OPTIMIZING RAM TESTING METHOD FOR TEST TIME SAVING USING AUTOMATIC TEST EQUIPMENT

By

PREMKUMAR A/L KESAVAN PRABAGARAN

A Dissertation submitted for partial fulfilment of the requirement for the degree of Master of Science (Electronic Systems Design Engineering)

August 2017

Acknowledgment

As a matter of first importance, I might want to thank my supervisor, Dr. Aeizaal Azman Abd Wahab for his direction and guidance all through the entire research. He has given me a ton of helpful data which I can use to finish my thesis. I truly value his understanding, proposal, and energy amid a few meetings that we had. His productive and profitable advice have been helped a ton in creating thoughts for my research.

Moreover, I might want to thank my immediate manager, Mr. Vincent Lee for his persistent support and understanding which urge me to completely use my time for both working undertaking and research.

Then again, I might want to thank my mentor, Mr. Tan Tze Sin for his recommendation and remark on helping me to create a quality outcome in my research. His readiness to share his experience on this research has contributed in improving the general procedure of test methodology development.

Lastly, I might want to thank my relatives particularly my parent who are continually supporting me and keep me roused on finishing my research. Finally, I might want to thank most my colleagues for their assistance and support amid this research development.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Abbreviations and Nomenclature

Abbreviation	Meaning
ATE	Automate Test Equipment
SOC	System on Chip
DFT	Design for Testability
DPM	Defect Per Million
FPGA	Field Programable Gate Array
BIST	Built-in Self-Test
BIRA	Built-in Redundancy Repair
SAF	Stuck-AT Fault
SOF	Stuck Open Fault
TF	Transition Fault
DRF	Data Retention Fault
CF	Coupling Fault
BF	Bridging Fault
NPSF	Neighbor Pattern Sensitive Fault
MBIST	Memory Built-In Self-Test
TCL	Tool Command Language
SRAM	Static Random Access Memories
JTAG	Joint Test Action Group
IJTAG	Internal Joint Test Action Group
VCS	Verilog Compiler Simulator
LSB	Least Significant Bit
MSB	Most Significant Bit

OPTIMIZING RAM TESTING METHOD FOR TEST TIME SAVING USING AUTOMATIC TEST EQUIPMENT

Abstract

Oleh sebab saiz memori meningkat secara drastik dalam “Field-Programmable Gate Array” (FPGA) atau peranti sistem-atas-cip (SOC), ia menjadi sukar untuk memenuhi bajet kos ujian untuk produk peranti kos rendah. Salah satu faktor utama penyumbang kos ujian adalah masa ujian. Bagi produk kos rendah, nombor toleransi kecacatan setiap juta (DPM) adalah relatif tinggi berbanding produk kos tinggi. Dengan kelebihan ini, kaedah ujian memori yang optimum dapat dilaksanakan untuk meminimumkan masa ujian tanpa menjejaskan liputan ujian. Memori Built-in-Self-test (BIST) direka dengan keupayaan untuk menangkap urutan algoritma yang gagal dan dilaksanakan dalam aliran Alat Ujian Automatik (ATE) untuk skrin pengeluaran. 3 algoritma yang terpilih telah diuji pada 8 unit pengesan dalam aliran ATE untuk membuktikan konsep kaedah ini. Urutan algoritma yang gagal telah dimasukkan ke dalam pangkalan data dan dianalisis untuk pemangkasan algoritma. Lokasi pemangkasan algoritma dan pengiraan penjimatan masa ujian telah ditunjukkan dengan contoh yang tepat dalam kajian ini. Menurut contoh ini, anggaran 33% pengurangan masa ujian telah diperhatikan untuk ujian memori 1Kbyte dengan algoritma Hammer Head. Secara ringkasnya, penyelidikan ini telah mencadangkan penjimatan masa ujian memori dengan mengoptimumkan algoritma ujian pada aliran ATE.

OPTIMIZING RAM TESTING METHOD FOR TEST TIME SAVING USING AUTOMATIC TEST EQUIPMENT

Abstract

Due to the memory size increase drastically in the field programable gate array (FPGA) or system on chip (SOC) device, it become hard to meet the tests cost budget of the product especial for low-cost device. One of the major factor of test cost contributed is the test time. For the low-cost product, the tolerance number of the defects per million (DPM) are relative high compare to high cost product. By taking this advantage, an optimizing memory testing method able to implement to minimize the test time without jeopardize the test coverage. A memory Build-in Self-test (BIST) design with capability of algorithm failing sequence capture have been developed to implement in the Automate Test Equipment (ATE) flow for production screen. 3 selected algorithm have been tested on the 8 detect units in ATE flow to prove the concept of this method. The failing algorithm sequence of the units have been logged into database and analyzed for algorithm trimming. With the proper examples, the algorithm trimming location and test time saving calculation have been shown in this research. For this examples, approximate 33% of test time reduction observed for 1Kbyte memory testing with Hammer Head algorithm. In summary, this research has proposed the memory test time saving by optimizing the tests algorithm on the ATE flow.

CHAPTER 1

INTRODUCTION

1.1 Overview

Looking at the current design of SOC or FPGA, the area of chip is mostly covered by memory. The design complexity increase is proportional with increase of memory size especially SRAM. A research estimate roughly at least 68% of System on Chip (SOC) design will be occupied by memory by year 2017 as shown in Figure 1.1 [1]. The researcher believe that memory failure will be the big contribution to yield loss [2]. Per the data presented by the International Technology Roadmap for Semiconductors, the Static Random Access Memories (SRAM) will occupied a major part in the high performance and highly integrated digital system [3].

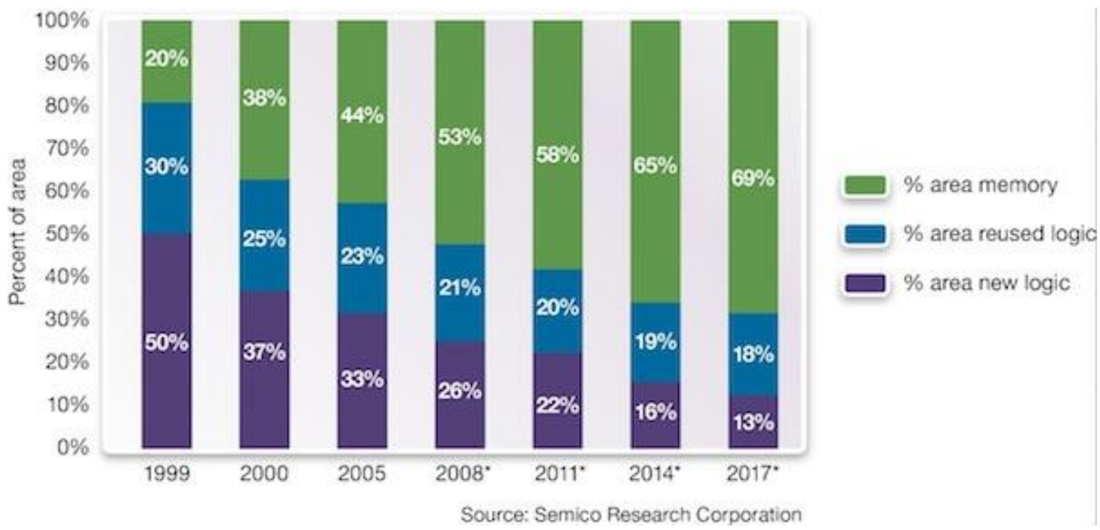

Figure 1.1. Percent of Logic and Memory Area in SOC [1]

Due to size increase for memory, memory testing for defect required high test time. This becomes serious problem for the test engineering to meet the testing cost limitation.

Besides that, the shrink of the technology node increase the various type of fault occur in memory. More effective and complex algorithm need to be introduced to give good test coverage and meet defect per million (DPM) specifications. Since the transistor are very close to each other, memory circuits suffer from a very high average number of physical defects per unit chip area compared with other circuits. This fact has motivated researchers to develop efficient memory test sequences that provide good fault coverage within test cost budget.

1.2 Problem Statements

For current testing method for huge size memories, the algorithm taking excessively high test time. For examples, GALPAT and WALKING [4] algorithms required test times of order of N^2 and $N^{3/2}$ where N is the number depth or address of the memories. At that rate, accepting a period duration of 100 ns, testing a 16Mbit memories would require 500 hours for a N^2 test and 860 seconds for a request $N^{3/2}$ test. Other more established tests, for example, Zero-One and Checkerboard, are of order of N, however they have poor fault coverage. Table 1 demonstrates the memory testing time as a component of memory size [5].

Table 1.1. Test time as a function of memory size [5].

Size	Complexity			
n	n	$n \log n$	$n^{3/2}$	n^2
1K	0.0001s	0.001s	0.0033s	0.105s
4K	0.0004s	0.0048s	0.0262s	1.7s
16K	0.0016s	0.0224s	0.21s	27s
64K	0.0064s	0.1s	1.678s	7.17m
256K	0.0256s	0.46s	13.4s	1.9h
1M	0.102s	2.04s	1.83m	1.27d
4M	0.41s	9.02s	14.3m	20.39d
16M	1.64s	39.36s	1.9h	326d
64M	6.56s	2.843m	15.25h	14.3y
256M	26.24s	12.25m	5.1d	229y
1G	1.75m	52.48m	40.8d	3659y

1.3 Objectives

The objectives of this research are as the following:

i. To develop a built-in self-test (BIST) design that able to capture the failure sequence in the algorithms.

ii. To implement BIST design or tests in automatic test equipment (ATE) for data logging on the failing sequence of the algorithms.

iii. To analyze the effectively way of algorithms trimming and the impacted to tests time.

1.4 Project Scopes

This research scope will be focus on the developing of BIST design with error state capture and implement the tests in ATE platform. This project will mainly describe how algorithm trimming is done and impacted to the test time.

The research is limited to define the methodology of algorithm trimming and BIST development. No production data of products is shared or discussed in this research.

1.5 Research Contribution

This research is to develop an effective test solution for low cost device memory by proper study on tests algorithm during production test using Automatic Test Equipment (ATE). This research will help the Test Engineer to have a better understanding on the algorithm used for memory testing and type of fallout occurs during production testing. Outcome from this research will improve the memory testing methodology and test time. Organization could benefit from test cost saving and production cycle time to meet customer requirement.

1.6 Thesis Organization

In Chapter 2, the outcome of the literature review has been carry out. A study on the implementation of the BIST, memory algorithm notation and fault coverage, and some of memory test time saving methodology have reviewed and discussed. A comparison has been done for the existing method with proposed method.

Chapter 3 discuss about the methodology used for developing the proposed BIST and the test program for ATE. Besides that, the calculation and analysis method for defects per million (DPM) value and tester data have been elaborate.

In Chapter 4, detail of simulation results of the BIST operation have been shared and discussed. All the important signals of the design in different test case are shared in the waveform for the verification of the BIST operation. Following by ATE tester results on implementing the tests in tester platform. Few units have been selected to tested with this BIST and results have shared in table. Finally, the chapter is closed by discussion of the overall results.

Chapter 5 summarizes and concludes the results from the research on test time saving with this methodology. Lastly, the future work for this project have been suggested.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter provides research study about memory testing and test time impact to production cost of device. Add more, few type of test time saving in memory testing briefly discussed and compared in this chapter. In section 2.2 covers the functional model of memory. Section 2.3 explains the memory testing methodology and fault coverage of testing algorithm. Section 2.4 describes the memory test time impact base on algorithms. In section 2.5, the advantage of build-in Self-test (BIST) use in memory testing is discussed. Few techniques that have developed for test time saving is analysis in section 2.6. Lastly, section 2.7 discuss about memory test time reduction methodology used in industrial for comparison.

2.2 Functional Model of Memory

Memory is a huge array of cells which contain data. These cells are unique addressable on a matrix with row and column address. There are 3 major signal used to operate the memory. These signals are address signal, control signal and data signal. The address signal used to identify the location of the cells to be operated. While control signal is to determine the operation between write and read to the cells. Finally, the data signal carry the data in or out of the memory cells [6].

2.3 Memory Testing Methodology

Memory testing algorithm is involved with combination of write and read sequence to capture all kind defect in memory. There are number of algorithm is implemented in memory to screening the device in production flow. Each of algorithm have their own defect coverage for the memory [7].

A study has been carry out to determine coverage fault for few industrial algorithms that been used in production screening. The fault coverage able to represent in percentage value to identify the most effectively algorithm compare to others. The percentage is determined by accumulating the number of faults captured by the algorithm in relation to the total number of faults. Higher the percentage value, more fault coverage detected by the algorithm. Table 2.1 and Figure 2.1 shows the algorithm notation and fault coverage respectively [8].

From the Figure 2.1 data, the highest coverage fault is achieved by Algorithm B, Hammer Walk, March U, March LR and March SS algorithm, around 80% of the coverage faults compared to others algorithms. The SCAN and HAM5W is giving below 30% of faults coverage. As cross check with the notation Table 2.1, the higher faults coverage algorithms are complex and high test time compared to SCAN or HAM5W algorithm [8].

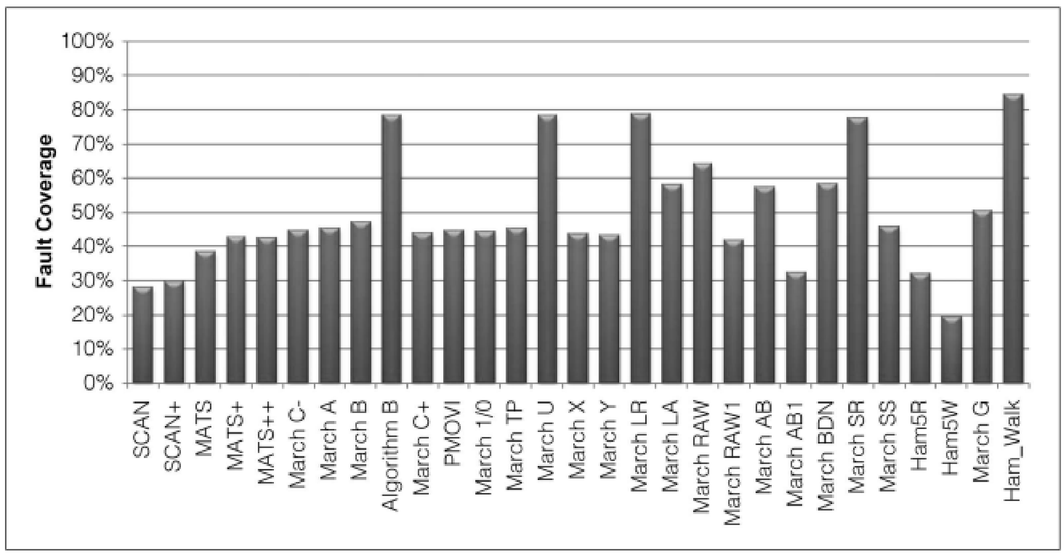

Figure 2.1. Fault coverage of memory test algorithms [8].

Table 2.1. Algorithm notation [8].

#	Algorithm	Sequence
1	SCAN	{⇕(w0);⇕(r0);⇕(w1);⇕(r1)}
2	SCAN+	{⇧(w0);⇧(r0);⇧(w1);⇧(r1);⇩(w0);⇩(r0);⇩(w1);⇩(r1)}
3	MATS	{⇕(w0);⇕(r0,w1);⇕(r1)}
4	MATS+	{⇕(w0);⇧(r0,w1);⇩(r1,w0)}
5	MATS++	{⇕(w0);⇧(r0,w1);⇩(r1,w0,r0)}
6	March C-	{⇕(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1);⇩(r1,w0);⇕(r0)}
7	March A	{⇕(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}
8	March B	{⇕(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0)}
9	Algorithm B	{⇕(w0);⇧(r0,w1,w0,w1);⇧(r1,w0,r0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,r1,w0)}
10	March C+	{⇧(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0); ⇩(r0)}
11	PMOVI	{⇩(w0);⇧(r0,w1,r1);⇧(r1,w0,r0);⇩(r0,w1,r1);⇩(r1,w0,r0)}
12	March 1/0	{⇧(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇧(w1);⇧(r1,w0,r0); ⇩(r0,w1,r1)}
13	March TP	{⇩(w0);⇧(r0,w1);⇧(r1,w0);⇩(r0,w1,r1);⇩(r1,w0,r0)}
14	March U	{⇕(w0);⇧(r0,w1,r1,w0);⇧(r0,w1);⇩(r1,w0,r0,w1);⇩(r1,w0);⇩(r0)}
15	March X	{⇕(w0);⇧(r0,w1);⇩(r1,w0);⇕(r0)}
16	March Y	{⇕(w0);⇧(r0,w1,r1);⇩(r1,w0,r0);⇩(r0)}
17	March LR	{⇕(w0);⇩(r0,w1);⇧(r1,w0,r0,w1);⇧(r1,w0);⇧(r0,w1,r1,w0);⇧(r0)}
18	March LA	{⇕(w0);⇧(r0,w1,w0,w1,r1);⇧(r1,w0,w1,w0,r0);⇩(r0,w1,w0,w1,r1);⇩(r1,w0,w1,w0,r0);⇩(r0)}
19	March RAW	{⇕(w0);⇧(r0,w0,r0,r0,w1,r1);⇧(r1,w1,r1,r1,w0,r0);⇩(r0,w0,r0,r0,w1,r1);⇩(r1,w1,r1,r1,w0,r0);⇕(r0)}
20	March RAW1	{⇕(w0);⇕(w0,r0);⇕(r0);⇕(w1,r1);⇕(r1);⇕(w1,r1);⇕(r1);⇕(w0,r0);⇕(r0)}
21	March AB	{⇕(w1);⇩(r1,w0,r0,w0,r0);⇩(r0,w1,r1,w1,r1);⇧(r1,w0,r0,w0,r0);⇧(r0,w1,r1,w1,r1);⇕(r1)}
22	March AB1	{⇕(w0);⇕(w1,r1,w1,r1,r1);⇕(w0,r0,w0,r0,r0)}
23	March BDN	{⇧(w0);⇩(r0,w1,r1,w1,r1);⇩(r1,w0,r0,w0,r0);⇧(r0,w1,r1,w1,r1);⇧(r1,w0,r0,w0,r0);⇧(r0)}
24	March SR	{⇩(w0);⇧(r0,w1,r1,w0);⇧(r0,r0);⇧(w1);⇩(r1,w0,r0,w1);⇩(r1,r1)}
25	March SS	{⇕(w0);⇧(r0,r0,w0,r0,w1);⇧(r1,r1,w1,r1,w0);⇩(r0,r0,w0,r0,w1);⇩(r1,r1,w1,r1,w0);⇕(r0)}
26	Ham5R	{⇧(w0);⇧(w1,$r1^5$);⇧(w0,$r0^5$);⇩(w1,$r1^5$);⇩(w0,$r0^5$)}
27	Ham5W	{⇧(w0);⇧($w0^5$,r0);⇧($w1^5$,r1);⇩($w0^5$,r0);⇩($w1^5$,r1)}
28	March G	{⇕(w0);⇧(r0,w1,r1,w0,r0,w1);⇧(r1,w0,w1);⇩(r1,w0,w1,w0);⇩(r0,w1,w0);***D***;⇧(r0,w1,r1);***D***;⇧(r1,w0,r0)}
29	Ham_Walk	{⇧(w1);⇧(w0);⇧(r0,w1,r1,w0,r0);⇧(r0,w1);⇧(r1,w0,r0,w1,r1);⇧(r1)}

Memory testing algorithm basically can be divided into 3 components, algorithm's direction, background and notation. Direction indicates whether the algorithm needs to be run X directory or Y directory of memory. Meanwhile background indicates type of data should to perform the testing such as checker broad, column stripe or solid. Finally, the notation indicates the algorithm flow of write and read. Each algorithm need to be repeated with different background and direction [4] [9].

2.4 Type of Memory Faults

Faults may occur due to mass production, electrical errors, manufacturing, logical error or random fluctuations in device parameters [10]. The memory faults able to categories into 3 major faults, memory cell faults, dynamic faults and address decoder faults as shown in Figure 2.2 [11].

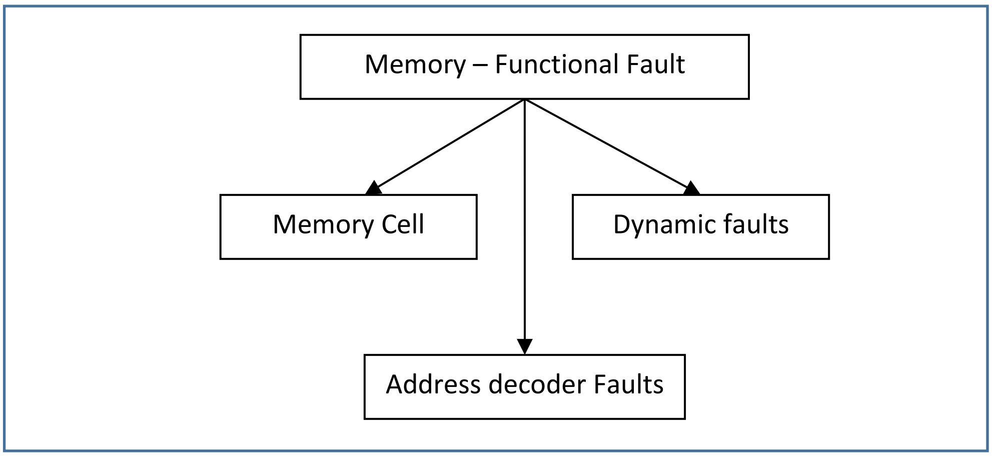

Figure 2.2. Memory Faults [11].

2.4.1 Memory Cell Faults

There are number of different type memory cell faults defects happen in memory, some of them as describe below:

1. Stuck-at fault (SAF), where the cell or line is stuck at VCC or Ground [11].
2. Stuck-open fault (SOF), open cell or broken line within the memory. [21]
3. Transition fault (TF), the cell fails to transit from 0 to 1 or vice versa [11].
4. Data retention fault (DRF), cell changes the value after some period [21].
5. Coupling fault (CF), any operation on aggressor cell impact the victim cell [4].
6. Bridging fault (BF), short between cells [11].
7. Neighborhood Pattern Sensitive Fault (NPSF), the surrounding cell causing the base cell to changes its logic value [11]

2.4.2 Address Decoder Faults

Type of faults that could happen in the address decoder:

1. No cell accessed by certain address [11].
2. Multiple cells accessed by certain address [11].
3. Certain cell not accessed by any address [11].
4. Certain cell accessed by multiple addresses [11].

2.4.3 Dynamic Faults

Type of dynamics faults in memory:

1. Recovery faults, slow in transit from the previous state within the memory [21].

2. Disturb faults, any write or read operation on cell causing the cell itself or others to change its logic value.

Any type of effort taken to improve the memory testing methodology should not impact the fault coverage or quality of the memory. The sequence of the tests algorithm for RAM tests coverage only can be adjusted or modified base on the proper study on the yield data and DPM value.

2.5 Memory Faults Detection Base on Algorithms

Table 2.2. summarizing conducted survey in [10] presents algorithms used to cover the basic faults in memory testing. Some popular kinds of failure on memory are defined such as Stuck at Fault (SAF), Transition Fault (TF), Address Decoder Fault (ADF), Coupling Fault (CF).

Table 2.2. Algorithms to Detecting Memory Faults

Test	Fault Coverage
MATS+	SAF
March Y	SAF, TF, ADF, some CFs, some linked TFs
March X	SAF, TF, ADF, some CFs
March LR	Also linked faults
March A	SAF, TF, ADF, some CFs, some linked CFs
March LA	SAF, TF, ADF, some CFs, some linked faults

Anyway, the algorithms in Table 2.2 are not enough to cover all memory faults that are relating to patterns of neighborhood cells [12]. The data of memory cell is changed due to the neighboring memory cell during write or read is called as Neighborhood Pattern Sensitive Fault (NPSF). [4]

2.6 Test Time Base on Algorithm

Table 2.1 shows the notations for memory testing algorithms. Based on the notations, the test time can be calculated for each algorithm. Prior to that, the algorithm notation need to be understand. The notation briefly described in [13] [20]. The algorithm sequence is delimited by the parentheses. Any sequence within these parentheses need to be completed to entire memory before moving next sequence. The up, down and double side arrow is indicating the addressing sequence of algorithm operation. The up-arrow sequence will execute the operation from first address to last address [9]. Meanwhile for double side arrow, the sequence can be executed either up direction or down direction. The W0, R0, W1 and R1 notation represent write data 0 into cell, read and compare with data 0, write data 1 and read data 1 respectively. A D notation in March G algorithm indicate delay between operation sequence [13].

The test time table will be created based on number of sequence in the algorithm with respect to the memory size (N). To calculate the actual test time for the memory, the size of memory and executing clock frequency must be determined in prior. Table 2.3 and Figure 2.3 show the execution cycles for selected algorithm.

Table 2.3 Algorithm Cycles

Algorithm	Cycles	Algorithm	Cycles
SCAN	4N	March Y	8N
SCAN+	8N	March LR	14N
MATS	3N	March LA	14N
MATS+	5N	March RAW	26N
MATS++	6N	March RAW1	13N
March C-	10N	March AB	22N
March A	11N	March AB1	11N
March B	17N	March BDN	22N
Algorithm B	17N	March SR	14N
March C+	14N	March SS	22N
PMOVI	13N	Ham5R	25N
March 1/0	14N	Ham5W	25N
March TP	11N	March G	23N + 2D
March U	14N	Ham_Walk	15N
March X	6N		

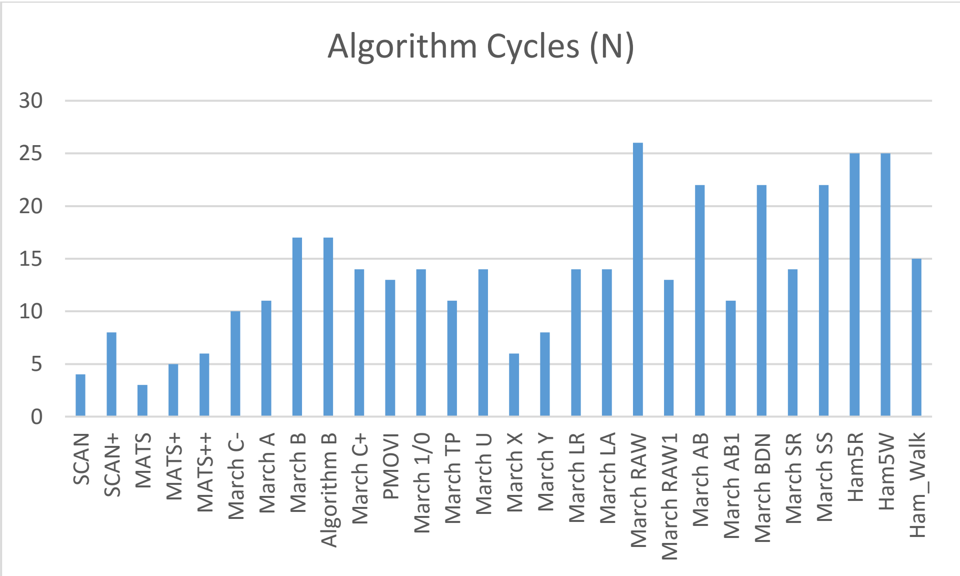

Figure 2.3 Algorithm Cycles Count

With the time model for each algorithm, the tests time able to be calculated as below formula:

Test Time = (Algorithm Cycle x Total Address of Memory)/Frequency

Figure 2.4 shows the test time required for each algorithm for 1Mbyte memory with 100Mhz functional or testing frequency.

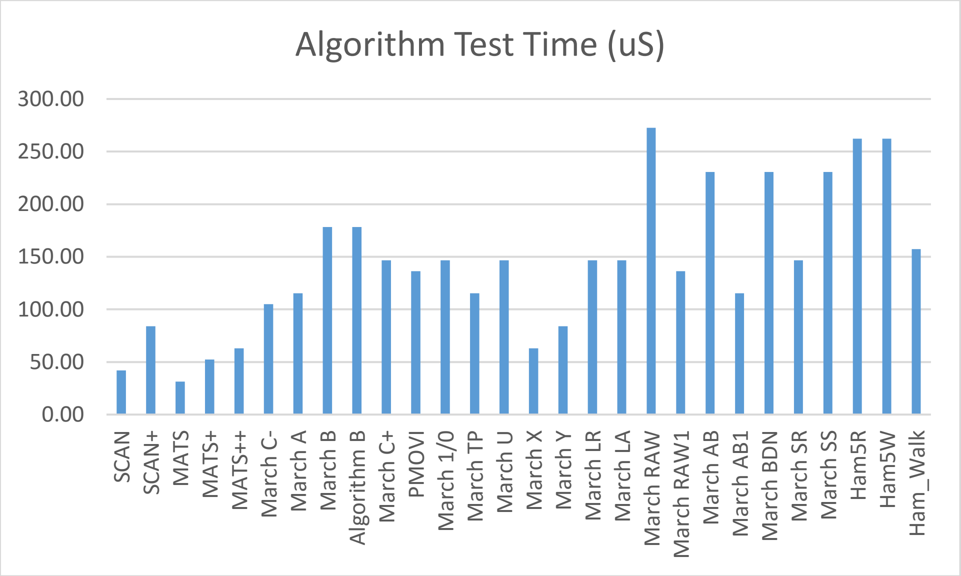

Figure 2.4. Algorithm Test time.

2.7 Built-in Self-Test (BIST) for Memory

Built-in Self-Test is one of design for testability (DFT) features. Memory BIST (MBIST) is automated testing for memory. It reduces the effort of write and read operation manually through external pin. By having this features, the memory testing can be run at variable speed using phase loop lock of the test device [2]. During failure analysis, BIST able to identify the exact failing address to locate the exact location in physical design [14].

Memory BIST consist of data and control generator, address generator, state machine and comparator. There are few type of memory BIST in industrial such as soft-programming MBIST, hard-programming MBIST, reconfigurable MBIST and self-repair MBIST.

Soft-programming Memory BIST capable of changing the algorithm sequence without impacting the design changes to the Memory BIST controller. This features become handy during debugging the failure at tester [6]. For Memory BIST with built-in redundancy analysis (BIRA) is capable of repair the memory with spare row and column during testing [7]. For this research, the hardcoded algorithm Memory BIST is selected due to the design overhead and easy to developed.

Besides that, there are few types of interface used for MBIST such as Internal Joint Test Action Group (IJTAG), Joint Test Action Group (JTAG) and external pin scan chain [15].

During the Memory BIST insertion into the device, the placement of Memory BIST and grouping of the memory is key parameter in design planning to reduce the overhead on physical device and influences the tests time [16]. The main factor of memory grouping are memory shape, depth, power and edge distances between memory [17].

There are few types of Memory BIST architecture available in industrial, each of Memory BIST have its own advantage and disadvantage. Main factor that involve in choosing suitable the MBIST are testing frequency, area for DFT design, type of memory and type of algorithm.

2.8 Test Time Reduction Method

There are few method have been developed to reduce the memory testing test time. Some of the method have been discussed in following section.

2.8.1 Memory Test Time Reduction by Interconnecting Test Items

The interconnect test items method able to reduce the test time by sharing or reuse the algorithm sequence between tests. By this, the algorithm can be simplified to use the state of previous algorithm to continue with current algorithm sequence. This able to save the initialization and verification sequences in the algorithm. Figure 2.5 shows the flow chart of the combination of few algorithms [18].

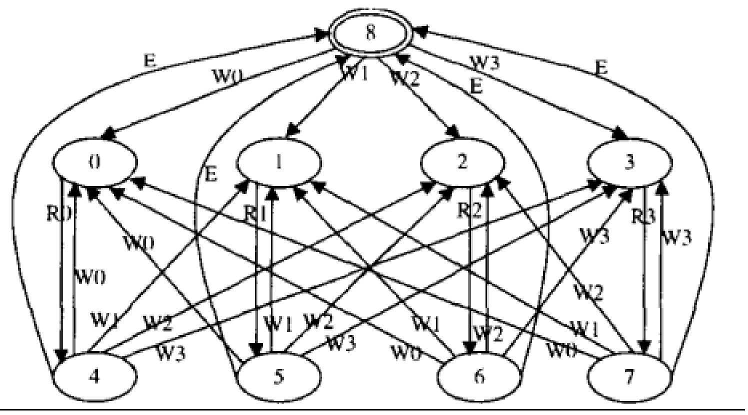

Figure 2.5. Flow of algorithm state with Interconnecting

On the disadvantage, this method is difficult to be developed for tests due to complexity in algorithms sequence. Besides that, the tests lose the details of the failing symptom of the algorithms. Without this data, the fault coverage for product unable to be identified and unable to remove the redundant algorithm base production data [14].

2.8.2 Remove the Ineffective Algorithm Base on Production Data

This method is driven by production data. During the memory testing in production flow, the testing will be halt whenever a failing detected in the one of the algorithm in the flow. After certain sample of size achieved, any algorithm with 0 fallout or low than DPM tolerance will be removed from the flow for test cost saving.

2.8.3 Remove the Redundant Algorithm in Production Flow

This method is driven by production failing data. An algorithm is mark as redundant whenever two or more algorithms are failing on same device. For examples, 20 units is failing on algorithm A and 10 units out of 20 units is failing on the algorithm B. The algorithm B is subtest fault coverage of algorithm A. Since the 10 units still able to fail during algorithm A testing, the algorithm B is call as redundant and removed in the flow. This analyses only can carry out after a certain number sample size is achieved [19]. Figure 2.6 illustrates the scenario.

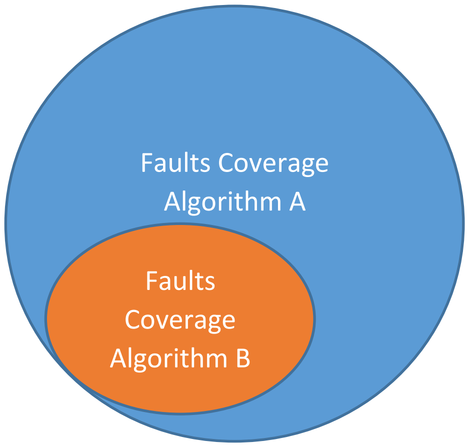

Figure 2.6. Faults coverage algorithm B is subtest of algorithm A

2.9 Summary

Some of past research on memory test time reduction have been discussed in this chapter. Base on the outcome, a summarizes table 2.4 have been put up for comparisons.

Table 2.4. Comparison of Memory Test Time Reduction Method.

		Test Time Saving method		
Memory Test Time Reduction Method	**Test Implement Turnaround**	**BIST**	**Test Removable**	**Test Trimming**
Remove the Redundant Algorithm in Production Flow	Low, the methodology is useable for any new product	Yes	Yes	No
Remove the ineffective Algorithm base on production data	Low, the methodology is useable for any new product	Yes	Yes	No

Memory Test Time Reduction by Interconnecting Test Items	High, required high turnaround time to develop and implement the tests.	Yes	No	Yes
Using specific BIST for test time reduction	Average, required change the architecture of the BIST base on algorithm changes.	Yes	Yes	No
The proposal method	Low, the methodology is useable for any new product	Yes	Yes	Yes

Based on the comparison table above, the proposal method has an advantage compare to the rest of the method discussed in the previous section. Even though the proposal method briefly discusses about the algorithm trimming methodology in this research, the architecture of the BIST and testing methodology are carefully defined to accommodate all others existing saving method. The algorithm trimming methodology is an add-on to existing test time saving method for better reduction in the memory testing time.

The algorithm trimming method is not implemented in most of existing method except memory test time reduction using Interconnecting test item. But, the interconnect test item has different trimming methodology compare to the proposal method. The interconnects method able to reduce the test time by reusing same sequence between the algorithms, while the algorithm trimming methodology will remove the ineffective sequence in the algorithm based on yield data. Furthermore, the interconnect method has high turnaround time of test development for any new process node. While, the proposed method is reuse for any new process node except the production yield data.

CHAPTER 3

METHODOLOGY

3.1 Overview

In the previous chapter, background of the memory testing with algorithm and impacted of test time have been studied and discussed. In this chapter, a strategy and methodology is proposed for effective way to optimize the memory algorithm testing to reduce tests time for low cost product.

Due to cost, the low-cost product has acceptable number of reject unit from customer end which refer to DPM number (Defects per million). By utilizing this tolerance and proper risk assessment, the algorithm can be optimized for tests time saving without impacting the quality of product delivered to customer.

This methodology is developed using the Quartus software tool, Verilog programming language and test program for ATE tester.

The following design flow is used to developed the proposed test methodology:

1) Analysis and Calculation of DPM

2) Develop BIST (Built-in Self-Test) with failure report capability

3) Develop test program flow

4) Analysis on data

A detail flow chart of the methodology process is shown in Figure 3.1. With the planning through the flow chart, every objective of the stage involved in the research can be achieved effectively.

In the stage, which is "**Analysis and Calculation of DPM**", the tolerance of allowable reject at customer end is calculated. This number is required to measure the product quality risk by modifying the test algorithm.

For "**Develop BIST with Failure Report Capability**" phase, the design architecture with Quartus software tools are described and discussed in details. Besides that, the design is simulated using VCS tools to make sure the functionality of the BIST.

In "**Develop Test Program Flow**" phase, the step to implement the tests in production flow in ATE tester is discussed. Besides that, the methodology of data logging in database have been discussed.

In the final stage of flow which is "**Analysis Output Data**", the output data from the tester is analyzed to identify the location of trimming in testing algorithm sequence. Besides that, the test time saving is calculated based on the memory size and modified testing algorithm.

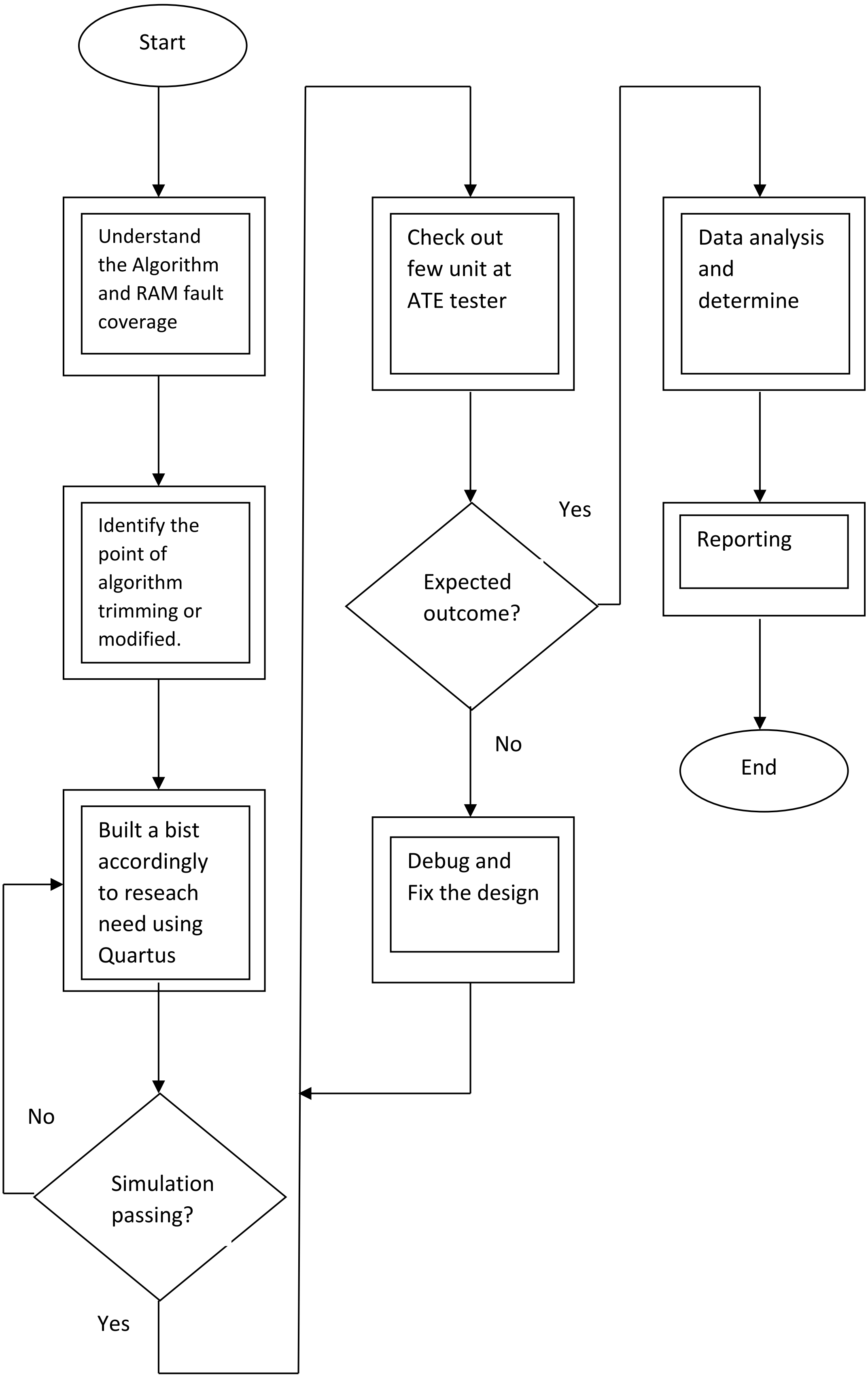

Figure 3.1. Methodology Flow Chart.

3.2 Analysis and Calculation of DPM

Defects per million, is a measurement of the defect rate in certain product. Its refer to failures which are time related, meaning units which are expected die at the customer. High-end or high-cost product have tighter DPM number compare to low-cost or low-product.

One of cost item in low-cost product is testing cost. Testing cost carry all factor or cost related to testing include tester, test engineer cost and test time. To reduce the test cost, the low-cost product usually has loose testing specification compare to high-cost product. This is one of factor that contribute to higher DPM number for low-cost product.

Formula to calculate DPM:

DPM = (a / b) x 1,000,000; where

a = Quantity of units with defects.

b = Quantity of units tested.

Example if given a product A have tolerance of 500 DPM, this product can have reject of 500 units out of 1 million units delivered at customer due time related degradation or over stress. During initial stage, rough estimation done per the formula above due to small sample size.

Example if the product A have 15 reject units from the 10000 units delivered to customer. The DPM number can estimate as (15/10000) x 1000000 which equal to 1500 DPM. Per examples, this is not acceptable since allowable DPM number is 500.