REDUCED GALLOPING COLUMN ALGORITHM FOR MEMORY TESTING

NGIENG SIEW CHING

UNIVERSITI SAINS MALAYSIA

2015

REDUCED GALLOPING COLUMN ALGORITHM FOR MEMORY TESTING

By

NGIENG SIEW CHING

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

August 2015

ACKNOWLEDGEMENTS

This dissertation is dedicated to everyone in the field of memory semiconductor research who embarks the journey of expanding the collection of knowledge and great passion for continuous improvement of test time reduction in memory testing in the semiconductor industry.

First of all, I would like to express my gratitude to Dr. Zaini Abdul Halim, my dissertation advisor and supervisor, for seeing the promise of this thesis and achieving research conducted under her watchful eyes. Besides, her invaluable support, guidance and insightful advice has resulted in the completion of this project.

My special thanks to my colleagues in INTEL, Ukchukphan Saeng Fiserm, Pui Poh Khong and Kumar Suriya A for their informative inputs and technical support during development of this project. They shared ideas by providing incessant information on the research techniques and skills. Because engineering is not a collection of knowledge, I truly appreciate their feedback and suggestion in providing me a deeper and better understanding to the project.

In addition, appreciations are dedicated to Soh Meng Wah and Yii Wen Wen for their inputs, moral supports and friendships. They contributed valuable suggestions in the discussion when I faced problems in this project development and implementation. Their efforts are greatly appreciated.

Last but not least, I offer my regards and blessing to my beloved family who supported me in any aspect throughout this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AF	Address-Decoder Fault
ATE	Automated Test Equipment
BIST	Built-In-Self-Test
CE	Consumer Electronics
CF	Coupling Fault
CPU	Central Processing Unit
DFT	Design for Test
DPM	Defect Per Million
DRAM	Dynamic Random Access Memory
DUT	Design under Test
FC	Fault-Coverage
FFMs	Functional Fault Models
FP-MBIST	Full-Speed Field Programmable MBIST
GalCol	Galloping Column
GalDiag	Galloping Diagonal
GALPAT	Galloping Full Pattern

GalRow	Galloping Row
HDMT	High Density Modular Tester
HVM	High Volume Manufacturing
IO	Input Output
JTAG	Joint Test Action Group
KB	Kilo-Byte
L2	Level 2
MBIST	Memory Built-In-Self-Test
NPMBIST	Non-linear Programmable Memory BIST
NPSF	Neighborhood Pattern Sensitive Fault
PBIST	Programmable Built-In-Self Test
RAM	Random Access Memory
RF	Register File
ROM	Read Only Memory
SAF	Stuck-At-Fault
SOC	System-On-Chip
SRAM	Static Random Access Memory
TF	Transition Fault

TTR	Test Time Reduction
VCS	Verilog Compiler Synopsys
6T	6-Transistor

ALGORITMA PENGURANGAN LAJUR "GALLOP" UNTUK

PENGUJIAN MEMORI

ABSTRAK

Pengujian memori amat penting dalam perkembangan pantas teknologi bagi sistem dalam cip. Ini disebabkan oleh peningkatan memori kapasiti dalam rekaan cip yang semakin kecil, kompleks dan berkuasa rendah. Masa pengujian untuk memori dalam cip merupakan satu cabaran yang hebat untuk mencapai produk yang berkualiti tinggi dan kos rendah dalam masa ke pasaran yang singkat. Pengurangan masa pengujian untuk memori amat penting kerana kos pengeluaran dalam industri amat bergantung pada masa pengujian produk dalam mesin. Terdapat banyak algoritma yang boleh digunakan untuk menguji memori, termasuk lajur "gallop" (GalCol). GalCol merupakan algoritma yang penting untuk mengesan kecacatan gandingan dan peralihan yang unik. Namun demikian, algoritma GalCol asal memakan masa pengujian yang lama. Algorithma pengurangan lajur "gallop" dibangunkan untuk mempercepatkan masa pengujian memori. Algorithma pengurangan GalCol mempunyai ciri ciri algoritma yang sama dengan GalCol asal. Perbezaan ketara dalam pendekatan algoritma pengurangan GalCol adalah pengurangan bilangan pergerakan sel sasaran. Pergerakan sel sasaran dihadkan pada 8, 16 dan 32 sel untuk setiap sel asas. Projek ini melibatkan dua peringkat, iaitu pembangunan perisian menggunakan perisian INTEL dan Synopsys dan pembangunan algoritma pengurangan GalCol dalam process pengeluaran milik INTEL. Memori L2 SRAM bersaiz 64KB dalam 15 cip telah diuji dengan algoritma pengurangan GalCol. GalCol X8 algoritma mencapai pengurangan masa pengujian memori yang tertinggi, iaitu sebanyak 79.5% dan 75.7% pada frekuensi 600MHz dan 1.6GHz dan ketepatan hasil yang selaras dengan GalCol asal.

REDUCED GALLOPING COLUMN ALGORITHM FOR MEMORY TESTING

ABSTRACT

Memory testing is significantly important nowadays especially in SOC's design, due to their rapid growth in the memory density and design complexity in smaller chip area and low power design. Thus, test time in memory testing is a key challenge to accelerate time to market, high yield and low test cost in high volume manufacturing. Test time reduction in memory testing is important in industry, as test cost is directly related to validation time of each product on the tester. There are lots of memory algorithms used for memory testing, including the galloping column algorithm (GalCol). The GalCol algorithm test is important to detect unique coupling and transition faults. However, the existing GalCol algorithm takes huge test time due to its test complexity. To overcome the test time issue in industry, reduced GalCol algorithms with solid data background are proposed. The reduced GalCol algoritms have similar test behavior as original GalCol algorithm with major difference in the number of galloping of the target cells. The galloping of target cells are reduced to first and last 8, 16 and 32 of cells of every base cell. This project is progressed in two stages, which are the software development using INTEL software and Synopsys tool and test implementation on INTEL production flow. These algorithm are verified on 15 units of 64KB L2 SRAM memory. In this project, test time reduction and consistent pass fail test results are achieved in the reduced GalCol algorithm tests. The GalCol X8 algorithm obtains the highest test time reduction of about 79.5% at 600MHz and 75.7% at 1.6GHz with consistent pass or fail test results comparable to original GalCol algorithm in the HVM test flow.

CHAPTER 1

INTRODUCTION

1.1 Background

Nowadays, system-on-chip (SOC) designs that combines processing core, graphics, audio, video and input output components (IO) onto a single chips are increasingly important on consumer electronic market segments. Example are tablet and smartphone devices (Vermeulen & Goossens, 2014). The consumer electronics (CE) manufacturers spend tremendous effort to develop lower power, more intelligent, higher performance and cost-effective SOC designs and to accelerate time to market. Figure 1.1 and Figure 1.2 show the table market segment and smartphone platform roadmap by INTEL

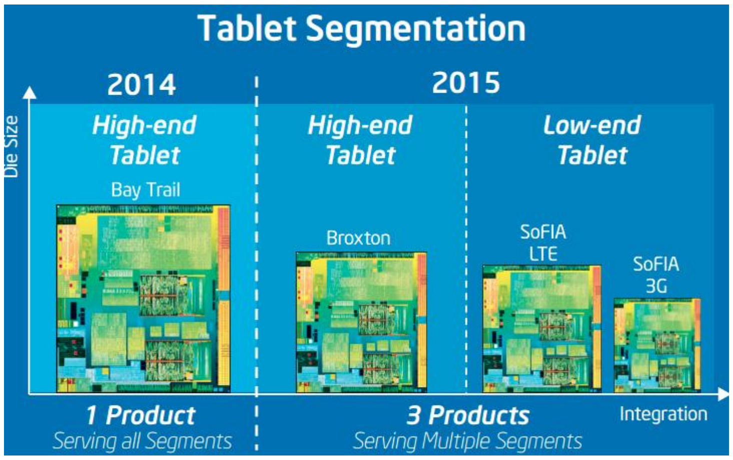


Figure 1.1: Tablet segmentation by INTEL

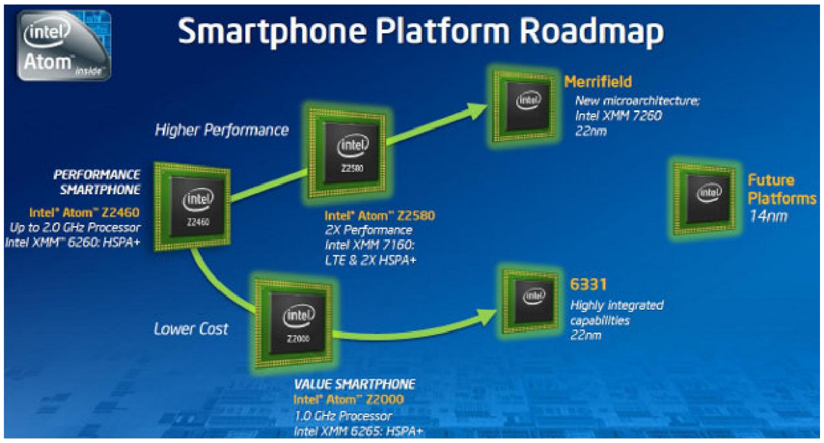


Figure 1.2: Smartphone platform roadmap by INTEL

Nowadays, the embedded memories occupy over 90% of the most system-on-chip (SOC) designs in 2014 (Nisha & Siva, 2014), which is shown in Figure 1.3 The numbers and variety of embedded memories such as random access memory (RAM), read only memory (ROM) and registers file memory (RF) are increasing tremendously in SOC designs. Figure 1.4 shows the semiconductor trends, which gate density will be increasing exponentially over years with rapid improvement in data rate speed.

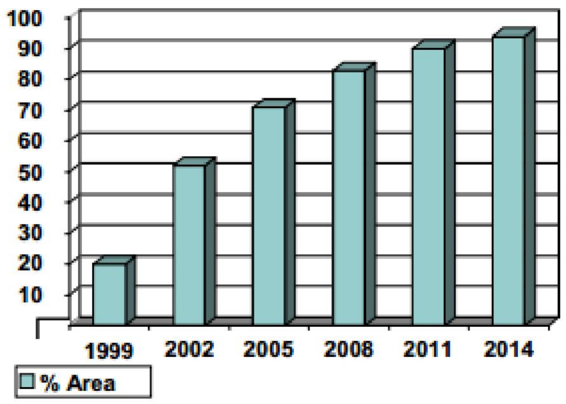


Figure 1.3: Chip are occupied by memories (ITRS, 2007)

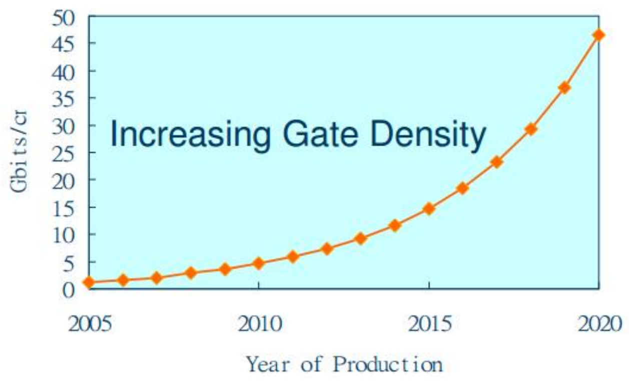


Figure 1.4: Semiconductor trends in gate density (ITRS, 2007)

The rapid growth of memory density, design complexity, speed and reduction of SOC chip area leads to challenges for SOC high volume manufacturing (HVM) especially on testing time, costs and yields impact. The test costs are largely driven by test time factor. Test time is the time consumption for validating each product on tester. The test time of SOC designs are increasingly due to tremendous growth in memory size, memory types and different design for test (DFT) methodologies. With the rapid growth in test time, HVM test costs will be higher. This will impact time to market and costs of the SOC products. Thus, memory test time reduction (TTR) has been an important and critical issue especially in HVM environment that requires extremely low defect per million (DPM) budget on entire SOC.

The typical flow of memory testing in HVM is complex and time consuming to ensure high yield and good quality products, which is shown in Figure 1.5 (Zaid et al., 2008). The flow are divided into 2 main phases, which are the wafer test (also named as sort test) and package test (also named as class test). During the wafer test phase, it consists of cache structural test, where each internal functional blocks are tested individually instead of testing the die as whole system through its input/output pins (I/O).

Normally wafer test stage has lesser pins than the package test phase. Fault localizing tests, which identifying the failing cells are run in this stage in order to repair the failing cells with redundant row or column. During the package test phase, cache is retested using the fault detecting tests to eliminate the fault chips from the HVM flow. Functional testing is the final stage in the package test flow, where the entire chip is validated at-speed through its I/O pins. The purpose of this testing is to guarantee the functionality of the processor and to categorize different chips according to their speed and power consumption, which is called as speed and power binning. The entire memory testing in package test flow consumes significant test time.

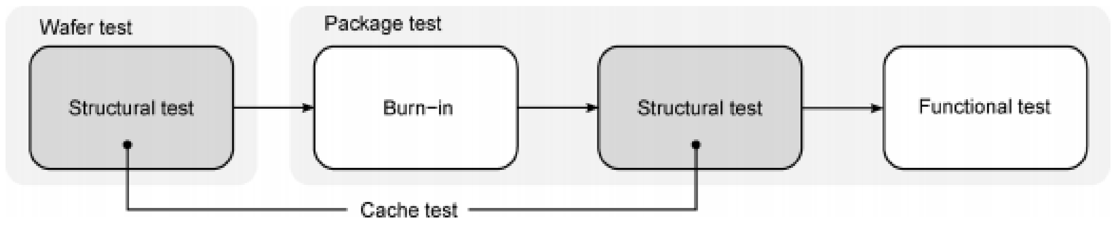
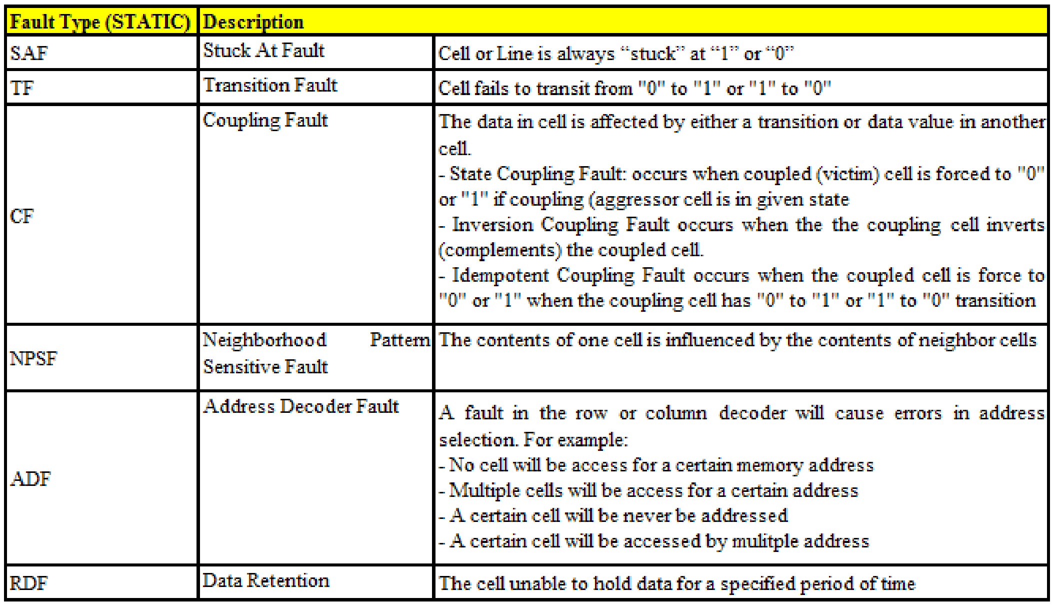


Figure 1.5: Typical representation of a microprocessor test flow (Zaid et al., 2008)

Although the test time reduction (TTR) issue is challenging in HVM flow, there are some useful heuristic memory tests that still can be implemented to improve test time and provides high fault coverage. There are lots of well-known memory algorithms widely used for Built-In-Self-Test (BIST) SOC memory testing in industry, such as SCAN (4N), Partial Moving Inversion (13N), March-C (10N), Mats++ (6N) and Galloping algorithms ($4N^2$) (Michael et al., 2013). GALPAT, Walking 1/0 and Sliding Diagonal are classified as complex and conventional test algorithms (Van & A.J, 1999). All these different algorithms in industrial can detect different unique faults coverage, such as stuck-at-fault (SAF), transition fault (TF), coupling fault (CF), address-decoder fault (AF) and neighborhood pattern sensitive fault (NPSF) (Rusli et al., 2012). Common functional static fault types are described in Table 1.1.



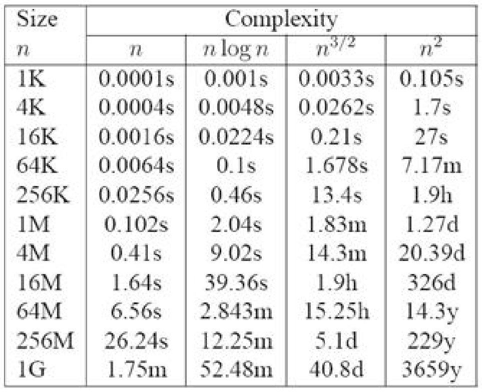
Fault Type (STATIC)	Description	
SAF	Stuck At Fault	Cell or Line is always “stuck” at “1” or “0”
TF	Transition Fault	Cell fails to transit from "0" to "1" or "1" to "0"
CF	Coupling Fault	The data in cell is affected by either a transition or data value in another cell. - State Coupling Fault: occurs when coupled (victim) cell is forced to "0" or "1" if coupling (aggressor cell is in given state - Inversion Coupling Fault occurs when the the coupling cell inverts (complements) the coupled cell. - Idempotent Coupling Fault occurs when the coupled cell is force to "0" or "1" when the coupling cell has "0" to "1" or "1" to "0" transition
NPSF	Neighborhood Pattern Sensitive Fault	The contents of one cell is influenced by the contents of neighbor cells
ADF	Address Decoder Fault	A fault in the row or column decoder will cause errors in address selection. For example: - No cell will be access for a certain memory address - Multiple cells will be access for a certain address - A certain cell will be never be addressed - A certain cell will be accessed by mulitple address
RDF	Data Retention	The cell unable to hold data for a specified period of time

Table 1.1: SRAM functional fault types (Ashokkumar & Subhash, 2014)

1.2 Problem Statement

Galloping algorithm is serving as an effective test for reliable minimum voltage searches, maximum frequency searches and detecting most faults in HVM flow. The tests stress the interaction between the base cell under tests and the remaining cells of the memory. It can detect and locate all SAF’s, TF’s, AF’s and CF’s with single galloping test (Grzegorz et al., 2011). The galloping test can be isolated within row, column or diagonal, which use to derive galloping column (GalCol), galloping row (GalRow) and galloping diagonal (GalDiag) tests. This helps to reduce the galloping test complexity. Galloping tests require test time of n^2 order, where n is the memory size in n bits. From Table 1.2, complex test algorithm with n^2 order are significantly higher than other non n^2 tests with assumption a cycle time of 100nS. Testing a 256 Kbit memory on the chip will require 1.9 hours for test with complexity of n^2. Thus, although the galloping test can detect most of the silicon faults, the test time for galloping test is a major concern in achieving TTR effort in mass production. The huge test time will cause significant impact on the time to market and test costs of the SOC products.

Hence, a modified galloping algorithm that focusing on GalCol is investigated in order to improve the memory test time validation especially on the SOC's high volume manufacturing.



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

Table 1.2: Test time as function of memory size with cycle time of 100nS (Arvind, 2012)

1.3 Objectives

The project is conducted to achieve the objectives as below:

1. To develop reduced GalCol algorithm on simulation and implement on HVM production test flow.
2. To study the effectiveness of the reduced GalCol algorithm in term of test time reduction as compared to original GalCol algorithm.
3. To study the effectiveness of the reduced GalCol algorithm in term of product failing or passing status as compared to original GalCol algorithm. The test status indicates whether memory under test is fault-free or not.

1.4 Scope of project

This project covers the software development of Reduced GalCol algorithm and test implementation on HVM test production flow. Memory testing using other test algorithms are not implemented in this project.

The reduced GalCol with solid data background tests are investigated and modified from existing GalCol test which is used in industrial. For this project, reduced Galcol algorithm analysis is specifically targeted on reducing galloping address by 32, 16 and 8, instead of galloping all cell in the same column as base cell. The reduced GalCol tests are developed using INTEL built-in software and simulation verification is done using Synopsys Verilog Compiler (VCS) tool. Synopsys Verdi debug tool is used, which provides powerful debugging features, such as waveform viewer that helps to analyze the design behavior and the algorithm activity over time and source code browser for design signal tracing.

To implement the reduced GalCol tests on HVM production test flow, automated test pattern generation flow is studied. The reduced GalCol tests from simulation are converted to tester friendly format using INTEL automated test pattern generation tools. The automated test equipment (ATE) used in this project is limited to INTEL in-house tester, named as High Density Modular Tester (HDMT). Despite there are many different ATE, HDMT is chosen because it is cheaper and HVM production test costs are lower by 10X. This tester provides test time collection capability and pattern editor to modify the test pattern on tester, which are suitable for the application of this project.

There are lots memory sizes and types in SOC designs. However, in this project, test time reduction analysis of reduced GalCol algorithm is targeted only on 64KB L2 Data SRAM with single-port six transistor (6-T) memory design, which using 14-nm

INTEL technology. The L2 Data SRAM is chosen because it is among the biggest memory in the SOC designs. The memory of design under test (DUT) is tested only under room temperature 25’c in two different core frequencies, 600MHz and 1.6GHz on pre-burn-in flow after wafer test and die packaging. The test time analysis of reduced GalCol algorithm is specifically focus on test time taken for memory write and read operations, and not covering test time analysis for taken for HVM reset or tester setup time.

To study the effectiveness of reduced GalCol algorithm, test status pass or fail on HVM test flow is being investigated in this project, which telling if the DUT is fault-free or not. This project will not be covering the fault identification, which identify the failing location of memory cell and the type of detected fault.

1.5 Report Outline

This report is organized into five main chapters that explain the details from the introduction to conclusion of reduced GalCol test algorithm project. The first chapter is describing the introduction of this project, which covering the project background, problem statement, objectives and scope of the project.

Chapter 2 discusses on previous works in the memory test algorithms done by past researchers. The relevant fundamental background of previous research is also explained in this chapter.

In chapter 3, the proposed reduced Galloping Column (GalCol) test algorithm is introduced, which includes the methodology for software development and test implementation on HVM test flow. The methodology for software development explains on pseudo-codes, test coding and simulation of reduced GalCol algorithm. This chapter also describes HVM test pattern generation methods in details, test flow implementation and validation on ATE.

Chapter 4 presents the experimental results and discussion of this project. This chapter shows the results to proof the efficiency of the reduced GalCol algorithm. The test time reduction and DUT pass or fail test status are analyzed in this chapter.

Lastly, chapter 5 presents the conclusion of this project. Summary of the project implementation and achievements are explained. This chapter also outlines the limitation and future research works that needed for the reduced GalCol test algorithm.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

As discussed in previous chapter, over 90% of SOC's chip area is occupied by diverse memory. The tremendous growth of memory density, capacity and design complexity leads to large challenges in test time, test cost and yield in SOC high volume manufacturing, which eventually will impact the product time to market and product cost effectiveness. Thus, memory test time issue is long time research issue and unceasing efforts in high volume manufacturing, as product test cost is directly driven by the product validation time on tester. This chapter covers the past researches and analysis that have been carried out on test time reduction efforts, memory algorithm tests and built-in-self test (BIST) that is used by algorithm testing to achieve test time reduction.

2.2 Test time reduction efforts

There are lots of test time reduction approaches are carried out to solve the test time issue in memory testing in semi-conductor industry. A study was performed to remove redundant test patterns by adapting two methods. The two methods were named as fault detectability analysis and fail label analysis, in order to achieve memory test time reduction while maintaining the fault detectability (Wu et al., 1996). According to authors, well-defined fault model was needed in fault detectability analysis, based on which fault detectability of all test were analyzed. The fault coverage evaluation are done from the result of the analysis. On other hand, the fail label analysis began with fail labels collection of each failing DUT, which this information was useful to analyze dependence relation between the tests. The result of the study showed that these two approaches are just dual to each other, which encouraging the test engineer in industry to perform test time reduction by analyzing fail labels in the test result directly without any knowledge of the fault model. The fault detectability analysis is useful for academic field since the faults are analyzed thoroughly and fault models are well defined. Whereas the fail label analysis is more practical for industrial field.

Another study had been carried out on memory test time reduction. The author proposed to interconnect all test patterns, which re-using the memory states left from previous test (Wu et al., 2000). The objectives of this study was to minimize the tester settling time and share the sequences initialization time between two consecutive test patterns being applied on the tester. The analysis of interconnecting the test item was categorized into the three main parts, which were the initialization sequences, fault activation and verification sequences and pure verification sequence. For the initialization sequence part, there was commonly a sequence of write operations to initialize memory cells into certain initial value. Next part, was the fault activation and verification sequences. This key part of a test consisted a sequence of read and write operations in certain addressing order to activate, detect the designated faults and verify the result. The last part was the pure verification sequence, which aimed to verify the results through a sequence of read operations. The commonality between part 1 and part 3 was, they care only the state of memory and do not focus on how to perform their operations or reach their state. Furthermore, part 2 was usually started with read operations, similar to part 3, which could also be used to verify memory state of previous state. Thus, part 1 and part 3 were eliminated and replaced by memory states, where memory state from previous test can be reused as initialized state for the following test. However, there was unique case, where the verification sequence may not be always sharable and could not be classified as part 3. The reason was the verification sequence at end of test need to be executed with specific addressing order, operation mode or voltage or timing condition. Thus, this is the constraint of the interconnecting all test in the production.

In addition, Junichi presented a novel method for memory test time reduction by installing virtual tester hardware on the tester central processing unit (CPU) memory (Junichi, 2001). The study showed that this approach helped to delete the duplicated tester hardware setting instructions, such as waveform timing setting, input output voltage setting and pin conditional setting, which required a long setup time amounting to a 0.3 to 1.0 mili-second. The virtual tester hardware was comprised of memory that houses the addresses that enabled it in order to identify the different tester hardware parts and their actual setting values. The comparison between value that was already set on the real tester hardware and value sent from tester CPU was performed in each instruction of the test program. If the values are identical, the test program proceeded to next instruction. If the values are different in test instances, the test program ran again to set the value in the real tester hardware. Thus, the results proved the test time reduction of 5% to 25%, which verified the effectiveness of this method. Moreover, the test time can be reduced by minimizing the number of times the power supply was turned ON and OFF. The power-supply voltage will be turn OFF, reset with new value and was turned back ON if the existing test module had different set voltages values compared to previous test module.

Hence, this TTR approach is practical and greatly adapted in high volume manufacturing environment.

Besides that, a systematic approach was studied and proposed for memory test time reduction. The authors discussed on identifying redundant test patterns in the test flow, then removing and/or combining the test patterns to minimizing the overall memory test time (Yeh et al, 2005). From this study, the original tests can be removed or merged and new tests can be developed without losing the fault coverage. The authors had also developed a TTR tool, which was useful for test time reduction for industrial products, which is illustrated in Figure 2.1.

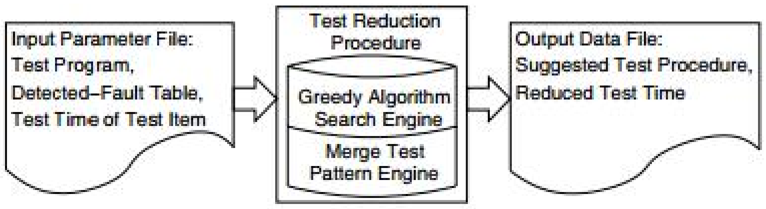


Figure 2.1: The proposed TTR tool (Yeh et al., 2005)

This TTR tool required the input parameter file and test reduction procedure, which eventually produces the output data file. The test program, detected-fault table and test time of each test are the important input parameter files. The detected-fault table consisted of the detected faults for each test. The output data file provided the recommended reduced test procedure and the reduced test time. In the test reduction procedure, the authors proposed the greedy algorithm based search engine and the test pattern merging engine. The greedy algorithm based search engine is illustrated in Figure 2.2. The weight was generated by sorting the tests based on test time and the weight was distinguished by number of faults detected if there were tests that had same test time. The test with shortest test time will have the minimum weight. The minimum weight test was put into solution set using the greedy approach and removed from test set. The fault detected for the minimum weight test was also removed from fault set. This step was continued until fault set was empty. The final solution set was the reduced test set. Thus, this heuristic TTR algorithm could identify the redundant test patterns, suggested a proper test list and provided the correlation between the test items. The results of the study showed the overall test time reduction about 19.5% without reducing the fault coverage on an industrial DRAM chip on a DRAM tester. However, it is challenging to obtain the accurate fault-detected table as input parameter file for this TTR tool implementation, especially on the new or in-mature process technology.

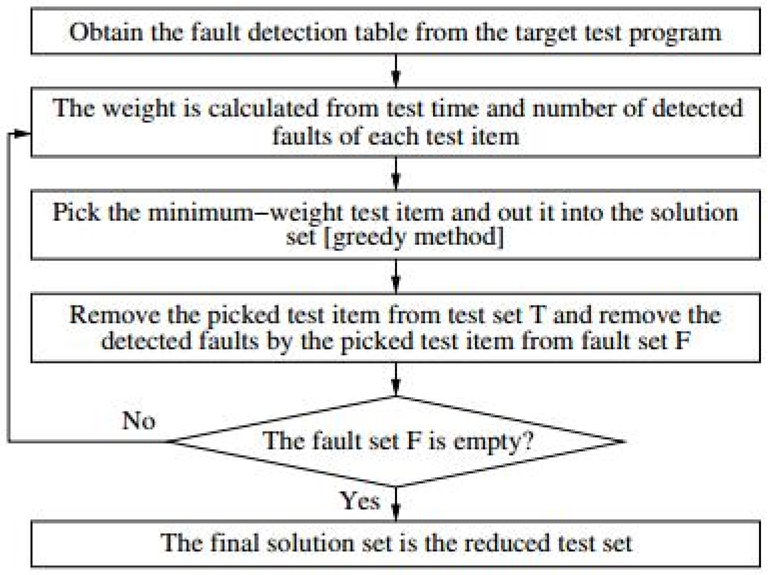

Obtain the fault detection table from the target test program
The weight is calculated from test time and number of detected faults of each test item
Pick the minimum−weight test item and out it into the solution set [greedy method]
Remove the picked test item from test set T and remove the detected faults by the picked test item from fault set F
No
The fault set F is empty?
Yes
The final solution set is the reduced test set

Figure 2.2: The proposed greedy algorithm (Yeh et al., 2005)

Furthermore, a novel architecture for test time reduction in March-based algorithms was proposed by enabling in parallel more than one words of the RAM during the write operations (Voyiatzis et al, 2012). The recommended architecture needed additional write drivers in the driving logic, the gates required to enable the outputs of the decoder in parallel and additional flip-flops in address generator. Figure 2.3 illustrates the general structural of SRAM and the modification on driving logic. The results of the study showed the reduction in test time, ranges from 25% to 60% depending on the March algorithm with an increase of less than 2,5% in hardware overhead for 1Kbyte SRAM. There were few advantages of the proposed approach, such as it did not have much impact on the cell arrays of the RAM, which was carefully optimized with respected to area and delay as well as not affecting on normal RAM operation.

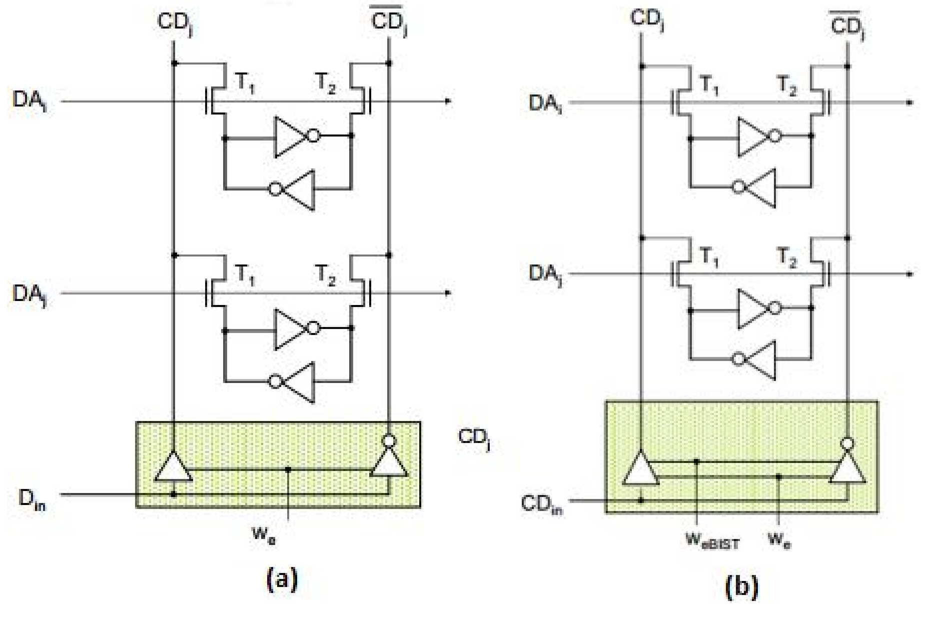


Figure 2.3: (a) General structural of SRAM; (b) Modified driving logic (Voyiatzis et al, 2012)

2.3 Memory test algorithms analysis

Throughout the years, there are lots of researches and developments of memory test algorithms such as SCAN, March C-, March SR, MATS++, PMOVI, Hammer, GALPAT, GalCol, and so on (Nisha & Siva, 2014). The performance and effectiveness of heuristic test algorithms are confirmed through simulation. In the high volume manufacturing environment, an efficient selection of memory test algorithms is crucial to accomplish high fault coverage and low test time at the same time.

A comprehensive study on memory test algorithm in industry had been presented. In the study, 12 well-known and 3 fault-primitive-based memory test algorithms were performed at INTEL, which is listed in Table 2.1 (Hamdioui et al., 2006). The "n" denoted the size of memory cell array, C was the number of columns and R was the number of rows. (^) and (v) were the increasing or decreasing address order. An operation applied to a cell can be "w0" (write 0), "w1" (write 1), "r0" (read 0) or "r1" (read 1). The results of the study showed that the tests with most promising fault coverage in theory had similar highest fault coverage in real silicon practice. However, the results revealed that some algorithms, such as GALPAT and Walking 1/0 detected faults that cannot be explained with existing fault models and lacked of fault models.

The 12 sets of traditional algorithms that selected in the case study had most promising fault coverage and unique faults detected. March C tests have become the dominant type of tests in memory testing due to their high fault coverage and linear complexity in test application time (Dharma et al, 2013). Many different flavor of March tests had been introduced, including the fault-primitive March SS, March Raw and March SL algorithms (Zaid et al., 2008)

Table 2.1: Description list of the used algorithm tests (Hamdioui et al., 2006)

Algorithms	**Test Length**	**Description/Short notation**
Fault-primitive algorithms		
March SS (Hamdioui et al., 2002)	22n	^(w0); ^(r0,r0,w0,r0,w1);^(r1,r1,w1,r1,w0); v(r0,r0,w0,r0,w1);v(r1,r1,w1,r1,w0);^(r0)
March RAW (Hamdioui et al., 2002)	26n	^(w0);^(r0, w0,r0,r0,w1,r1); ^(r1,w1,r1,r1,w0,r0);v(r0,w0,r0,r0,w1,r1); v(r1,w1,r1,r1,w0,r0);^(r0)
March SL (Hamdioui et al., 2004)	41n	^(w0);^(r0,r0,w1,w1,r1,r1,w0,w0,r0,w1); ^(r1,r1,w0,w0,r0,r0,w1,w1,r1,w0); v(r0,r0,w1,w1,r1,r1,w0,w0,r0,w1); v(r1,r1,w0,w0,r0,r0,w1,w1,r1,w0)
Traditional algorithms		
SCAN (Abadir & Reghbati, 1983)	4n	^(w0);^(r0);^(w1);^(r1)
MATS+ (Nair, 1979)	5n	^(w0);^(r0,w1);v(r1,w0)
MATS++ (Breuer et al., 1976)	6n	^(w0);^(r0,w1);v(r1,w0,r0)
March C- (Marinescu, 1982)	10n	^(w0);^(r0,w1);^(r1,w0);v(r0,w1);v(r1,w0);v(r0)
PMOVI (Jonge et al., 1976)	13n	v(w0);^(r0,w1,r1);^(r1,w0,r0);v(r0,w1,r1);v(r1,w0,r0)
March SR (Hamdioui et al., 2000)	14n	v(w0);^(r0,w1,r1,w0);^(r0,r0);^(w1); v(r1,w0,r0,w1);v(r1,r1)
March G (Suk et al., 1981)	23n	^(w0);^(r0,w1,r1,w0,r0,w1);^(r1,w0,w1); v(r1,w0,w1,w0);v(r0,w1,w0);^(r0,w1,r1);^(r1,w0,r0)
Hammer10 (Van et al., 1999)	49n	^(w0);^(r0,10*w1,r1);^(r1,10*w0,r0); v(r0,10*w1,r1);v(r1,10*w0,r0)
GalColumn (Breuer et al., 1976)	6n + 4nR	^(w0);^b(w1b, col(r0,r1b), w0b); ^(w1);^(w0b,col(r1,r0b),w1b)
GalRow (Breuer et al., 1976)	6n + 4nC	^(w0);^(w1b, row(r0,r1b), w0b); ^(w1);^(w0b,row(r1,r0b),w1b)
WalkColumn (Van et al., 1999)	8n + 2nR	^(w0);^(w1b, col(r0),r1b,w0b); ^(w1);^(w0b,col(r1),r1b,w0b)
WalkRow (Van et al., 1999)	8n + 2nC	^(w0);^(w1b, row(r0),r1b,w0b); ^(w1);^(w0b,row(r1),r1b,w0b)

March SS was developed based on fault simulation, which targets all simple static memory faults (Hamdioui et al., 2002). Simple faults are the faults which cannot influence the behavior of each other. Fault masking cannot happen as the behavior of a simple fault cannot change the behavior of another one. This algorithm initializes the entire memory cell. Each cell is then read twice, written back to data, read again and then write inverse data before moving to next cell. The operations occur for both data and data complement in both incrementing and decrementing addresses. A last read at the end verifies the last read happens correctly.

March RAW was also developed based on fault simulation (Hamdioui et al., 2002). The fault model used for the test development was the single port, simple dynamic fault and the test detects read after write (RAW) failures. The algorithm will write the array to initial data and then perform a read data, write the data, read the data again twice, write the inverse of the data and then read it before changing to the next address. The read and write operation happen for both data and data complement in both increment and decrement address sequences.

Furthermore, March SL algorithm which developed based on fault simulation is used to detect all simple linked faults (Hamdioui et al., 2004). The faults that will influence the behavior of each other are defined as linked faults. Fault masking can occur as the behavior of certain fault can change the behavior of another, which causes complexity in linked fault testing as compared to simple fault testing.

The GALPAT (galloping pattern) algorithm with fast X-addressing and solid data-background can detect and locate all address faults, stuck-at-faults, transition faults and coupling faults in memory cells. This test is a robust non-linear algorithm, with test complexity of $4n^2$ (Arvind, 2012). Thus, the test time for this algorithm is also larger than other traditional algorithm, as test time is driven by the memory size and test length or complexity. The galloping algorithms begin by initializing the entire array to a known data background. The galloping action starts by writing the base cell to data complement followed by a read of all target cells. Finally the base cell is restored to original value. Once the entire set of cells selected as targets have been validated, the base cell is incremented and the galloping action begins again. This sequence will continues until all the cells selected to be base cells have been tested (Prasanna & Saroja, 2014). For a full galloping pattern, the target set of cells include all other cells in a given block other than the base cell.

Due to test complexity and test time constraint of GALPAT algorithm, galloping column (GalCol) and galloping row (GalRow) algorithms that are less complexity are also used in memory testing. The GalCol algorithm behaves exactly similar as full gallop except the read action is restricted to only the cells in the same column as the base cell instead of galloping through entire memory. Figure 2.4 illustrates the test sequences of the GalCol algorithm. The reduction of the target cells shorten the test time compared to the GALPAT and makes it production test program worthy. The GalCol algorithm has been a reliable test for active minimum voltage searches and frequency maximum searches for data collection. This algorithm can detect same detects as GALPAT but some coupling faults can be missed.

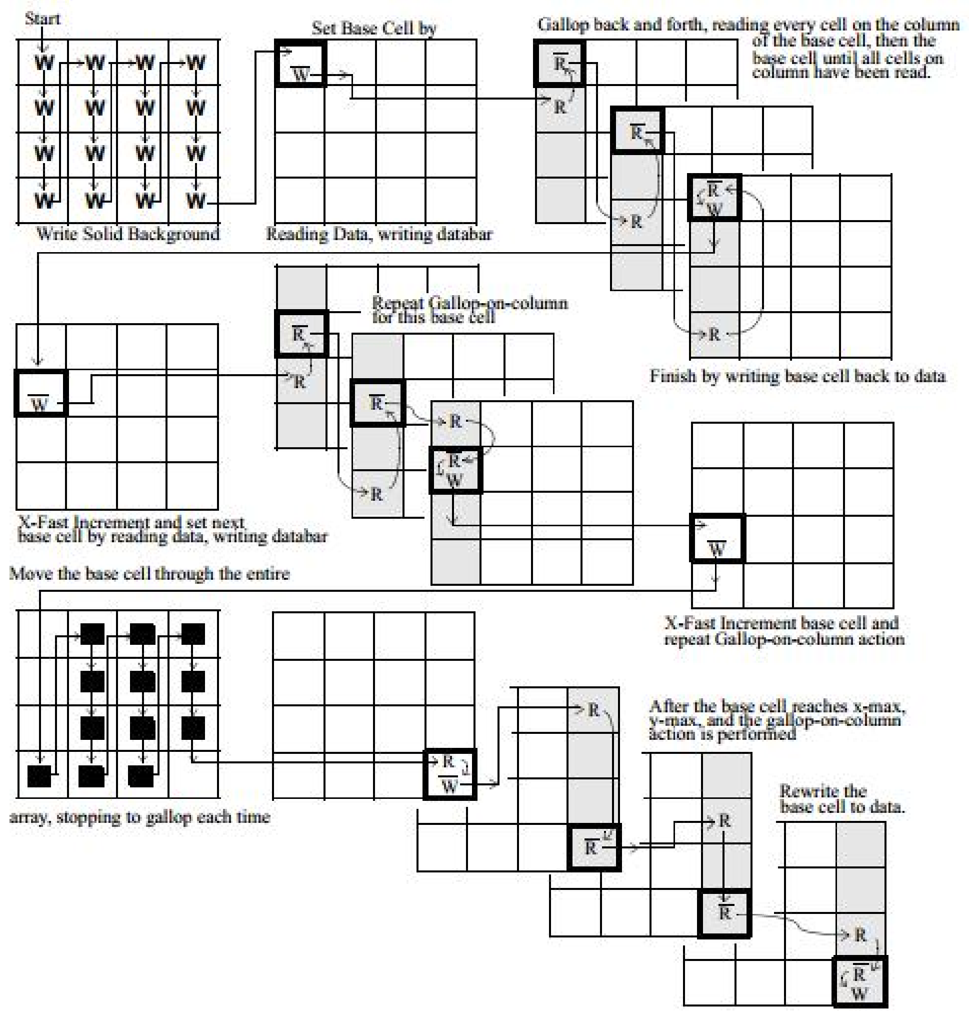

Start
Write Solid Background
Set Base Cell by
Reading Data, writing databar
Gallop back and forth, reading every cell on the column of the base cell, then the base cell until all cells on column have been read.
Finish by writing base cell back to data
Repeat Gallop-on-column for this base cell
X-Fast Increment and set next base cell by reading data, writing databar
X-Fast Increment base cell and repeat Gallop-on-column action
Move the base cell through the entire
array, stopping to gallop each time
After the base cell reaches x-max, y-max, and the gallop-on-column action is performed
Rewrite the base cell to data.

Figure 2.4: Galloping column algorithm

Similar comprehensive research on test algorithm effectiveness and classification had also be performed (Michael et al., 2013). However, the number of test algorithms was restricted to 29 linear marching algorithms, which did not include the complex and non-linear algorithms such as GALPAT, GalCol, GalRow and Walking I/0. Reason was because the Memory BIST (MBIST) used in the experiment only supported marching algorithms. The objectives of the study was to obtain high fault coverage with low test time through classification of algorithms. In this study, the authors discussed about statistical data analysis in term of fault coverage, efficiency and similarity, classification and grouping of the algorithms. The percentage of faults that are detected by algorithm in relation to the total number of faults are defined as fault coverage (FC). If the fault coverage is high, an algorithm is characterized as effective test. However, the higher fault coverage of specific algorithms is not necessary related to many complex faults discovery, but can also detect the simple faults and more different types of faults. The efficiency and similarity of test algorithms in a set was measured by taking the ratio of intersection and union of faults detected. The groups of similar algorithms were allocated to the functional fault model (FFMs), which helped in combining the test algorithms efficiently. The results of the study showed that at least 13 test algorithms would be needed to detect all the faults, such as the simple static single-cell faults, simple static coupling faults, linked faults and dynamic faults. However, this will require huge effort for HVM testing because MBIST needs to be reconfigured for each algorithm in the test set. Thus, this approach is suitable when only few test algorithms can be used to meet limited test time budget and high fault coverage goal.

2.4 BISTs for memory test algorithm

There are lots research being performed to study on the design for test (DFT) implementation for memory testing and validation strategy. Built-in-self-test (BIST) is one of the most extensively used for DFT technique. It offers full speed test application, high fault coverage, extensive diagnostics and less sophisticated in tester hardware (Neelima & RangaCharyulu, 2014).

Programmable built-in-self test (PBIST) is a key evolution from hard-wired BIST. A study have been carried out to discuss on comprehensive array DFT strategy to achieve extremely low defect per million (DPM) for entire arrays by (David et al., 2004). In the study, PBIST was used to test large array in Level 2 (L2 Data, Tag, LRU and State arrays), which is shown in Figure 2.5. PBIST is small micro-code machine connected to fairly sophisticated address generation logic, distributed data generation and comparison logic. PBIST is accessible and programmed using JTAG controller that can be applied with nearly any test platform for at-speed production testing, including functional or structural ATE testers, burn-in drivers, or even on a system board that has JTAG access. Besides, PBIST is also a powerful DFT to generate all the algorithmic tests and to raster memory data at-speed for fault diagnosis, memory repair and yield improvement. Furthermore, PBIST is capable of looping and repeating a set of instructions continuously while instructing the address generation logic to modify the address through each loop. A few programming code in PBIST (normally less than 16) are used to construct long and complex algorithms. Thus, the PBIST discussed by the author can be widely used in production memory testing due to its flexibility in generating all algorithmic tests with different data background and addressing, at-speed testing to detect subtle faults, high fault-coverage and low area overhead. It also helps in test time reduction and management because different set of algorithms can be implemented at different phases of test, such as wafer sort, package and burn-in test.

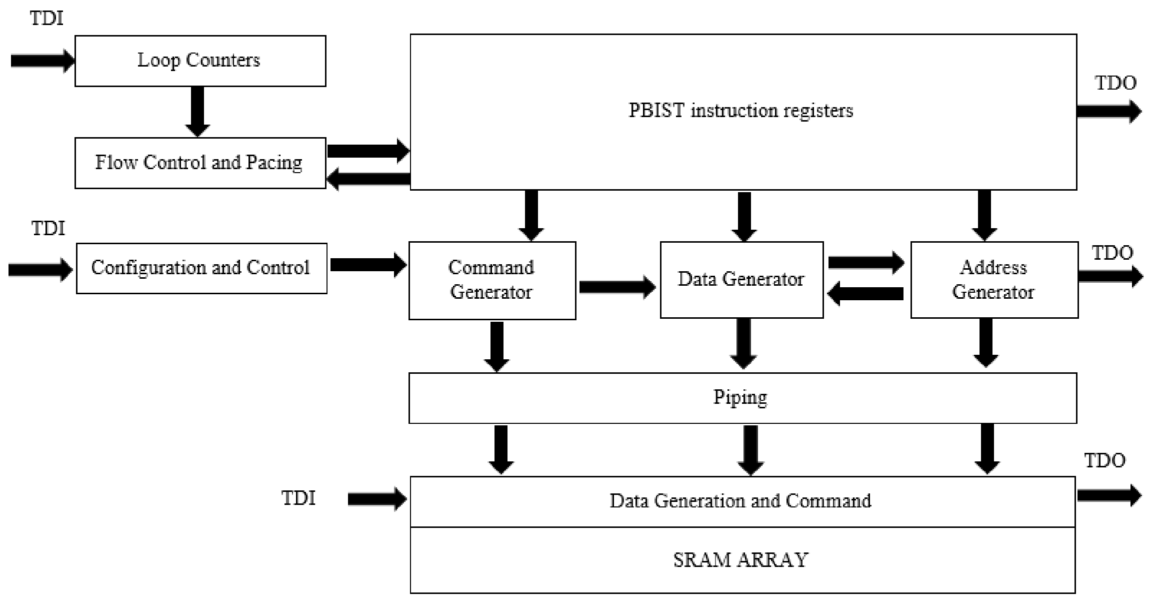


Figure 2.5: The PBIST architecture (David et al., 2004)

Another new multiple-buffer BIST methodology was proposed, which capable to concurrently test a set of memory with different sizes that is shown in Figure 2.6 (Jone et al., 2002). The RSMarch algorithm was investigated to enable all buffers to share the same BIST controller. The fundamental concept of RSMarch algorithm was to tolerate redundant inputs and output responses in order for all buffers to receive the same control and data signals from the BIST controller even if the buffer sizes were different. Nevertheless, this redundant operations do not affect the fault coverage obtained by the non-redundant operations. With this new method, the test requirements of low hardware overhead, short test time and high fault coverage can be achieved. This proposed approach can be implemented on the SOC design that consists of various memory with different sizes.

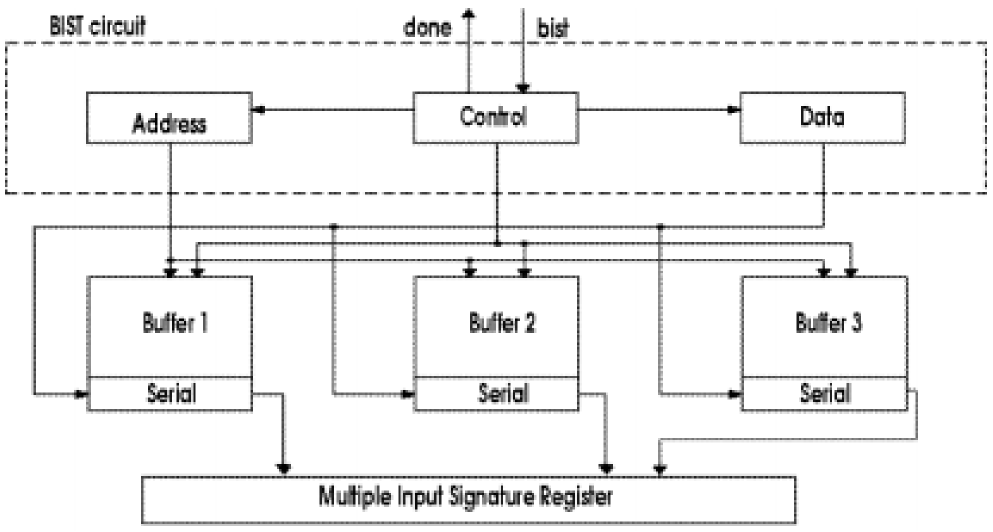


Figure 2.6: The proposed BIST architecture for multiple RAMs
(Jone et al., 2002)

Programmable memory BIST (MBIST) is another commonly used DFT for memory test, which can be classified into two main categories: FSM-based and micro-based (Kamran & Shambhu, 1999). Pre-selected elements representing different combination of memory operations from conventional March algorithms are encoded in an FSM-based programmable MBIST, which supports mostly linear algorithms and has less area overhead. Whereas, micro-code based programmable MBIST only encodes the basic memory operations instead of encoding March elements, which provides flexibility to program any linear and non-linear algorithms but with a higher area overhead.

A full-speed field programmable MBIST (FP-MBIST) controller was implemented for March algorithms and also some non-linear algorithms, such as Galloping, Walking, Sliding Diagonal and Butterfly algorithms (Du et al., 2005). Figure 2.7 illustrates the top-level architecture of FP-MBIST, which consists of instruction