FSM-BASED ENHANCED MARCH C- ALGORITHM FOR MEMORY BUILT-IN SELF-TEST

BY

CH'NG MING ZONG

A DISSERTATION SUBMITTED FOR PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE

(MICROELECTRONIC ENGINEERING)

AUGUST 2017

Acknowledgement

Firstly, I would like to express my greatest appreciation to my supervisor, Dr. Zulfiqar Ali Abdul Aziz. He willingly to spend his time on supervising me with pertinent comment and guidance. With the guidance and advice given throughout the research period had enabled me to complete the dissertation without any difficulties.

Besides that, I would also like to thank my family members to provide all the support that I had been needed while I was researching. They had given a lot of assistance that enable me to fully focus on my research works.

Last but not least, I would like to take this opportunity to thanks all of my friends that showed their support. They kindly provide suggestions and advice when I was in doubts of the facts or methodologies. Without their support and encouragement, this researching process will not be as easy as it was.

PENAMBAHBAIKAN MARCH C- ALGORITHMA MENGGUNAKAN ASAS FSM UNTUK MEMORI BIST

Abstrak

Algoritma memain peranan yang penting dalam Memori BIST dimana struktur algoritmanya berupaya mentakrif pengesan liputan kesalahan di dalam sesuatu sistem. Oleh yang demikian, jika penambahbaikan algoritma itu dibuat, ia membenarkan lebih banyak jenis kesalahan dapat dikesan walau hanya dengan satu ujian sahaja. Bagi algorithma MARCH C-, ia merupakan salah satu algoritma stabil yang mempunyai liputan jenis kesalahan yang tinggi kerana ia mampu mengesan banyak kesalahan seperti Address Decoder Fault, Stuck-At Fault, Transition Fault dan Coupling Fault namun begitu ia tidak berupaya untuk mengesan jenis kesalahan seperti Read Destructive Fault dan Data Retention Fault. Bagi meningkatkan prestasi algoritma MARCH C- untuk mengesan dua jenis kesalahan tersebut, beberapa pengubahsuaian perlu dilakukan. Melalui hasil analisa corak-corak kesalahan dan jenis algoritma lain, dua metodologi bagi penambahbaikan algorithma MARCH C- telah dikesan. Dengan menggunakan operasi pembacaan yang berganda dalam elemen tunggal MARCH yang terdapat didalam algoritma MARCH-NU boleh membantu untuk mengesan Read Destructive Fault manakala penggunaan HOLD time dalam algoritma MARCH 9N mampu mengesan Data Retention Fault. Melalui gabungan kedua-dua cara tersebut dalam algoritma MARCH C- membolehkannya untuk mengesan kesalahan Data Retention Fault dan Read Destructive Fault secara 100% tanpa menjejaskan prestasi asal MARCH C-. Selain daripada itu, penambahbaikan algoritma MARCH C- juga boleh mengesan jenis-jenis kesalahan secara struktur dan mengasingkan kesalahan

diantara kesalahan tingkahku (Data Retention Faults) dan kesalahan umum memori serta dianatara kesalahan Stuck-At Fault dan Transition Fault. Fault Injection test telah dijalankan bagi memastikan Enhanced MARCH C- boleh mengenal pasti corak-corak kesalahan. Enhanced MARCH C- mempunyai kadar lulus 100% dalam mengenal pasti Stuck-At Fault, Transition Fault, Data Retention Fault, Inversion Read Fault, Incorrect Read Fault dan Read Destructive Fault. Ia mempunyai kadar 95.16% dalam mengenal pasti State Coupling Fault dan Idempotent Coupling Fault. Sebagai konklusi, Enhanced MARCH C- algorithm telah berjaya meningkatkan liputan jenis kesalahan dalam Data Retention Fault dan Read Destructive Fault manakala ia juga mempunyai abiliti untuk mengenal pasti jenis kesalahan secara struktur.

FSM-BASED ENHANCED MARCH C- ALGORITHM FOR MEMORY BUILT-IN SELF-TEST

Abstract

Algorithms plays an important role in the Memory Built-In Self-Test as its structure will define the fault coverage of the system. Thus, the improvement of the algorithms will allow more fault types being identified in single test. For MARCH C- algorithm, it was a quite balance algorithm as it has the ability to cover Address Decoder Fault, Stuck-At Fault, Transition Fault and Coupling Fault but it was not able to identify Read Destructive Fault and Data Retention Fault. To increase the fault coverage of the MARCH C- algorithm on that two fault types, some enhancements were needed to make on the algorithm. Through the analysis of the fault patterns and other algorithm, it was found that multiple read operation within a single MARCH element from MARCH-NU algorithm can aid in the identification of the Read Destructive Fault whereas having HOLD time in the MARCH 9N algorithm would expose the Data Retention Fault. By integrating both new fault identification methodologies into the MARCH C- algorithm and enhance it so that it would able to increase the identification of the Data Retention Fault and Read Destructive Fault by 100% and without causing any performance to interfere with the original fault identification ability. Besides that, the enhanced MARCH C- algorithm can structurally identify the fault types and differentiate among the behavioural faults (Data Retention Faults) and common memory faults as well as among Stuck-At Fault and Transition Fault. Fault injection test were carried out to ensure the coverage of the enhanced MARCH C- algorithm. It was having high passing rate which are 100%

identification of the Stuck-At Fault, Transition Fault, Data Retention Fault, Inversion Read Fault, Incorrect Read Fault and Read Destructive Fault. It also obtained 95.16% for the State Coupling Fault and Idempotent Coupling Fault. As a conclusion, the enhanced MARCH C- algorithm had increased its fault type identification in Data Retention Fault and Read Destructive Fault while having the ability to structurally identify the fault types.

CONTENTS

Acknowledgement i

Abstrak ii

Abstract iv

CONTENTS vi

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATION xiii

CHAPTER 1 INTRODUCTION 1

1.1 Background 1

1.2 Problem Statement 4

1.3 Research Objectives 5

1.4 Research Scope 5

1.5 Thesis Outline 6

CHAPTER 2 LITERATURE REVIEW 7

2.1 Introduction 7

2.2 Memory Types (DRAM, SRAM) 7

2.3 Faults 11

2.4 MARCH algorithms 18

2.5 Finite State Machine (FSM) Methodology and Microcode Methodology 28

2.6 Summary 30

CHAPTER 3 DESIGN METHODOLOGY 32

3.1 Introduction 32

3.2 Analysis of previous MARCH algorithms 33

3.3 Proposed Algorithm: Enhanced MARCH C- Algorithm 35

3.3.1 Detail Analysis of Enhanced MARCH C- algorithm 38

3.4 Proposed Memory BIST system 46

3.4.1 Block Diagram and Functions 47

3.4.2 Structure of proposed MBIST system 49

3.4.3 Flow Chart of the MBIST system 55

3.5 Performance Evaluation of the Enhanced MARCH C- algorithm 57

3.6 Summary 58

CHAPTER 4 RESULTS AND DISCUSSIONS 59

4.1 Introduction 59

4.2 Fault Coverage Test of Enhanced MARCH C- algorithm 59
4.2.1 Data Retention Fault (DRF) Test 61
4.2.2 Stuck-At Fault (SAF) Test 64
4.2.3 Transition Fault (TF) Test 67
4.2.4 State Coupling Fault (CFst) Test 71
4.2.5 Idempotent Coupling Fault (CFid) Test 76
4.2.6 Inversion Read Fault (CFin) Test 80
4.2.7 Incorrect Read Fault (CFir) Test 83
4.2.8 Read Destructive Fault (CFrd) Test 87
4.3 Comparison Analysis of original MARCH C- and Enhanced MARCH C- algorithm 91
4.3.1 Data Retention Fault test for original MARCH C- algorithm 91
4.3.2 Read Destructive Fault test for original MARCH C- algorithm 93
4.4 MBIST System Execution Time Test 96
4.4.1 Execution time test for different RAM size in MBIST system with enhanced MARCH C- algorithm 97
4.4.2 Execution time test between enhanced MARCH C- algorithm and original MARCH C- algorithm 100
4.5 Summary 102
Chapter 5 Conclusion and Future Improvements 105
5.1 Conclusion 105
5.2 Future Improvements 106
REFERENCE 108
APPENDICES I
APPENDIX A: Fault Free MBIST System Full Waveform View I
APPENDIX B: Fault Free RAM Response Overview II
APPENDIX C: Full Waveform Overview for Data Retention Fault Test III
APPENDIX D: Full Waveform Overview for Stuck-At Fault Test (SAF 1) IV
APPENDIX E: Full Waveform Overview for Transition Fault Test (DTF) V
APPENDIX F: Full Waveform Overview for State Coupling Fault Test (CFst (1,1) Victim Cell > Aggressor Cell) VI
APPENDIX G: Full Waveform Overview for Idempotent Coupling Fault (CFid (U,1) Victim Cell > Aggressor Cell) VII
APPENDIX H: Full Waveform Overview for Inversion Read Fault (CFin (U, ~) Victim Cell > Aggressor Cell) VIII
APPENDIX I: Full Waveform Overview for Incorrect Read Fault (CFir (1, -) Victim Cell > Aggressor Cell) IX

APPENDIX J: Full Waveform Overview for Read Destructive Fault (CFrd (1, ~) Victim Cell > Aggressor Cell) X

APPENDIX K: Full Waveform of MBIST with original MARCH C- algorithm test on Data Retention Fault XI

APPENDIX L: Full Waveform of MBIST with original MARCH C- algorithm test on Read Destructive Fault (CFrd (1, ~) Victim Cell > Aggressor Cell) XII

APPENDIX M: Full Waveform of MBIST executing RAM size of 8 words XIII

APPENDIX N: Full Waveform of MBIST executing RAM size of 16 words XIV

APPENDIX O: CTOP Module Coding XV

APPENDIX P: FAULTY_RAM Module Coding XVI

APPENDIX Q: Comparator Module Coding..... XIX

APPENDIX R: TPG Module Coding XXV

APPENDIX S: Fault_Type_Indicator Module Coding..... XXXI

LIST OF TABLES

Table 2-1: Table of MARCH Operations and its functions. 21
Table 2-2: MARCH algorithms and Fault Coverage 22
Table 2-3: Enhanced MARCH C- algorithm and its strength 27
Table 2-4: Comparison of FSM-based microcontroller and Microcode-based microcontroller 29
Table 3-1: Fault Signature for Data Retention Fault, Stuck-At Fault, Transition Fault 41
Table 3-2: Fault Signature for State Coupling Fault (CFst), Idempotent Coupling Fault (CFid) and Inversion Coupling Fault (CFin) 42
Table 3-3: Fault Signature for Incorrect Read Fault (CFir) and Read Destructive Fault (CFrd) 43
Table 3-4: Summary of Fault Signature of Enhanced MARCH C- algorithm 45
Table 4-1: Result Summary for Section 4.2 Fault Coverage Test 102
Table 4-2: Comparison table between original MARCH C- and enhanced MARCH C- algorithm 103

LIST OF FIGURES

Figure 1-1: Memory area in SoC as per year growth [3] 1
Figure 1-2: Percentage of Faults found in RAM [5] 2
Figure 2-1: Normal Memory Cells 11
Figure 2-2: Stuck-At 0 Fault 12
Figure 2-3: Stuck-At 1 Fault 12
Figure 2-4: Transition (⇑, 0) Fault 12
Figure 2-5: Example of Address Decoder Fault (ADF) [16] 13
Figure 2-6: Example of State Coupling Fault (CFst (0,1)) 14
Figure 2-7: Example of Idempotent Coupling Fault (CFid (U,0)) 15
Figure 2-8: Example of Inversion Coupling Fault (CFin (D, ~)) 16
Figure 2-9: Example of Incorrect Read Fault (CFir) 16
Figure 2-10: Example of Read Destructive Fault (CFrd) 17
Figure 2-11: Example of Deceptive Read Destructive Fault (CFdrd) 18
Figure 2-12: General Block Diagram for Memory BIST [17] 19
Figure 3-1: Flow of designing enhanced MARCH C- algorithm 33
Figure 3-2: General Block Diagram of the proposed MBIST system 47
Figure 3-3: Block Diagram of the Faulty_RAM 49
Figure 3-4: Block Diagram of Test Pattern Generator (TPG) 51
Figure 3-5: Moore Finite State Machine diagram of Test Pattern Generator (TPG) 52
Figure 3-6: Block Diagram of Comparator 53
Figure 3-7: Block Diagram of Fault Type Identifier 54
Figure 3-8: Flow Chart of the MBIST system 55
Figure 4-1: DRF Test Result Report 61
Figure 4-2: Detailed View of DRF 62
Figure 4-3: DRF Fault response 63
Figure 4-4: SAF0 Test Result Report 64
Figure 4-5: SAF1 Test Result Report 65
Figure 4-6: Detailed View of SAF 1 66
Figure 4-7: SAF1 Fault Response 66
Figure 4-8: UTF Test Result Report 68
Figure 4-9: DTF Test Result Report 69
Figure 4-10: Detailed View of DTF 69
Figure 4-11: DTF fault response 70
Figure 4-12: CFst (0,0) Victim cell's location < Aggressor cell's location Test Result Report 72
Figure 4-13: CFst (0,0) Victim cell's location > Aggressor cell's location Test Result Report 72
Figure 4-14: CFst (0,1) Victim cell's location < Aggressor cell's location Test Result Report 72
Figure 4-15: CFst (0,1) Victim cell's location > Aggressor cell's location Test Result Report 72
Figure 4-16: CFst (1,0) Victim cell's location < Aggressor cell's location Test Result Report 72
Figure 4-17: CFst (1,0) Victim cell's location > Aggressor cell's location Test Result Report 72

Figure 4-18: CFst (1,1) Victim cell's location < Aggressor cell's location Test Result Report 73
Figure 4-19: CFst (1,1) Victim cell's location > Aggressor cell's location Test Result Report 73
Figure 4-20: Detailed View of the CFst (1,1) > Fault. 73
Figure 4-21: CFst (1,1) Victim Cell > Aggressor Cell Faulty RAM Responses 74
Figure 4-22: CFid (D,0) Victim cell's location < Aggressor cell's location Test Result Report 77
Figure 4-23: CFid (D,0) Victim cell's location > Aggressor cell's location Test Result Report 77
Figure 4-24: CFid (D,1) Victim cell's location < Aggressor cell's location Test Result Report 77
Figure 4-25: CFid (D,1) Victim cell's location > Aggressor cell's location Test Result Report 77
Figure 4-26: CFid (U,0) Victim cell's location < Aggressor cell's location Test Result Report 77
Figure 4-27: CFid (U,0) Victim cell's location > Aggressor cell's location Test Result Report 77
Figure 4-28: CFid (U,1) Victim cell's location < Aggressor cell's location Test Result Report 77
Figure 4-29: CFid (U,1) Victim cell's location > Aggressor cell's location Test Result Report 77
Figure 4-30: Detailed View of CFid (U,1) > Fault 78
Figure 4-31: CFid (U,1) Victim Cell > Aggressor Cell Faulty RAM Responses 78
Figure 4-32: CFin (D, ~) Victim cell's location < Aggressor cell's location Test Result Report..... 80
Figure 4-33: CFin (D, ~) Victim cell's location > Aggressor cell's location Test Result Report..... 80
Figure 4-34: CFin (U, ~) Victim cell's location < Aggressor cell's location Test Result Report..... 81
Figure 4-35: CFin (U, ~) Victim cell's location > Aggressor cell's location Test Result Report..... 81
Figure 4-36: Detail View of CFin (U, ~) > Fault 81
Figure 4-37: CFin (U, ~) Victim Cell > Aggressor Cell Faulty RAM Responses 82
Figure 4-38: CFir (0, -) Victim cell's location < Aggressor cell's location Test Result Report 84
Figure 4-39: CFir (0, -) Victim cell's location > Aggressor cell's location Test Result Report 84
Figure 4-40: CFir (1, -) Victim cell's location < Aggressor cell's location Test Result Report 84
Figure 4-41: CFir (1, -) Victim cell's location > Aggressor cell's location Test Result Report 84
Figure 4-42: Detail View of CFir (1, -) > Faults..... 85
Figure 4-43: CFir (1, -) Victim Cell > Aggressor Cell Faulty RAM Responses 86
Figure 4-44: CFrd (0, ~) Victim cell's location < Aggressor cell's location Test Result Report..... 88
Figure 4-45: CFrd (0, ~) Victim cell's location > Aggressor cell's location Test Result Report..... 88

Figure 4-46: CFrd (1, ~) Victim cell's location < Aggressor cell's location Test Result Report 88
Figure 4-47: CFrd (1, ~) Victim cell's location > Aggressor cell's location Test Result Report 88
Figure 4-48: Detail View of the CFrd (1, ~) > Fault 89
Figure 4-49: CFrd (1, ~) Victim Cell > Aggressor Cell Faulty RAM Responses 89
Figure 4-50: Partial Waveform of Data Retention Test for Original MARCH C- algorithm test case 92
Figure 4-51: Partial Waveform of Data Retention Test for enhanced MARCH C- algorithm test case 92
Figure 4-52: Partial Waveform of Read Destructive Fault for Original MARCH C- algorithm test case. 94
Figure 4-53: Partial Waveform of Read Destructive Fault for enhanced MARCH C- algorithm test case 95
Figure 4-54: Execution time of MBIST when RAM size is 8*4 bit 97
Figure 4-55: Execution time of MBIST when RAM size is 16*4 bit 98
Figure 4-56: Execution time of MBIST built with original MARCH C- algorithm when RAM size is 16*4 bit 100

LIST OF ABBREVIATION

CFid	Idempotent Coupling Fault
CFin	Inversion Read Fault
CFir	Incorrect Read Fault
CFrd	Read Destructive Fault
CFst	State Coupling Fault
DRF	Data Retention Fault
FSM	Finite State Machine
MBIST	Memory Built-In Self-Test
RAM	Random Access Memory
SAF	Stuck-At Fault
SoC	System-on-Chip
TF	Transition Fault
TPG	Test Pattern Generator

CHAPTER 1

INTRODUCTION

1.1 Background

System-On-Chip (SoC) is result of the development of the CMOS technology that enable the integration of systems into a single chip [1]. Normally, a SoC would contain microcontrollers, memory blocks, oscillators, internal and external peripherals and power management circuits. Among all these partitions, memory blocks has played an important role in terms of functioning and has grown its size for accounting up to 50% or more in the chip area [2]. From the data gathered by the Semiconductor Industry Association (SIA), the area size of memory in SoC had increase from 20% to 71% in 2005 and SIA had predicted that in 2014 it would even reach up to 94% of the SoC area [3]. This statistic had been gathered for International Technology Roadmap for Semiconductors (ITRS) report and are tabulated in the Figure 1-1.

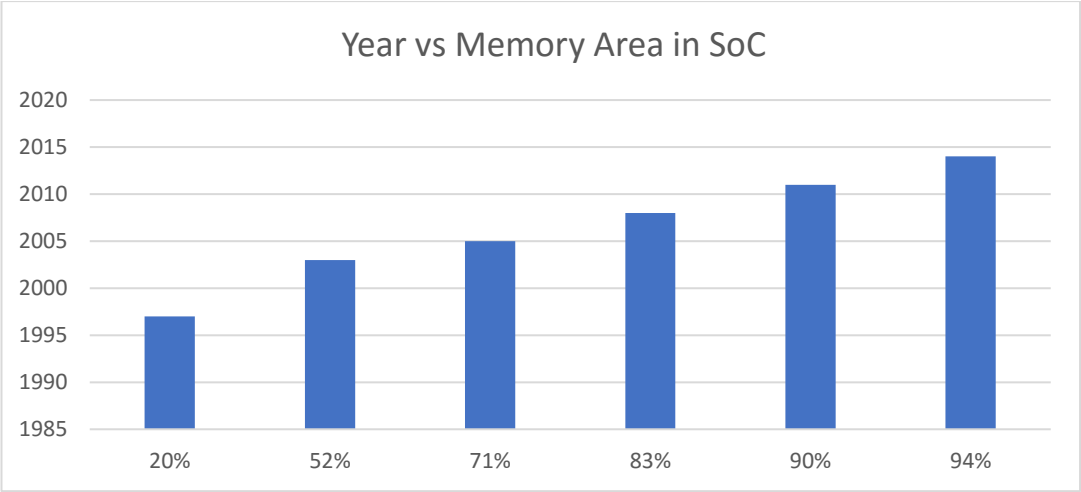

Figure 1-1: Memory area in SoC as per year growth [3]

As the technology advances with memory accounting for more and more area in SoC, it has become necessary to maintain a good production yield for the memory partition, else it would drop the yield in SoC manufacturing which will cost a fortune. Normally, the yield drop in memory partition is due to the presence of faults in it. RAM as part of the memory partition having its own fault types. The common faults found in the RAM are Stuck-At-Fault (SAF), Address Decoding Fault (AF), Coupling Fault (CF), Transition Fault (TF) and many more [4]. Figure 1-2 shows the percentages of memory faults in the RAM. Manufacturers have used automated test equipment (ATE) to identify these faults. Automated test equipment are tester devices that are used to attach on the device under test (DUT) or circuit under test (CUT) to exercise SoC healthiness.

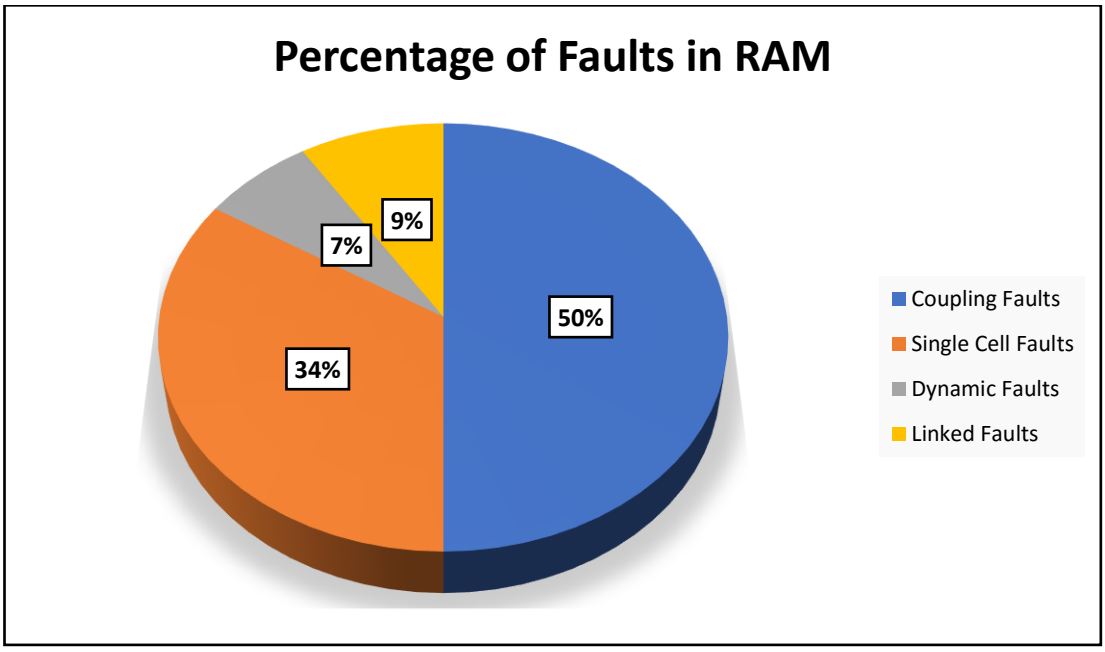

Figure 1-2: Percentage of Faults found in RAM [5]

As cost of fixing a bug is directly proportional with the process stage, it would be causing a little cost in the early stage but greater financial effect is higher in the later stage when discover a fault and bug fix it. One of the methods is to simulate the test

with the help of ATE. However, executing the test with the aid of ATE has drawbacks such as increasing the testing cost and building up reliance upon external test equipment. Thus, Memory Built-In Self-Test (MBIST) system that does not require external hardware has become the new preferred method to carry out fault identification tests.

MBIST is a system that able to perform the memory functional check for the memory partition without the needs of external components. A MBIST is normally made up of test controller, Test Pattern Generator (TPG), Circuit Under Test (CUT), Output Response Analyser (ORA), Output Response Compactor (ORC) and comparator.

There are many types of algorithms that able to be used with MBIST to detect faults in the memory partition. The most common algorithm used with MBIST is MARCH algorithm. This algorithm has a lot of derived algorithms based on it. For example: MATS, MATS+, Marching I/O, MATS++, MARCH X, MARCH C, MARCH A, MARCH Y, and MARCH B [6]. These algorithms can detect the common faults and have different advantages and disadvantages in term of fault coverage, test length, area overhead and power consumption.

After identifying the faults, the next process will be BISR (Built-In Self-Repair) which will be having the self-reparability of the faulty memory cell that had been identified by the MBIST. By synchronizing the BIST and BISR systems, it would save the cost and functionality of the product [2] as the system will be able to repair the fault immediately after identifying it. This process will be efficient in terms of increasing the productivity and saving manufacturing time and cost.

1.2 Problem Statement

As the size of the memory increases in the System-on the Chip (SoC), it has become one of the major partitions that requires high passing rate of the self-checking test to secure the production yield and reduce the cost spent in the fault repairing.

The algorithms that had developed for the MBIST system having the disadvantages of it able to cover its target fault only. For example, MARCH C- algorithm can only covers Address Decoder Fault, Stuck-At Fault, Transition Fault and Coupling Fault but it was unable to extend its fault coverage to other types of fault such as Data Retention Fault and Read Destructive Fault. Data Retention Fault and Read Destructive Fault are not covered 100% by the fault detection in the previous algorithms [7]. Thus, to increase the algorithm efficiency the methodology of detecting those faults are needed to integrate to MARCH C- for further enhancement.

Despite of able to detect the fault types of the respective fault, it was also recommended to having the MBIST system able to provide precise location of the fault memory cells. With the precise location of the fault memory cells, it would help and aid the Built-In Self-Repair works to replace the faulty memory cells with using lesser spare memory cells [8].

Lastly, it was found that previous works and the algorithms can only detect the faults in the memory but do not come with the ability to clearly classify the types of faults that had occurred [3]. To further enhance the MBIST system, it would be good to having the system for clearly identify between the types of the fault detected (Normal Memory Fault and Behavioural Fault) in the system. However, this feature has its own challenges as some of the faults (Stuck-At Faults and Transition Faults) will shared almost the same fault signature.

1.3 Research Objectives

The objectives of this dissertation are to develop a better performance MBIST algorithm with the following criteria:

- Increase the fault type coverage of the MARCH C- algorithm and precisely identify the faulty memory cell up to bit scale
- Classify the memory fault types structurally after identified the faulty memory cell location and able to distinguish between Normal Memory Fault with Behavioral Fault as well as differentiate Stuck-At Fault with Transition Fault.

1.4 Research Scope

This dissertation will be carried out using Modelsim software version 10.4d as the MARCH algorithm will be coded in Verilog to generate the register transfer level (RTL) abstraction of the system.

The types of fault that are included in the fault coverage to be tested out are the behavioural fault (Data Retention Fault) and common memory faults (Stuck-At Fault (SAF), Transition Fault (TF) and various Coupling Faults such as State Coupling Fault (CFst), Idempotent Coupling Fault (CFid), Inversion Coupling Fault (CFin), Incorrect Read Fault (CFir) and Read Destructive Fault (CFrd)).

The proposed system should be able to differentiate the types of fault between Stuck-At Fault (SAF) and Transition Fault (TF) as both fault signatures are very similar. It should also can differentiate between behavioural faults and common memory faults.

1.5 Thesis Outline

This dissertation is made up of five chapters which are Chapter 1: Introduction, Chapter 2: Literature Review, Chapter 3: Design Methodology, Chapter 4: Results and Discussions and Chapter 5: Conclusion and Future Improvements.

Chapter 2: Literature Review will be describing on the previous research that had been done by other researchers for the MBIST system and MARCH algorithm. The ideas and methodologies will be analysed in this chapter. Besides that, detail explanations regarding memory faults types and the respective fault signatures will be discussed as well. Lastly, a comparison of the BIST methodology between Finite State Machine (FSM) methodology and Microcode methodology will be studied.

In Chapter 3: Design Methodology will be introducing enhanced MARCH algorithm and MBIST system proposed by providing sufficient explanations on it and how it is able to fulfil the objectives. There will also be explanation on how the system and algorithm work and with validation methodologies that show its fault coverage.

Chapter 4: Results and Discussion analysed the results obtained from tests that had been executed to prove that the system was able to cover the faults as expected. Each of the test situation will be analysed and conclusion are drawn at the end of each test case. Additionally, the test time of the BIST system using different memory cell address ranges will be tested out to show that the execution time of the system is based on the testing address range.

Lastly, Chapter 5 will be summarizing all the work that had been done so far for this dissertation and discussing on the future improvement of this field that may aid others to enhance in this research topic.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will be discussing and analysing the previous research works that had been done by others in Memory Built-In Self-Test field. In section 2.2, detail discussions will be carried out based on the different types of memory and previous methodology that had been done to discover the faults in those memory types. Section 2.3 will be discussing regarding the types of faults that occur in memory. Section 2.4 will be discussing on MARCH algorithm that had been used in previous researchers works and the coverage of faults by the selected algorithms. The methodology of how the MBIST system had been carried out via finite state machine (FSM) methodology and Microcode methodology by the previous researches will be discussed in section 2.5. Lastly, a summary will be drawn based on the previous research works and define the specification needed for this dissertation to meet the aim and objectives.

2.2 Memory Types (DRAM, SRAM)

Memories can be split into two big groups which are the Random-Access Memory (RAM) and Read Only Memory (ROM). RAM is a volatile memory that will lost its data when the power supply is turned off whereas ROM is a non-volatile memory that able to retain its own data although there is no power supply. As ROM characteristic ability to retain data with or without power supply, it is more commonly used to store

critical programs. Whereas for RAM volatile characteristic, it is more often used as a temporary storage for data by the operating system. For Memory BIST, normally RAM are selected to be tested compared with the ROM.

RAM can be further split into two smaller groups which are Static Random-Access Memory (SRAM) and Dynamic Random-Access Memory (DRAM). SRAM does not require periodical refresh like DRAM. A DRAM would require constant refresh to preserve its memory content. This is due to the capacitor inside the DRAM integrated circuit as it will keep on leaking its charge (leakage current) and if it was not refreshed periodically, then the data might had contaminated.

Both RAM cases had been tested out with MBIST by many previous works done by researchers. Previous findings explained a lot on the fault characteristics of each respective RAM and the methodology been used. This sub-chapter will be discussing on the types of RAM and the respective fault types that were being identified.

The work of [5], focused on testing out the common fault that will be found in the SRAM which were Stuck-At Fault, Transition Fault and Coupling Faults using macro and micro codes. The work analyzed that the highest fault percentage among the SRAM were Coupling faults (50%) and followed up by single-cell faults (34%). Single-cell faults refer to the faults that happen in single memory cells and does interfere with neighbour memory cells. Single memory cell faults were commonly referring to Stuck-At Fault (SAF) and Transition Faults (TF). Thus, a few MARCH algorithms had been tested out on micro and macro codes environment. The algorithm used and fault coverage will be discussed in the MARCH algorithm sub-chapter (Section 2.4) whereas the macro and micro code methodology detail will be discussed in the FSM methodology and Microcode methodology sub-chapter (Section 2.5).

Work of [9] used MARCH CL algorithm to test out coupling faults, intra-word stuck-at, and transitions faults in SRAM. Faults that were covered in this work were Stuck-At Faults, Transition Faults, State Coupling Faults, Idempotent Coupling Faults and Write Disturb Coupling Faults. To achieve that, MARCH CL algorithm was designed to have a complexity of 12N (N is the number of memory words) and was split to three steps for fault identification. At first, MARCH CL was performed to filter out the fault. After that, the aggressor word was being identified by a 3N or 4N March-like algorithm and followed up by a last step which was to identify the aggressor bit from the aggressor word. The last step would be using a March-like algorithm of 9(1+logB) (B is the number of bit in the word) to detect and locate the inter-word faults. With this precise fault determination methodology, it has a higher fault identification ability and much lower time complexity. The number of read operation performed via this methodology was 41.6% lower than MARCH 17N in determine inter-word fault.

However, [10] had come out with an improvement from [9] methodology to have more fault coverage such as faults that happen in single memory cell and multiple memory cells. The extra covered faults were Write Destructive Faults, Read Destructive Faults, Deceptive Read Destructive Fault, Incorrect Read Fault, Transition Coupling Faults, Read Destructive Coupling Faults, Deceptive Read Destructive Coupling Faults and Incorrect Read Coupling Faults [10]. MARCH PD was being proposed as it has the complexity of 31N to enable better coverage than the previous work mentioned above. With the similar methodology of identifying the fault via narrowing down the scope, MARCH PD was used to identify and partially diagnose the DUT and followed up by a MARCH algorithm of 3N to 5N for identifying the aggressor word. Another MARCH algorithm having complexity of 16logB+18 were

used to locate aggressor bit from the aggressor word. Through this methodology, it had proven to have a better fault coverage than the previous methodology.

In the research work presented by [11] had further improvement by reducing N in the MARCH algorithm was used to perform fault filtering and partial diagnostic to 18N. The methodology performed are the same as [9], [10]. Although the number of MARCH element had decreased but it remains its high fault type coverage ability such as state fault, transition fault, coupling faults etc. with low time complexity.

The research work performed by [12] focused on performing test coverage in SRAM common faults (Stuck-At Faults, Transition Faults, Coupling Faults by using MARCH AB that was made up of 22N and able to cover all SRAM common faults. The test had successfully reduced the test complexity by 15.4% compared to others MBIST tests.

From [13]'s research on the MBIST for DRAM, it used three MARCH like algorithm to test on the following fault coverage: Stuck-At Fault (SAF), Coupling Fault (CF), Address Decoder Fault (ADF) and Transition Fault (TF). By using anti-random test pattern generation methodology, the test efficiency had been increased largely due to the test vector was different each time when a test was being executed. This had increased the fault coverage rate and eventually hit up to 96% of accuracy.

MATS++ algorithm had been used in [14]'s work to cover the identification of Stuck-At Fault, Transition Fault and four types Address Decoding Fault in DRAM. Although the work was focusing on the Built-In Self-Repair functionality, the model can identify the types of common faults that occurred on DRAMs. From the test result, MATS++ had the result of covering up to 96.72% of the mentioned faults.

As a conclusion, the fault types that had caught the interest of the previous researchers are the commonly found fault in memory partitions (SRAM and DRAM) such as Stuck-At Fault (SAF), Coupling Fault (CF) and Transition Fault (TF).

2.3 Faults

The word faults in this dissertation are referring to the error conditions in the memory array. It can be split two types: behavioral fault and common memory fault. Behavioral fault is related to the fault occurred due to the characteristic of the memory cells, whereas common memory fault can be categorized as two major types which are the single cell faults and multiple cell faults. Single cell faults are the memory faults that occur within single memory cells, whereas multiple cell faults involve two or more memory cells. Examples of single cell faults are stuck-at fault, transition fault and address decoding fault [10]. The example of multiple cell fault is coupling faults. Figure 2-1 shows the healthy memory cell having write/read operations in two states.

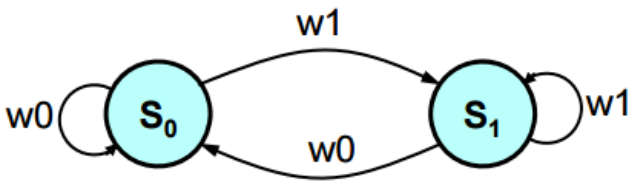

Figure 2-1: Normal Memory Cells

As shown in Figure 2-2 and Figure 2-3, Stuck-At fault is the fault having the value of the memory stuck in line or memory cells. It can be differentiated into Stuck-At-0 (SA0) or Stuck-At-1 (SA1). Just like its name, Stuck-At-0 refers to the value is always stuck at logic 0 whereas Stuck-At-1 having permanent logic of 1 [15].

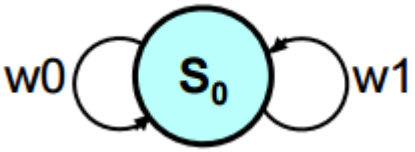

Figure 2-2: Stuck-At 0 Fault

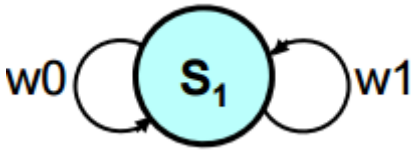

Figure 2-3: Stuck-At 1 Fault

Transition Fault is (TF) is the fault having failure on performing transition of logic. Same as Stuck-At Fault, Transition Fault can be differentiated into two subclasses: Transition $(\Uparrow, 0)$ and Transition $(\Downarrow, 1)$. Transition $(\Uparrow, 0)$ refers to the situation where the defective memory cell fails to undergo changes of logic 0 to logic 1 during rising transitions. As opposed to Transition $(\Uparrow, 0)$, Transition $(\Downarrow, 1)$ refer to the situation where the defective memory cell fails to undergo changes of logic 1 to logic 0 during falling transitions [9]. Figure 2-4 shows the Transition $(\Uparrow, 0)$ Fault where it fails to change the state from S0 to S1 when W1.

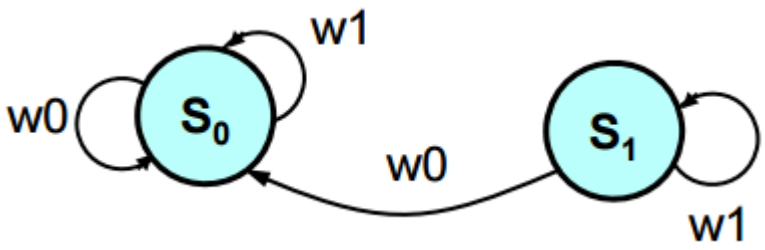

Figure 2-4: Transition $(\Uparrow, 0)$ Fault

Address Decoder Fault is the fault caused when the address is wrongly decoded. It can be categorized into four types which are no memory cell that can be accessed

with the given address, none of the memory cells that are able to be accessed, multiple memory cells are being accessed with given address and certain memory cell that can be accessed with multiple addresses. Normally, Address Decoder Fault is a combination of two different cases stated above as it is not able to exist alone due to addresses are generated based on multiple memory cells [14]. It exists as the following combination as shown in Figure 2-5:

1. Fault A: address failed to access the memory cell and memory cells failed to be accessed by address
2. Fault B: address failed to access targeted memory cells whereas the targeted memory cells are being accessed by another address range
3. Fault C: certain memory cells fail to be accessed by designated addresses as the addresses were accessing wrong memory cells
4. Fault D: having multiple memory cells that are able to be accessed by multiple addresses.

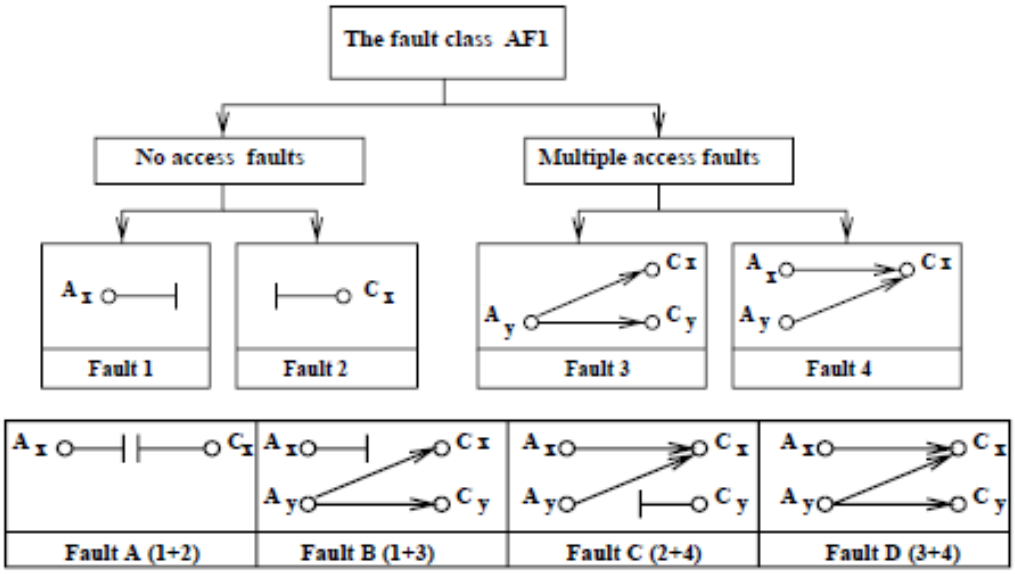

Figure 2-5: Example of Address Decoder Fault (ADF) [16]

Coupling Fault is a multiple cell fault. It involves two cells which was the aggressor cell and victim cell. Aggressor cell was the memory cell that was a normal cell but its state or transaction will affect the behavior of the victim cells. Victim cell was the faulty memory cell that will change the state or transaction when the aggressor cell hit a certain condition. Coupling faults occur when the aggressor cell was written with a value or in a certain transaction, the victim cells will have the value changed [15]. It can be differentiated into a few types as stated below [9]:

1. State Coupling Fault (CFst)

   State Coupling Fault (CFst) happens when the victim cell is forced into a certain logic state (can be either logic 0 or 1) as the aggressor cell is at a certain state (logic 0 or 1) without any operation being performed to the victim cell. Figure 2-6 shows an example of the CFst. The victim cell changed its data from 0 to 1 when the aggressor cell is in state (0). This fault can be further separated into four types which are: CFst (0,0); CFst (0,1); CFst (1,0); CFst (1,1). The first parameter is representing the aggressor cell's state and the second parameter is the value changed by the victim cell.

Address: 0,2 Data: 0	Address: 0,1 (Victim cell) Data: 0 ➔ 1	Address: 0,0 Data: 0
Address: 1,2 Data: 0	Address: 1,1 (Aggressor cell) Data: 0	Address: 1,0 Data: 0
Address: 2,2 Data: 0	Address: 2,1 Data: 0	Address: 2,0 Data: 0

Figure 2-6: Example of State Coupling Fault (CFst (0,1))

2. Idempotent Coupling Fault (CFid)

   Idempotent Coupling Fault (CFid) is the case when the aggressor cell is having a transition, the victim cell will be forced to a certain value. Figure 2-7 shows an example of the Idempotent Coupling Fault where when the aggressor cell is having an up transaction, it changed the value of the victim cell. Same as State Coupling Fault (CFst), it can be further separated into four types: CFid (D,0); CFid (D,1); CFid (U,0); CFid (U,1). The first parameter is the transaction of the aggressor cell and the second parameter is the value changed by the victim cell.

Address: 0,2 Data: 1	Address: 0,1 (Victim cell) Data: 1 ➔ 0	Address: 0,0 Data: 1
Address: 1,2 Data: 1	Address: 1,1 (Aggressor cell) Data: 0 ➔ 1	Address: 1,0 Data: 1
Address: 2,2 Data: 0	Address: 2,1 Data: 0	Address: 2,0 Data: 0

Figure 2-7: Example of Idempotent Coupling Fault (CFid (U,0))

3. Inversion Coupling Fault (CFin)

   Inversion Coupling Fault (CFin) occurs when the aggressor cell undergoes a transaction, the content of the victim cell is inverted (logic 0 → logic 1 and vice versa). Figure 2-8 is an example of the Inversion Coupling Fault when the aggressor cell had down transactions, it inverts the value of the victim cell. The Inversion Coupling Fault can be split into two types which are: CFin (U, ~) and CFin (D, ~). The first parameter is for the transaction of the aggressor cell and the second parameter is the value changed by the victim cell.

Address: 0,2 Data: 0	Address: 0,1 (Victim cell) Data: 0 ➔ 1	Address: 0,0 Data: 0
Address: 1,2 Data: 0	Address: 1,1 (Aggressor cell) Data: 1 ➔ 0	Address: 1,0 Data: 0
Address: 2,2 Data: 1	Address: 2,1 Data: 1	Address: 2,0 Data: 1

Figure 2-8: Example of Inversion Coupling Fault (CFin (D, ~))

4. Incorrect Read Fault (CFir)

   Incorrect Read Fault happens when a read operation applied to the victim cell, it returns an incorrect value when the aggressor word is in the given state. The state of the victim cell is not changed. Figure 2-9 shows the example of Incorrect Read Fault as when aggressor cell is at the certain state (0), the data read out by the victim cell is 1 but the data it holds remains 0.

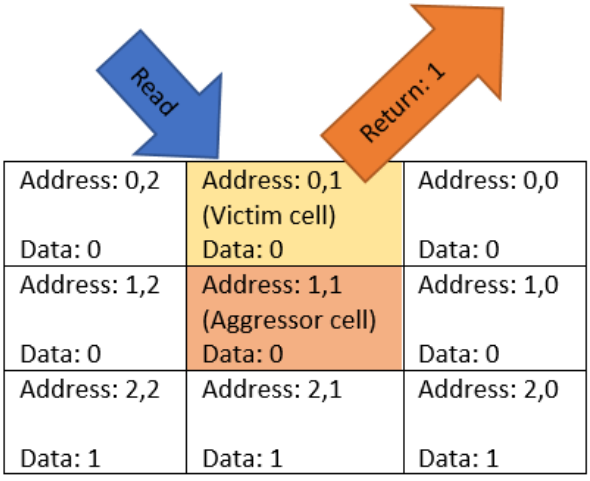

Figure 2-9: Example of Incorrect Read Fault (CFir)

5. Read Destructive Fault (CFrd)

Read Destructive Fault occurs when having a read transaction towards the victim cell while aggressor cell is in the given state, it will return an invalid value for the read process. The state of the victim cell is changed to the invalid value. Figure 2-10 shows how Read Destructive Fault occurs as the aggressor cell is in the certain state (0), the read operation will change the data value inside the victim cell and return the invalid value to the read operation.

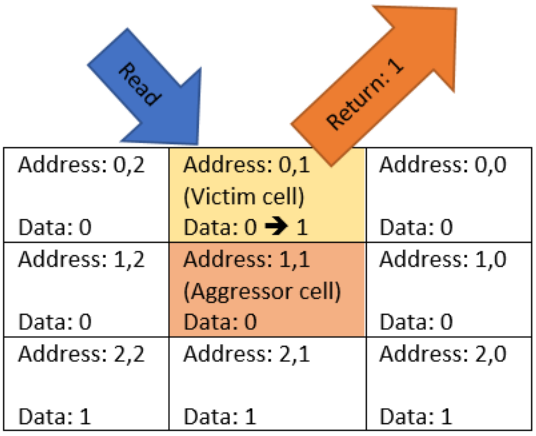

Figure 2-10: Example of Read Destructive Fault (CFrd)

6. Deceptive Read Destructive Fault (CFdrd)

Deceptive Read Destructive Fault is caused by having a read operation to the victim cell and it return a correct value with a transition that will change the value of the victim cell when the aggressor cell is at a certain state. Figure 2-11 shows the example of Deceptive Read Destructive Fault having the value of the victim cell changed in the second read when the aggressor cell in a certain state (0).

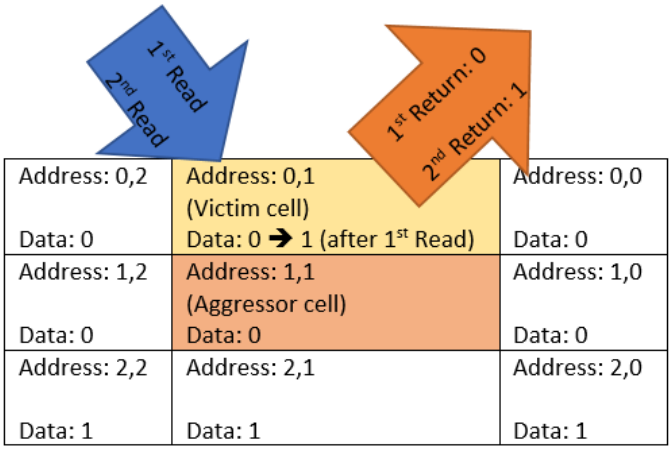

Figure 2-11: Example of Deceptive Read Destructive Fault (CFdrd)

Data Retention Fault is a behavioral fault. It happens when the data get corrupted after some time before the refresh of the RAM occurs. This fault would be corrupting the existing data as it was unable to retain the data's integrity after a period of time. It can be tested out by adding some extra time before the read operation is run.

2.4 MARCH algorithms

As the size of the SoC chips are getting smaller and the memory size in the SoC chips had increasing rapidly, Automatic Test Equipment and Built-In Self-Test are being proposed as ways to ensure the healthiness of the memory partition.

Among both methods, Built-In Self-Test are preferable by the testers due to it having lesser cost needed as the usage of external equipment is avoided. The usage of BIST will also reduce the effort for development of test patterns in the ATE. Most importantly, better observation and controllability on the generation of the test pattern can be obtained via this method [14]. Furthermore, BIST allow the test to be executed at full functional speed, which will save a lot of the test time.

MBIST environment is made up of address bus, data bus, multiple read/write control signals that was controlled by the test pattern generator [17]. There were many MBIST techniques that had been developed, but mostly been conducted by using exhaustive test of the MARCH algorithm that promises a high percentage of fault coverage [13]. MBIST techniques share almost the same methodology as the BIST with a slight difference as it consists of some extra components. The common components that exist in the Memory BIST are test controller, Test Pattern Generator (TPG), test collar, Circuit under Test (or is known as Device under Test), Output Response Analyzer (ORA), Output Response Compactor (or known as Multiple-input Signature Register (MISR)), comparator and a ROM. Figure 2-12 shows how the Test Pattern Generator (TPG) affects the Memory Under Test.

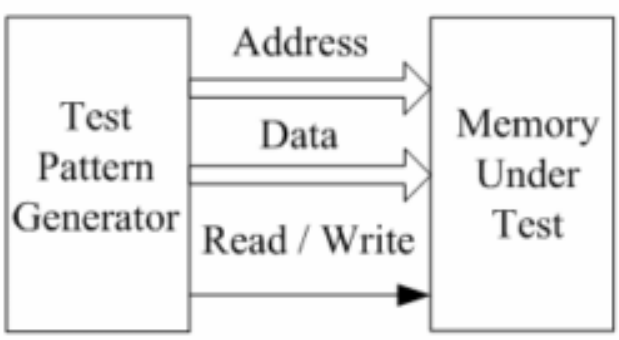

Figure 2-12: General Block Diagram for Memory BIST [17]

There were many algorithms that generated under the concept of the Memory BIST such as Checkerboard (CB), Address Decoder Open (AD), Address Complement (AC), Waltz, Column Disturb and MARCH algorithm [18].

Checkerboard (CB) is an algorithm that works in the concept of having 0101 patterns in the neighbouring cells. Thus, all the neighbouring memory cells are having

opposite status with each other [18]. The values will be inverted in the next state and remain 0101 differences among the cells.

Address Decoder Open is the method used to detect the open faults in the address decoder by writing the neighbouring address with a Hamming distance of one and check the neighbour cells to ensure that it was not affected by the neighboring values [18].

Address Complement is the method to overcome the worst scenario of the address decoder would ever face. With all the address bits are being switched at the same time could easily cause address decoder fault due to the decode timing might be mismatching and causing the address to be wrongly decoded. This method requires state changes in all the NAND gates in the decoder to settle down to generate the correct address [18].

MARCH algorithm is a common algorithm that is been used for MBIST. This is due to its short testing time and high fault coverage [4]. There were many types of the MARCH algorithm that had been derived and each of them have their strength and weakness in identifying certain types of faults. A normal MARCH algorithm is made up of a series of March Elements.

March Elements are the operations that will applied to each cell in the memory before applying to another memory cell based on the address order [8]. March elements are separated from each other by using a semicolon (;) between each of the March elements. There were seven types of notations that had been used to generate a March element which has its function linked with address and write/read controllability. The following equation is an example a MARCH algorithm known as MATS++ that has three march elements:

$$MATS++: \Updownarrow (W0);\ \Uparrow (R0, W1); \Downarrow (R1, W0, R0);$$

MATS++ algorithm can be deciphered as firstly performing write 0 to the memory cells either from minimum address range to maximum address range or writing 0 from maximum address range to minimum address range. After that, a read operation is performed and expecting the returning memory value would be 0 and simultaneously performing write 1 to that memory cells after the read from minimum address range to maximum address range. Lastly, read operation is performed with expecting the return value is 1 and writes 0 before another read operation that expecting reading 0 from the memory cells. The last operation is performed from maximum address range to minimum address range.

Table 2-1 are the detailed explanation of the operation that used to construct March elements.

Table 2-1: Table of MARCH Operations and its functions.

Operations	**Descriptions**
⇑	Minimum address to the Maximum address
⇓	Maximum address to the Minimum address
⇕	Can be either Minimum address to Maximum address or Maximum address to Minimum address
W0	Write logic 0 to that memory cell
W1	Write logic 1 to that memory cell
R0	Read from the memory cell which should contain the value of 0.
R1	Read from the memory cell which should contain the value of 1.

Table 2-2 explains the existing MARCH algorithms with respective definitions and fault coverage:

Table 2-2: MARCH algorithms and Fault Coverage

MARCH Algorithms	**Definitions**	**Fault Coverage**
MATS + [5], [19] [20], [21]	$\{\Updownarrow (w0);\ \Uparrow (r0,w1);\Downarrow (r1,w0)\}$	SAF
MATS ++ [14]	$\{\Updownarrow (w0);\Uparrow (r0,w1);\Downarrow (r1,w0,r0)\}$	ADF, SAF, TF
MARCH A [5], [19] [20], [22]	$\{\Updownarrow (w0);\ \Uparrow (r0,w1,w0,w1);\ \Uparrow (r1,w0,w1);$ $\Downarrow (r1,w0,w1,w0);\ \Downarrow (r0,w1,w0)\}$	SAF, TF, ADF, some CF
MARCH B [5], [19] [20], [21] [22]	$\{\Updownarrow (w0);\Uparrow (r0,w1,r1,w0,r0,w1);$ $\Uparrow (r1,w0,w1);\ \Downarrow (r1,w0,w1,w0);$ $\Downarrow (r0,w1,w0)\}$	SAF, TF, ADF, some CF
MARCH C- [4], [5] [19], [20] [21], [22]	$\{\Updownarrow (w0);\ \Uparrow (r0,w1);\ \Uparrow (r1,w0);\Downarrow (r0,w1);$ $\Downarrow (r1,w0);\ \Updownarrow r0)\}$	SAF, TF, ADF, CF
MARCH X [19], [20] [22]	$\{\Updownarrow (w0);\ \Uparrow (r0,w1);\ \Downarrow (r1,w0);\ \Updownarrow (r0)\}$	SAF, TF, ADF, some CF
MARCH U [5], [19] [20], [21]	$\{\Updownarrow (w0);\ \Uparrow (r0,w1,r1,w0);\ \Uparrow (r0,w1);$ $\Downarrow (r1,w0,r0,w1);\ \Downarrow (r1,w0)\}$	SAF, TF, CF

MARCH Y [19], [22]	$\{\Updownarrow(w0);\ \Uparrow(r0,w1,r1);\ \Downarrow(r1,w0,r0);\ \Updownarrow(r0)\}$	SAF, TF, ADF, some CF
MARCH AB [22]	$\{\Updownarrow(w0),\Downarrow(r0,w1,r1,w1,r1);\ \Downarrow(r1,w0,r0,w0,r0);\ \Uparrow(r0,w1,r1,w1,r1);\ \Uparrow(r1,w0,r0,w0,r0);\Updownarrow(r0)\}$	SAF, TF, CF
MARCH LR [5], [20], [21], [22]	$\{\Updownarrow(w0);\ \Uparrow(r0,w1,w0,w1,r1);\ \Uparrow(r1,w0,w1,w0,r0);\ \Downarrow(r0,w1,w0,w1,r1);\ \Downarrow(r1,w0,w1,w0,r0);\ \Downarrow(r0)\}$	SAF, TF, CF
MARCH CL [9]	$\{\Uparrow(w0);\ \Uparrow(r0,w1);\Updownarrow(r1);\Uparrow(r1,w0);\ \Downarrow(r0,w1);\Updownarrow(r1);\Downarrow(r1,w0);\ \Updownarrow(w0)\}$	SAF, TF, CF
MARCH PD [10]	$\{\Uparrow(w0);\Uparrow(r0,r0,w1,w1,r1,r1);\ \Uparrow(r1,w0,w0,r0);\Uparrow(r0,w1\};\ \Uparrow(r1,w0);\Uparrow(r0);\Downarrow(r0,w1);\ \Downarrow(r1,w0);(r0,r0,w1,w1,r0,r0),\ \Downarrow(r1,w0,w0,r0);\Downarrow(r0)\}$	SAF, TF, CF
MARCH MSS [11], [19]	$\{\Uparrow(w0);\Uparrow(r0,w1,w1,r1);\Uparrow(r1,w0,w0,r0);\ \Downarrow(r0,w1,w1,r1);\ \Downarrow(r1,w0,w0,r0);\Downarrow(r0)\}$	SAF, TF, CF

MARCH 9N with Pause Test [23]	$\{\Downarrow (w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Uparrow (r0, w1);$ $\Uparrow (r0, w1)\ Delay;$ $\Downarrow (r0, w1)\ Delay; \Downarrow (r1)\}$	DRF
MARCH-NU [7]	$\{\Updownarrow (w0);$ $\Uparrow (r0, w0, r0, r0, w1, r1, w1, r1, r1, w0, r0);$ $\Uparrow (r0); \Uparrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (w1);$ $\Downarrow (r1, w0, r0, r0, w0, r0, w1, r1, r1, w1, r1);$ $\Downarrow (r0)\}$	SAF, TF, SOF, DRF, CF, ADF, NPSF,

As mentioned in Table 2-2, there are many available MARCH algorithms and each of it has each specific fault coverage. The following are the research works that had been done by using MARCH algorithm mentioned above.

[14]'s work had implemented MATS++ algorithm to cover Stuck-At Fault, Transition Fault and four types of Address Decoder Fault. The MBIST system design consumed 1884 gates having 100% coverage for Stuck-At Fault, Transition Fault, Address Decoder Fault Types A and B. However, the design has 88% fault coverage for Address Decoder Fault Type C and D due to the tests are generating fixed data for BIST which are unable to cover certain condition cases.

The methodologies applied on [9], [10] and [11] were similar three steps with different algorithms: identifications of the functional faults model (common fault), identifications of the aggressor word, and finally identification of the aggressor bit within the aggressor word. The MARCH algorithms used were MARCH_PD, MARCH CL and MARCH MSS. These algorithms have a high coverage in the