41], is defined. If PD and PG have similar mobility, β can be simplified to PG [37] by its β ratio to its physical dimension ratio (i.e. to W_{CH} / L_{CH}).

The read method is shown in Figure 2.6 when data in a 6T SRAM are at 1 at *Q* node. It is about the same if the data stored at *Q* is '0.' The reader may see that it is the same as Figure 2.6 just by horizontally twisting Figure 2.5. Therefore if the data is '1' within a 6T SRAM cell, the data kept in a 6T SRAM may be represented as the reading procedure when it is '0' [38]. The reading operation can explain this.

The channel width of flat bulk transistors of CMOS may be changed without many limitations. However, channel width cannot be adjusted as quickly as the flat mass CMOS transistors for Fin FET . In order to retain its superior electrostatically and planar resistance, the width and height of the fine are limited by the look ratio that must be observed [42].

Furthermore, the Fin FET channel width is measured. Therefore, the transistor channel width of a Fin FET-based 6T SRAM cell can only be modified by the number of fins [37].

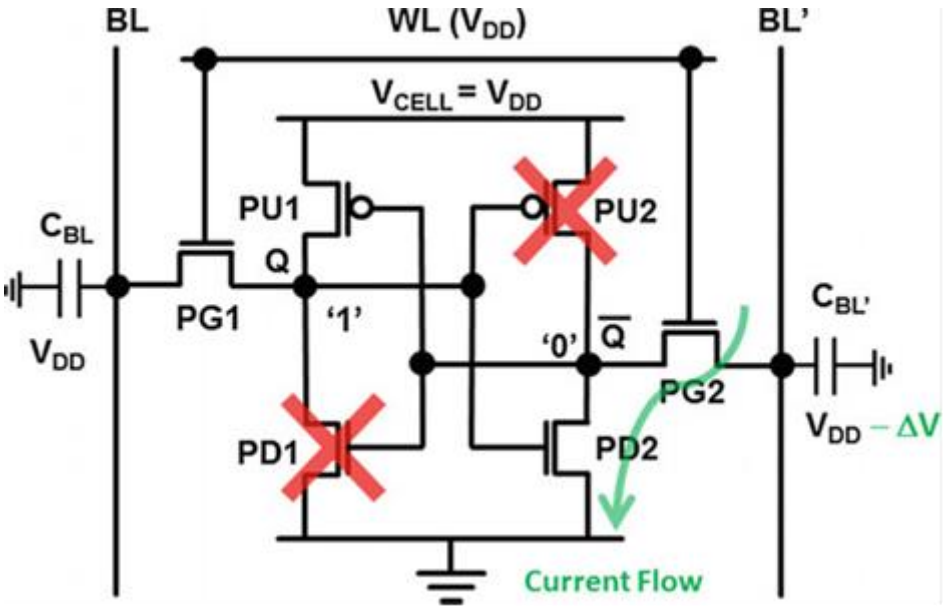


Figure 2.6: Reading from SRAM when BL=(i.e., = 1) [38]

2.3.5 SRAM Bit cell write Operation

The write operation can be classified into four distinct categories:

Case one: 0 is saved and writing 0 is asked,

Case two: 0 is saved and writing 1 is asked,

Case three: 1 is saved and writing 0 is asked,

Case four: 1 is saved and writing 1 is asked.

Cases one and four are inconsequential since they involve the overwriting of previously recorded data with identical data. As a result, only Cases 2 and 3 will be examined [36].

To read data from a 6T SRAM bit cell, a charge is applied to both the BL and the BL' . The pre-charged voltage levels on the BL and must be reversed, however, in order to put data into the SRAM cell. The BL is driven from V_{DD} to GND while pre-charged to V_{DD} in order to write 0 at the node *Q* in the SRAM cell's node. In contrast, BL is pre-charged and driven from V_{DD} to GND to write 1 at the node *Q* in the SRAM cell's node.

Both PG1 and PG2 are switched on by providing power to their gate electrodes through the word line after the BL and BL' have been adjusted to the proper voltage levels. Figures 2.7 and 2.8 for Case 2 and Case 3 illustrate the voltages at each data storage node in the SRAM cell ,respectively[37].

As soon as the power is turned on for the two PGs (PG1 and PG2), current flows through BL–PG1, PD1 and PU2–PG2 in Case 2 (i.e. when zero is in storage and writing a one is requested) (as shown in Figure 2.7). This results in a drop in voltage at the node $\overline{Q}$ followed by an increase in voltage at the node *Q*. Nodes *Q* and PD1 and PU1 are turned off and on until the voltage at the node lowers to a level low enough to turn PU1 on, or it rises enough to turn PU2 off. What we have here is a "flip," in which the voltage levels at the nodes of the circuit have been switched from *Q* to V_{DD} and GND, respectively [37].

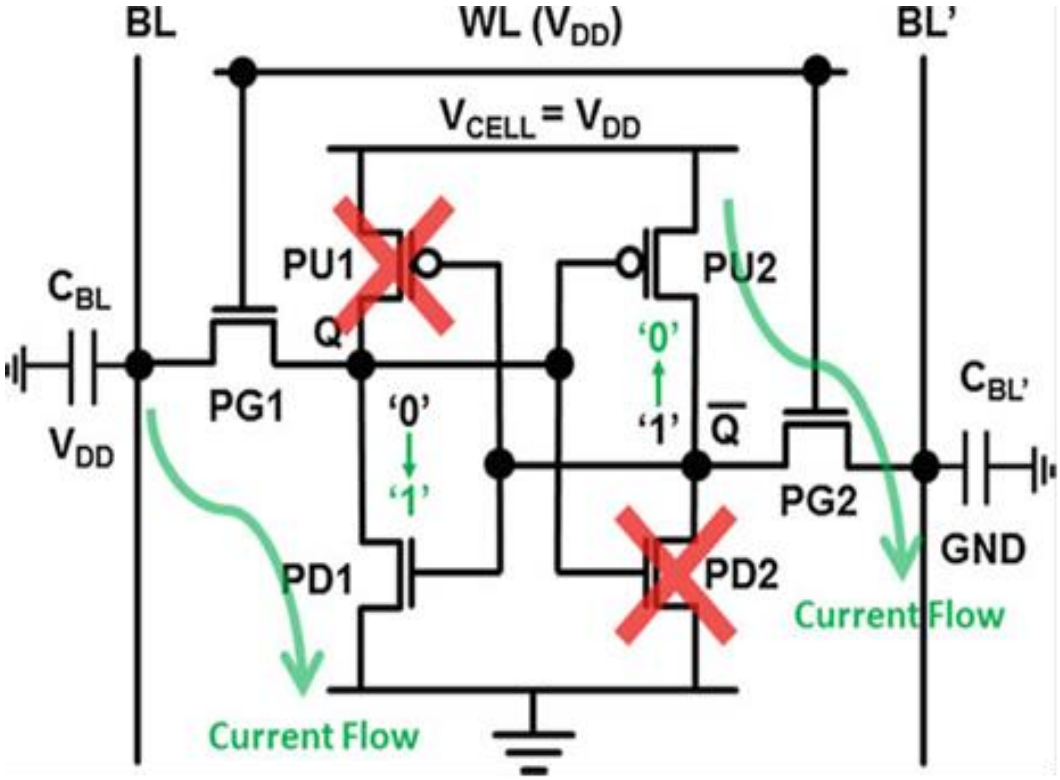


Figure 2.7: Voltage level and Current flow at individually data storing node (case 2) [37]

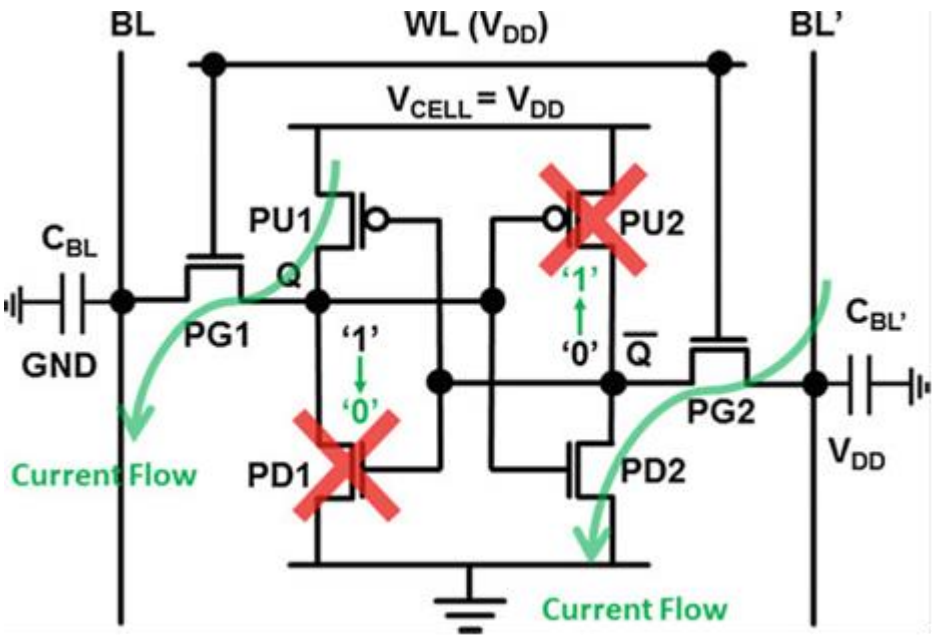


Figure 2.8: Voltage level and Current flow at each storing node (case 3)[37]

It is possible to regard Case 3 to be the same as Case 2 (that is, storing '1' and requesting '0'). After PG1 and PG2, the paths PU1–PG1–BL and BL–PG2–PD2 are activated. Current flows are enabled, because V_{DD} is preloaded and BL determined from V_{dd} to GND (as shown in Figure 2.8). Thus the *Q* node voltage falls, the node $\overline{Q}$ voltage waxes till that get the *Q* node voltage small enough to switch on PU2 and off PD2 and/ or (ii) the node $\overline{Q}$ tension is sufficient to switch off and on PD1 PU1 . Consequently, the node voltage $\overline{Q}$ and *Q* are turned to V_{DD} and GND [37] correspondingly.

2.3.6 SRAM Address Decoders:

One common method is to assemble linear memory words to build an N-word memory with a bits of M of each word. Each word is picked for reading or writing using N selected lines for N separate places. However, this strategy seems easy and works well for little memories, but it becomes difficult when the number of memories is enormous. Therefore, if it is structured in a linear way, a great number of lines or signals (~1 million) is needed to access that word-based memory. This (linear) method therefore results to insurmountable cables (interconnections) and packaging needs. An address decoder is introduced to decrease the number of lines or otherwise the number of interconnections. Address decoder allows the

number of select lines in the SRAM to be reduced by a factor of $log2N$, where N is the number of independent locations[35].

In SRAM there are two decoders: row decoder and column decoder . SRAM's performance and power consumption is driven by the design of these decoders. Row decoders must pick word line rows in the array by address bits from a number of rows.

In the selected line bit line, the pairs will be chosen from the column decoder's bit line pairs. Using AND/NAND and OR/NOR Gates you may build a quick decoder. The decoders can be statically and dynamically constructed in two distinct methods. The selection of design models depends on the performance, power and architecture area of SRAM. Due to its low power usage during decoded line changes, the static AND type structure may be employed [35].

For large SRAM arrays, the whole address space is A0, A1. Row decoder must be used with a decoder of 10 bit row and a column of 10 bit. A 10-input AND gate is required per row for row decoder implementation.

This presents many problems such as big fan in which performance, power dissipation etc. is negatively influenced. The division of a big gate into tiny logic levels hence generates faster, more efficient and less expensive implementation. Single-stage row decoders, on the other hand, are the best answer for small single-block memories. For multi-phase decoder designs, two broad kinds have been identified as show in figure 92.9)and (2.10) : the Divided Word Line (DWL) [43] and the Hierarchical Word Decoding (HWD) [44].

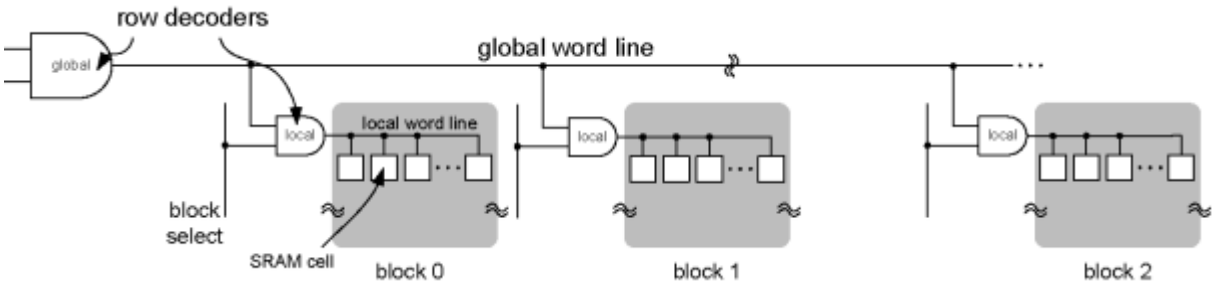


Figure 2.9: Figure 2.9 Divided word line row decoder [43]

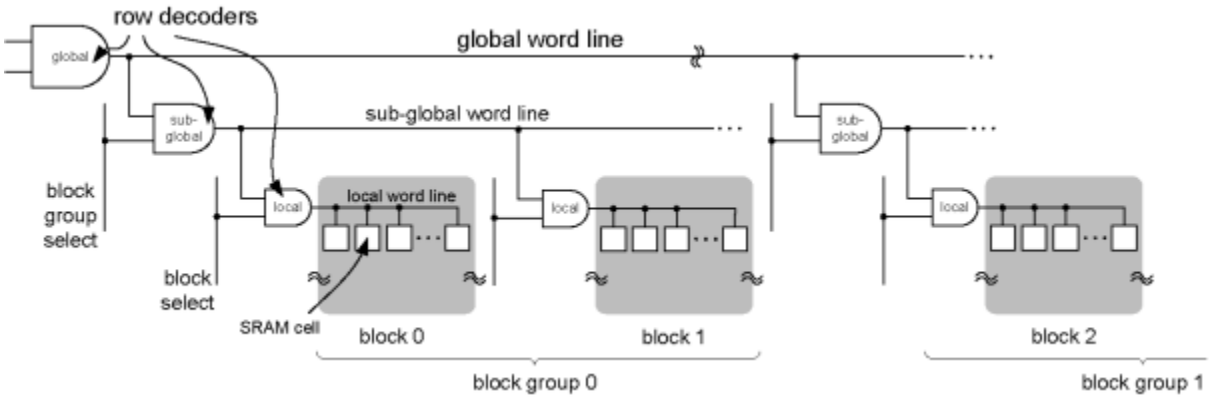


Figure 2.10: Figure 2.10"Hierarchical Word Decoding" (DWD) scheme[44]

Figure 2.9 illustrates the DWL structure in which the SRAM is broken into blocks. When both the global word line and the choose block are inserted to read or write a block, the local word line is activated to read or write the block. Since a read or write operation is active just one block at a time, the DWL structure decrease word line delay and power consumption. In order to meet growing delays and electricity consumption, the hierarchical text line decoding structure was proposed for high density and large SRAMs above 4 MB, as shown in Figure 2.10 [35].

2.4 An Overview of Testing

The product's life cycle consists of three phases in which testing is crucial. First, the design test aims to discover and identify design flaws during the design process to assure that the produced product performs its intended functions properly. The testing in the form of design verification. Next, in the production phase, testing will aim at identifying any production flaws that prohibit the supply of the quality and performance to the final system it is intended for (installed circuitry, impressed circuit board, etc.). Lastly, testing aims to identify any defects (functional defects) that occurred during operation and which would cause the system to be operated incorrectly [45].

Various test challenges must be solved during product design and development, but the ultimate aim is to accomplish cost-effective quality testing. This objective has gained increasingly challenging when accomplished on Very Large Scale Integrated (VLSI)

components has increased, such as integrated Random Access Memory (RAMs), so that several companies have stated that testing costs typically account for up to 55% of the product's total cost [46].

2.4.1 Kinds of Tests

Depending on the device being tested, the equipment testing it and the purpose of test, various types of testing are performed. Below, some commonly used terminologies are discussed as follows [47].

1- External testing: An external instrument might include a chip, a board or a computer they need for device testing .

2- Internal Testing or (DFT): The tester is placed in the same packaging as the unit for a device called the Design For Testing (DFT). The hardware tester is typically integrated with and on the same chip as the device being tested in the case of a Built-in Self-Test (BIST).

3- Online testing: Testing takes place during regular activities of a gadget.

4- Offline testing: The test device must stop to operate normally, and then it must be tested. Internal or external test gear can do offline testing.

5- Competitive testing (Online): Competitive testing is carried out when normal data is used by the gadget to fulfill its usual tasks.

6- Competitive testing (Automatic Test Equipment (ATE): The concomitant testing is conducted by the Automatic Test Equipment nomenclature when a tester simultaneously tests different sections of the chip. For example, when the device is in the testing head, memory and logic components are checked simultaneously.

7- DC Testing: The device is tested far more slowly than its frequency of operation. This permits all events to spread prior to sampling outputs.

8- In-Circuit Testing: The testing device is not removed for testing from its assembly position.

9- Speed testing: equipment that was examined at a constant rate of operation. The AC Testing is also known.

10- Guided test: Testing is carried out to discover the source of an error in the circuit outputs in a reverse testing procedure from outputs to inputs.

11- Diagnosis: When tested to detect the reason of failure, diagnosis is carried out.

2.4.2 Testing Difficulty

Testing process has become an extra complicated problematic in the last two to three periods. Although the number of I/O pins has grown for several VLSI procedures, the numeral of transistors in several VLSI procedures has improved. Moreover, a Circuit Under Test (CUT) is viewed as a black box by most operative test methods, the only nodes that can be checked by the testers are their major inputs and one is the primary output. To overcome this testing challenge, digital circuits need to be tested extensively by combining test design techniques [45].

To this end, creators must take part in the test process through the incorporation of testing hardware in their designs and the assessment of testability designs. DFT methods provide for more controlled and simpler observation of the internal structure of a design.

2.4.3 Design for Testability (DFT)

Design For Testability (DFT) methods aim to increase circuit's testability by incorporating extra circuits to improve the CUT control and observability. Whilst the extra DFT circuitry might increase design time, it typically decreases development time. Moreover, certain DFT methods decrease test creation time by Automatically Test Patterns Generating (ATPG) for all detectable circuit errors, with over 100 percent fault analysis [48]. Other DFT techniques (especially BIST approaches) allow for the construction of internal test patterns and the compression of output answers inside the CUT, which reduces the amount of further check time required after the first check [49].

2.5 Built-in Self-test" (BIST)

BIST represents a DFT method where tests are performed using integrated hardware [48]. The main concept is to have a testing VLSI chip. In addition to the CUT, the conventional BIST design includes, as illustrated in Figure 2.11, Four hardware modules.

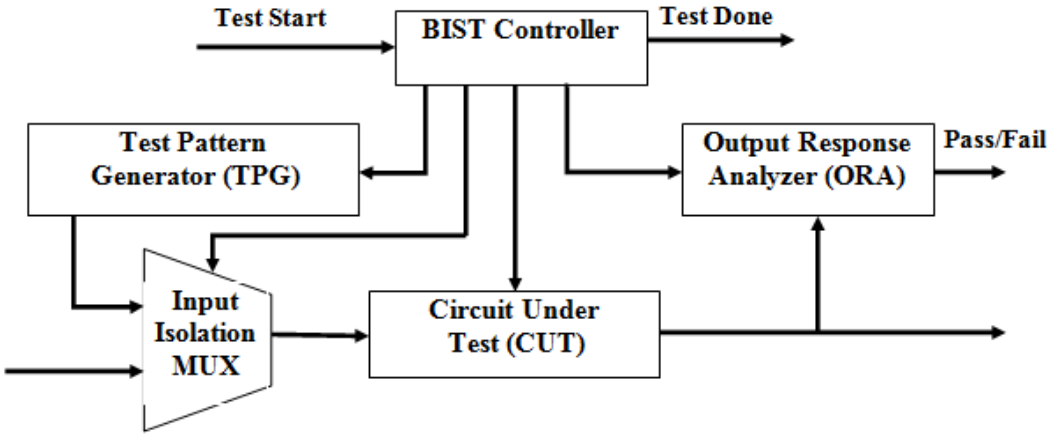


Figure 2.11: BIST Architecture

The Test Pattern Generator (TPG) provides the CUT test pattern. In order to establish the validity of the CUT, the Output Response Analyzer (ORA) compared or evaluates the exam answers. The input isolation multiplexer used for isolating system CUT input from testing input, by switching between normal and testing mode. The BIST controller is the main unit for all BIST activities, which includes the BIST sequence's initiation and duration. BIST controls at each circuit hierarchy, such as modules, chips, boards and system levels, are available in a BIST system hierarchy.

The control of the BIST lower level BIST activities, and the reporting of the test findings to the higher level is the responsibility of each BIST-controller. The design of a TPG is based on the test strategy. A fault coverage, overhead testing and test times determines the test technique chosen [45].

2.5.1 Test Pattern Generator (TPG)

Multiple types of test patterns exist. Sometimes TPGs are categorized by their class of test patterns [35]. The following classes are:

1- Deterministic test patterns: particular errors and/or structural defects for a certain CUT are designed for detection. For example, a ROM with a counter might have a device that applies deterministic vectors. The application of this sort of technique for BIST is restricted. In BIST applications, this method is commonly called "stored test patterns."

2- Algorithmic testing patterns : It is similar to deterministic patterns in that they are specific to a given CUT and are developed to detect specific fault models in the CUT. However,

because of the repetition and/or sequence typically associated with algorithmic test patterns, the hardware for generating algorithmic vectors is usually a Finite State Machine (FSM) or a microprocessor (Micro code), and usually used March algorithms. This test pattern generation approach is specified for Memory Built-in Self-test (MBIST)

3- Exhaustive test patterns*:* It generates input test patterns using a counter-based method. With an N-input combinational logic circuit, it uses an N-bit counter to produce all possible test patterns and identifies gate-level stuck-at faults, wired AND/OR and dominant bridging faults without fault simulation.

4- Pseudo-exhaustive test patterns : represent a substitution to exhaustive test patterns. Each partitioned combinational logic sub circuit will be exhaustively evaluated in this example; each K-input sub circuit will be exposed to all conceivable patterns, where K < N, while the sub circuit's outputs are monitored during the testing process[50]. Counters and Linear Feedback Shift Registers (LFSRs) are examples of hardware that might be used to generate this sort of test pattern for BIST.

5- Pseudo-random test patterns*:* are the most common TPG hardware-generated BIST patterns. These test patterns are mostly generated by LFSRs.

6- Weighted pseudo-random test patterns : This is advantageous for circuits that are resistant to random design. To generate pseudo-random test patterns, this method begins with an LFSR and then filters the patterns in the test patterns utilized in the CUT using AND/NAND and OR/NOR gates.

7- Random testing patterns: were often utilized both for external microprocessor functional testing and for ATPG software [51]. However, the group of highly unplanned patterns for the BIST application is restricted, because such patterns cannot be reproduced and the error coverage gained would thus change from a BIST sequence performance to the next one.

2.5.2 Output Response Analyzer (ORA)

The findings from the CUT will be 'compacted' in most ORA procedures to 'Signatures,' in comparison with the anticipated error free circuit signature. Instead of a single pass/fail bit,

the signal may contain a BIST signature sequence of data bits. It is used because compression does not mean loss of data, while most ORA systems cause some loss of data. There are

numerous types of ORA circuits dependent on how the output data is compressed [45]. These classes:

Concentration: Data compression is normally done in a single data stream from multiple CUT outputs. Concentration: A classic concentrating apparatus is a tree of exclusive OR doors coupled to a parity circuit .

Comparative: ORAs utilize a comparator to identify error-free and defective circuit error. While this specific BIST method is not generally employed because of its wide overhead area,

Counting systems: Count the number of nulls or 1s in the CUT output response at the completion of the test sequences and sign a signature with the resultant count of the associated characteristics for the CUT Pass/Feeling indication.

Signature investigation: this is the most often employed BIST implementation approach for ORAs [48]. The LFSR is the main component of the ORA program used for signature analysis. Divide the CUT polynomial by a distinctive LFSR polynomial for the implementation of ORA is the basic principle underlying the signature analysis.

Accumulators: Use the binary magnitude of each answer to summarize the CUT output replies.

Parity control: for the output response, compression methods can also be utilized. The principal notion is to compute parity and control parity at the conclusion of the BIST method for the output response data.

2.5.3 BIST Advantages and Disadvantages

BIST enables a circuit to test itself, so there are many advantages of BIST include:

1. Low cost: In comparison to external testing utilizing expensive automated test equipment, it is more efficient and cost effective (ATE).

2. Vertical testing: may be used at all test levels simply. All production tests and system level tests on the ground may be carried out using the same test technique in wafer and device-level tests (BIST).

3. High diagnostic resolution: BIST error detection not only shows that there is a problem, but also that the VLSI device is faulty.

4. Minimize test time: all BIST testing is carried out at the frequency of the system clock. This is usually known as at-risk tests and is significant since it allows for errors to be detected that contribute to undue latency in a CUT that works otherwise. This means that the testing takes place at its highest rate, which reduces the testing period for the gadget.

Disadvantages of implementing BIST include:

1. Further time and effort to design: two systems have to be created, one BIST system at the top and the two systems have to work for the project's success.

2. Overhead area: BIST circuit boosts the total device area.

3. Strom discharge: BIST circuitries that increase power dissipation due to excessive BIST activity may be troublesome in certain system applications, like as power-consumption applications or temperature restrictions.

2.6 SRAM Fault Models

SRAM is a sequential logic of many storage cells, so it is impractical testing a large sequential logic using regular sequential tests, because it takes a very long time. It cannot be done by overhead hardware to use sequential SRAM in conjunction with combinational testing techniques. However, they have the same fundamental criteria as structural and functional fault model for reading or writing specific areas.

2.6.1 SRAM Structural Fault Model

The structural defect model of SRAM (Figure 2.12) is a storage array made up of n/m or m cells . The Memory Hardware is a series of decoding, read and write logic cells, decoders and column addresses, and a logic block to manage the input and output data .

The actual configuration of memory cells in the array depends on a chip and not on a word or space. Testing this structural defect model one by one would take great resources, like it is carried out in logic testing, given the numbers of cells and the potential structural defects each cell may have. The SRAM solution consists of utilizing a functional defect model, rather than a structural one, and simultaneously evaluating defects in the surrounding circuitry and the memory array [47].

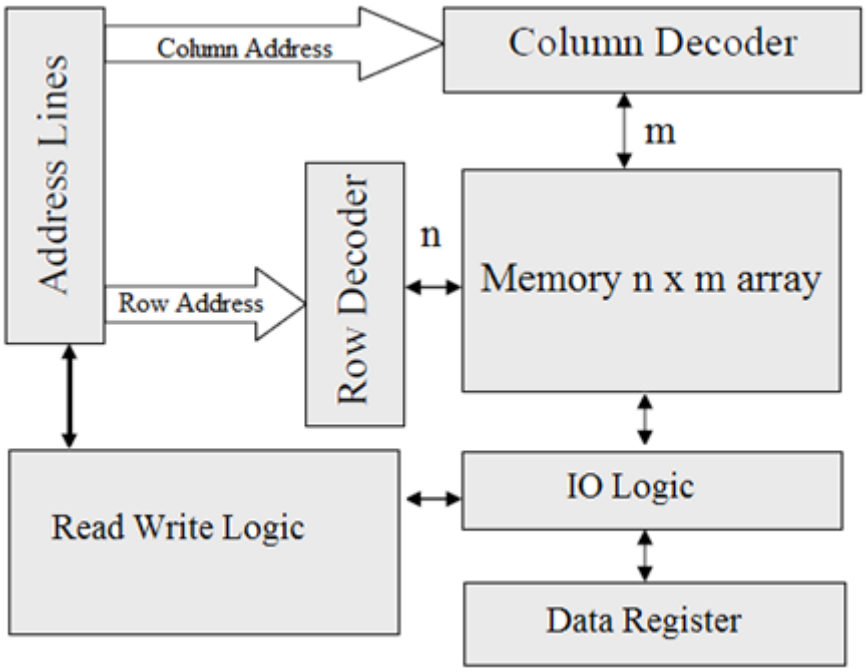


Figure 2.12: SRAM structural defect model [47]

2.6.2 SRAM Functional Fault Model

The functional fault model for SRAM depicts the defects that arise when reading and writing memory. The classical and non-classical functional failure models are two main features connected with SRAM. While the conventional functional error models are due to the memory cell structure, the dense packaging of the cells in the array will result in non-classical functional defect modeling. Classical models of functional defects include [52]:

1- Stuck-At Faults (SAF): result when the cell contents are stuck-at-0 or stuck-at-1. To detect these faults, one must test each cell with both logic 0 and logic 1 values.

2- Address Decoder Faults (AF): outcome if owing to a failure in address decoder logic, an incorrect address will be picked. All addresses have to be tested using unique data to detect these defects.

3- Data line faults : occur when the registers of data input and output (or bit and row when the bit row is supplemented) contain defects that do not allow proper data to be entered or read from the cells inside the array. The logic 0 and logic 1 value must be transmitted via each input and output of data to identify these errors.

4- Read/write faults: they occur from read/write check lines or read/write check logic flaws which prohibit cell array read/write operations. So, must write and read all cells to discover these defects.

Non-classical functional fault models include [53]:

1- Transition Faults (TF): These fault happen as a result of a cell being unable to undergo a transition or being unable to undergo a transition. To find these problems, both transitions must be tested.

2- Data retention faults: These faults occur as a consequence of which a cell's contents are lost after a particular amount of time. To discover these errors, data must be read following a time of inactivity (no read or write operation).

3- Destructive read faults: These faults cause a read operation to be performed, which changes the contents of a cell These kind of defects are referred to as read disturb faults in some circles. It is necessary to make numerous readings of the cells in the array after data has been written in order to discover these errors. Reads for both logic 0 and logic 1 values should be carried out in the same manner.

4- Neighborhood Pattern Sensitive Faults (NPSF): the contents of the specific cell are impacted by other cell content. One example of pattern sensitivity is a bridging failure between the cells. If this failure is to be detected, the test cell must be circumferred in neighboring cells by special logic values.

5- Coupling fault (CF): It causes other cell activities to affect the contents of a specific cell. The key difference is that the defect in pattern of sensitivity depends on the static level of the adjacent cell, whereas connective defects are generated by transitions in distant cells by writing or reading. Clouding fails and pattern sensitivity faults look similar. There are several sorts of connection errors and the identification of them depends on the connection type [53].

2.7 March Algorithms

The impacts of a memory cell on each other may be seen through a functioning defect model test. However, every cell must be read and written to 0 and 1, in order to complete and exhaustively test an array, to check what they are affecting, while all other cells are being read. Obviously, it might lead to extended test periods if the method is exhaustively carried out . The time complexity of testing to $O(N)$, where N is the total number of memory bits available, may be reduced by using specific approximations, and the march algorithms are the most effective method of doing so.

2.7.1 Traditional March Algorithms

A March examination algorithm may be thought of as a series of March elements in sequence. In every memory cell, a mark element is comprised of an order of reading and/or

writing that is applied from cell 0 to cells N-1 or from cell N-1 to cell 0. The next March tests can be specified more accurately [54]:

• The March element is a finite series of operation applied for each cell in the memory in one of the 2 addresses commands, rising (↑) address command from 0 to N −1 address or lowering (↓) address command from N −1 address to 0, where each cell's operations have to be the similar for each cell.

• A test in March is the first in a limited sequence of tests in March.

According to the above definition, the way the test proceeds to the next cell is determined by the address order (increasing or decreasing). For some March elements, the address order can be chosen arbitrarily and this is indicated by the ( ) symbol. Moreover, the address order may be irrelevant. However, that the only real requirement for the March tests is that the address orders and must be inverses of each other. An operation applied to a cell can be “w0” (write 0), or “w1”, a “r0” (read 0), or a “r1”. The complexity of March algorithms testing is denoted by "N", which represents the number of operations on each address, rather than each bit, within the memory. Table 2.1 shows some classical March algorithms and its fault coverage [53].

Table 2.1: Some classical March algorithms and its fault coverage

March algorithm	Operations Sequence	N	Detected Faults			
			SAF	AF	TF	CF
Zero-One	$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(w1);\Updownarrow(r1)\}$	4N	+	-	+	-
MATS	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Updownarrow(r1)\}$	4N	+	+/-	-	
MATS+	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$	5N	+	+	-	-
MATS++	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)\}$	6N	+	+	+	-
March Y	$\{\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)\}$	8N	+	+	+	+/-
March C	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	11N	+	+	+	+
March C-	$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$	10N	+	+	+	+

2.7.2 Zero-One Algorithm

Zero-One algorithm described as described below:

$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(w1);\Updownarrow(r1)\}$

Each operation of write and read represents a full address sweep. Some refer to the pattern of this algorithm as the blanket pattern or as MSCAN. The " $\Updownarrow$ " denotes that the order is non-consequential and that no preference is given to the addressing order. Each address is accessed only four times and is referred to as a complexity of 4N algorithm, due to its four operations (two write and two read). If a memory was only sensitive to stuck-at faults, the Zero-One algorithm would be sufficient to catch all defects. With fault grading, the Zero-One pattern achieves 100% stuck-at Fault coverage in the memory cells [32].

2.7.3 March C Algorithm

This algorithm has 7 March elements and all of the order/direction oriented, unlike the Motorola Scalable Controller Area Network (MSCAN) algorithm.

$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$

1. $\Updownarrow(w0)$: Write 0s to all cells in any order.
2. $\Uparrow(r0,w1)$: Repeat this process until you reach the highest address (the expected read value is 0), and then read from the lowest address (the expected write value is 1) and write a 1.
3. $\Uparrow(r1,w0)$: Continue reading from the lowest address (where the expected read value is 1), writing a 0 at this location, and repeating the process until the highest address has been reached.
4. $\Updownarrow(r0)$: Read from any and all of the cells in any sequence (expected read value is 0).
5. $\Downarrow(r0,w1)$: Repeat this process until the lowest address is reached, starting with the highest address (anticipated read value is 0), writing a 1 at this address, and so on.
6. $\Downarrow(r1,w0)$: Repeat this process until you reach the lowest address (the anticipated read value is 1), then read from the highest address (the expected read value is 1), and so on.
7. $\Updownarrow(r0)$: Read from any and all of the cells in any sequence (expected read value is 0) .

Because it has 11 operations, two for each of the elements two, three, five, and six, and one for each of the elements one, four, and seven, the March C algorithm is deemed to have the

complexity of the 11N algorithm. The algorithm used in March C is not redundant (A redundancy is a repeat of an operation that does not allow any further faults to be detected). It encompasses a broad range of defects, including stuck-at faults, several Coupling Faults, Transition Faults, and a large number of Address Decoders

2.7.4 March C- Algorithm

The March C- algorithm is a slight reduction of the March C, which had inefficiency or redundancy that was eliminated by the March C algorithm. The March C- algorithm has the following March elements [47]:

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}.$$

The March C- algorithm is considered as a complexity of 10N algorithm, due to its ten operations, two for each of elements two through five and one for each of elements one and six.

The March C- algorithm is ostensibly stated as covering unlinked idempotent coupling faults. In reality it covers a host of faults including stuck-at faults, many coupling faults, transition faults, and many address decoder faults are also covered.

2.7.5 MATS Algorithm

The Modified Algorithmic Test Sequence (MATS) , is a 4N-complexity test sequence [55]. It is focused on finding stuck-at faults as well as detecting some address decoder faults. MATS has the same length as the Zero-One algorithm but is far superior. And it has the following March elements:

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Updownarrow(r1)\}.$$

2.7.6 MATS+ Algorithm

This algorithm is less complex than March C– and is considered optimal for unlinked stuck-at faults [56]. MATS+ is a 5N complexity and can be represented as following March elements:

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}.$$

2.7.7 MATS++ Algorithm

The MATS++ algorithm is an improved MATS+. It is a 6N complexity [53]. This technique identifies several types of address decoder failures, including stuck-at fault , transition fault , and coupling fault problems. It can be represented as following March elements:

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)\}.$$

2.7.8 MARCH Y Algorithm:

March Y tests connected transition and inversion coupling problems. Additionally, it identifies address decoder and stuck-at issues [32]. It is an 8N complexity and can be represented as following March elements:

$\{\Updownarrow(w0); \Uparrow(r0, w1, r1); \Downarrow(r1, w0, r0); \Updownarrow(r0)\}$.

2.8 Memory BIST (MBIST)

MBIST (as shown in Figure 2.13) is a special type of BIST, with following characteristics:

1. CUT: is an embedded memory under test (SRAM).

2. TPG: is an algorithmic test patterns generator (March algorithm).

3. ORA: is a comparator-based output response analyzer.

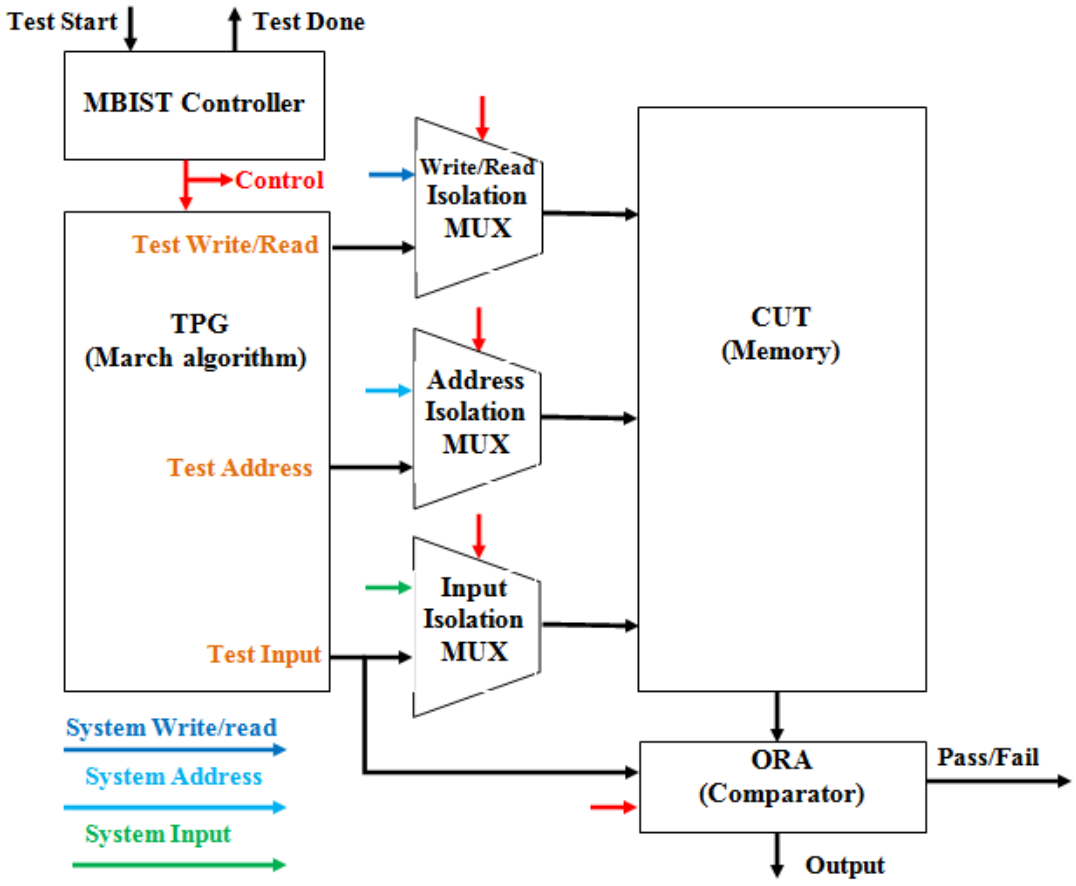


Figure 2.13: MBIST Architecture

2.9 Summary

Embedded memories occupy the largest part of the chip, and its capacity, density and complexity increases linearly throughout the year, also its defect types are becoming more complex and needing more complex and too costly Automatic Test Equipment (ATE). A promising solution to this dilemma is MBIST which adds low-cost test circuitry to the memory itself and provides an acceptable yield.

Semiconductor memories classified to Read–Write Memories (RWMs), which is most often called Random Access Memory (RAM), Read Only Memories (ROMs), and Nonvolatile Read–Write Memories (NVRWMs or NVRAMs). Of all semiconductor memories, the Static Random Access Memory (SRAM) is the faster and easiest to use, so it used across a wide range of microelectronics and System On Chip (SoC) applications. SRAM is a memory array consists of words of 6T bit cell each, with word line (WL) in each row and a bit line pairs

(BL) in each column. Each 6T bit cell consists of four n-type MOS (NMOS) transistors and two p-type MOS (PMOS) transistors, and its basic operation is read, write and data retention,

and it works just like as a cross-coupled inverter latch. SRAM address decoders consist of row and column decoders, with multi stage decoder structures which are classified into Divided Word line (DWL) and Hierarchical Word Decoding (DWD).

There are many types of Logic tests, like Internal Testing which uses a tester for a device called Design For Testability (DFT) like Built-in Self-test (BIST), and Concurrent Testing in case of ATE. The main problem for logic testing is limitation of controllability and observability for circuits under test (CUT), which overcame by incorporation of design for test (DFT) techniques. DFT seek to improve the testability of a circuit, and minimize test development time by providing automatic test pattern generation (ATPG) for all detectable faults. Main types of DFT approaches are adhoc, scan design and BIST. BIST is a DFT technique in which testing is accomplished through built-in hardware features and enable VLSI chip to tests itself. The typical BIST architecture is composed of Test Pattern Generator (TPG), Output Response Analyzer (ORA) and input isolation multiplexer. The main type of TPG that used in BIST is Algorithmic test patterns which usually use March algorithms and implemented by Finite State Machine (FSM) or a Micro code. The main type of OPA that used in BIST is Comparison-based and implemented by an RS latch. There are many advantages of BIST include: low-cost, vertical testability, High diagnostic resolution and

Minimize testing time, but there are some disadvantages include: Additional design time and effort, Area overhead and Power dissipation.

SRAM is a sequential logic of many storage cells, so it is impractical testing a large sequential logic using regular sequential tests. The solution is by using functional fault model which represents the faults that occur during memory write and read operations. SRAM functional fault models include: Stuck-At Faults (SAF), Address Decoder Faults (AF), Transition Faults (TF) and Coupling Faults (CF). The best way for achieving SRAM functional fault models tests is by using March algorithms, which is a sequence of read and/or write operations that are all applied to every cell in the memory. The classical March algorithms, include Zero-One, March C, March C- , MATS, MATS+, MATS++ and Mach Y, are used for detecting bit-oriented memories faults. MBIST is a special type of BIST by using

SRAM as CUT and March algorithm test pattern generator as TPG and comparator-based ORA.

Chapter 3

Chapter Three

3. The Proposed Methodology

3.1 Introduction

In this chapter, two new word-oriented Memory Built-In Self-Test (MBIST) algorithms (Fast March and FSM March) were developed from a traditional bit-oriented MBIST ZERO-ONE and March C- algorithms, by using a test patterns. VHDL is used to design and implement the proposed algorithms, which offer the flexibility in design and simulation and portability which enable to apply MBIST designs directly for testing FPGA (Field Programmable Gate-Array) based SoC (System-on- Chip) embedded memories. The Fast March algorithm is simple so it is designed on the basis of counter to reduce resources and make performance as fast as possible . On the other hand, the FSM March algorithm is complex, and the best way to design it is by using Finite State Machine approach, which represents complex operations as detailed in the state diagram which enable to programming it with VHDL language. The main advantage of the proposed algorithms (Fast March and FSM March) is it can be used to test an embedded memory with any size if its word length is one byte, by using a simple modification on address bus to fit a size of tested memory.

3.2 Word-oriented memories:

The previous algorithms (ZERO-ONE and March C-) could not be used to test word-oriented memories because they were designed primarily to test bit-oriented memories, and since each bit cannot be accessed individually in word-addressed memories. Therefore, it was necessary to find a way to reflect the interaction of each bit in the word with the rest of the word bits. The solution was by using a test patterns with the length of the memory word instead of (0 and 1), where the number of needed test patterns for memory testing determined by the law in Equation (1) where m is the length of the word. The word length is expressed in bits, if the word length is 8 bits, normal of test- patterns is 4 , and the inverse is 4, shown in Table 3.1.

$$log2(m+1) \text{ …….. } (1)$$

Table 3.1: Test pattern

number	Test Patterns	
	Normal	Inverse
0	00000000	11111111
1	01010101	10101010
2	00110011	11001100
3	00001111	11110000

3.3 Fast March Algorithm:

Fast March algorithm is a modified algorithm from the traditional ZERO-ONE algorithm, and it was designed to test word-oriented memories based on test patterns. It uses the stages of the ZERO-ONE algorithm after replacing '0' with normal test patterns and '1' with inverse test patterns and iterating according to the number of normal test patterns which based on the length of the memory word. If the length of the memory word is 8 bits, normal of test-patterns is 4, so the stages of the Fast March algorithm become as follows:

M0,1 : ⇕ (*w*00000000).
M0,2 : ⇕ (*r*00000000).
M0,3 : ⇕ (*w*11111111).
M0,4 : ⇕ (*r*11111111).
M1,1 : ⇕ (*w*01010101).
M1,2 : ⇕ (*r*01010101).
M1,3 : ⇕ (*w*10101010).
M1,4 : ⇕ (*r*10101010).
M2,1 : ⇕ (*w*00110011).
M2,2 : ⇕ (*r*00110011).
M2,3 : ⇕ (*w*11001100).
M2,4 : ⇕ (*r*11001100).
M3,1 : ⇕ (*w*00001111).
M3,2 : ⇕ (*r*00001111).
M3,3 : ⇕ (*w*11110000).
M3,4 : ⇕ (*r*11110000).

Where Mi,j means elements of algorithm , the table (3.2) below are the interpretation of this test. The list below or the above expression show that Fast March uses 16 read/write operations, and it therefore of 16N order.

Table 3.2: list of interpretation [53]

R	Memory Read operation
r0	Read a 0 from the memory
r1	Read a 1 from the memory
W	Memory Write Operation
w0	Write a 0 to the memory
w1	Write a 1 to the memory
⇑	Increasing memory address ordering
⇓	Decreasing memory address ordering
⇳	There is no difference between different addressing orders.

3.4 VHDL Design for Fast March Algorithm with 16x8 bit SRAM

The first proposed methodology is designed using VHDL . The design is based on Fast March algorithm with 16x8 bit tested memory as shown in Figure 3.1. It includes the following components:

1. MBIST_Controller: Built-in Self-test controller.

2. MARCH_Counter: Test counter.

3. Pattern_Decoder: Test patterns decoder.

4. MBIST_Comparator: Test comparator.

5. MBIST_Error_Counter: Fault counter.

6. ADDR_MUX: Addresses multiplexer.

7. DATA_MUX: Data multiplexer.

8. WR_MUX: Read or write enabling signal multiplexer.

9. SRAM16: Under Test Memory.

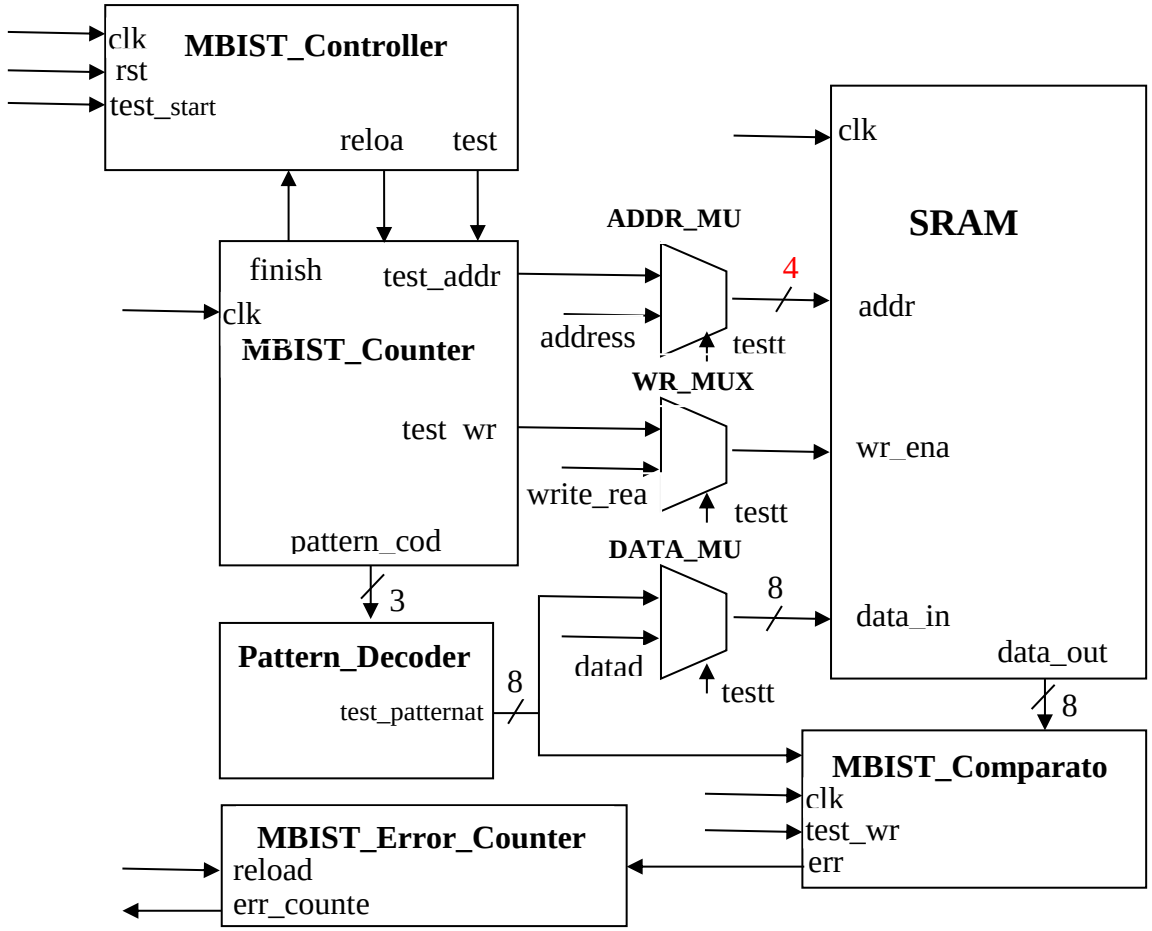


Figure 3.1: Block diagram of testing 16x8 bit SRAM using Fast March

3.4.1 MBIST_Controller

The test controller "MBIST_controller" function is to switch between the normal working and test mode of the memory, and to organize the memory self-test process. The test start signal "test_start" initiates the transition between the normal working and test mode of the memory, and the value of "test_start=0", the memory in the normal working mode and the test controller is idle (the "test" signal that go from the controller to the rest of the components is equal to "test=0"), and data is transmitted normally between memory and the environment without interference from the self-test circuit.

The test mode, as shown in Figure 3.2, starts when the test signal becomes "test_start=1", whereby the controller sends the test signal "test=1" to the test counter "MARCH_Counter" to start working, as well as to the multiplexers "ADDR_MUX", "DATA_MUX", "WR_MUX" to isolate the SRAM from the surrounding environment, and connecting it with the components of the Built-in Self-test circuit. It is possible at any moment to reset the test process via the controller input signal "rst=1", then the controller sends the reset signal "reload=1" to all components of the test circuit to return it to the initial state. When the test is finished, the test counter sends the finish signal "finish=1" to the test controller to stop the test process "test=0", and reset all components by "reload=1" signal, and disconnect the memory from the test circuit and connect it again with the surrounding environment.

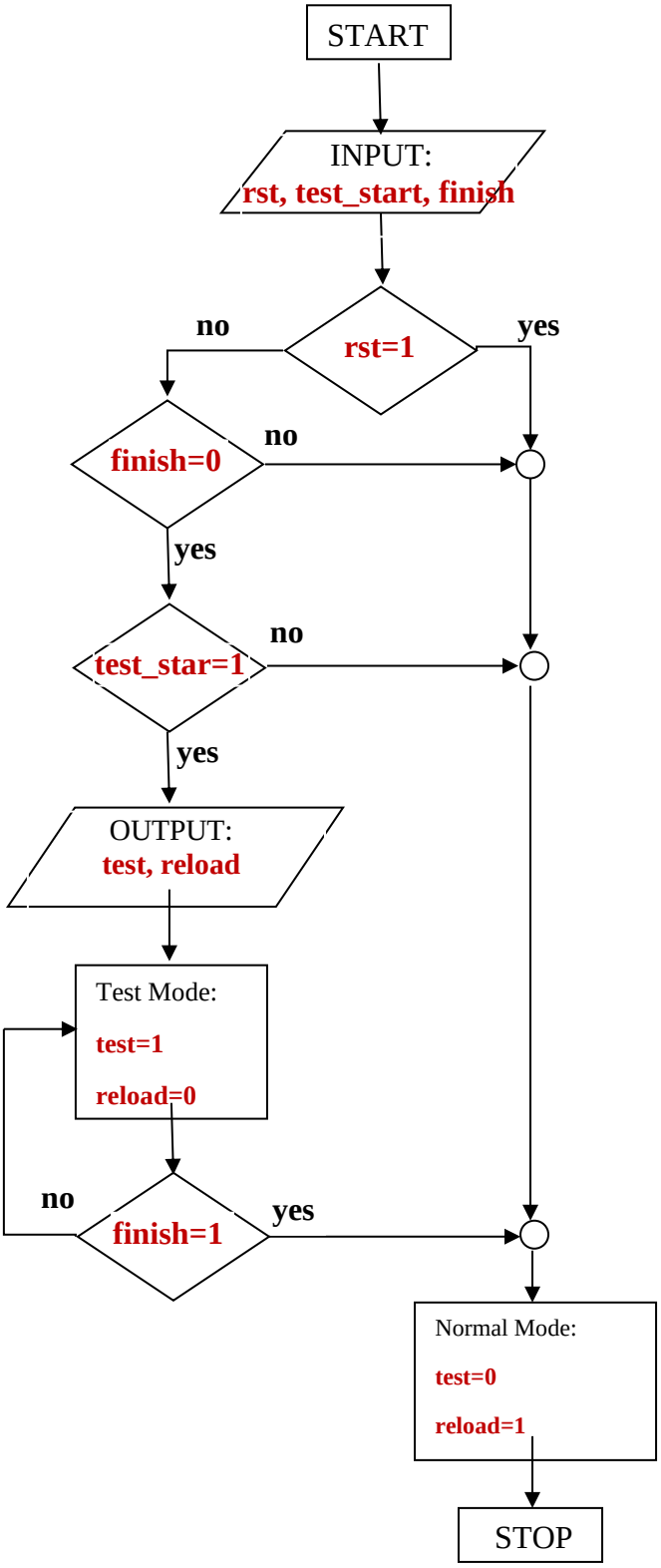


Figure 3.2: Flow chart for MBIST_Controller

3.4.2 MARCH_Counter

It is a test counter whose output length depends on the length of the memory word and the number of addresses. If the length of the memory word is 8 bits and the number of addresses is 16, then the length of the counter's output is 4+1+3+1 bits where bits (0...3)of the counter output are allocated to generate addresses "test_addr" (The number 16 needs 4 bits to be represented in binary). Bit (4) to create write/read enable signal "test_wr". Bits (5…7) to generate the "pattern_code" signal (we have 8 test samples that need 3 bits to be represented in binary) and bit (8) to generate a finish signal as in Figure 3.3.

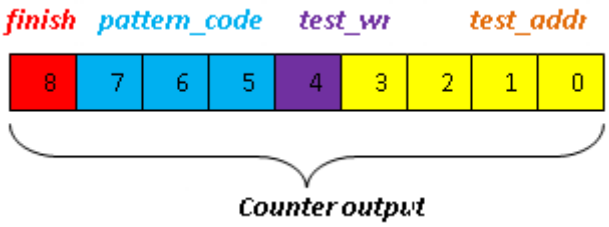


Figure 3.3: Counter output for 16x8 bit SRAM

The test counter implements the Fast March algorithm as shown in Figure 3.4, according to the following sequence:

1. The initial value of the counter output is "**000000000**" which means that primitive address "test_addr=0000", memory-write is enabled "test_wr=0", current test pattern code "pattern_code=000" and test is in progress "finish=0".

2. When the counter receives the test signal "test=1" from the MBIST_controller it starts counting and during that time the test pattern corresponding to the code "pattern_code=000" is written to memory "test_wr=0". This write process is repeated with each increment of the address "test_addr=test_addr+1" up to it reaches the last memory location "test_addr=1111", and the counter output becomes "**000001111**".

3. As the counter keeps counting, its output becomes "**000010000**" which means the address goes back to the start "test_addr=0000" and goes to the read from memory process "test_wr=1" and compares its output with the corresponding test pattern "pattern_code=000". The process of reading and comparing is repeated as the counter continues to operate until the last memory address is reached, so the counter's output becomes "**000011111**".

4. As the counter keeps counting, its output becomes "**000100000**", which means that the address goes back to the beginning "test_addr=0000" and goes to the write to memory process "test_wr=0" the test pattern corresponding to the code "pattern_code=001", until it reaches the last memory location "test_addr =1111", so the counter output becomes "**000101111**".

5. As the counter keeps counting, its output becomes "**000110000**" which means the address goes back to the beginning "test_addr=0000" and goes to the reading from memory process "test_wr=1" and compares its output with the corresponding test pattern "pattern_code=001". The process of reading and comparing is repeated as the counter continues to operate until the last memory address is reached, so the counter's output becomes "**000111111**".

6. The previous writing and reading process is repeated with all test patterns until the last test pattern corresponding to the code "pattern_code=111"is finished writing and reading when the counter output becomes "**011111111**".

7. As the counter continues to count, its output becomes "**100000000**", then the test finish signal becomes "finish=1" and is sent to the MBIST_controller to finish the test process.

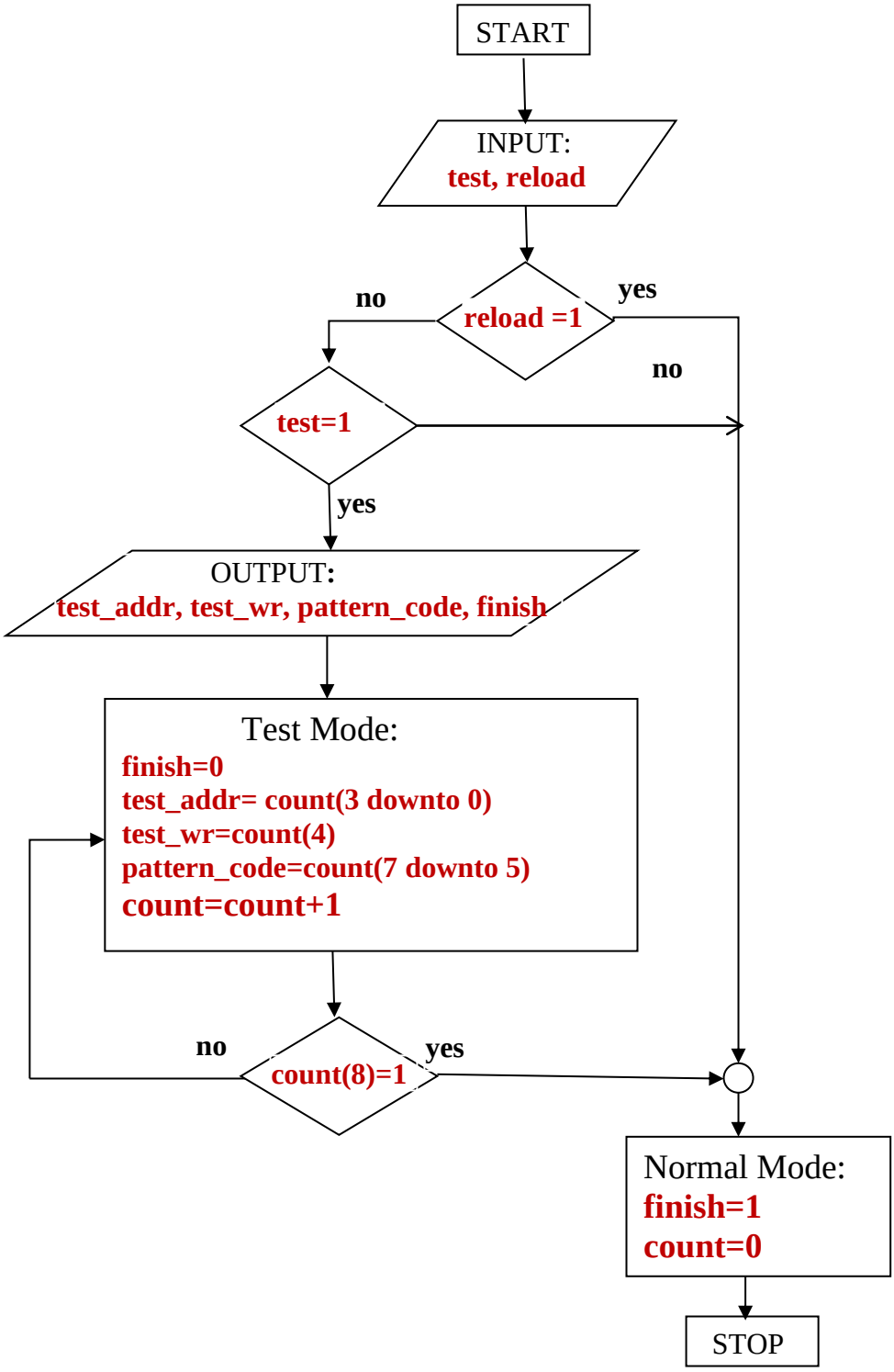


Figure 3.4: Flow chart of march_counter for testing 16x8 bit SRAM

3.4.3 Pattern_Decoder

This decoder is used to generate test data "test_pattern" by decoding a 3-bit input vector "pattern_code" according to Table 3.3 .

Table 3.3: Test Pattern

Pattern Number	Encoded Input	Test Pattern
0	000	00000000
1	001	00001111
2	010	00110011
3	011	01010101
4	100	11111111
5	101	11110000
6	110	11001100
7	111	10101010

3.4.4 MBIST- Comparator

MBIST- Comparator is used to detect the memory faults, by comparing test data "test_pattern" with memory data "data_out" when the memory test read operation "test_wr = '1'" is active. MBIST_Comparator sends "err =0" signal if there was no faults, and "err =1" when the fault was detected.

3.4.5 MBIST-Error- Counter

MBIST-Error- Counter is used to count the error numbers at the testing session.

3.4.6 ADDR-MUX

The address multiplexer is used as an isolating circuit to switch between functional address "address" and test address "test_addr" at the Test Mode.

3.4.7 Data-MUX

The data multiplexer is used as an isolating circuit to switch between functional input data "data" and test data "test_pattern" at the Test Mode.

3.4.8 WR-MUX

The write read enable signal multiplexer is utilized as an isolating circuit in the Test Mode to switch between functional write read enable "write_read" and test write read enable "test_wr".

3.4.9 SRAM16

The model of tested embedded Static Random Access Memory (SRAM) , consist of 16 words of 8 bit length.

3.5 FSM March Algorithm

This FSM March algorithm is a modified algorithm from the traditional March C- algorithm, and was designed to test word-oriented memories based on test patterns. It uses the stages of the March C- algorithm after replacing '0' with normal test patterns and '1' with inverse test patterns and iterating according to the number of natural test patterns which based on the length of the memory word. If the length of the memory word is 8 bits, test- patterns will be four , so the stages of the FSM March algorithm become as follows:

M0,0 : $\Updownarrow (w00000000)$.
M0,1 : $\Uparrow (r00000000, w11111111)$.
M0,2 : $\Uparrow (r11111111, w00000000)$.
M0,3 : $\Downarrow (r00000000, w11111111)$.
M0,4 : $\Downarrow (r11111111, w00000000)$.
M0,5 : $\Updownarrow (r00000000)$.
M1,0 : $\Updownarrow (w01010101)$.
M1,1 : $\Uparrow (r01010101, w10101010)$.
M1,2 : $\Uparrow (r10101010, w01010101)$.
M1,3 : $\Downarrow (r01010101, w10101010)$.
M1,4 : $\Downarrow (r10101010, w01010101)$.
M1,5 : $\Updownarrow (r01010101)$.
M2,0 : $\Updownarrow (w00110011)$.
M2,1 : $\Uparrow (r00110011, w11001100)$.
M2,2 : $\Uparrow (r11001100, w00110011)$.
M2,3 : $\Downarrow (r01010101, w11001100)$.
M2,4 : $\Downarrow (r11001100, w00110011)$.
M2,5 : $\Updownarrow (r00110011)$.
M3,0 : $\Updownarrow (w00001111)$.
M3,1 : $\Uparrow (r00001111, w11110000)$.
M3,2 : $\Uparrow (r11110000, w00001111)$.

M3,3 : $\Downarrow (r00001111, w11110000)$.
M3,4 : $\Downarrow (r11110000, w00001111)$.
M3,5 : $\Updownarrow (r00001111)$.

Also Table (3.2) discusses this test interpretation. Since each memory address is accessed 40 times (20 reads and 20 write) operations, the this algorithm complexity is of class 40N.

3.6 VHDL Design for FSM March Algorithm with 16x8 bit SRAM

The second proposed methodology is implemented using a VHDL design for a Built-in Self-testing system based on the FSM March algorithm with 16x8 bit tested memory as shown in Figure 3.5. The design includes the following components:

1. MBIST_Controller.
2. MARCH_C_FSM.
3. MBIST_Comparator.
4. MBIST_Error_Counter.
5. ADDR_MUX.
6. DATA_MUX.
7. WR_MUX.
8. SRAM16.

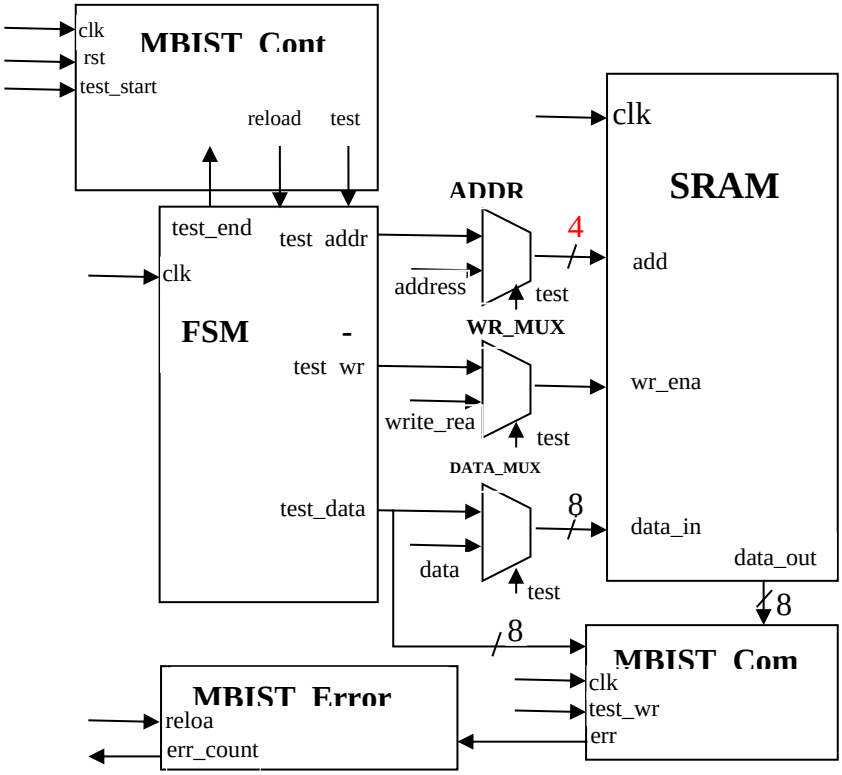


Figure 3.5: Block diagram of testing 16x8 bit sram using FSM March

The best way to implement the FSM March C- algorithm is to use the FSM finite state diagram as shown in Figure 3.6 .

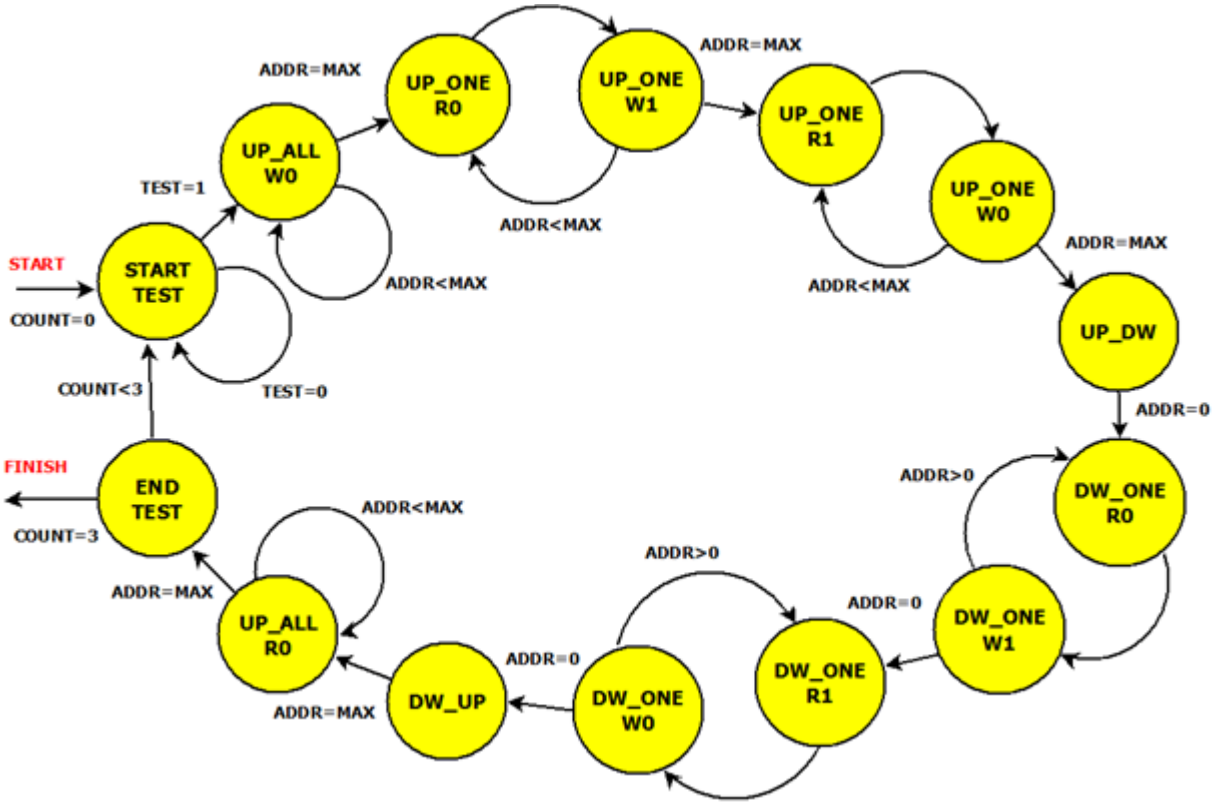


Figure 3.6: The finite state diagram of the FSM March algorithm

The finite state diagram of the FSM March algorithm consists of a set of sequential states which represent one test cycle "i", and the full test consists of several test cycles according to the number of normal test patterns that used in the test. If the length of the memory word is 8 bits, 'the number of normal test patterns 4 and thus the number of test cycles 4:

(i=0, normal pattern= "00000000", inverse pattern="11111111").

(i=1, normal pattern= "01010101", inverse pattern="10101010").

(i=2, normal pattern= "00110011", inverse pattern="11001100").

(i=3, normal pattern= "00001111", inverse pattern="11110000").

The state-to-state transition is mainly based on the current value of the test address "ADDR ", and the following states:

1. Start test "START_TEST" state: It is the initial state in the test cycle, in which the value of the test address is initialized with the maximum value "ADDR=MAX", and the test finish

signal is zeroed "finish=0". Transition from this state to next state can only be carried out when the test signal becomes "test=1".

2. Mi,0- "UP_ALL_W0" state: Write normal test pattern "i" in all memory locations from primitive address "0" to "MAX". Where normal test pattern "i" is related to the test cycle number "count=i".

3. Mi,1- "UP_ONE_R0" and "UP_ONE _W1" states: Read the first memory location and make sure it contains the normal test pattern "i" and write the inverse test pattern "i" in this location, then move to the next location, and repeat the previous process until reaching the maximum address "MAX".

4. Mi,2- "UP_ONE_R1" and "UP_ONE_W0" states: Read the first memory location and make sure that it contains the inverse test pattern "i" and write the normal test pattern "i" in this location, then move to the next location, and repeat the previous process until reaching the maximum address "MAX".

5." UP DOWN" state: Switching the operation of memory reading and writing from ascending to descending addresses ,by setting the test address to the primitive value "ADDR=0".

6. Mi,3- "DOWN_ONE_R0" and "DOWN_ONE _W1" states: Read the last memory location and make sure that it contains the normal test pattern "i" and write the inverse test pattern "i" in this location, then move to the previous location, and repeat the previous process until reaching the primitive address "0".

7. Mi,4- "DOWN_ONE_R1" and "DOWN_ONE_W0" states: Read the last memory location and make sure that it contains the inverse test pattern "i" and write the normal test pattern "i" in this location, then move to the previous location, and repeat the previous process until reaching the primitive address "0".

8. "DOWN UP" state: Switching the operation of memory reading and writing from descending to ascending addresses ,by setting the test address to the maximum value "ADDR= MAX".

9. Mi,5- "UP_ALL_R0" state: Read all memory locations from primitive address "0" to "MAX", and compare it with the normal test pattern "i".

10. "END_TEST" state: It is the final state in which the test address is initialized with the maximum value "ADDR=MAX" and the value of the test counter (count) is checked, if it is (count=3), the test is terminated by sending the test finish signal (finish=1) to the "MBIST_controller", else if it is (count<3), the next test cycle will begin after increasing the value of the test counter (count=count+1=i), where this value represents the next test cycle normal and inverse test pattern.

The March FSM circuit in states (Mi,5; Mi,4; Mi,3; Mi,2; Mi,1; Mi,0) generates the test address "test_addr", test patterns "test_data" and the write/read enable signal "test_wr" as it is shown in the detailed flow chart diagram Figure 3.7. During each memory read operation "test_wr=1", the "MBIST_Comparator" compares the memory output "data_out" with the test patterns "test_pattern" to check that they are matching , and generates an error signal "err=1" if no match occurs.

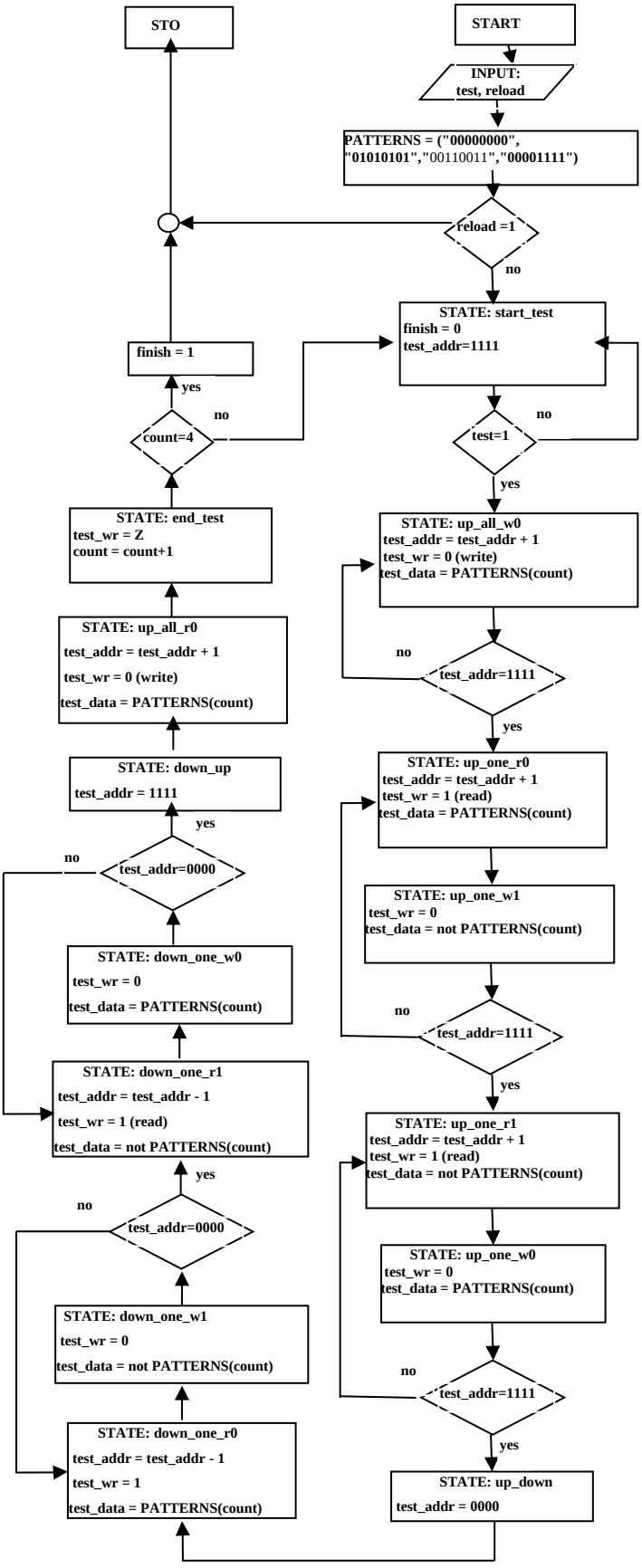


Figure 3.7:Flow chart of March FSM for testing 16x8 bit SRAM

3.7 VHDL Design for Fast March Algorithm with 32x8 bit SRAM:

To verify the effectives of the proposed methodologies ,both algorithms were applied on SRAM (32x8 bit) . This design is the same as section 3.4 except using (32x8 bit) tested memory. This leads to increase length of "test_addr" signal of **MARCH_Counter** to become 5 bit (2^5=32) instead of 4 bit in previous circuit, so the counter length becomes 9 bit as shown in Figure 3.8 .

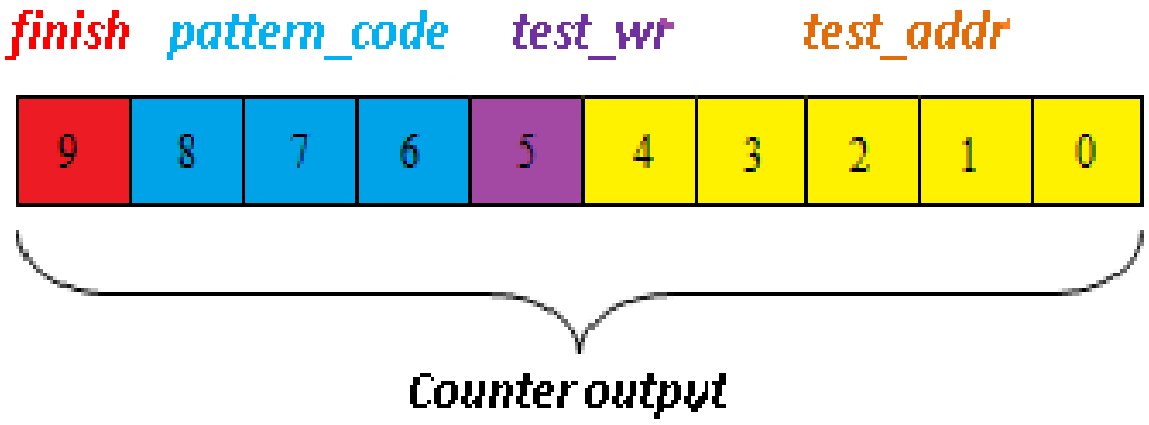


Figure 3.8: Counter output for 32x8 bit SRAM

The flow chart of **MARCH_Counter** in this case study shown in Figure 3.9 .

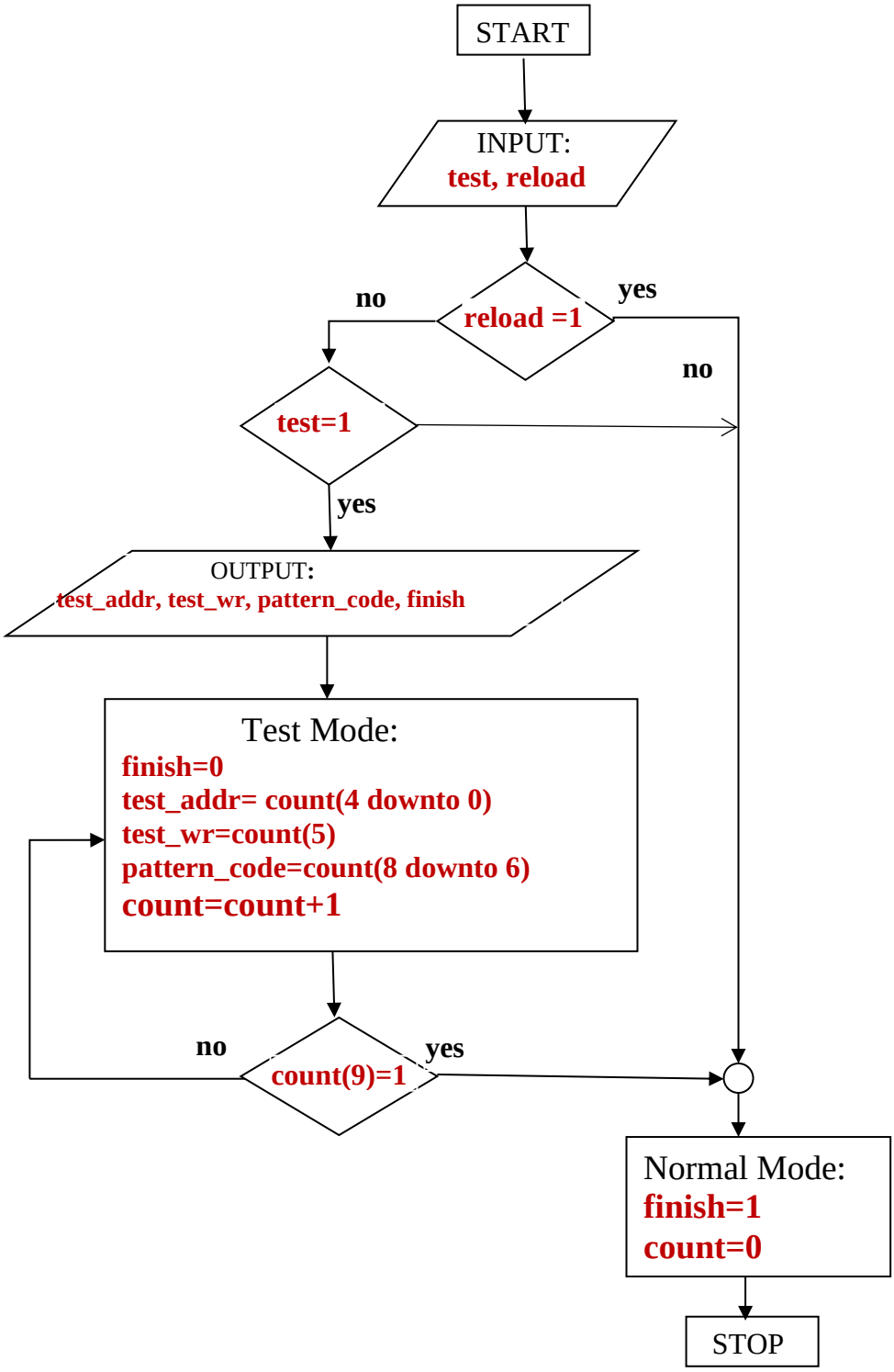


Figure 3.9: Flow chart of MARCH_Counter for testing 32x8 bit SRAM

Figure 3.10 shows the component of the proposed design.

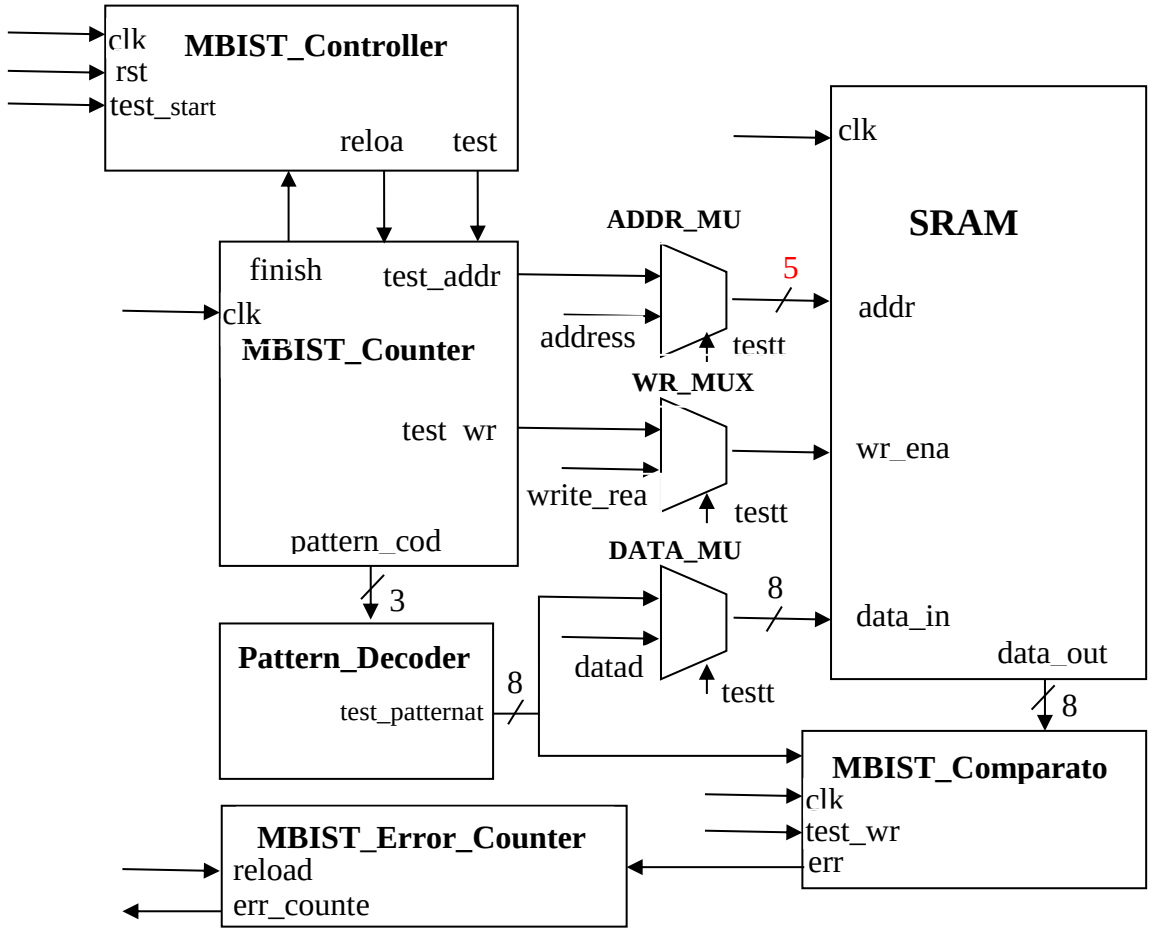


Figure 3.10: Block diagram of testing 32x8 bit SRAM using Fast March

3.8 VHDL Design for FSM March Algorithm with 32x8 bit SRAM

In this design, a (32x8 bit) SRAM is used as under test memory . This leads to increase length of "test_addr" signal of March FSM component to become 5 bit (2^5=32) instead of 4 bit in previous circuit as show in Figure 3.10 .

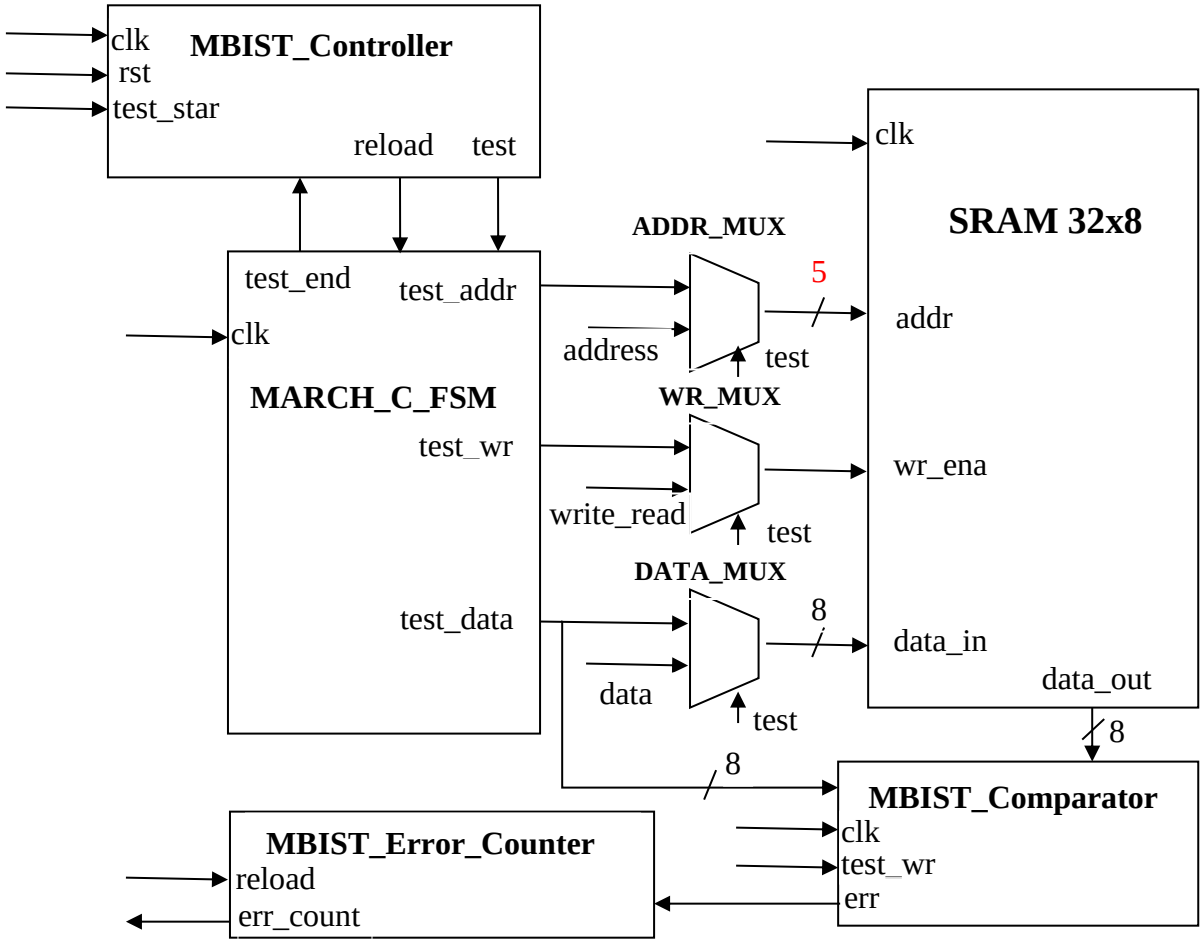


Figure 3.11: Block diagram of testing 32x8 bit SRAM using FSM March

The change of "test_addr" signal will also lead to some changes in internal architecture of March FSM components which generate this signal, and can be summarized in the value of max address which become "11111"and the main address which becomes "00000" as shown in March FSM flow chart(Figure 3.12).

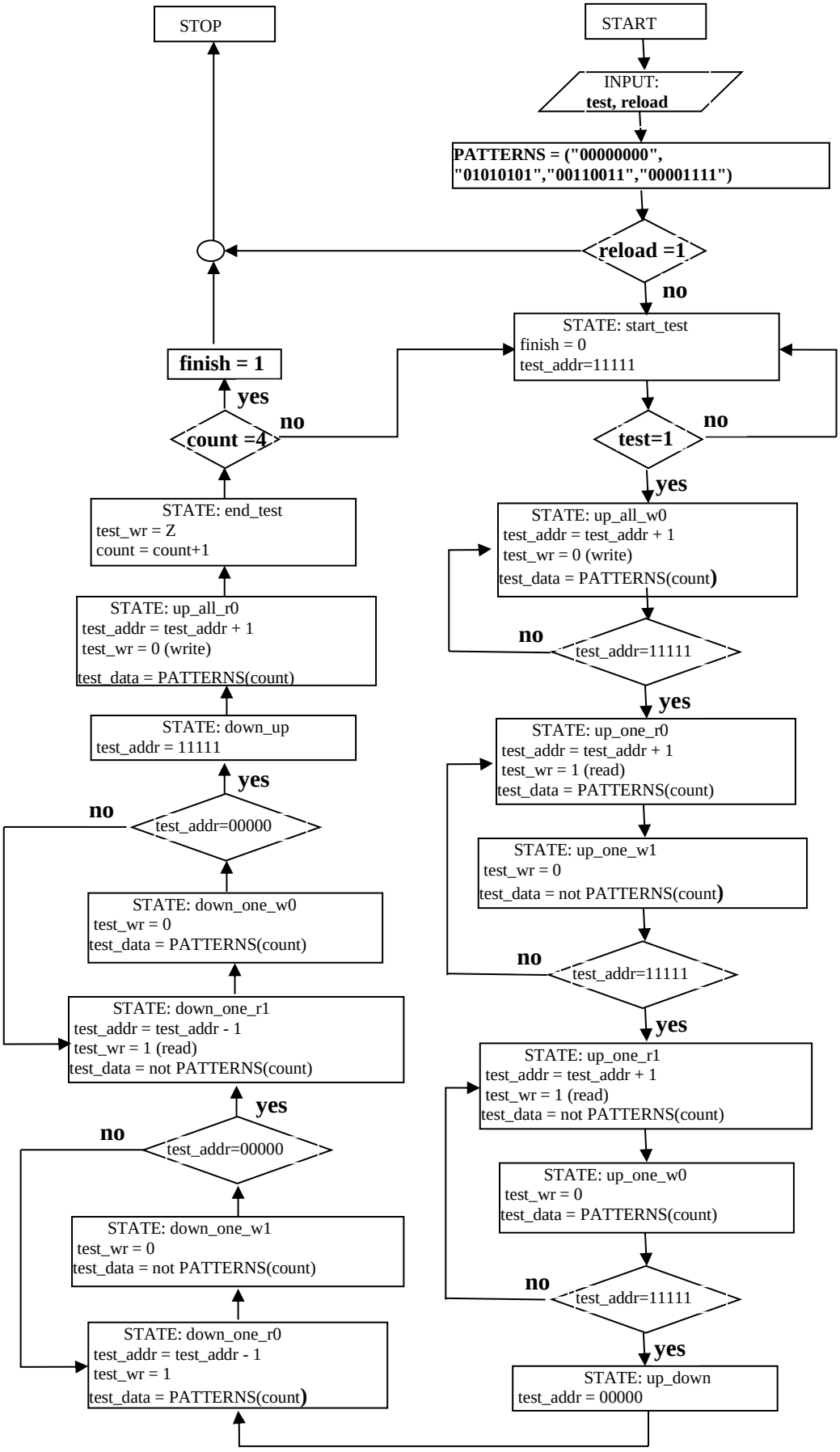


Figure 3.12: Flow chart of March FSM for testing 32x8 bit SRAM

CHAPTER 4

Chapter Four

4. Results and Discussion

4.1 Introduction

The simulation findings collected and documented in this chapter are divided into two parts, fault models simulation and performance analyze, which will be described in detail.

4.2 Memory fault models simulation:

There are numerous fault models that are applicable to memories. While logic predominantly uses the stuck-at fault model plus some other fault models only sparingly, memory fault modeling is much broader. The time is 100mHz and simulation result in Modaslim altera 6.5 There are four classic memory fault models:

1. **The Stuck-at-Fault (SAF)**.
2. **Address -Fault (AF).**
3. **Coupling -Fault (CF).**
4. **Transition-Fault (TF).**

A Markov diagram is a useful approach to explain memory faults by describing the states of fault-free and faulty memories.

4.2.1 Memory fault-free model:

Figure 4.1 shows the Markov diagram for a (fault-free) memory cell.

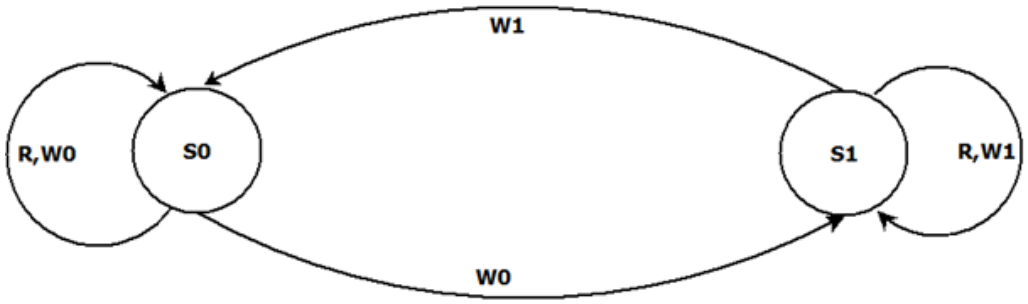


Figure 4.1: Markov diagram for fault free memory

A single defect-free cell may be inscribed to any state and, while read, maintains the data it previously had. A letter "R" denotes as read process, whereas "W0" and "W1" denote write operations of "0" and "1," respectively. S0 and S1 show whether the cell is in a "0" or "1" state .

The simulation results for testing fault free memory with Fast March circuit is illustrated in Figure 4.2.

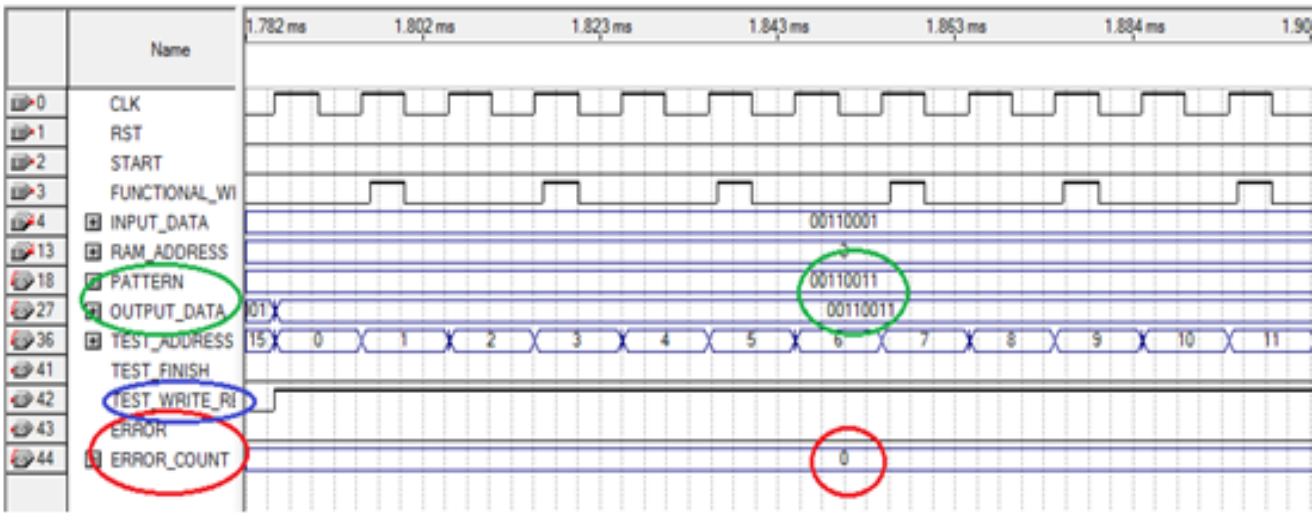


Figure 4.2: Simulation of testing fault-free memory with Fast March

The memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are the same on all memory addresses (**TEST_ADDRESS**) ("00110011" = "00110011") during memory read operation (**TEST_WRITE_READ=1**), so there are no error occurred (**ERROR=0, ERROR_COUNT=0)** during the memory test.

The simulation result for testing fault free memory with FSM March circuit is depicted in Figure 4.3.

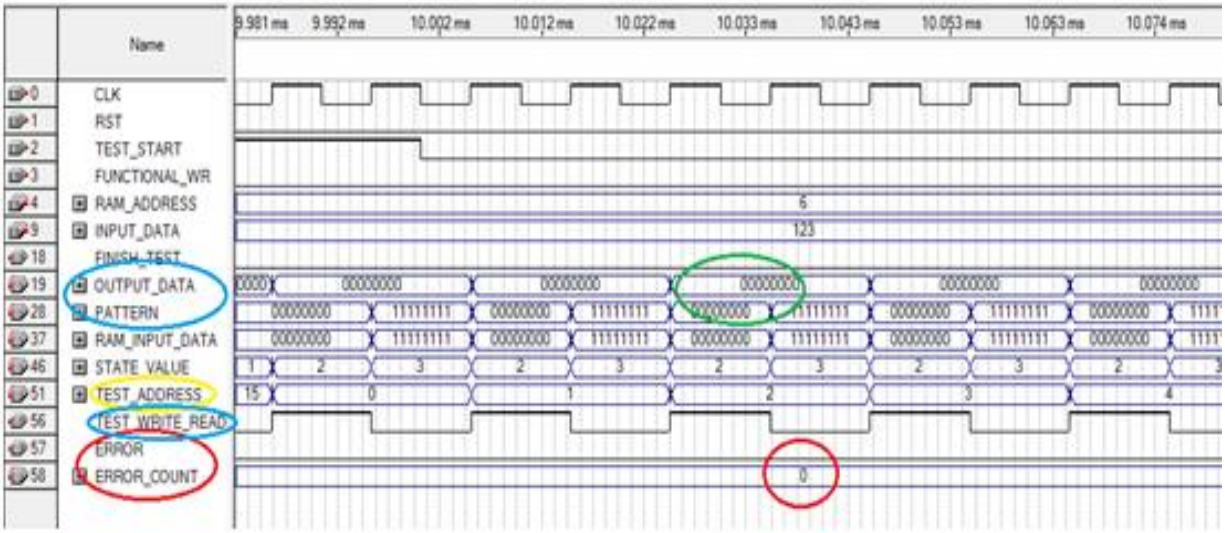


Figure 4.3:Simulation for testing fault-free memory with FSM Marchtion for testing fault-free memory with FSM March.

No difference between the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) during memory read operation (**TEST_WRITE_READ=1**), so there are no error occurred (**ERROR=0, ERROR_COUNT=0)** during the memory test.

4.2.2 The Stuck-at-Fault (SAF) Model

This type of fault indicates that a cell or line has been stuck at logical "1" or "0," and are known as Stuck-at-0 and Stuck-at-1, correspondingly. Figure 4.4 depicts the Markov diagrams for Stuck-at-0 and Stuck-at-1.

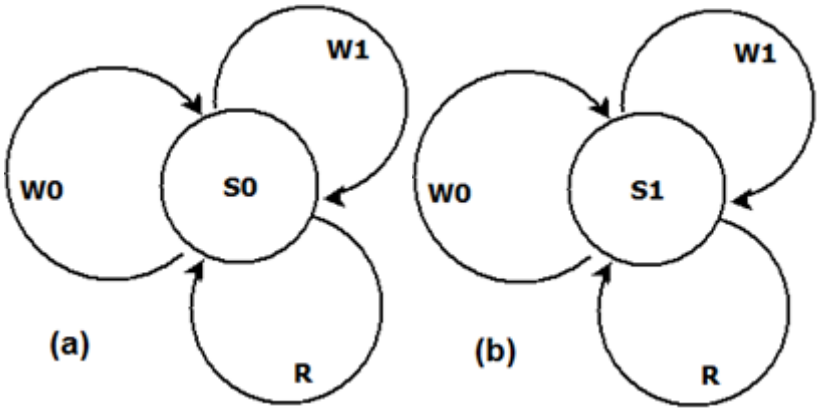


Figure 4.4: Markov diagrams (a) stuck at 0 , (b) stuck at 1.

4.2.2.1 Simulation the test of Stuck-at-Fault (stuck-at-0 fault):

The simulation results for testing stuck-at-0 fault in SRAM 16 and SRAM 32 bit with Fast March circuit are shown in the Figure 4.5 and 4.6.

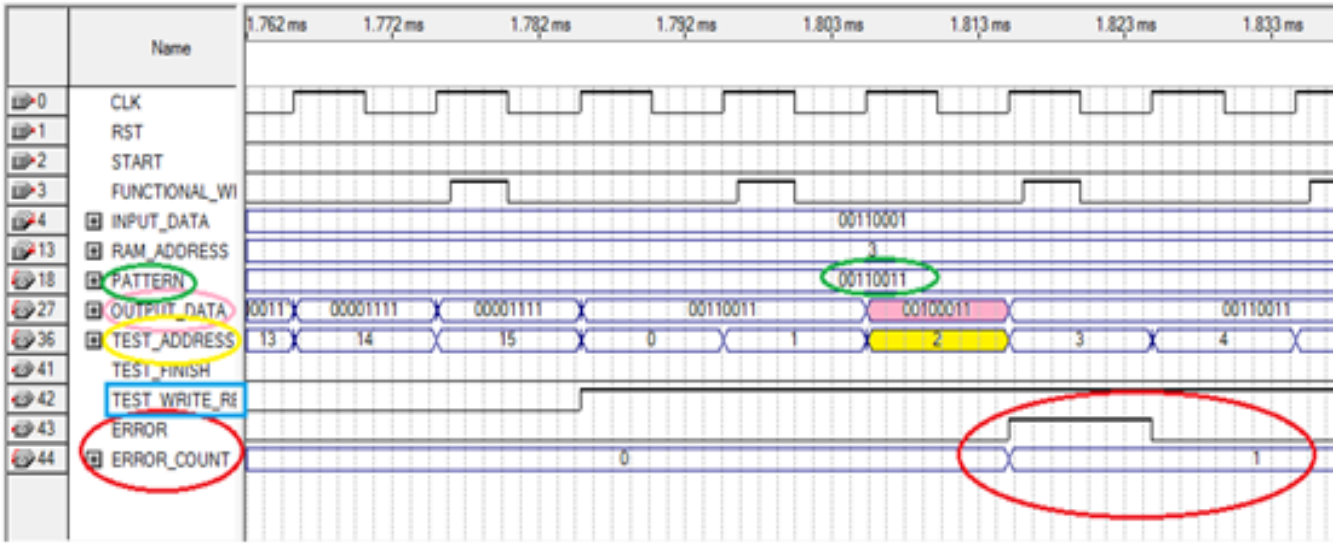


Figure 4.5: Simulation for Stuck-At-0 Fault in SRAM 16 bit with Fast March

.

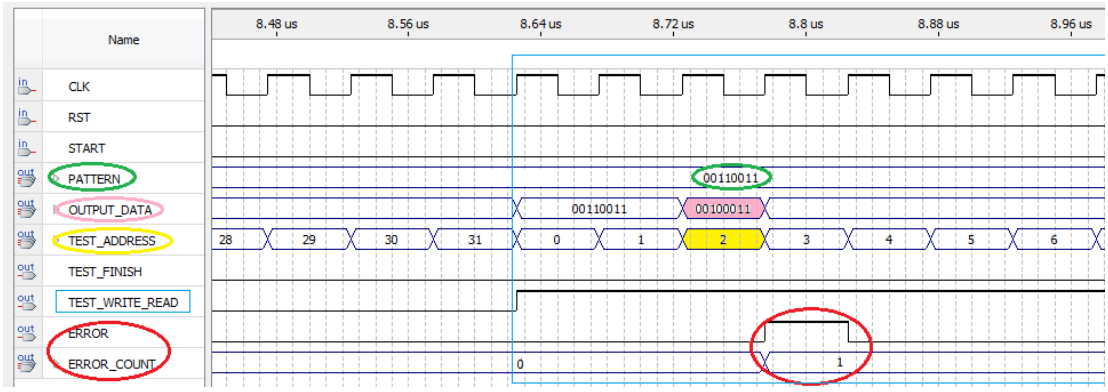


Figure 4.6: Simulation for Stuck-At-0 Fault in SRAM 32 bit with Fast March

.

The testing results of the two memories are the same, where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) during memory read operation (**TEST_WRITE_READ=1**) are not the same at memory address(**TEST_ADDRESS=2**) ("00100011"~="00110011"), which leads to an error (**ERROR=1, ERROR_COUNT=1)** at that address.

The simulation results for testing Stuck-At-0 fault in SRAM 16 and SRAM 32 bit with FSM March circuit are shown in the Figure 4.7 and the Figure 4.8.

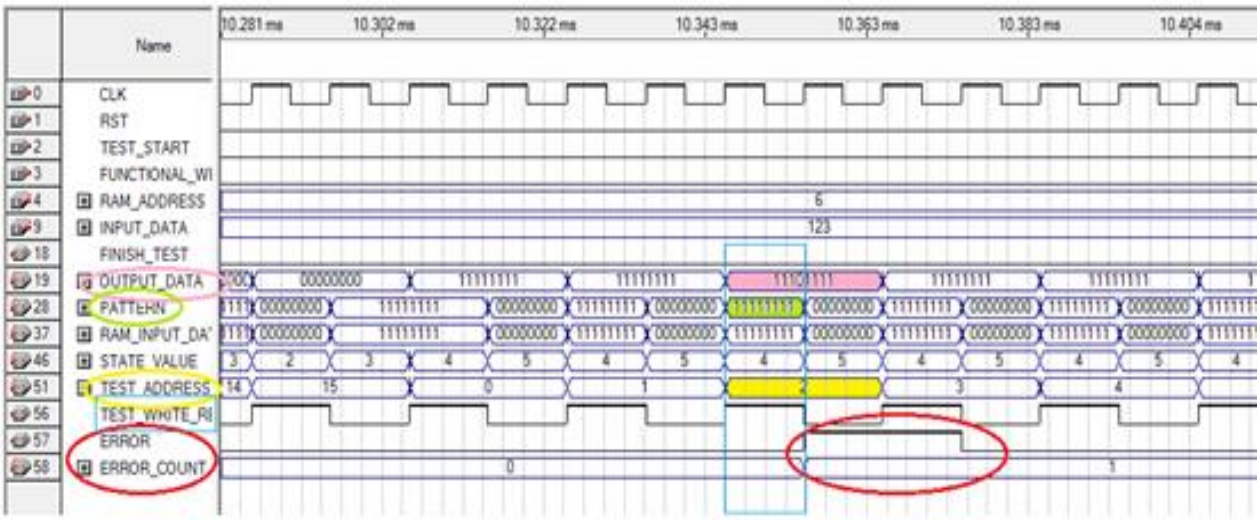


Figure 4.7: Simulation for Stuck-At-0 fault in SRAM 16 bit with FSM March.

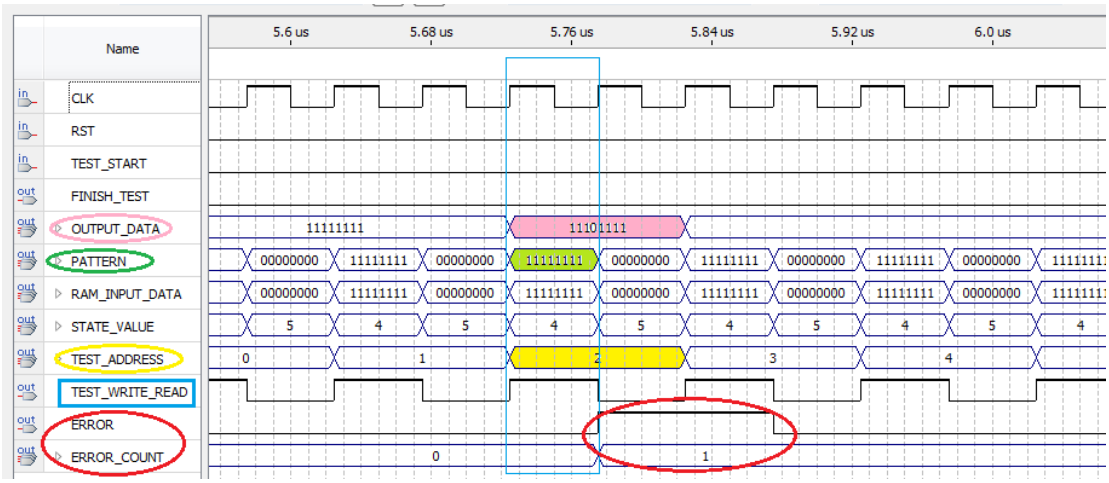


Figure 4.8: Simulation for testing Stuck-At-0 fault in SRAM 32 with FSM March.

The results of testing the two memories are the same, and there are an error occurred (**ERROR=1, ERROR_COUNT=1)** at the memory address (**TEST_ADDRESS=2**), because the memory output data is not equal the test patterns ("11101111"~="11111111") at that address.

4.2.2.2 Simulation the test of Stuck-At-Fault (stuck-at-1 fault)

The simulation results for testing stuck-at-1 fault in SRAM 16 and SRAM 32 bit with Fast March circuit are shown in the Figure 4.9 and 4.10.

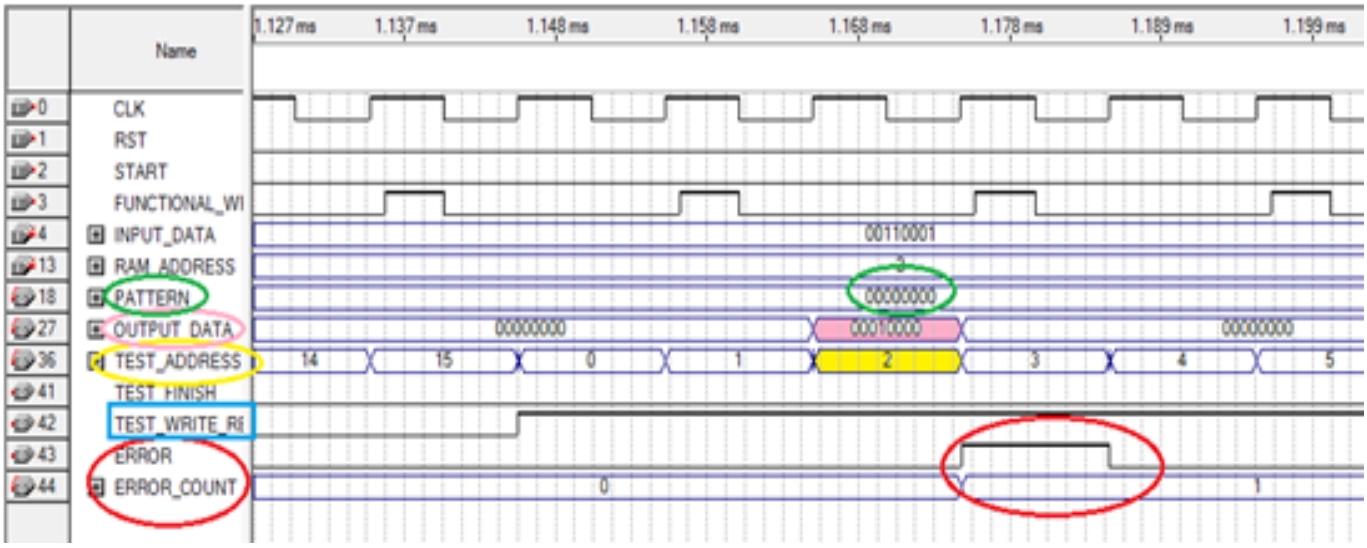


Figure 4.9: Simulation of testing Stuck-At-1 fault in SRAM 16 with Fast March

.

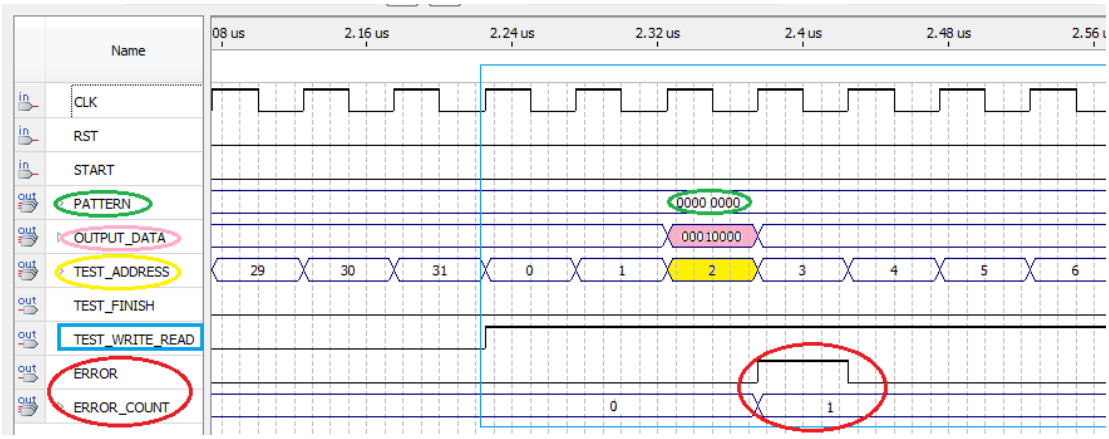


Figure 4.10: Simulation of testing Stuck-At-1 fault in SRAM 32 with Fast March

.

The simulation results shows that there are no difference between the two memories test, where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same ("00010000"~="00000000") at the memory address (**TEST_ADDRESS=2**) during memory read operation (**TEST_WRITE_READ=1**), which leads to an error (**ERROR=1, ERROR_COUNT=1)** at that address.

The simulation results for testing stuck-at-1 fault in SRAM 16 and SRAM 32 bit with FSM March circuit is shown in the Figure 4.11 and 4.12.

Figure 4.11: Simulation of testing Stuck-At-1 fault in SRAM 16 with FSM March

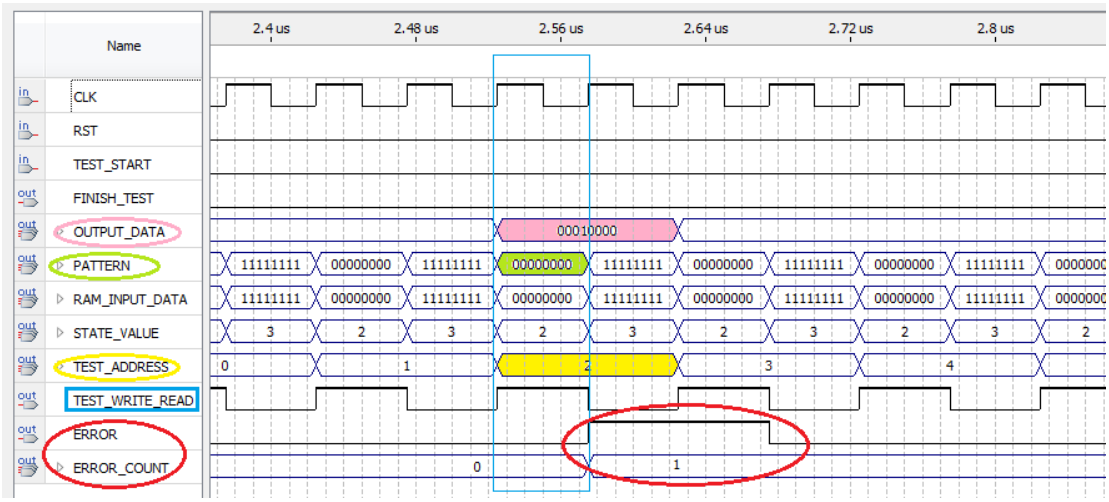


Figure 4.12: Simulation of testing Stuck-At-1 fault in SRAM 32 with FSM March

.

There are no difference in simulation results of testing the two memories, and these results show that the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same ("00010000"~="00000000") at the address (**TEST_ADDRESS=2**) during memory read operation (**TEST_WRITE_READ=1**), which leads to an error (**ERROR=1, ERROR_COUNT=1)** at that address.

4.2.3 The Transition Fault (TF) Model

Stuck-at-Fault appears to be comparable to transition faults. However, once the memory cells have been written to one state, it is difficult to restore to the previous state. As a result, when the memory is triggered, the cell might be in either a "0" or "1" state. Because there is only one way to write it, there are transition-to-0 faults when a memory cell may write to S0 but not to S1 after transitioning to S0, as shown in Figure 4.13 of the Markov diagram.

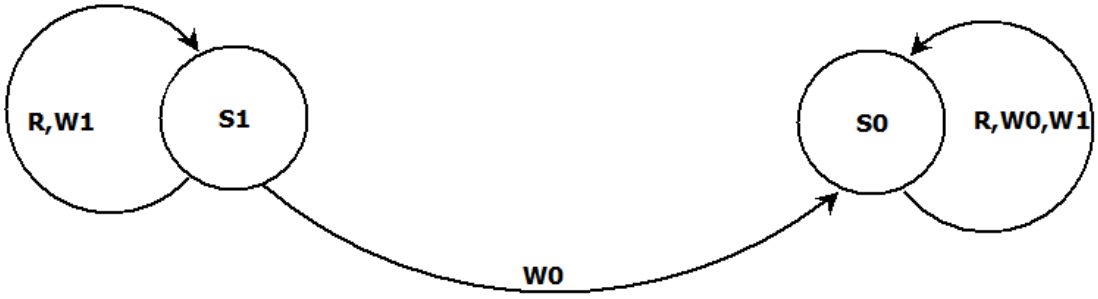


Figure 4.13: Markov diagram for transition-to-0 fault.

transition-to-1 fault when memory cell can be written to **S1** state but can't written to **S0** state after transition to state **S1** as shown in Figure 4.14.

Figure 4.14: Markov diagram for transition-to-1 fault

4.2.3.1 Simulation the test of Transition Fault (transition-to-0 fault)

The simulation result for testing transition-to-0 fault in SRAM 16 and SRAM 32 bit with Fast March circuit is shown in the Figure 4.15 and 4.16.

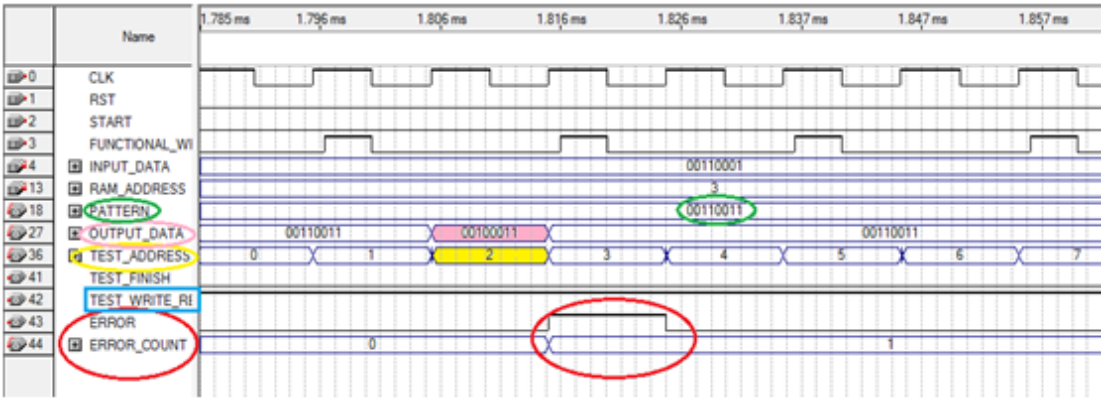


Figure 4.15: Simulation for transition-to-0 fault in SRAM 16 with Fast March

.

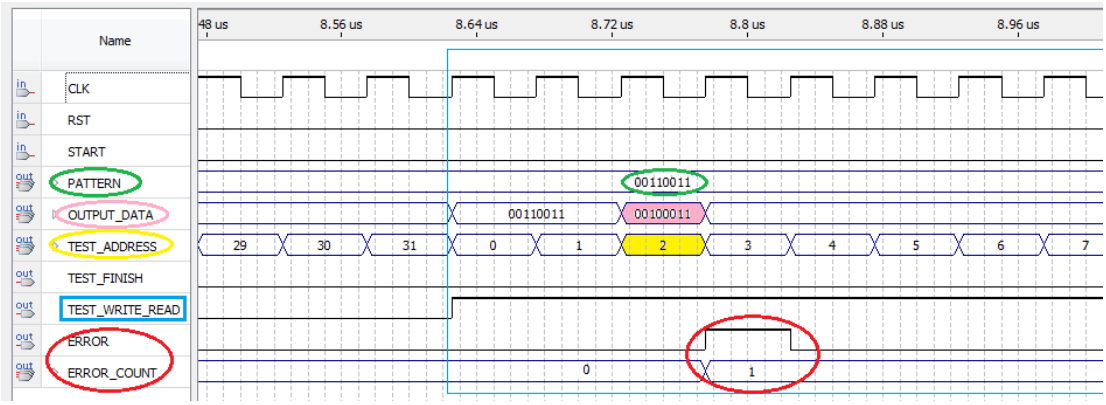


Figure 4.16: Simulation for transition-to-0 fault in SRAM 32 with Fast March

.

The results show that the two memories test is similar, where there are an error occurred (**ERROR=1, ERROR_COUNT=1)** at that address (**TEST_ADDRESS=2**)during memory read operation (**TEST_WRITE_READ=1**), because the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same ("00100011"~="00110011") at that address.

The simulation results for testing transition-to-0 fault in SRAM 16 and SRAM 32 bit with FSM March circuit is shown in the Figure 4.17 and 4.18.

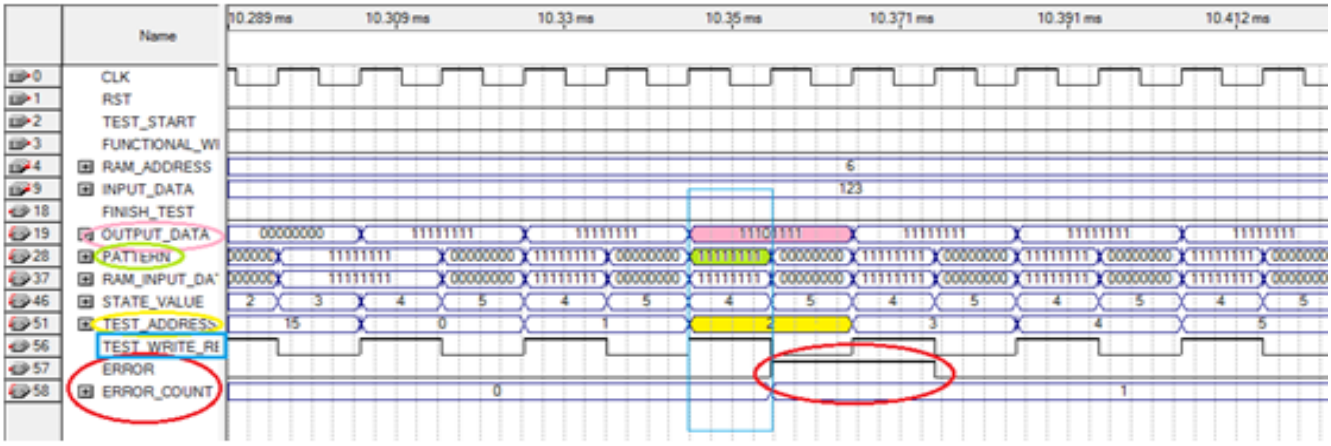


Figure 4.17: Simulation for transition-to-0 fault in SRAM 16 with FSM March

.

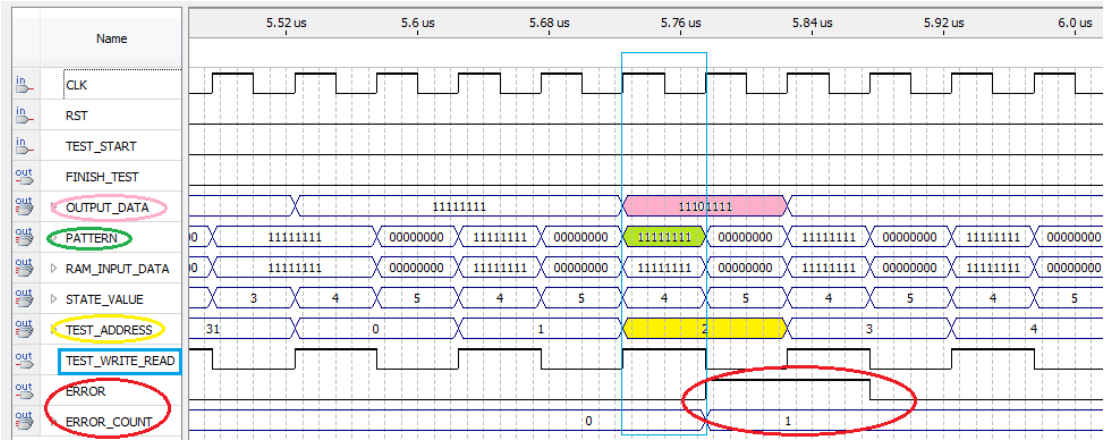


Figure 4.18: Simulation for transition-to-0 fault in SRAM 32 with FSM March

.

The results show that the testing of the two memories are the same, and the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same at the memory address (**TEST_ADDRESS=2**) ("11101111"~="11111111") during memory read operation (**TEST_WRITE_READ=1**), so there are an error occurred (**ERROR=1, ERROR_COUNT=1)** at that address.

4.2.3.2 Simulation the test of Transition Fault (transition-to-1 fault)

The simulation results for testing transition-to-1 fault in SRAM 16 and SRAM 32 bit with Fast March circuit is explained in the Figure 4.19 and Figure 4.20.

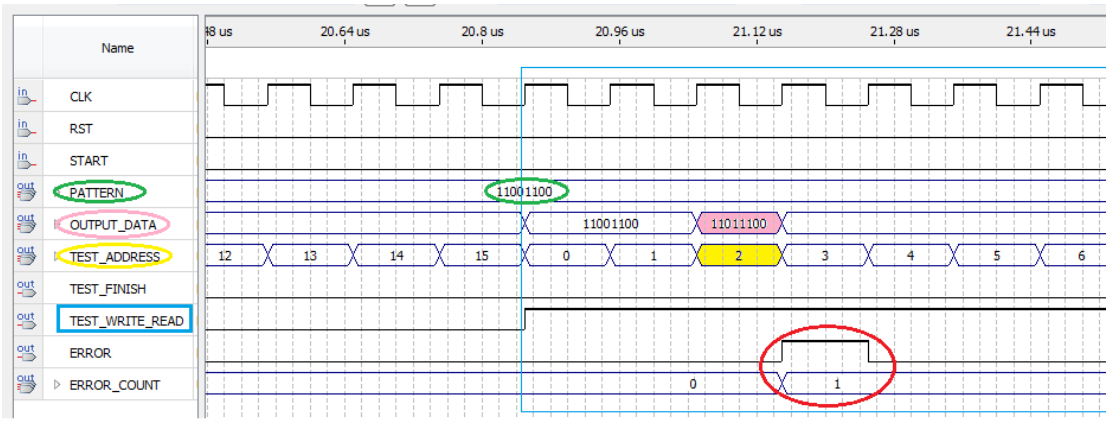


Figure 4.19: Simulation for transition-to-1 fault in SRAM 16 with Fast March

.

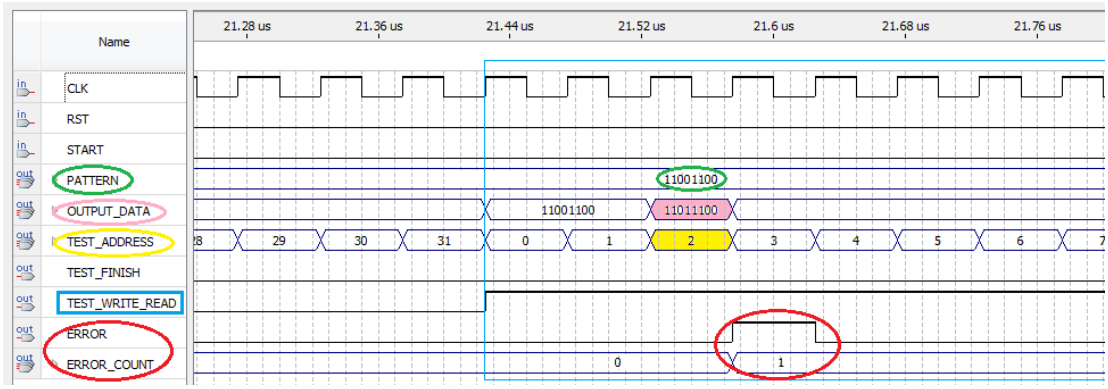


Figure 4.20: Simulation for transition-to-1 fault in SRAM 32 with Fast March

.

There is no difference in the two memories test, and the results show that there are an error occurred (**ERROR=1, ERROR_COUNT=1)** the address (**TEST_ADDRESS=2**), because the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are different ("11011100"~="11001100") during memory read operation (**TEST_WRITE_READ=1**) at that address.

The simulation results for testing transition-to-1 fault in SRAM 16 and SRAM 32 bit with FSM March circuit is illustrate in Figure 4.21and Figure 4.22.

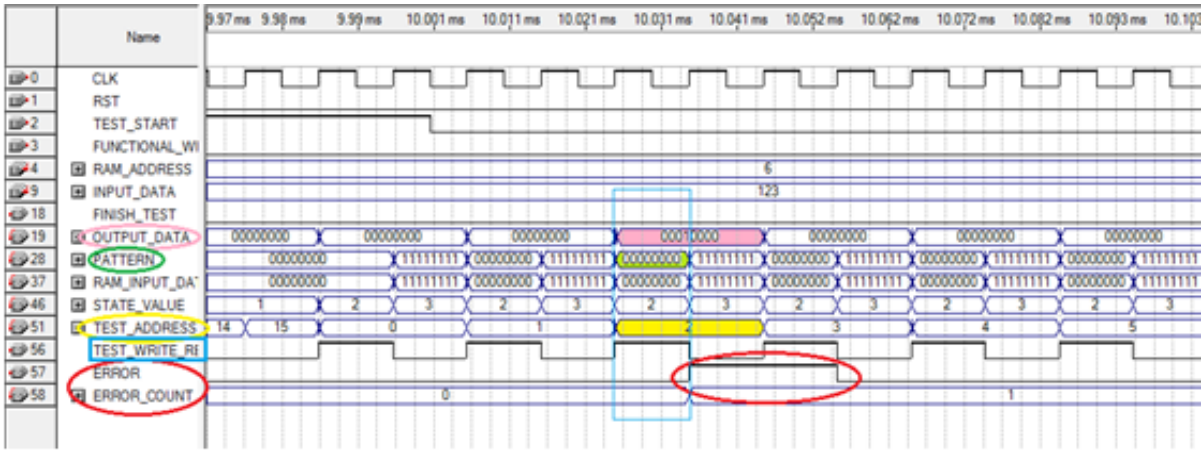


Figure 4.21: Simulation for transition-to-1 fault in SRAM 16 with FSM March

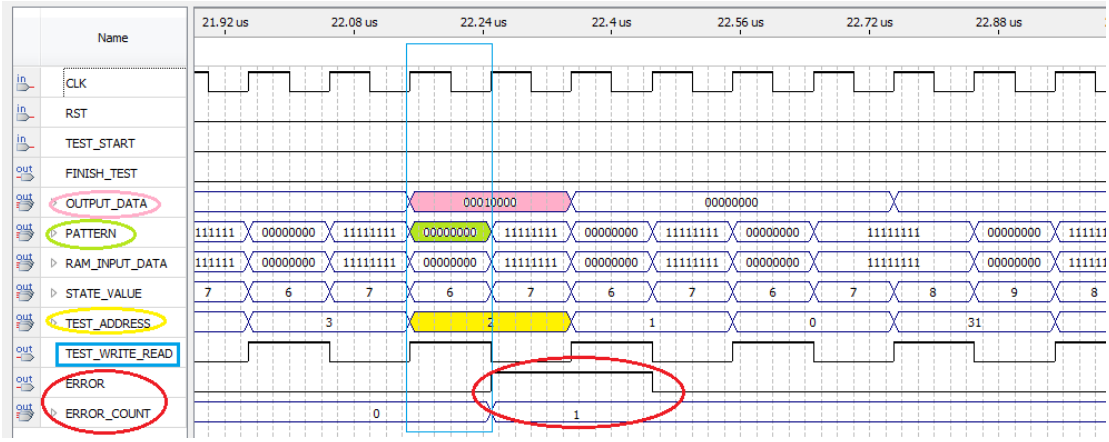


Figure 4.22: Simulation for transition-to-1 fault in SRAM 32 with FSM March

.

The simulation results are the same for two memories, and there are an error occurred (**ERROR=1, ERROR_COUNT=1)** at the memory address (**TEST_ADDRESS=2**) where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same ("00010000"~="00000000") at that address.

4.2.4 The Coupling Fault (CF) Model

Coupling fault occurs in memory because of the regularity of its structure, in which a change in one cell causes a change in another one. A parasitic diode connection between two cells causes the coupling. There are two kinds of CF: Fault of Idempotent and Inversion Coupling

a) Inversion-Coupling fault (CFin) model:

The interaction of two neighboring cells "i" and "j", cause the inverse transition on cells "i" by the alteration write process (write 0 - 1 or 1 - 0) of the cell "j". Cell "j" is known as the linked (aggressor) cell, and its subjects are reversed in cell I, recognized as the Coupled (victim) cell. For instance, a 0 - 1 transition write process results in a 1 - 0 CFin model in cell "i". The Markov diagram for Inversion Coupling Faults is shown in Figure 4.23.

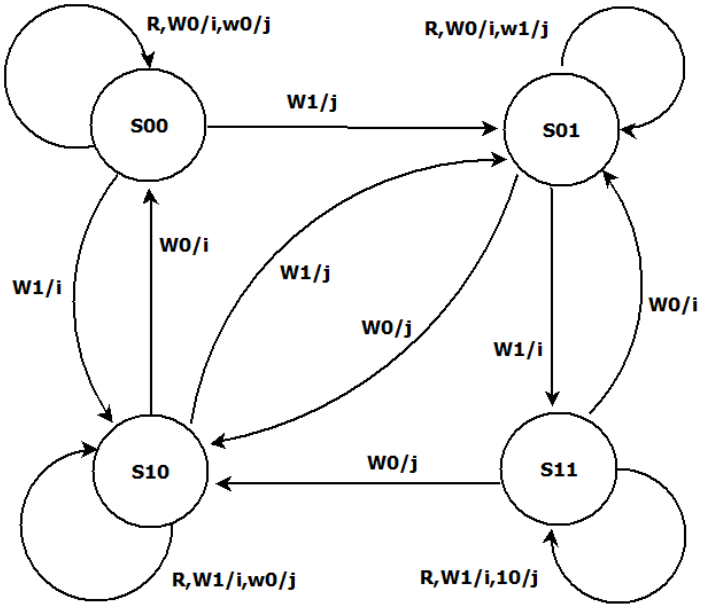


Figure 4.23: Markov diagram for inversion coupling faults

.

b) Idempotent Coupling Faults (CFid) model: In this type of coupling fault, the coupled cell is forced to '1' or '0' if coupling cell transits from '0' to '1' or '1' to '0'**.** The Markov diagram for Idempotent Coupling Faults is shown in Figure 4.24.

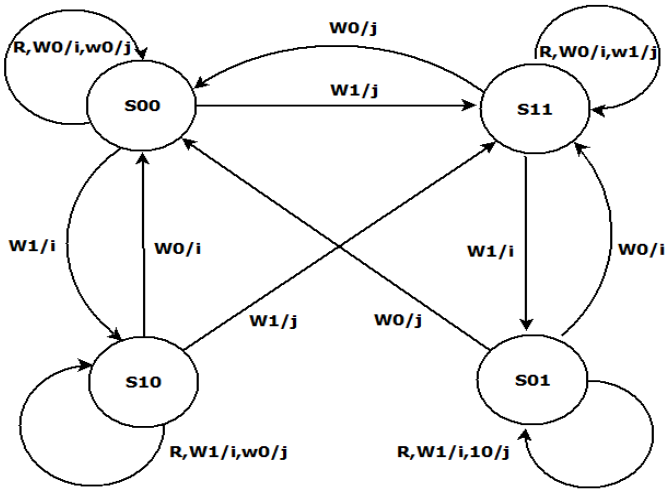


Figure 4.24: The markov diagram for idempotent coupling faults

4.2.4.1 Simulation the test of Inversion Coupling Faults (CFin):

The simulation results for testing CFin fault in SRAM 16 and SRAM 32 bit with Fast March circuit is shown in Figure 4.25 and 4.26.

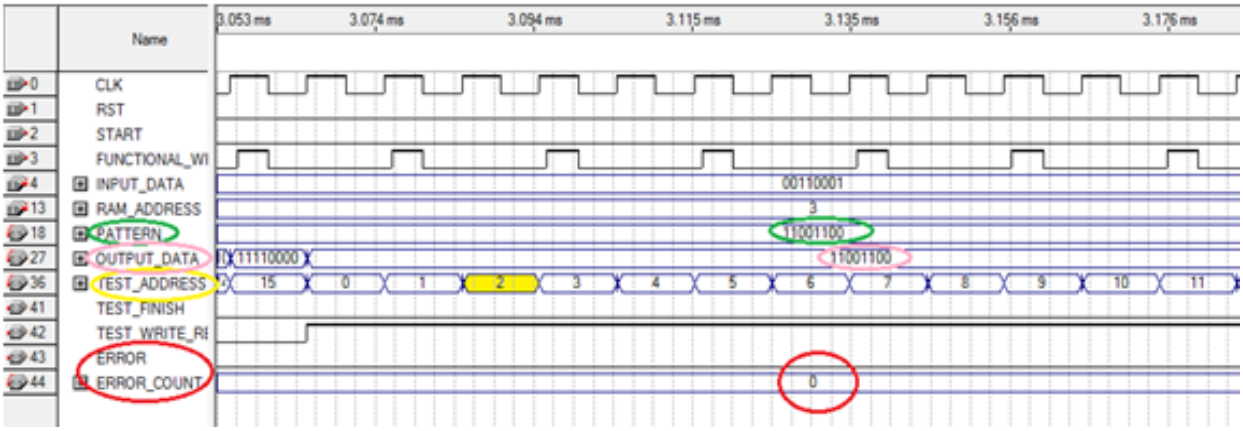


Figure 4.25: Simulation for testing CFin fault in SRAM 16 with Fast March

.

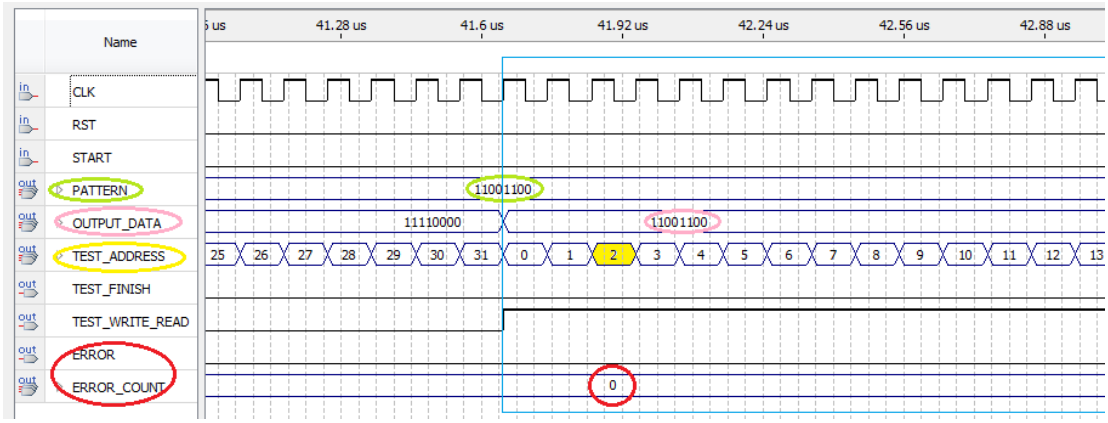


Figure 4.26: Simulation for testing CFin fault in SRAM 32 with Fast March

The simulation results show that the testing of the two memories are the same, also show that the memory output data (**OUTPUT DATA**) and testing patterns (**PATTERN**) are the same on all memory addresses during memory read operation (**TEST WRITE READ=1**), so no errors occurred (**ERROR=0, ERROR COUNT=0**) at the faulty address (**TEST ADDRESS=2**).

Therefore the Fast March algorithm is unable of detecting the CFin fault. The reason can be explained by the stages of the Fast March algorithm as follows:

1.[⇕(*w*00000000)]: Write "00000000" in all memory locations, so coupling bit j(5) become '0', and coupled bit i(4) becomes '0' as written value because no transition has yet occurred on coupling bit j(5).

2.[⇕(*w*11111111)]: Write "111111111" in all memory locations, so coupling bit j(5) becomes '1', and coupled bit i(4) becomes '1' as written value because its previous value was the same as coupling bit j(5) (this transition affect only on '1' and make it '0').

3.[⇕(*w*01010101)]: Write "0101010101" in all memory locations, so coupling bit j(5) remains '1', and coupled bit i(4) becomes '0' as written value because no transition has occurred on coupling bit j(5).

4.[⇕(*w*10101010)]: Write "10101010" in all memory locations, so coupling bit j(5) becomes '0', and coupled bit i(4) becomes '1' as written value because it is the same as coupling fault value (1-0 transition of coupling bit j(5) make coupled bit i(4) transits 0-1).

5.[⇕(*w*00110011)]: Write "00110011" in all memory locations, so coupling bit j(5) becomes '1', and coupled bit i(4) becomes '0' as written value because it is the same as coupling fault value (0-1 transition of coupling bit j(5) make coupled bit i(4) transits 1-0).

6.[⇕(*w*11001100)]: Write "11001100" in all memory locations, so coupling bit j(5) becomes '0', and coupled bit i(4) becomes '1' as written value because it is the same as coupling fault value (1-0 transition of coupling bit j(5) make coupled bit i(4) transits 0-1).

7.[⇕(*w*00001111)]: Write "00001111" in all memory locations, so coupling bit j(5) remains '0', and coupled bit i(4) remains '1' as written value because no transition has occurred on coupling bit j(5).

8.[⇕(*w*11110000)]: Write "11110000" in all memory locations, so coupling bit j(5) becomes '1', and coupled bit i(4) becomes '0' as written value because it is the same as coupling fault value (0-1 transition of coupling bit j(5) make coupled bit i(4) transits 1-0).

This problem is due to not covering all possible combination of the previous and new written values in the coupling and coupled cell. The combination which leads to detecting CFin fault can be summarized in the following Table 4.1:

Table 4.1: CFin detecting combinations

Coupled cell		**Coupling cell**	
Previous value	**New value**	**Previous value**	**New value**
0	0	1	0
1	1	0	1

As the example to that, if coupling bit j(5) is inversed with coupled bit i(4), the Fast March circuit will detecting the CFin fault as shown in Figure 4.27:

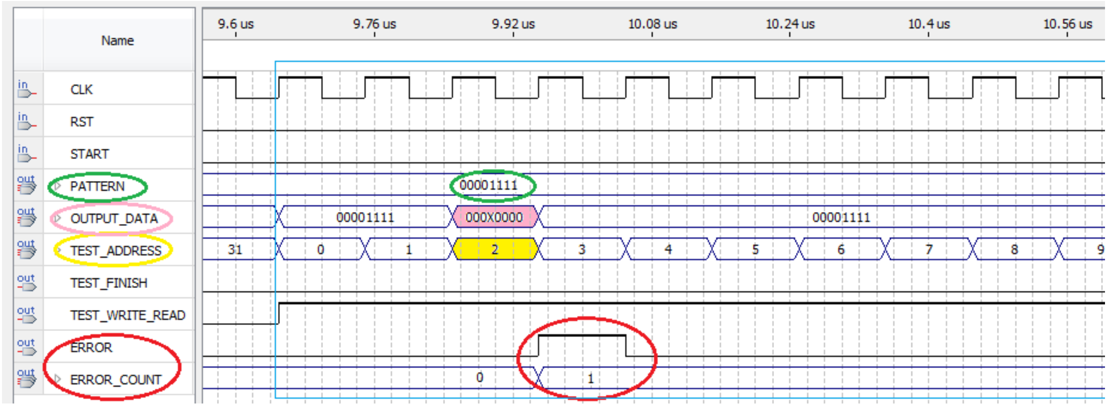


Figure 4.27: Simulation for detected CFin fault in SRAM 32 with Fast March

The simulation results for testing CFin fault in SRAM 16 and SRAM 32 bit with FSM March circuit is shown in the Figure 4.28 and 4.92 :

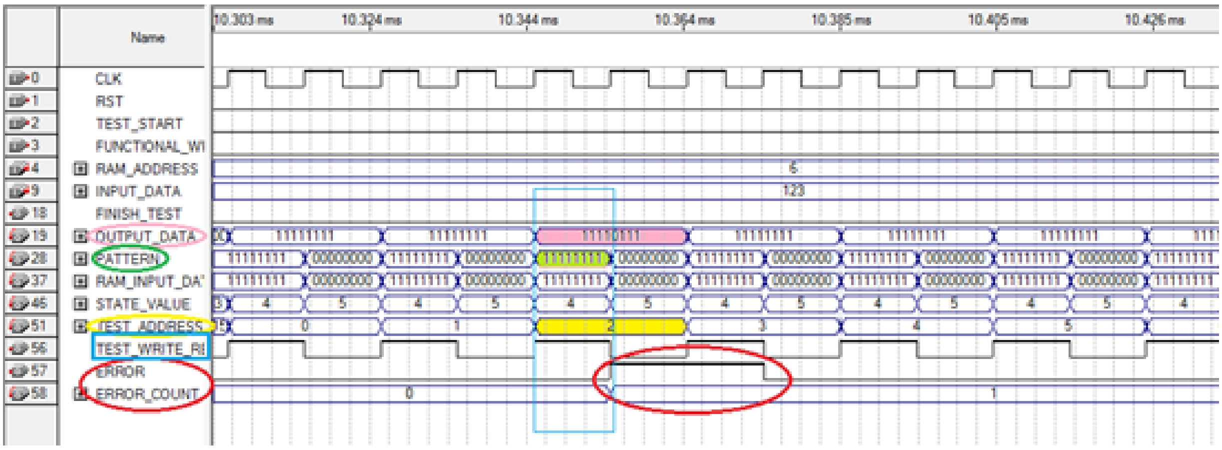


Figure 4.28: Simulation for testing CFin fault in SRAM 16 with FSM March

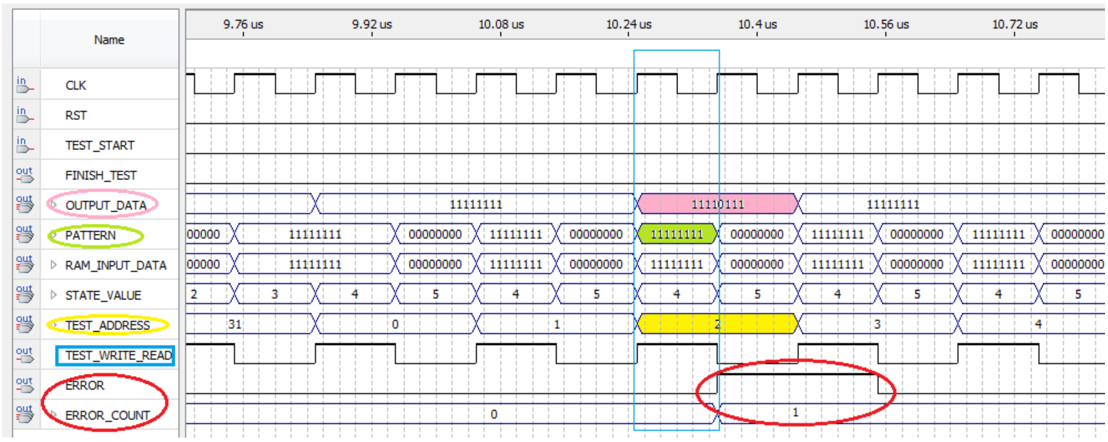


Figure 4.29: Simulation for testing CFin fault in SRAM 32 with FSM March

.

The simulation of testing the two memories are the same, and show that an error occurred (**ERROR=1, ERROR_COUNT=1)** at the address (**TEST_ADDRESS=2**) during memory read operation (**TEST_WRITE_READ=1**) where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not equal ("11110111"~="11111111").

Compared to a Fast March circuit which can detect only some cases of CFin faults, a FSM March circuit can detect all of them because it uses all possible combination of previous and new written values in all memory cells.

4.2.4.2 Simulation the test of Idempotent Coupling Faults (CFid)

The simulation results for testing CFid fault in SRAM 16 and SRAM 32 bit with Fast March circuit is shown in the Figure 4.30 and 4.31.

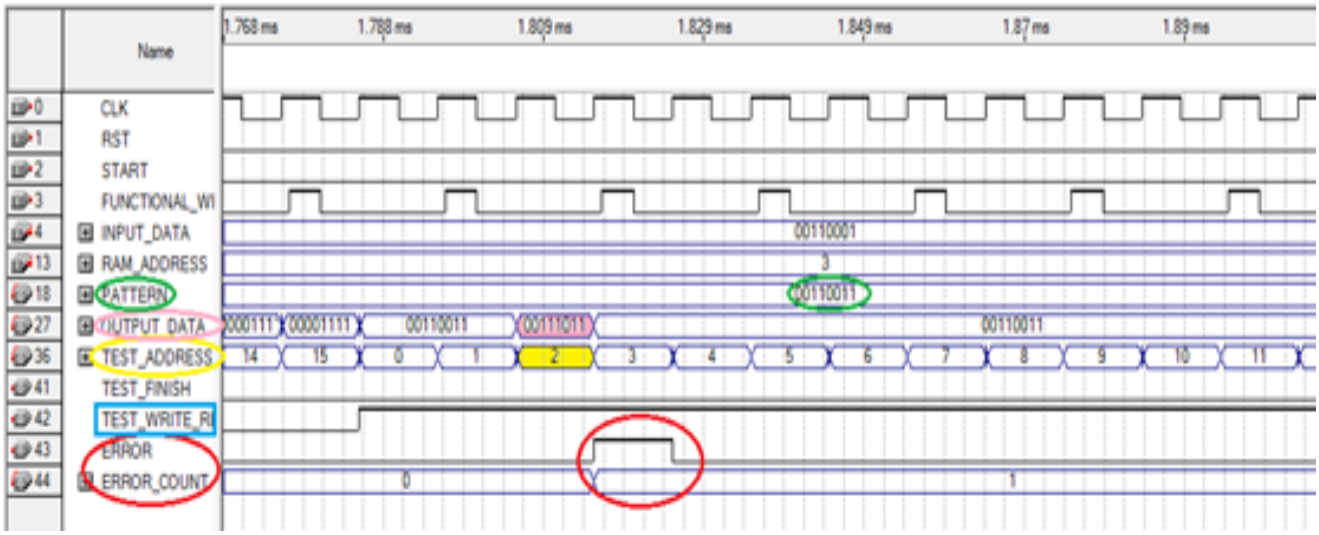


Figure 4.30: Simulation for testing CFid fault in SRAM 16 with Fast March

.

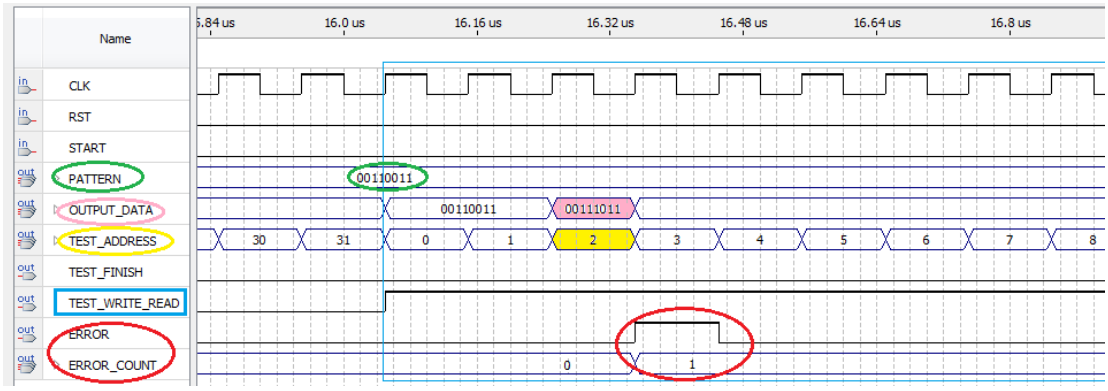


Figure 4.31: Simulation for testing CFid fault in SRAM 32 with Fast March

.

The results show that there is no difference in testing of the two memories, where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not equal ("00111011"~="00110011") at the faulty memory cell (**TEST_ADDRESS=2**) during memory read operation (**TEST_WRITE_READ=1**), which leads to an error (**ERROR=1, ERROR_COUNT=1)** at that address.

The simulation results for testing CFid fault in SRAM 16 and SRAM 32 bit with FSM March circuit are shown in the Figure 4.32 and 4.33

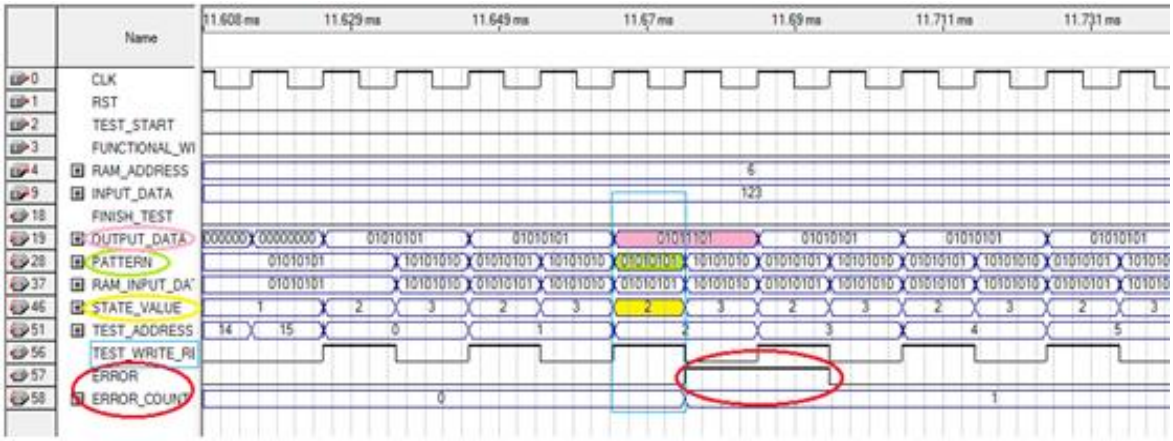


Figure 4.32: Simulation of testing CFid fault in SRAM 16 with FSM March

.

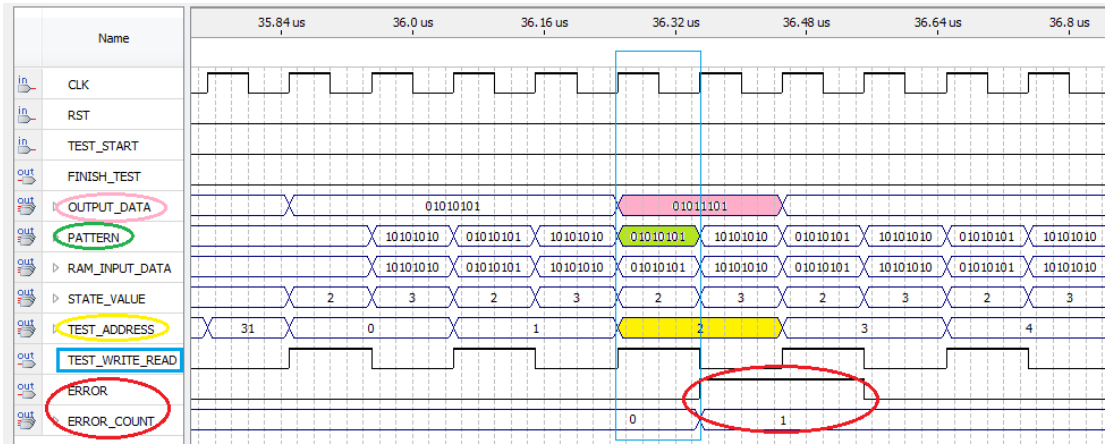


Figure 4.33:Simulation of testing CFid fault in SRAM 32 with FSM March

.

The testing results of the two memories are similar, and the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are not the same ("01011101"~="01010101") at the memory address (**TEST_ADDRESS=2**), which leads to an error (**ERROR=1, ERROR_COUNT=1)** at that address.

4.2.5 The Address Decoder Fault (AF) Model

Address decoder faults (AFs) are caused by address decoder errors. Address decoder errors are classified into three categories.

1) No cell is reached with a specific statement.

2) multiple cells are accessible at the same time using a specific address (one address to multi cells).

3) A specific cell can be reached using numerous addresses (multi addresses to one cell).

The first fault can be treated as Stack-at, but second and third faults can be modeled as the same fault by sharing multiple memory address with the same value.

The simulation result for testing AD fault in SRAM 16 and SRAM 32 bit with Fast March circuit is shown in the Figure 4.34 and 4.35 .

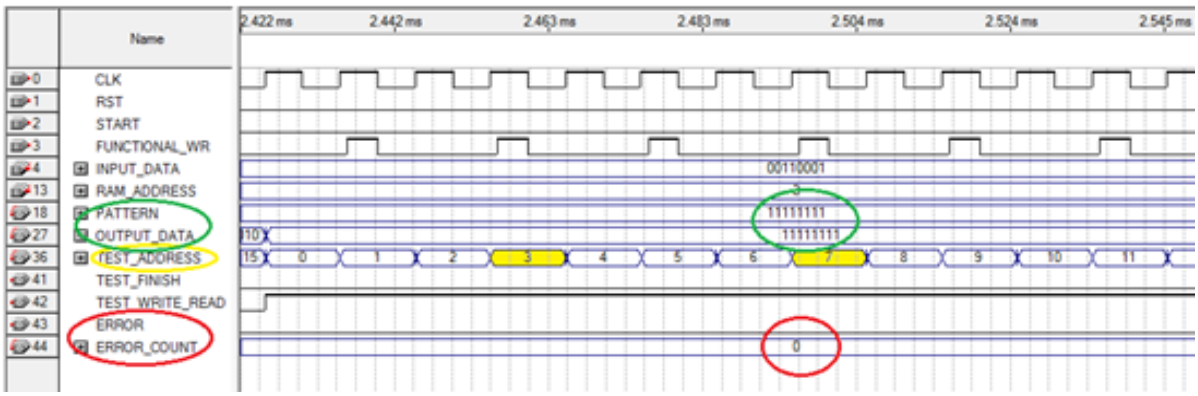


Figure 4.34: Simulation of testing AD fault in SRAM 16 with Fast March

.

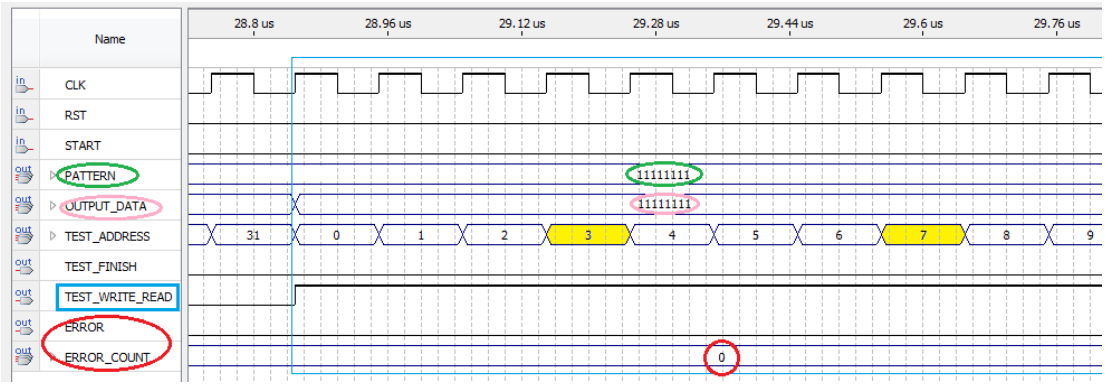


Figure 4.35: Simulation of testing AD fault in SRAM 32 with Fast March

.

As shown in Figures (4.36 and 4.39) that's testing of the two memories are the same, where the memory output data (**OUTPUT DATA**) and testing patterns **(PATTERN**) are the same at all memory locations (**TEST ADDRESS**) during memory read operation (**TEST WRITE READ=1**), so there are no errors occurred (**ERROR=0, ERROR COUNT=0**) at the faulty addresses **3** and **7**.

Therefore the Fast March algorithm can't detect AF faults. This is because a Fast March circuit does not read memory locations and compare them with the test pattern until finish of writing all memory locations with the same test pattern. Therefore, the fault of sharing several memory locations with the same value won't be detected because the same value is written in these memory locations before the reading and comparing processes.

This problem can be solved by making writing , reading and comparing process in each memory location independent from other locations as in FSM March circuit, in which a

reading and comparing written test pattern in each memory location made directly after writing process in each memory location.

The simulation results for testing AD fault in SRAM 16 and SRAM 32 bit with FSM March circuit are shown in the Figure 4.36 and 4.37 .

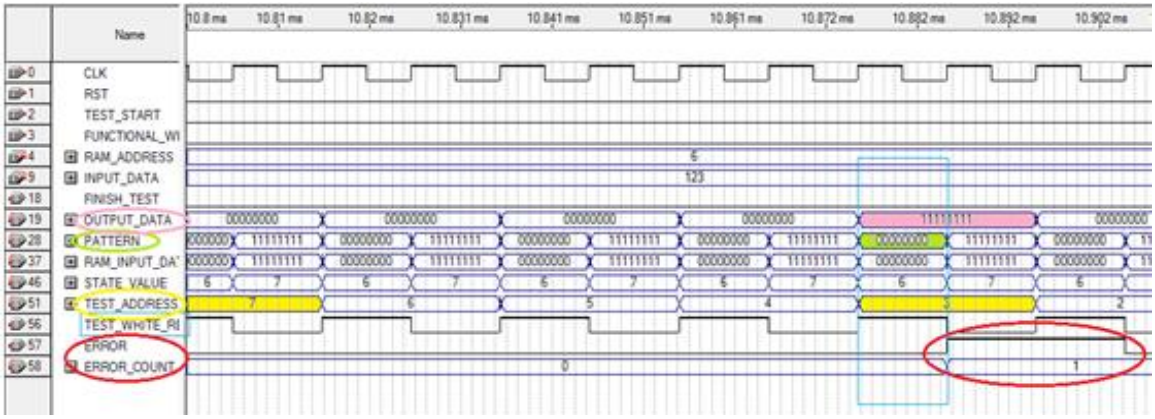


Figure 4.36: Simulation of testing AD fault in SRAM 16 with FSM Marc

h.

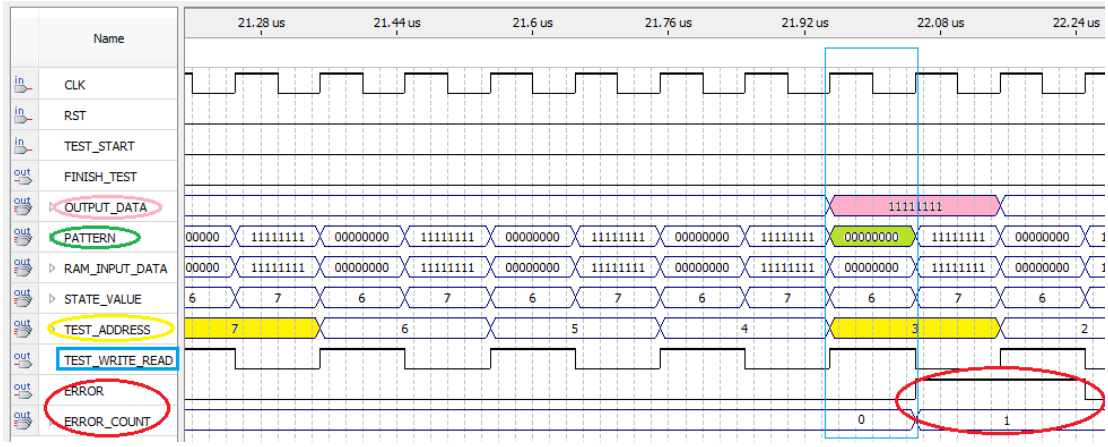


Figure 4.37: Simulation of testing AD fault in SRAM 32 with FSM March

.

It can be informed from the figures (4.38, 4.39) that memory size does not affect in fault detection, where the memory output data (**OUTPUT_DATA**) and testing patterns (**PATTERN**) are different ("11111111"~="00000000") at the address (***TEST_ADDRESS=3***)

during memory read operation (**TEST_WRITE_READ=1**), so there are an error occurred (**ERROR=1, ERROR_COUNT=*1)*** at that address.

4.3 Performance analyzes

The previous simulation results showed that the two proposed algorithms Fast March and FSM March working effectively in detecting memory faults, but the FSM March algorithm is better because it can detect all types of Coupling Fault (CF) and Address Decoder Fault (AF)**.**Where Fast March algorithm can detect some types of CF and can't detect AF. On the other hand the FSM March algorithm is slower than Fast March algorithm, because it is more complex and need more components .

To measure the testing time of both algorithms, two VHDL designs section (3.7) and (3.8) have been used to test SRAM 16 model (16x8 bit SRAM) with processing time 100 MHz clock frequency . Two simulation results have been obtained as shown in Figure 4.38 and Figure 4.39

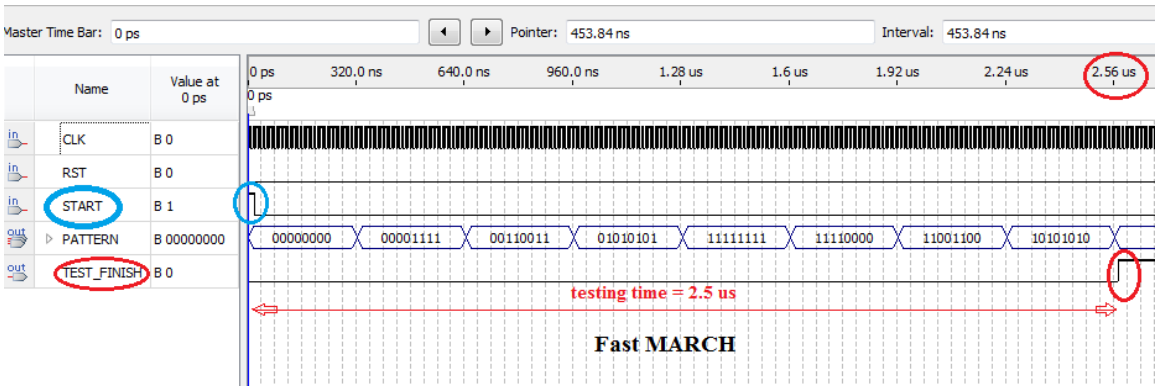


Figure 4.38: Testing time for SRAM 16 with Fast March

.

From Figure 4.38 it can be seen that the testing time for SRAM 16 with Fast March algorithm is 2.5 µs, while testing time for SRAM 16 with FSM March algorithm is 6.5 µs as shown in Figure 4.39.

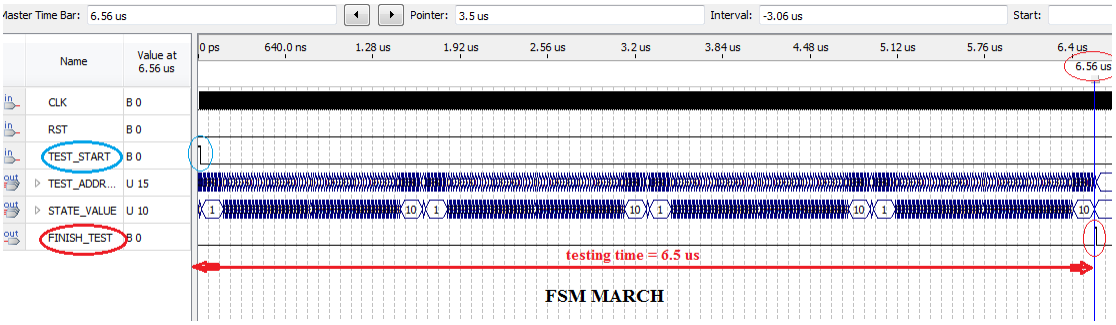


Figure 4.39: Testing time for SRAM 16 with FSM March

Secondly , two VHDL designs, section (3.9) and (3.10), have been used to test SRAM 32 model (32x8 bit SRAM) with 100 MHz clock frequency. Two simulation results are shown in Figure 4.40 and Figure 4.41.

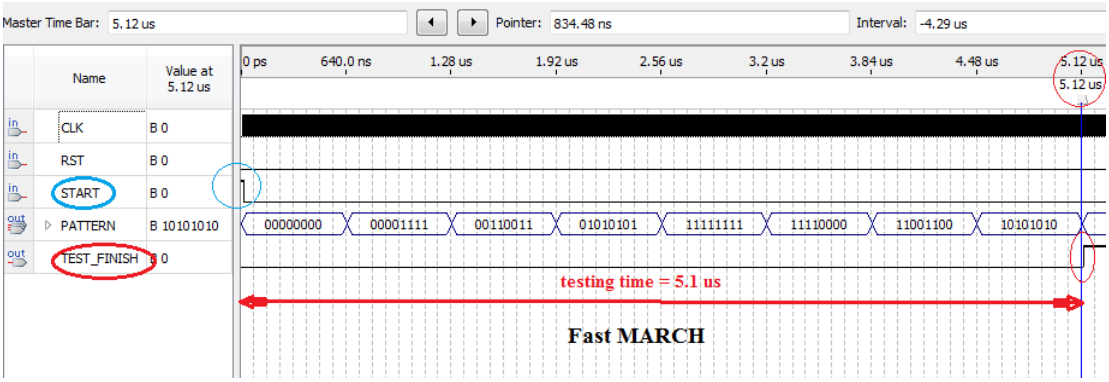


Figure 4.40: Testing time for SRAM 32 with Fast March

.

From Figure 4.40, it can be noted that the testing time for SRAM 32 with Fast March algorithm is 5.1 µs , while testing time for SRAM 32 with FSM March algorithm which is 13 µs as shown in Figure 4.41.

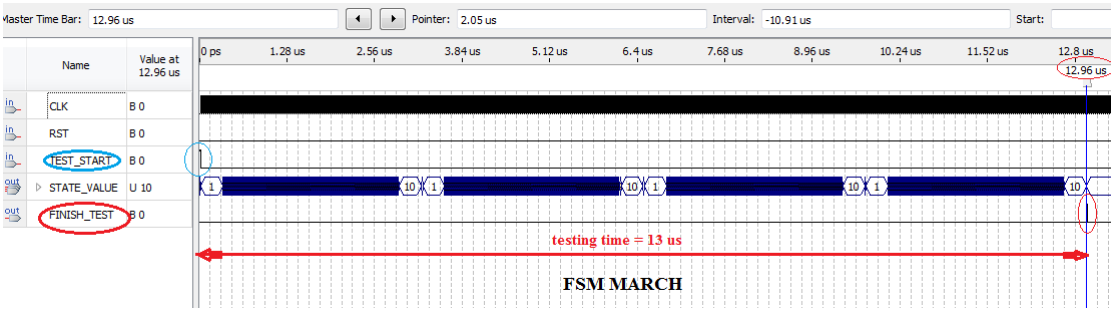


Figure 4.41: Testing time for SRAM 32 with FSM March

.

Also it can be noted from Figure 4.42 that the testing time of both algorithms affected by the size of memory. The increase of SRAM size leads to increase of testing time, which can be represented in the following curve:

Fast_March testing time = (2.5/16) x SRAM_Size.

FSM_March testing time = (6.5/16) x SRAM_Size.

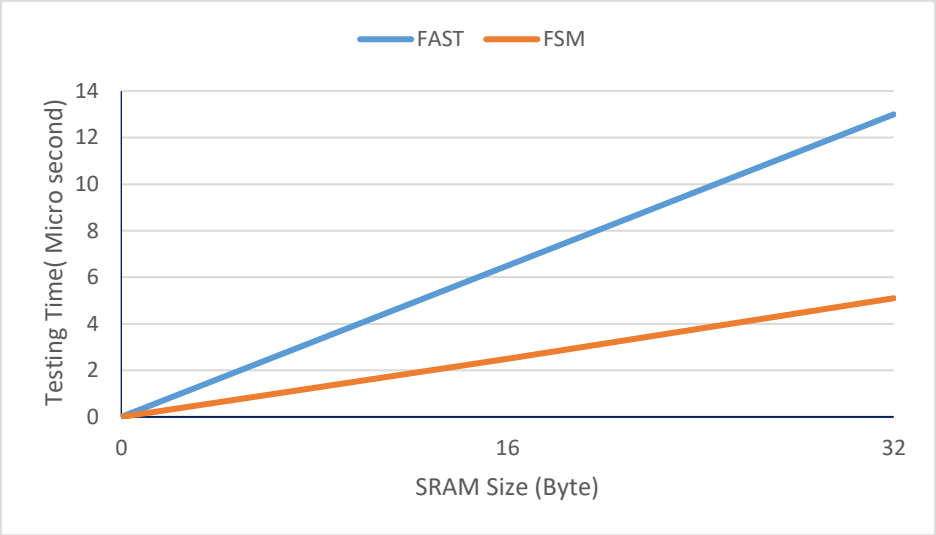


Figure 4.42: Testing time vs. SRAM size

In addition , from above simulation it can be seen that the FSM March algorithm is the most practical test method (from the proposed test algorithms) in terms of fault coverage. Table 4.2 summarizes the fault detection capabilities of March test algorithms in detail.

Table 4.2: Fault detection of March test algorithms

March algorithm	Detected Faults			
	SAF	AF	TF	CF
Fast March	Yes	No	Some	Yes
FSM March	Yes	Yes	Yes	Yes

All previous research in table (4.3) focused on the development of MBIST systems. Some of them were interested in developing a MBIST systems that targeting a specific type of

memories (bit-oriented) in order to make performance the best possible by using simple algorithms such as pseudorandom and checkerboard and zero-one algorithms which applied using simple architectures such as counters or LFSRs, but it can't be used with other types of memories and doesn't cover all types of memory faults. While others tried to develop a comprehensive MBIST system for all faults and memory types, by applying advanced algorithms such as March C- and MATS using complex architectures such as microprocessors and state machines programmed with hardware description languages (HDL), but the performance of this systems was low, especially with modern and large memories.

In proposed work, we will developed two MBIST algorithms Fast March and FSM March that combine comprehensiveness and performance and implementing them using VHDL language.and dedect all fault in two size of SRAM but more complexity.

Table 4.3: Compare related work

Related Work	Algorithm	RAM Size	Fault detected	CLK	Oriented type	Complexity
S.K.Jain, et al.[7]	Pseudorandom data pattern	32 bit	SAF,CF		Bit oriented memory	--------
N.Z.Haron, et al .[11]	MATS, MATS++, March X, March C, March C-	16 bit	SAF	20MHz	Bit oriented memory	-------

Related Work	Algorithm	RAM Size	Fault detected	CLK	Oriented type	Complexity
Masnita ,et al.[8]	MATS++, March C-	---------	SAF , TF	--------	Bit oriented memory	5N
Reeja J ,et al [3]	March C-	8 bit	SAF, TF	--------	Bit oriented memory	------
M.A Ahmed ,et al [18]	March 17N	32bit	Semiconductor fault	497.47 MHz	Bit oriented memory	17N
T.Koshy,etal [25]	MATS, March C , March X	32 bit	SAF,CF, TF	Testing time 3,092.44 ns	Bit oriented memory	---------
p.K.John [21]	March C-	16 bit	SAF,AF	-------	Bit oriented memory	------

Related Work	Algorithm	RAM Size	Fault detected	CLK	Oriented type	Complexity
T.Tewary, et al [23]	Linear Feedback Shift Register	8 bit	No detected any fault	345.173 MHz	Bit oriented memory	--------
Proposed (1)	**Fast March**	**16 bit , 32bit**	**SAF, CF some TF**	**100 MHz**	**Word oriented memory**	**16N**
Proposed (2)	**FSM March**	**16 bit ,32 bit**	**SAF,AF, CF,TF**	**100MHz**	**Word oriented memory**	**40N**

CHAPTER 5

5. Chapter Five

Conclusions and Future Work

5.1 Conclusions

After the proposed system has been designed and implemented, several points have been concluded can be explained in the following paragraphs:

1- The simulation results have shown that the Fast March algorithm is faster than FSM March algorithm. The test time for 16x8 SRAM using Fast March algorithm was 2.5 μs while the test time was 6.5 μs using FSM March algorithm for the same SRAM. For SRAM (32x8), the test time using Fast March was 5 μs while it was 13 μs for FSM March.

2- In terms of fault coverage, Fast March algorithm has detected Stuck-At- Faults , Coupling- Fault and some of Transition- Faults but it didn't detect the address -fault. Therefore, it can be used in low cost applications that need a small amount of recourses. On other hand, FSM March algorithm was slower than Fast March algorithm, but it was more reliable and has detected all types of memory faults.

3- The benefits of the proposed algorithms Fast March and FSM March:

- Working perfectly in testing modern complex word-oriented memories.
- Designed and Implemented in generic manner using VHDL language, so they can be modified and applied on wide range of devices for testing wide range of memories.
- Simple because they using minimum number of March elements but they can detect wide range of memory faults.
- Working faster than Microcode based MBIST, because they work directly, and don't have a processor complexity and delay (Sequential execution of instructions).
- Robust and fast so they don't need a compression technique to enhance performance.

5.2 Future Work

1. Designing and implementing a model for all embedded memory faults using VHDL language, to be used in testing and comparing the designed MBIST systems.
2. Adding a self-repair mechanism to the embedded memory in addition to the MBIST system, because performing a self-test of the embedded memory without the possibility of fixing the detected faults is useless in practical applications.
3. One of the directions for future work is a co optimization study in which soft error protection, packaging density and speed for a verity cell architectures are analyzed to increase the memory fabrication yield.
4. Test techniques that are currently used, cover the static faults, while dynamic faults can also appear in SRAM, when a fast read after write operations take place. Therefore, detecting these faults is also a major task for researchers.

References

References

[1] J. Patil, S. Pujari, and A. Shaligram, "Designing, Modeling and Implementation of Memory Built In Self Test (BIST) Controller," in *Proc. On Int. Conference on Control, Communication and Power Engineering*, 2010, pp. 359–361.

[2] T. Tewary and A. Sen, "A novel approach to realize built-in-self-test (BIST) enabled UART using VHDL," in *2014 First International Conference on Automation, Control, Energy and Systems (ACES)*, 2014, pp. 1–6.

[3] M. Husin, S. Leong, M. Sabri, and R. Nordiana, "Built in self test for RAM Using VHDL," in *2012 IEEE Colloquium on Humanities, Science and Engineering (CHUSER)*, 2012, pp. 272–276.

[4] K. Zarrineh and S. J. Upadhyaya, "On programmable memory built-in self test architectures," in *Proceedings of the conference on Design, automation and test in Europe*, 1999, pp. 136-es.

[5] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, "Enabling embedded memory diagnosis via test response compression," in *Proceedings 19th IEEE VLSI Test Symposium. VTS 2001*, 2001, pp. 292–298.

[6] P. J. Ashenden, *The Designer's Guide to VHDL Second Edition*. Elsevier Science (USA), 2002.

[7] S. Jain and C. Stroud, "Built-in Self Testing of Embedded Memories," *IEEE Des. Test. Comput.*, vol. 3, no. 5, pp. 27–37, 1986, doi: 10.1109/MDT.1986.295041.

[8] M. I. Masnita, W. H. W. Zuha, R. M. Sidek, and A. H. Izhal, "March-based SRAM diagnostic algorithm for distinguishing Stuck-At and transition faults," *IEICE Electron. Express*, vol. 6, no. 15, pp. 1091–1097, 2009, doi: 10.1587/elex.6.1091.

[9] R. Venkatesh, S. Kumar, J. Philip, and S. Shukla, "A fault modeling technique to test memory BIST algorithms," in *Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002)*, Isle of Bendor, France, 2002, pp. 109–116.

[10] Q. Wei Chun, P. Wai Leong, C. Kah Yoong, L. It Ee, and C. Gwo Chin, "VHDL Modelling of Low-CostMemory Fault Detection Tester," *JETAP*, vol. 2, no. 2, pp. 17–23, Dec. 2020.

[11] N. Z. Haron, S. A. Mat Junos, and A. S. Abdul Aziz, "Modeling and simulation of microcode Memory Built In Self Test architecture for embedded memories," in *2007 International Symposium on Communications and Information Technologies*, Sydney, NSW, Oct. 2007, pp. 136–139.

[12] M. I. Masnita, W. H. W. Zuha, R. M. Sidek, and I. A. Halin, "The data and read/write controller for March-based SRAM diagnostic algorithm MBIST," in *2009 IEEE Student Conference on Research and Development (SCOReD)*, UPM Serdang, Malaysia, 2009, pp. 296–299.

[13] A. J. van de Goor and I. B. S. Tlili, "March tests for word-oriented memories," in *Proceedings Design, Automation and Test in Europe*, Paris, France, 1998, pp. 501–508.

[14] I. Bayraktaroglu, O. Caty, and Yickkei Wong, "Highly Configurable Programmable Built-In Self Test Architecture for High-Speed Memories," in *23rd IEEE VLSI Test Symposium (VTS'05)*, Palm Springs, CA, USA, 2005, pp. 21–26.

[15] P. E. Joseph and P. R. Antony, "VLSI design and comparative analysis of memory BIST controllers," in *2014 First International Conference on Computational Systems and Communications (ICCSC)*, Trivandrum, India, Dec. 2014, pp. 372–376.

[16] Q. Liyan, B. Shi, Z. Xin, and W. Ge, "Design of generic embedded memory built in self test circuit," in *2009 9th International Conference on Electronic Measurement & Instruments*, Beijing, China, Aug. 2009, pp. 2-244-2–247.

[17] R. K. Sharma and A. Sood, “Modeling and Simulation of Multi-operation Microcode-Based Built-In Self Test for Memory Faults,” in *2010 International Conference on Signal Acquisition and Processing*, Bangalore, India, Feb. 2010, pp. 8–12.
[18] M. A. Ahmed, D. Elizabeth Rani, and S. A. Sattar, “FPGA based High Speed Memory BIST Controller for Embedded Applications,” *Indian Journal of Science and Technology*, vol. 8, no. 33, Dec. 2015.
[19] Chih-Sheng Hou, Jin-Fu Li, and Ting-Jun Fu, “A BIST Scheme With the Ability of Diagnostic Data Compression for RAMs,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 33, no. 12, pp. 2020–2024, Dec. 2014.
[20] N. Z. Haron, S. A. M. Junos, A. H. A. Razak, and Mohd. Y. I. Idris, “Modeling and simulation of finite state machine Memory Built-in Self Test architecture for embedded memories,” in *2007 Asia-Pacific Conference on Applied Electromagnetics*, Melaka, Malaysia, Dec. 2007, pp. 1–5.
[21] N. Maneshinde, P. Hegade, R. Mittal, N. Palecha, and M. S. Suma, “Programmable FSM based built-in-self-test for memory,” in *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, India, May 2016, pp. 194–199.
[22] N. Q. M. Noor, Y. Yusof, and A. Saparon, “Low area FSM-based memory BIST for synchronous SRAM,” in *2009 5th International Colloquium on Signal Processing & Its Applications*, Kuala Lumpur, Malaysia, Mar. 2009, pp. 409–412.
[23] P. K. John and R. Antony P, “BIST Architecture for Multiple RAMs in SoC,” *Procedia Computer Science*, vol. 115, pp. 159–165, 2017.
[24] T. Tewary, S. Dey, and S. Roy, “Realization of Built-In Self Test(BIST) Enabled Memory(RAM) Using VHDL and Implementation in Spartan6 FPGA board,” in *2020 IEEE VLSI DEVICE CIRCUIT AND SYSTEM (VLSI DCS)*, Kolkata, India, Jul. 2020, pp. 322–326.
[25] T. Koshy and C. S. Arun, “Diagnostic data detection of faults in RAM using different march algorithms with BIST scheme,” in *2016 International Conference on Emerging Technological Trends (ICETT)*, Kollam, India, Oct. 2016, pp. 1–6.
[26] J. Reeja and L. Anusree, “Design of Built-in Self-Test Core for SRAM,” *International Journal of Engineering Research*, vol. 3, no. 3, 2014.
[27] S. Hamdioui and A. J. van de Goor, “Efficient tests for realistic faults in dual-port SRAMs,” *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 460–473, May 2002.
[28] M. Sasikumar, R. Bhakthavatchalu, K. N. Sreehari, and A. S. Kumar, “Scalable and Rapid Fault Detection of Memories Using MBIST and Signature Analysis,” in *Advances in Signal and Data Processing*, vol. 703, S. N. Merchant, K. Warhade, and D. Adhikari, Eds. Singapore: Springer Singapore, 2021, pp. 351–367.
[29] A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, and A. Virazel, “Advanced test methods for SRAMs-Effective Solutions for Dynamic Fault Detection,” *Nanoscaled Technologies*, 2010.
[30] M. Barr, “Memory types,” *Embedded Systems Programming*, vol. 14, no. 5, pp. 103–104, 2001.
[31] F. Masuoka, M. Asano, H. Iwahashi, T. Komuro, and S. Tanaka, “A new flash E^2PROM cell using triple polysilicon technology,” in *1984 International Electron Devices Meeting*, 1984, pp. 464–467.
[32] R. D. Adams, *High performance memory testing: design principles, fault modeling and self-test*, vol. 22. Springer Science & Business Media, 2002.
[33] I. Mrozek, *Multi-run Memory Tests for Pattern Sensitive Faults*, 1st ed. 2019. Cham: Springer International Publishing : Imprint: Springer, 2019.

[34] H. G. Cragon, “Memory Systems,” in *Memory systems and pipelined processors*, Sudbury, Mass: Jones and Bartlett, 1996.
[35] J. Singh, S. P. Mohanty, and D. K. Pradhan, *Robust SRAM designs and analysis*. Springer Science & Business Media, 2012.
[36] B. H. Calhoun and A. P. Chandrakasan, “A 256-kb 65-nm Sub-threshold SRAM Design for Ultra-Low-Voltage Operation,” *IEEE J. Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, Mar. 2007.
[37] C. Shin, *Variation-Aware Advanced CMOS Devices and SRAM*, vol. 56. Dordrecht: Springer Netherlands, 2016. doi: 10.1007/978-94-017-7597-7.
[38] K. Kim, H. Mahmoodi, and K. Roy, “A Low-Power SRAM Using Bit-Line Charge-Recycling,” *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 446–459, Feb. 2008.
[39] K. Kim, J. B. Kuang, F. H. Gebara, H. C. Ngo, C.-T. Chuang, and K. J. Nowka, “TCAD/Physics-Based Analysis of High-Density Dual-BOX FD/SOI SRAM Cell With Improved Stability,” *IEEE Trans. Electron Devices*, vol. 56, no. 12, pp. 3033–3040, Dec. 2009.
[40] S. Irobi, Z. Al-Ars, and S. Hamdioui, “Bit line coupling memory tests for single-cell fails in SRAMs,” in *2010 28th VLSI Test Symposium (VTS)*, Santa Cruz, CA, USA, Apr. 2010, pp. 27–32.
[41] C. Shin, “Advanced MOSFET Designs and Implications for SRAM Scaling,” UC Berkeley, 2011. Accessed: May 14, 2022. [Online]. Available: https://escholarship.org/uc/item/2nt3f2dn
[42] Ludwig *et al.*, “FinFET technology for future microprocessors,” in *2003 IEEE International Conference on Robotics and Automation (Cat No 03CH37422) SOI-03)*, Newport Beach, CA, USA, 2003, pp. 33–34.
[43] T. Hirose *et al.*, “A 20 ns 4 Mb CMOS SRAM with hierarchical word decoding architecture,” in *1990 37th IEEE International Conference on Solid-State Circuits*, San Francisco, CA, USA, 1990, pp. 132–133.
[44] M. Yoshimoto *et al.*, “A divided word-line structure in the static RAM and its application to a 64K full CMOS RAM,” *IEEE J. Solid-State Circuits*, vol. 18, no. 5, pp. 479–485, Oct. 1983.
[45] C. E. Stroud, *A designer’s guide to built-in self-test*. Boston: Kluwer Academic Publishers, 2002.
[46] I. D. Dear, C. Dislis, A. P. Ambler, and J. Dick, “Economic effects in design and test,” *IEEE Des. Test. Comput.*, vol. 8, no. 4, pp. 64–77, Dec. 1991.
[47] Z. Navabi, *Digital System Test and Testable Design_ Using HDL Models and Architectures*. SPRINGER-VERLAG NEW YORK, 2016.
[48] V. Agrawal, C. Kime, and K. Saluja, “KA tutorial on built-in self-test, part 1: Principles,” *Design & Test of Computers, IEEE*, vol. 10, no. 1, pp. 73–82, 1993.
[49] D. L. Wheater, P. Nigh, J. T. Mechler, and L. Lacroix, “ASIC test cost/strategy trade-offs,” in *Proceedings., International Test Conference*, Washington, DC, USA, 1994, pp. 93–102.
[50] McCluskey, “Verification Testing—A Pseudoexhaustive Test Technique,” *IEEE Trans. Comput.*, vol. C–33, no. 6, pp. 541–546, Jun. 1984.
[51] H.-P. Klug, “Microprocessor testing by instruction sequences derived from random patterns,” in *International Test Conference 1988 Proceeding@m_New Frontiers in Testing*, Washington, DC, USA, 1988, pp. 73–80.
[52] M. L. Bushnell and V. D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Boston: Kluwer Academic, 2000.
[53] A. J. Van de Goor, *Testing semiconductor memories: theory and practice*. John Wiley & Sons, Inc., 1991.

[54] A. Chandra and K. Chakrabarty, “Test resource partitioning for SOCs,” *IEEE Des. Test. Comput.*, vol. 18, no. 5, pp. 80–91, Oct. 2001.
[55] Nair, “Comments on ‘An Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories,’” *IEEE Trans. Comput.*, vol. C–28, no. 3, pp. 258–261, Mar. 1979.
[56] M. S. Abadir and H. K. Reghbati, “Functional Testing of Semiconductor Random Access Memories,” *ACM Comput. Surv.*, vol. 15, no. 3, pp. 175–198, Sep. 1983.

الخلاصة

أصبح الاختبار الذاتي المدمج (BIST) عاملاً مهمًا في عمليات التصنيع الحالية للذاكرة. ويستخدم في زيادة الموثوقية وتحسين إنتاجية الذاكرة. تعتمد فعالية أي خوارزمية اختبار على تغطية الخطأ وسرعة اكتشاف الخطأ. في هذا الاطروحة ، تم اقتراح طريقتين BIST لذاكرة الوصول العشوائي الثابتة (SRAM). النهج الأول هو خوارزمية Fast March بينما الأسلوب الثاني هو خوارزمية March-C. في النهج الأول ، تم تطوير خوارزمية الصفر واحد التقليدية لخوارزمية Fast March باستخدام طريقة توجيه الكلمات التي تم فيها استخدام خطوات مختلفة لنمط الاختبار. في النهج الثاني ، تم تطوير خوارزمية March- C التقليدية لخوارزمية آلة الحالة المحدودة (FSM) حيث أصبحت تعتمد على توجيه تقنية الكلمات باستخدام نمط اختبار مناسب. تم تطبيق النهجين على حجمين من الذاكرة (16 × 8) بت لكلمة و (32 × 8) كلمة بت حيث تم افتراض أربعة أنواع من الأخطاء للتحقق من فعالية هذه الخوارزميات. الاخطاء هي خطأ العنوان ، عالق في الخطأ ، خطأ اقتران وخطأ في النقل.

لقد أثبتت نتائج المحاكاة أن Fast March أسرع من خوارزمية March- C ثلاث مرات (16 عملية مقابل 40 عملية) ، لكنها لا تكشف سوى نوع واحد من الأخطاء (عالق عند الخطأ). تعد خوارزمية March -C أبطأ من Fast March ، ولكنها أكثر موثوقية ويمكنها اكتشاف معظم أخطاء الذاكرة. تم إجراء المحاكاة باستخدام برنامج QUARTZ 13.0 حيث تم استخدام لغة VHDL في هذا العمل.

جمهورية العـراق

وزارة التعليم العالي والبحث العلمي

جامعــة بابـل

كليـــــة الهندســـة

نهج الاختبار الذاتي الداخلي لمجموعة ذاكرة الوصول العشوائي الثابتة

قـدمت هـذا الرسالة إلى قسم الهندسة الكهربائية – كلية الهندسة – جامعة بابل كجـزء من متطلبات الحصول على درجــة الماجستير في الهندسة الكهربائية الهندسة / الهندسة الكهربائية/ الكترونيك صناعي

قــدمت من قبـل :

فنار عباس عبدالله حسن

إشــراف

أ.د. قيس كريم عمران

1443هـ 2022 م