IMPROVED TEST TECHNIQUES FOR NETWORK-ON-CHIP BASED MEMORY SYSTEMS

Bibhas Ghoshal

IMPROVED TEST TECHNIQUES FOR NETWORK-ON-CHIP BASED MEMORY SYSTEMS

Thesis submitted to the
Indian Institute of Technology, Kharagpur
for the award of the degree

of

Doctor of Philosophy

by

Bibhas Ghoshal

under the supervision of

Prof. Indranil Sen Gupta

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

JANUARY 2015

APPROVAL OF THE VIVA-VOCE BOARD

Date: / / 20

Certified that the thesis entitled **"IMPROVED TEST TECHNIQUES FOR NETWORK-ON-CHIP BASED MEMORY SYSTEMS"** submitted by BIBHAS GHOSHAL to the Indian Institute of Technology, Kharagpur, for the award of the degree of Doctor of Philosophy has been accepted by the external examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

(Member of DSC) (Member of DSC) (Member of DSC)

(Supervisor)

(External Examiner) (Chairman)

CERTIFICATE

This is to certify that the thesis entitled **"Improved Test Techniques For Network-on-Chip Based Memory Systems"***, submitted by* **BIBHAS GHOSHAL** *to the Indian Institute of Technology, Kharagpur, for the award of the degree of Doctor of Philosophy, is a record of bona fide research work carried out by him under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for the award of the degree of Doctor of Philosophy in accordance with the regulations of the Institute.*

Professor Indranil Sen Gupta

Date:

DECLARATION

I certify that

a. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in writing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

f. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

BIBHAS GHOSHAL

ACKNOWLEDGMENTS

This thesis would not have taken shape without the help and contribution from my well wishers. Thus, I would like to thank all who have contributed directly or indirectly in realizing this thesis. First of all, I want to thank my family, especially my parents, my wife, my son, my in-laws and my brother who have always been by my side and have inspired me and helped me in every possible way during my PhD. It was the dream of my parents that I pursue a career in research and earn a PhD. I sincerely thank them for nurturing such a dream and the support they provided to achieve it. No words of praise are enough for my wife, Runa, who took all the responsibilities of the family on herself while I was busy with my research. She has been my pillar of strength and motivation. I would like to earnestly thank her for being on my side at every moment. I would also like to thank my son Arya, who never complained about lack of enough interaction with his father but motivated me in my work. I would also like to thank my in-laws and my brother for providing me the mental support needed during my PhD work.

I thank IIT Kharagpur and the Department of Computer Science and Engineering (CSE) for providing me with the opportunity of pursuing my PhD, along with research facilities which were of the very best quality. I am grateful to my supervisor Prof. Indranil Sen Gupta, whose guidance and motivation made this thesis into a reality. He not only provided me technical guidance on my research, but also taught other aspects of research work, such as managing resources, improving writing skills, delivering presentations etc. The knowledge gained from him helped me in every stage of my PhD career and at the same time developed me into a better individual. I sincerely thank Professor Chittaranjan Mandal, Dept. of CSE, IIT Kharagpur for mentoring me during my PhD

work. He not only provided technical guidance but also helped me in improving the interaction skills. I would also like to thank Professor Santanu Chattopadhay, Dept. of E&ECE, IIT Kharagpur for providing me an oppurtunity to work with him. He was kind enough to allow me to be a part of his research group and shared the tools with me. He was also responsible for improvement of my technical document writing skills.

I sincerely thank the undergraduate and graduate students who worked with me on various problems over the duration of my PhD – Anurag Kayal, Dhawal Gadiya. I would also like to thank Soham and Bodhisatwa, students of Netaji Subhas Engineering college, who helped me in coding while working on one my research problems. I am particularly grateful to Subhadip Kundu and especially Kanchan Manna, for the help they provided during my PhD work. The discussions and interactions with them were very helpful and were responsible in determining the course of the PhD. I would like to thank my friends Arghya, Subhasis, Rishiraj, Sumanta and especially Dharmendra for making my stay at IIT Kharagpur a fun-filled and memorable one. I also express my sincere gratitude to the faculty members and staff of CSE, IIT Kharagpur.

Finally, I am grateful to God for his blessings and for giving me the strength to persevere throughout the long and arduous journey towards a PhD.

Bibhas Ghoshal

Kharagpur, India

ABSTRACT

The search for a cost effective interconnect architecture and increasing communication demand in Systems-on-Chip (SoC) designs have paved the route to Network-on-Chip (NoC) research a decade back. The Network-on-Chip (NoC) communication architecture is a packet based network where cores communicate among themselves by sending and receiving packets. High parallelism, smaller latency in data transmission and facility of Intellectual Property (IP) re-use have made NoCs overcome the problem of bandwidth and latency in conventional bus-based interconnects. However, like all other designs, NoC based SoC designs must also be tested for defects. Survey of different test techniques developed for NoC based systems suggest that focus has been primarily on finding improved test techniques for NoC based logic cores. However, the embedded memory content in NoC based systems have increased over the years and will continue to increase. Due to their high density, these embedded memories are more prone to defects than other type of on-chip circuits and therefore, require more importance when it comes to testing NoC based systems. In this thesis, the objective has been to devise cost effective test techniques for memory modules in a NoC based memory system, targeting minimum area overhead at optimized test time and test power.

The research was aimed at improving the existing approaches of test of NoC based memory cores along the following directions : test architecture, test scheduling algorithms and on-line test techniques. A distributed test architecture has been proposed to allow hardware sharing and eventually reduction of Design-For-Testability (DFT) area overhead. The test technique utilized by the test hardware tries to incorporate the advantages of both parallel and serial approaches of testing embedded memories, thereby reducing the test time. For the test architecture proposed in the thesis, a test scheduling algorithm has been proposed focusing on limiting the number of concurrent test blocks

under power constraint with the aim of performing a power aware test of the memory cores. Experiments performed on ITC'02 benchmark circuit confirms that our proposed test schedule performs a more power constrained test as compared to dedicated Built-In-Self Test (BIST) techniques for embedded memories.

In addition to improvement of existing approaches, novel test architectures have also been devised. These architectures involve utilization of the existing on-chip resources, such as refresh circuit for test purpose. The refresh based test technique has been effectively utilized to perform on-line tests of embedded DRAMs interconnected by the NoC infrastructure. To perform the on-line tests, transparent March tests have been used in place of standard March tests, to ensure restoration of initial contents of the memories after test. Reusing refresh allows periodic testing of DRAM without interruption while overcoming the requirement of additional Design-For-Testability (DFT) hardware. Analytic results have been used to explore the fault coverage of the proposed test technique.

The transparent test technique has also been employed for detection of permanent faults developed in FIFO buffers during field operation of NoC. A prototype implementation of the proposed test algorithm has been integrated into the router-channel interface and the on-line test has been performed with synthetic self-similar data traffic. The performance of the NoC after addition of the test circuit has been investigated in terms of throughput using a System C based NoC simulator. Area overhead of the test circuit has been studied by synthesizing the test hardware with a industry standard design library.

To summarize, the research work presented in this thesis is an attempt to provide system-level solutions for effective power constrained testing of hundreds of embedded memories connected using NoC with low overhead in Design For Testability hardware.

Contents

Author's Biography

Bibhas Ghoshal received M.Sc degree in Electronic Science from Jadavpur University, Kolkata, in 2002, and an M.E. degree in Computer Science and Engineering from West Bengal University of Technology in 2005. He has been pursuing Ph.D. at the Department of Computer Science and Engineering, IIT Kharagpur, since January 2010. He was university topper in B.Sc examination and received merit award for excellence in M.Sc examination. Even though his Ph.D. is on Testing of NoC based memory systems, his general research interests include FPGA based system design, CAD for VLSI, Image Processing, Open source application development.

Publications made out of this thesis

(listed in reverse chronological order)

1. Bibhas Ghoshal,Kanchan Manna, Santanu Chattopadhay, Indranil Sengupta , "On-line field test for permanent faults in NoC buffers", *IEEE TVLSI,vol.PP, no.99, pp.1,1*.
2. Bibhas Ghoshal,Chittaranjan Mandal and Indranil Sengupta , "Re-using Refresh for Self-testing DRAMs", *in Proceedings of the International Symposium on Eletronic System Design (ISED 2013) held at NTU Singapore from 12-13 December, 2013*.
3. Bibhas Ghoshal, and Indranil Sengupta , "A Distributed BIST Scheme for NoC-based Memory Cores", *in Proceedings of the 16th EuroMicro Conference on Digital System Design held at Santander, Spain (4th - 6th September 2013)* pp. 567-574 .
4. Bibhas Ghoshal,Subhadip Kundu, Indranil Sengupta, Santanu Chattopadhay, "Particle Swarm Optimization Based BIST Design for Memory Cores in Mesh Based Network-on-Chip", *in Proceedings of the 16th International Symposium on VLSI Design and Test (VDAT 2012)* pp. 343-349 .

List of Figures

List of Tables

Chapter 1

Introduction

Intellectual Property (IP) based design and advancement in manufacturing technology have allowed today's Systems-On-Chip (SoCs) to include hundreds of embedded cores. In such dense SoCs, the choice of interconnect architecture is a big concern as it governs the performance, power dissipation and cost of the system. Designers using bus based interconnect network for complex System-on-chip (SoC) designs often face difficulty relating to bandwidth, signal integrity, and power dissipation of the chip. A new communication architecture called Network-on-Chip (NoC) [8] has been proposed to solve these issues. In a NoC-based chip, the cores communicate among themselves by sending and receiving packets which contain network dependent information required to route the data from its source to its destination. The communication network consists of network interfaces, routers and channels which connect the routers. Like all other SoCs, NoC based SoCs must also be tested for defects. In majority of works reported in literature, the focus has been to find improved test techniques for minimization of test time and test power at reduced area overhead for logic cores interconnected using NoC. However, embedded memory content in NoC based systems has increased from one-tenth to more than three-fourth of the chip area today and will continue to increase [103]. Due to their high density, these embedded memories are more prone to defects than other type of on-chip circuits. To the best of the knowledge, not much research has been done on exploring test techniques for NoC based memory cores.

The present thesis discusses improved techniques devised for technology-independent functional testing of memory cores interconnected using NoC. Different approaches have been employed for different types of memories, based on their respective fault models and modes of operation. Both off-line and on-line techniques have been proposed for detection of manufacturing faults and run-time faults in memories respectively. To re-use the NoC for testing interconnected cores, it must be ensured that the elements of NoC are fault free. First-In-First-Out

(FIFO) buffers present within the routers occupy significant amount of area of the NoC infrastructure. Thus, test of FIFO buffers has major significance in the test of the NoC infrastructure. The thesis also proposes improved test technique for memory modules which are part of the NoC infrastructure, namely the Static Randon Access Memory (SRAM) based FIFO buffers. Thus, the work presented in this thesis is an attempt to devise cost- effective test techniques, in terms of area overhead, test power and test time for memory modules that are interconnected using NoC as well as part of the NoC infrastructure.

1.1 Motivation

An important test issue for SoCs is the design of an efficient Test Access Mechanism (TAM). However, TAM design is not a concern during test of memory cores since they are tested by employing memory built-in-self test (MBIST) techniques, where test generation as well as comparison of results are done on-chip. Moreover, MBIST reduces pin count at system level, requires no external test equipment, reduces development efforts, tests are carried out at speed and test time gets reduced as number of cores can be tested in parallel. As a result, MBIST has been accepted as the most preferred technique for detection of faults in memories. With more and more memories embedded in circuits, accessibility becomes an issue and TAM design becomes more and more complex. The situation then leads to accepting MBIST as the solution of choice. Even for memory cores interconnected using NoC, employing MBIST is the most cost effective test technique because it avoids adding the TAM area overhead to the existing overhead of the communication infrastructure. However, MBIST for memories connected using NoC also face the same test challenges as faced by any other BIST technique for embedded memories. Unless carefully designed, NoC based MBIST may induce excessive power, in addition to performance degradation and increased area overhead [59].

The overall real estate to be devoted to the BIST circuitry is significant. As a result, there is a major incentive to reduce the area overhead. To reduce the BIST area and routing overhead, distributed approaches are necessary. However, as hardware resource sharing is introduced in distributed memory BIST, the testing technique must be carefully considered to reduce the routing congestion and to facilitate rapid power-constrained testing. Research suggests that the approaches taken so far have explored two main directions to gain improvement in performance of MBIST. One has been the parallel and serial interconnection techniques of BIST wrapper sharing as proposed in [28], [47] while the other has been memory grouping algorithms for optimized area, power and time proposed in [17], [67]. Parallel testing can reduce test time, but power consumption may be a factor. Sequential testing has the opposite effect. A balance of

both techniques may be best. In order to reduce memory power dissipation during test, proposals for low power memory BIST have been reported in [16], [30] and [99]. However, all these proposals have targeted memories in SoCs and cannot be directly applied to memories interconnected using NoC as there are a number of other constraints that need to be considered such as test time, area overhead, routing overhead, etc. Although the constraints have been tackled separately by previous researchers involving BIST design optimization for memory cores in SoCs, there are no system-level solutions for effective power constrained testing of embedded memories connected using NoC with low overhead in Design For Testability hardware. Thus, the motivation of the thesis has been to exploit specific features of the NoC architecture while applying the *hardware resource sharing* approach so that a cost effective system level solution can be proposed for testing of memory cores interconnected using NoC.

Moreover, special features and functionalities of memory cores have been explored which could be utilized for test purpose. Refresh operations require reading the contents of a memory location and writing them back to the same location. March tests for detecting functional faults in memories also require writing some patterns in to the memory and reading them back. There is a similarity in the operations performed on the memory during both refresh and word-oriented March test. The manner in which the operations need to be performed are also similar. Both require scanning the entire memory row by row and performing read followed by write operation on each row. These similarities in refresh and word-oriented March test have motivated us to re-use the refresh circuit for test purpose. Refresh circuit has earlier been used for detection of errors by Hellebrand et al. in [42] and Yarmolic et al. in [101]. However, both the works utilize refresh circuit for detection of soft errors. The soft error detection schemes do not perform active test as they do not alter the contents of the memory. Therefore, they cannot detect functional faults in memories as mentioned by Thaller in [86]. *Off-line* and *On-line* BIST architectures have been proposed which re-use refresh circuit in performing March tests [13] on the Dynamic Random Access Memory (DRAM) cores for detection of manufacturing and run-time permanent faults. Then, both the architectures are utilized in testing number of embedded DRAMs (eDRAMs) interconnected using NoC.

Significant amount of area of the present NoC data transport medium is occupied by First In First Out (FIFO) buffers. Accordingly, the probabilities of faults or defects occurring in buffers are significantly higher compared to the other components of the NoC. Thus, test process for the NoC infrastructure must begin with test of the FIFO buffers. Although researchers have given importance to detection of functional faults in FIFO buffers, the detection of in-field functional latent faults (faults which develop over time) have been overlooked. One probable reason could be the belief that with advent of deep sub-micron technology (DSM), permanent faults (faults

that occur due to permanent damage in routers/link) are not as frequent as transient faults (faults occurring at run-time) as mentioned in Hwang et al. [49]. However, recent studies of memory failures in field by Sridharan et al. [83], Schroeder et al. in [78] and Hwang et al.in [49] gave strong evidence that memories experience both transient fault (soft error) and permanent (hard error) faults in the field but permanent faults constitute bulk of all failures. It is therefore necessary to find a cost-effective test technique that can detect permanent faults developed during field operation of FIFO buffers.

1.2 Objectives of the thesis

Through this research, we have tried to provide comprehensive system-level solutions for effective power constrained testing of embedded memories interconnected using NoC and for memory modules which are part of the NoC infrastructure with high test concurrency and low overhead in DFT hardware. The objective of the thesis is as follows.

> *To devise cost effective test techniques for memory modules in a NoC based memory system, targeting minimum area overhead at optimized test time and test power. The memory modules targeted are SRAM and DRAM cores which are interconnected using NoC and FIFO buffers which are part of the NoC communication infrastructure.*

The research presented in this thesis has been aimed at improving the existing approaches of test of NoC based memory cores along the following directions.

a) *Test architecture:* For memories interconnected using NoC, the objective has been to propose a distributed test architecture to allow hardware sharing and eventually reduction of area overhead due to test circuitry. Moreover, the proposed architecture should employ a hybrid test approach incorporating the advantages of both parallel and serial testing approaches to optimize test time and test power. The objective has also been to re-use the on-chip resources such as refresh circuit for test purpose. Re-using refresh for test avoids use of extra DFT logic, bringing down area overhead. It also saves read cycles during March test operation on the DRAM, as read operations get performed during refresh. Moreover, test circuit for each type of memory should be designed for programmability, to support multiple test algorithms for higher fault coverage.

b) *Test scheduling algorithm:* For optimized time and power during test of number of memory cores, improved test architectures should be supplemented by efficient test scheduling algorithms. Thus, the aim of the test scheduling algorithm proposed in this research has

been to limit the number of concurrent test blocks under power constraints and to reduce the total time required for test.

c) *On-line test:* Recent studies of memory failures in field gave strong evidence that memories when deployed for field operation experience permanent faults more than transient faults. It is therefore necessary to find a cost- effective on-line test technique that can detect hard faults during field operation of memories. Such a test technique must possess the following characteristics :

- The test must be an active test so that functional defects are uncovered.
- The test must be performed periodically to ensure that no fault gets accumulated, thus requiring an on-line transparent test technique.
- The test hardware must be cost-effective in terms of area.

Thus, the objective has been to develop on-line tests for embedded DRAMs (e-DRAMS) that possess the above mentioned characteristics and can detect run-time faults.

d) *Modified test algorithm for on-line test :* One of the requirements of any on-line test is restoration of data after test. Therefore, the objective of the research has been to propose transparent tests for detection of permanent faults developed during field operation of DRAMs and for detection of latent hard faults which develop in FIFO buffers during field operation of NoC.

1.3 Contributions of the thesis

This section summarizes the significant contributions of the thesis.

1.3.1 Network-On-Chip based Memory BIST

The problem of optimized test power and test time at reduced area overhead during test of embedded memories in SoCs have been tackled separately by previous approaches found in literature. However, there are no system-level solutions for the same in case of memories interconnected using NoC. In this respect, this thesis focuses on providing a system level solution for test of NoC based memory cores utilizing NoC as TAM and targeting optimization of test power, test time and reduced DFT area overhead.

a) **Distributed and hybrid test architecture :**
A distributed MBIST architecture has been proposed for testing heterogeneous memory

cores interconnected using NoC. In the proposed architecture, the memory cores form different groups based on distance and timing constraints. Each group has a dedicated BIST controller which performs parallel March test on all the cores in a group. The groups are tested in a pipeline fashion. The NoC is re-used to act as TAM for delivering test instructions to the BIST controllers. The hybrid test technique and the distributed BIST architecture allows the test of memory cores to be performed at much lesser time than required in [59]. The proposed architecture is an improvement on similar architectures found in literature as it allows test of memory cores of any size while others had allowed only test of homogeneous memory cores. Utilizing a distributed BIST architecture leads to less area overhead than dedicated BIST for each core.

b) **Test time reduction :**
The reuse of the available on-chip network to act as TAM brings down the area overhead. However, reducing the time to test still remains a problem due to latency in transporting the test instruction from BIST circuit to the memory cores. A Particle Swarm Optimization (PSO) based technique has been used to place the BIST controllers at fixed locations and to form clusters of memories sharing the BIST controllers. This reduces the test instruction transport latency which in turn reduces the total test time of memory cores.

c) **Power aware test schedule :**
Based on the proposed NoC based MBIST architecture, a test schedule has been proposed which involves a grouping technique whose aim is to group memory cores which are at same distance from a BIST controller. Then, groupings are improved to make the test schedule satisfy the power constraint. Experiments performed on *ITC'02* benchmark circuit confirm that our proposed test schedule performs better power constrained test as compared to dedicated BIST technique.

1.3.2 Re-using refresh circuit for test of NoC based eDRAMs

- **Refresh re-use based off-line test :**
A two-pronged approach has been adopted in this work. First, the refresh circuit of the DRAM is leveraged to participate in the MBIST. Next, the BIST specific circuitry is shared between neighboring memories. The refresh re-use overcomes the requirement of additional DFT hardware. Moreover, to perform a read followed by a write operation on a DRAM, interleaving write cycles in between refresh cycles brings about the required effect while saving time for extra read cycles. However, the total test time increases for a commodity DRAM if refresh based test is performed. We extend the refresh re-use

technique to test a number of eDRAMs utilizing a distributed approach of testing. The increase in test time during test of a single DRAM core is effectively utilized to schedule the test of groups of eDRAMs so that power dissipation during parallel test of a number of eDRAMs remains within the power budget.

- **Refresh re-use based on-line test :**
  A BIST architecture has been proposed which reuses the refresh circuit of DRAM in performing periodic Transparent March (TMARCH) tests on the DRAMs. A TMARCH test generation algorithm is proposed for DRAMs targeting permanent faults developed during DRAM operation. The proposed algorithm generates a more efficient word-oriented TMARCH tests compared to the conventional transparent test generation techniques by avoiding signature based prediction phase. The read followed by write operations performed during the refresh burst cycles of the DRAM are re-used for the proposed TMARCH tests. Re-using the the refresh cycles for test purpose avoids waiting for idle cycles of the processor to perform the test as required in other proposed on-line transparent test techniques. It allows periodic testing of DRAM without interruption and test finishes within a definite time. Re-using the refresh circuit overcomes requirement of additional DFT hardware. Therefore, the proposed refresh re-use based transparent test technique provides a cost-effective solution by providing facility for periodic tests of DRAM without requiring costly test such as Error Correcting Code (ECC) and additional test hardware.

1.3.3 On-line field test for permanent faults in NoC buffers

Significant amount of area of the present NoC data transport medium is occupied by First In First Out (FIFO) buffers. Accordingly, the probabilities of faults or defects occurring in buffers are significantly higher compared to the other components of the NoC. Thus, test process for the NoC infrastructure must begin with test of the FIFO buffers. In this work, an on-line transparent test technique has been proposed for detection of run-time faults developed in FIFO buffers present within the routers of the NoC infrastructure. The test performs active fault detection over the entire FIFO buffer. In this work, the FIFO buffers are tested following a Transparent March algorithm instead of conventional March test algorithms to ensure that the memory contents are not lost during test. The Transparent March test is repeated periodically to avoid accumulation of faults in the FIFO buffers. The data traffic moving in and out of the FIFO buffers during normal operation of the NoC is used as data background during test. Thus no data background needs to be loaded in the FIFO buffers prior to testing. A prototype implementation of the test circuit performing the Transparent March test on the FIFO buffers is proposed. The test circuit is integrated into the router-channel interface and the on-line test is performed with different data

traffic of different applications. The performance of the NoC after addition of the test circuit is investigated in terms of throughput and latency using a *System C* based simulator.

1.4 Organization of the thesis

The rest of the thesis is organized as follows.

Chapter 2 provides the reader with a background on memory testing as well as NoC based systems. The chapter has been divided into two parts. The first part discusses the architecture and working principles of the memory cores commonly used in NoC based systems. It is followed by discussion on NoC based system, detailing architecture of each component . The second part of the chapter presents a background on different faults in memories, fault models and the different memory test techniques employed by different researchers.

Chapter 3 provides a detailed review of past literature, while identifying research gaps and scope for further work. The chapter presents a survey of the work already done in the field of Memory BIST optimization, off-line and on-line test techniques that have targeted test of DRAMs and test of FIFO buffers.

Chapter 4 presents the proposed distributed MBIST technique employed for testing heterogeneous memory cores interconnected using NoC. It provides details on the proposed architecture employed for testing and also discusses the efficient test schdule developed for the proposed test architecture that aims at reduction of test time and test power.

Chapter 5 discusses the proposed Built-In-Self test technique that utilizes refresh circuit to perform functional tests on DRAMs. The refresh re-use technique overcomes the requirement of additional DFT hardware and avoids separate test read cycles.

Chapter 6 discusses a proposed transparent test technique for testing permanent faults developed during field operation of DRAMs. The proposed transparent test is structured in a way that facilitates its implementation during refresh cycles of the DRAM. Moreover, the on-chip refresh circuit is modified to allow its re-use during implementation of the proposed transparent test.

Chapter 7 presents the proposed on-line transparent test technique for detection of latent hard faults which develop in FIFO buffers of routers during field operation of NoC. The test is repeated periodically to prevent accumulation of faults. Analytic results are used to explore the fault coverage of the proposed test technique.

Chapter 8 concludes the thesis by summarizing the contributions and indicating a few issues for future work that have been opened up by the studies in this thesis.

Chapter 2

Background

The research work presented in this thesis is about the test challenges for Network-on-Chip (NoC) based memory cores and the improved proposals to overcome them. Since the thesis involves Design-for-Testability (DFT) proposals for memory cores interconnected using NoC, it is imperative to provide the reader a basic background on design and working of each component of NoC based memory system (including memory cores) to have an understanding of the DFT architecture. Thus, we have divided this chapter into two parts. In the first part, we describe in details the internal structure and operation of memory cores and NoC based systems while in the second part, we discuss about the fault model for memories, test algorithms and memory Built-In-Self Test (BIST) architecture. The memory cores considered are SRAM, DRAM and FIFO buffers. We end the chapter with discussion on test methods for NoC based systems, with a brief on the concept of re-using the NoC to act as Test Access Mechanism (TAM).

2.1 Part I : Architecture and Working Principle

2.1.1 SRAM operation

The basic architecture of a static RAM is shown in Figure 2.1. A location of a SRAM can be randomly accessed for read/write by inputing the address of the corresponding location. Each address is linked to a particular data input/output pin. The architecture of the SRAM includes a rectangular array of memory cells arranged in rows (*word-lines*) and columns (*bit-lines*) and additional circuits for decoding addresses and implementing read and write operations.

A memory cell is a bistable flip-flop made up of four to six transistors. The flip-flop may be in either of two states that can be interpreted by the support circuitry to be a 1 or a 0. Each memory cell has a unique location or address defined by the intersection of a row and column.

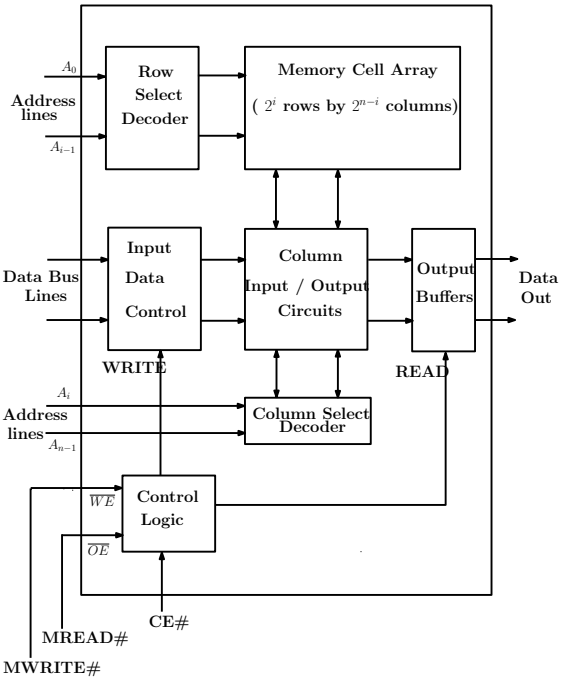


Figure 2.1: Basic Architecture of SRAM

The address of a memory location comprises of two parts. The first part is the row address (A_0 to A_{i-1} lines shown in Figure 2.1), which selects one row/word-line. The second part,(A_i to A_{n-1} lines shown in Figure 2.1) is the column address which selects one bit out of all the bits of the word activated during the word-line selection. Since row and column addresses are not required at the same time, they can be multiplexed on the same address lines. The read/write operations are controlled by the *Control* logic as shown in Figure 2.1. The chip select enable (CE#) is an additional input signal required to activate the chip. Two more control input pins exist on the chip, the WE#(write enable) and the OE#(output enable).

Read and Write operation of SRAM

During a write operation, the WE# pin triggers the chip to store in an internal location the data bits present at the input data pins. This data is translated into appropriate signal and stored in the appropriate memory cell. During a read operation, the OE# pin triggers the chip to place the data bits from the internally accessed location on its output data pins. The performance of a SRAM is measured using two parameters, the *access time* and *cycle time*. *Access time* is defined as the minimum amount of time required to read a bit from memory, measured with respect to the initial rising clock edge in the SRAM read operation. *Cycle time* is the amount of time required to perform a single read or write operation and reset the internal circuitry so that

another operation can begin.

2.1.2 DRAM operation

The basic architecture of a DRAM core is shown in Figure 2.2. Similar to SRAM, the typical DRAM chip also consist of array of memory cells, arranged in *word-lines* and *bit-lines* and support circuit for address decoding and control for read/write operation. However, DRAM chips have two additional circuits, *sense amplifiers* and *refresh counters* which are not present in SRAM. The *sense amplifiers* are used to detect and amplify the charge in the capacitor of the memory cell. *Refresh Counters* keep track of refresh sequence.

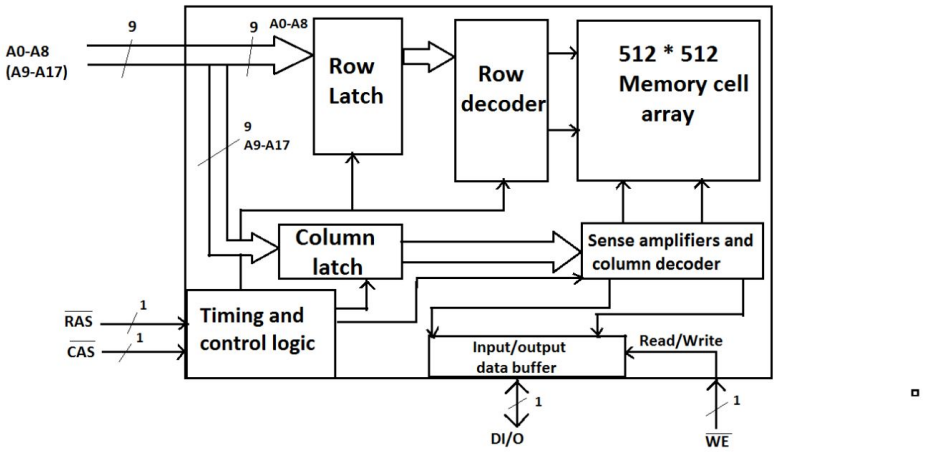


Figure 2.2: Basic Architecture of DRAM

Accessing data in DRAM is different from accessing data in SRAM. It is observed that DRAM chips have input address pins equal to half the size of the number of addresses submitted to them. In case of DRAM chip shown in Figure 2.2, the memory cell array is accessed by multiplexing the 9 input address lines: first, external memory interface logic (or "DRAM controller" discussed later) gates the row address (address bits A0 to A8) on the DRAM's address input pins and asserts the $\overline{RAS}$ signal. The $\overline{RAS}$ signal forces the DRAM chip to latch the row address, decode it and and select one of the 512 rows. Then, the memory interface logic gates the column address (address bits A9 to A17) on the same address input pins of the DRAM and asserts the $\overline{CAS}$ signal. The $\overline{CAS}$ signal will force the DRAM chip to latch the column address, decode it, and select one of the 512 columns. The memory cell at the intersection of these two selections is the accessed bit location. The write enable ($\overline{WE}$) signal allows or disallows a write operation to be performed on a memory cell array. In other words it helps to choose a write or a read operation.

DRAM memory cell

The memory cell consists of a select transistor and a storage capacitor as shown in Figure 2.3.

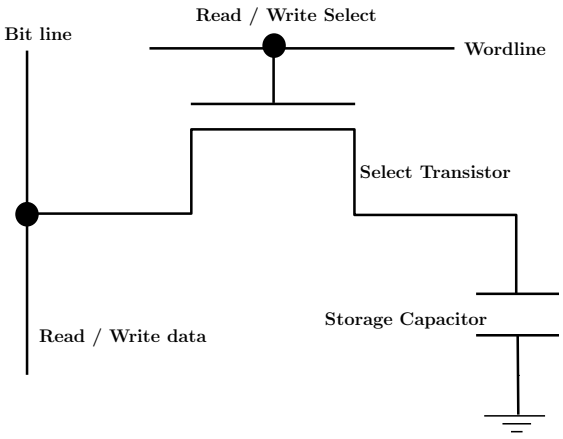


Figure 2.3: Architecture of DRAM Memory Cell

The DRAM cells are arranged in a rectangular structure as shown in Figure 2.4. The word-lines control the gate of the transistors while bit lines collect data from a large number of DRAM cells arranged in a column. Each column of cells is composed of two bit-lines, each connected to every other storage cell in the column. Figure 2.4 shows a DRAM layout where two DRAM arrays are located next to each other. Bit-lines in the same position of the DRAM arrays are paired and gated into a sense amplifier.

DRAM read

The read operation of a DRAM chip involves row access followed by column access, followed by write back and precharge [66]. The read cycle of a DRAM chip is illustrated using the timing diagram shown in Figure 2.5 and the steps involved in the read cycles are listed as follows.

a) Initially, both $\overline{RAS}$ and $\overline{CAS}$ are high. All bit-lines are precharged to Vcc/2 as shown in Figure 2.4. All word lines are at GND level, ensuring transistors are off.

b) The row address is applied to the address pins and $\overline{RAS}$ is asserted low. The row address is latched and decoded to activate a word-line. The bit lines are disconnected from the Vcc/2 bias and allowed to float.

c) Once the word-line gets activated, the memory cell of the word-line gets connected to the bit line pairs and transfers its stored charge to the bit-line. This either lowers or raises the voltage in the bit line.

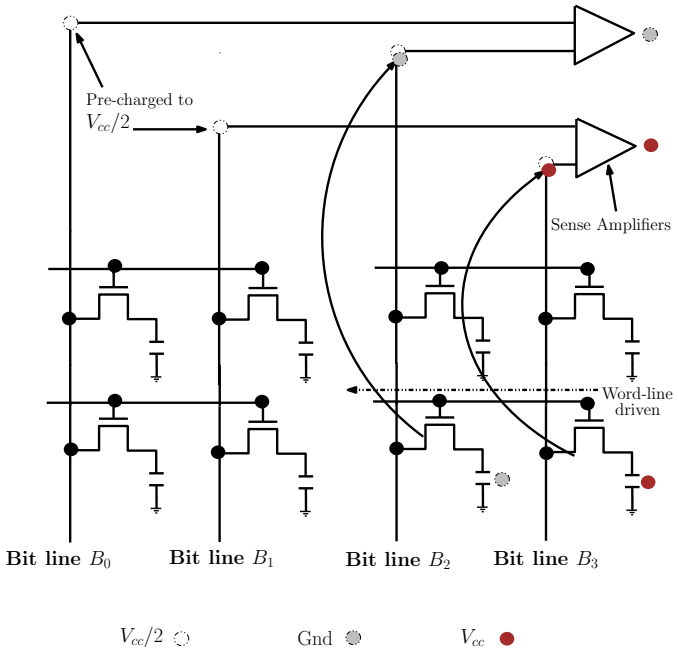


Figure 2.4: DRAM array

d) The sense amplifier then detects the differential voltage between the two bit lines (such as B0 and B2 of Figure 2.4) and amplifies this voltage.

e) During this time, the column address gets latched into the column address buffer. When $\overline{CAS}$ falls, the column address is decoded and one of the sense amplifiers is connected to the data out buffer and the data appears at the output pins after a prescribed amount of time.

f) When $\overline{RAS}$ is de-asserted, the word line goes to low. As a consequence, all DRAM cells in the row are now disconnected from the bit line.

DRAM characterization

DRAMs are characterized by different access times.

a) *Random Cycle Time* (t_{RC}) : Minimum time between any two successive reads.

b) *Access Time from* $\overline{RAS}$ (t_{RAC}) : The time elapsed from asserting the signal $\overline{RAS}$ until valid data is present on the output data pins.

c) *Access Time from* $\overline{CAS}$ (t_{CAC}) : The time elapsed from asserting the signal $\overline{CAS}$ until valid data is present on the output data pins.

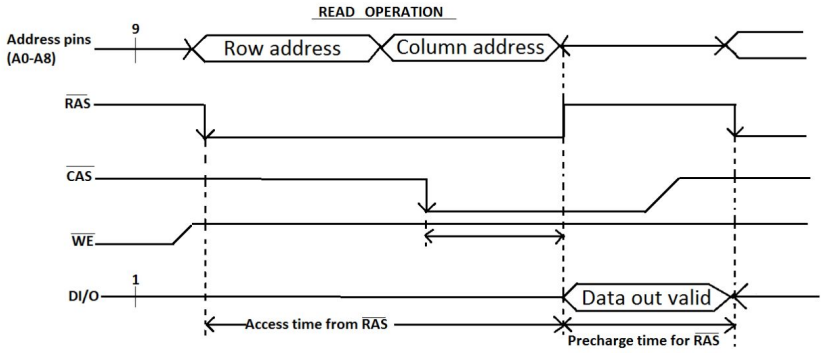


Figure 2.5: DRAM read operation timing diagram

d) *Row setup time* (t_{rs}) : The time elapsed from the moment the row address is gated to the input pins until $\overline{RAS}$ is asserted.

e) *Column setup time* (t_{cs}) : The time elapsed from the moment the column address is gated to the input pins until $\overline{CAS}$ is asserted.

From the timing diagram shown in Figure 2.5, it can be observed that *cycle time is almost twice the access time*. This additional time that must be lapsed before $\overline{RAS}$ can be asserted again is the *precharge time* (t_{PR}).

DRAM write

A DRAM write operation is similar to DRAM read, except that a write driver circuitry is used for placing the data in the cell.

a) Initially, $\overline{RAS}$ and $\overline{CAS}$ are high, all bit lines are precharged, the row address is applied to the row address decoder and $\overline{RAS}$ goes low.

b) Once the address gets decoded, a single row line (corresponding to the address) goes high. This connects all the cells in this row to the bit lines.

c) The bit lines are pulled up or down by the sense amplifiers according to the contents of the cell.

d) $\overline{WRITE}$ is de-asserted and the data is applied.

e) The column address is applied to the column address decoder and $\overline{CAS}$ is asserted low. The write driver overdrives the sense amplifier selected by the column address decoder.

f) $\overline{RAS}$ and $\overline{CAS}$ go high again. The row line goes low and all cells are now disconnected from the bit lines.

DRAM refresh

Information in DRAM is stored as electrical charge on a capacitor of a transistor cell, which provides temporary storage of data because with passage of time, charges in DRAM memory cells leak away. Therefore, to prevent data loss in DRAM, the memory cells must be periodically refreshed using special external circuits. The external circuitry periodically reads each cell and rewrites it, restoring the charge on the capacitor to its original level. When the word line is de-asserted, all cells in the row have their contents restored at full charge / discharge level. Thus, a refresh operation refreshes all cells in the same row at the same time. During memory refresh cycles, address originates from external address refresh logic and not from the normal address source and as a result, the memory is not available for normal read or write during refresh cycles.

Types of DRAM refresh

DRAM refresh can be categorized based on the following parameters [66].

- *Refresh Cycle*:The time to refresh one row of a DRAM. Since refresh is performed one row at a time, the number of refresh cycles equals the number of rows in the DRAM.
- *Refresh Time*: Interval of time within which all rows must be refreshed. It is determined by the silicon technology and design of the memory cell. If the refresh time divided by the number of refresh cycle is 15.6μsec, it is called a *Standard Refresh* device. If the result is 125μsec, it is called *Extended Refresh* device.

Some of the standard DRAMs and their specifications have been listed in Table 2.1.

Refresh cycle distribution

Refresh in a DRAM is achieved either by a *burst* method or by a *distributed* method [2]. Figure 2.6 (a) represents the distributed refresh while Figure 2.6 (b) illustrates the burst refresh. In *distributed refresh*, the refresh cycles are evenly spaced, and a refresh cycle is executed every 15.6μsec such that all rows are turned on before repeating the task. In *burst refresh*, refresh cycles are performed consecutively until all rows have been refreshed. During refresh, other memory operations are disallowed.

Table 2.1: Standard DRAMs and Refresh Specifications [2]

DRAM	Refresh Time	Number of Cycles	Refresh Rate
4Mx1	16ms	1,024	15.6μ sec
256Kx16	8ms	512	15.6μ sec
256Kx16	64ms	512	15.6μ sec
4Mx4	32ms	2,046	15.6μ sec
4Mx4	64ms	4,096	15.6μ sec

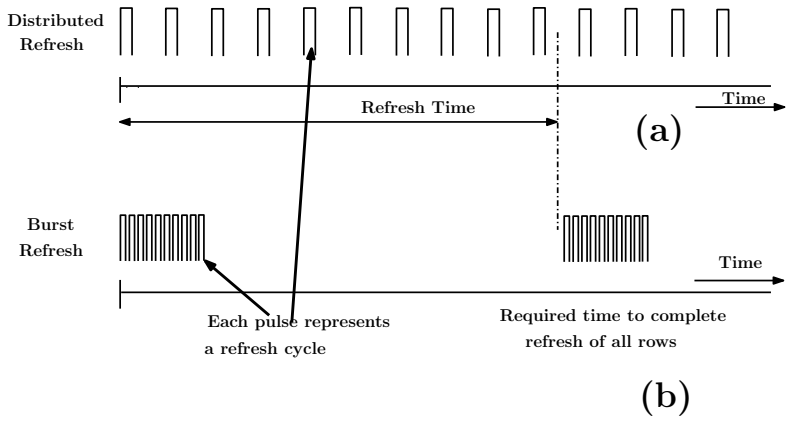


Figure 2.6: Distributed (a) and Burst (b) refresh cycles of DRAM

Refresh cycle operation

There are four kinds of refresh cycle operations used by DRAM vendors as mentioned in [66].

a) *RAS Only Refresh (ROR)* : Address of the row along with $\overline{RAS}$ is applied to the DRAM. Throughout the cycle, $\overline{CAS}$ is held high. The refresh addresses are generated by the DRAM controller.

b) *CAS before RAS Refresh (CBR)* : First the $\overline{CAS}$ is lowered, followed by $\overline{RAS}$ signal. The $\overline{WE}$ signal must be held high while $\overline{RAS}$ is lowered. Each transition from $\overline{RAS}$ from high to low does one refresh. The refresh addresses are generated by an internal refresh counter.

c) *Hidden Refresh*: Read/Write operation is followed by a refresh operation. After a read/write cycle, $\overline{CAS}$ is asserted low, while $\overline{RAS}$ is toggled to high, and then brought back to low. Since $\overline{CAS}$ was low, lowering $\overline{RAS}$ brings about CBR refresh. The refresh is hidden in a sense that data remains valid for a longer time on the output.

d) *Self Refresh (SR)*: This refresh involves a CBR cyle with $\overline{RAS}$ active for a minimum of 100μsec. The refresh is performed using a counter and an oscillator to generate the row address and to monitor the refresh interval.

DRAM refresh circuitry

Figure 2.7 shows the functional block diagram of a DRAM with the refresh circuit. The refresh control and refresh counter shown in Figure 2.7 are present within the Memory Controller chip and are responsible for initiating the refresh operations of a DRAM.

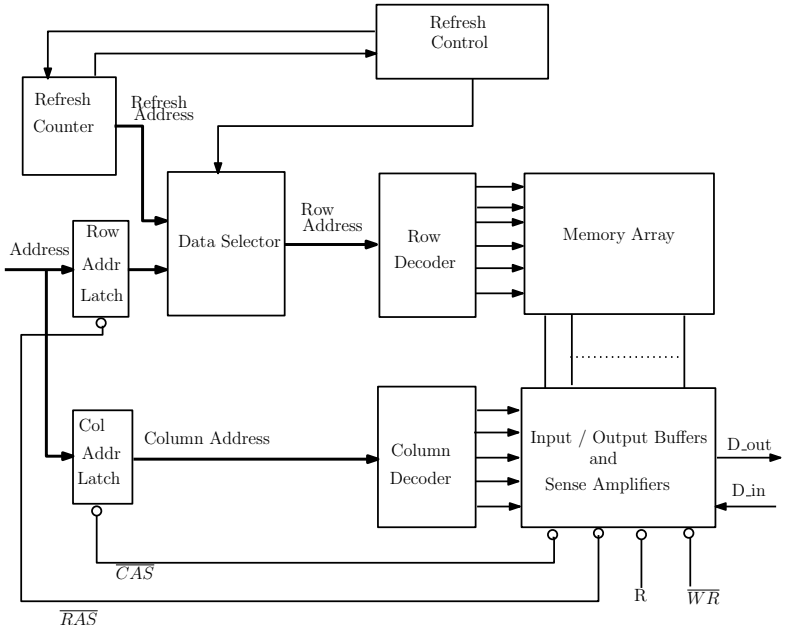


Figure 2.7: DRAM with refresh circuitry

In case of a Standard Refresh Device, after every 15.6μs, the internal timer of the Refresh Control generates a Refresh Request for the DRAM signifying the start of a Refresh Cycle and instructs the Refresh Counter to generate a Refresh Address. This Refresh Address generated by the Refresh Counter is Multiplexed by the Data Selector during the refresh operation to the Row Decoder. With the $\overline{RAS}$ signal asserted low, the contents of the address generated by the Refresh Counter are read by the Sense Amplifiers and then out into a Refreshment Register through the D_{out} pin. The contents of the Refreshment Register are again brought in through the D_{in} pin to be written into the same location which has still been addressed by the address generated by the Refresh Counter

2.1.3 FIFO buffer

Systems with multiple interacting cores often require buffers for storing data if the cores operate at different data rates. These buffers are FIFO memories which are either shift type FIFOs or RAM type FIFOs [93]. The shifting type FIFOs use shift register to shift data from the write port into the last unused location of the read port.

RAM type FIFOs as the name implies uses a SRAM and write and read address register to access data. According to the classification mentioned in [93], the RAM type FIFOs can be recognized as either *ring address* RAM type FIFO, *dual-port* RAM type FIFO and *arbitration* RAM type FIFO. In *ring address* RAM type FIFO, an *n*-bit shift register specifies the read and write addresses. The *dual-port* RAM type FIFO has a *N* bit counter specifying the read and write addresses, which accesses memory via separate ports. In *arbitration* RAM type FIFOs read and write addresses are specified with an *N*-bit counter accessing a single-port memory using an arbiter.

A functional model of an *arbitration* RAM type FIFO is illustrated in Figure 2.8. It consist of SRAM memory of *n B*-bit words. The address is of *N-bit* and is provided by either the WAR (Write Address Register) or the RAR (Read Address Register). The DI (data input) line receives the input data while the output data can be taken from the DO lines. The control block generates control signals meant for providing different functions, such as serialization of a simultaneous Read Request (RR) and Write Request (WR). The read and write acknowledge are provided through the RA and WA lines respectively. The Full Flag(FF) and the Empty Flag (EF) indicate the full and empty status of the FIFO. The Reset (RS) control input initializes the WAR and RAR such that they specify the same, initial address IA. The Re-Transmit (RT) control input resets the RAR to its initial value.

2.1.4 NoC based system

With advancement of semiconductor technology, the computation cost in today's chip design is reducing day by day. However, communication is getting more and more expensive, thus shifting the focus to more communication-centric designs. To this effect, researchers have been trying to come up with efficient schemes such as the on-chip interconnects in order to reduce the cost of designs. Conventional bus-based on-chip interconnects are severely limited in performance (latency and bandwidth). Advanced bus-based solutions overcome the problem of bandwidth, but fail to provide the scalability for cost effective communication scheme [70]. Thus, the design specific global on-chip interconnection in SoC designs have made way for a more general purpose on-chip interconnection network, referred to as NoC. High parallelism in NoCs leads to smaller latency in data transmission and as a result the the performance loss is avoided.

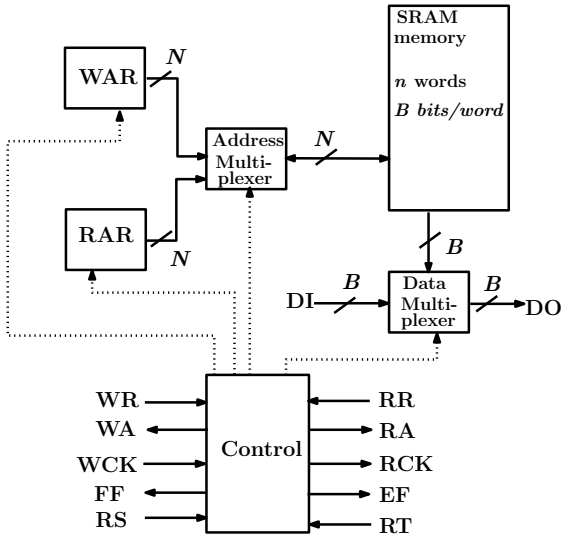


Figure 2.8: Functional Model of SRAM type FIFO [93]

Moreover, by providing standard interfaces which are compatible with existing protocols, NoC facilitates reusing of IP blocks and increases modularity in SoCs [70]. The advantages provided by NoCs have allowed the SoC designers to consider them for high density SoCs.

NoC architecture

The components of the NoC communication infrastructure are routers, Network Interfaces (NI) and links. Routers are responsible for routing data units of transmission from source to destination. Links connect the routers together and also the routers to the NIs. Network interfaces are used to interface a core to the interconnection network. Figure 2.9 shows the general architecture of NoC.

Network topology

The network topology refers to the connection technique of the routers in the NoC. A number of network topologies, with different performance behavior have been proposed by different researchers. Some topologies are regular, such as *mesh*, *torus* and *octagon* and some are irregular application specific [56]. Irregular topologies are more efficient in terms of resource usage while regular topologies are more advantageous when it requires control of electrical parameters. Some regular and irregular topologies have been shown in Figure 2.10.

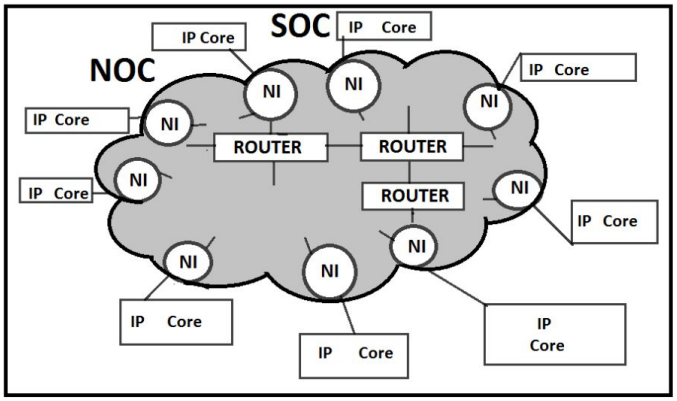


Figure 2.9: General Architecture of NoC

Communication in NoC

Cores in NoC communicate among themselves by sending and receiving messages which are sent in the form of *packets*. A packet contains information necessary for routing it from source to destination. Packets are divided into smaller data transfer units called *flits*. Flits are the basic unit of storage allocation and bandwidth. They do not have any routing information and have to follow the route for the whole packet. Flits are composed of physical words called *phits* which actually construct the packets. They are units which are transferred in a single clock cycle.

Routers

The router consist of a Routing Logic Block (RLB) and input/output ports. RLB connects the input ports to proper output ports.

Switching techniques

Switching techniques determine how the internal switches of the network are set to connect router inputs to outputs for transferring messages. Switching techniques are of two types, namely, *circuit* switching and *packet* switching. In *circuit* switching, links are reserved for a special connection from source to destination. However, this technique produces excessive blocking leading to communication latency affecting bandwidth. In *packet* switching, each message is partitioned into fixed length packets and the packets are transmitted without reserving the

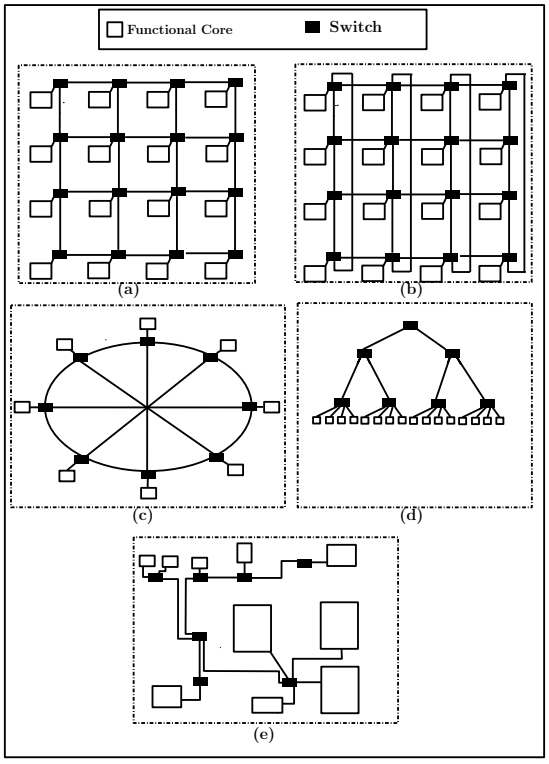


Figure 2.10: NoC regular and irregular topologies [8] : Mesh-based(a), Torus(b), Octagon(c), Binary tree(d) and Irregular application specific (e)

entire path. Since links are not reserved and the arrival time of packets are not known, contention exists for access of links and buffers. The contention problem is overcome by arbitration which determines which packet is eligible for access. Packets of connections which loose out in arbitration need to be stored and thus a buffers scheme is required for storage. Buffers can be placed at input or output ports. Packet switched network is further classified as *store-and-forward(SAF)*, *virtual-cut-through(VCT)*, *wormhole* and *virtual channel*.

- SAF switching : A packet if completely buffered at each intermediate node before it is forwarded to the next node. Therefore, it needs huge silicon area [56].
- VCT : A packet is forwarded to the next router as soon as there is enough space to store the packet. VCT overcomes the latency penalty of SAF, but also requires high silicon area to store the entire packet [79].
- Wormhole : Packets are divided into flits. The header flit contains information about source and destination addresses. Payload flit contains data, while tailer flit contains end of packet information. Header flit decoding enables the switches to establish the path

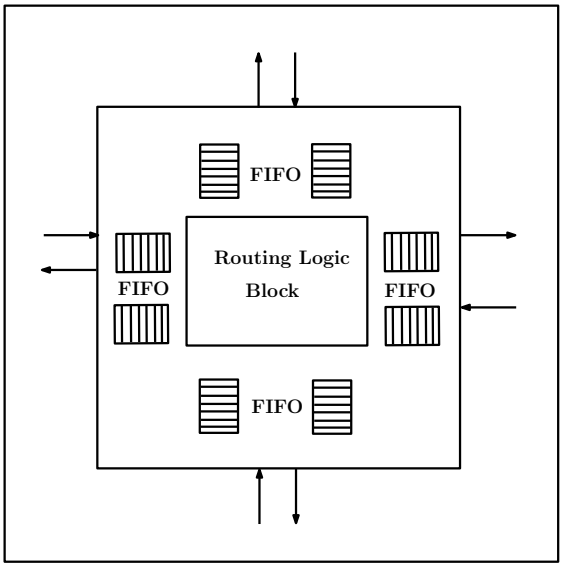


Figure 2.11: Router Architecture [8]

while payload and tailer flits simply follow this path in a pipelined way. If a certain flit faces busy channel, subsequent flits also wait at their respective channels.

- Virtual channel (VC) : To mitigate the effect of *Head-of-line* blocking, each physical channel uses several virtual channels. When a particular packet is blocked, VC allows other packets to use the link that would otherwise be left idle.

Flow control

Flow control protocol determines how packets travel through the network from source to destination. It is either end-to-end level or switch-to-switch level. In end-to-end level, the sender NI controls the amount of space available in the receiving NI. If there is enough space in the receiver NI to receive one unit of data, the sender NI will send the data. In link-level flow control, each router should be assured that there is enough space in the next router of the path and then the data will be sent to the next router.

Buffering

In NoCs, flits travel through multiple First-In-First-Out (FIFO) buffers from source to destination. As a result, if the latency of the FIFO is high, performance of the overall network is degraded. To design a low latency FIFO, independent read and write clocks are provided to it. The *FULL* and *EMPTY* signals are dependent on both the clocks.

Network Interface

The NI module interfaces the core to the network. It decouples computation from communication and performs protocol conversion between the core and router to which it is connected. According to the work reported by [80], the NI architecture can be divided into three parts, *Generic Core Interface (GCI)*, *Packet Maker (PM)* and *Packet Disassembler (PD)*. The function of each part is described as follows.

- *Generic Core Interface (GCI)* : It abstracts the network communication protocol from the core specific wrapper for heterogeneous system implementation.
- *Packet Maker (PM)* : The core specific wrapper transmits the message to PM memory. PM performs the following tasks.
  - Packetizes the message stored in PM memory and breaks them into several flits before queuing them into asynchronous FIFO having independent read and write clocks.
  - In case of source routing, maintains the routing information in look-up-table.
  - Insert redundant bits (parity) in the packet tailer for supporting end-to-end flow control.
- *Packet Disassembler (PD)* : PD performs the following tasks.
  - Writes incoming flits from asynchronous FIFO to PD memory.
  - Decodes packet header from PD memory, extracts control information required by the core, and passes it to GCI.
  - The core wrapper reads the payload and the tailer to obtain the total message. It also performs error detection and end-to-end flow control.
  - Ensures in-order delivery of packets which is extremely important for adaptive routing.

As routers and IP cores are operating at completely independent clocks, the asynchronous FIFO in NI performs the synchronization task in clock domain crossing.

Routing Strategies

Routing algorithms are classified based on the following decisions.

- Number of destinations of a single packet:

  - *unicast*: each packet has a single destination.
  - *multicast*: single packet has multiple destinations.

- Position of routing decision:
  - *source routing* : pre-computed routing table in NI
  - *distributed routing* : each packet carries source and destination address. The routing decision is implemented in each router by a routing table or by executing a finite state machine.

- *Deterministic* : packets always follow specific path from source to destination. This assures in-order delivery of packets.

- *Oblivious* : Selects path randomly or cyclically.

- *Adaptive* : Routing decisions are taken based on the current state of the network (congestion, available link, etc.) and alternative paths are chosen dynamically.

The challenges in routing scheme are as follows.

- *Livelock* : It arises when packets travel around its destination node, but unable to reach it because the channel to do so are occupied by other packets. It occurs only in adaptive routing when packets are allowed to follow minimal path.

- *Deadlock* : It arises when a set of messages are blocked forever because each message in the set holds one or more resources needed by another message in the set. Dimension order routing is the most simplest approach to avoid deadlock in deterministic routing. In two dimensional (2-D) mesh type network, it is called *X-Y* routing, where a packet is first forwarded in the X-direction until it reaches the X coordinate of the destination and then forwarded in the Y-direction until it reaches the destination. Like X-Y routing, Y-X routing is also deadlock free. For ring and torus type networks, deadlock is avoided by splitting each physical channel into group of virtual channels.

NoC testing

Testing NoC based systems involve three steps. First, it must be ensured that the *NoC infrastructure* is fault free so that in the next step while testing the *IP Cores*, the NoC infrastructure can be utilized as Test Access Mechanism (TAM). Finally, in the last step, the *whole integrated circuit* is tested which checks for correctness of the interaction between IP Cores and the interconnect.

a) *Testing NoC infrastructure :* Testing the NoC infrastructure involves test of links, routers and NIs. Links have to be tested first as they are responsible for transferring data between switches. Moreover in some design methodologies the NoC infrastructure is reused as TAM for testing routers. In such cases, links are the basic elements for transferring test data, therefore their correct functioning should be guaranteed prior to using them as TAM. Tests are performed for detection of of crosstalk in wires and for detection of timing relations in interconnection links when the NoC is used in *Globally Synchronous Locally Asynchronous (GALS)* systems [84]. Router testing is done in two parts; the combinational logic part used for data routing (RLB) and the buffers. The RLB is tested using standard test methods for combinational circuits such as scan and BIST [65]. However, testing buffers is difficult as they are small and distributed all over the chip. Thus employing traditional BIST approaches leads to increased area overhead.

b) *Testing IP cores :* One of the major design issue for test of deeply embedded memory cores in SoCs is accessibility of their terminal through input/output ports of the SoC. For solving this problem, designers use a special hardware for transferring test data from I/O pins of chip to core terminals. This hardware is called Test Access Mechanism (TAM). In case of NoC based SoCs, researchers have proposed the re-use of the NoC infrastructure to act as TAM [23]. The re-use mechanism allows concurrent transferring of test data to different cores leading to reduction of test time. Moreover, the technique is cost-effective as no extra hardware is required.

c) *Testing the whole integrated circuit :* After the NoC infrastructure and the Intellectual Property (IP) cores have been tested, the interaction between computing cores must also be functionally tested. This includes test of the I/O functionality of each IP core and of the routers.

2.2 Part II : Test Methods

More than 90% of the chip area in today's SoCs are dominated by memory cores. The advancement of memory design technology and structural simplicity of memories have resulted in high density of memories. However, high packaging density of memories causes more susceptible to physical defects. Since memories have high impact on the overall chip area, efficient and high quality tests are required to detect the physical defects in memories. The most popular and widely used tests have been discussed in this part of the chapter. Moreover, we also briefly cover the memory test architectures used to perform the test algorithms. The details have been covered in the subsequent subsections.

2.2.1 Faults in memories

A *fault* refers to an underlying cause of failure such as radiation induced bit flip or stuck-at bit [83]. The presence of a *Memory Fault* can be detected when there is physical difference between the correct and incorrect behavior of a memory. The faults which cause memory failures can be classified as *Permanent* faults, *Transient* faults and *Intermittent* faults.

Permanent (Hard Faults) corrupt bits in a persistent manner due to a physical defect such as stuck at faults. Permanent faults are generated during manufacturing process and are detected by post-manufacturing tests [reference]. Hard faults can only be repaired by replacing or bypassing the faulty device. The mechanisms which cause hard faults are bad electrical connection, broken components (IC mask defect), bur-out chip wire, corroded connection between chip and package and logic error.

Transient faults occur randomly and without significant physical damage. They cause incorrect data to be read from a memory location. The following are possible factors that contribute to transient faults: Cosmic rays, alpha particles, electromagnetic interference, voltage fluctuations, humidity, pressure. Errors in memories introduced by transient faults are often called *soft errors*. The device that experiences transient fault can be repaired by overwriting the location with correct data.

Intermittent Faults, which cause memory locations to sometimes return incorrect values. These faults are caused by non-environmental conditions such as loose connection, aging components, noise in the system, critical timing. Similar to transient faults, intermittent faults are non-permanent faults. However, unlike transient faults, intermittent faults indicate device damage or malfunction.

2.2.2 Testing Methods

The real physical defects in memories are often too numerous to be analyzable. As a result, there is requirement of a model which would reduce the set of target faults to a sizable number and would make their analysis possible. A *fault model* can be defined as an abstract representation of defects. The representation can be at different levels of abstraction, such as behavioral, functional, logic level or electrical level. Depending on the level of abstraction of the fault model which is being used by the testing method, we can classify the testing method to structural and functional testing [70].

Functional testing is a comprehensive testing which tests if the Device-Under-Test (DUT) can behave as it is expected to behave. It can be done in two different ways: (a) Applying functional test vectors and (b) Using hardware redundancy. Structural testing is testing the structure of the device-under-test by using lower levels of fault models. For example, structural testing of a 4-input AND gate tests all the input lines and the output line for stuck-at faults. Therefore the total number of test patterns that should be applied to the gate is 10 test patterns.

2.2.3 Functional Fault Models

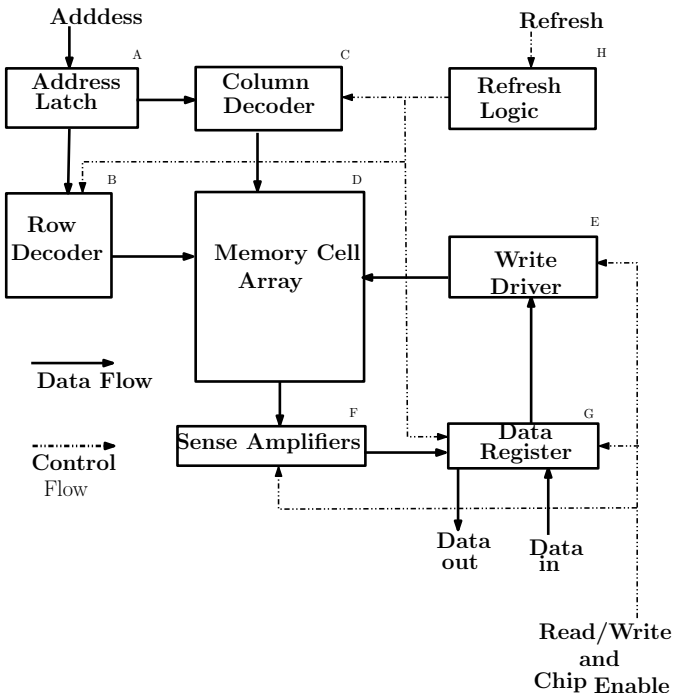


Figure 2.12: Functional memory model [13]

Functional fault modeling has been the primary level for memory test. Functional modeling

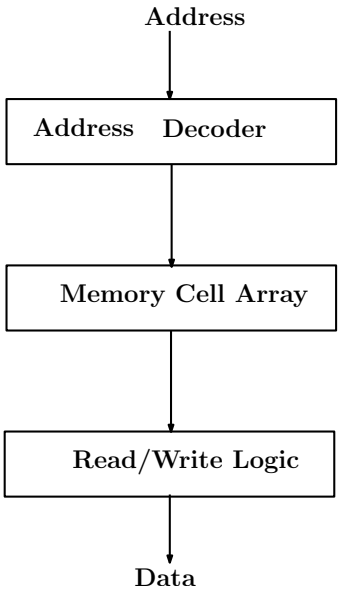


Figure 2.13: Simplified functional memory model [13]

identifies the circuit as a function but visibility to the inner workings of the memory is not provided. Figure 2.12 shows a functional memory model and Table 2.2 lists some of the functional faults based on the functional model that can occur in memories.

However, the functional fault model of Figure 2.12 can be reduced to a more simplified model as shown in Figure2.13 and the functional faults in Table2.2 can be mapped to reduced functional faults of Table2.2.3.

The four fault models listed in Table 2.2.3 form the classic memory fault models. The fault models have been explained using the Markov model based state diagrams [3]. The "R" indicates a read operation while "W0" indicates write 0 and "W1" indicates write 1. "S0" and "S1" indicate the cell in a "0" or a "1" state. The state diagram for the good memory cell is shown in Figure2.14 (a). The defect free single cell can be written to either state and retains the original information when read. The definition of classic memory fault models are as follows.

- SAF : Indicates that a cell has been stuck to a particular state. Figure2.14 (b) represents the stuck-at-0 fault where the cell contains a "0" regardless of any data ("0" or "1") written to it.

- TF : Indicates memory cell retains either state. If a cell is written to a state, it cannot transition back. The cell can only be written in one direction. For example, as shown in Figure2.14 (c), if the cell is in state S0 and is written a "1", the cell moves to S1 state. However, in S1 state, a write "0" will not bring the cell back to S0, rather the cell will

Table 2.2: Subset of functional memory faults [13]

	Functional Fault		Functional Fault
a	Cell stuck	*j*	Address line open
b	Driver stuck	*k*	Address lines short
c	Read/Write line stuck	*l*	Open circuit in decoder
d	Chip-select line stuck	*m*	Wrong address access
e	Data line stuck	*n*	Multiple simultaneous address access
f	Open circuit in data line	*o*	cell can be set to 0 but not to 1 (or vice versa)
g	Short circuit between data lines	*p*	Pattern sensitive applications
h	Cross talk between data lines		
i	Address line stuck		

Table 2.3: Reduced functional memory faults

Notation	Fault
SAF	Stuck-at-fault
TF	Transition fault
CF	Coupling fault
NPSF	Neighborhood pattern sensitive fault

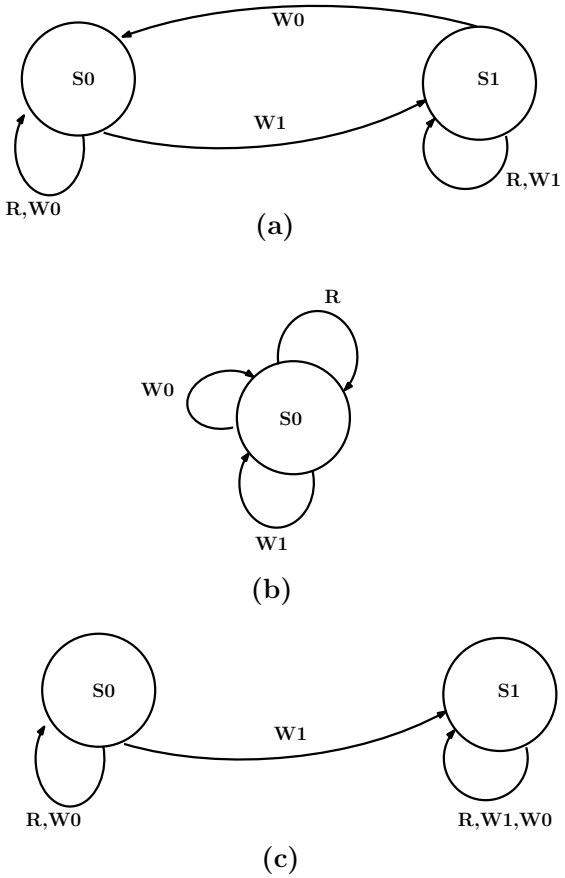
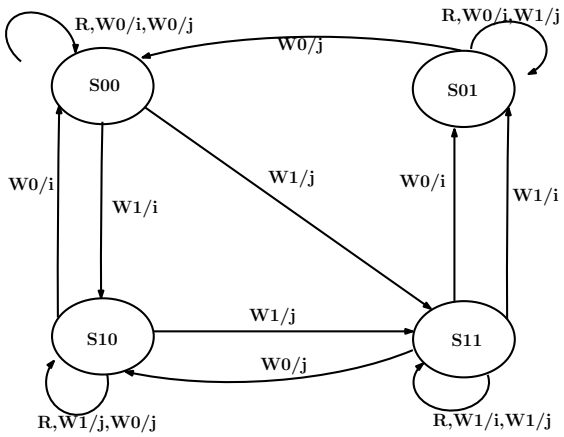


Figure 2.14: Markov model representation : good memory cell (a), stuck-at-0 cell (b), transition fault model (b) [13]

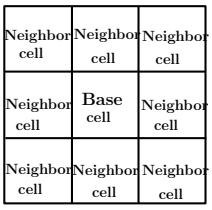
remain in S1 state.

- CF : It indicates that a transition in cell *j* causes an unwanted change in memory cell *i*. The cell which does the coupling is the *aggressor* and one which transitions is called *victim*.
- NPSF : The memory cell is dependent upon cells in its neighborhood, which may be pattern of "0"s or "1"s or a pattern of transitions. For example in a nine-cell neighborhood, the base cell (cell under test) is dependent on the surrounding eight neighboring cells.

The coupling fault and NPSF fault have been illustrated in Figure2.15 (a) and (b) respectively. In this chapter of the thesis we have restricted our discussion of memory faults to only those which are considered during the course of the research work. Since detailed discussion of the classical memory faults can be found in number of literatures, extensive coverage of them would not provide any deeper insight. A detailed discussion of coupling faults and pattern sensitive faults can be found in [13] and [61].



(a)



(b)

Figure 2.15: Markov model representation of coupling fault (a) and nine cell neighborhood representation (b)

There are faults in memories which depend on the actual circuit operation such as *Read Disturb* Fault Model (RDF), *Write Disturb* Faults (WDF), *Data Retention* Fault (DRF), *Address Decoder* Faults (AF), *SOI* Faults, *False Write Through* and *Pre-Charge* Faults. In this chapter, we briefly describe only the *RDF*, *DRF* and the *WDF* since these faults have been considered in the research problems discussed in the succeeding chapters. The details of the remaining faults can be found in [3].

- Read Destructive Fault (RDF) : RDF occurs when a read operation performed on the defective cell changes the data in the cell and returns an incorrect value on the output [3]. RDF are predominant in SRAMs, where a defective SRAM loses its data state on a read and behaves as a dynamic cell. Since DRAMs refresh data after every read operation, read disturb faults are less likely to occur in DRAMs.To detect read disturb fault, a 1 and a 0 should be read from each cell.

- Write Disturb Fault (WDF) : A WDF occurs when a non transition write operation performed on the defective cell causes a transition in the cell. To detect a write-disturb fault, each cell should be read after a non-transition write. For example, with a cell initially

holding a 0 value, if a write 0 is performed, then the cell must be read for a 0 immediately after the read. Similarly, with a cell initially holding a 1 value, if a write 1 is performed, then the cell must be read for a 1 immediately after the read.

- Data Retention Fault (DRF) : A DRF occurs when a memory cell loses its previously stored logic value after a certain period of time during which it has not been accessed [32]. This kind of fault is the consequence of resistive open defects in SRAM core-cells, in particular in the self refreshment loop circuit. A detailed discussion of analysis and detection procedures of DRFs can be found in [29]

2.2.4 Memory test algorithms

A number of RAM tests such as *Memory Scan (MSCAN)*, *Galloping Pattern (GALPAT)*, *Checker Board Pattern* have been found in literature [61]. However, these tests have poor fault coverage. As a result, researchers came up with a simple yet efficient test for memories called *March* based tests. March tests are the most widely accepted tests for detection of faults due to their high fault coverage and linear relation of their test time with respect to the memory size [13]. A March instruction consists of sequence of operations, called *March element* applied to each cell before proceeding to the next cell. An operation can be writing a 0 (w0) or reading a 1 (w1), reading an expected 0 from a cell (r0) and reading an expected 0 from a cell (r1). March element can be done in either one of two address orders: the ascending order (↑) or descending order (↓). When the address order is irrelevant, then (↕) is used. A bit oriented March test is represented as follows.

$$\{\updownarrow (w0); \uparrow (r0, w1); \downarrow (r1, w0)\}$$

. There are three March elements in the March test notation. The March elements are separated by semicolons and are denoted as M_0, M_1 and M_2. In each March element, first the address sequence is specified followed by the operations. The execution of all the three March elements is illustrated in Algorithm 1. The memory considered in Algorithm 1 is assumed to be having N locations with each location of size B bits (cells). The variables i and j in Algorithm 1 represent the pointers for address and cells respectively.

In the March element representation, the March element M_0 begins by performing a write operation (with a data 0) on each cell of the first address and then continues to the next address. This is illustrated in the lines 1-7 of Algorithm 1 representing the example March test. Thus, after the M_0 element has been performed, every cell of the entire memory is filled with zero. Next, the March element M_1 is performed where on each address (memory cell) a read operation (with an expected 0 in the fault-free case) is performed followed by a write back of the complemented

bit immediately. Then, the same operations are continued for each cell of the next address. The M_1 operation execution is illustrated in lines 8-15 of Algorithm 1. The M_2 element is a repeat of M_1 except that the read and write data are reversed. The M_2 March element execution is illustrated in lines 16-23 of Algorithm 1. The example March test algorithm is also called a March $5N$ algorithm as it requires $5N$ read/write operations, N being the size of the memory. The March test used in the Algorithm 1 is called MATS+ test [13]. A list of different March test is presented in Table2.2.4

Algorithm 1 March Test Example

```
1: for i = 0 to N do
2:     for j = 0 to B do
3:         Write(0, cell[j])
4:         j ← j + 1
5:     end for
6:     i ← i + 1
7: end for
8: for i = 0 to N do
9:     for j = 0 to B do
10:        Read(cell[j])
11:        Write(1, cell[j])
12:        j ← j + 1
13:    end for
14:    i ← i + 1
15: end for
16: for i = 0 to N do
17:    for j = 0 to B do
18:        Read(cell[j])
19:        Write(0, cell[j])
20:        j ← j + 1
21:    end for
22:    i ← i + 1
23: end for
```

Fault detection using the March test

The fault detection using March test has been excellently illustrated by Bushnell and Agarwal in [13]. We use their detection technique to explain the detection of Stuck-at-fault 1 (SA1) of at cell (2,2) in a memory of size 3x3. Figure2.16 illustrates the detection technique using a MATS+ test consisting of March elements M0, M1, and M2. The fault is detected by March element M2

Table 2.4: March Test Algorithms [13]

Algorithm	Description
MATS	$\{\updownarrow (w0); \uparrow (r0, w1); \updownarrow (r1)\}$
MATS+	$\{\updownarrow (w0); \uparrow (r0, w1); \downarrow (r1, w0)\}$
MATS++	$\{\updownarrow (w0); \uparrow (r0, w1); \downarrow (r1, w0, r0)\}$
MarchX	$\{\updownarrow (w0); \uparrow (r0, w1); \uparrow (r1, w0); \downarrow (r0, w1); \downarrow (r1, w0)\}$
MarchC-	$\{\updownarrow (w0); \uparrow (r0, w1); \uparrow (r1, w0); \downarrow (r0, w1); \downarrow (r1, w0); \updownarrow (r0)\}$
MarchA	$\{\updownarrow (w0); \uparrow (r0, w1, w0, w1); \uparrow (r1, w0, w1); \downarrow (r1, w0, w1, w0); \downarrow (r0, w1, w0)\}$
MarchY	$\{\updownarrow (w0); \uparrow (r0, w1, r1); \downarrow (r1, w0, r0); \updownarrow (w0)\}$
MarchB	$\{\updownarrow (r0); \uparrow (r0, w1, r1, w0, r0, w1); \uparrow (r0, w0, w1); \downarrow (r1, w0, w1, w0); \downarrow (r1, w1, w0)\}$

as it moves from the highest memory address downward and expects to read a 0 in cell (2,2), but instead gets a 1.

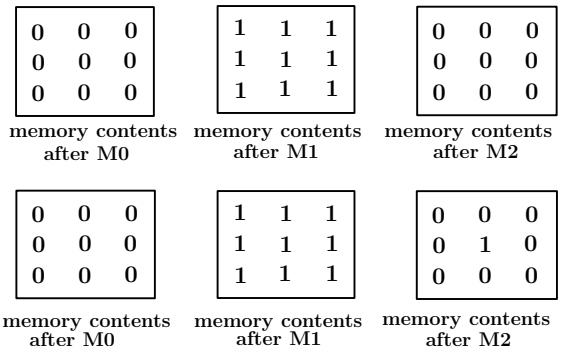


Figure 2.16: SA1 Fault Detection using MATS+ test

2.2.5 Test for word oriented memory

For a word oriented memory, the read/write operations in the March tests are performed on a data word instead of a bit. The data word read or written in case of a word oriented memory is called the *background pattern* or *data background*. For a word-oriented memory having *N* number of data words and *w* bits/word, the word-oriented MATS+ is represented as

$$\{\updownarrow (wa); \uparrow (ra, wb); \downarrow (rb, wa)\}$$

where a is a background word and b is its complement [61]. Different techniques have been proposed by researchers for converting a bit-oriented March test to word-oriented March test. The first work in this regard was proposed by Vand de goor in [89]. In his work, Van de goor proved that fault models developed for single cell in bit-oriented memories can be applied for word-oriented memories. However, for faults involving two or more cells, further classification is required according to whether they are within the same word or not (i.e., intra-word or inter-word faults) [89] .

2.2.6 Memory BIST architecture

With more and more memories embedded in circuits, accessibility becomes an issue in tester based methods making Built-in-Self Test (BIST) the solution of choice. The memory Built-in-Self Test architecture (MBIST) comprises of a BIST controller, a memory with a wrapper and an interconnect as shown in Figure 2.17. The MBIST wrapper further includes an *address generator* to provide the complete memory address sequences (for n address lines all the 2^n lines are visited in complete sequence); a *background pattern generator* to produce data patterns when testing word-oriented memories [13]; a *comparator* to check the memory output against the expected correct data pattern; and a *finite state machine* (*FSM*) to generate proper test control signals based on the commands received from upper level controller (Automatic Test Equipment). The *interconnect* between the wrapper and the controller could either be serial (a single command line shared by all wrappers) or parallel (dedicated multiple command lines linking different wrappers to the controller). The collar(or wrapper) included with the memory also consists of multiplexers that select between address, data and control signals from the BIST controller in test mode and the signals from the CPU in mission mode. A comparator is used to check if the expected data from the read operation matches the data actually read out from the memory.

The basic operation of memory BIST is straightforward: First, the memory is put into a test mode by the use of muxes placed on every data, address, and control line. A finite state machine writes a test pattern to a memory cell, reads it back, and compares it to the original value. If a mismatch occurs, a flag is set to show that the memory cell under test has a failure (the Pass/Fail line in Figure 2.17). The address is incremented and the process continues recursively. The process of stepping through the entire memory space can be done multiple times, using different patterns to more fully exercise the memory.

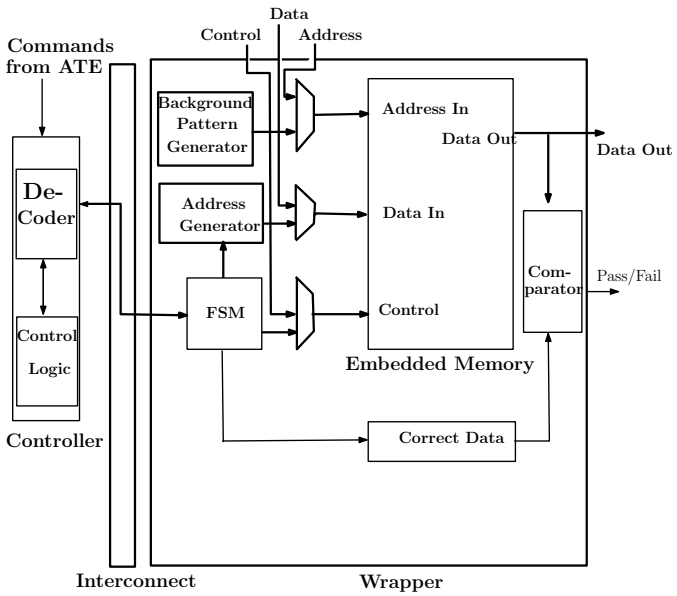


Figure 2.17: Generic Memory BIST Architecture

2.3 Summary

In this chapter, both the design and the test aspects of memory cores and NoC based systems have been presented, covering architectural descriptions, working principles and test issues. The brief discussion covered in this chapter would pave the path for clear understanding of the topics discussed in the forthcoming chapters. Following this chapter is Chapter 3, which presents a survey on the literature related to test of NoC based memory systems. The chapter touches upon the different aspects of memory testing elaborating on the test challenges and proposed solutions by exploring works of different researchers.

Chapter 3

Literature Review

Memory testing has been a research topic of interest for almost three decades. Researchers over the years have contributed to the three different aspects of memory testing, namely, *fault modeling*, *test algorithm design* and *memory BIST architectures*. Some of the earlier works in fault modeling such as by Dekker et al. in [26] and Goor et al. in [91] were limited to static fault modeling. However, with emergence of dynamic faults in memories, fault models for dynamic faults were also proposed in works by Hamidoui et al. in [37] and [39]. With introduction of fault models, since early 1980s, March tests have become the dominant type of tests for memories. A detailed discussion of different March tests have been covered in several text books such as the one by Van de goor in [90]. With advent of technology and emergence of SoCs, accessibility of embedded memory cores has become an issue during testing of SoCs. Thus, BIST has been chosen as the viable test option for embedded memories in SoCs. As a result, over the past decade, a lot of text has been dedicated to design of innovative MBIST architectures. Some of the commonly used MBIST architectures have been covered in books by Bushnell and Agarwal [13] and Wang et al. [96].

Another important issue in SoCs has been the communication infrastructure. The increasing communication demand in present day SoCs has led to research on novel interconnection networks such as NoC. Mullins has listed more than 400 articles on different aspects of NoC involving communication infrastructure, software and operating system services and Computer Aided Design (CAD) tools for NoC in the website [1]. The focus of the present thesis has been to integrate both the above mentioned issues, memory testing and NoC. To this effect, the thesis aims to highlight test challenges for memory cores interconnected using NoC and to come up with effective solutions for the same.

The present chapter presents a survey of different test techniques that have been proposed by researchers with the aim of optimizing either area overhead or test time or power dissipation

during test of memory cores interconnected using NoC and also during test of memory cores which are part of the NoC infrastructure. Figure 3.1 provides an illustrative view of the structure that has been followed during the survey. The subsections of the chapter highlight important features of different proposals found in literature and also mention the shortcomings which served as a motivation for the research problems of the presented thesis. The presentation of the survey has been arranged following a pre-order traversal of the tree structure shown in Figure 3.1.

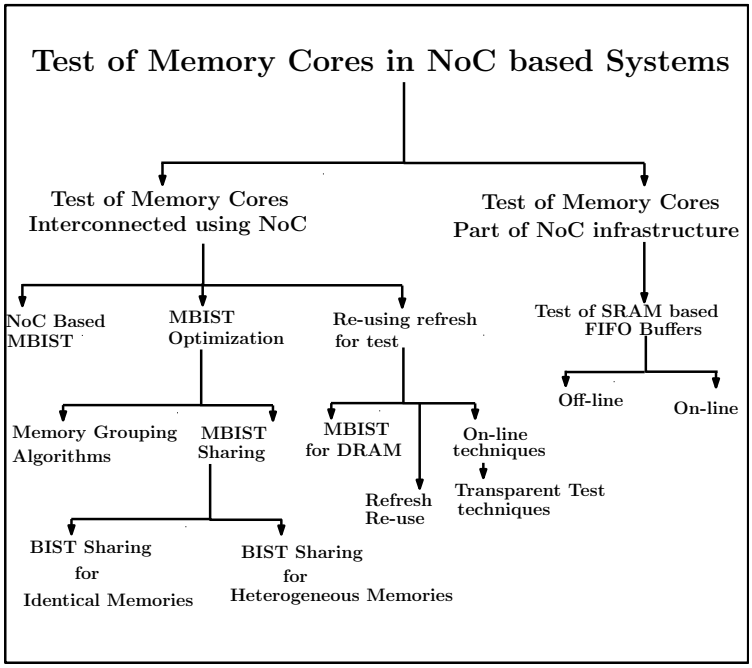


Figure 3.1: Structure of the survey done on related work

The research on the test techniques for NoC based memory cores has been guided along three directions. The first being a survey of literature related to NoC based MBIST. However, it has been found that not much has been done in this regard. Thus, the motivation shifted to search for optimization techniques employed for embedded memory cores in SoCs and applying them for NoC based SoCs. It has been found that researchers have followed two paths to bring about improvement, either by proposing memory grouping algorithms or by proposing efficient BIST circuitry sharing techniques among memory cores as illustrated in Figure 3.1. Thus, a survey has been performed on both the approaches and later applied in the proposals presented in the thesis. The third direction of research has been to exploit on-chip resources such as refresh circuit for test purpose with the aim to bring down the test area overhead. The survey of refresh based test methods presented in the thesis have included both on-line and off-line test techniques. However, survey of literature reveal that proposals related to refresh based test have been mainly

off-line test proposals targeting run-time transient errors and not faults and to the best of the knowledge, no literature has been found related to refresh based on-line test. However, the similarity of refresh and memory test algorithms motivated the proposal of an on-line test for DRAMs using refresh circuit presented in the thesis. Thus, a survey of literature related to on-line test of memories in general has been performed. An outcome of the survey has been the fact that transparent March tests have been preferred over standard March tests while performing the on-line test of memories. As mentioned in Chapter 1, one of the research directions of the thesis has been test of memory cores which are part of the NoC infrastructure. The memory cores considered have been the SRAM based First-In- First-Out (FIFO) buffers present in the routers of the communication medium. Thus, a survey of literature related to test of FIFO buffers has been performed which includes both on-line and off-line test techniques of FIFO buffers.

3.1 Studies on Network-on-chip based MBIST

Test time reduction by means of test parallelization can be achieved if an on-chip network is implemented as test access mechanism (TAM). Ever since Cota et al. proposed to use NoC as TAM in [23] and [24], the concept has gained immense popularity among research groups. The main advantage of the reuse of an on-chip network is the availability of a number of access paths to each core, depending on the number of interfaces between the system and the external tester. If the system functional inputs and outputs can be reused, the pin overhead is null, while the area overhead is minimum, since the access mechanism is already available on the chip. Test time can be also minimized if the intrinsic parallelism of the communication platform is explored to increase the test parallelization. Test parallelization in NoC-based systems can be further improved by the use of BISTed cores as mentioned in [22].

However, not much research has been done on exploring the NoC based BIST technique for memory cores and to the best of our knowledge only Liu et al. [59] have proposed a NoC based MBIST. Liu et al. in [59] have proposed a packet based BIST scheme which tests memories in a NoC. The packet based BIST scheme proposed in [59] is shown in Figure 3.2. The BIST uses the network resource to transport test patterns such that one BIST circuit can test all memories attached to the network resource without incurring the routing and timing problems. The proposed BIST scheme applies test operations to multiple memories in a pipeline. If a memory executes read test operation, the next memory will perform a write test operation. Thus, the memories are divided into two groups, one receives the read test operation while the other receives the write test operation. In static memory, the write power is larger than read power. Therefore, the number of memories that can be tested in parallel under a limited test power consumption

increases. However, the proposed work of Liu et al. has several problems. It proposes a single BIST controller for testing a number of identical memories. If we assume all memory cores are non-BISTed, then a single BIST controller has to send address, data and control for all the memories. This will substantially increase the network transport latency and hence negatively impact the test time. However, if we assume all memories are BISTed (each memory has an Address generator, Data generator and Test pattern generator), then the area overhead problem is the same as the case where each core has a dedicated BIST. No additional advantage is gained by sharing the BIST controller.

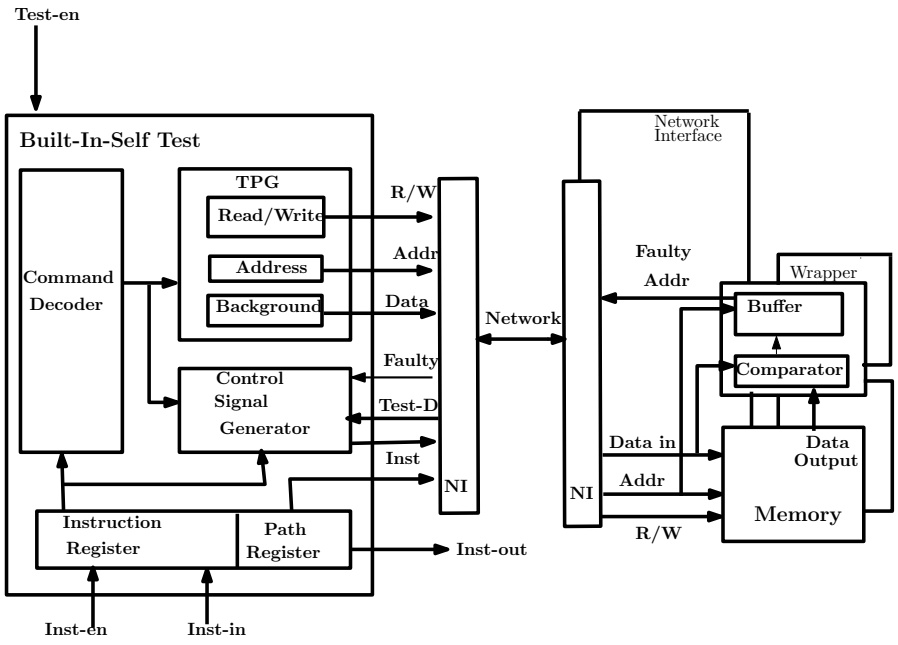


Figure 3.2: Block diagram of the packet based BIST scheme proposed in [59]

The experimental results presented by Liu et al. in [59] have been based on the assumption that the memory cores connected to the NoC are homogeneous. Such an approximation is too simple given the complexity of today's memory intensive NoC based applications and the different sizes of memories used in them. Moreover, no experimental results on any benchmark circuit have been provided to give an estimate of test time or power.

3.2 Studies on Memory BIST optimization

Memories with self contained BIST circuits in NoC based SoCs lead to high area overhead. Thus, researchers have proposed the concept of BIST circuit sharing. Literature suggests that there have been two approaches in sharing of BIST circuit. One is the *stand alone* approach

where there is a dedicated wrapper or controller for a memory core or cluster of memory cores and the other is the *distributed* approach where a single controller is shared to manage some or all of the MBIST wrappers in a SoC. However, in some cases, memories have different widths and addressing spaces and they cannot be connected to the same BIST logic. Therefore, to achieve a satisfactory solution, a decision on *which memory cores to be grouped* should be taken along with *how to connect the memories in a group*. Several heuristics have been used for deciding on grouping memories which share MBIST components for reduced area overhead. Memories that are physically far apart and/or belong to different physical hierarchies are placed in different groups. Memories that serve the same functionality are preferably grouped together to promote concurrent testing. For the same reason, memories that have same type (single port memories or dual port memories) and similar size parameters are grouped.

In the following subsections, we present a survey of different memory sharing techniques that have been proposed by different researchers. The proposed approaches have considered two main directions to gain improvement in performance of Memory BIST by hardware sharing: *Memory grouping algorithms* for optimized area, power and time and *Memory BIST architectures*.

Memory grouping algorithms

As far as we know, the first complete work on formulation of memory grouping problem has been proposed by Miyazaki et al. in [67]. A two pronged approach has been considered in [67]. First, a BIST methodology has been proposed which allows sharing of wrappers. Second, a memory grouping problem has been formulated with the aim of finding the optimized BIST architecture for any input design. The memory grouping problem formulation considered has been focused towards optimization of area under the constraints of power and time. To formulate the memory grouping problem, a graph theoretic approach has been utilized involving identification of serial and parallel compatibility subgraphs based on design constraints. The authors have proposed two methods for sharing BIST logic among memory cores, *serial* (connectivity of memories with same bit-width) and *parallel* (connectivity of memories with same word-width). Serial connection reduces the area overhead while requiring more test time than parallel connection. Several other problems have also been identified for the proposal. Components far apart on the chip have not been allowed to share their components due to congestion issues. Moreover, the proposed memory grouping problem formulation has been found suitable only if optimization is carried out after placement of memories.

An improvement on the optimization approach of [67] has been proposed by Zaourar et al. in [108]. The authors of [108] have presented a *Genetic Algorithm* (GA) based method to op-

timize the choice of BIST architecture among all valid possibilities with regard to additional area, peak power and testing time. Zaourar et al. have also used a graph theoretic approach of finding serial and parallel compatibilities. However, unlike min-cut algorithm used for partitioning problem used in [67], the authors in [108] have used the idea of picking equivalence classes for serial and parallel compatibilities. This alleviates the algorithmic difficulty of the memory grouping problem. Moreover, they have considered the memory grouping problem as a multi-objective problem, with area, time and power as the optimization criteria. A set of *pareto optimal* solutions for the optimization problem have been proposed and a multi-objective evolutionary GA has been used to sample the *pareto optimal* solutions. However, the shortcoming of the work presented in [108] is that, as all test wrappers are launched simultaneously during test, the peak test power becomes high.

A software based solution for MBIST, optimized for area, peak power and test time have been proposed by Zaourar et al. in [107]. The authors have proposed a software approach of three modules operating in sequence. The first module creates memory compatibility group, the second module applies *Genetic Algorithm* (GA) to the memory groups for optimized parameters and the last module checks the obtained solutions for the sharing constraints. However, no test results have been reported and no comparison with existing work is cited in [107]. Another BIST design optimization problem has been discussed in [17] where the authors have proposed an optimization tool for memory BIST design to minimize total test time, total routing length and total area under practical design constraints. They have proposed an iterative process of reaching optimization objective. The first step involves assignment of memory cores to controllers based on Integer Linear Programming (ILP) formulation. In the next stages, by an iterative algorithm, the initial assignment has been checked for violation of user defined constraints. On violation, the core violating the constraint is moved from one group to the other. A basic assumption of the work has been test of only one group at a time. This assumption leads to testing of memory groups in sequence which eventually increases test time.

Forming memory core clusters which can share memory wrappers has been accepted as a solution for reducing the Design For Testability (DFT) area overhead problem as discussed so far. However, sharing wrappers without caring for physical design problems has an adverse effect on at-speed operation of the memory cores. Thus, some works have proposed physical design aware memory grouping problem and their solutions as in [55] and [95]. Baosheng et al. in [95] have proposed a memory grouping technique for BIST circuit sharing which reduces the Design For Testability (DFT) area overhead utilizing the floor-plan information. Since at floor-plan level, the physical information about the functional blocks with embedded memories becomes available, grouping of memories at this level reduces the DFT area overhead. The au-

thors of [95] have also proposed a mechanism to avoid routing congestion involving grouping of blocks with the same type of memories and recreating a single BIST controller for the group at Register Transfer Level (RTL). Next, the new design is checked for routing congestion with the original design timing constraints. If no violations occur, further grouping proceeds in the similar way.

Kokarady et al. in [55] have presented a layout aware Programmable Memory BIST design (PBIST) synthesis flow. PBIST allows test algorithm alteration at the time of test application. Running different test algorithms make the test procedure more exhaustive and accurate as more number faults are detected. The authors have thus used PBIST to allow for use of several test algorithms at run time. However, PBIST has a more complex BIST insertion flow. If number of memory instances are large, PBIST controller sharing leads to sequential testing causing high test time. On the other hand, use of multiple controllers lead to area and power overhead. The authors argue that the present day synthesis tools are physical design friendly and make important decisions regarding cell placement and wire routing. Even insertion of DFT such as scan chains is layout aware. Thus, the problem of PBIST synthesis must also be layout aware as decisions taken at layout level influence routing congestion. The synthesis of Programmable Memory BIST (PBIST) is formulated as a combinatorial optimization problem where logical and physical architecture of the PBIST solution is obtained through several transformations. They have proposed an optimization framework called (Layout Aware Memory PMBIST Synthesis) LAMPS which is primarily intended for PBIST synthesis. It makes use of physical and logical transformations to PMBIST data path to improve rout-ability, total wire length, area, timing and power dissipation.

BIST sharing for identical memories

In present day NoC based SoCs, it is quite common to have many identical memories on-chip. Several works have proposed to share test wrapper for these identical memories to reduce test area overhead. Baosheng et al. in [95] have proposed a technique for partly sharing wrapper for identical memories. The technique involves sharing memory BIST controllers for embedded SRAMs (*e-SRAMs*) embedded in different functional blocks. The proposal considered each identical memory to have its own address generator, response compactor and memory control signal generator. The data generator and the command generator have to be shared among all identical memories. The proposal in [95] also considered a method for minimizing the number of BIST controllers by sharing them among memories which are neighbors. The authors argue that since the memory BIST controllers are generally implemented at the register transfer level (RTL), every functional block with e-SRAMs contains one or more BIST controllers. After en-

tering the gate level design stage, the physical information of those functional blocks is known and can be utilized to reduce DFT area overhead. The functional blocks that are close to each other in floor-plan and have memories with the same port type can share a single BIST controller. However, this way of reduction leads to routing congestion. The authors have proposed a mechanism to avoid routing congestion on account of reduction in number of controllers. The mechanism involves grouping of functional blocks having same type of memories and allotting a single BIST controller for the group.

A modular design of a wrapper enabling BIST for small memories has been investigated in [4]. The wrapper provides a standardized interface between memory and test controller, so that the wrapper allows for at-speed test at low area overhead. Both serial and parallel interfacing between BIST circuits and memory cores have been researched and number of modifications have been proposed for reduced area overhead. However all of them either lead to routing congestion suffering at-speed testability or increase the test time or suffer from increase in power dissipation during testing. A shared BIST scheme has been proposed by Chen et al. in [15] for testing multiple memories in parallel. The main drawback in such type of parallel connection lies in the interconnect between controller and wrappers, which uses one parallel command line to configure all the memory BIST wrappers to run the same test commands. This implies that for large SoCs, different types of memories (or memories requiring different test algorithms) cannot be tested simultaneously using the same BIST controller, thus increasing testing time as well as test control complexity. Moreover, using parallel interconnects between the controller and the wrappers, the routing congestion may become a potential problem when hundreds of embedded memory cores are present. Furthermore, the testing time for each test session is dominated by the largest memory, which may lead to prohibitively long testing time under power dissipation constraints. Serial interfacing techniques have also been proposed as in [69] and [51]. Serial techniques can reduce routing overhead. However, they lead to long testing time.

Identical memories are physically located at neighborhood and therefore read data of a Random Access Memory (RAM) can be used as written data of another RAM. By taking advantage of this feature, pipelined BIST has been proposed in [46], [47] and [59]. Huang et al. [46] have proposed a low cost BIST scheme to test identical memories in a pipeline. The BIST scheme uses serial interfacing technique to test each memory in an array of memories such that its area cost is small. On the other hand, it tests multiple memories in a pipeline to avoid the problem of long test time. The pipelined structure of BIST proposed in [47] allows only one test pattern generator and one test controller to support testing of multiple static random-access memories (SRAMs). This subsequently brings down the area overhead. In [47], the authors also propose a systematic procedure to convert March tests into a pipelined March test so that the tests can

be made compatible with respect to the pipelined BIST architecture. Chao et al. in [48] have presented an enhanced IEEE 1500 wrapper design for memory testing. The IEEE 1500 interface is mainly used for logic core testing and is not used for memory core testing as it leads to long test time. The enhanced IEEE 1500 wrapper wrapping the logic cores presented by Chao et al. in [48] can also generate March tests for RAMs. Thus, the RAM cores attached to these logic cores can also be tested by the enhanced IEEE 1500 wrappers. This brings down the area cost of the DFT circuits for RAMs in SoC.

BIST sharing for heterogeneous memories

Memory cores tend to have different configurations in word size, address space, pipeline latency and control protocol. Thus, clusters of memory cores with different configurations sharing BIST controllers or wrappers is referred as BIST sharing for heterogeneous memories. Concurrent testing is a simple and effective solution for minimizing test time for heterogeneous memory cores. However, in concurrent testing a dedicated BIST is attached to each core which leads to significant area overhead and complex test control. Researchers have proposed different BIST architectures such that area overhead problem can be reduced. Different parallel and serial interconnection techniques of BIST wrapper sharing have been proposed.

Benoso et al. in [9], present a programmable BIST architecture based on single programmable BIST processor and a set of memory wrappers to simplify the test of a system containing a large number of distributed multi-port memories of different sizes. The BIST processor is implemented as micro-programmable machine providing the test engineer a flexible and reusable block that can be used to manage the BIST of any number of memories of any size and is independent of test algorithm. However, this technique has the disadvantage of large test time. Wang etal. in [96] have proposed a Built-in-Self Diagnosis (BISD) architecture where multiple heterogeneous memory cores are tested parallely. The SRAM cores share the BISD but have dedicated background generator, control signal generator and the comparator. By using a different data back ground generator, the same test command can be applied to test memories of different word widths. The BISD implements the address generator to meet the largest address space among the memory cores. Cores with smaller address space will be turned off by the mask register when the BISD goes beyond its address space. The whole test finishes when the test completes for the largest core.

A BIST sharing technique for memories with multi ports as well as memories operating at different clock domains has been proposed in [96]. To exploit the advantages of both parallel and serial serial sharing of BIST wrappers, Denq et al. [28] have proposed a hybrid BIST scheme for testing heterogeneous memory cores. The proposed hybrid BIST scheme reduces

interconnections to minimize the routing area while uses parallel interface to test the memories at-speed. The MBIST components are the same as in [15]. The Test Pattern Generators (TPGs) are placed right next to their corresponding memory cores, and are connected with a scan chain instead of by dedicated serial input/outputs (I/Os). The area overhead of the routing wires thus is reduced significantly. The hybrid communication supports parallel testing, so it reduces test time effectively as compared with serial test. The hybrid BIST scheme reduced the routing overhead by having a test pattern generator for each memory. In addition, the position of the test pattern generators help in at-speed test of the memory cores. A serial control interface is used to control the test pattern generators of the memories under test.

In [30], a new flexible, hierarchical and distributed power-constrained embedded memory built-in self-test (BIST) architecture for complex and heterogeneous systems-on-a-chip (SOCs) is presented. The proposed architecture consists of a shared technology-independent BIST controller, low area and low power memory BIST wrappers and serial interconnect between them for low routing-overhead. Due to its flexibility, in addition to reducing routing complexity and achieving high test concurrency under power constraints, the presented solution can simultaneously support multiple test algorithms for heterogeneous memories, as well as embed custom test algorithms required for new memory faults.

One way to improve the effectiveness of nearly every memory test algorithm is to execute it at-speed. At-speed BIST operation means running the test at the same frequency as the fastest system operation. In some cases, at-speed testing even may exceed normal system operation to achieve better test quality. However, at-speed testing of heterogeneous memory cores is a difficult task. Traditional BIST architectures may fail to mimic true system at-speed operation because they take three cycles for each cell test: one for writing the pattern, one for reading it back, and one for comparing. One method to avoid this operation delay is to use pipelining stages; while reading from one cell, data can be readied for writing to the next cell [46], [47]. Not only does this method offer a higher quality test, but test time also can be decreased by up to a factor of three. However, if memories are working at high frequencies, and placed physically far apart, then testing memories at-speed with a shared BIST becomes impossible. Bahl and Srivastava [6] have proposed a low area shared BIST architecture that can test memories of different sizes using a single BIST at-speed. All the memories are surrounded by dedicated wrappers. The proposed BIST controller is a shared central block. It is like a normal BIST engine except it is capable of delaying the comparison of the expected output by a programmable number of clock cycles. The test procedure comprises of four steps. The first step involves selecting the memory to be tested. In the second stage the controller generates an initialization sequence to calculate the total pipelined delay. In the third stage, the controller delays the expected data by

the calculated number of clock cycles and in the final stage the test algorithm is run for detection of faults. The proposed architecture offers many advantages. Different memories on the chip can be tested using a single BIST, irrespective of their placement on the chip. The proposed method greatly reduces the test logic present on the chip as there is no need of a dedicated BIST for every memory. The proposed architecture can also be used for a programmable BIST, thereby allowing a single BIST to be shared between different types of memories that require different test algorithms.

To minimize test effort for embedded memories in SOC, automatic test integration of multiple and heterogeneous memory cores in a SOC environment is required. Linag et al. in [15] have presented an automatic generator for memory BIST circuits called BRAINS (BIST for RAM in Seconds). It has a graphic user interface and is integrated with a memory compiler to form an Intellectual Property (IP) generator for various memory configurations. It also features an automatic test grouping and scheduling which can optimize the overhead in test time, performance, and power consumption. With a configurable and extensible architecture, the proposed framework facilitates easy memory test integration for core providers as well as system integrators. The automatic generation of memory BIST cores extends the ability of system-level test integration, multi-port and multi-memory support.

3.3 Studies on re-using refresh for test of DRAM cores

One of the research direction of the thesis has been to exploit on-chip resources for test purpose. To this effect, re-using refresh circuit of DRAM cores for test purpose avoids use of additional BIST hardware and thus reduces area overhead. Therefore, a survey of literature related to refresh re-use based testing has been carried out. However, to the best of the knowledge, not much work has been done on the topic. The only notable works have been by Hellebrand et al. in [41] and [42] and by Yarmolic et al. in [101]. However, extensive work has been done on BIST implementation of DRAMs. A brief overview of the research on BIST for DRAMs based on March test algorithm have been provided in the following subsection.

3.3.1 BIST for DRAM testing

Some of the notable works in the field of BIST implementation for DRAM testing have been by Sridharan et al. [82], Inoue et al. [50], and Ohsawa et al. [73]. The proposals have been targeting detection of Stuck-at-fault (SAF), Coupling Fault (CF) and Address Decoder Fault (AF) in DRAMs. The BIST proposed in [82] uses a parallel signature analyzer which accesses bit lines from different arrays simultaneously. The problem with this architecture is that it requires

external tester to scan in test data and has low fault coverage. In [50], the parallel write operation allows only certain patterns to be applied to the cells so the technique cannot identify interference between memory cells. The BIST proposed in [73] has a low fault coverage. Pattern sensitive faults are considered typical of DRAMs and for DRAM testing, a neighborhood pattern sensitive fault (NPSF) testing model is more appropriate due to high fault coverage. A number of BIST techniques have been proposed for both commodity and embedded DRAM testing using the NPSF testing models [19], [76], [106]. However, no NPSF test can detect address decoder faults, whereas all March test can. Therefore the appropriate scheme would be to put both test algorithms in the BIST hardware [13]. BIST architectures have also been proposed such that both March test and NPSF tests can be programmed into the same hardware [10]. This allows both the tests to be run using the same hardware and thus has a higher fault coverage. However, putting both the tests in the same hardware increases the complexity of the BIST hardware leading to increased area overhead. Moreover, the time for programming the BIST for a particular test algorithm adds up to the total test time thus increasing the test time as well.

3.3.2 Refresh re-use for test

Cells in DRAM cores have to be refreshed periodically to prevent data loss. Refresh degrades performance of the system and wastes energy. However, refresh can be put to good use when applied for test of DRAMs. For example, Hellebrand et al. in [42] and Yarmolic et al. in [101] have tried to utilize refresh circuit for soft error detection in DRAMs. Both the propsals in [42] and [101] target the periodic consistency checking for embedded memories and are based on the BIST architecture. In [101], a signature based scheme of error detection has been used. The proposed strategy used by the authors has been sketched in Figure 3.3.

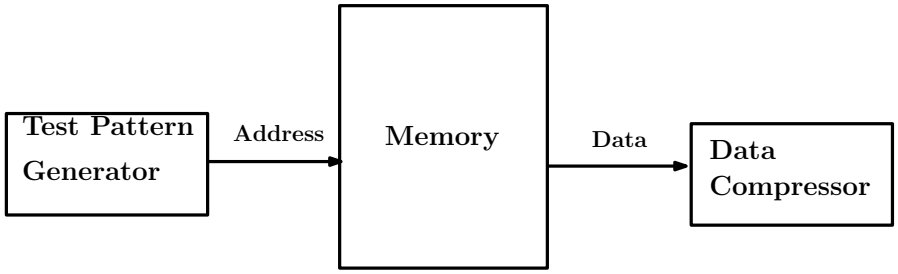


Figure 3.3: BIST structure for memories proposed in [42] and [101]

The test pattern generator generates a sequence of memory addresses which are predetermined. The corresponding data from each address is fed into the output data compressor and the final state of the compressor is used to represent a characteristic of memory. Initially, a

characteristic is generated for the correct memory content (C_{ref}) and stored in some particular location. Then the characteristic generation procedure is performed periodically for the memory. The subsequent characteristic generated for the memory is referred to as C_{test}. Each time a C_{test} is generated, it is compared with C_{ref} to reveal inconsistencies. The above proposed technique of Yarmolic et al. is an off-line test technique and incorporates an overhead of adjusting the reference signatures each time a write operation is performed. In other words, whenever a write operation is performed, reference signature is computed by scanning the entire memory. However, the authors of [101] also provided a self adjusting correction technique to overcome the overhead of scanning the entire memory. Their proposed adjustment technique performed adjustment of characteristic of a fault-free RAM concurrently with the write operations computed as modulo-2 address characteristic while ensuring same test quality is achieved as by a conventional approach based on signature analysis. However, the problem with the proposal of [101] is that being an offline test, it has to wait for the idle time of the memory.

Hellebrand et al. proposed an improvement on the work by Yarmolic et al. by providing an on-line error detection scheme in [41] and [42]. The basic testing technique in the proposals by Hellebrand et al. remained the same as in [101], that is, concurrently computing memory characteristic and comparing it with a precomputed reference characteristic. However, in [41] and [42], the authors have used a periodic refresh operation for concurrently computing the memory characteristic that needs to be compared with the reference characteristic. This on-line checking technique lowers error detection latency. However the technique leads to increased area overhead. The authors of [42] and [101] suggest that their architecture can be used for production test using any test algorithm. However, there are problems of using the signature based scheme. Firstly, the approach does not support active error detection. It does not alter the memory contents to trigger and detect functional faults. Secondly, the area overhead of storing the signatures. Thirdly, the problem of aliasing associated with any signature based scheme.

3.3.3 Online test of memories

Recent studies of DRAM failures in field ([49], [78] and [83]) provided strong evidence that DRAMs experience both *transient* (*soft*) faults and *permanent* (*hard*) faults in field but *permanent* faults constitute bulk of all DRAM failures. The field study published by Schroeder et al. [78] was based on Google's server fleet with data collected over a period of 2.5 years while the one published by Sridharan et al. [83] was based on Jaguar's memory system with data collected over 11 months. The study published in [49] was based on data collected from four different production systems. The most important conclusion drawn from the studies of [49], [78] and [83] has been the increase of hard faults alongside the soft errors during field operation of DRAMs.

The BIST based architectures proposed for DRAMs performing off-line March test on memories cannot detect latent hard faults which occur during field operation of memories. Thus, there is a necessity of on-line tests. An on-line test technique for memories has been provided by Corno et al. in [21]. However, the proposal resulted in duplication of memory. To prevent accumulation of both soft and hard faults, periodic test of the whole memory is necessary without disturbing the system operation. Moreover, such a test must ensure that the test must not destroy the contents of the memory and does not prolong the system operation. Such a test is referred to as *Transparent test*. Nicolaidis et al. in [71] first proposed transparent March test for memories and gave a systematic way to convert a standard March test to transparent March test. As mentioned in [71], transparent March test consist of two testing phases: (i) *signature prediction phase*, in which the golden signature is obtained, and (ii) *fault testing phase* where the fault is activated. The obtained signature after fault testing is compared with the golden signature to check whether the memory under test is faulty. Thus, the transparent technique proposed by [71] and all techniques based on it ([18], [45] and [64]) require signature computation for error checking. As a result, these do not perform active test and thus they fail to detect functional faults. Moreover, all the transparent test schemes mentioned above suffer from aliasing effect and increase in test time.

To reduce the test time while performing transparent test, a symmetric transparent test methodology has proposed in [102]. In this methodology, if a transparent test is not symmetric, then an additional state is added to make it symmetric. This causes the final content of the signature analyzer to be zero if no faults exist. Since the methodology eliminates the signature prediction phase, test time reduces. However, this methodology still suffers from the problem of aliasing. Transparent scheme avoiding aliasing has been reported in [87]. In [87], the authors propose a transparent on-line memory test (TOMT) for word-oriented memories for detection of soft errors as well as functional faults. The proposed TOMT technique uses transparent March tests with check bits instead of computing signatures. However, there are two major drawbacks with the work in [87]. First, the TOMT algorithm is not time efficient as it executes bit-wise manipulation to obtain the word-oriented transparent test. Secondly, the hardware implementation of the proposed TOMT algorithm incurs an area overhead proportional to the size of the memory. Naturally, with increase in size of memory the area overhead of the test circuit for the TOMT algorithm increases.

Since *Error Correcting Code (ECC)* is one of the most widely used method for increasing memory reliability, researchers have tried to integrate transparent testing with *ECC* ([58], [60] and [87]). This integration gives power to *ECC* so that it can be used for detection of functional faults as well. However, as already mentioned in the last section, *ECC* memories are costly

and hence these techniques will not find much use in DRAMs targeted for consumer electronics where cost is a major factor.

3.4 Studies on Test of FIFO Buffers

FIFO buffers in NoC infrastructure are large in number and spread all over the chip. Accordingly, the probabilities of faults are significantly higher for the buffers compared to the other components of the router. Both on-line and off-line test techniques have been proposed for test of FIFO buffers in NoC. FIFO buffers are tested in two ways, either when they are not in application (*Off-line*) or when they are in application (*On-line*). Off-line tests are suitable for detection of permanent faults developed during manufacturing process while on-line test are suitable for detection of run-time faults. On-line test can be further divided into two sub-classes. In the first sub-class, buffers are either in operational mode or test mode while in the second sub-class buffers being tested are in application and test at the same time.

Off-line test techniques have been reported in [34] and [43]. The authors in [43] propose a concurrent off-line functional test technique for NoC interconnect and routers. The proposal involves sending test packets through the network during NoC operation to locate interconnect faults. Special sequences such as *Walking One* sequences are sent as test data to detect interconnect faults which inherently test routers as well. Additional test packets are included with different test sequences to activate all faults in FIFO buffers. However, the proposal in [43] aims at detecting manufacturing faults in FIFO buffers while in this work, we consider faults which develop during in-field operation of the buffers. The proposal of Grecu et al. in [34], to have shared BIST controller for FIFO buffers is also suitable for detection of manufacturing fault.

On-line test techniques for detection of faults in FIFO buffers of NoC routers have been proposed by Nazarian in his thesis [70] and by Kakoee et al. in [52]. Both techniques reported in [52] and [70] fall in the first sub-class of on-line test. Nazarian proposed a new platform for on-line -structural test of routers in NoC in [70]. The main contribution of Nazarian's thesis is design of a wrapper with specific characteristics that can be used in the proposed platform for on-line structural NoC test.

The authors of [52] have proposed an on-line test technique where each router in the NoC is tested separately using its neighbor in different phases. Only the router under test remains in test mode while all other parts of the NoC continue to operate in functional mode. While considering faults in FIFOs, the authors have devised separate detection techniques for faults in the flip-flops of the FIFO and control circuit of FIFOs. The format of a data flit in the packet is judiciously chosen to cover all stuck-at faults and bridging faults in data-path of the routers in-

cluding the flip-flops of the FIFOs. To detect the control path faults, several test rounds or phases are performed. Each phase is responsible to cover some of the combinational paths inside the router under test. However, the problem with the approach proposed in [52] is that it allows only one router to be tested at a time. Thus, only one FIFO buffer gets tested. Moreover, the test process involves transfer of test packet between router under test and its neighbors. As a result, a number of test cycles are spent in test packet transfer. Thus, increasing test time and affecting the normal operation. On-line test technique proposed for FIFO buffers in both [52] and [70] consider standard cell based FIFO buffers while we consider SRAM based FIFO designs. Thus, faults considered in this work are different from those targeted in [52] and [70].

To the best of our knowledge, no work has been reported in literature that proposes on-line test of SRAM based FIFO buffers present within routers of NoC infrastructure. However, as the FIFOs considered in this work are SRAM based fifos, we were motivated to survey on-line test techniques for SRAM based FIFOs in general. SRAM based FIFOs are tested using either of the following two approaches, dedicated BIST approach as proposed by Barbagallo et al. in [7],Van de Goor et al. in [92] and [93] and Zorian et al. in [109] or distributed BIST proposed by Grecu et al. in [34]. Since dedicated BIST for each FIFO buffer would mean prohibitively large area for BIST, Grecu et al. in [34] proposed a better approach of dedicated BIST with a shared controller and stimulus generator. The BIST controller writes the same test packets to all the FIFO buffers on the chip. Then, the Local Response Analyzer (LRA) detects each buffer and reports back error information (if any). However, the approach proposed in [34] increases routing cost as each FIFO is connected to the BIST controller. Both dedicated and distributed BIST approaches being off-line test techniques fail to detect permanent faults which develop over time.

3.5 Summary

This chapter presents different horizons of research in the field of test of memory cores interconnected using NoC. It has been observed that not much has been done in this field. As a result, the survey was directed towards test of memory cores in the SoC environment with the aim of reducing area overhead at optimized test time and test power. Moreover, other aspects of memory testing such as on-line test techniques have been explored revealing the necessity of transparent testing during on-line test of memories. As this thesis focuses on testing memory cores interconnected using NoC, Chapter 4 will emphasize on the NoC based MBIST architecture, covering its test challenges and elaborating the proposed distributed and hybrid test technique. It will also cover the details of the power aware schedule supplementing the proposed distributed test architecture.

Chapter 4

Network-on-Chip based MBIST

The inter-core communication architecture in SoCs has observed a paradigm shift from traditional bus- based methods to packet based communication over the last decade. The use of NoC to act as the communication medium has allowed the SoC designers to overcome the problem of bandwidth, low performance and high power dissipation encountered in bus based methods [8]. In a NoC-based chip, the communication network consists of network interfaces (NI), routers, and channels to connect the routers. The cores communicate among themselves by sending and receiving packets. Details of the NoC infrastructure has already been covered in Chapter 2.

As in case of any SoC based design, cores interconnected using NoC must also be tested for manufacturing defects. However, a major design issue in SoCs has been the design of Test Access Mechanism (TAM). In NoC based SoCs, the area overhead due to the communication infrastructure is large. In such a situation, a dedicated TAM solely for the purpose of test is impractical as it would further increase the area overhead. Thus, the re-use of the NoC as TAM has been an attractive solution as it overcomes the problem of additional TAM area overhead and facilitates test parallelism. Ever since, it had been proved by Erika Cota et al. in [23] that reduced test times can be achieved by the network reuse even under power constraints while pin count and area overhead are strongly minimized, the NoC re-use for TAM has gained immense popularity among different research groups. Based on the re-use of NoC for data communication as TAM, different researchers have proposed NoC based SoC test scheduling techniques.

4.1 Motivation

Majority of research related to re-use of NoC to act as TAM have focused on testing of logic cores. A few notable among them have been by Amory et al. in [5] and Erica Cota et al. in [24]. However, not much has been done on exploring the NoC to act as TAM while testing memory

cores interconnected using NoC. One probable reason may be that Built-In-Self Test (BIST) is the most preferred technique for testing memories and Memory BIST (MBIST) does not need any TAM as tests generation as well as comparison of results are done on-chip.

However, MBIST for memories connected using NoC also face the same test challenges as faced by any other BIST technique for embedded memories. Unless carefully designed, NoC based MBIST may induce excessive power, in addition to performance and area overhead. For instance, with hundreds of embedded memories, dedicated BIST for each memory core leads to high routing and gate area overhead. To reduce the BIST area and routing overhead, distributed approaches are necessary. However, as hardware resource sharing is introduced in distributed memory BIST, the testing technique must be carefully considered to reduce the routing congestion and to facilitate rapid power-constrained testing. Parallel testing can reduce test time, but power consumption may be a factor. Sequential testing has the opposite effect. Test area overhead reduction while allowing at-speed testing can be achieved if an on-chip network is implemented as test access mechanism (TAM) for testing memory cores.

The focus of the research proposed in this chapter has been towards devising a MBIST scheme for memory cores interconnected using NoC at optimized test time and test power at minimum area overhead. To reduce the area overhead a distributed BIST architecture has been proposed which involves grouping memory cores into clusters and each cluster being tested by a dedicated BIST controller. An important requirement for a distributed BIST scheme is the choice of the grouping technique for the memory cores which is governed by the design metric (time/power) that needs to be optimized.

In this chapter two memory grouping technique formulations have been presented. Although the BIST architecture considered for both the grouping techniques involve a distributed approach, the objective of both are different. The first memory grouping technique is based on a Particle Swarm Optimization (PSO) formulation focusing on reduction of test instruction/data transport latency to bring down the overall test time. The second technique applies a heuristic to group memory cores applying distance and timing constraints and focusing on reduction of total power dissipation during test.

4.2 Proposed Method : Distributed and Hybrid Test Architecture

Literature suggests that researchers have either proposed a serial testing approach or a parallel testing approach during BIST design optimization for memory cores embedded in SoCs as can be found from some notable works by Baoshen et al. in [95], Dostie at al. in [69], Huang et al. in [47],and Denq et al. in [28]. Testing the memory cores one by one keeps the power dissipa-

tion under check but increases the test time while testing a number of cores in parallel makes testing faster at the cost of higher power dissipation. Thus, the best solution is to leverage on the advantages of both the approaches.

The proposed test architecture is a distributed Memory BIST (MBIST) architecture having a number of BIST controllers each of which is responsible for testing a group of memory cores. The groups are tested in a pipeline while memories in a group are tested in parallel. The hybrid test technique and the distributed BIST architecture allows the test of memory cores to be performed at much lesser time than required in [59]. The packet based BIST re-uses the NoC to act as TAM bringing down the routing cost. Moreover, utilizing the already existing on-chip network for test purpose avoids requirement of test circuits and thus reduces area overhead.

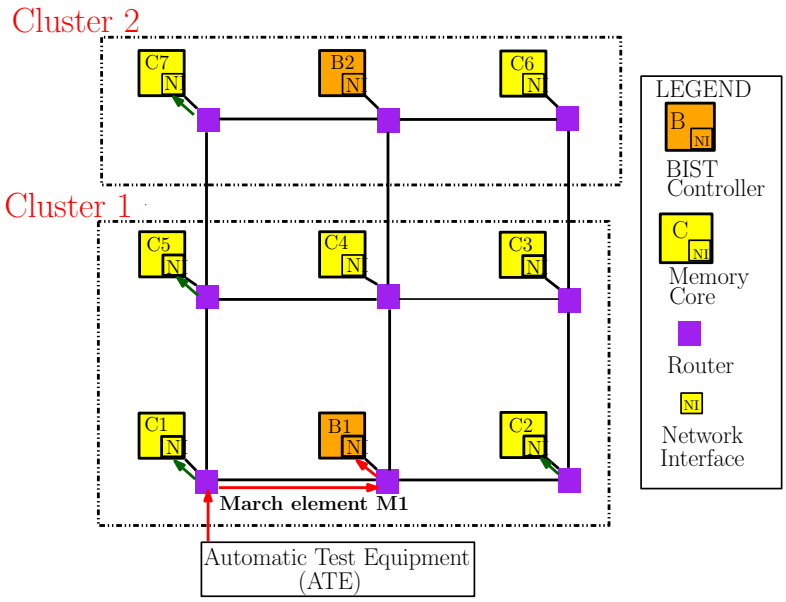


Figure 4.1: Proposed NoC based MBIST Test Architecture (first test phase : First March element M1 transferred to B1)

The basic idea of the hybrid approach of testing used in the proposed architecture is illustrated in Figures 4.1, 4.2 and 4.3.

A 3x3 mesh type NoC has been considered in all the three figures which follows a X-Y routing mechanism and utilizes wormhole switching.

The memory cores to be tested are grouped in two clusters. The grouping is done based on either a PSO based approach or by applying some heuristic (both techniques have been discussed in the next sections). After the grouping has been done, based on the timing precedence relation, the test schedule determines the order in which the BIST controllers will receive the March tests. Let the BIST controllers for the Clusters 1 and 2 be B1 and B2 respectively and the maximum

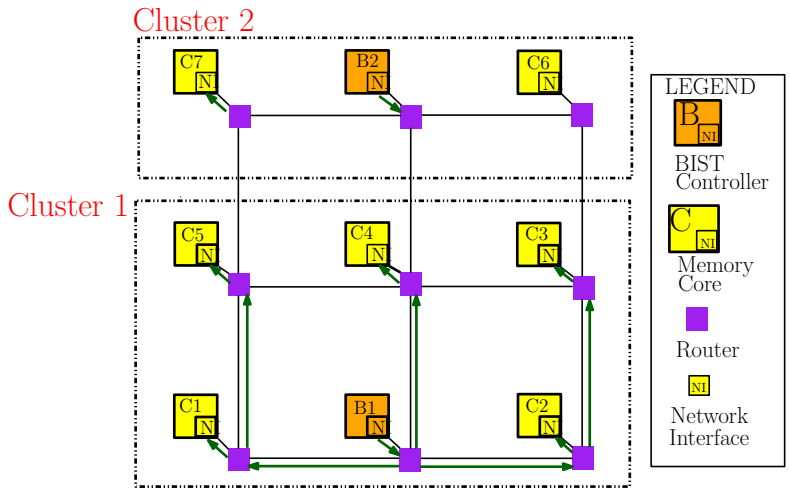


Figure 4.2: Proposed NoC based MBIST Test Architecture (second phase : memory control signals from B1 to cores of cluster 1)

test time of Cluster 1 is greater than maximum test time of Cluster 2. Hence B1 receives March tests earlier than B2. Assume that the memory cores are tested using a March test with two March elements M1 and M2. The sequence of test operations will be as follows:

a) First phase of the test session: The ATE transports M1 to B1 (in this case B1 is the BIST controller of the group that has the core having the maximum test time among all cores) as shown in Figure 4.1.

b) Second phase of the test session: B1 decodes the instruction and accordingly generates low level memory control signals for all the memory cores in cluster 1 as shown in Figure 4.2. All the cores in Cluster 1 are tested in parallel for M1. Once every core in cluster 1 passes the test for M1, the information of test completion is returned to B1.

c) Third phase of the test session: Once Cluster 1 completes the test for M1, B1 passes M1 to B2 and itself fetches M2 from ATE which is shown in Figure 4.3. B1 decodes M2, B2 decodes M1 and both B1 and B2 generate low level memory control signals for their respective clusters Figure 4.3. Cores in Cluster1 and Cluster 2 are tested parallely for test M2 and M1 respectively.

The testing time of a memory core is determined by the *address space* of the memory core and the length of the March algorithm. The length of the March element can be calculated by the *word width* of the core. Therefore, the testing time of the memory core can be computed using the product of *address space* and *word-width*. If N is the *address space* of a memory core and w be the word width then, the test time of the core = $N * w$. For a group of memories tested in parallel using the same March test, the testing time of an i-th core (i = 1,2.....,n) t_i = N_i*w_i and the test time of the group is the maximum of all the test time of all the cores in the group.

Pipelining brings in test parallelism among groups and performs test faster as compared to

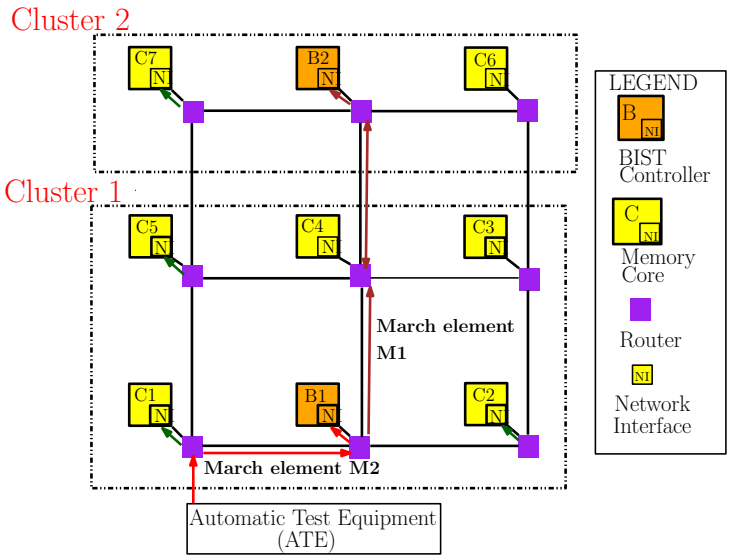


Figure 4.3: Third phase : March element M2 transferred to B1 and March element M1 transferred to B2

other proposed serial test techniques. However, as in the case of any pipelined architecture, it must be ensured that there are no pipeline stalls. Pipeline stalling in the proposed architecture can be avoided if a timing relation is maintained among the testing groups. The group with the largest test time has to be tested first followed by the one with the second largest and so on. For example, according to Figure 4.3, to avoid pipeline stalling in the second test session, Cluster 2 must finish test for M1 before Cluster 1 finishes test for M2. This precedence constraint of testing the memory core groups has to be taken care of while scheduling test for the architecture.

The proposed test architecture uses a hybrid approach where number of cores in a group share a BIST controller. Thus, an obvious question that comes up is *what should be the condition* for group formation and *how many groups* should be formed? It is expected that the number of groups that are formed is minimum so that the area overhead due to the BIST controllers is minimum. Moreover, the groupings should also ensure that the power dissipation during test is within the power budget. To ensure minimum power dissipation and reduced test time, two different strategies have been proposed which are discussed in the next two sections. The first proposal utilizes a PSO based approach of grouping memory cores with the aim to reduce test instruction transport latency and hence bring down the area overhead. The memory grouping problem has been formulated as a placement problem considering reduction of test instruction transport latency as the only objective. The locations of the BIST controllers are computed using the Particle Swarm Optimization (PSO) algorithm and memory cores are assigned to the controllers based on a greedy approach. The second proposal applies a heuristic grouping tech-

nique whose aim is to group memory cores which are at same distance from a BIST controller. Then, groupings are improved to provide a test schedule that satisfies the power constraint. The proposed test scheduling strategy consists of two phases: the initial phase of memory grouping, followed by the power aware test schedule phase.

4.2.1 PSO based memory grouping

This section presents the formulation of the problem of finding the best position of the BIST controllers in the proposed NoC based test architecture and allocating memory cores to the controllers such that the transport latency is minimized. The transport latency develops due to the time required to transport the test instruction from one controller to the other as well as transportation of low level control signals from a BIST controller to its corresponding cores. Out of the two components of the test transport time, the time for transportation of instructions among controllers is the more dominant one. For a mesh type network-on-chip, the controllers are placed at different locations in the mesh, and transportation of a instruction involves its movement from the source controller to the destination controller, hopping through a number of routers in between. Each hop from one router to the next involves a number of router clock read cycles (governed by the design of the router). Thus, greater the distance between two BIST controllers, larger is the time required for the instruction to travel from source to the destination and more the number of clock cycles required to finish a test. Thus, the transport of test instruction brings about a latency in testing and hence we referred to it as test transport latency in the thesis. This test transport latency is applicable for each test pattern that is communicated among the BIST controllers.

The problem of optimized transport latency can be stated as,

> *Given a mesh-based NoC with X-Y routing algorithm; a number of memory cores, and the location of these cores in the NoC, maximum number of allowed BIST controllers, determine the location of the controllers and assign memory cores to controllers such that the total transport latency is minimized.*

The problem stated above is treated as a variant of the Uncapacitated Facility Location problem (UFL) [35], where a number of customers (memory cores) have to be alloted to a number of facilities (BIST controllers) such that the total cost is minimized. The total cost in the UFL problem is the distance traveled by a customer to its nearest serving facility. In the problem stated above the total cost is the sum of setup cost (C_{setup}) and service cost ($C_{service}$).

The set up cost (C_{setup}) of a BIST controller is the total latency in transporting the March instruction to each of the BIST controllers. The service cost ($C_{service}$) of a BIST controller is the maximum of the number of hops from the BIST controller to its alloted memory cores. The mathematical formulation of the problem considering Figures 4.1, 4.2 and 4.3 is as follows.

$$C_{setup} = h_{ATE_B1} + h_{B1-B2} \tag{4.1}$$

$$C_{service} = Max(h_{B1-C1}, h_{B1-C2}, h_{B1-C3}, h_{B1-C4}, h_{B1-C5}) + Max(h_{B2-C6}, h_{B2-C7}) \tag{4.2}$$

where, (h_{i-j}) is the number of hops from core i to core j.
The objective function *Z = Min (C_{total})* where, C_{total} = C_{setup} + $C_{service}$ subject to the constraints that the maximum number of controllers to be alloted is fixed and each memory core is alloted to exactly one BIST controller.

4.2.2 PSO based optimization algorithm

Research suggests that meta-heuristic solutions have been preferred over exact solutions for the Uncapacitated Facility Location (UFL) problem [35], as it is an NP-hard problem. Since, the problem stated in the previous section can be treated as a variant of the UFL problem, we have also used a meta-heuristic approach like Particle Swarm Optimization (PSO) for the problem similar to [35]. Other optimization and search techniques such as Genetic Algorithm (GA) could also have been used. However, the use of PSO has been motivated by the fact that PSO is more computationally efficient (uses less number of function evaluations) than the GA as mentioned in [40]. To investigate this claim, Hassan et al. set two statistical tests to examine the two elements of this claim, equal effectiveness but superior efficiency for PSO over the GA. The results of the *t-tests* support the hypothesis that while both PSO and the GA obtain high quality solutions, with quality indices of 99% or more with a 99% confidence level for most test problems, the computational effort required by PSO to arrive to such high quality solutions is less than the effort required to arrive at the same high quality solutions by the GA.PSO [54] is a population based stochastic technique which is initialized with a group of particles with random position and searches for optima by updating their position through generations. The different parameters used in the PSO based formulation are defined as follows.

a) *Particle structure*: *n*-bit binary array, *n* being the total number of possible controller position. So a 1 in i^{th} position indicates presence of a controller in the i^{th} location, and 0 otherwise.

b) *Fitness function*: Objective function mentioned in the previous section.

c) *Global Best (GB) and Particle best (PB)*: Particle best of a particle has the minimum fitness among all particles generated through changes to that particle, across generations. Within a particular generation, the particle resulting in the minimum fitness function is the global best. Both GB and PB (for a particle) are updated after each iteration.

d) *Evolution of particles*: Mask operators have been defined to find the new position of a particle. Mask operators are calculated separately based on GB and PB positions. Mask operator is calculated by comparing bit by bit with GB and the particles current position. If controller is present/absent at a particular bit position in both GB and particle then bit at the same location of the Mask operator is set to 0 and 1 otherwise. In other words, when a bitwise-XOR is performed between particles current position and mask operator, it will produce GB. However, not all bits are XOR-ed. Based on a random probability, certain bits in particle position are left unaltered. The rest is found by XOR-ing the current position with Mask operator. After applying the mask operators for GB, the particle positions also go through the same process for their respective PB. The position obtained after applying the PB mask operator is considered as the particles new positions.

e) *Termination condition*: The algorithm terminates if the particle with minimum fitness does not change for the last 20 generations. Moreover, a maximum iteration condition is also (200 in this problem) considered to terminate the process.

4.2.3 Experimental results and evaluation

Since there is no published work for the optimization problem stated in this paper, the PSO based method has been compared with a heuristic method, *Neighborhood Allocation* method (*NA*) similar to the approach taken in [17]. In the PSO based approach, the number of allowed BIST controllers are placed at locations using the Mask operator technique explained in the previous section. After the BIST controllers have been placed, memory cores are alloted to them using a greedy approach. Simulation of the NA and PSO based techniques have been performed on a 2.0 GHz dual core Linux workstation with 4 Gb memory. The experimental results have been shown in Table 4.1. The values in Table 4.1 represent calculated fitness values. The percentage allotment is the percentage of total number of blocks in the test architecture that are BIST controllers. The data from Table 4.1 reveal that the best percentage improvement of PSO based approach over NA technique is obtained with a mesh size of 32x32 and with 25% allotment.

Table 4.1: Cost values calculated for different mesh sizes wih different percentage allotment of blank spaces for BIST controllers

Mesh size	Technique	Allotment		
		25%	50%	75%
4x4	NA [17]	30.00	21.62	20.73
	PSO	24.67	17.73	17.65
8x8	NA [17]	96.63	52.46	43.00
	PSO	54.66	31.13	31.00
16x16	NA [17]	144.00	76.00	67.00
	PSO	71.00	50.00	54.00
32x32	NA [17]	323.61	132.53	105.46
	PSO	122.13	97.53	91.00

The results illustrated in Table 4.1 reveal that for the two-level test architecture considered in the chapter, as the number of BIST controllers increase, the cost value decreases indicating reduction in transport latency. However, there are a few shortcomings for the PSO based grouping methods. The basic assumption to consider homogeneous memory cores is an over simplification of the problem as in practical NoCs, the memory cores are of different sizes. Similarly, the objective of optimization considered in PSO based approach is rather a simple one, considering only transport latency. However, test time cannot be considered independent of test power.

The above two shortcomings mentioned for the PSO based approach has been the motivation for the power aware memory grouping technique discussed in the next section.

4.3 Power Aware Memory Grouping Technique

The grouping technique presented in this section is an improvement over the PSO based approach discussed in the last section as it allows test of heterogeneous memory cores (memory cores of any size) unlike the PSO based proposal which allowed only homogeneous memory cores. Additionally, the PSO based approach has been focused only towards reduction of network transport latency. However, the proposal presented in this section considers a grouping technique which takes into account both timing and power constraints. Based on the proposed NoC based MBIST architecture, experiments have been performed on d695 ITC'02 benchmark circuits [63] to confirm that the proposed heuristic memory grouping based test schedule performs a more power constrained test as compared to dedicated BIST technique.

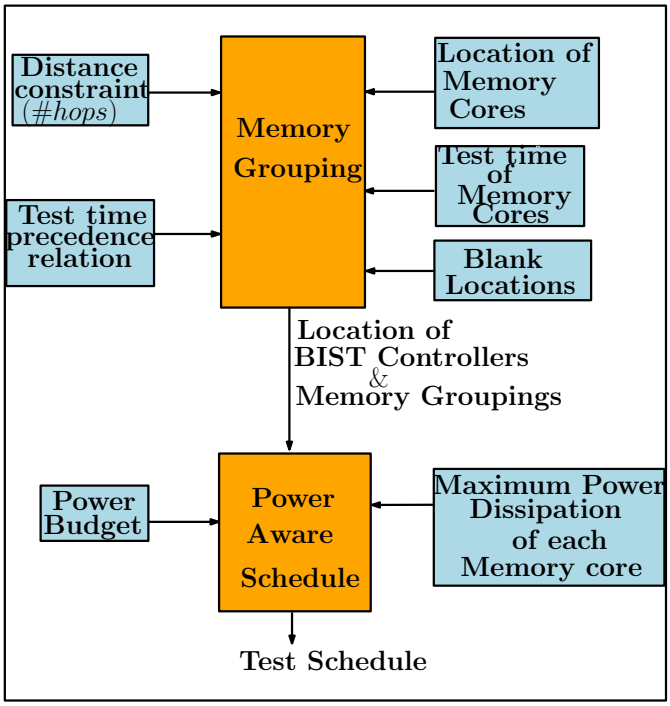


Figure 4.4: Phases in the proposed test schedule

4.3.1 Test scheduling problem

The proposed test scheduling strategy consists of two phases: the initial phase of memory grouping followed by the power aware test schedule phase. As shown in Figure 4.4, the memory grouping phase is based on the timing precedence constraint as well as on the distance constraint. Before the memory grouping problem and the power aware test schedule is formally defined, certain notations are described as follows.

n : Total number of memory cores

r : Number of rows in the NoC ; cl : Number of columns in the NoC

A : l-element march test $\{M_1, M_2, ..., M_l\}$ where M_l is the l-th March element

C : Set of memory cores $\{C_1, C_2, ..., C_n\}$

C_i : i-th core in the set C, where {i = 1,2,.....n}

$t_{C_i}(M_l)$: test time of the core C_i for the l-th March element M_l

B : Set of possible locations for placing BIST controllers $\{B_1, B_2, ..., B_m\}$ where $m < n$

B_j : j-th core in the set C, where {j = 1,2,.....m}

x_{C_i} / x_{B_j} : row number of C_i / B_j in a mesh type NoC, where {x = 1,2.....r}

y_{C_i} / y_{B_j} : column number of C_i / B_j in a mesh type NoC, where {y = 1,2,.....cl}

G : Set consisting of groups of memory cores

g_j : a memory group, $g_j \in$ G and $g_j \subseteq$C

$T_{g_j}(M_l)$: Maximum test time of a group $g_j \in$ G for March element M_l

$T_{g_j}(M_l)$ = Max$(\forall_{C_i \in g_j} t_{C_i}(M_l))$

h : number of allowed hops in the NoC

4.3.2 Memory grouping problem

The Memory grouping problem is stated as:

> *Given a NoC with n number of memory cores, their testing time and their positions (in terms of row numbers and column numbers), m number of possible BIST locations and their positions, partition n into groups and allot a BIST controller to each group based on timing precedence constraint and distance constraint.*

Distance constraint: The distance between the memory core and its alloted BIST controller is less than a pre-determined number of hops.

Timing precedence constraint: It requires that the group that is tested for a March element finishes the test before it receives the next March element from a group that was operating on it.

4.3.3 The memory grouping algorithm

The objective of the memory grouping algorithm is to form clusters of memory cores where each cluster can contain memories of different sizes and the clusters be formed in such a way that the cores which are at same distance from a controller fall in one group. The groups are then ordered to maintain timing precedence relation. The memory grouping problem also assumes available blank positions where BIST controllers can be placed. These blank positions can be at the center or along the sides or at the corner of a NoC. For example, for a NoC of size *r*cl* (where *r* is the row number and *cl* is the column number), the different positions are identified as follows. The position of a core is given by an ordered pair (x_j,y_j) where *x* is the row number and *y* is the column number. Corner locations are the ones which have (x_j,y_j) such that x_j can be either 1 or *r* and y_j can be either 1 or *cl*. Similarly, the side locations are the ones which satisfy the condition that if x_j can have values either 1 or *r* then j can be anything from 2 to *cl*-1 and conversely if y_j can have values either 1 or *cl* then x_j can be anything from 2 to *r*-1. The center locations are the ones which allow x_j and y_j to allow any value within the range 2 to *r*-1 and 2 to *cl*-1 respectively.

In the next subsections, it is shown that if BIST controllers are placed at these blank locations, then for particular number of allowed hops from a BIST controller to a core, the number

of cores that can be reached from a BIST controller is maximum for the controllers placed at the center locations followed by the ones at the side and then the ones at the corner (provided as a Lemma and its corollary in the next subsection). The grouping technique utilizes a greedy approach of trying to allot as many cores as possible to the locations at the center assuming that BIST controllers are placed at these locations. It is ensured that each core is alloted to exactly one controller.

After allotment of cores to all center positions, if it is found that some memory cores remain unallocated, then the grouping algorithm tries to allot these unallocated cores to the controllers at the side locations followed by controllers at corners. After trying to allot each core to a controller in either of the three positions, still unallocated memory cores remain, then the steps for grouping are repeated with increased allowed hops. Once every memory core has been alloted to a controller, the groups formed are the ones which share the same BIST controller. The groups are then sorted in decreasing order of test time. The results of the memory grouping phase is number of memory groups (number of required BIST controllers), position of the BIST controllers and allotment of memory cores to the BIST controllers. The test architecture allows for parallel testing of memory cores in a group. Moreover, the pipelined technique of overlapped test operation of two or more groups will cause a number of memories of different groups to perform the same test operation. This may result in power dissipation which may exceed the power budget. Thus, a power aware schedule is required after the initial memory grouping. Based on the results of the memory grouping phase and on the power constraint, the next phase which is the improvement phase generates a power aware test schedule.

4.3.4 Placement problem for the BIST controller

As mentioned in the previous section, the best position for placing a BIST controller is the center location of the NoC followed by ones at the side and then the ones at the corner. This placement problem is provided as a lemma. The lemma and its proof is discussed next.

Lemma 4.3.1. *In the proposed test architecture, for a given number of hops (h), the number of memory cores that can be reached from a blank location is given by*

$$T_h = (h/2)[d(1+h) + (1-h)] \tag{4.3}$$

where d = 2 for blank locations at center positions, d = 3 for blank locations at side positions and d = 4 for blanks at center positions.

Proof by induction. The proof is provided for only the corner case. The same approach can be

used for other cases, and hence is not detailed here.
Let P(h) be the proposition that

$$T_h = (h/2)[2(1+h) + (1-h)] \tag{4.4}$$

where d = 2 (a BIST controller placed at a corner location can hop only in two directions).

Basis step : To prove P(1) is true, we assume the left most top corner. The cores that can be reached from there at one hop are cores to the right and at the bottom of the corner position only. The same logic can be applied for the other three corner positions and in each case the number of cores that can be reached is 2,
i.e. T_1 = 2 and (1/2)[2(1+1) + (1-1)] = 2. Thus, P(1) is true.

Inductive Step: Assume P(h) is true. Thus,

$$T_h = (h/2)[2(1+h) + (1-h)] \tag{4.5}$$

where d = 2 is true. It must be shown that

$$T_{h+1} = (h+1/2)[2(1+(h+1)) + (1-(h+1))] = (h+1)(h+4)/2 \tag{4.6}$$

where d = 2 is also true.
It can be seen that if a corner blank location is considered, then to obtain the number of cores that it reach in two hops is equal to number of cores that it can hop to in single hop plus an additional 3 places. Extending that to case of h and $h+1$, we can derive that the number of cores that can be reached from a blank location at a corner position in $h+1$ hops is equal to places that can be reached in h hops plus an additional $h+2$ locations. Mathematically, it can be written as $T_{h+1} = T_h +$ h+2. Since we have assumed P(h) is true, then substituting the value of T_h we obtain

$$T_{h+1} = (h/2)[2(1+h) + (1-h)] + h + 2, \tag{4.7}$$

where d = 2
Therefore, $T_{(h+1)}$ = (h+1)(h+4) / 2. Thus, P(h+1) is true and thus proves the lemma.

□

Corollary 4.3.2. *For a particular number of allowed hops, the number of cores that can be reached from a blank location depends on the position of the blank location (the value of* d *in* T_h*). Greater the value of* d *more the number of cores that can be reached. Thus, a blank at the*

center position of the NoC is the best place to put a BIST controller in the NoC.

Illustrative example

The operation of the Memory grouping algorithm has been illustrated using Figure 4.3.4 which uses a 4×4 matrix to represent a 4×4 NoC. The example is shown for a particular permutation of memory cores and BIST controllers. The BIST controllers are represented by B_1 and B_2 while the memory cores have been enumerated from C_1 to C_{10}.

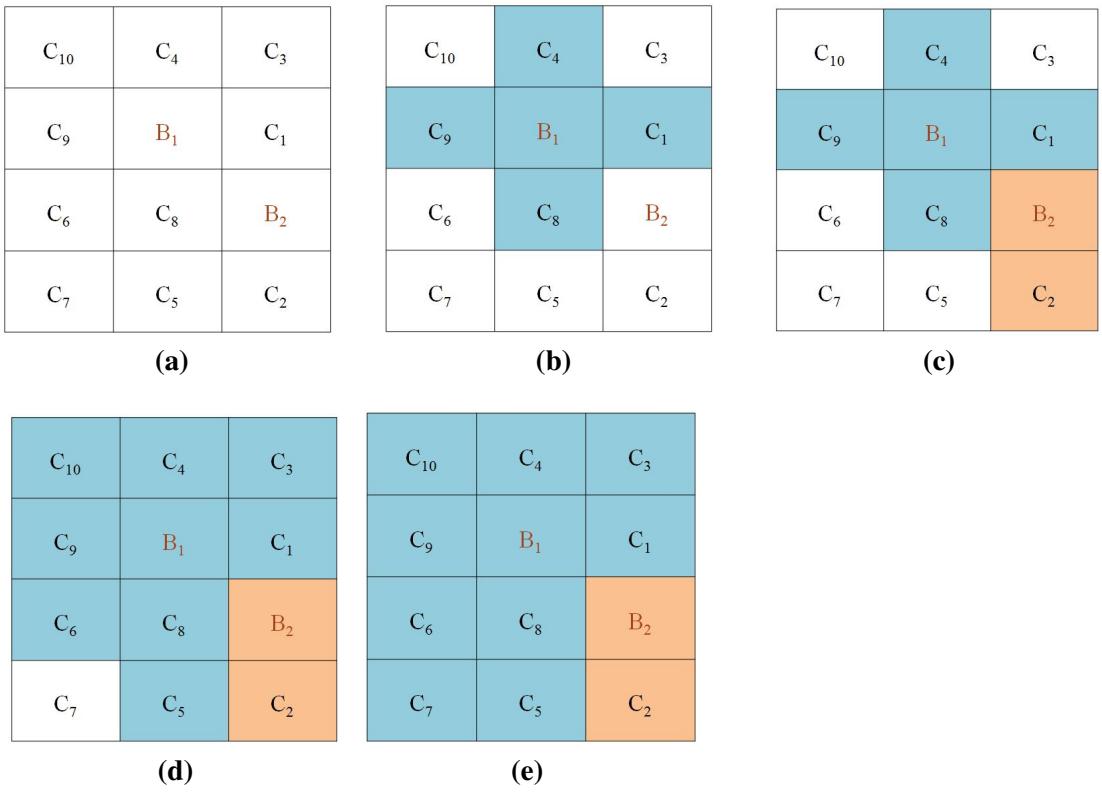


Figure 4.5: Illustrative example of Algorithm 2 applied on 4x4 NoC (a): initial configuration, (b): grouping cores to centrally located controller for h = 1, (c): grouping cores to edge located controller for h = 1, (d): grouping cores to centrally located controller for h = 2, (e): grouping cores to centrally located controller for h = 3

4.4 Experimental results

ITC'02 benchmarks [63] were used to evaluate the proposed test architecture and the test scheduling algorithm. As the original benchmarks do not include test data for memory cores, we have

Algorithm 2 Memory Grouping Algorithm

Input :

Size of the NoC = $r * cl$

Set of memory cores C and (x_{C_i},y_{C_i}) of each core $C_i \in$C

Set of possible locations for placing BIST controllers B and (x_{B_j},y_{B_j}) of each location B_j

x_{C_i}/ x_{B_j} where $\{x = 1, 2.....r\}$

y_{C_i} / y_{B_j} where $\{y = 1, 2, ...cl\}$

t_{C_i} : test time of the core C_i, t_{C_i} = N_i*w_i where N_i is the address space of the core C_i and w_i is the word width

p_{C_i} : maximum power dissipation of each core C_i

/* In the algorithm we assume that t_{C_i} = $t_{C_i}(M_l)$ */

Step 1 : Initialization

(a) Let the number of hops be 1.

(b) For each memory core $C_i \in$ C,$\forall$i(1,n) use a flag (say visited(C_i)) to keep track of whether the core has been grouped (1 for grouped and 0 for ungrouped). Initialize the visited flag for each core as 0.

(c) For each blank location B_j, $\forall$j(1,m) create an empty set g_j. Each g_j will have a T_{g_j} representing maximum test time of the group. Initialize each T_{g_j} equal to 0.

(d) Let G be a set of g_j,$\forall$j(1,m)

Step 2 : Sort the set of blank locations B such that the blanks at the center positions of the NoC appear first in the sorted set followed by the blanks at the side positions and then the blanks at the corner positions. For two or more blanks at the same position, sorting is done based on increasing order of row number (x_{B_j}) followed by increasing order of column number (y_{B_j}).

/* Sorting condition:

Center : (x_{B_j} = 2,....r-1) $\wedge$ (y_{B_j} = 2,.....cl-1)

Side : $\{x_{B_j}$ = (1$\vee$r)$\wedge$ y_{B_j} = (2,.....cl-1)$\}\vee$ $\{(x_{B_j}$ = (2,....r-1) $\wedge$ (y_{B_j} = (1$\vee$cl-1)) $\}$

Corner :$\{x_{B_j}$ = (1$\vee$r)$\}$ $\wedge$ $\{$ y_{B_j} =(1$\vee$cl))$\}$ */

Step 3 : Satisfying the distance constraint

(a)Assume a BIST controller is placed at B_j. Initialize the value of j to 1.

(b) Check whether set C = $\emptyset$. If yes, then goto Step 5

(c) For each core $C_i \in$ C,

if (dist(C_i,B_j) $\leq$ h) and visited(C_i)= 0 **then**

(i) Include C_i in the set g_j : g_j = $g_j \cup C_i$

(ii) Update G with new g_j : G = G $\cup$ g_j

(iii) Mark the core C_i as visited : visited(C_i)= 1

(iv) Check whether the test time of the core t_{C_i} is greater than the maximum test time of the group T_{g_j}. If yes, then replace T_{g_j} with t_{C_i}. This ensures that the test time of the group is same as the test time of the core which has maximum test time in the group.

(v) Drop the core C_i from the set C : C = C$\setminus \{C_i\}$

end if

(d) Repeat steps (b) and (c) for the rest of the blank locations $B_j \in B$,$\forall$j(2,m)

Step 4: Grouping the memory cores which could not be grouped for h=1

while (C $\neq \emptyset$) **do**

Repeat Step 3 with h incremented by 1

end while

Step 5 : Satisfying the timing precedence relation and the resulting grouping result

(a) Drop empty groups from G

(b) Sort G in decreasing order of T_{g_j}

(c) return (G)

added the data ourselves. We assume all cores in the benchmark are memory cores similar to [104] and use the same data for d695 as used in [74] and [104]. The test data used for d695 circuit is shown in Table 4.2.

4.4.1 Experimental setup

Table 4.2: Data for d695 benchmark circuit

Core	time (units)	power (units)
1	38	660
2	1029	602
3	2507	823
4	5829	275
5	12192	690
6	11978	354
7	4219	530
8	4605	753
9	1659	641
10	7586	1144

The first step of the experiment was mapping the d695 benchmark to a NoC. We followed the same mapping as done in [24]. The implementation involved mapping the 10 cores of d695 to 4x3 size mesh type NoC. Thus, two of the 12 routers are not connected to any core. The mapping has been modified by connecting two BIST controllers to the unconnected routers and each of the ten cores is assumed to be a memory core. The modified implementation of the d695 bechmark is shown in Figure 4.6. The proposed test scheduling algorithm was written in C and was run on a PC with 2.0 GHz processor and 2GB RAM. The input to the algorithm was the test time and test power values of d695 circuit given in Table 4.2 and the locations of the cores and BIST controllers. In the first experiment we assumed dedicated BIST for each core. Thus there was no requirement of the BIST controllers and the mapping was same as in [24]. However, we used another nine different mappings and for each mapping we calculated the total power dissipation for a (*rw*) March operation. In the next experiment, test scheduling algorithm was applied to the modified d695 implementation as shown in Figure 4.6 and was repeated for another 9 different mappings.

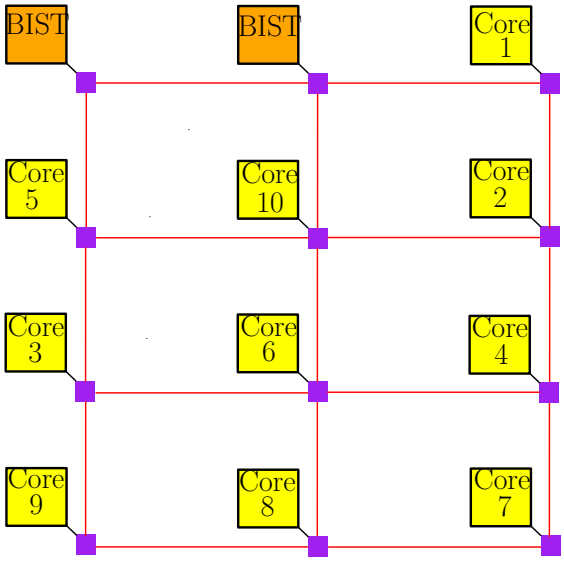


Figure 4.6: Modified mapping of the cores of System d695 to 4x3 size mesh type NoC assuming all memory cores

4.4.2 Results for the d695 benchmark circuit

Area estimate

The BIST circuit was coded in Verilog and synthesized using Faraday 180nm library. The area estimate of the individual components are given in Table 4.3. Table 4.4 compares the area overhead of the proposed approach compared to the other approaches. The table 4.4 shows that the proposed test architecture is far better than the dedicated (parallel) BIST approach and also better than one proposed by Liu [59] where they assumed BISTed cores. However, the proposed approach falls behind Liu's technique when non-BISTed cores are assumed. The reason being Liu's approach requires one controller whereas the proposed approach requires two controllers in the case of d695 circuit mapping.

Table 4.3: Area estimate of the BIST controller synthesized in 180nm library

Component	**Area** (μm^2)
Controller	95
Test Pattern Generator	607

Table 4.4: Comparison of the BIST area overhead

BIST Technique	Area overhead (μm^2)
Dedicated BIST	7030
Li etal. [59] (BISTed cores)	6070
Li etal. [59](non-BISTed cores)	703
Proposed	1406

Power variation

Figure 4.7 illustrates the read and write power variations for the parallel BIST scheme (each core with dedicated BIST) for 10 different mappings of the cores of d695 to 4x3 mesh type NoC during the (*rw*) March operation. During the (*rw*) operation, simultaneous read is performed on all the 10 cores followed by simultaneous write on all 10 cores. Figure 4.7 shows that during

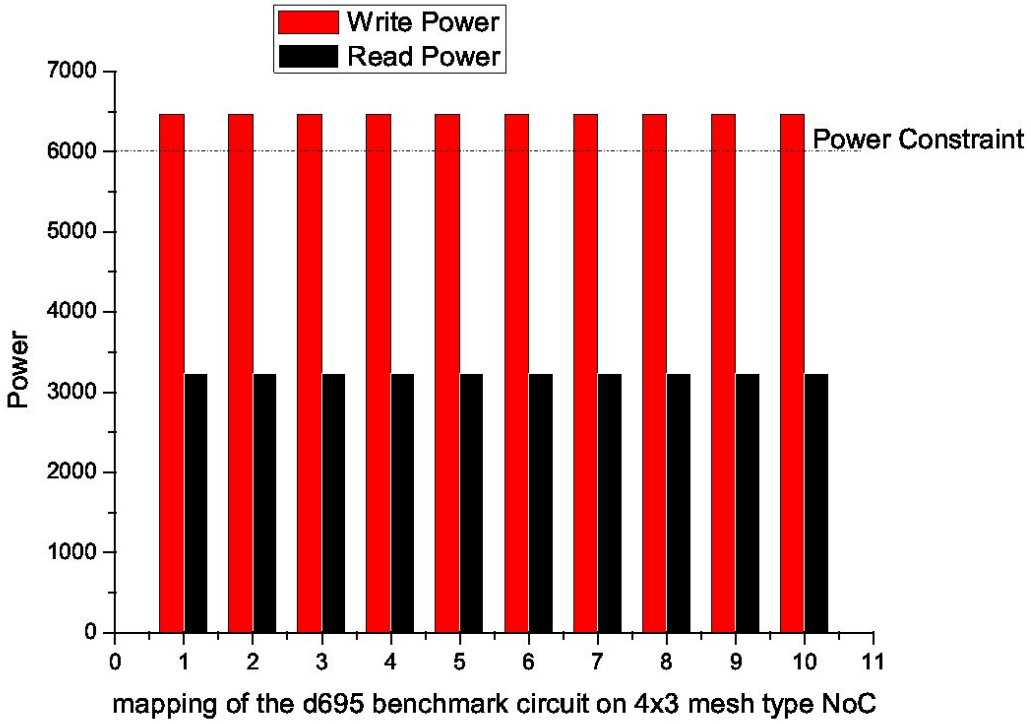


Figure 4.7: Test Power variation during (*rw*) March operation on ITC'02 benchmark circuit.

simultaneous reads ('*r*' operation) on all 10 cores the total read power remains within the power

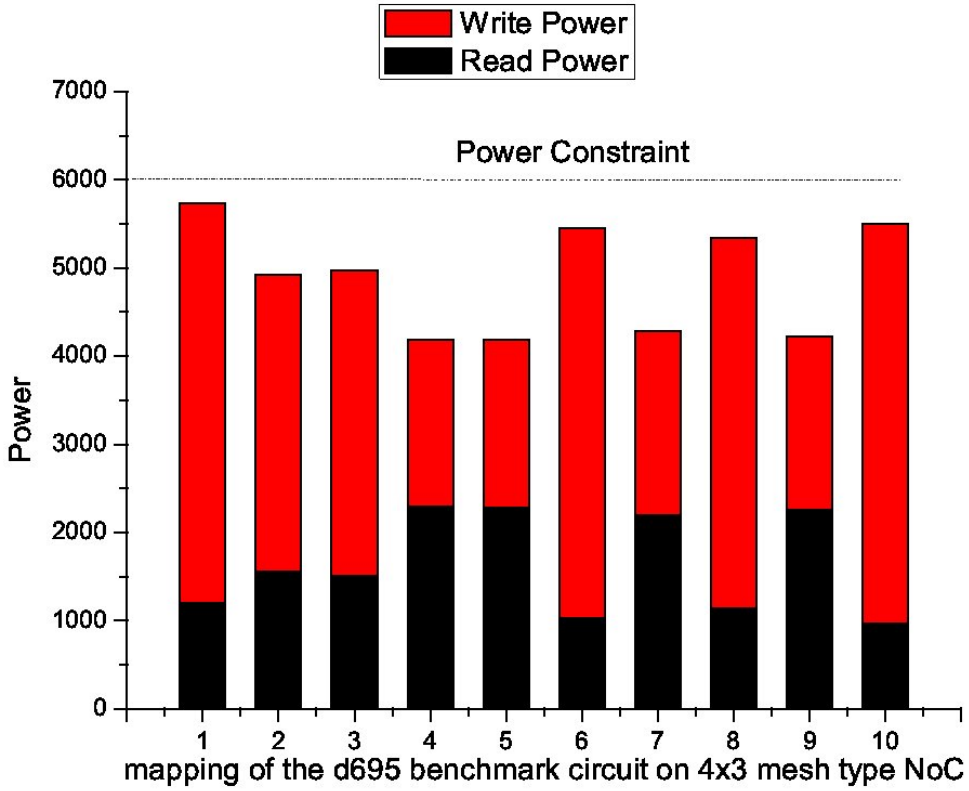


Figure 4.8: Test Power variation during (rw) March operation on ITC'02 benchmark circuit.

budget. However, during the ('*w*') operation the total write power for all the cores exceeds the power constraint.

Figure 4.8 shows the power variation for the proposed scheduling technique with two BIST controllers for 10 different mappings of the cores to the NoC. In the graph, it shows that for each mapping of the cores the total power dissipation for (*rw*) operation is within the power constraint. The justification for such power constrained test results is explained using the mapping of d695 circuit to a 4x3 mesh type NoC as shown in Figure 4.6. According to the proposed test scheduling algorithm, for the mapping as shown in Figure 4.6, two groups of memories are formed. Memory cores 1,3,5,9 form one group and the rest the other group. According to the data in Table 4.2, core 5 has the highest test time, and thus BIST controller of the group containing core 5 is the one which receives the March element level instructions first. Thus, during the (*rw*) operation, when the group with core 5 performs read operation, the other group performs

write operation. Thus at any instant of time, the total power dissipation will be :

$$P_d = \sum_{i=3,5,9} P_r(i) + \sum_{j=1,2,4,6,7,8,10} P_w(j) \tag{4.8}$$

where P_d : total power dissipation, $P_r(i)$: read power for core *i*, and $P_w(j)$: write power for core *j*. We assume that the power values of the cores given in Table 4.2 are the power values during write operation to the core and that read power for each core is approximately half of write power [33]. Thus, power dissipation P_d, calculated using the read and write power values remains within power budget. As shown in Figure 4.8, for the other mappings also P_d remains below power constraint.

Test time estimate

We provide a time estimate of test for the mapping of Figure 4.6 and use the data of Table 4.2. We assume a March test with two March elements. According to the proposed test scheduling algorithm, the Memory cores 1,3,5,9 form one group (say group 1). The rest of the cores form the other group (say group 2). The test time of group 1 is given by core 5 (core having the largest test time in the group) while for group 2 the test time is the same as test time for core 6. Thus, the total test time is calculated as
T = T_{g1} + T_{g2} = 12192 + 11978 = 24170.
For the parallel BIST scheme, as all cores are tested in parallel, the total test time is given by the core with largest test time. For the mapping of Figure 4.6, test time for parallel BIST scheme is same as test time of core 5 and is equal to 11292. Thus, compared to parallel BIST scheme, our proposed test technique requires additional test time equal to the test time of group 2.

4.5 Summary

In this chapter, a distributed BIST based test architecture has been proposed for memories interconnected using NoC that utilizes a hybrid test technique. Memory cores were grouped in clusters based on a heuristic which ensured that the test time and power dissipation was minimum during test. Experiments were performed on ITC'02 benchmark circuit where it was shown that the BIST area overhead for the proposed architecture was much less compared to other reported techniques. Experiments on the benchmark circuit for the proposed test schedule have shown that the proposed test schedule performs a more power constrained test than the dedicated BIST technique. The second direction to achieve the research objective has been utilization of on-chip resources for test purpose. This approach has been targeted with the aim to overcome

the additional area overhead of the DFT hardware. Both off-line and on-line test techniques have been proposed for test of DRAMs which utilize the refresh circuit for test purpose in Chapter 5 and Chapter 6 respectively.

Chapter 5

Re-using Refresh for Off-line Test of DRAMs

Dynamic Random Access Memories (DRAMs) are preferred over Static Random Access Memories (SRAMs) in most computer systems as DRAMs provide a reliable and low-cost memory solution. DRAMs require only one transistor and an intrinsic capacitor to store each data bit compared to six transistors required in SRAMs. Due to this structural simplicity, DRAMs achieve very high density resulting in large capacities. For small form factor systems requiring large amounts of memory such as gaming consoles or mobile phones, embedded DRAMs are preferred to embedded SRAMs. However, high packaging density make DRAMs more susceptible to faults than SRAMs. The faults which cause DRAM failures can be classified as *Permanent faults* (corrupt bits in a persistent manner due to a physical defect such as stuck-at-faults), *Transient faults* (occur randomly and without significant physical damage) and *Intermittent faults* (caused by non-environmental conditions such as loose connection, aging components) [83].

As density is increasing by four times for every new generation, test cost of memory is increasing proportionately. Built-In Self-Test (BIST) has emerged as an essential and necessary technology which can enable low- cost manufacturing test for eDRAMs. However, for Systems-on-Chip (SoCs) with hundreds of eDRAMs, dedicated BIST for each DRAM leads to high routing and gate area overhead. To reduce the BIST area and routing overhead, distributed approaches for testing memory cores are necessary. In distributed approach of test, memory cores are placed in different groups and each group is tested by a dedicated BIST circuit. All the cores in a group share the BIST circuit, and thus avoid the requirement of dedicated BIST wrappers which in turn brings down the area ovehead. However, the number of memories that can share the BIST circuit is limited by the maximum power dissipation of the chip.

5.1 Motivation

Contemporary NoC based SoCs employ a number of DRAMs on-chip. The overall real estate to be devoted to the BIST circuitry is significant. There is, therefore, a major incentive to reduce this area overhead which is the focus of the work described in this chapter. Refresh operations are periodic and require reading the contents of a memory location and writing them back to the same location. March test [13], the most popular tests for detecting functional faults in memories, also require writing some patterns in-to the memory and reading them back. Thus, there is a similarity in the operations performed on the memory during both refresh and March test. The manner in which the operations need to be performed are also similar. Both require scanning the entire memory row by row and performing read followed by write operation on each row. These similarities in refresh and March test has been the motivation for re-use of the refresh circuit for test purpose.

In this chapter, a Built-In-Self test technique has been proposed that utilizes refresh circuit to perform functional tests on DRAMs. A two-pronged approach has been adopted in this work. First, the refresh circuit of the DRAM is leveraged to participate in the BIST. Next, the BIST specific circuitry is shared between neighboring memories. The refresh re-use technique overcomes the requirement of additional Design-For-Testability hardware as tests are performed via the on-chip refresh circuit. Moreover, to perform test read followed by test write operations on a DRAM, each read operation gets completed within the refresh operation of the DRAM, avoiding separate test read cycles. As a result, the entire time between two refresh cycles is allowed for write operation. This increase in write cycle time is utilized in performing power aware test of a number of DRAM cores embedded in NoC based SoCs.

5.2 Fault Models and Test Algorithm

An industrial test set for DRAMs functional test requires a series of different test algorithms to ensure its complete functionality and coverage [91]. In [53], the authors mention that not all faults in SRAM apply to DRAM. For example, stuck-open faults in SRAM behave as stuck-at-faults in DRAM as the cells are not implemented as bistable elements. Moreover, the data retention faults do not occur in DRAM due to the absence of pull-up devices. The faults in SRAMs that also apply to DRAMs are stuck-at-faults (SAF), Transition faults (TF) coupling faults (CF) and address decoder faults (AF).

To consider the test budget a simple fault model has been considered involving cell array faults and leaving out address decoder faults. Thus, the faults considered have been stuck-at faults, transition faults and coupling faults. Neighborhood pattern sensitive faults have not been

considered due to the complexity and vast number of ways in which the fault can occur as mentioned in [44]. It has been mentioned in [13] and [89] that March test has been the most preferred test algorithm for detecting the above mentioned faults and hence the proposed BIST architecture has been designed to support the March based tests. Since, the focus has been to re-use the refresh circuit for test purpose, and refreshing involves reading a complete row of a memory array, word-oriented March tests [89] have been used instead of standard March tests. However, March test for word-oriented memories leads to the problem in detecting Coupling Faults. Dekker et al. in [26] and Goor et al. in [88] have proposed solutions to overcome this problem by repeating the test using ($\lceil \lg_2 w \rceil$+1) different data backgrounds, where w represents the word-width of the memory under test. Therefore, the proposed BIST architecture of this work has been designed to support word-oriented March tests with different data backgrounds.

In the next section, the proposed technique has been explained using the word-oriented MATS+ test. However, other March tests could have well be used for the same. The MATS+ test has been chosen due to the smaller number of March elements involved in it. As a result, it is easier to explain its operation on the proposed technique. Moreover, the test covers all the cell array faults considered in the work. The test is represented as :

$$\{\Uparrow (wa); \Uparrow (ra, wb); \Downarrow (rb, wa)\}$$

where, *a* is the data background and *b* is the complement of the data background. The word-oriented MATS+ test has been formed from the bit-oriented MATS+ test using the systematic technique described in [89].

5.3 Refresh re-use based test technique

Rows are repeatedly refreshed one after another in DRAMs. As already mentioned in Chapter 2, the refresh of a row involves reading contents of all cells of the row and writing them back. Read and write operations are synchronized with the refresh operation of that row. In the work described in this chapter, the reading operation during refresh have been utilized to double as the read operation of the test cycle, thus avoiding separate read cycles for performing the March test. The MATS+ instruction for word-oriented memory is a sequence of three March elements M1, M2 and M3 to be performed on each row using a suitable data background. M1 is a write operation (wa). M2 is a read followed by a write (ra wb) and M3 is write followed by a read (ra wa).

Fig. 5.4 shows the interleaved refresh and test cycles. The cycles shown without shade in Fig. 5.4 are the Refresh Cycles (Read Cycles) while the cycles shown in shade are the Write

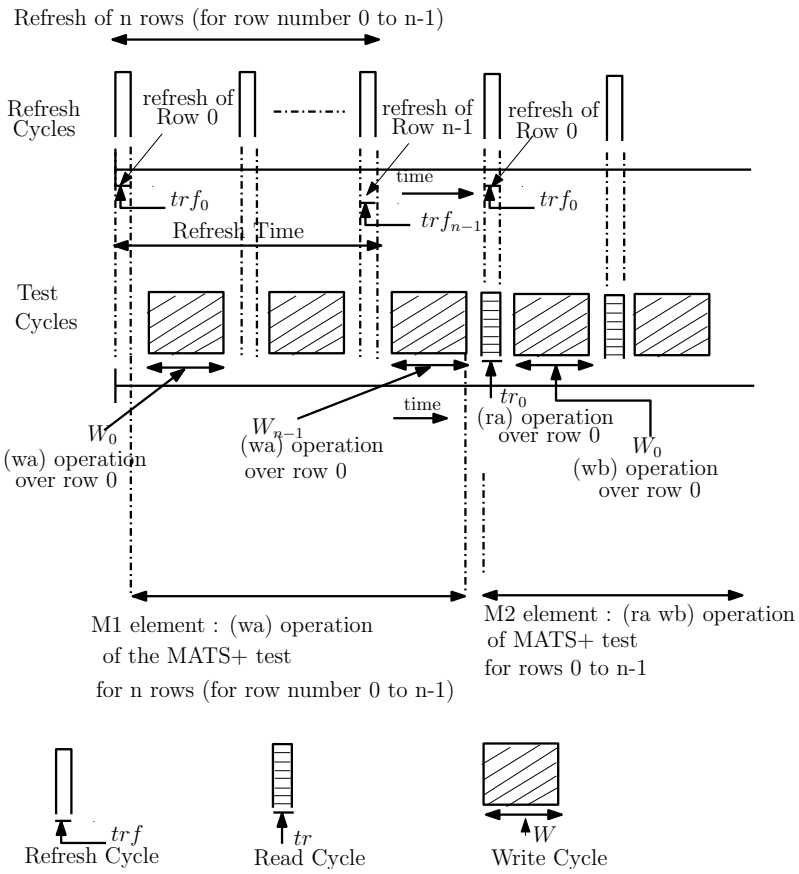


Figure 5.1: Interleaving of DRAM Refresh and Test Cycles

cycles. A Write cycle exists between two Refresh Cycles (Read Cycles). We assume the DRAM to be tested has *n* rows. Thus, refreshing the DRAM requires *n* number of refresh cycles within the Refresh time of T_n. The *n* number of refresh cycles are numbered as trf_0 to trf_{n-1}. These *n* cycles repeat after every T_n time. The Write cycles are also numbered as W_0 to W_{n-1} representing write operation on Row number 0 through Row number *n-1*. During the first *n* cycles, data read from each row during each refresh cycle is loaded in the refreshment register but instead of writing the same data in the row, the test data required for M1 element of the MATS+ test, residing in a data register is loaded (*wa* in this case, *a* being the data background). Thus, the M1 element of the MATS+ test gets performed in the first *n* cycles.

In the next *n* cycles, when a refresh is performed, the data read from each row is loaded in the refreshment register. After the data gets loaded in the refreshment register, its contents are compared with the contents of a data register. The data register holds the expected data from the previous write operation (during M1 operation). Any difference of result detects a stuck-at fault. However, to detect all stuck-at faults and transition faults a repeat of the M2 operation needs

to be performed with different data background. During M2 operation, the data that is written back is the complement of the data that was read (complement of the data register). Thus, the successive refresh and write operations performed completes the required (*ra wb*) operation of the M2 element of the MATS+ test.

The M3 operation, performed in another *n* refresh cycles, is a repeat of the M2 operation with a data background which is the complement of the data register used during M2 operation. Thus, at the end of the 3*n* refresh cycles, each row of the DRAM has been successively tested by writing a pattern, reading it, writing the complement and reading it once again. These four operations on each row guarantees detection of all stuck-at and transition faults.

To detect coupling faults, in addition to M1, M2 and M3, the M2 and M3 cycles are repeated in reverse address order of refresh (similar to the March C- test [13]). Thus, an additional 2*n* refresh cycles are required to perform the test of coupling faults. Reversing the addressing sequence for refresh of rows can easily be handled by making a minor change in the memory controller hardware.

5.4 Proposed BIST Architecture

Figure 5.2 illustrates the proposed BIST architecture based on the refresh-reuse technique. It includes the following modules:

1. Memory Controller : Generates control signals for memory read and write operations as well as control for periodic refresh operations. The proposed architecture requires modifying the normal DRAM controller (allow to have variable refresh rate, normal and reverse addressing sequence)

2. BIST controller : A Finite State Machine (FSM) which receives test instruction and accordingly initiates the test process.

3. Glue Logic: It includes storage registers (data register), combinational logic (gates, multiplexers, comparator).

The operations performed by the BIST hardware during refresh and test cycles are as follows.

1. a) *Refresh Cycle* : In this cycle, the contents of the row addressed by the ref_{add} is loaded in the Refreshment register and then written back to the same address. The ref_{add} is generated by the *Refresh Address Generator* as shown in Figure 5.2. The refresh circuit in

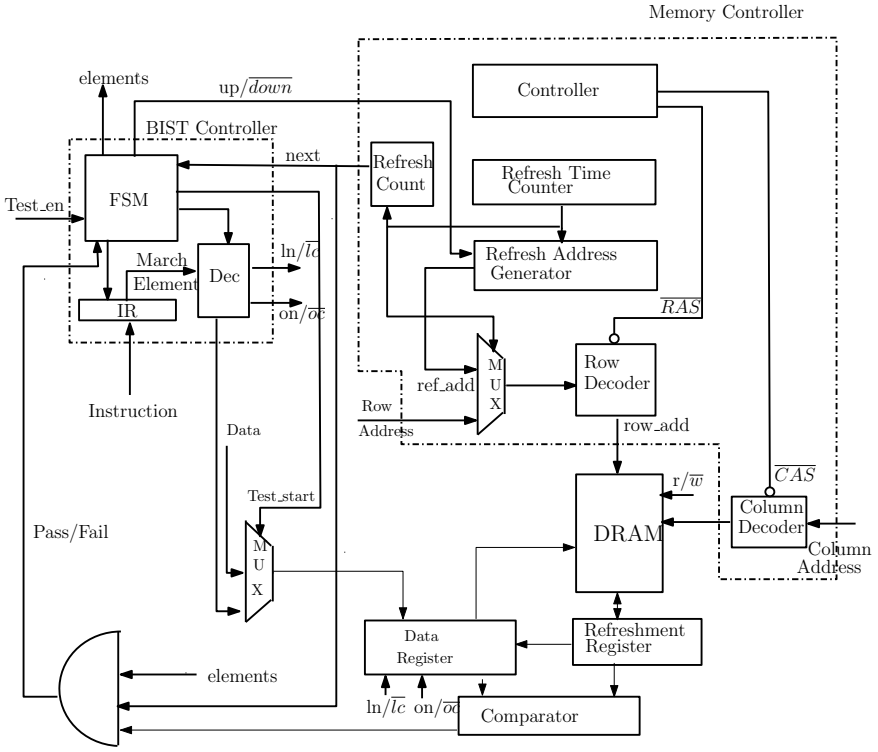


Figure 5.2: The Proposed BIST Architecture

the memory controller consists of *Refresh Time Counter (RTC)* , *Refresh Counter (RC)* and *Refresh Address Generator (RAG)*. The *RTC* generates a tick called ref_{req} to mark the start of the Refresh cycle. *RTC* is initialized with a count corresponding to the time between two refresh cycles. *RC* is a self decrementing counter which is initialized to number of refresh cycles that occur within the Refresh time or in other words to the number of rows. On reaching zero, *RC* wraps around to mark the completion of refresh of the all rows once. On receiving ref_{req}, *RC* decrements itself by one count and wraps around once the count reaches zero.

2. b) *Test Cycle :* March Read/Write operations are performed on the row addressed by ref_{add}. The test cycle involves the following sequence of steps:

   (a) The Test cycle is initiated by the $Test_{en}$ signal from the external environment.

   (b) The encoded MATS+ instruction is loaded in the *Instruction Register (IR)* of the BIST controller which is passed on to the Decoder.

   (c) The Decoder scans the March test instruction and finds the number of constituent March elements in it and passes this information to the Finite State Machine (FSM).

The FSM maintains a count of the number of March elements in the March instruction.

(d) The Decoder then loads the data background for the first March elements in the Data Register on the assertion of the $Test_{start}$ signal by the FSM. For each March element that is being performed on the DRAM, the FSM decrements the element count by one and when all elements have been tested it makes the element line high.

(e) March elements involving interleaved DRAM refresh and write operations are performed. To perform the required operations, during the refresh cycle the *Refresh Address Generator* generates refresh address ref_{add}. The row decoder decodes the address and refresh (read) is done on the row addressed by ref_{add}. Following the refresh cycle, the data from the data register is written to the row addressed by ref_{add}. Since the Row decoder holds the address until the beginning of the next refresh cycle, the data written during the write cycle is written to the same location from where the data is read. On completion of the M1 element, the Memory controller sets a high on the next line to request BIST controller to send the next March element.

5.5 Experimental results for commodity DRAM

The proposed BIST architecture implementation was described in Verilog. Then it was synthesized on a commercial 90nm standard cell library and the area of the synthesized BIST architecture was estimated. The DRAM considered was of size 4M$\times$1 with refresh time of 16ms. The considered DRAM requires 1024 refresh cycles to refresh all rows within the refresh time at a rate of 15.6μs and at a refresh cycle time of 130ns [2].

5.5.1 Area estimation

The Verilog description of the comprehensive BIST architecture consists of three main blocks. The memory controller including the refresh circuit, the BIST controller and the combinational logic circuits. The BIST controller consists of the March decoder, the IR and Finite State Machine based controller. The FSM based BIST controller was implemented similar to the one proposed in [14]. The logic circuits block includes the Refreshment register, the Data register, the comparator and the AND gate. Table 5.1 reports the area occupancy of the blocks in the proposed BIST module.

Evaluation :

Memory BIST (MBIST) executing a March element requires an address generator to generate

Table 5.1: Area estimate of the proposed BIST architecture

Blocks	**Number of gates**	**Area (μm^2)**
Memory controller	266	2661
BIST Controller	114	703
Registers and Logic circuits	71	984

the required address sequence. Researchers have proposed different implementations for the address generators; up/down counter, gray code based address generator [100], Linear Feedback Shift Register (LFSR)([42], [85] and [101]). The proposed BIST architecture presented in this chapter does not require any address generator circuit and hence scores over all the three techniques of address generation found in literature in terms of area overhead as shown in the Table 5.2 for a DRAM of size 4M×1. The %Overhead mentioned in Table 5.2 is the % area overhead in using each of the address generation techniques and is calculated as follows:

$$\%\text{Overhead} = \frac{\text{Area of the block used as Address generator}}{\text{Area of the proposed BIST architecture}} \tag{5.1}$$

Each of the Address generator blocks used in techniques listed in Table 5.2 have also been implemented in Verilog and synthesized using the same 90nm standard cell library. In fact, Table 5.2 can also be viewed as % Improvement in using our proposed BIST technique compared to the other address generation techniques.

Table 5.2: Area overhead of existing approaches with respect to the proposed BIST technique

Address generation technique	**%Overhead**
LFSR	7.00%
Binary Counter	9.61 %
Gray code counter	23.24%

5.5.2 Test time analysis

The DRAM considered for analysis is of size 4M×1 with a distributed technique of refreshment. The number of cycles to be refreshed within the refresh time is 1024. Using the proposed refresh re-use technique, it requires 5*1024 = refresh cycles to complete the MATS+ test for detecting the cell array faults (SAF,TF and CF). With refresh rate of 15.6μs and at a refresh cycle time of

130ns [2], the total time of the MATS+ test performed on the DRAM is approximately 80ms. The high test time for this method is due to dependence of the test on the refresh time. This increase in test time is a problem when considering an isolated commodity DRAM. However, when performing test over a number of eDRAMS, the increase in test time can be effectively utilized to schedule the test of groups of eDRAMs so that power dissipation during parallel test of a number of eDRAMs remains within the power budget. The extension of the refresh re-use technique for testing e-DRAMs is discussed next.

5.6 Refresh re-use technique for e-DRAMs interconnected using the NoC infrastructure

Figure 5.3 shows extension of the refresh re-use technique in performing tests on a number of DRAMs interconnected by the NoC communication infrastructure. As shown in Figure 5.3, the DRAM cores are grouped to form clusters. The memory grouping criteria and memory grouping algorithm are those that have been discussed in Chapter 4. The e-DRAMs of a cluster share a BIST controller (*BIST CTR*) as shown in Cluster 1 of Figure 5.3. [1] The architecture of the BIST controller is as shown in Figure 5.2. The *Mem Ctr* module associated with each e-DRAM is the *Memory Controller* responsible for performing the read and write operations of the e-DRAM.

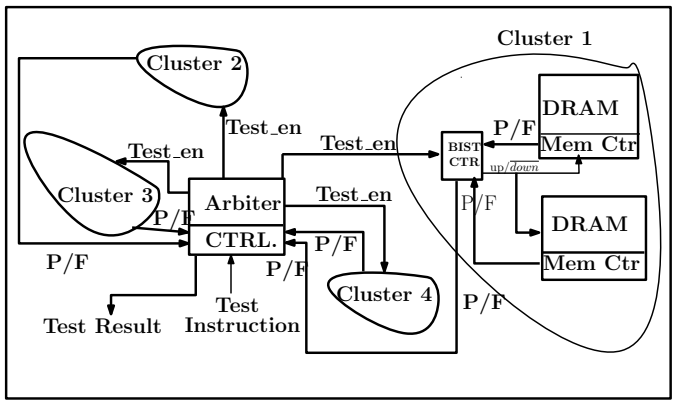


Figure 5.3: Refresh reuse technique for eDRAMs

The March test instruction is received by a top level controller (*CTRL*). To perform a read followed by a write instruction, *CTRL* enables *Test_en* signal of all clusters during read. However, during write, based on a test schedule, the arbiter enables the *Test_en* signal of any one cluster and the test instruction is transferred from the top-level controller to the BIST controller

[1] For clarity the grouping has been shown only for Cluster 1 and the other clusters follow the same.

of the cluster. [2] Once the BIST controller receives the instruction, it initiates the test process by activating the refresh circuit of the Memory controllers for each DRAM. On completion of the test, the DRAMs report the result of the test, *Pass* or *Fail* (*P/F*) to the BIST controller. The BIST controller then passes on the information to *CTRL* which delivers the result through the *Test Result* output.

5.6.1 Impact of refresh re-use on test of eDRAMs

Overall chip area overhead reduction

The experimental results shown in the last section reveal real estate benefits of using the refresh re-use on commodity DRAMs. For single commodity DRAM, the area overhead of few Kilo gates may not be an issue. However, for NoCs (where the communication infrastructure already brings about an area overhead) with a number of e-DRAMs, avoiding address generators for each e-DRAM will reduce the area of the BIST wrapper associated with each DRAM and hence the area overhead of the entire chip gets reduced substantially.

Power aware parallel test of e-DRAMs

The refresh time dependence of the refresh re-use technique is effectively utilized to facilitate parallel test of number of eDRAMs without exceeding the power budget.

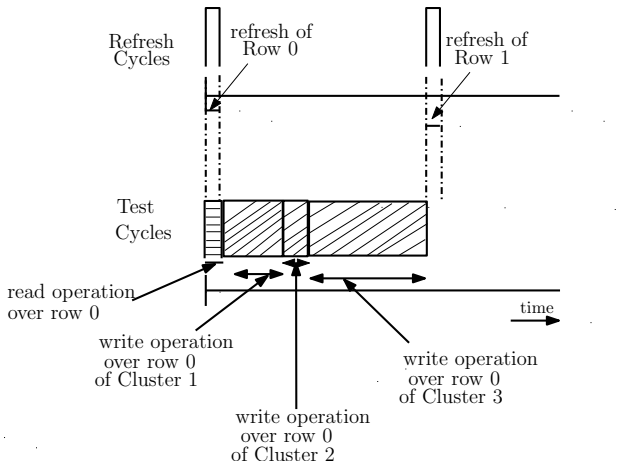


Figure 5.4: Scheduling the write operation over Row 0 for different clusters of eDRAMs

Figure 5.4 illustrates the write operation over Row 0 for the clusters of eDRAMs. For simplifying the explanation, the following assumptions are made. There are 10 eDRAMs that

[2] For clarity of the figure this interface has not been shown

need to be tested (a read followed by write is performed), the refresh cycle (read cycle) time is 50ns, the time between two refresh cycles (refresh rate) is 300ns, each core dissipates 50μw during write, and the maximum allowed peak power dissipation during write is 250μw.

If no refresh reuse is used, the refresh rate of 300ns is utilized by a read followed by a write. The read cycle takes up 50ns leaving the write to be performed on all DRAMs within 250ns. Suppose a test schedule forms two clusters of eDRAMs with five cores in each cluster. Cluster 1 has maximum write time of 125ns and write power of 250μw. Similarly cluster 2 also has maximum write time of 125ns and write power of 250μw. Thus, write of all DRAMs are completed by 250ns at a maximum peak power dissipation of 250μw,

However, if the refresh re-use technique is used on the DRAMs, then read cycle is performed during refresh and the entire 300ns between two refresh is utilized for write operation. Since an additional 50ns is available for write, the memory groupings in the former case is modified. eDRAMs with minimum test time within each cluster are moved from the clusters and grouped to form a separate cluster 3 of two cores. Removing the core with minimum test time does not affect the test time of each cluster but reduces the maximum power dissipation of each group by 50μw. In such a scenario, the test schedule forms three clusters with four cores in cluster 1, four cores in cluster 2 and two cores in cluster 3. Cluster 1 requires write time of 125ns with maximum power dissipation of 200μw, cluster 2 also requires write time of 125ns with maximum power dissipation of 200μw and cluster 3 requires write time of 50ns with maximum power dissipation of 200μw. Thus, write of all DRAMs are completed by 300ns at a maximum peak power dissipation of 200μw, a 20% improvement compared to the case when no refresh re-use technique was used.

5.7 Summary

One of the research direction of the thesis has been effective utilization of on-chip resources for test purpose. To this effect, the refresh circuit of DRAMs have been used for testing of DRAMs. This chapter presented an offline BIST based technique for testing DRAMs interconnected using NoCs. The proposed technique has been initially developed for commodity DRAMs and then extended to e-DRAMs connected using NoC. The proposed BIST architecture avoids use of address generators, typical of MBIST architecture. Experimental results show that about 7% area overhead can be reduced using the proposed technique compared to MBIST architectures which use LFSR based address generators. At minimum area overhead, the proposed MBIST technique for a single DRAM when extended for test of number of eDRAMs, allows larger test write cycle time which allows more clusters of eDRAMs to be accommodated. Distributing the

write cycle time over large number of clusters reduces the peak power dissipation during test.

The off-line technique proposed in this chapter reduces area overhead. However, the proposed technique is suitable for detection of manufacturing faults and cannot be utilized for run-time fault detection. To target detection of faults developed during in-field operation of DRAMs, the detection technique must involve an on-line test mechanism. One such on-line BIST technique has been discussed in the next chapter where the refresh circuit of the DRAMs have been used to perform the on-line test of DRAMs.

Chapter 6

Refresh Re-Use for Online test of DRAMs

Since last two decades, a lot of research has been devoted in devising efficient fault detection techniques for DRAMs. The fault detection techniques are broadly classified into two categories. The first category targets *permanent* faults developed during manufacturing process while the second category targets occasional run-time faults in DRAMs which are either *transient* (affecting different location) or *intermittent* faults (affecting the same location) in nature. The detection techniques targeting *permanent* faults developed during manufacturing process are hardware based. These techniques are performed either using memory testers built with very complex test algorithms or using on-chip built-in-self test (BIST) circuitry running popular memory test algorithms [13]. The detection techniques for run-time faults in DRAMs are software test techniques involving memory diagnostic software programs used to check for memory failures on a computer such as *ECC* [81], *Chipkill* [27], or *memory scrubbers* [68].

Though run-time faults are either *transient* or *intermittent*, the detection techniques for run-time faults reported in literature mainly focus on detection of *transient* faults such as *soft errors*. However, for deeply scaled CMOS based DRAMs, the occasional run-time faults in DRAMs which are a result of physical effects such as environmental susceptibility, aging and low supply voltage, are *intermittent* faults [11]. These *intermittent* faults usually exhibit a relatively high occurrence rate and eventually tend to become permanent [11]. Moreover, wear out of DRAM can also cause intermittent faults to become frequent enough to be classified as permanent [20]. Thus, there is a need for on-line test technique that can detect the run-time faults which are *intermittent* in nature but gradually become *permanent* over time.

6.1 Motivation for this work

Recent studies of DRAM failures in field ([49], [78] and [83]) provide strong evidence that DRAMs experience both *transient* (*soft*) faults and *permanent* (*hard*) faults in field but *permanent* faults constitute bulk of all DRAM failures. The field study published by Schroeder et al. [78] has been based on Google's server fleet with data collected over a period of 2.5 years while the one published by Sridharan et al. [83] has been based on Jaguar's memory system with data collected over 11 months. The study published in [49] has been based on data collected from four different production systems. The most important conclusion drawn from those studies is the presence of hard faults along with soft errors during field operation of DRAM. Thus, it is essential to develop test techniques that can detect these hard faults. The soft error data detection and protection schemes used by system designers are not sufficient for testing in-field *permanent* faults in DRAMs due to the following reasons.

- Soft error detection schemes such as *ECC*, *memory scrubbing* or signature based schemes are concerned with the integrity of stored data, without caring about functionality of the memory. They do not perform active test - they do not alter the contents of the memory. Therefore, they cannot find functional faults in memories [87].
- *ECC* systems rely on redundancy and extra computation to detect faults. This extra computation has negative impact on cost, power and performance of the system [105].
- Memories with error detection and data protection schemes are costlier than those without them.

It is therefore necessary to find a cost-effective test technique that can detect hard faults during field operation of DRAM. Such a test technique must possess the following characteristics:

- The test must be an active test so that functional defects are uncovered.
- The test must be performed periodically to ensure that no fault gets accumulated.
- The test hardware must be cost-effective.

To perform an active test on field operative DRAM, March based tests are mostly preferred due to their high fault coverage [13]. Performing a March test involves writing pre-defined test patterns into the DRAM and reading the same patterns. However, writing test patterns into the DRAM means the initial contents of the DRAM are lost after test. Such a technique is allowed during manufacturing test but cannot be afforded for in-field operation where the normal operation of DRAM resumes after test. Thus, we prefer Transparent March tests [102] in place of standard

March tests. Transparent testing is a technique where the original contents of the memory remain unchanged after test.

However, Transparent testing also has a limitation. Transparent tests are interrupted by normal operation leading to increase in test time. One way to overcome this problem is to allow un-interrupted Transparent testing. To achieve this goal, we were motivated to utilize the DRAM refresh in performing the Transparent March tests on DRAM because refresh is an un-interrupted process and if tests are performed during refresh then tests are performed un-interrupted.

Refresh operations require reading the contents of a memory location and writing them back to the same location. March tests for detecting functional faults in memories also require writing some patterns in to the memory and reading them back. There is a similarity in the operations performed on the memory during both refresh and word-oriented Transparent March test. The manner in which the operations need to be performed are also similar. Both require scanning the entire memory row by row and performing read followed by write operation on each row. These similarities in refresh and word-oriented Transparent March test further motivated us to re-use the refresh circuit for test purpose. Moreover, DRAM is refreshed periodically. Thus tests performed with refresh will also be periodical and will prevent fault accumulation. Further, utilizing refresh circuit for test purpose overcomes additional DFT overhead due to test circuit.

In this chapter a BIST architecture is being proposed which re-uses the refresh circuit of DRAM in performing periodic Transparent March tests on the DRAMs targeting permanent faults developed during DRAM operation. Re-using refresh allows periodic testing of DRAM without interruption and test finishes within a definite time. Most importantly, reusing the on-chip refresh circuitry for test purpose overcomes additional DFT area overhead due to BIST hardware. The refresh-reuse concept can be extended to even embedded memory cores of SoCs or those which are interconnected using NoCs.

6.2 Fault Models Considered in this Work

The runtime *permanent* faults considered in this work are assumed to be intermittent faults which have become permanent over time. Consequently, the fault models considered in this paper are that of intermittent faults. The factors which lead to intermittent faults are variations of temperature, voltage and aging effects such as Time Dependent Dielectric Breakdown (TDDB), Electro-migration, Negative bias temperature instability (NBI) and hot carrier injection (HCI).

TDDB is a phenomena where the oxide underneath the gate material of a MOSFET degrades over time resulting in a short circuit. As technology scales down, the oxide becomes thinner and more fragile allowing the effects of TDDB to become more severe. TDDB causes

hard gate shorts which are modeled as stuck-at-faults. Electro-migration reduces interconnect conductivity with passage of time and leads to open circuit [31]. The open circuits caused by electro-migration are modeled as stuck-open faults. However, stuck-open faults behave as stuck-at faults in DRAM as cells are not implemented as bistable elements [87]. Negative bias temperature instability (NBI) and hot carrier injection (HCI) [31] increase threshold voltage of transistors leading to decrease in mobility. As a result, performance of the memory core decreases bringing in read and write failures. We model these read and write failures as read disturb fault and write disturb faults respectively. However, read disturb faults are predominant in SRAMs, where a defective SRAM loses its data state on a read and behaves as a dynamic cell. Since DRAMs refresh data after every read operisation, read disturb faults are less likely to occur in DRAMs. Thus, we consider only write disturb faults for our work.

The changes in temperature and voltage affect the overall speed of the memory by modifying the circuit delay and also reduces the robustness [31]. However, in this work, we consider only the functional faults, and hence the effects of temperature and voltage change are not considered. Also, the data retention faults do not occur in DRAM due to the absence of pull-up devices [87].

To summarize, the target fault models considered for this work are stuck-at faults and write disturb fault. Detailed behavior of these faults are as follows.

1. Stuck-at-Fault (SAF) - the defective cell permanently contains a 0 (SA0) or 1 (SA1) and cannot be changed. To detect a stuck-at-fault, a write operation at the cell must be followed by a read operation. For example, to detect SA0, a write 1 must be followed by a read 1. Similarly, to detect SA1, a write 0 must be followed by a read 0.

2. Write-Disturb Fault - if a non transition write operation is performed on the defective cell, it causes a transition in the cell. To detect a write-disturb fault, each cell should be read after a non-transition write. For example, with a cell initially holding a 0 value, if a write 0 is performed, then the cell must be read for a 0 immediately after the read. Similarly, with a cell initially holding a 1 value, if a write 1 is performed, then the cell must be read for a 1 immediately after the read.

6.3 Proposed Transparent Test Generation Technique for DRAMs without ECC

March tests are the most widely accepted tests for detection of permanent faults due to their high fault coverage and linear relation of their test time with respect to the memory size [36]. A March instruction consists of sequence of operations applied to each cell before proceeding to the next

cell. An operation can be reading or writing of 0 or 1. Application of March tests involves writing patterns into the memory and reading them back. As a result, the memory contents are destroyed. However, on-line memory test techniques require restoration of the memory contents after test. Thus, researchers have modified the March tests to Transparent March test so that tests can be performed without the requirement of external data background and the memory contents can be restored after test.

In this section, we describe our proposed technique to convert a word-oriented March test into a Transparent March test so that stuck-at faults and write-disturb faults mentioned in the previous section can be detected. Since we tried to re-use the refresh circuit for test purpose, and refreshing involves reading a complete row of a memory array, we were motivated to use word-oriented March tests and transform them to their respective Transparent versions.

Nicolaidis et al. in [71] first proposed Transparent March test for memories and gave a systematic way to convert a standard March test to Transparent March test. As mentioned in [71], Transparent test consist of two testing phases: *signature prediction phase* in which the golden signature is obtained, and the *fault testing phase* where the fault is activated. Since our proposed test technique involves performing active test, use of signature based scheme is avoided and hence the signature prediction phase in the Transparent test generation is not required.

6.3.1 Transparent March test

The proposed Transparent test generation technique for word-oriented memories is explained using the word-oriented March X test [89] as it covers fault models considered in this chapter. However, other March tests such as MATS+, MATS++, etc. which also cover single cell faults can also be used for the same. The procedure to convert a bit-oriented March X test to word-oriented March X test is assumed to be the one mentioned in [89]. The word-oriented March X test is represented as

$$\{\updownarrow (wa); \uparrow (ra, wb); \downarrow (rb, wa)\}; \updownarrow (ra)\}$$

where, *a* and *b* are the data background and its complement respectively. $\uparrow$ and $\downarrow$ are increasing and decreasing addressing order of memory respectively. $\updownarrow$ means memory addressing can be either increasing or decreasing.

The first element of the test, (*wa*) is the initialization step meant for writing to each row of the memory a finite data background. The next two elements, (*ra,wb*) and (*rb,wa*) perform read and write operations on the known data background. The element (*ra, wb*) reads the data and writes its invert before proceeding to the next row while (*rb,wa*) reads the invert and writes the re-invert on each row. At this point of the test, each bit of each row gets flipped in both directions. The last element (*ra*) reads each row and expects the same data background as the

initial phase. Any mismatch ensures detection of stuck-at-fault at the particular row. The March X test mentioned above is non-Transparent . The modifications made to the test to generate its Transparent version is discussed next.

Since the operations performed by the elements (*ra,wb*) followed by (*rb,wa*) of the March X test involve inversion and re-inversion of data irrespective of the order of scan, restoration of the original data background after the operations is guaranteed. Hence, the test can easily be performed with any data background. Thus, a modification in the test has been proposed where instead of using any specific data background, the operations will be performed on the data already present in each memory location. This makes the test independent of the test data.

To convert the March X test to Transparent March X test, the following steps need to be performed.

- a) Drop the initial element; Since the Transparent test is independent of any background data, no initialization step is required.

- b) Replace the *a* and *b* in the March X test representation (given above) by *x* and $\overline{x}$ respectively, where *x* indicates the data value at a row prior to performing the March test on the row and $\overline{x}$ indicates the complement of *x*.

Thus, the Transparent March X test generated can be represented as :

$$\{\uparrow (rx, w\bar{x}); \downarrow (r\bar{x}, wx); \uparrow (rx)\}$$

where, the read access (*rx*) and ($r\bar{x}$) expects (x) and $\bar{x}$ to be read respectively from the row to be tested while (*wx*) and ($w\bar{x}$) refers to writing x and $\bar{x}$ to the row that is tested.

6.3.2 Modified transparent March test - proposed technique

The Transparent March X (TMX) test generated from the March X test is modified to suite its implementation using the refresh circuit. Following are the modifications made to the TMX test.

Since refresh cycles involve read followed by write operations, the TMX test is structured such that each element consist of a read followed by write operation. Thus, we add a write operation to the last read operation of the TMX test. In TMX test, same operations are performed on each row during execution of a March element. However, in the proposed Modified TMX (MTMX) test, operations performed on the row targeted for test is different from the operations performed on rest of the rows. Moreover, in MTMX test, addresses are scanned in the same order for each test element.

Thus, the steps involved in transforming Transparent March X test (TMX) to Modified Transparent March X (MTMX) test are as follows.

- a) Insert a write operation (*wx*) after the read operation of the last element in the TMX test. On adding the write operation, the last March element also has two operations to perform (read followed by a write).

$$\{\uparrow (rx, w\bar{x}); \downarrow (r\bar{x}, wx); \uparrow (rx, wx)\}$$

- b) (i) Shift the address order sequence (the up / down arrows in the representation) to the right of each element

$$\{(rx, w\bar{x}) \uparrow; (r\bar{x}, wx) \downarrow; (rx, wx) \uparrow\}$$

  (ii) Add (*rx*,*wx*) after the arrow.

$$\{(rx, w\bar{x}) \uparrow (rx, wx); (r\bar{x}, wx) \downarrow (rx, wx); (rx, wx)\} \uparrow (rx, wx)\}$$

  The representation used in (ii) means the following. The test has three March elements M1, M2 and M3. March elements are separated by semicolons. For each March element, the part to the left of address order sequence (up/down arrows) represents operations performed on the first row addressed during the test cycle while the part to the right of the address order sequence represents operations performed on the rest of the rows.

- c) Replace all down order and up down order of address scan to up order of address scan.

The modified Transparent March X Test (MTMX) thus generated is

$$\{(rx, w\bar{x}) \uparrow (rx, wx); (r\bar{x}, wx) \uparrow (rx, wx); (rx, wx)\} \uparrow (rx, wx)\}$$

In the Modified Transparent March X Test, the pair of read / write operations performed during March elements M1, M2 and M3 represent three phases of the test namely *invert* phase, *restore* phase and *refresh* phase as shown in Table 7.1.

The pair (*rx,w*$\bar{x}$) represents the invert phase where the initial contents of the row under test are read and its complement is written back to the same row. The invert phase is followed by restore phase where the inverted contents of the row under test are re-inverted and written back *(r*$\bar{x}$*, wx)*. The pair *(rx,wx)* is the refresh phase where the contents of the row addressed is read

Table 6.1: Phases in MTMX test

Phase	Operation
Invert	*(rx, w$\bar{x}$)*
Restore	*(r$\bar{x}$, wx)*
Refresh	*(rx,wx)*

and the same is written back to the row.

The interesting thing to note is the similarity of the operations performed in each phase; a read followed by a write. While in invert and the restore phase the data which is written back is the invert of the one read, in refresh phase it is the same data for both the read and the write operations.

6.3.3 Modified transparent March X algorithm

The Modified Transparent March X Test Algorithm is described in Algorithm 3. The algorithm requires three inputs : the number of rows of the DRAM (N), address targeted for stuck-at-fault test ($saf\_test\_address$) and address targeted for write-disturb fault test ($wdf\_test\_address$). However, the algorithm considers $wdf\_test\_address$ to be the location following $saf\_test\_address$. Thus, $wdf\_test\_address$ = $saf\_test\_address$ +1. The tests required for Algorithm 3 and their order of execution have been listed in Table 6.2.

Table 6.2: MTMX test for Stuck-at-fault and Write Disturb Fault

Run	Address	Phase	Element	Test	Operation
Run1	$saf\_test\_address$	Invert	M1	T1	*(rx, w$\bar{x}$)*
Run1	$wdf\_test\_address$	Refresh	M1	T3	*(rx,wx)*
Run1	other addresses	Refresh	M1	T3	*(rx,wx)*
Run2	$saf\_test\_address$	Restore	M2	T2	*(r$\bar{x}$, wx)*
Run2	$wdf\_test\_address$	Refresh	M2	T3	*(rx,wx)*
Run2	other addresses	Refresh	M2	T3	*(rx,wx)*
Run3	$saf\_test\_address$	Refresh	M3	T3	*(rx,wx)*

According to Table 6.2, test for stuck-at-fault test for location $saf\_test\_address$ and write-

disturb-fault test for location $wdf\_test\_address$ requires three address runs, 1,2 and 3. In runs 1 and 2, the DRAM is scanned for all rows. In run 3, only the row with address given by $saf\_test\_address$ is operated. Thus, for a DRAM with N rows, the total number of test cycles required to cover run 1,2 and 3 of the MTMX test is $2N + 1$.

In address run 1, the test T1 is applied to $saf\_test\_address$ and T3 is applied to the rest of the locations. T1 excites stuck-at-fault for data in $saf\_test\_address$ (invert phase) while T3 excites write-disturb-fault for location $wdf\_test\_address$. For the rest of the locations, applying T3 means regular refresh operation. During address run 2, the test T2 is applied to $saf\_test\_address$ (restore phase) to restore the original data prior to run 1 and to detect the stuck-at-fault excited during run 1. The write-disturb fault excited at$wdf_test_address$ during run 1 is detected during run 2 by applying of T3 to $wdf\_test\_address$ again. For the rest of the locations of DRAM, T3 is applied during run2. Run 3 is a single address scan where T3 is performed on $saf\_test\_address$.

Algorithm 3 begins with initialization of two variables J and $COUNT$. The variable J holds the current address of the DRAM while $COUNT$ keeps track of the $2N + 1$ test cycles required for performing the MTMX test on the DRAM. $COUNT$ starts with a zero value and J is initialized to the address meant for test of stuck-at-fault ($saf\_test\_address$). Thus, the scan of the memory during test cycles begins with $saf\_test\_address$. The data read from a memory location is first loaded in the ref_reg variable (line5). Depending on whether the data is required for performing a stuck-at fault test or write-disturb fault test, it is either backed-up in $data\_register$ 1 or $data\_register$ 2 (lines 8 and 11).

For the first N cycles of $COUNT$ i.e. $COUNT$ = 0 to N-1 (run1), J is first checked for its match with $saf\_test\_address$. If the two match, it marks the invert phase for $saf\_test\_address$. The data loaded in the refresh register ref_reg is backed up in $data\_register$ 1 and complement of the data is written in J(lines 7-9). If J and $saf\_test\_address$ do not match, J is then checked for match with $wdf\_test\_address$. If J matches to $wdf\_test\_address$, data loaded in the refresh register ref_reg is backed up in $data\_register$ 2 and the same data is written back to location addressed by J (lines 10-12). If J neither matches $saf\_test\_address$ nor $wdf\_test\_address$, the data residing in ref_reg is loaded back to J (line 14).

The next N cycles of $COUNT$, i.e. $COUNT$ = N to $2N - 1$ (run 2) are a repeat of the first N cycles, except that if J = $saf\_test\_address$, then the data read from address J into ref_reg is compared with $data\_register$ 1 (line 17-19). An ex-or operation of both should result in all 1's as the result. Deviation from the all 1 result at any bit will mean a stuck-at fault at that bit. Similarly, if J = $wdf\_test\_address$, then the data read from address J into ref_reg is compared with $data\_register$ 2 (line 20-22). An ex-or operation of both should result in all

Algorithm 3 Modified Transparent March X Algorithm

Input: N = number of rows of DRAM

$saf\_test\_address$ = row to be tested for stuck-at-fault

$wdf\_test\_address$ = row to be tested for write data fault

1: $J \leftarrow saf\_test\_address$
2: $COUNT \leftarrow 0$
3: **while** $COUNT \leq 2N$ **do**
4: $J \leftarrow JMODN$
5: $ref\_reg \leftarrow read(J)$
6: **if** $(0 \leq COUNT < N)$ **then**
7: **if** $(J = saf\_test\_address)$ **then**
8: $data\_reg1 \leftarrow ref\_reg$
9: $write(J, !ref\_reg)$
10: **else if** $(J = wdf\_test\_address)$ **then**
11: $data\_reg2 \leftarrow ref\_reg$
12: $write(J, ref\_reg)$
13: **else**
14: $write(J, ref\_reg)$
15: **end if**
16: **else if** $(N \leq COUNT < 2N)$ **then**
17: **if** $(J = saf\_test\_address)$ **then**
18: $compare(data\_reg1, ref\_reg)$
19: $write(J, !ref\_reg)$
20: **else if** $(J = wdf\_test\_address)$ **then**
21: $compare(data\_reg2, ref\_reg)$
22: $write(J, ref\_reg)$
23: **else**
24: $write(J, ref\_reg)$
25: **end if**
26: **else**
27: compare$(data_reg1, ref_reg)$
28: write(J, ref_reg)
29: **end if**
30: $COUNT \leftarrow COUNT + 1$
31: $J \leftarrow J + 1$
32: **end while**
33: $saf\_test\_address \leftarrow saf\_test\_address + 1$

0's as the result. Deviation from the all 0 result at any bit will mean a write disturb fault at that bit position.

6.3.4 Fault coverage of the proposed MTMX algorithm

The MTMX algorithm is intended for test of stuck-at fault and write disturb fault tests developed during field operation DRAMs. The fault coverage of the algorithm is illustrated in Figures 6.1, 6.2 and 6.3. In all figures, the word size of each row of DRAM is assumed to be of 4 bits. The text in italics against the arrows indicate the operation performed as mentioned in Algorithm 3. The text in normal font indicate the name of the storage as used in Algorithm 3.

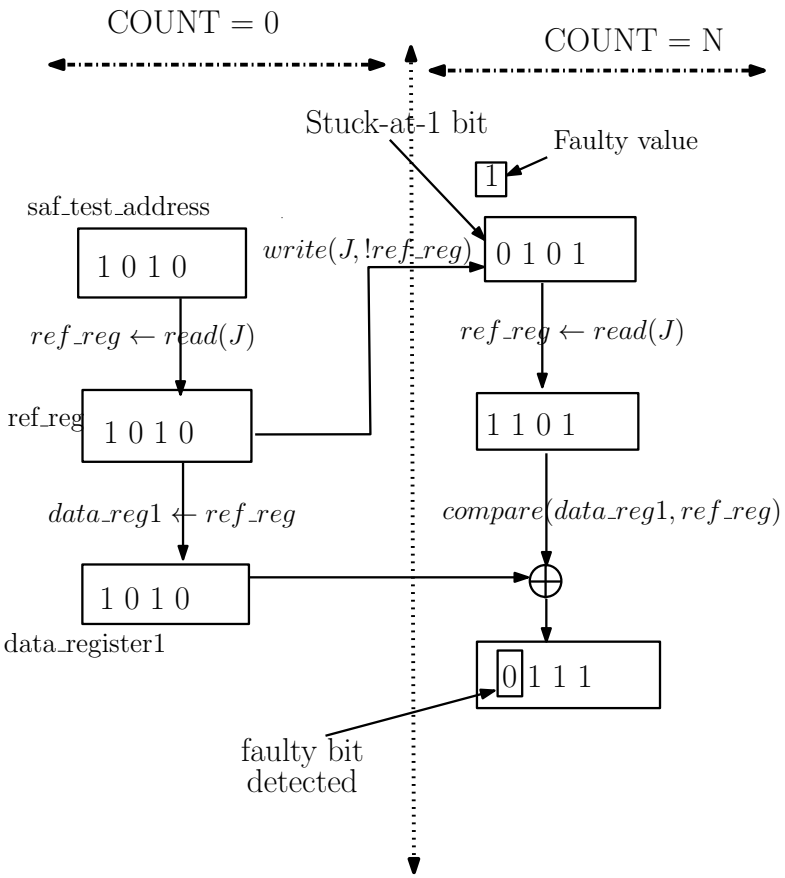


Figure 6.1: Stuck-at-1 Fault Detection using MTMX test

Stuck-at fault detection

As shown in Figure 6.1, assume the data present in the DRAM row addressed by $saf\_test\_address$ be "1010". The test cycles begin with the invert phase ($COUNT$= 0 value) during which the data addressed by $saf\_test\_address$ is read into the ref_reg register and then backed-up in the $data\_register1$. The data written back to $saf\_test\_address$ is the complement of data that is read in the ref_reg. Thus, at the end of the cycle, the data present in ref_reg and $data\_register$ 1 is "1010" while the data in location addressed by $saf\_test\_address$ is "0101". Assume a stuck-at-1 fault at the most significant bit (MSB) position of the word stored in $saf\_test\_address$. Thus, instead of storing "0101", it actually stores "1101" and as a result the stuck-at-fault at the MSB gets excited.

After N cycles ($COUNT$= N), when row $saf\_test\_address$ is re-addressed, the data now read in to ref_reg is "1101". At this point, the data present in ref_reg and $data\_register1$ is bit-wise ex-ored. Since, the data in the $data\_register1$ and the one stored in $saf\_test\_address$ during $COUNT$= 0 were complementary, it is expected that for a fault-free case, ex-oring their contents would result in a all-1 pattern. Any 0 within the pattern would mean a stuck-at fault at that bit position. This situation is illustrated in Figure 6.1, where the ex-or of "1010" and "1101" yields a 0 at the MSB position of the result indicating a stuck-at-fault at the MSB position.

If a stuck-at-fault is not excited during the $COUNT = N$, it is excited during the $COUNT = 2N$ phase as shown in Figure 6.2. It is assumed that the target memory location has a stuck-at-0 fault at the MSB position. During $COUNT = N$, since "0101" is written into location, the stuck-at-0 fault is not excited. However, during the $COUNT = 2N$ phase, when the complement of "0101" i.e. "1010" is written to the target memory location, the stuck-at-0 fault at the MSB gets excited and the value read into the ref_reg thereafter is "0010". Thus, during the following ex-or operation of the $data\_reg1$ and ref_reg, a 1 is obtained at the MSB of the result indicating detection of the faulty bit at the MSB.

Write-Disturb fault Detection

The detection of the write-disturb fault is explained in Figure 6.3. Assume a write disturb fault exists at the MSB of the location addressed by $wdf\_test\_address$. At COUNT = 1 value, the data stored in the location addressed by $wdf\_test\_address$ undergoes a refresh cycle. Thus, the data read from the location $wdf\_test\_address$ is copied in ref_reg as well as backed-up in $data\_regsiter2$. The same data is also written back to the row addressed by $wdf\_test\_address$. Thus, at the end of $COUNT = 1$, the data present in ref_reg, data_regsiter2 and $wdf\_test\_address$ are all "1010" as illustrated in Figure 6.3. However, as the same data is written to the MSB of

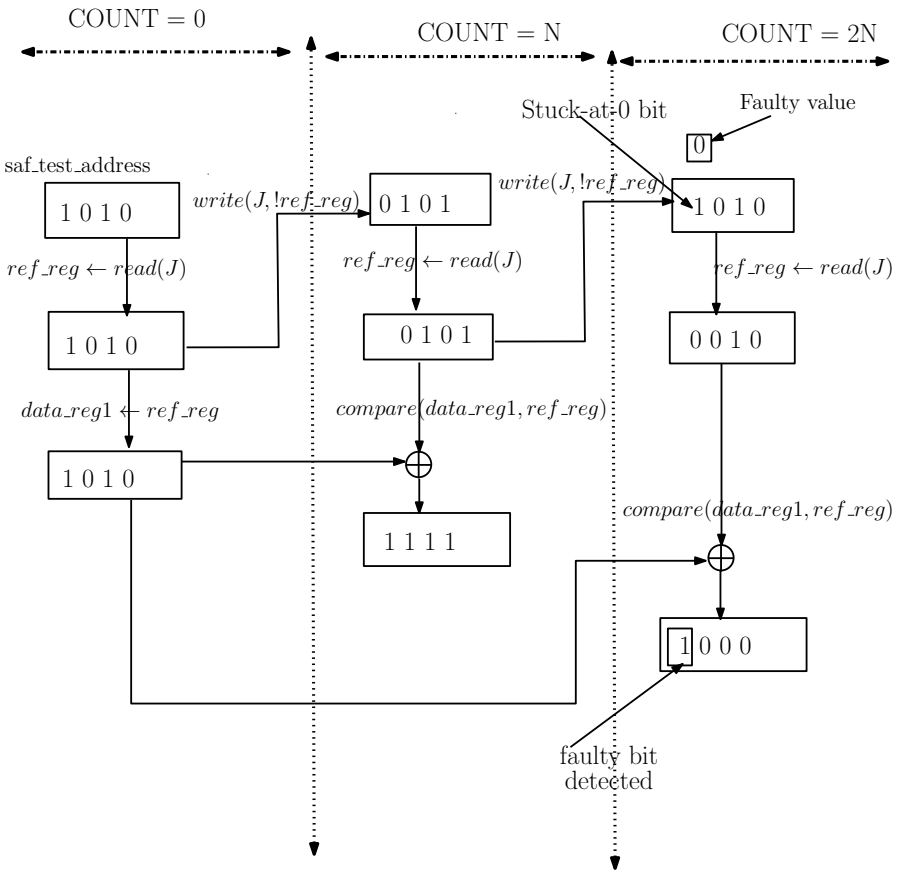


Figure 6.2: Stuck-at-0 Fault Detection using MTMX test

the location addressed by $wdf\_test\_address$, the write-disturb fault gets excited and instead of storing "1010", it stores "0010".

During cycle number $N+1$, the data read from the the location addressed by $wdf\_test\_address$ is loaded in the ref_reg. This data is then compared (bit-wise ex-ored) with data stored in $data\_register2$ (stored during $COUNT = 1$). For a fault-free case an all zero pattern is expected as result. Presence of any 1 at any bit position of the result would mean presence of a write-disturb fault at that bit position. For example as shown in Figure 6.3, at the $N + 1$ cycle, bit wise ex-or of ref_reg holding "1010" and $data\_regster2$ holding "0010" would result in "1000" meaning a write-disturb fault detected at MSB of the data in location addressed by $wdf\_test\_address$.

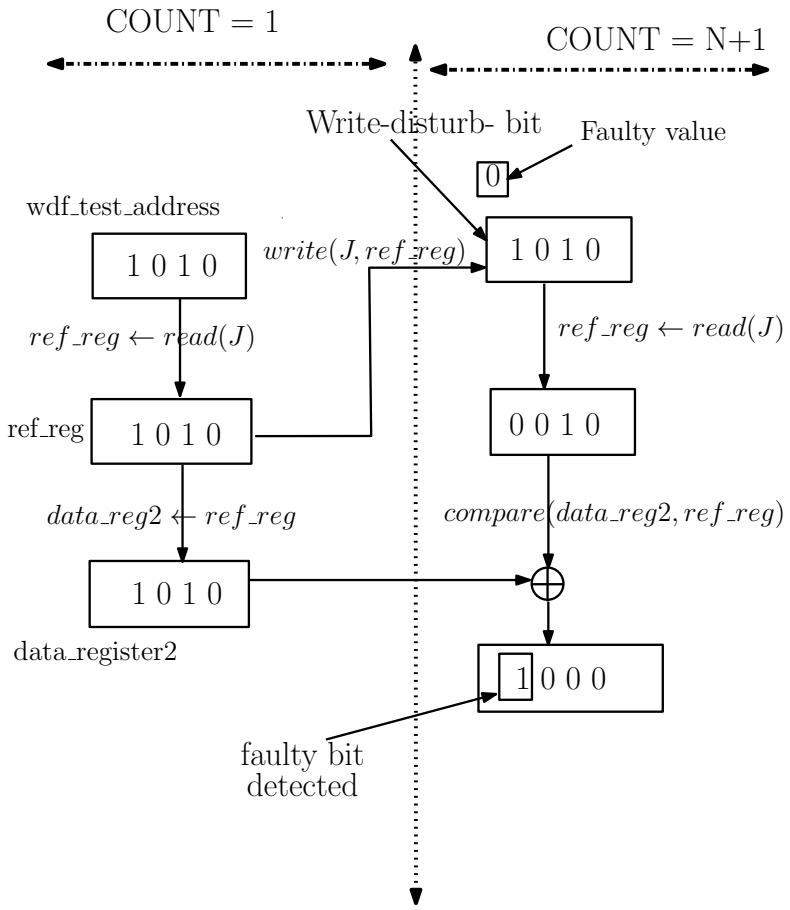


Figure 6.3: Write Disturb Fault Detection using MTMX test

6.4 Refresh re-use based test technique

To prevent the contents of a DRAM from being lost, the DRAM must be refreshed. Refresh is the process of recharging, or re-energizing the cells in a DRAM. Cells are refreshed one row at a time (usually one row per refresh cycle). Refresh in DRAMs has already been covered in Chapter 2. However, before we proceed any further, it is worth reviewing some concepts regarding refresh since they will be used during the subsequent discussions.

6.4.1 Review of DRAM refresh

Refresh cycle refers to the time required to refresh one row. The refresh of a row of bit cells involves reading all cells of that row and writing them back to the same bit cells. Read and write operations on a row are synchronized with the refresh operations of that row.

For a DRAM array to operate correctly, all rows need to be refreshed within certain interval

of time called *Refresh Time*. In other words, *Refresh Time* is how often a row can go without being refreshed before it is in danger of losing its contents. Assume, a DRAM with 512 rows and requiring *Refresh Time* of 10ms. Thus, 512 refresh operations need to be performed within 10 ms period and it means an average of 1 row is refreshed every $1.95\text{x}10^5$ ms.

Refresh rate is the total number of rows it takes to refresh the entire DRAM array. It is determined by the total number of rows that have to be refreshed in a DRAM. DRAM chips are designed for a particular type of refresh. For example, chips using 4K refresh will have about 4000 rows, which means it will take about 4000 cycles to refresh the entire array. Chips using 2K refresh will have about 2000 rows and chips with 1K refresh will have 1000 rows.

Table 6.3 lists the commonly used refresh rates for DRAMs, where all DRAM chips in Table 6.3 have same total capacity of 16Mb. However, the number of rows and columns are different depending on the type of refresh used.

Table 6.3: Different types of DRAM refresh

DRAM Size	4K refresh	2K refresh	1K refresh
4MX4	4000 rows / 1000 columns	2000 rows / 2000 columns	1000 rows / 4000 columns
2MX8	4000 rows / 500 columns	2000 rows / 1000 columns	1000 rows / 2000 columns
1MX16	4000 rows / 250 columns	2000 rows / 500 columns	1000 rows / 1000 columns

In addition to various refresh rates, there are different refresh methods. The two most basic methods of refresh are *Distributed Refresh* and *Burst Refresh* [2]. In *Burst refresh*, a series of refresh cycles are performed one after another until all the rows have been refreshed, after which normal memory accesses occur until the next refresh is required. In *Distributed refresh*, refresh cycles are performed at regular intervals, interspersed with memory accesses [2]. In the presented work, burst refresh cycles are considered as shown in Fig 6.4. The time between two refresh bursts is allowed for normal memory operation (random access) shown as the shaded block in Fig 6.4.

6.4.2 Implementing the MTMX Test using refresh

According to the MTMX test explained in the previous section, the total number of test cycles for test of stuck-at-fault at a location and test of write-disturb fault at the next location requires $2N + 1$ test cycles for a DRAM with N rows. During each test cycle, a read followed by a write operation is performed similar to refresh cycle of a refresh burst. However, the number of refresh cycles in one refresh burst is N for a DRAM with N rows. Thus, to use the refresh

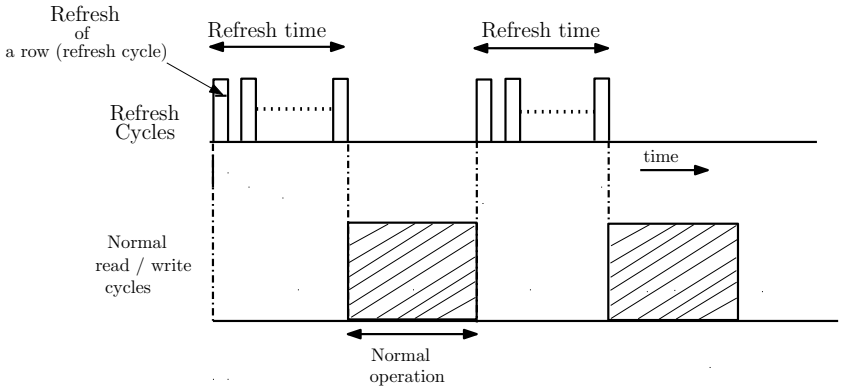


Figure 6.4: Interleaved random accesses and burst refresh cycles

cycles as test cyles for the MTMX test, the refresh burst is extended to cover $2N + 1$ refresh cycles.

The basic idea of the test technique is explained using Figure 6.5. The DRAM considered is assumed to have four locations, with addresses ranging from 0 to 3 as shown in Figure 6.5(i). The numbers 0, 1, 2 and 3 correspond to the variable J in Algorithm 1. These addresses are generated by a $modulo - N$ counter (here N = 4). Thus, for total count of $2 * 4 + 1 = 9$ refresh cycles, the address counter wraps around twice. As shown in Figure 6.5(i), after every four refresh cycle, the addresses are repeated.

The initial contents of the memory (just before the refresh operations) is shown in Figure 6.5 (iii). The data bytes for rows 0,1,2,3 are *a,b,c* and *d* respectively. We assume the rows targeted for test of stuck-at-fault and write-disturb-fault are row 0 and row 1 respectively. Out of the 9 refresh cycles, the first cycle is a test cycle involving the invert phase of the test as shown in Figure 6.5 (ii). The data in row 0 is backed-up in $data\_register1$ and the complement is written back to row 0. Thus, at the end of the first refresh cycle, the contents of the DRAM are $\bar{a}, b, c$ and *d* as shown in Figure 6.5 (iii). The second refresh cycle involves the refresh phase where row 1 is addressed. The data in row 1 is backed-up in $data\_register$ 2 and same data is written back to row 1. Thus, at the end of the second cycle, the contents of DRAM remain same as the DRAM contents after the first cycle, that is $\bar{a}, b, c$ and *d*. For rest of the rows, it is a repeat of the process for row 1 except backing up data in $data\_register2$

After the address counter reaches count 3, the address counter wraps around to again start addressing from row 0. Thus, the fourth refresh cycle is again a test cycle for row 0, where the present contents of row 0 is read and its complement is written back to row 0. Thus, at the end of the fourth refresh cycle, the contents of the DRAM are *a,b,c* and *d*. At the end of this test cycle, original contents of the row 0 is restored and thus is the restore phase of the test.

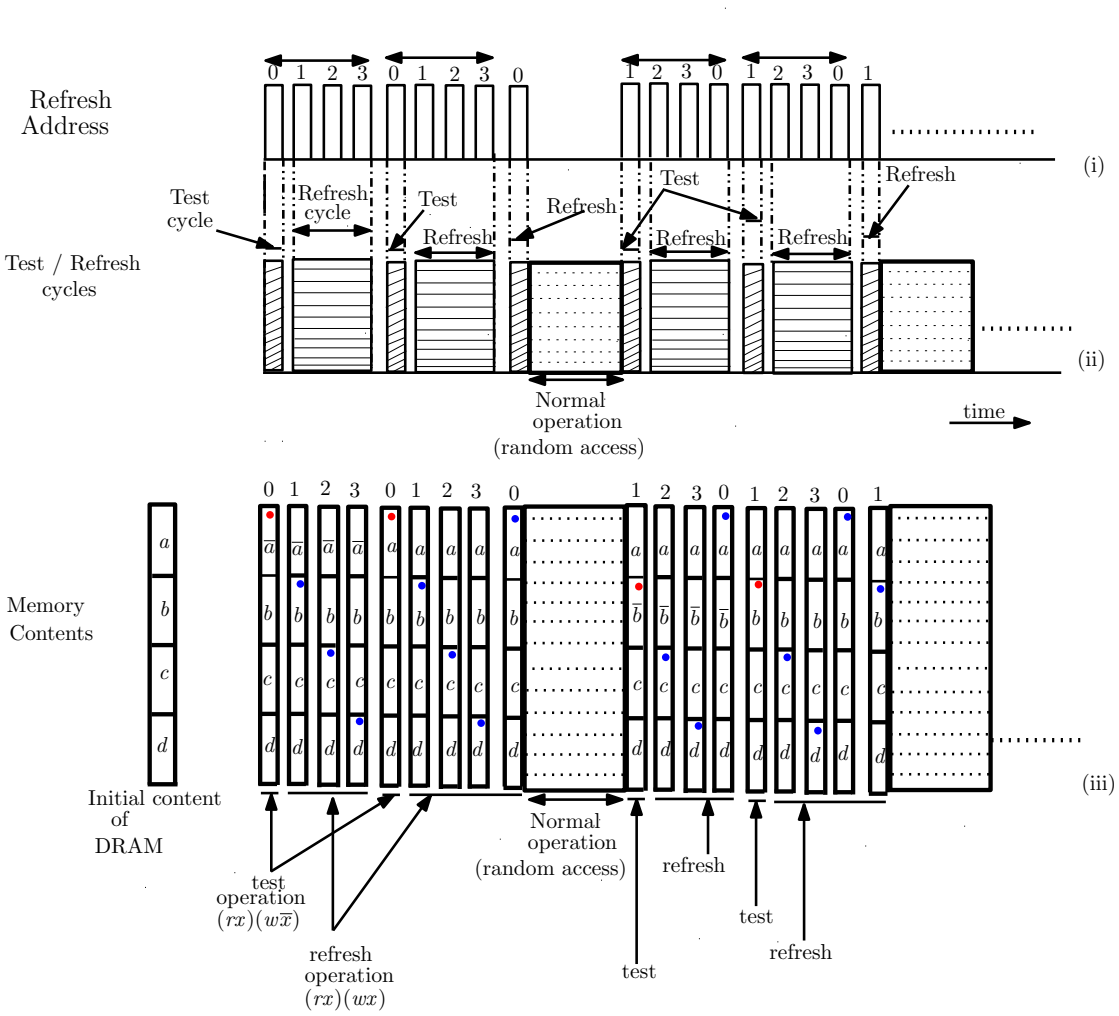


Figure 6.5: (i)Refresh address (ii) Interleaved test and refresh cycles (iii) Memory contents after a Test/Refresh cycle

The next three refresh cycles involve refresh phases where the same data is written back to the row which is read. Thus, at the end of the eighth refresh cycle, the contents of the DRAM are *a,b,c* and *d* and the address counter once again wraps around to point to row 0. By this time, the original data in row 0 has been flipped and then restored to account for detection of any stuck-at-fault that exists in any bit of row 0. At the same time, data in row 1 has also been read twice and written back to the same location. This ensures detection of write-disturb fault at row 1. If any stuck-at-fault is not excited during the first test cycle, it is definitely excited during the second test cycle. In such a situation, one more read followed by write cycle (*rw*) is required for the stuck-at-fault detection. Thus, the ninth refresh cycle, meant for refresh of row 0, involves the normal refresh cycle of reading the content of row 0 and writing it to the same row.

At the end of nine refresh cycles, DRAM assumes normal operation. The address counter generating refresh addresses releases control of the address bus of the DRAM while incrementing itself with its next value and holds it till the next refresh burst. After a certain time (according to the specification of the DRAM), the refresh circuit is again enabled and a new refresh burst of $2N + 1$ refresh cycles is repeated as shown in Figure 6.5 (ii) and (iii). However, this time, the target stuck-at-fault address becomes row 1 and target write-disturb-fault address becomes row 2. The second refresh burst starts with refresh address equal to row 1 and thus is a test cycle meant for row 1.

At the end of the first refresh cycle of the second refresh burst, the contents of DRAM are ($a, \bar{b}, c$ and d) as shown in Figure 6.5 (iii). The rest of the three refresh cycles (meant for row 2, row 3 and row 0) are normal refresh cycles. The fourth refresh cycle is again a test cycle for row 1, at the end of which the original contents of row 1 are restored. The next five refresh cycles are normal refresh cycles which ensure detection of stuck-at-fault (if any) for row 1 and write-disturb fault (if any) for row 2. The second refresh burst ends with address counter pointing to row 2 and releasing the hold of address bus to allow normal operation to proceed.

Continuing in this way, after four refresh burst, all the rows have been tested for stuck-at-fault and write-disturb fault at least once. The fifth burst would again be the same as first refresh burst targeting the same row 0 for stuck-at-fault and row 1 for write-disturb fault. The sixth refresh burst is same as the second refresh burst and so on. Thus, after every four refresh bursts, the same rows are targeted for stuck-at-fault and write-disturb fault. This periodic testing of every row prevents accumulation of faults for all rows including rows which are less frequently accessed during normal operation and are more prone to faults.

6.5 Hardware implementation of the proposed approach

6.5.1 BIST hardware

Figure 6.6 (a) illustrates the proposed Memory Built-In-Self-Test architecture based on the refresh-reuse technique. It includes the following modules.

1. *Memory Controller* : Issues memory read and write controls during normal operation of the memory and during periodic refresh cycles.The memory controller consist of *Controller* and *Refresh circuit*.

   - *a) Controller* : It is a Finite State Machine (FSM) responsible for generating control signals during different states of operation of the proposed Memory Controller. The State diagram of the controller is shown in Figure 6.6 (b) and its operation based on the state diagram explained later.
   - *b)Refresh circuit*: It consists of *Refresh Time Counter*(*RTC*), *Refresh Counter* (*RC*) and *Refresh Address Generator* (*RAG*).
     - *(i) RTC :* It is a self-decrementing counter initialized with a count corresponding to the time between two refresh bursts . Normally, the time between two refresh burst is the sum of the refresh time for all rows and the time of normal operation. However, to suite our proposed technique of refresh based MTMX test, we have extended the count of *RTC* to cover the time for two refresh burst, one refresh cycle and time for normal operation. Once *RTC* count reaches zero, it wraps around and generates *refresh_request* to mark the start of a refresh burst.
     - *(ii) Refresh Count :* It is a $modulo-(2N+1)$ counter, where N is the number of rows in the DRAM. *Refresh Count* corresponds to the number of refresh cycles required to complete the MTMX test for a target row. *Refresh Count* is initialized to zero value and on receiving $refresh\_request$ from *RTC*, the counter starts incrementing at every clock. On reaching the maximum count, it wraps around and stops.
     - *(iii) Refresh Address Counter :* It is a $modulo-N$ counter (N being the number of rows of the DRAM) that generates the refresh addresses for the DRAM. During refresh bursts, the *Refresh Address Counter* is initialized with the address targeted for detection of stuck-at-fault. On receiving the enable signal (*EN*) from the *Controller*, this $modulo-N$ counter starts counting and on completing N cycles, wraps around to mark the completion of refresh of the all rows

once.

2. *Refreshment register*: The data fetched during read cycle is temporarily stored in this register. During normal read operation, data is delivered from *refreshment register* to data register while during refresh, the data read into the *refreshment register* is written back to the same location from where it is read.
   Backup Registers: Two back-up registers $data\_register1$ and $data\_register2$ are used to hold back up data during the *invert* and *restore* phases of the MTMX test when performed on the DRAM.

3. *Glue Logic*: Includes combinational logic like multiplexers and comparators. Multiplexers act as switches for selecting modes of operation (refresh /normal), selecting back-up registers, selecting data to be written in normal or complementary form. Comparators used are mainly ex-or/ex-nor gates performing bit-wise ex-or/ex-nor of *refreshment register* and data registers depending on the phase of the MTMX test.

6.5.2 Operation of the controller

The operations performed by the BIST hardware during refresh and test cycles are explained next with the help of the state diagram of the *Controller* illustrated in Figure 6.6(b).

- *a) Normal Operation :* It is the usual random access read and write operations carried out on a DRAM. The read and write control signals in this state are the ones initiated by the CPU. These signals are routed through the multiplexers (M1 and M2) on de-assertion of $refresh/\overline{normal}$ signal by the *Controller* as shown in Figure 6.6(b). The data read from the DRAM is routed through the de-multiplexer (DM1) to the Data-In-Out Buffer. On receiving the *refresh_request* from the *RTC*, the *Controller* moves from Normal State to Test State.

- *b) SAF Test State :* In this state, the invert phase and the restore phase of the MTMX test is carried out for detection of stuck-at-fault at a location. The $refresh/\overline{normal}$ signal is asserted *high* to allow *ref_add* generated from the *Refresh Address Counter* to be the row address as shown in Figure 6.6(b). The $inv/\overline{noinv}$ line is asserted *high* to allow complement of the data read from a location to be written to the same location.
  During the invert phase, the $test/\overline{ref}$ and saf/$\overline{wdf}$ is asserted *high* to allow the data read in the *refreshment_register* to be backed-up in $data\_register1$. During restore phase, the control lines *E1* and *CE1* are asserted *high*. Asserting *CE1*, allows $data\_register1$ to deliver its content to the *Comparator* while assertion of *E1* enables the *Comparator* so

that comparison of contents of $data\_register1$ and emphrefreshment_ register can be performed. The result of comparison is delivered as *high* or *low* on the stuck-at-fault pass/fail line. A *high* denotes no fault while a *low* denotes faulty. When the *Controller* is in SAF Test state and COUNT reaches $2N$, it marks end of the refresh burst and the *Controller* switches back to *Normal Operation*.

- *c) WDF Test State :* On completion of the invert phase of the test cycle, if COUNT=1, the *Controller* moves from SAF Test state to WDF Test state. In WDF Test State, the test /$\overline{ref}$ is asserted *high* and saf/$\overline{wdf}$ is *low* to allow the data in the *refreshment_register* to be backed-up in Data_Register2 and written back without inversion to the same location. The *Controller* moves to WDF Test State again during COUNT= $N + 1$ value.
  At this time, *E2* and *CE2* are asserted *high*. Asserting *CE2*, allows Data_Register2 to deliver its content to the *Comparator* while assertion of *E2* enables the *Comparator* so that comparison of contents of $data\_register2$ and *refreshment_register* can be performed. The result of comparison is delivered as *high* or *low* on the Write-disturb-fault test pass/fail line. A *high* denotes no fault while a *low* denotes faulty.

- *d) Refresh State :* In this state, the data read in the *refreshment_register* is written back to the same location by asserting the $refresh/\overline{normal}$ signal *high* and inv / $\overline{noinv}$ *low*. The *Controller* remains in the refresh state as long as COUNT is not equal to N or $2N$.

6.6 Analysis and Comparison

6.6.1 Hardware overhead

The proposed BIST architecture implementation has been described in Verilog, synthesized on a commercial 90nm standard cell library and the area of the synthesized BIST architecture estimated. The DRAM considered was of size 4M×16 with *refresh time* of 16ms. The considered DRAM requires 4096 refresh cycles to refresh all rows within the *refresh time* at a *refresh cycle time* of 130ns [2]. The area estimate of different modules in the modified controller architecture is illustrated in Table 6.4.

As shown in Figure 6.6 and as mentioned in the previous section, during normal operation of the DRAM, the modules used are the Memory Controller and Refresh Register. Thus, we can assume that a DRAM without test circuit would only consist of these two modules. However, when refresh cycles are used as test cycles, along with the Memory Controller and Refreshment register, the modules used are Data Registers and Glue logic (multiplexers and comparators). Thus, the area overhead during test is due to the additional modules mentioned above. The area

Table 6.4: Area estimate of the modified Memory Controller

Blocks	**cells(comb)**	**cells(seq)**	**Area(μm^2))**
Controller	31	14	461
refresh_time_counter	22	17	562
refresh_count	22	14	496
refresh_address_generator	21	14	497
row_decoder	19	0	311
column_decoder	19	0	311
M1, M2 and M3 in Fig. 6.6	3*1	0	21
Total	137	59	2659

overhead for the DRAM of size 4M$\times$16 is estimated in Table 6.5.

Table 6.5: Area overhead estimate

Blocks	**cells**	**Area(μm^2)**
Comparator	2*16	288
Data Registers	2*9	211
De-mux(DM1 and DM2 in Fig. 6.6)	2*1	14
Mux(M5 in Fig. 6.6)	1	7
Total area overhead	53	530

The area of all the blocks listed in Table 6.5 depend on the data width of the DRAM. Therefore, we varied the data widths of the DRAM and estimated the area overhead in each case as shown in Table 6.6. The results confirm that the area overhead of the proposed test hardware varies linearly with data width. Mathematically, we can put it as $A(n) = c1 \times n$ with, A = area overhead, n = data width and $c1$ = constant.

The only published work reporting hardware implementation of a Transparent on-line memory avoiding signature prediction and aliasing has been reported in [87]. Since our proposed architecture also avoids signature prediction and aliasing, we compare the area overhead of our proposed test hardware to that of hardware proposed in [87]. The area overhead of the TOMT hardware proposed in [87] was estimated as

Table 6.6: Comparison of Area overhead for different data width

Data width	**Area(μm^2)**
4MX16	530
4MX32	1060
4MX64	2120
4MX128	4240

$$A(N, n, cb) = c2 \times N + c3 \times (n + cb) + c4 \tag{6.1}$$

with A = area overhead, n = data width, cb = number of check bits and $c2$,$c3$ and $c4$ = constants If we only consider active fault test by the TOMT hardware proposed in [87], we can simplify equation (2) as

$$A(N, n) = c2 \times N + c3 \times (n) + c4 \tag{6.2}$$

Comparing equations 6.1 and 6.2, it can be concluded that area overhead of the hardware for the proposed MTMX test of DRAM is independent of number of rows (N) of the DRAM and hence is more area efficient as compared to the one proposed in [87].

Moreover, the proposed architecture is area efficient compared to off-line test techniques as well. MBIST executing a March element is required to generate either ascending or descending address sequence and thus requires an address generator to generate the required address sequence. The address generator is typically implemented with an up/down counter,gray code based address generator [100] or using Linear Feedback Shift Register (LFSR) [85]. Thus, every implementation of DRAM Memory BIST require some address generation technique as a part of the BIST circuitry and thus add to the area overhead associated with the overall BIST. However, our proposed BIST architecture does not require any address generator circuit and hence brings down the overall area overhead of the Memory BIST substantially.

6.6.2 Test cycle time

Consider a $N \times B$-bit DRAM and a word-oriented March test with P Read/Write operations. We assume that B is power of two. As described in section 6.3.1, when the March test is converted into its Transparent version, the initial write operation is dropped. Thus, the number of read/write operations required by the Transparent March test is (P-1). However, during transformation of Transparent March test (TMX) to MTMX test, a write operation is added after the last

read operation bringing back the number of Read / Write operations again to *P*. According to Algorithm 3, out of the *P* Read/Write operations, (*P-2*) operations are performed during the first *2N* cycles while the last two operations are performed in the $(2N+1)^{th}$ cycle. Thus, the number of memory accesses required for test of a location for stuck-at-fault (including write-disturb fault test for a different location) = *(P-2)(2N) + 2*.

Since our proposed MTMX test targets DRAMs without *ECC*, we compared the test time of our proposed algorithm with that of works where Transparent test scheme avoiding *ECC* for memories were proposed as in [87] and [58]. However, our proposed MTMX test starts assuming an available word-oriented March test unlike the algorithms proposed in both [87] and [58] where they assume availability of a bit-oriented March test. Thus, the test cycle times for [87] and [58] include the time for bit-oriented March test to word-oriented March test conversion. This is reflected in Table 6.7, where both cycles times for Scheme 1 and Scheme 2 are $\mathcal{O}(BN)$. The bit-oriented to word-oriented conversion for our proposed MTMX test involves replacing every bit in the bit-oriented March test by the data word (or its complement) of the row targeted for stuck-at-fault test. Thus, the bit-oriented to word-oriented test conversion for our proposed MTMX test is $\mathcal{O}(1)$.

Table 6.7: Comparison of Different Transparent Test Schemes

Scheme	**Test Cycles**
Scheme1 [87]	(4+8B)*N
Scheme2 without *ECC* [58]	(P+5$\log_2 B$+2)*N
Proposed MTMX Test	*(2N(P-2) + 2)*N*

As shown in Table 6.7, the running time of MTMX test is $\mathcal{O}(N^2)$ when considered for test of all rows. This is large considering Schemes 1 and 2 which have running time of $\mathcal{O}(BN)$.

However, Schemes 1 and 2 are performed during idle times of the processor, which can be small or large depending on the application run by the processor. Thus, for cases where idle time of processor is small, the Transparent test performed in Schemes 1 and 2 would get interrupted frequently and the total test time would get prolonged. However, when the proposed MTMX test is implemented using the refresh circuit, the test time is governed by a constant refresh time of the DRAM. Thus, the tests would be performed periodically without interruption and would finish after a definite time as shown in Table 6.8.

A refresh cycle involves read followed by write operation. Therefore, each refresh cycle involves two memory accesses. Thus, when MTMX test is performed using the refresh circuit,

the number of memory accesses required for test of stuck-at-fault at a row and write-disturb fault test at the next row is $\{(P-2)(2N)+2\}/2$.
Therefore, the test cycle time is calculated as follows.
*Test cycle time = no. of memory accesses required *no. of refresh cycles * refresh cycle time.*
Therefore, *Test cycle time* = $\{(P-2)*(N)+1\}$*N * *refresh cycle time* where, N and P have the same definitions as mentioned above. The MTMX Test cycle time for different refresh rates of a $16Mb$ DRAM tested for Stuck-at-fault/Write-Disturb-Fault is illustrated in Table 6.8 . The refresh cycle time has been assumed to be 130 ns.

Table 6.8: MTMX Test cycle time for different refresh rates of a $16Mb$ DRAM

#(r/w) operations	Refresh rate	Test cycle time
6	4K	4.16 sec
6	2K	1.04 sec
6	1K	0.26 sec

6.6.3 Other features

1. Refresh re-use for memory cores interconnected using NoC : The refresh re-use based online detection technique for a commodity DRAM can be extended to a number of embedded DRAM cores interconnected using NoC. Since the proposed technique utilizes DRAM controller as BIST hardware, no additional test hardware will be required for the DRAM cores as each will be tested by its own DRAM controller. However, to further reduce the area overhead, the DRAMs core can be grouped with each group sharing a single DRAM controller and tests being performed on all cores in a group concurrently.

2. Multiple fault detection per row: Stuck-at-faults occurring at more than one bit of a row can also be detected using MTMX test. The test involves bit wise ex-or of $data\_register1$ (or $data\_register$ 2) with ref_reg. Deviation of ex-or/ex-nor operation result in any bit from the expected result would mean fault at that bit. Since intra-bit coupling effects are not valid for field deployed DRAMs, there is no chance that the fault effect at a bit would be masked by any other bit of the same row. Thus, even if more than one bit suffer from stuck-at-fault effect, each can be detected by just observing the ex-or/ex-nor result at each bit position.

6.7 Summary

The Modified transparent March Test proposed in this chapter has been shown to be a cost effective technique that can detect errors developed in DRAMs during field operation. We have shown that the MTMX test technique performs an active test of DRAM by uncovering functional defects such as stuck-at-faults and write-disturb faults. The refresh cycles of DRAM have been re-used to act as test cycles while performing MTMX test on a DRAM. Since DRAM is refreshed periodically, using refresh cycles for test implies periodic testing. Thus, in the refresh re-use based implementation of the MTMX test, it was shown that the test would be repeated periodically and thus would avoid accumulation of faults.

Since the focus of the thesis has been on test of NoC based memory core systems, the discussion remains incomplete without the coverage of test techniques for memory cores which are part of the NoC infrastructure. The next chapter (Chapter 7) highlights the importance of testing SRAM based FIFO buffers present in the routers of the NoC. It also discusses the proposed on-line test technique for testing these FIFO buffers.

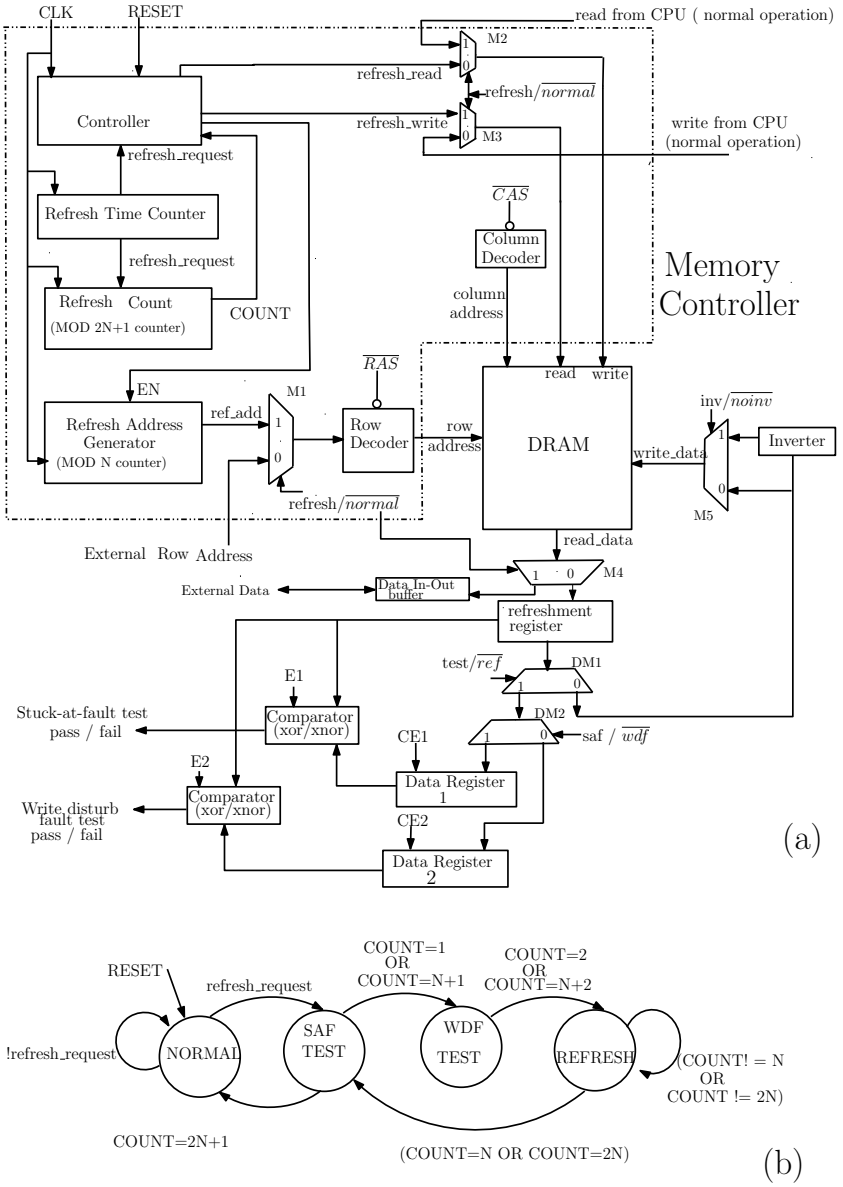


Figure 6.6: (a) The Proposed Memory BIST Architecture (b) State diagram of the Controller

Chapter 7

Test of FIFO Buffers in NoC Routers

A major design issue during test of cores interconnected using NoC is the choice of Test Access Mechanism (TAM). The re-use of the NoC to act as Test Access Mechanism (TAM) as proposed by Cota et al. in [23] and [24] is a very attractive solution as it overcomes the problem of additional TAM area overhead . However, to use the NoC as TAM, it must be ensured that the elements of NoC are fault free. Testing the elements of the NoC infrastructure involves testing routers and inter-router links [43], [72]. A generic router of a NoC infrastructure is composed of combinational part and FIFO buffers. The combinational part is responsible for routing data from source to destination while FIFO buffers are present at every input port to receive incoming data and at every output port data to store these data before they are put on the output port. Considering the router architecture, proposals for test of NoC routers reported in literature have considered separate test strategies for combinational part and FIFO buffers as in [34], [43] and [52].

Significant amount of area of the routers present in the NoC data transport medium is occupied by FIFO buffers. Accordingly, the probabilities of faults or defects occurring in buffers are significantly higher compared to the other components of the NoC. Thus, test process for the NoC infrastructure must begin with test of the FIFO buffers. Conventional FIFO buffer designs are based on SRAMs as they offer an area efficient bit cell [98]. Thus, FIFO buffers are tested with functional defects similar to SRAMs using either dedicated BIST architectures as proposed by Wielage et al. in [98] and Prinetto et al. in [75] or distributed BIST architectures proposed by Grecu et al. in [34]. The functional faults which develop in SRAM based FIFO buffers are of two types. The first type are the *permanent* faults such as stuck-at-faults. The second type are the run-time faults which are developed when the buffer is in operation. These run-time faults are either *transient* faults or *intermittent* faults. A discussion on both these faults has already been provided in the last chapter.

7.1 Motivation

Though researchers have given importance to detection of functional faults in FIFO buffers, the detection of run-time permanent functional faults have been overlooked. One probable reason could be the belief that with advent of deep sub-micron technology (DSM), permanent faults developed during run-time are not as frequent as transient faults [62]. However, as already discussed in the motivation section of Chapter 6 that studies done in [49], [78] and [83] revealed the true picture that memories experience both transient fault (soft error) and permanent (hard error) faults in-field. Moreover, soft error data detection and protection schemes have not proven to be effective for testing in field permanent faults in FIFO buffers as they do not alter the contents of the memory and therefore cannot find functional faults in memories as mentioned in [87].

It is therefore necessary to find a cost-effective test technique that can detect permanent faults developed during field operation of FIFO buffers. Such a test technique must possess the following characteristics :

- The test must be an active test so that functional defects are uncovered.
- The test must be performed periodically to ensure that no fault gets accumulated; this necessitates on-line test technique.
- The on-line test must ensure that the contents of the buffer are not destroyed during test
- The test hardware must be cost effective in terms of area.

The work presented in this chapter is about the proposed on-line Transparent test technique for FIFO buffers present within the routers of the NoC infrastructure. The proposal involves generation of a SOA-MATS++ test generation algorithm targeting in-field permanent faults developed in SRAM based FIFO memories and repeating the test periodically to perform active fault detection over the entire FIFO buffer.

7.2 Fault Models Considered

The runtime *permanent* faults considered in this work are assumed to be *intermittent* faults which have become permanent over time. Consequently, the fault models considered in this chapter are that of intermittent faults. The factors which lead to intermittent faults are variations of temperature, voltage and aging effects such as Time Dependent Dielectric Breakdown (TDDB), Electro-migration, Negative bias temperature instability (NBI) and hot carrier injection (HCI) as

mentioned in [97]. A brief discussion of each of the physical effects mentioned above and their effect on the logical behavior of the system have already been discussed in Chapter 6. The target fault models considered for this work are stuck-at fault, stuck-open fault, read disturb fault and transition fault. Detailed behavior of these faults are as follows.

1. stuck-at-fault (SAF) - the defective SRAM cell permanently contains a 0 (SA0) or 1 (SA1) and cannot be changed. To detect a stuck-at-fault, a write operation at the cell must be followed by a read operation. For example, to detect SA0, a write 1 must be followed by a read 1. Similarly, to detect SA1, a write 0 must be followed by a read 0 [13].

2. stuck-open-faults (SOF) - the defective cell cannot be accessed due to an open word line or bit line. A stuck-open fault is detected as if it were a stuck-at fault if the sense amplifier is combinational and thus passing a proper defined logical value to the output pin [26].

3. transition fault (TF) - A memory cell with a transition fault fails to undergo at least one of the transitions $0 \rightarrow 1$ (up transition) or $1 \rightarrow 0$ (down transition) when it is written with 0 or 1 respectively [13]. To detect a transition fault, each cell must undergo an up transition and down transition and be read after each.

4. read disturb fault (RDF) - a read operation performed on the defective cell changes the data in the cell and returns an incorrect value on the output [12]. To detect read disturb fault, a 1 and a 0 should be read from each cell.

All faults considered in this work are single port, static and simple faults as mentioned in [38]. These faults can be detected using standard March tests such as MATS++, March C-test [13] or special March test such as March SS test [38] for SRAMs. However, to ensure restoration of contents after test, the March tests to be applied are modified to their Transparent versions as discussed in the next section.

7.3 Proposed Transparent Test Generation Technique

All faults considered in this chapter, if applied for SRAMs or DRAMs, can be detected using standard March tests mentioned in Chapter 2. However, if the same set of faults are considered for SRAM type FIFOs, then March test cannot be used directly due to the restriction in SRAM type FIFOs mentioned in [93] and stated below.

On completion of read or write operation in a FIFO, the address increments. This implicit address modification (increment) restricts each *March* element to a maximum of one read and/or one write operation. Thus, the addressing order of test that suits the above restriction for SRAM

type FIFOs is single address order (SOA) because the addresses are specified implicitly and address modification in the form of incrementing is also performed implicitly in SOA. The address order restriction for SRAM type FIFOs and the choice of single order addressing motivated us to choose single order address MATS++ test (SOA-MATS++) [93] for detection of faults considered in this work. The SOA-MATS++ test is represented as

$$\{\uparrow (w0); \uparrow (r0, w1); \uparrow (r1, w0)\}; \uparrow (r0)\}$$

Since we considered word-oriented March test, we converted the bit-oriented SOA-MATS++ test to word-oriented SOA-MATS++ test. The procedure to convert a bit-oriented March test to word-oriented March test is assumed to be the one mentioned in [89]. The word-oriented SOA-MATS++ test is represented as

$$\{\updownarrow (wa); \uparrow (ra, wb); \downarrow (rb, wa)\}; \updownarrow (ra)\}$$

where, *a* is the data background and *b* is the complement of the data background. $\uparrow$ and $\downarrow$ are increasing and decreasing addressing order of memory respectively. $\updownarrow$ means memory addressing can be increasing or decreasing. Unlike the proposal in [89], we did not consider varying the data backgrounds as coupling faults are not considered in this work.

The first element of the test, (*wa*) is the initialization step meant for writing to each row of the memory a finite data background. The next two elements, (*ra,wb*) and (*rb,wa*) perform read and write operations on the known data background. The element (*ra, wb*) reads the data and writes its invert before proceeding to the next row while (*rb,wa*) reads the invert and writes the re-invert on each row. After (*rb,wa*) operation on each row of the memory, each bit of each row has been flipped in both directions. The last element (*ra*) reads each row and expects the same data background as the initial phase. Any mismatch ensures detection of fault at the particular row.

Application of SOA-MATS++ test to the FIFO involves writing patterns into the FIFO memory and reading them back. As a *result*, the memory contents are destroyed. However, on-line memory test techniques require restoration of the memory contents after test. Thus, researchers have modified the March tests to Transparent March test so that tests can be performed without the requirement of external data background and the memory contents can be restored after test.

To convert the SOA-MATS++ test to Transparent SOA-MATS++ (TSOA-MATS++) test, the following steps have been performed.

a) Dropping the initial element - Since the Transparent test is independent of any background data, no initialization step is required.

b) Replacing a and b in the SOA-MATS++ test representation (given above) by x and $\overline{x}$ respectively, where x indicates the data value at a row prior to performing the March test on the row and $\overline{x}$ indicates the complement of x. At this point, the SOA-MATS++ test can be represented as :

$$\{\uparrow (rx, w\bar{x}); \uparrow (r\bar{x}, wx); \uparrow (rx)\}$$

The SOA-MATS++ test has three March elements M1, M2 and M3 (separated by semi-colons). For each March element, up/down arrows represent the address order sequence of the test. The read / write operations following the up / down arrows represent the particular operation performed at an address before moving on to the next address. The read access (rx) and ($r\bar{x}$) expects x and $\bar{x}$ to be read respectively from the row to be tested while (wx) and ($w\bar{x}$) refers to writing x and $\bar{x}$ to the row that is tested.

c) All three March elements from step 2 are merged to form a single March element with addressing order same as each of M1, M2 and M3.

Thus, the Transparent SOA-MATS++ test generated is represented as

$$\{\uparrow (rx, w\bar{x}, r\bar{x}, wx, rx)\}$$

The operations performed during the test represent three phases of the test namely *invert* phase, *restore* phase and *read* phase as shown in Table 7.1.

Table 7.1: Phases in MTMX test

Phase	**Operation**	**Test Run**
Invert	*(rx, $w\bar{x}$)*	Run 1
Restore	*($r\bar{x}$, wx)*	Run 2
Read	*(rx)*	Run 3

The first two operations forms a read write pair (*rx,wx*) representing the invert phase where the initial content (content before start of test) of the FIFO buffer location under test (*lut*) is read and its complement is written back to the same location. The invert phase is followed by restore phase involving the operations *($r\bar{x}$, wx)*, where the content of *lut* are read and re-inverted. At

this point of the test, the contents of *lut* have been flipped twice to get back the original content. The last phase *(rx)* involves reading the content of *lut* without any write operation to follow.

7.3.1 The test algorithm

The algorithmic interpretation of the Transparent SOA-MATS++ test is presented in Algorithm 4. It describes the step by step procedure to perform the three phases of the Transparent SOA-MATS++ test for each location of the FIFO memory. The target location for test is given by the loop index *i* which varies from *0* to (*N-1*), where *N* is the number of locations in the FIFO memory. In other words, *i* represents the address of the FIFO memory location presently under test. For each location, the three test runs mentioned in Table 7.1 are performed during three iterations of the loop index *j*.

Algorithm 4 : Transparent SOA-MATS++ Test Algorithm

Input: N = number of rows of the FIFO memory

```
1: i ← 0;                          /* Initialization */
2: while (i ≤ N − 1) do
3:     j ← 0;
4:     while (j ≤ 2) do
5:         temp ← read(j);
6:         if (j = 0) then
7:             original ← temp;
8:             write(j, !temp);
9:         else if (j = 1) then
10:            result ← compare(temp, original);
11:            write(i, !temp);
12:        else
13:            result ← compare(temp, original);
14:        end if
15:        j ← j + 1;
16:    end while
17:    i ← i + 1;
18: end while
```

For a particular FIFO memory location (present value of *i*), the first iteration of *j* performs the invert phase, where the content of the FIFO location is inverted. The invert test phase involves the following operations. The content of location under test (*lut*) is read into a temporary variable *temp* and then backed-up in variable called *original*. Then, the inverted content of *temp* is written back to *lut*. At this point, the content of *lut* is invert of content of *original*.

In the next iteration of *j*, the restore phase is performed. The content of *lut* is re-read into *temp* and compared with content of *original*. The comparison should result in all 1's pattern. However, deviation from the all 1's pattern at any bit position indicates fault at that particular bit position. Next, the inverted content of *temp* is written back to *lut*. Thus, the content of *lut* which were inverted after the first iteration get restored after the second.

The third iteration of *j* performs only a read operation of *lut*, where the content of *lut* is read into *temp* and compared with contents of *original*. At this stage of the test, all 0's pattern in the result signifies fault-free location while deviation at any bit position from all 0's pattern means fault at that particular bit position. The last read operation ensures detection of faults which remained undetected during the earlier two test runs. At the end of the three test runs (iterations of *j*), the loop index *i* is incremented by one to mark the start of test for the next location.

7.3.2 Fault coverage of the proposed algorithm

The Transparent SOA-MATS++ algorithm is intended for test of SAF, TF and RDF fault tests developed during field operation of FIFO memories. The fault coverage of the algorithm is illustrated in Figures 7.1 and 7.2. In both the figures, the word size of FIFO memory is assumed to be of 4 bits. The text in italics against the arrows indicate the operation performed while the text in bold font correspond to the variables used in Algorithm 4.

Fault detection during invert and restore phase

With reference to Figure 7.1, assume the data word present in location under test (*lut*) addressed by *i* be "1010". The test cycles begin with the invert phase (j= 0 value) during which the content of location addressed by *i* is read into *temp* and then backed-up in the *original*. The data written back to *lut* is the complement of content of *temp*. Thus, at the end of the cycle, the data present in *temp* and *original* is "1010" while *lut* contains "0101". Assume a stuck-at-1 fault at the most significant bit (MSB) position of the word stored in *lut*. Thus, instead of storing "0101", it actually stores "1101" and as a result, the stuck-at-fault at the MSB gets excited.

During the second iteration of *j*, when *lut* is re-addressed, the data read in to *temp* is "1101". At this point, the data present in *temp* and *original* are compared (bit-wise exor-ed). An all 1's pattern is expected as result. Any 0 within the pattern would mean a stuck-at fault at that bit position. This situation is illustrated in Figure 7.1, where the ex-or of "1010" and "1101" yields a 0 at the MSB position of the *result* indicating a stuck-at-fault at the MSB position.
However, for cases where the initial data for a bit position is different from the faulty bit value, the stuck-at-fault cannot be detected for the bit position after the restore phase of the test. For example for Figure 7.1, if *lut* initially contains "0010", then the stuck-at-one fault at MSB does

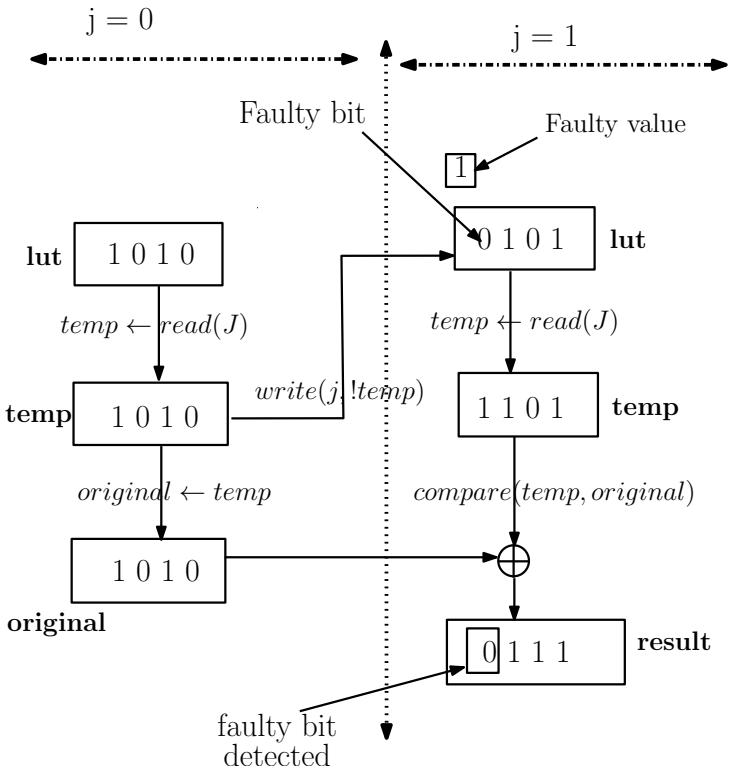


Figure 7.1: Fault Detection during invert phase and restore phase of the Transparent SOA-MATS++ test

not get excited during the second iteration of *j*. It thus requires one more test cycle to excite such faults. Moreover, read disturb faults and transition faults which require flipping bits in both directions also remain undetected after the restore phase.

Fault detection during the read phase

If a stuck-at-fault is not excited during the first two phases, it is excited during the last read phase ($j = 2$) phase of the SOA-MATS++ test. Such a situation is shown in Figure 7.2. In Figure 7.2, it is assumed that the target memory location has a stuck-at-0 fault at the MSB position. During the invert phase, since "0101" is written into *lut*, the stuck-at-0 fault is not excited. However, during the restore phase, when the complement of ("1010") is written to the *lut*, the stuck-at-0 fault at the MSB position gets excited and thereafter, the value read into *temp* is "0010". Thus, during the following ex-or operation of *temp* and *original*, a 1 is obtained at the MSB of the *result* indicating presence of a faulty bit at the MSB position.

The detection of RDF and TF requires writing to each bit a 0 and 1 and reading them after each write. Thus, the Transparent version of the test for RDF and TF requires flipping bits of the memory location in both directions and reading them after each flip. Since the test process for

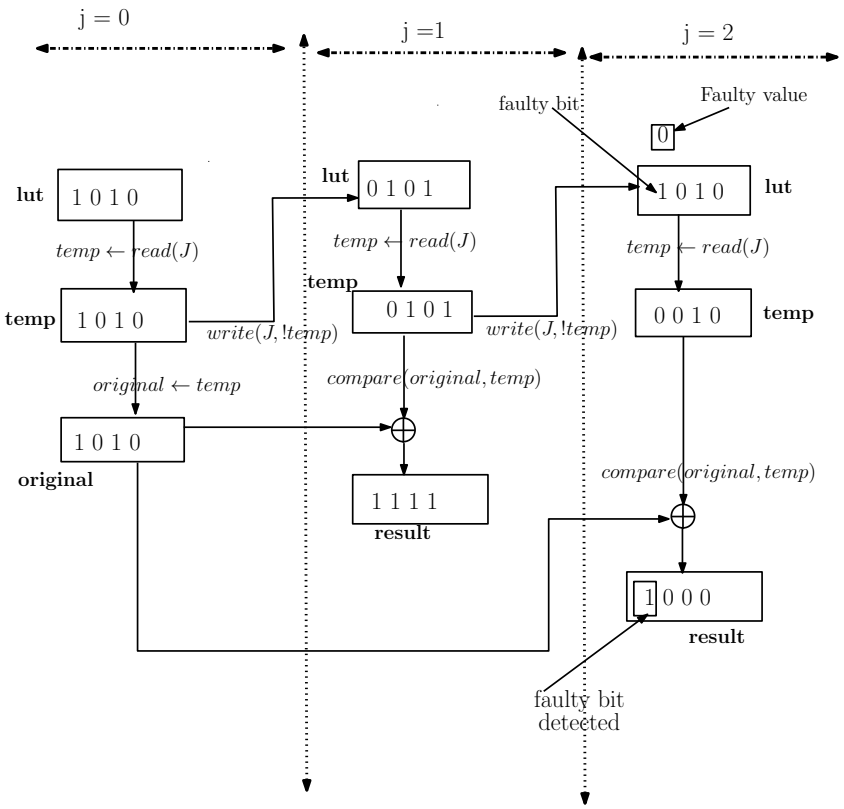


Figure 7.2: Fault Detection during read phase of the Transparent SOA-MATS++ test

detection of stuck-at-faults involves bit flips in both directions, the detection technique for RDF and TF are same as for SAFs and thus is not detailed out.

7.4 Proposed test technique

In this section, we present the implementation of the Transparent SOA-MATS++ test on a 2x2 mesh type NoC as shown in Figure 7.3 (a). The network has four cores, namely, *c1*, *c2*, *c3* and *c4* connected to the four routers, *r1*, *r2*, *r3*, and *r4* respectively. The switching technique is assumed to be *wormhole* switching [25]. Accordingly, data packets are divided into flow control units (*flits*) such as header flit, payload flit and tailer flit and are transmitted in pipeline fashion [25]. The flit movement in the considered NoC infrastructure is assumed to require buffering only at the input channels of routers. Thus, FIFO buffers are required only at the input channels as shown in Figure 7.3(b).

Assume an application requires data traffic movement from core *c1* to core *c2* (shown by

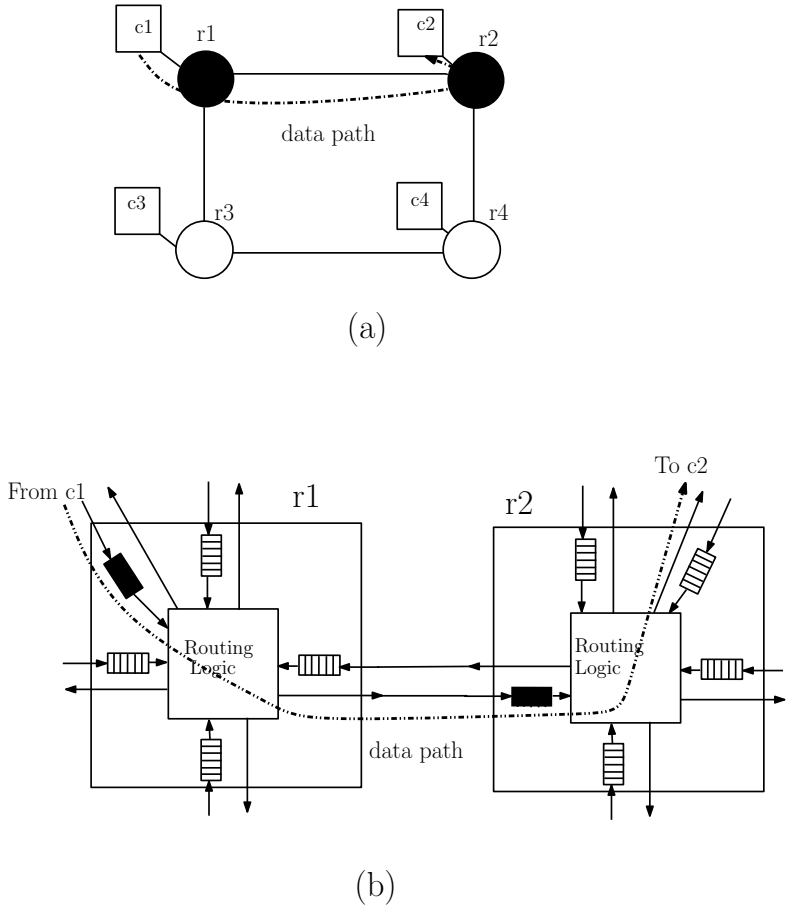


Figure 7.3: (a) data traffic movement in 2x2 mesh type NoC (b) FIFO buffers involved during the data movement

dotted lines in Figure 7.3 (a)). As data moves from *c1* to *c2*, it crosses routers *r1* and emphr2 (shown by dark circles in Figure 7.3 (a)) along the path. Considering wormhole switching for data movement from *c1* to *c2* via the NoC, flits will have to be buffered along the path at intermediate nodes before passing on to the next node. The storage will be done at the input FIFO buffers of the routers *r1* and *r2* (shown by dark rectangles in Figure 7.3 (b)). This movement of flits from *c1* to core *c2* along the NoC infrastructure is the normal operational cycle of the NoC and will be henceforth referred to as *normal* mode of the FIFO buffers. The normal mode and test mode of operation of a FIFO buffer are synchronized with two different clocks. The clock used for test purpose (referred as *test_clk* in the chapter) is a faster clock compared to the clock required for normal mode (*router clock*).

The FIFO buffers are allowed to be operative in normal mode for sufficient amount of time before initiating their test process. This delay in test initiation provides sufficient time for run-time intermittent faults developed in FIFO buffers to transform into permanent faults. The test

process of a targeted FIFO buffer is initiated by a counter which switches the FIFO buffer from normal mode to test mode by asserting a test control line (*test_ctrl* signal) as shown in Figure 7.4. As long as the *test_ctrl* signal remains *high*, the FIFO buffer remains in test mode and on asserting the *test_ctrl* signal *low*, the buffer is switched back to normal mode.

The test initiating counter is a decrementing counter synchronized with the router clock. It is initialized with a count value corresponding to the pre-determined allowed normal cycle time shown in Figure 7.5. On reaching the count value *zero*, the counter wraps around, gets de-activated and asserts the *test_ctrl* signal to mark the start of the test process. On completion of the test process, the counter is activated and it starts the down count until the next test burst.

The switching of FIFO buffers from normal mode to test mode occurs after a certain period of time without caring about the present state of the FIFO buffer. It may be argued that at the instant of switching, the buffer may not be full, and as a result not all locations would be tested during the test cycle. However, test initiation after the buffer gets full would cause the following problems. First, waiting for the buffer to get full would unnecessarily delay the test initiation process and would allow faults to get accumulated. Second, test of the entire buffer would prolong the test time and would negatively affect the normal mode of operation. Moreover, as coupling faults are not considered in this work, the fault effect in a buffer location due to faults in neighboring locations need not be considered. The problem of testing the entire buffer is solved by repeating the test periodically and is explained later in this section.

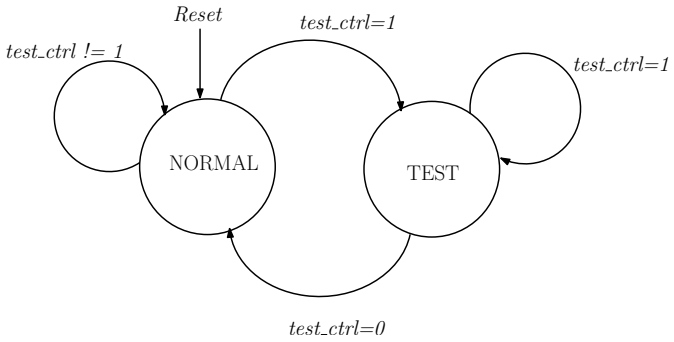


Figure 7.4: State diagram representation of the test process

7.4.1 The test process : periodic and on-line

A test burst involves series of test read and write cycles as shown in Figure 7.5. The read and write cycles are performed respectively on the rising and falling edges of the test clock. Consequently, a number of test clock cycles can be accommodated within the time the *test_ctrl* signal remains high. It requires three read and two write cycles, or in other words three cycles of

the faster test clock to perform a Transparent SOA-MATS++ test on a single location of a FIFO buffer. However, as shown in Figure 7.5, the number of test cycles accommodated within the test time interval is more than three. This allows performing the SOA-MATS++ test on more than one location of the buffer. However, it may be argued that during a test burst, not all FIFO buffer locations are tested or a test of a location can get interrupted as the *test_ctrl* signal gets asserted *low*. Such a situation is shown in Figure 7.5.

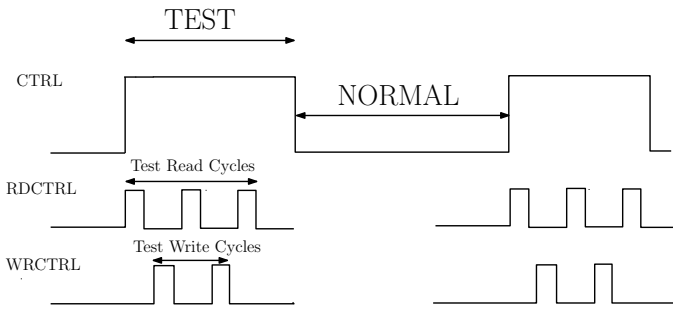


Figure 7.5: Interleaved test and normal cycles

These two problems can be avoided by periodically testing the FIFO buffers. Periodic testing of a FIFO buffer allows test of a different set of locations of the FIFO buffer in each test burst. Every time the buffer is switched to test mode, the normal process gets interrupted. The FIFO memory location currently addressed in normal mode, at the instant of switching, becomes the target location for test. Since, normal operation is interrupted at different instants in different test bursts, the locations tested in each burst would be different. Thus, repeating the test bursts for a number times on a FIFO buffer would cover the test of each location as the number of locations in a FIFO buffer are few (4 or 6 or 8). Periodic testing also prevents accumulation of fault in the buffer. Periodic testing of a FIFO buffer using the Transparent SOA-MATS++ test is illustrated in Figure 7.5. The periodicity of test bursts is maintained by the test initiation counter.

As long as the FIFO buffer remains in test mode, no data from the input channel is allowed to be written to the buffer. Consequently, flits traveling through *r1* and heading towards *r2* as shown in Figure 7.3 (a) and (b) have to wait until the test of input FIFO buffer (represented by dark rectangles in Figure 7.3 (b)) is over. However, other FIFO buffers which are not considered for test at that instant, continue to remain in normal mode. For example in Figure 7.3 (b), the buffers in router *r1* and *r2* represented by shaded rectangles continue to be in normal mode. Thus, flits traveling through *r1* and heading towards a direction other than *r2* will travel undisturbed. This feature of the proposed test process justifies it to be an on-line test process.

7.4.2 Test architecture

The FIFO buffer present in each input channel of a NoC router consist of a SRAM based FIFO memory of certain depth. During normal operation, data flits arrive through a *data_in* line of the buffer and are subsequently stored in different locations of the FIFO memory. On request by the neighboring router, the data flits stored are passed on to the output port through the *data_out* line. Figure 7.6 (a) shows the FIFO memory with *data_in* and *data_out* line.

To perform the Transparent SOA-MATS++ test on the FIFO buffer, we added a test circuit, few multiplexers and logic gates to the existing hardware as illustrated in Figure 7.6 (a). The details of the test circuit are shown in Figure 7.6 (b). The read and write operations on the FIFO buffer are controlled by the read enable (*read_en*) and write enable (*write_en*) lines respectively. The multiplexers *mu6* and *mu7* select the read and write enable during the normal and test process. During normal operation when the *test_ctrl* is asserted low, the internal write and read enable lines, *wen_int* and *ren_int*, synchronized with the router clock, provide the write and the read enable respectively. However, during test process, the write enable and read enable are synchronized with the test clock.

As mentioned earlier, the read and write operations during test are performed at alternate edges of a test clock. The read operations are synchronized with the positive edges as shown in Figure 7.6 (a). The *write_clk* is obtained by inverting the test clock. In test mode (*test_ctrl* high), the test read and write addresses are generated by test address generators implemented using gray code counters similar to the normal address generation. MUXes *m4* and *m5* are used to select between normal addresses and test addresses.

Consider the situation when the FIFO buffer is in normal mode with flits being transferred from the memory to the *data_out* line. After a few normal cycles, the *test_ctrl* is asserted high, switching the buffer to test mode. As long as the buffer is in test mode, no external data is allowed to be written to the buffer, or in other words, the buffer is locked for the test period. As a result, the input data line for the FIFO memory is switched from the external *data_in* line to *test_data* line available from the test circuit. At the switching instant, the flit which was in the process of being transferred to the *data_out* line is simultaneously read into the *Test Circuit*. However, a one clock cycle delay is created for the flit to move to the *data_out* line. This delay ensures that the flit is not lost during the switching instant and is properly received by the router which requests for it. The single cycle delay in the path of the traveling flit is created by the D-type flip flop and the multiplexer *m3* as shown in Figure 7.6 (a). The flit which is read in the test circuit is stored in a temporary register *temp* and the test process begins with this flit.

The test circuit consist of a test controller, storage registers (*temp*, *original* and *result*), logic circuit (comparator and inverter) and multiplexers as illustrated in Figure 7.6 (b). The operation

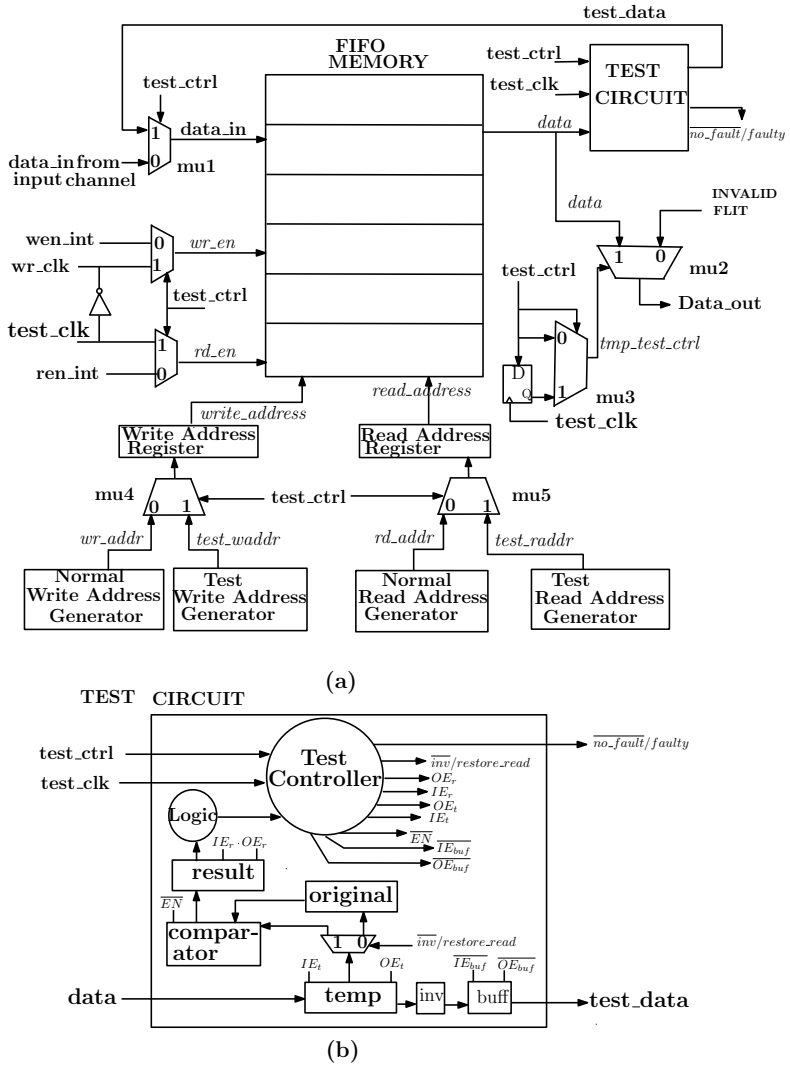


Figure 7.6: (a) Hardware implementation of the test process for the FIFO buffers (b) Implementation of test circuit

of the test circuit is explained with the help of Figure7.7. The three phases of the test process namely invert, restore and read, discussed in the last section are represented as three states. When the *test_ctrl* is asserted high, the test controller moves from normal state to invert state. In this state, the $\overline{inv}/restore\_refresh$ select line of the multiplexer is asserted low. This allows the data from the *data_in* line to be stored in *temp* and then backed-up in register *original* during read cycle of the *test_clk*. In the next write_cycle, the contents of *temp* are inverted and fed to the *test_data* line which writes it to the FIFO buffer location. The necessary input and output enable lines for the registers are provided by the test controller.

With *test_ctrl* asserted high, the test controller moves from invert state to the restore state at the end of one *test_clk* cycle. In restore state, the contents of the *temp* register during the read

cycle is delivered to the comparator by asserting the $\overline{inv}/restore\_refresh$ select line high. The comparator performs bit-wise ex-or operation on the contents of *temp* and *original* and stores the result in *result* register. The *result* register is then checked for an all-1 pattern. Any deviation from all one pattern is reported back to the test controller which asserts a high on the $\overline{no\_fault}/faulty$ denoting presence of fault in the FIFO buffer. Similar to invert state, during the write cycle, the contents of *temp* are inverted and put on the *test_data* line which writes them in the location under test of the FIFO memory.

At the end of the second *test_clk* cycle, with *test_ctrl* remaining high, the test controller moves from restore state to the read state. In this state, only a read operation is performed. The operations are similar to the operations performed during read cycle of the restore state, except that the contents of the *result* register is checked for all-0 pattern. Any deviation, is reported to the controller which asserts the $\overline{no\_fault}/faulty$ high.

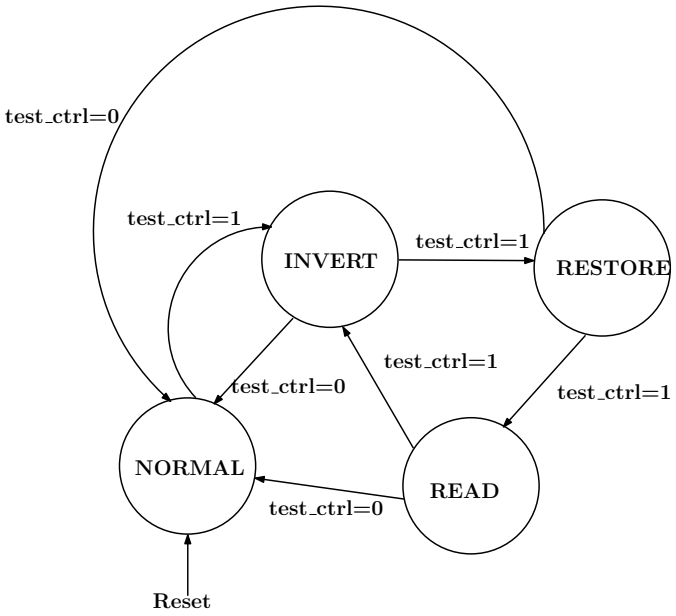


Figure 7.7: State diagram representing operation of the test controller

7.5 Experimental Results

The performance of the mesh based NoC after addition of the test circuit is investigated in terms of throughput using a System C based NoC simulator [57]. A prototype implementation of the proposed test circuit has been integrated into the router-channel interface. On-line Transparent SOA-MATS++ test is performed with synthetic self-similar data traffic. Test cost estimation of the router has been performed in terms of the silicon area overhead of the test hardware. The

router design considered in this chapter has been taken from [57]. Each router has the following configuration.

a) Number of global ports : 2 for corner routers, 3 for edge routers, 4 for others.

b) Number of virtual channels per link : 4

c) Number of flits per packet : 64

d) Arbitration : Round robin; Frequency : 1.5 GHz

e) Buffers : Nil in output channel.
However, for input channel, three different FIFO depth values have been considered. Three simulation runs are performed considering input channel FIFO buffer depth of 4, 6 and 8 respectively. All routers are assumed to operate at the same speed.

7.5.1 Area estimation of the test hardware

The router design considered for this paper is taken from [57]. Therefore, the area occupied by each router is obtained from the synthesis results provided in [57] and is shown in Table7.2.

Table 7.2: Area estimate of a router considered for the mesh type network [57]

Position	**Connectivity**	**Area (mm^2)**
Center	5	0.331
Edge	4	0.242
Corner	3	0.22

Depending on the connectivity, the routers of the mesh network considered in [57] have been classified as (i) center, having node degree 5, (ii) edge, having node degree 4, and (iii) corner, having node degree 3. The three position entries of Table 7.2 correspond to the three types of routers.

The proposed hardware for the test circuit has been described in Verilog HDL and synthesized using *Synopsys Design Vision* supporting 90nm CMOS technology. We used the same target technology as in [57] to ensure fair comparison of test hardware and router in terms of area. The area estimate of different modules in the test circuit is illustrated in Table 7.3.

Each input channel of a router requires the test hardware. Table 7.4 gives the total area occupied by additional circuitry in each input channel used for test purpose.

Table 7.3: Area estimate of test circuit

Module	**Area (μm^2)**
32-bit register with chip enable	679
multiplexer	230
comparator (xor)	288
address generator	35
controller	488

Table 7.4: Area occupied by test hardware for each input channel

Module	**No. of modules**	**Area (μm^2)**
register	4	2716
multiplexer	8	1840
comparator (xor)	1	288
address generator	2	70
controller	1	488
Total		5402

The area occupied by the additional test hardware is an overhead when compared to the area of each router. Table 7.5 gives the area overhead estimates considering the three different types of routers mentioned in [57]. Larger the connectivity of a router, more is the number of input channels present. Thus, a router centrally located in NoC would have more number of input channels than the ones at the side or corner. Since each input channel has a test circuit associated with it, the area overhead due to test circuit would be larger for a centrally located router than routers located at side or corner positions in a NoC. This situation is reflected in entries of Table 7.5.

From Table 7.5, it is inferred that the proposed on-line test hardware for the FIFO buffers incurs 7-9 % area overhead in each router. It may seem that for a NoC system with a number of routers, the area overhead due to test hardware would be significant. Thus, we tried to estimate the overall area occupied by test hardware in a NoC of a particular size. Such an estimate is provided for a NoC of size 4x8 and presented in Table 7.6. The overall area overhead due to test hardware is calculated using the following relation:

Table 7.5: Area overhead estimate of different routers located at different positions in a NoC

Router position	area occupied by test h/w (mm^2)	area overhead
center	0.0270	8.1%
edge	0.0216	8.9 %
corner	0.0162	7.3%

Area overhead = (*area occupied by test hardware*)/ (*area occupied by routers*) *100 %
Thus, the calculated area overhead comes out as
Area overhead = (0.7344/ 8.724)*100% = 8.41%
The above result suggests that the increase in area of the routers due to test hardware (area overhead) is 8.41%, which is comparable to that of area overhead associated with a single router. Next, we consider the area overhead for the entire NoC. The area occupied by routers in a NoC is an overhead compared to the area of the cores. Assuming each core in the 4x8 size NoC to be of size 1 mm^2, the area overhead for all routers without test hardware turns out to be 27.2% while with test hardware the router area overhead is 29.53%.

Table 7.6: Estimate of area occupied by test hardware for a 4x8 size NoC

Router position	No. of routers	router area (mm^2)	test h/w area (mm^2)
center	12	3.972	0.324
edge	16	3.872	0.3456
corner	4	0.88	0.0648
	Total	8.724	0.7344

7.5.2 Throughput estimation

For evaluating the performance of a NoC based network, an System C based cycle-accurate NoC simulator [57] has been utilized. The simulator works at the granularity of individual architectural components of the router. It supports mesochronous clocking strategy where the routers are driven by same clock frequency with varying phase. Synthetic self-similar traffic has been used during simulation, guided by the communication requirement of cores in the application. Self similar traffic has been observed in the burst traffic between on-chip modules in typical video and networking applications [77]. Detailed description of such traffic can be found in [94]. The

simulator has been utilized to compute the throughput of the network with and without the test circuit. The results are then compared to estimate the effect of inclusion of the test circuit on the performance of the network. The definition of *Throughput* considered in this chapter is the same as in [57], i.e. $Throughput = \frac{PC^* PL}{IP^* T}$
where *PC* (*total packets completed*) refers to number of packets that successfully arrive at their destination IP cores, *PL* (*packet length*) is measured in terms of number of flits, *IP* (*number of IP blocks*) is the number of IP blocks involved in the communication and *T*(*total time*) denotes the simulation time (in clock cycles). Hence, the throughput is represented as *flits/cycle/IP*.

Table 7.7 shows the throughput results for a mesh type NoC of size 4x8. Packets sent refers to number of packets sent from a source core during a particular application. Packets received is the number of packets received at the destination core after the required number of simulation cycles. The difference of packets sent and received gives the number of packets in transit, or in other words, flits that have not yet reached the destination core.

Simulations have been performed with self similar traffic with the routers having specifications as mentioned above. FIFO depth (number of locations of FIFO memory) is varied as 4, 6 and 8. Each simulation has been run for 200,000 clock cycles. As FIFO depth increases, more packets are stored in FIFO memory of the routers. As a result, the number of packets received at the destination core increases, eventually increasing the throughput as reflected in Table 7.7.

Then, we tried to investigate the effect on overall throughput by including the test circuit within the routers. Table 7.8 shows the throughput results for the same 4x8 size NoC with self-similar traffic. However, the routers considered in this case included the test circuit and tests were performed at periodic intervals. We varied the periodicity of tests to judge the effect of frequent testing on performance of the NoC. The periodicity of test was chosen judiciously to explore the effects due to frequent tests (5000ms), not so frequent (20,000ms) and delayed test period (100000 ms). Moreover, tests were repeated for varying FIFO depths.

From the definition of throughput mentioned earlier, it can be observed that the throughput of the NoC during an application run is influenced by the number of participating IP blocks (cores). Thus, we were motivated to study the effect of number of cores participating in the on-line test on overall throughput of the NoC. To perform the experiment, we partitioned the NoC in two partitions and allowed the SOA-MATS++ test to be operated on the partitions at different times. The throughput was estimated after both partitions were tested. The experiment was repeated for four partitions. The throughput estimation results for two partitions and four partitions are illustrated in Table 7.9 and Table 7.10 respectively. The inferences drawn from the results of Table 7.7, 7.8, 7.9 and 7.10 are discussed in the next subsection.

Table 7.7: Throughput estimation of the NoC without test circuit

FIFO depth	Packets sent	Packets received	Throughput
4	4908	4885	0.244
6	5646	5625	0.281
8	5974	5949	0.297

7.5.3 Analysis

Comparing the first row entries of Table 7.7 and Table 7.8, we observe that for FIFO depth 4, the throughput drops by 5.3% in case FIFO memory is tested after every 5000ms period. However, as periodicity of test is delayed, the throughput becomes comparable to result obtained when no tests were performed. This is reflected by comparing first row entry of Table 7.7 with second and third row entries of Table 7.8. Similar nature of test is observed for FIFO depths of 6 and 8 as reflected by the data in Table 7.8. Thus, it may be concluded that if on-line Transparent March tests are frequently performed on FIFO memory, the overall throughput of the NoC decreases. However, the interval between occurrence of two run-time permanent faults is large enough to avoid frequent test of the buffers.

When we consider performing test on two partitions of the NoC at different times, the throughput results after test were almost comparable with the results when no partitioning was done. No significant changes in the throughput results were obtained even by partitioning the NoC into four partitions. The probable reason for such results is that even if the NoC was partitioned into two or four partitions, the source and the destination cores were included in the same partition. Thus, all cores which participated during a test cycle belonged to the same partition. As a result, the throughput results obtained after partitioning the NoC remained same as results without partition.

7.6 Summary

This chapter highlighted the requirement of in-field testing of FIFO buffers present within the routers of the NoC infrastructure and discussed an efficient on-line test proposal for the same. The proposal applied a two pronged approach. First, a Transparent SOA-MATS++ test generation algorithm has been proposed that can detect run-time permanent faults developed in SRAM based FIFO memories. Next, the proposed Transparent test is utilized to perform on-line and periodic test of FIFO memory present within the routers of the NoC. Periodic testing of buffers

Table 7.8: Throughput estimation of the NoC with test circuit

FIFO depth	interval of test (ms)	Packets sent	Packets received	Throughput
4	5000	4749	4680	0.231
4	20000	4874	4842	0.241
4	100000	4929	4903	0.245
6	5000	5527	5466	0.270
6	20000	5660	5628	0.280
6	100000	5652	5637	0.281
8	5000	5942	5839	0.289
8	20000	5998	5944	0.296
8	100000	6078	6078	0.302

Table 7.9: Throughput estimation of the NoC having two partitions with different test start times

FIFO depth	Interval of test (ms)	Packets sent	Packets received	Throughput
4	5000	4792	4704	0.232
4	20000	4865	4823	0.239
4	100000	4899	4826	0.241
6	5000	5649	5560	0.275
6	20000	5706	5667	0.282
6	100000	5738	5715	0.285
8	5000	5821	5721	0.282
8	20000	5872	5822	0.289
8	100000	6024	6037	0.299

Table 7.10: Throughput estimation of the NoC having four partitions with different test start times

FIFO depth	Interval of test (ms)	Packets sent	Packets received	Throughput
4	5000	4783	4696	0.223
4	20000	4941	4896	0.244
4	100000	4945	4920	0.245
6	5000	5530	5443	0.268
6	20000	5703	5664	0.282
6	100000	5726	5699	0.284
8	5000	5899	5798	0.286
8	20000	5872	5822	0.293
8	100000	6024	6037	0.301

prevents accumulation of faults and also allows test of each location of the buffer. On-line testing allows modules of NoC which do not participate in test to function normally.

Simulation results reveal that periodic testing of FIFO buffers do not have much effect on the overall throughput of the NoC except when buffers are tested too frequently. Even when buffers are frequently tested, throughput drops only by 5%. The test hardware for the proposed on-line test of the FIFO buffers incurs 7-9 % area overhead in each router. However, when compared to the overall area of all routers in the NoC, the area overhead due to test hardware increases by only 8%. Thus, the proposed Transparent SOA-MATS++ test and its implementation in test of FIFO buffers of routers provides a cost effective solution for detection of run-time permanent faults in FIFO buffers of routers. The throughput results obtained after partitioning the NoC remained same as results without partition. As already discussed, the probable reason could be presence of sender and receiver core in the same partition.

The previous chapters have covered improved test techniques for memory cores interconnected using NoC, either by improving the test mechanism or by re-using on-chip resources for test and the present chapter covers testing of memory cores present in the NoC infrastructure. Thus, the focus of the thesis to provide a system level test solution for NoC based memory cores has been provided. The next chapter provides the summary of contributions of thesis and provides directions for future work.

Chapter 8

Conclusions and Future Work

The focus of the thesis has been to devise improved test techniques for NoC based memory systems which include SRAM or DRAM cores interconnected using NoC and FIFO buffers which are present within the routers of the NoC infrastructure. The objective of the presented work has been to find the research gap in the existing works related to the topic and bring about improvements in them by reducing the test cost. The test cost estimation has been done in terms of area overhead, test time and power dissipation during test.

Three directions of improvement have been proposed in the thesis. The first direction has been innovations in test architectures involving efficient re-use of the communication medium for test purpose while keeping test time and power under check, such as the distributed and hybrid testing scheme proposed in Chapter 4. The second direction of improvement has been the re-use of on-chip circuit for test purpose to perform both off-line and on-line test of DRAMs, such as the techniques proposed in Chapter 5 and 6. The third direction of improvement has been along the algorithmic way. The standard March based tests have been modified to transparent tests and then structured accordingly to suit their application on DRAMs and FIFO buffers as proposed in Chapter 7.

8.1 Summary of the Contributions

A chapter-wise summary of the contributions of the thesis is presented as follows.

8.1.1 NoC based MBIST

In Chapter 4, a two-level test architecture has been proposed for memory cores interconnected by a mesh type NoC. The contributions of the chapter are as follows:

a) *Test Architecture:* A distributed MBIST architecture has been proposed for testing memory cores interconnected using NoC. The memory cores form different groups based on distance and timing constraints and each group has a dedicated BIST controller. A fixed number of BIST controllers are placed at calculated locations and each controller is shared by a group of memory cores. The BIST controller performs utilizes a hybrid test technique where March test is perfromed on all on all the cores in a group parallely while the groups are tested in a pipeline fashion. The hybrid test technique and the distributed BIST achitecture allows the test of memory cores to be performed at much lesser time than required in dedicated BIST architectures.

b) *Memory Grouping Algorithm:* The efficient utilization of the distributed and hybrid test technique depends on the judicious choice of the memory grouping algorithm which is governed by a test objective. Initially the memory grouping problem is treated as a placement problem considering reduction of test instruction transport latency as the only objective. The locations of the BIST controllers are computed using the Particle Swarm Optimization(PSO) algorithm and memory cores are assigned to the controllers based on a greedy approach. Experimental results for transport latency obtained from the PSO based approach is compared with heuristic allocation technique of assigning cores which are physically close to the controllers for reduced test time and referred to as Neighbourhood Allocation (NA) technique.

c) *Power Aware Test Schedule:* A test schedule for the proposed BIST architecture has been proposed such that the test power is kept within the power budget. Experiments performed on ITC'02 benchmark circuit confirms that the proposed test schedule performs a more power constrained test as compared to dedicated BIST technique.

8.1.2 Re-using refresh for off-line test of DRAMs

Chapter 5 presents a BIST architecture for DRAMs has been proposed that utilizes the exisiting on-chip refresh circuitry of DRAM. The contributions are follows.

a) The refresh re-use technique overcomes the requirement of additional Design-For-Testability hardware as tests are performed via the on-chip refresh circuit.

b) The read cycles during the DRAM testing are avoided as read cycles get performed during refresh cyles. As a result, the entire time between two refresh cycles is allowed for write operation.

c) The increase in write cycle time is utilized in performing power aware test of a number of DRAM cores embedded in SoCs. Analytic predictions indicate that the refresh re-use technique when applied for testing a number of DRAMs, allows parallel write operation on a larger number of DRAMs within a given test power budget as compared to normal BIST approaches.

8.1.3 Refresh re-use for on-line test of DRAMs

Chapter 6 presents a proposal which involves extension of the refresh re-use based technique to perform performing periodic transparent testing of DRAMs. The contributions are as follows.

a) A transparent March test generation algorithm has been proposed for DRAMs targeting permanent faults developed during DRAM operation. The proposed algorithm generates a more efficient word-oriented transparent March tests compared to the conventional transparent test generation techniques by avoiding signature based prediction phase.

b) The read followed by write operations performed during the refresh burst cycles of the DRAM are re-used for the proposed transparent March tests. Re-using the the refresh cycles for test purpose avoids waiting for idle cycles of the processor to perform the test as required in other proposed online transparent test techniques. Re-using refresh allows periodic testing of DRAM without interruption and test finishes within a definite time.

c) A prototype implementation of the refresh re-use based test circuit is proposed. Re-using the refresh circuit overcomes requirement of additional DFT hardware. Therefore, the proposed refresh re-use based transparent test technique provides a cost effective solution by providing facility for periodic tests of DRAM without requiring costly test such as *ECC* and without additional test hardware.

8.1.4 Test of FIFO buffers in NoC routers

Chapter 7 provides a cost-effective solution for detection of run-time permanent faults in FIFO buffers of routers. An on-line transparent test technique has been proposed that is repeated periodically to performs active fault detection over the entire FIFO buffer. The contributions have been as follows.

a) A transparent SOA-MATS++ test generation algorithm is proposed targeting in-field permanent faults developed in SRAM based FIFO memories.

b) The proposed transparent test is utilized to perform online and periodic test of FIFO memory present within the routers of the NoC.

c) The performance of the NoC after addition of the test circuit is investigated in terms of throughput using cycle accurate SystemC based simulator. Simulation results show that periodic testing of FIFO buffers do not have much effect on the overall throughput of the NoC except when buffers are tested too frequently. Even when buffers are frequently tested, throughput drops only by 5%.

d) The area overhead of the test circuit is compared with respect to the area of an individual router and the overall area of the NoC infrastructure. The test hardware for the proposed on-line test of the FIFO buffers incurs 7-9% area overhead in each router. However, when compared with the overall area of all routers in the NoC, the area overhead due to test hardware increases by only 8%.

8.2 Directions of future work

The presented thesis aimed at providing an offline and online cost effective solution for testing NoC based memory cores. However, during the course of the research work, it has been realized that there have been issues that were left unattended and further research needs to be done. These issues have been listed as follows.

a) Experiments on the benchmark circuit for the proposed test schedule in Chapter 4 have been performed for only March elements with alternate read and write operations. The test schedule needs to be tested for more complex March tests involving elements consisting of more than one read write elements.

b) The test scheduling algorithm of Chapter 4 needs to be applied on a number of benchmark circuits involving more than two memory groups.

c) Testing without repair is incomplete. To this effect, the future work should be directed towards repair schemes for memories, including the FIFO buffers, detected with run-time permanent faults.

Bibliography

[1] On-Chip Networks Bibliography. http://www.cl.cam.ac.uk/~rdm34/onChipNetBib/browser.htm#Jantsch:2003:NOC. Accessed: 2014-05-30.

[2] Various Methods of DRAM Refresh. In *Technical Note TN-04-30*. Micron Technology, Inc., 1999.

[3] R. D. Adams. *High Performance Memory Testing: Design Principles, Fault Modeling and Self Test*. New York, USA, 2002.

[4] R. Aitken. A Modular Wrapper Enabling High Speed BIST and Repair for Small Wide Memories. In *Proceedings of International Test Conference*, pages 997 – 1005, October 2004.

[5] A. M. Amory, C. Lazzari, M. Lubaszewski, and F. G. Moraes. A New Test Scheduling Algorithm Based on Networks-on-Chip as Test Access Mechanisms. *Journal of Parallel Distributed Computing*, 71(5):675–686, 2011.

[6] S. Bahl and V. Srivastava. Self-Programmable Shared BIST for Testing Multiple Memories. In *13th European Test Symposium (ETS)*, pages 91 –96, May 2008.

[7] S. Barbagallo, M. L. Bodoni, D. Medina, G. de Blasio, M. Ferloni, F. Fummi, and D. Sciuto. A Parametric Design of a Built-in Self-Test FIFO Embedded Memory. In *Proceedings of the Workshop on Defect and Fault-Tolerance in VLSI Systems (DFT '96)*, pages 221–229, 1996.

[8] L. Benini and G. D. Micheli. Chapter 1 - Networks on Chip. In *Networks on Chips*, pages 1 – 22. Morgan Kaufmann, San Francisco, 2006.

[9] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and M. Bodoni. Programmable Built-in Self-Testing of Embedded RAM Clusters in System-on-Chip Architectures. *Communications Magazine, IEEE*, 41(9):90 – 97, September 2003.

[10] P. Bernardi, M. Grosso, M. Reorda, and Y. Zhang. A Programmable BIST for DRAM Testing and Diagnosis. In *Proceedings of International Test Conference*, pages 1 –10, November 2010.

[11] A. Bondavalli, S. Chiaradonna, F. D. Giandomenico, and F. Grandoni. Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults. *IEEE Transaction on Computers*, 49(3):230–245, 1998.

[12] S. Borri, M. Hage-Hassan, L. Dilillo, P. Girard, S. Pravossoudovitch, and A. Virazel. Analysis of Dynamic Faults in Embedded-SRAMs: Implications for Memory Test. *Journal of Electronic Testing*, 21(2):169–179, April 2005.

[13] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000.

[14] C. Cheng, C.-T. Huang, J.-R. Huang, C.-W. Wu, C.-J. Wey, and M.-C. Tsai. BRAINS: A BIST Compiler for Embedded Memories. In *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 299 –307, 2000.

[15] K.-L. Cheng, C.-M. Hsueh, J.-R. Huang, J.-C. Yeh, C.-T. Huang, and C.-W. Wu. Automatic Generation of Memory Built-In Self-Test Cores for System-on-Chip. In *Proceedings of the 10th Asian Test Symposium*, pages 91 –96, 2001.

[16] H. Cheung and S. Gupta. A BIST Methodology for Comprehensive Testing of RAM with Reduced Heat Dissipation. In *Proceedings of International Test Conference*, pages 386–395, Oct 1996.

[17] T.-F. Chien, W.-C. Chao, C.-M. Li, Y.-W. Chang, K.-Y. Liao, M.-T. Chang, M.-H. Tsai, and C.-M. Tseng. BIST Design Optimization for Large-Scale Embedded Memory Cores. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 197–200, November 2009.

[18] B. Cockburn and Y.-F. Sat. A Transparent Built-In Self-Test Scheme for Detecting Single V-Coupling Faults in RAMs. In *Proceedings of the International Workshop on Memory Technology, Design and Testing*, pages 119–124, 1994.

[19] B. Cockburn and Y.-F. Sat. Synthesized Transparent BIST for Detecting Scrambled Pattern-Sensitive Faults in RAMs. In *Proceedings of International Test Conference*, pages 23 –32, October 1995.

[20] C. Constantinescu. Impact of Intermittent Faults on Nanocomputing Devices. In *Proceedings of the Workshop on Dependable and Secure Nanocomputing (DSN 2007)*, 2007.

[21] F. Corno, M. Damiani, L. Impagliazzo, P. Prinetto, M. Rebaudengo, G.Sartore, and M. Reorda. On-Line Testing of An Off-The-Shelf Microprocessor Board for Safety-Critical Applications. In *Second European Dependable Computing Conference (EDCC-2)*, pages 190–201. 1996.

[22] E. Cota, L. Carro, F. Wagner, M. Lubaszewski, F. W. M. Lubaszewski, P. Depto, and E. Eltrica. BISTed Cores and Test Time Minimization in NOC-based Systems. In *Proceedings of International Workshop on Test Resource Partioning (TRP)*, pages 1–6, 2003.

[23] E. Cota, M. Kreutz, C. Zeferino, L. Carro, M. Lubaszewski, and A. Susin. The Impact of NoC Reuse on the Testing of Core-Based Systems. In *Proceedings of the 21st VLSI Test Symposium*, pages 128 – 133, April- May 2003.

[24] E. Cota and C. Liu. Constraint-Driven Test Scheduling for NoC-Based Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2465–2478, November 2006.

[25] W. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Annual Design Automation Conference*, pages 684–689, 2001.

[26] R. Dekker, F. Beenker, and L. Thijssen. A Realistic Fault Model and Test Algorithms for Static Random Access Memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(6):567–572, June 1990.

[27] T. J. Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. IBM Microelectronics Division, July 1997.

[28] L.-M. Denq and C.-W. Wu. A Hybrid BIST Scheme for Multiple Heterogeneous Embedded Memories. In *Proceedings of the 16th Asian Test Symposium*, pages 349–354, October 2007.

[29] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. B. Hage-Hassan. Data Retention Fault in SRAM Memories: Analysis and Detection Procedures. In *Proceedings of the 31st VLSI Test Symposium (VTS)*, pages 183–188, 2005.

[30] B. H. Fang and N. Nicolici. Power-Constrained Embedded Memory BIST Architecture. In *Proceedings of the 18th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 451–458, November 2003.

[31] S. Ghosh and K. Roy. Parameter Variation Tolerance and Error Resiliency: New Design Paradigm for the Nanoscale Era. *Proceedings of the IEEE*, 98(10):1718–1751, 2010.

[32] A. Goor. *Testing Semiconductor Memories: Theory and Practice*. J. Wiley & Sons, 1991.

[33] M. Gottscho, A. Kagalwalla, and P. Gupta. Power Variability in Contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(2):37–40, 2012.

[34] C. Grecu, P. P. Pande, B. Wang, A. Ivanov, and R. Saleh. Methodologies and Algorithms for Testing Switch-Based NoC Interconnects. In *Proceedings of the 20th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 238–246, 2005.

[35] A. R. Guner and M. Sevkli. A discrete particle swarm optimization algorithm for uncapacitated facility location problem. *Journal of Artificial Evol. App.*, 2008:10:1–10:9, January 2008.

[36] S. Hamdioui. *Testing Static Random Access Memories: Defects, Fault Models and Test Patterns*. Frontiers in Electronic Testing. Springer, 2010.

[37] S. Hamdioui, Z. Al-ars, Ad, J. Van, D. Goor, M. Rodgers, and A. Ivanov. Dynamic Faults in Random-Access-Memories:Concept, Fault Models and Tests. *Journal of Electronic Testing: Theory and Applications*, 19(2):2003, 2003.

[38] S. Hamdioui, A. J. Van de Goor, and M. Rodgers. March SS: A Test for All Static Simple RAM Faults. In *Proceedings of the IEEE International Workshop on Memory Technology, Design and Testing (MTDT 2002)*, pages 95–100, 2002.

[39] S. Hamdioui, R. Wadsworth, J. D. Reyes, and A. J. Van De Goor. Memory fault modeling trends: A case study. *Journal of Electronic Testing : Theory and Applications*, 20(3):245–255, June 2004.

[40] R. Hassan, B. E. Cohanim, and O. L. de Weck. Comparison of Particle Swarm Optimization and the Genetic Algorithm. In *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, number AIAA-2005-1897, Austin, Texas, April 18-21 2005. American Institute of Aeronautics and Astronautics.

[41] S. Hellebrand, H. Wunderlich, A. Ivaniuk, Y. Klimets, and V. Yarmolik. Efficient Online and Offline Testing of Embedded DRAMs. *IEEE Transactions on Computers*, 51(7):801–809, July 2002.

[42] S. Hellebrand, H.-J. Wunderlich, A. Ivaniuk, Y. Klimets, and V. Yarmolik. Error Detecting Refreshment for Embedded DRAMs. In *Proceedings of the 17th VLSI Test Symposium*, pages 384 –390, April 1999.

[43] M. Herve, P. Almeida, F. Kastensmidt, E. Cota, and M. Lubaszewski. Concurrent Test of Network-on-Chip Interconnects and Routers. In *11th Latin American Test Workshop (LATW)*, pages 1–6, 2010.

[44] D. Huang, W.-B. Jone, and S. Das. An Efficient Parallel Transparent BIST Method for Multiple Embedded Memory Buffers. In *Proceedings of the Fourteenth International Conference onVLSI Design*, pages 379–384, 2001.

[45] D. C. Huang and W. B. Jone. A Parallel Transparent BIST Method for Embedded Memory Arrays by Tolerating Redundant Operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(5):617–628, November 2006.

[46] Y.-J. Huang, C.-W. Chou, and J.-F. Li. A Low-Cost Built-in Self-Test Scheme for an Array of Memories. In *Proceedings of the 15th European Test Symposium (ETS)*, pages 75 –80, May 2010.

[47] Y.-J. Huang and J.-F. Li. A Low-Cost Pipelined BIST Scheme for Homogeneous RAMs in Multicore Chips. In *Proceedings of the 17th Asian Test Symposium*, pages 357–362, 2008.

[48] Y.-J. Huang, Y.-C. You, and J.-F. Li. Enhanced IEEE 1500 Test Wrapper for Testing Small RAMs in SOCs. In *Proceedings of the International SOC Conference (SOCC)*, pages 236 –240, September 2010.

[49] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic Rays Don't Strike Twice: Understanding the Nature of DRAM Errors and the Implications for System Design. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 111–122, 2012.

[50] J. e. Inoue. Parallel Testing Technology for VLSI Memories. In *Proceedings of International Test Conference*, pages 1,066–1,071, 1987.

[51] W. Jone, D. Huang, S. Wu, and K. Lee. An Efficient BIST Method for Distributed Small Buffers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):512 –515, August 2002.

[52] M. R. Kakoee, V. Bertacco, and L. Benini. A Distributed and Topology-Agnostic Approach for On-Line NoC Testing. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, pages 113–120, 2011.

[53] M. G. Karpovsky, A. J. van de Goor, and V. N. Yarmolik. Pseudo-Exhaustive Word-oriented DRAM Testing. In *Proceedings of the European Design and Test Conference*, 1995.

[54] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942 –1948, November/December 1995.

[55] A. Kokrady, C. Ravikumar, and N. Chandrachoodan. Layout-Aware and Programmable Memory BIST Synthesis for Nanoscale System-on-Chip Designs. In *Proceedings of the 17th Asian Test Symposium (ATS '08)*, pages 351 –356, November 2008.

[56] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A Network-on-Chip Architecture and Design Methodology. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, ISVLSI '02, pages 105–112, 2002.

[57] S. Kundu, J. Soumya, and S. Chattopadhyay. Design and Evaluation of Mesh-of-Tree Based Network-on-Chip Using Virtual Channel Router. *Microprocessors and Microsystems*, 36(6):471–488, August 2012.

[58] J.-F. Li. Transparent-Test Methodologies for Random Access Memories Without/With ECC. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 26(10):1888–1893, 2007.

[59] H.-N. Liu, Y.-J. Huang, and J.-F. Li. Memory Built-In Self Test In Multicore Chips With Mesh-Based Networks. *IEEE Micro*, 29(5):46–55, September-October 2009.

[60] H.-C. Lu and J.-F. Li. A Programmable Online/Off-Line Built-In Self-Test Scheme for RAMs with ECC. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 1997–2000, 2009.

[61] X. W. Luang-Terng Wang, Chenh-Wen Wu, editor. *VLSI Test Principles and Architectures*. Morgan Kaufmann, New York, NY, USA, 2006.

[62] A. Maheshwari, W. Burleson, and R. Tessier. Trading off Transient Fault Tolerance and Power Consumption in Deep Submicron (DSM) VLSI Circuits. *IEEE Transactions on Very Large Scale Integrated Systems*, 12(3):299–311, March 2004.

[63] E. Marinissen, V. Iyengar, and K. Chakrabarty. A Set of Benchmarks for Modular Testing of SoCs. In *Proceedings of the International Test Conference*, pages 519 – 528, 2002.

[64] M.G.Karpovsky and V.N.Yarmolik. Transparent Memory BIST. In *Proceedings of the International Workshop on Memory Technology, Design and Testing*, pages 106–111, 1994.

[65] G. Micheli, P. Pande, C. G. A. Ivanov, and R. Saleh. Design, Synthesis and Test of Networks-on-Chips. *IEEE Design and Test of Computers*, 22(5):404–413, September 2005.

[66] T. Mitra. Dynamic Random Access Memory: A Survey. Research Proficiency Examination Report, SUNY at Stony Brook, March 1999.

[67] M. Miyazaki, T. Yoneda, and H. Fujiwara. A Memory Grouping Method for Sharing Memory BIST Logic. In *Proceedings of the Asia and South Pacific Conference on Design Automation (ASPDAC)*, pages 671–676, January 2006.

[68] S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt. Cache Scrubbing in Microprocessors: Myth or Necessity? In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 37–42, 2004.

[69] B. Nadeau-Dostie, A. Silburt, and V. Agarwal. A Serial Interfacing Technique for Built-In and External Testing of Embedded Memories. In *Proceedings of the Custom Integrated Circuits Conference*, pages 22.2/1 –22.2/5, May 1989.

[70] G. Nazarian. On-Line Testing of Routers in Networks-on-Chip. Master's thesis, Electrical Engineering, Delft University of Technology, 2008.

[71] M. Nicolaidis. Theory of Transparent BIST for RAMs. *IEEE Transactions on Computers*, 45(10):1141–1156, October 1996.

[72] R. Nourmandi-Pour, N. Mousavian, and A. Khadem-Zadeh. BIST for Network-on-Chip Communication Infrastructure Based on Combination of Extended IEEE 1149.1 and IEEE 1500 Standards. *Microelectronics Journal*, 42(5):667–680, 2011.

[73] T. Ohsawa, T. Furuyama, Y. Watanabe, H. Tanaka, N. Kushiyama, K. Tsuchida, Y. Nagahama, S. Yamano, T. Tanaka, S. Shinozaki, and K. Natori. A 60-ns 4-mbit CMOS DRAM with Built-in Self Test Function. *IEEE Journal of Solid-State Circuits*, 22(5):663 – 668, October 1987.

[74] J. Pouget, E. Larsson, and Z. Peng. Multiple-Constraint Driven System-on-Chip Test Time Optimization. *Journal of Electronic Testing: Theory and Applications*, 21(6):599–611, 2005.

[75] P. Prinetto, F. Corno, and M. Sonza Reorda. Fault Tolerant and BIST Design of a FIFO Cell. In *Proceedings of the Conference on European Design Automation*, pages 233–238, 1996.

[76] R. Sable, R. Saraf, R. Parekhji, and A. Chandorkar. Built-in Self-Test Technique for Selective Detection of Neighbourhood Pattern Sensitive Faults in Memories. In *Proceedings of 17th International Conference on VLSI Design*, pages 753 – 756, 2004.

[77] P. Sahu, T. Shah, K. Manna, and S. Chattopadhyay. Application Mapping Onto Mesh-Based Network-on-Chip Using Discrete Particle Swarm Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):300–312, February 2014.

[78] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM Errors in the Wild : A Large-Scale Field Study. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, pages 193–204, 2009.

[79] D. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi. Issues in the Development of a Practical NoC: The Proteo Concept. *Integration VLSI Journal*, 38(1):95–105, October 2004.

[80] S. P. Singh, S. Bhoj, D. Balasubramaniam, T. Nagda, D. Bhatia, and P. Balsara. Network Interface for NoC Based Architectures. *International Journal of Electronics*, 94(5):531–547, 2007.

[81] M. Spica and T. M. Mak. Do We Need Anything More Than Single Bit Error Correction (ECC)? In *Records of the International Workshop on Memory Technology, Design and Testing*, pages 111–116, 2004.

[82] T. Sridhar. A New Parallel Test Approach for Large Memories. *IEEE Design and Test*, 3(4):15–22, July 1986.

[83] V. Sridharan and D. Liberty. A Study of DRAM Failures in the Field. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, November 2012.

[84] P. Teehan, M. Greenstreet, and G. Lemieux. A Survey and Taxonomy of GALS Design Styles. *IEEE Design and Test of Computers*, 24(5):418–428, 2007.

[85] M. Tehranipoor, M. Nourani, and N. Ahmed. Low Transition LFSR for BIST-Based Applications. In *Proceedings of the 14th Asian Test Symposium*, ATS '05, pages 138–143, 2005.

[86] K. Thaller. A Highly-Efficient Transparent Online Memory Test. In *Proceedings of IEEE International Test Conference*, pages 230–239, November 2001.

[87] K. Thaller and A. Steininger. A Transparent Online Memory Test for Simultaneous Detection of Functional Faults and Soft Errors in Memories. *IEEE Transactions on Reliability*, 52(4):413–422, 2003.

[88] A. van de Goor, M. Abadir, and A. Carlin. Minimal Test for Coupling Faults in Word-Oriented Memories. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 944–948, 2002.

[89] A. van de Goor and I. Tlili. March Tests for Word-oriented Memories. In *Proceedings of Design, Automation and Test in Europe*, pages 501 –508, February 1998.

[90] A. J. van de Goor. *Testing Semiconductor Memories: Theory and Practice*. John Wiley & Sons, Inc., New York, NY, USA, 1991.

[91] A. J. Van de Goor. An Industrial Evaluation of DRAM Tests. *IEEE Design Test of Computers*, 21(5):430–440, September 2004.

[92] A. J. Van de Goor, I. Schanstra, and Y. Zorian. Functional Test for Shifting-Type FIFOs. In *Proceedings of European Design and Test Conference*, pages 133–138, 1995.

[93] A. J. Van de Goor and Y. Zorian. Functional Tests for Arbitration SRAM-Type FIFOs. In *Proceedings of First Asian Test Symposium (ATS)*, pages 96–101, September 1992.

[94] G. Varatkar and R. Marculescu. On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, January 2004.

[95] B. Wang and Q. Xu. Test/Repair Area Overhead Reduction for Small Embedded SRAMs. In *Procedings of 15th Asian Test Symposium*, pages 37 –44, November 2006.

[96] C.-W. Wang, R.-S. Tzeng, C.-F. Wu, C.-T. Huang, C.-W. Wu, S.-Y. Huang, S.-H. Lin, and H.-P. Wang. A Built-in Self-Test and Self-Diagnosis Scheme for Heterogeneous SRAM Clusters. In *Proceedings of the Proceedings of 10th Asian Test Symposium (ATS)*, pages 103 –108, 2001.

[97] J. Wei, L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. Comparing the Effects of Intermittent and Transient Hardware Faults on Programs. In *Proceedings of the 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 53–58, 2011.

[98] P. Wielage, E. J. Marinissen, M. Altheimer, and C. Wouters. Design and DFT of a High-Speed Area-Effficient Embedded Asynchronous FIFO. In *Proceedings of Design Automation and Test in Europe (DATE)*, pages 853–858, 2007.

[99] Y. Wu and A. Ivanov. Low Power SoC Memory BIST. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 197–205, October 2006.

[100] S. Yarmolik and V. Yarmolik. Modified Gray and Counter Sequences for Memory Test Address Generation. In *Proceedings of the International Conference on Mixed Design of Integrated Circuits and System*, pages 572 –576, June 2006.

[101] V. Yarmolik, S. Hellebrand, and H.-J. Wunderlich. Self-Aadjusting Output Data Cmpression: An Efficient BIST Technique for RAMs. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 173–179, February 1998.

[102] V. N. Yarmolik, I. V. Bykov, S. Hellebrand, and H.-J. Wunderlich. Transparent Word-Oriented Memory BIST Based on Symmetric March Algorithms. In *Third European Dependable Computing Conference (EDCC-3)*, pages 339–350, 1999.

[103] D. Yeh, L.-S. Peh, S. Borkar, J. Darringer, A. Agarwal, and W.-m. Hwu. Thousand-Core Chips. *IEEE Design Test of Computers*, 25(3):272 –278, May-June 2008.

[104] T. Yoneda, Y. Fukuda, and H. Fujiwara. Test Scheduling for Memory Cores with Built-In Self-Repair. In *Proceedings of the 16th Asian Test Symposium (ATS '07)*, pages 199 –206, oct. 2007.

[105] D. H. Yoon and M. Erez. Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 116–127, 2009.

[106] Y. You and J. Hayes. A Self-Testing Dynamic RAM Chip. *IEEE Journal of Solid-State Circuits*, 20(1):428 –435, February 1985.

[107] L. Zaourar, J. Chentoufi, Y. Kieffer, A. Wenzel, and F. Grandvaux. A Shared BIST Optimization Methodology for Memory Test. In *Proceedings of the 15th European Test Symposium (ETS)*, page 255, May 2010.

[108] L. Zaourar, Y. Kieffer, and A. Wenzel. A Multi-Objective Optimization for Memory BIST Sharing using a Genetic Algorithm. In *Proceeedings of the 17th International On-Line Testing Symposium (IOLTS)*, pages 73 –78, July 2011.

[109] Y. Zorian, A. J. van de Goor, and I. Schanstra. An Effective BIST Scheme for Ring-Address Type FIFOs. In *Proceedings of International Test Conference (ITC)*, pages 378–387, 1994.