Fault Models, Test Algorithms and Embedded Test Techniques for DRAM Circuits

George Sfikas

Ph.D. Dissertation

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ioannina, October 2015

FAULT MODELS, TEST ALGORITHMS AND EMBEDDED TEST TECHNIQUES FOR DRAM CIRCUITS

DISSERTATION

submitted to the Examination Commission,

designated by the General Assembly of Special Composition of
the Department of Computer Science and Engineering
of University of Ioannina

by

Yiorgos Sfikas

in partial fulfillment of the requirements

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

October 2015

ΜΟΝΤΕΛΑ ΣΦΑΛΜΑΤΩΝ, ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΕΝΣΩΜΑΤΩΜΕΝΕΣ ΤΕΧΝΙΚΕΣ ΕΛΕΓΧΟΥ ΟΡΘΗΣ ΛΕΙΤΟΥΡΓΙΑΣ ΚΥΚΛΩΜΑΤΩΝ ΜΝΗΜΩΝ DRAM

Η ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

υποβάλλεται στην

ορισθείσα από τη Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Μηχανικών Η/Υ και Πληροφορικής
Εξεταστική Επιτροπή

από τον

Γεώργιο Σφήκα

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ

Οκτώβριος 2015

COMMITTEES

Advisory Committee

- Yiorgos Tsiatouhas, Associate Professor at the Department of Computer Science and Engineering, University of Ioannina (Supervisor).
- Chrysovalantis Kavousianos, Associate Professor at the Department of Computer Science and Engineering, University of Ioannina.
- Aristidis Efthimiou, Assistant Professor at the Department of Computer Science and Engineering, University of Ioannina.

Examination Committee

- Yiorgos Tsiatouhas, Associate Professor at the Department of Computer Science and Engineering, University of Ioannina.
- Chrysovalantis Kavousianos, Associate Professor at the Department of Computer Science and Engineering, University of Ioannina.
- Aristidis Efthimiou, Assistant Professor at the Department of Computer Science and Engineering, University of Ioannina.
- Dimitrios Nikolos, Full Professor at the Department of Computer Engineering and Informatics, University of Patras.
- Angela Arapoyanni, Full Professor at the Department of Informatics and Telecommunications, University of Athens.
- Dimitrios Gizopoulos, Full Professor at the Department of Informatics and Telecommunications, University of Athens.
- Said Hamdioui, Full Professor at the Department of Computer Engineering, Delft University of Technology (The Netherlands).

DEDICATION

To my family and friends

ACKNOWLEDGEMENTS

First I would like to thank my supervisor, Prof. Y. Tsiatouhas for his guidance, support, encouraging, confidence in me and especially his patience.

I would like to sincerely thank the other members of the Advisory Committee, Prof. C. Kavousianos and A. Efthimiou, for the encouraging and the collaboration we had all these years.

I thank Prof. S. Hamdioui for serving as a member of the Examination Committee and for his valuable input and the discussions on many issues related to this research.

I thank Prof. D. Nikolos, A. Arapoyanni and D. Gizopoulos for serving as members of the Examination Committee.

I would also like to express my appreciation to the people who worked for the *Heracleitus II* program, for their continuous support and help.

Also, I want to thank my colleagues V. Tenentes, F. Vartziotis, P. Georgiou and G. Papatheodorou for the refreshing discussions and the collaboration we had all these years.

Last but not least, I would like to thank my family and friends for being the most important part of my life.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

INDEX

ΠΕΡΙΛΗΨΗ

Γεώργιος Σφήκας του Ιωάννη και της Βασιλικής
PhD, Τμήμα Μηχανικών Η/Υ και Πληροφορικής
Πανεπιστήμιο Ιωαννίνων
Οκτώβριος 2015
Τίτλος Διατριβής: Μοντέλα Σφαλμάτων, Αλγόριθμοι και Ενσωματωμένες Τεχνικές Ελέγχου Ορθής Λειτουργίας Κυκλωμάτων Μνημών DRAM
Επιβλέπων: Γεώργιος Τσιατούχας

Τις τελευταίες δεκαετίες οι ηλεκτρονικές συσκευές έχουν γίνει αναπόσπαστο κομμάτι της καθημερινότητας. Αυτό οφείλεται κυρίως στη ραγδαία πρόοδο της τεχνολογίας κατασκευής Ολοκληρωμένων Κυκλωμάτων (Ο.Κ.), η οποία επιτρέπει τη σμίκρυνση των διαστάσεων των ηλεκτρονικών κυκλωματικών στοιχείων και την ολοκλήρωση όλο και περισσότερων ηλεκτρονικών διατάξεων σε μια μικρή επιφάνεια πυριτίου. Ως άμεσο αποτέλεσμα αυτής της προόδου, τα σύγχρονα ηλεκτρονικά συστήματα συνδυάζουν εξαιρετικές επιδόσεις σε υπολογιστική ισχύ και αποθηκευτικό χώρο, φορητότητα και ικανοποιητική αυτονομία, ενώ ταυτόχρονα το χαμηλό κόστος τα κάνει προσιτά σε όλο σχεδόν το αγοραστικό κοινό.

Η εξέλιξη της τεχνολογίας κατασκευής ολοκληρωμένων κυκλωμάτων υπήρξε σχεδόν σταθερή για αρκετές δεκαετίες. Ταυτόχρονα όμως διάφοροι παράγοντες που δυσχεραίνουν την περεταίρω πρόοδο αυτής της τεχνολογίας κάνουν ολο και περισσότερο αισθητή την παρουσία τους. Ένας από τους βασικότερους παράγοντες είναι η ραγδαία αύξηση της δυσκολίας Ελέγχου Ορθής Λειτουργίας των ολοκληρωμένων κυκλωμάτων. Έλεγχος Ορθής Λειτουργίας (Ε.Ο.Λ.) είναι η διαδικασία που πραγματοποιείται στα ολοκληρωμένα κυκλώματα μετά την κατασκευή τους προκειμένου να διαπιστωθεί αν λειτουργούν σωστά και σύμφωνα με τις προδιαγραφές.

Οι Δυναμικές Μνήμες Τυχαίας Προσπέλασης (Dynamic Random Access Memories – DRAMs) είναι ανάμεσα στα πιο κρίσιμα μέρη των σύγχρονων ψηφιακών συστημάτων γιατί παίζουν καθοριστικό ρόλο τόσο από πλευράς επιδόσεων όσο και

από πλευράς αξιοπιστίας ενός συστήματος. Σε ότι αφορά την αξιοπιστία ενός ψηφιακού συστήματος, η αστοχία της κύριας μνήμης, η οποία είναι σχεδόν πάντα τύπου DRAM, είναι μια από τις συχνότερες αιτίες αστοχίας του συστήματος. Επομένως η αξιοπιστία των μνημών DRAM είναι κρίσιμη. Επιπρόσθετα, όπως συμβαίνει και με τους άλλους τύπους ολοκληρωμένων κυκλωμάτων, τα προβλήματα αξιοπιστίας των ολοκληρωμένων κυκλωμάτων μνημών DRAM αυξάνονται με ρυθμό ταχύτερο ακόμα και από το ρυθμό που ακολουθεί η εξέλιξη της τεχνολογίας κατασκευής ολοκληρωμένων κυκλωμάτων. Κατά συνέπεια, η ανάγκη για ανάπτυξη νέων αποτελεσματικότερων και πιο αξιόπιστων αλγορίθμων Ε.Ο.Λ. μνημών DRAM είναι επιτακτική.

Τα προβλήματα στον ΕΟΛ και στην αξιοπιστία των ολοκληρωμένων κυκλωμάτων οφείλονται κυρίως στη σμίκρυνση των διαστάσεων των στοιχειωδών κυκλωματικών στοιχείων. Στις DRAMs η σμίκρυνση των διαστάσεων των κυττάρων μνήμης, και των μεταξύ τους αποστάσεων επιφέρει ανεπιθύμητες επιδράσεις στη λειτουργία της μνήμης. Μία από τις σημαντικότερες είναι η αυξημένη αλληλεπίδραση μεταξύ γειτονικών κυττάρων μνήμης. Αυτή η αλληλεπίδραση μπορεί να προκαλέσει περίπλοκες εσφαλμένες συμπεριφορές οι οποίες συχνά είναι δύσκολο να εντοπιστούν κατά τη διάρκεια του Ε.Ο.Λ. καθώς πολύ συχνά εκδηλώνονται μόνο κάτω από συγκεκριμένες συνθήκες λειτουργίας (π.χ. όταν τα γειτονικά κύτταρα μνήμης βρίσκονται σε συγκεκριμμένη λογική κατάσταση).

Η αλληλεπίδραση μεταξύ των γειτονικών κυττάρων σε DRAMs και τα προβλήματα που δημιουργεί στον Ε.Ο.Λ. προσεγγίζονται σε αυτή τη διατριβή με δύο διαφορετικές μεθοδολογίες. Στην πρώτη ακολουθούμε ένα υπάρχον μοντέλο σφαλμάτων που περιγράφει τις αλληλεπιδράσεις μεταξυ γειτονικών κυττάρων μνήμης, το NPSF. Βασιζόμενοι σε αυτό το μοντέλο αναπτύσσουμε ένα νέο αλγόριθμο Ε.Ο.Λ. ο οποίος επιτυγχάνει μείωση κόστους Ε.Ο.Λ. σε χρόνο εφαρμογής κατά 57.7% σε σχέση με υπάρχοντες αλγορίθμους που καλύπτουν τα ίδια σφάλματα. Ταυτόχρονα προτείνουμε αλγορίθμους που καλύπτουν τις περιπτώσεις όπου τα NPSF σφάλματα συνδυάζονται με την επίδραση της Γραμμής Δεδομένων (Bit-Line influence) και της χωρητικής σύζευξης μεταξύ των Γραμμών Ενεργοποίησης (Word-Line capacitive coupling). Στη δεύτερη μεθοδολογία αναπτύσσουμε ένα νέο μοντέλο σφαλμάτων, το Μοντέλο Σφαλμάτων Διαρροών και Μεταβάσεων Γειτονιάς (*Neighborhood Leakage and Transition Fault – NLTF*), το οποίο στοχεύει

συγκεκριμμένους γνωστούς μηχανισμούς αλληλεπίδρασης. Ο αλγόριθμος Ε.Ο.Λ. που προκύπτει από το νέο μοντέλο μειώνει το κόστος σε χρόνο εφαρμογής κατά 87% σε σχέση με υπάρχοντες αλγορίθμους Ε.Ο.Λ. που καλύπτουν τα ίδια σφάλματα.

Άλλη μια δυσκολία στον Ε.Ο.Λ. των DRAM μνημών είναι το γεγονός ότι ακόμα και το πιο απλό κατασκευαστικό ελάττωμα μπορεί να προκαλέσει περίπλοκη εσφαλμένη συμπεριφορά, η οποία συχνά θα είναι δύσκολο να ανιχνευθεί κατά τη διάρκεια των διεργασιών Ε.Ο.Λ. Ο κύριος λόγος που συμβαίνει αυτό είναι ότι οι DRAM μνήμες είναι στην πράξη αναλογικά (και όχι ψηφιακά) ηλεκτρονικά κυκλώματα. Προσομοιώσεις της λειτουργίας μιας DRAM με σφάλμα αντιστατικού ανοιχτοκυκλώματος (resistive open defect) οι οποίες παρουσιάζονται στην παρούσα διατριβή καταδεικνύουν αυτή την περίπλοκη συμπεριφορά. Επιπρόσθετα παρατηρήθηκε και αναλύθηκε για πρώτη φορά ένα νέο φαινόμενο, η *συσσώρευση φορτίου* (*charge accumulation*) το οποίο επηρρεάζει σημαντικά τη διαδικασία Ε.Ο.Λ. Βασισμένοι στα ερευνητικά μας αποτελέσματα προτείνουμε ένα νέο χαμηλού κόστους αλγόριθμο Ε.Ο.Λ. ο οποίος παρέχει βελτιωμένη κάλυψη των σφαλμάτων ανοιχτοκύκλωσης.

Μιά από τις πιο ελκυστικές λύσεις στον Ε.Ο.Λ. είναι ο ενσωματωμένος αυτοέλεγχος (Built In Self Test – BIST), ο οποίος έχει κερδίσει μεγάλο ενδιαφέρον τις δύο τελευταίες δεκαετίες. Σε αυτή την κατεύθυνση αναπτύξαμε ένα κύκλωμα ενσωματωμένου αυτοελέγχου το οποίο χρησιμοποιεί τον αλγόριθμο Ε.Ο.Λ. του NLTF μοντέλου σφαλμάτων. Η λειτουργικότητα του κυκλώματος επιβεβαιώθηκε μέσω προσομοιώσεων. Αυτή η υλοποίηση του NLTF αλγορίθμου καταδυκνείει πως είναι δυνατό να ενσωματωθούν σε κύκλωμα ενσωματωμένου αυτοελέγχου αποτελεσματικά και με χαμηλό κόστος σε επιφάνεια πυριτίου ακόμα και περίπλοκοι αλγόριθμοι Ε.Ο.Λ.

Τέλος, μιά από τις πιο ελκυστικές λύσεις στον Ε.Ο.Λ. είναι ο ενσωματωμένος αυτοέλεγχος (Built In Self Test – BIST), ο οποίος έχει προσελκύσει μεγάλο ενδιαφέρον τις δύο τελευταίες δεκαετίες. Σε αυτή την κατεύθυνση αναπτύξαμε ένα κύκλωμα ενσωματωμένου αυτοελέγχου DRAM το οποίο υλοποιεί τον αλγόριθμο Ε.Ο.Λ. για το NLTF μοντέλο σφαλμάτων. Η λειτουργικότητα του κυκλώματος επιβεβαιώθηκε μέσω προσομοιώσεων. Αυτή η υλοποίηση του NLTF αλγορίθμου κατέδειξε πως είναι εφικτό να ενσωματωθούν σε κύκλωμα ενσωματωμένου

αυτοελέγχου, αποτελεσματικά και με χαμηλό κόστος σε επιφάνεια πυριτίου, περίπλοκοι αλγόριθμοι Ε.Ο.Λ. για κυκλώματα μνημών DRAM.

ABSTRACT

Yiorgos I. Sfikas.
PhD, Computer Science & Engineering Department.
University of Ioannina, Greece.
October 2015.
Title of Dissertation: Fault Models, Test Algorithms and Embedded Test Techniques for DRAM Circuits.
Thesis Supervisor: Yiorgos Tsiatouhas

Due to the revolutionary progress in the manufacturing process of Integrated Circuits (ICs) the last decades, electronic systems have become a part of everyday life. The direct results of this progress are the increased computing and storage capability of electronic systems, at an affordable or even low cost, and the mobility. However, although this progress rate has been constantly high for almost five decades, there are various threats to the further evolution of semiconductor technologies. One of the greatest threats is the rapidly increasing difficulty in testing ICs.

Dynamic Random Access Memories (DRAMs) are one of the most important parts in digital systems, both from a performance or a system failure perspective. Thus, their reliability is critical. Moreover, like in all IC technologies, the reliability issues grow more rapidly than the evolution of the manufacturing processes. Consequently, even if the manufacturing evolution is important, the development of new, more efficient and more reliable testing solutions turns out to be of equal importance.

The problems in testing and reliability of ICs mainly stem from the dimension shrinking of electronic devices aiming to scale up their integration in a small silicon area. In DRAMs, the shrinking of memory cells' dimensions and their in-between distances arise various undesired side effects. Among the most important side effects is the increased interaction between neighbouring cells. This interaction can cause complex faulty behaviors that are frequently hard to be detected since they appear

only under the presence of specific conditions (e.g. the neighbouring cells are at a certain state).

The neighbouring cell interaction issue is addressed in this dissertation with two different approaches. In the first approach, we refine an existing fault model that deals with neighbouring cell interactions, the NPSF model, in order to provide test solutions with an acceptable cost in test application time. The test application time reduction achieved by our new test algorithm is 57.7% with respect to well known test algorithms that cover these faults. At the same time we provide test solutions for the cases where the NPSF faults are combined with the Bit-Line influence and Word-Line capacitive coupling related faults. In the second approach, we propose a new fault model, the *Neighborhood Leakage and Transition Fault – NLTF model*, which targets specific well known interaction mechanisms. The test solution that derived from this new fault model further reduces the test application time up to 87% with respect to well known test algorithms that are also capable to cover these faults.

Another difficulty in DRAM memory testing is the fact that even the simplest defect can produce a quite complex faulty behavior. The main reason is that in practice a DRAM is an analogue circuit. Our electrical simulations on a DRAM circuit with a resistive open defect manifested this complex faulty behavior. Moreover, we observed an important unknown phenomenon, the *charge accumulation*, that significantly influences the testing procedure. Based on our observations we developed an efficient test algorithm that provides enhanced coverage of resistive open faults with respect to existing solutions.

Finally, one of the most attractive testing solutions is the Built-In Self-Test (BIST) circuits, which have gained great attention during the last two decades. Towards this direction, we have developed a BIST circuit that implements the NLTF test algorithm for DRAM testing. The outcome of this task manifested the ability to efficiently embed complex test algorithms in a memory at a low silicon area and design cost. The functionality of the BIST circuit was verified through simulations.

CHAPTER 1. INTRODUCTION

1.1 Prologue

Testing of electronic circuits is a very important procedure aiming to detect possible malfunctioning circuits before they are placed on an electronic system or they are channeled to the open market. An electronic circuit is characterized as malfunctioning either when it does not operate at all or when its operation does not comply with the specifications set by the manufacturer. The malfunction is caused by one or more failures in the fabrication process, which are called *defects*. The terms *fault models* or simply *faults* refer to the modeling of the behavior of the circuit under the presence of defects [1].

The rapid evolution of Integrated Circuits (*ICs*) technologies the last 50 years resulted in the development of very powerful digital ICs which nowadays are present in almost any electronic device used in everyday life. The first ICs in the early 60's consisted of only a few tenths of transistors in a single chip. Today, digital IC's integrate several billions of transistors in a single chip, enabling the manufacturers to build digital systems with huge computation and storage capabilities.

However, this evolution raised new challenges; a major one is related to the difficulty and complexity of the testing procedures for the digital ICs that have been dramatically increased [1] - [4]. The increased device density and design complexity achieved by modern technologies has turned testing into a first priority problem,

which requires reliable and efficient solutions. Otherwise, further evolution of semiconductor technology may turn to be almost meaningless.

Testing difficulties are mainly due to two reasons: the increased complexity of the ICs and the increased complexity of the faulty behaviors. Firstly, nowadays digital ICs implement complex functions, with too many states and input combinations that it is almost impossible to be exhaustively tested. Secondly, the faulty behaviors appear in various forms and conditions. In the first decades we could safely say that an IC either worked or not. Modern ICs may seem to operate correctly in most of the cases but may manifest a faulty behavior under very specific conditions or operation sequences. As a consequence, the testing procedure needs to be carefully designed in order to be able to reveal such faulty behaviors, with an acceptable cost in test time and sources.

Memories are a very important part of the digital systems [5]. Due to the evolution of the corresponding technologies, the storage capacity of the memories has significantly increased the last decades. Until the early 90s the number of bits that could be stored at a single chip would quadruple almost every 3.1 years. Although that increase ratio is no longer maintained nowadays, the storage capability of memories is still increasing rapidly. Thus, present days memories can store up to 4-16Gbits per IC. This is mainly achieved by the shrinking of the dimensions of the elementary storage units, which are called *memory cells*, and the distances between them.

As a drawback, like in all other IC types, modern memories suffer from increased difficulty in testing [5],[6]. The increased density of the memory cells and the shrinking of their in-between distances make their operation susceptible to various failure mechanisms like process variations, defects and interferences. The interaction among adjacent memory cells and their susceptibility to several sources of interference plays a significant role in the behavior of the memory. Thus, despite the fact that the memory is a relatively simple circuit due to its repetitive structure, the testing procedure remains a complex and time consuming task because it is not adequate to test each memory cell individually in order to determine if the memory is defective or not. On the contrary, it is mandatory to see each memory cell as a part of the total structure and in mutual interaction with the other cells in the memory array.

Moreover, several faulty behaviors appear only when certain operation sequences are applied, making the detection of the pertinent faults even more difficult.

1.2 Dissertation Scopes

The overall target of this dissertation is to develop viable solutions in the field of DRAM memory testing.

Initially, our goal is to develop new realistic fault models that accurately describe faulty behaviors observed in modern DRAM memories. Next, based on these fault models, new and effective test algorithms will be developed aiming to provide increased fault coverage at a reduced cost in test application time. Finally, our target is to provide a feasibility study on the ability to embed the proposed test algorithms in a DRAM with the use of Built-In Self-Test circuits.

1.3 Dissertation Structure

The Dissertation consists of 8 Chapters. Chapter 1 is a brief introduction. Chapter 2 presents the fundamentals on DRAM memories. Chapter 3 discusses the most popular memory fault models while Chapter 4 introduces some of the most well-known memory testing algorithms. In Chapter 5 a new approach in NPSF testing of Folded DRAM arrays is presented. A new fault model, the *NLTF*, which deals with neighborhood leakage current and cell transition related faults, is proposed in Chapter 6. In Chapter 7, the resistive open defects along with Bit-Line imbalance issues are studied through electrical simulations and a new important phenomenon that influences the faulty behavior, the *charge accumulation*, is introduced for the first time. In Chapter 8, a Built-In Self Test (BIST) circuit for DRAM memory testing, exploiting the NLTF test algorithm, is developed. Finally, in Chapter 9 the conclusions are drawn.

CHAPTER 2. DRAM MEMORIES

2.1 Abstract

In this Chapter we discuss basics issues on DRAM memories. Initially, the MOS transistor is presented since it is the fundamental device in semiconductor technologies. Next, we briefly present the most popular memory types. Finally, the DRAM memory structure and operation are analyzed.

2.2 The MOS Transistor

2.2.1 MOS Transistor Basics

The Metal Oxide Semiconductor (MOS) transistor is the fundamental device in semiconductor technologies. There are two types of MOS transistors: the NMOS, which uses a n-type channel, and the PMOS, which uses a p-type channel [7], [8].

As we can see in Figure 2.1a, the NMOS transistor consists of two n^+ regions formed in a p-type substrate. The region between the two n^+ regions forms the *n-channel*, and the distance between these regions is called the *length of the channel*, which is indicated with the symbol L. Above the channel is the *transistor gate*,

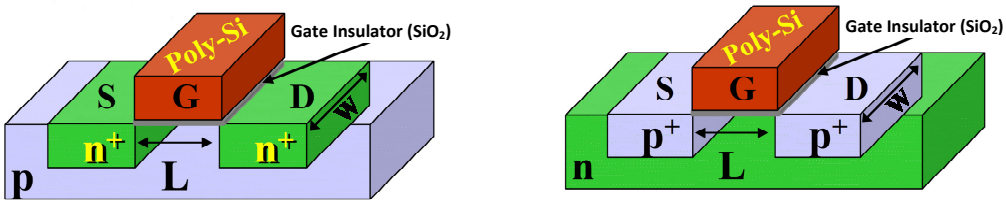


Figure 2.1.: Structure of the NMOS transistor

denoted as G in the Figure 2.1, which is a conductive area formed by Poly-Silicone and is electrically isolated from the channel and the n^+ regions by a thin layer of an oxide insulator. In the same figure another important characteristic of the transistor is shown, the *width of the channel* which is denoted as W.

The gate, the two n^+ regions and the substrate are the terminals of the transistor. When the transistor is connected to a specific circuit, the n^+ region with the lowest voltage level is called *source*, denoted as S, while the other one is called *drain* and is denoted as D. Note that the two n^+ regions are identical; only the voltage levels when the transistor is connected to a circuit determines which n^+ region plays the role of the source and which plays the role of the drain. Thus, the drain to source voltage is always positive ($V_{DS} \geq 0$). Moreover, in NMOS transistors the substrate is usually connected to the ground (0V)

Similarly, the PMOS transistor is formed by two p^+ regions in a n-type substrate and the transistor gate (see Figure 2.1b). In the PMOS transistor, the p^+ region with the highest voltage level is the source, while the other one is the drain. Consequently, for PMOS transistors $V_{DS} \leq 0$. The substrate in this case is usually connected to V_{DD}, where V_{DD} is the operation voltage of the circuit.

In Figure 2.2 we can see the schematic symbols of NMOS and PMOS transistors. The substrate terminal is usually omitted.

(a) (b)

Figure 2.2.: Symbols of NMOS (a) and PMOS (b) transistors

The MOS transistor operation is as follows; the drain to source current I_{DS} is determined by controlling the gate to source voltage, V_{GS}. Both transistors have three operating regions: the *cutoff region*, the *triode region* (or *linear* or *non – saturation region*) and the *saturation region*. In each of these areas the I_{DS} is given by the following equations:

$I_{DS} = 0$	cutoff region	Eq. 2.1
$I_{DS} = \beta\left[(V_{GS} - V_T)\cdot V_{DS} - \frac{1}{2}V_{DS}^2\right]$	triode region	
$I_{DS} = \frac{\beta}{2}(V_{GS} - V_T)^2$	saturation region	

where V_T is the *threshold voltage* of the transistor and it practically represents the lowest, in absolute value, V_{GS} for which the transistor is conductive. For NMOS transistors the V_T is positive, while for PMOS transistors it is negative.

A MOS transistor can be on one of the three regions mentioned above depending on the V_{DS}, V_{GS} and V_T values. The conditions needed to be satisfied for each region are given in Table 2.1.

TABLE 2.1.: MOS transistor regions of operation

	NMOS	PMOS
cutoff region	$V_{GS} \leq V_T$	$V_{GS} \geq V_T$
triode region	$0 < V_{DS} < V_{GS} - V_T$	$0 > V_{DS} > V_{GS} - V_T$
saturation region	$0 < V_{GS} - V_T < V_{DS}$	$0 > V_{GS} - V_T > V_{DS}$

The factor β is the *channel conductance* and it is given by the following equation:

$$\beta = \frac{W}{L}\frac{\varepsilon_{ox}\mu}{t_{ox}} = \beta_0 \frac{W}{L} \qquad \text{Eq. 2.2}$$

In equation 2.2, ε_{ox} and t_{ox} is the permittivity and the thickness of the gate oxide respectively, μ is the average surface mobility for the carriers, which is different for the two types of transistors (μ_n for NMOS, μ_p for PMOS), L is the channel length (the physical distance between source and drain) and W is the channel width. Moreover, the factors ε_{ox} , μ and t_{ox} (and consequently the *conduction factor* β_0) are specified by the given MOS technology and, thus, are not circuit design variables. However, the designer usually can chose between a few pre-defined t_{ox} values. A given MOS technology also specifies the minimum allowed values for the two design variables, the minimum channel length L_{min} and width W_{min}.

As we can see from equation 2.1 the transistor conductance increases with the channel width and decreases with the channel length. Since the area that the transistor occupies is determined by L and W, in most cases L_{min} is used and the W is chosen depending on the desired transistor conductance.

2.2.2 MOS Transistors In Digital Circuits

Digital circuits operate ideally in two discrete voltage values: one low voltage value, which is usually 0V, and one high voltage value, which is denoted as V_{DD}. Although the I_{DS} current of both NMOS and PMOS transistor is described by the same equations (equation 2.1), they have a fundamental difference as previously mentioned: the source in NMOS is the terminal with the lowest voltage, while in PMOS is the terminal with the highest voltage. Thus, the V_{GS} voltage in equation 2.1 is calculated differently in the two cases.

If we connect one of the n^+ terminals of the NMOS transistor to 0V, then this terminal automatically becomes the source. Then, when we apply V_{DD} voltage to the gate, the V_{GS} is constant and equal to V_{DD} regardless of the drain voltage. In this case, according to Equations 2.1 and Table 2.1, the I_{DS} can be 0 only when V_{DS}=0. This means that a NMOS transistor is capable of discharging a terminal connected to its drain to 0V.

On the other hand, if we connect a n^+ terminal of the NMOS transistor to V_{DD} this terminal becomes the drain. If we try to charge a terminal connected to the source by applying a V_{DD} voltage to the gate, the V_{GS} voltage will be initially V_{DD}, provided that the initial voltage of the source terminal is 0V. However, as the voltage of the source raises the V_{GS} decreases. Finally, when the terminal reaches the $V_{DD} - V_T$ voltage, the transistor will enter the cutoff region and the charging will stop. Consequently, the NMOS transistor cannot charge a terminal to a voltage higher than $V_{DD} - V_T$. Moreover, the charging procedure will be much slower than the discharging described in the previous paragraph since in this case $V_{GS} \leq V_{DD}$.

Following the same reasoning we can easily see that a PMOS transistor can charge its source terminal up to V_{DD} but cannot discharge its drain terminal to a voltage level lower than $|- V_T|$, while the discharging procedure is much slower than the charging one. For this reason the digital circuits are constructed using the *Complementary MOS* (*CMOS*) technology. According to this design approach the logic elements (e.g. logic gates) are constructed of two nets: one net which consists of PMOS transistors and is connected to V_{DD} and one net of NMOS transistors which is connected to 0V. This provides the ability to the logic element to charge its output at both V_{DD} and 0V. However this approach is not always feasible to be used, especially in memories, as we will see in the paragraphs that follow. Moreover, we should keep in mind that the conduction factor β_0 is larger, double or more, in NMOS transistors than in PMOS transistors. Thus, in general, aiming to achieve the same charge and discharge times the PMOS net should consist of larger transistors (larger W) with respect to the NMOS net, given that the channel lengths are the same.

2.3 Memory Types

A Computing System is mainly divided into three subsystems: i) the Central Processing Unit (CPU), ii) the input-output (I/O) devices and iii) the memory [6]. The memory subsystem usually consists of two memory categories [6], [9]:

- The mass storage devices (secondary memory), such as hard disk drives, compact disks etc. Their main characteristic is their large storage capacity with low cost/capacity ratio and low access speed. They are used for permanent data storage purposes.

- The main memory, the cache memory and the register file, which store data and programs during processing. They are characterized by high access speed and high cost/capacity ratio. The number of memory ICs in a computer system is significantly high compared to the total number of ICs of the system. Therefore a failure in the memory subsystem is one of the main causes of a computing system failure.

Various memory types of the second category are used, depending on the requirements of each subsystem [6], [9]. The most important of them are as follows:

- Static Random Access Memory – SRAM. This memory type has the highest access and data transfer speed, due to its very low latency and very high bandwidth. Its great disadvantage is the low data storage density per silicon area which highly increases the cost/capacity ratio. It is mainly used as cache memory, where the main demand is the high performance. Its lifetime is not significantly influenced by the number of I/O operations and it is a volatile type of memory.
- Dynamic Random Access Memory –DRAM. This memory type is characterized by high data storage density per silicon area and relatively high access and data transfer speed. In other words, it is large and cheap compared to cache and it is fast compared to a hard disk. Moreover, as in SRAMs its lifetime is practically not influenced by the number of I/O operations performed on it. For these reasons it is mainly used as the main memory of a system. Its main disadvantage is that a periodical refresh procedure is necessary in order to retain the data. It is a volatile type of memory, since the data stored are lost when the power supply is disconnected.
- Read-Only Memory – ROM. These memories are non volatile and the data stored at them are pre-stored by the manufacturer and cannot be altered, expanded or deleted by the user. Usually they store basic information that is needed by the microprocessors in order to perform basic operations such as interaction with keyboards, display, disks etc.
- Programmable Read-Only Memory – PROM. This is a variation of a ROM memory which is not pre-programmed by the manufacturer but it is

programmed by the user. After it is programmed, the data stored can no longer be altered.

- Erasable Programmable Read-Only Memory – EPROM. This is a variation of the Programmable Read-Only Memory which can be programmed by the user more than once. In order to be reprogrammed, firstly it has to be removed from the system and its data must be erased using ultra-violet radiation. Its main disadvantages are that the programming procedure requires special equipment, it is sensitive to light, it has lower performance than ROM and its packaging is expensive.
- Electrically Erasable, Programmable, Read-Only Memory – EEPROM. Nowadays, they are widely used and are known under the name "Flash Memories". They are read-write memories with non-volatile capability and are used as a permanent storage. Compared to hard disks, their I/O speed is significantly lower and their cost per bit is significantly higher. However, their extremely small dimensions and weight make them ideal for the storage unit of cameras, video cameras and other portable devices. Note that the write operations cause gradual wear out to the memory cells and, thus, they can sustain a limited number of write operations before they start to fail. Moreover, their data retention time has limitations.

2.4 DRAM Memory Structure

In general memories consist of the memory cells where the data are stored (memory array) and a number of assisting circuits [9] - [13]. In the latter case, the most important assisting circuits are the *sense amplifiers (S.A.)*, the *precharge circuits*, the *I/O circuits* and the *address decoder*. The functionality of these circuits will be analyzed in the paragraphs that follow.

2.4.1 The Memory Array

Memory cells are the elementary storage units that each store one bit of information and they are arranged in matrix like structures called *memory arrays*. Each memory cell is connected to one data transfer line which is called *Data-Line* or

Bit-Line. Additionally, every cell is activated (for a read or write operation) by an activation line called *Word-Line*.

In Figure 2.3 we see the structure of a DRAM memory cell. It consists of one capacitor, which stores the data in the form of electric charge, and a pass transistor (usually NMOS) which connects the capacitor with the Bit-Line. The pass transistor is activated by the Word-Line. When the cell stores data, the capacitor's voltage is ideally close to 0V or V_{DD}, where V_{DD} is the operating voltage of the memory. As we will see next, the voltage value of the capacitor is translated into logic 0 or logic 1 during the read operation. In order to achieve the lowest possible cost in silicon area per cell, the minimum allowable values are used for the L and W parameters of the transistor (L_{min}, W_{min}).

There are two types of capacitors as we can see in Figure 2.4 [7] - [13]; in the first type the capacitor is buried deep in the silicon under the transistor and is called *trench capacitor*, while in the second type the capacitor is located above the transistor level and is called *stack capacitor*.

Considering the memory array as a matrix, the memory cells located on the same row are activated by the same activation line (Word-Line), while those located on the same column are connected to the same Bit-Line. In DRAMs the Bit-Lines appear in pairs, since the Bit-Lines of a pair are connected to the same sense amplifier, precharge circuit and write buffer circuit, which, as we will see in the paragraphs that follow, are necessary for the operation of the DRAM. There are two architectures for

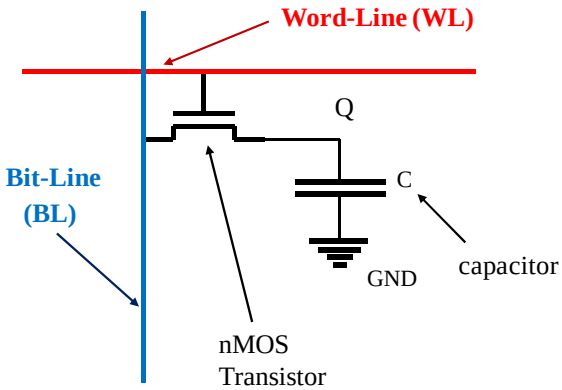
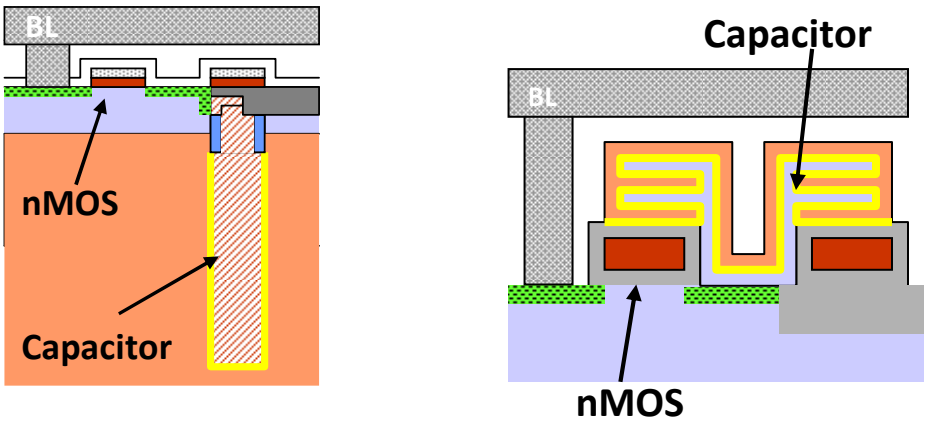


Figure 2.3. The DRAM memory cell

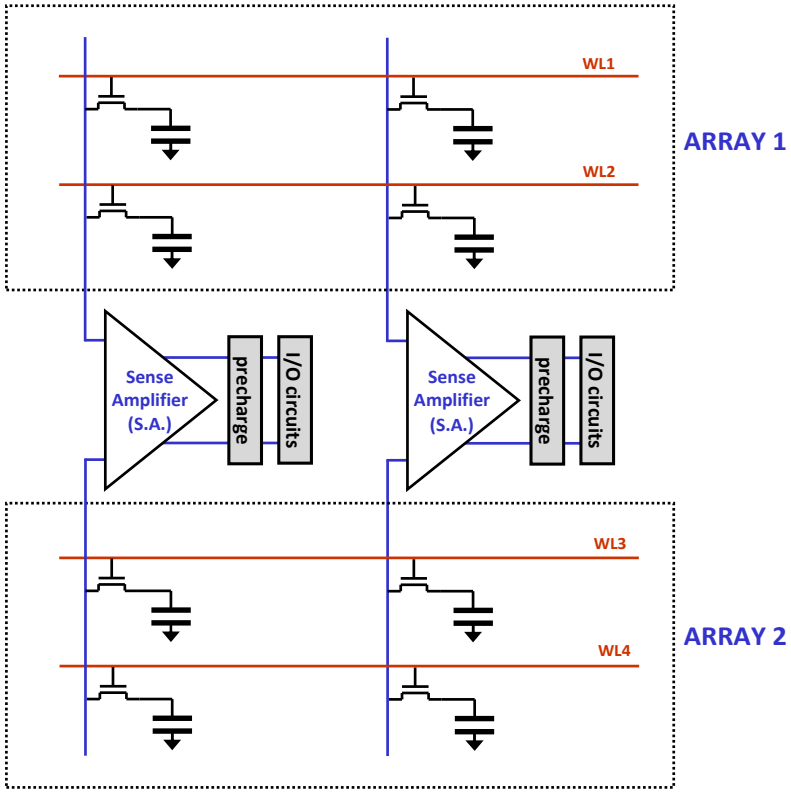


(a) (b)

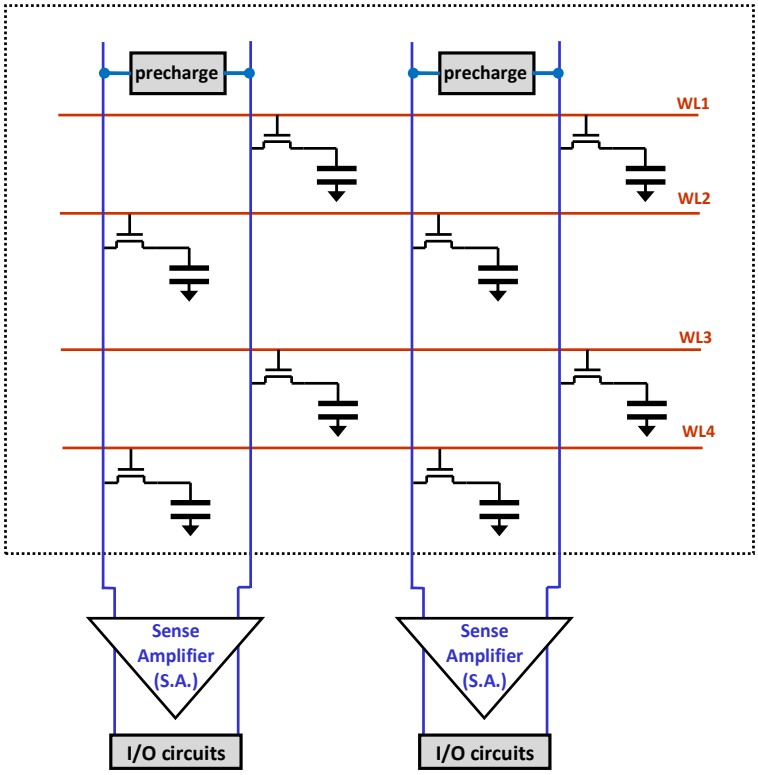
Figure 2.4.: Memory cell with trench (a) and stack capacitor (b)

the memory array, depending on how the two Bit-Lines that form a pair are placed: the *Open Bit-Line* architecture and the *Folded Bit-Line* architecture. As it is illustrated in Figure 2.5a, in the Open Bit-Line architecture each pair consists of Bit-Lines that are placed in different array blocks. On the other hand, in the Folded Bit-Line architecture (see Figure 2.5b), the Bit-Lines that form a pair are adjacent in the same array.

In both architectures the memory cells are placed in pairs as we can see in Figure 2.6, back to back, having a common Bit-Line contact. Thus, the layout of the two architectures is according to Figure 2.7 (a) and (b) respectively. In those figures we can also see silicon area occupied from the cell in each architecture, which is 2Fx3F = $6F^2$ for the open Bit-Line architecture and 2Fx4F = $8F^2$ for the folded, where F is the minimum lithographic feature size of the technology [10].



(a)



(b)

Figure 2.5.: The Open Bit-Line (a) and the Folded Bit-Line (b) architectures

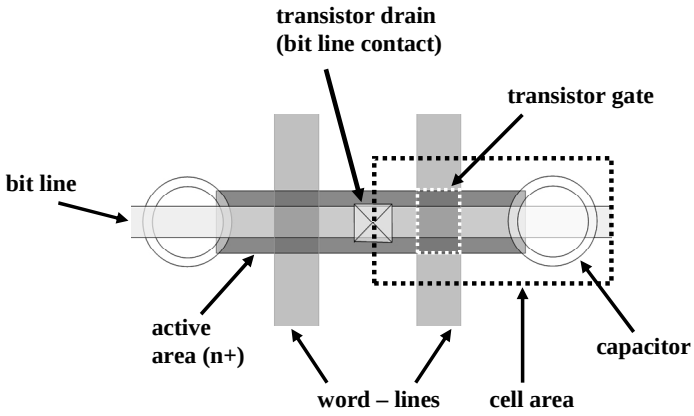
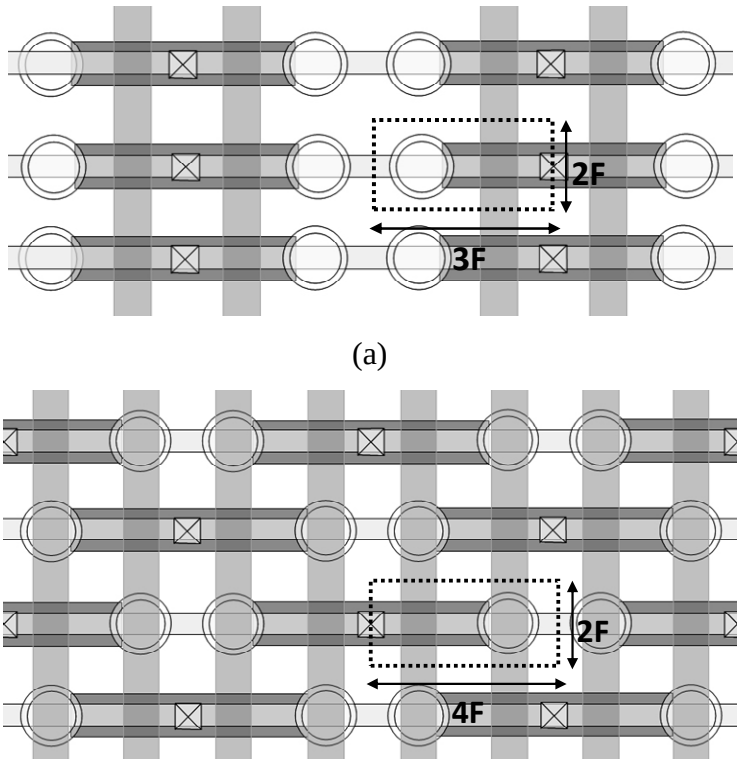


Figure 2.6.: Layout of the DRAM memory cells



(a)

(b)

Figure 2.7.: Layout of the Open Bit-Line Architecture (a) and the Folded Bit-Line Architecture (b)

2.4.2 The Precharge Circuit

The precharge circuit consists of three NMOS transistors, as shown in Figure 2.8. One of them, which is called the *equalization transistor*, connects the two Bit-Lines with each other while the other two transistors connect each Bit-Line with a $V_{DD}/2$ voltage source. When the circuit is activated the transistor gates are raised to V_{DD} and voltage of the two Bit-Lines starts to move towards $V_{DD}/2$. Usually all these transistors have the same L and W.

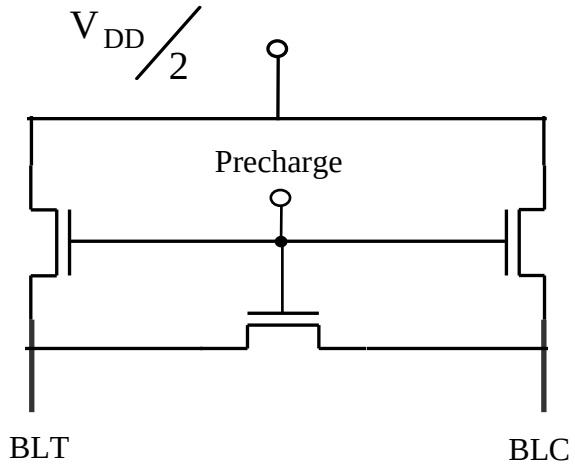


Figure 2.8.:The precharge circuit

It is easy to see that this precharging procedure is not symmetrical for the two Bit-Lines, if we take into account the equations of Table 2.1 (Subsection 2.2.1). Obviously the equalization transistor contributes equally to the charging of both Bit-Lines. However, the contribution of the two other transistors is not equal, as shown in the analysis that follows.

At the beginning of the precharging, one of the Bit-Lines is initially at 0V and the other at V_{DD}. Consequently, we observe that for both transistors the V_{DS} voltage starts from $V_{DD}/2$ and progressively decreases to 0V as the pertinent Bit-Line's voltage is getting closer to $V_{DD}/2$. The V_{GS} however is not the same for both transistors. The V_{GS} for the transistor which is attached to the Bit-Line that is discharging from V_{DD} to $V_{DD}/2$ is constantly equal to $V_{DD}/2$. On the other hand, the V_{GS} for the transistor attached to the other Bit-Line, which is charging from 0V to $V_{DD}/2$ is initially equal to V_{DD} and progressively reduces to $V_{DD}/2$ as the Bit-Line voltage raises to $V_{DD}/2$. Thus,

it is expected that the discharging of the Bit-Line having V_{DD} voltage will be slower than the charging of the Bit-Line having 0V. Also note the significant contribution to the precharging process of the equalization transistor, since its V_{GS} varies from V_{DD} to $V_{DD}/2$ and its V_{DS} varies from V_{DD} to 0V. Thus, its mean contribution is higher than any of the other two transistors.

2.4.3 The Sense Amplifier

As previously stated, the sense amplifier is the circuit that senses the voltage difference between BLT and BLC after the Word-Line is activated, enhances and finally maximizes this voltage difference, setting the one Bit-Line to V_{DD} and the other to 0V. In its simplest implementation, it consists of two cross-coupled inverters as we see in Figure 2.9. The p-net and n-net of the sense amplifier are not constantly connected to V_{DD} and 0V respectively; instead they are connected through two activation transistors. The sense amplifier is activated by setting the signal SA_EN_P to 0V and the SA_EN_N to V_{DD}; in the complementary situation the sense amplifier is inactive.

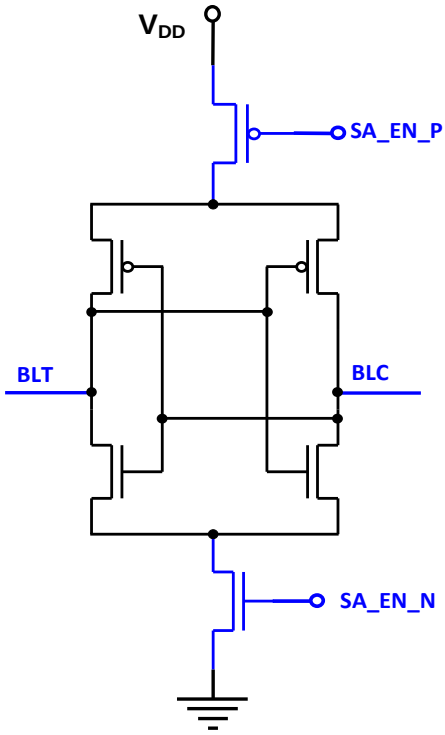


Figure 2.9.:The sense amplifier

Ideally, both NMOS transistors of the cross-coupled inverters are matched and the same stands for the two PMOS transistors so that the sense amplifier can sense correctly any voltage difference between BLT and BLC, almost no matter how small it is. However, in reality due to process variations the transistors cannot be identical; small differences in $V_{T,}$ the channel conductivity or other parameters may cause the sense amplifier to be biased in sensing 0 or 1. Although in the DRAM design process such bias should be taken into account, in testing it does not pose a big problem because it has a constant direction. In other words, if the sense amplifier constantly favors 1 over 0, independently of the previous state of the memory, then the faulty behavior will manifest itself when we are trying to read a 0. On the other hand, bias that do not have a constant direction but depend on the previous operations and states of the memory are a significant concern in testing.

2.4.4 The Word Line Driver

The word line driver is the circuit responsible for activating and deactivating the Word-Line. Ideally it raises the Word-Line to V_{BOOST} and drops it to 0V within the appropriate time interval. However, deviations from the ideal behavior may result to a delay or even to a failure to set the appropriate voltage on the Word-Line, influencing the operations performed on the cells attached to it.

2.4.5 The Address Decoder

The address decoder is responsible for accessing the appropriate cell according to the given address and is divided in two sub-circuits: the row decoder and the column decoder. The row decoder selects the appropriate Word-Line to be activated, according to the incoming address, while the column decoder selects the appropriate Bit-Line in order to obtain the data read or provide the data to be written.

2.4.6 The Hierarchical Structure Of The Memory

As previously stated memories consist of memory cells, which are arranged in memory arrays, and a number of assisting circuits. In Figure 2.10 we can see a DRAM memory array along with the basic assisting circuits [9].

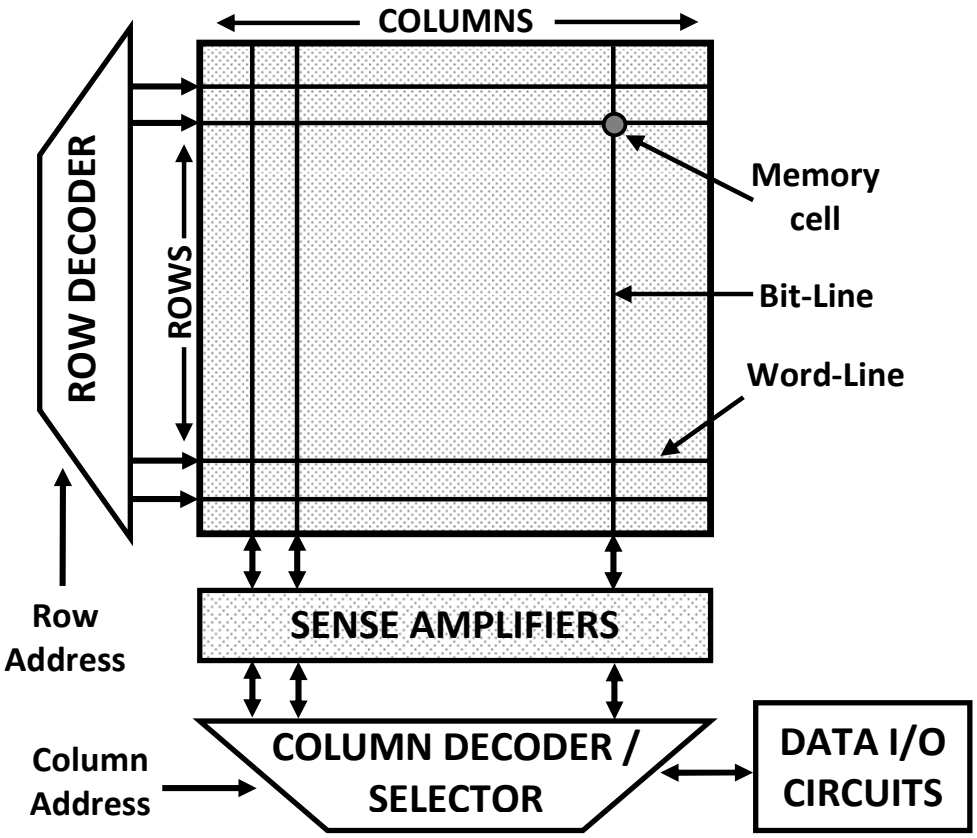


Figure 2.10. A DRAM array along with the (peripherals – assisting circuits?)

Also in Figure 2.11 we can see how the memory arrays and the assisting (peripheral) circuits are placed in a memory chip. Note that this structure generally applies to both Open and Folded Bit-Line architecture [10].

Memory arrays are organized in *banks* (also called *blocks*). A bank is generally a set of arrays that operates independently of other banks. Each DRAM IC may be organized in one or more banks. Having more than one bank is usually exploited for increasing the bandwidth of the memory data bus, by interleaving between multiple data banks. In this way the data bus of the memory IC has a higher bandwidth than the one that can be achieved by a single bank.

Moreover, if the DRAM has an I/O bus of more than one bit wide, this is usually achieved by using multiple arrays, while each array provides one bit. Thus, although the DRAM IC is usually word-oriented (i.e. it reads and writes a set of bits simultaneously), the actual read and write operations on each array may be bit-oriented [9].

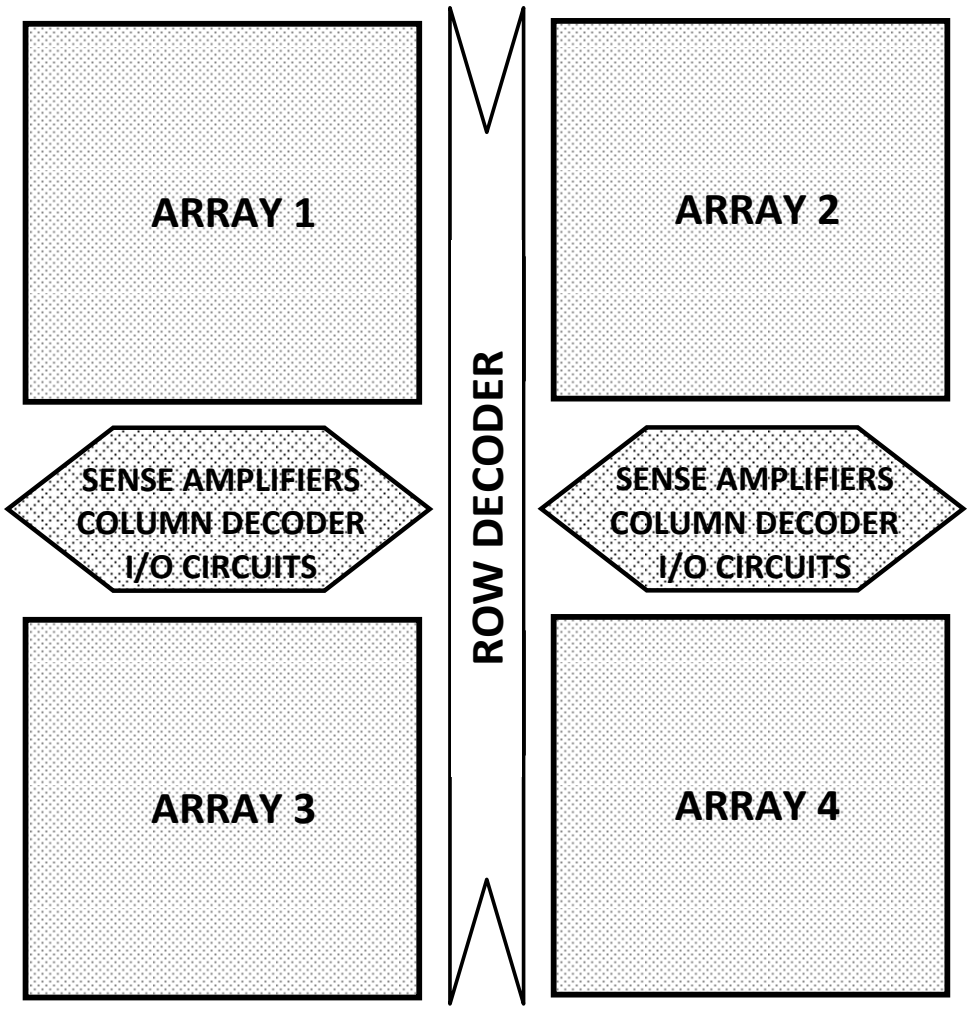


Figure 2.11. DRAM memory array organization

2.5 DRAM Operation

As previously stated, the Bit-Lines are arranged in pairs. In each pair, one of the Bit-Lines will be called *True Bit-Line (BLT)* and the other *Complementary Bit-Line (BLC)*. Both the BLT and the BLC are connected to the same Sense Amplifier, precharge circuit and write buffer circuit. Also note that every Word-Line activates only one memory cell that is either attached to the BLT or the BLC. Consequently, the set of Word-Lines that activate the cells of BLT is different than the set of Word-Lines that activate the cells of BLC.

Before a read or write operation both BLT and BLC must be precharged to (ideally) the same voltage level, the *precharge voltage*, which is close to $V_{DD}/2$. This is achieved through the activation of the precharge circuit.

After the precharging is complete and the precharge circuit is de-activated, the cell to be read is activated by raising the pertinent Word-Line to a voltage higher than V_{DD} which is called V_{BOOST}. The importance of raising the Word-Line to V_{BOOST} and not to V_{DD} will be explained later. The pass transistor of the cell is activated and charge sharing between the capacitor and the cell's bit line takes place. Due to this charge sharing, the voltage of the cell's Bit-Line will be shifted slightly towards 0V or V_{DD}, depending on the initial voltage of the cell's capacitor.

Assume for example that the cell we want to read belongs to BLT. After the Word-Line activation, the BLT voltage will be shifted to a higher voltage, if the capacitor voltage was V_{DD}, or to a lower voltage, if the capacitor voltage was 0V. Meanwhile the BLC remains at the precharge voltage, since none of its cells was activated. Thus, a voltage difference between the two Bit-Lines is created. The sense amplifier, which is activated next, senses, enhances and finally maximizes this voltage difference, forcing the one Bit-Line to 0V and the other to V_{DD}. If, for example, the cell's capacitor has initially a voltage of V_{DD} (or 0V) the sense amplifier will force BLT to V_{DD} (or 0V) and BLC to 0V (or V_{DD}). Similarly, in case the cell belongs to BLC, BLC will be forced to the same value as the initial voltage value of the corresponding cell's capacitor while BLT will be forced to the complementary voltage value. In Figure 2.12 we can see the waveforms for a write 0 and read 0 operation in a DRAM.

Note that in DRAMs it is crucial that the Bit-Lines are forced to the values V_{DD} and 0V because this is how the capacitor's voltage is restored to its initial value. This is true due to the fact that the charge sharing between the capacitor and the Bit-Line, which takes place when the Word-Line is activated, alters the voltage of the capacitor and brings it close to $V_{DD}/2$ (this means information loss). Thus, when the Bit-Line reaches its final voltage value, which is the same as initial voltage of the capacitor prior to the read operation, the capacitor voltage is restored.

The write operation is similar to the read operation with the difference that as the Word-Line is activated the *write buffer* of the I/O circuit is activated too. Since the strength of the write buffer is larger than the cell's strength, the voltage difference

sensed by the sense amplifier is the one dictated by the write buffer. Thus, the new value is stored to the cell.

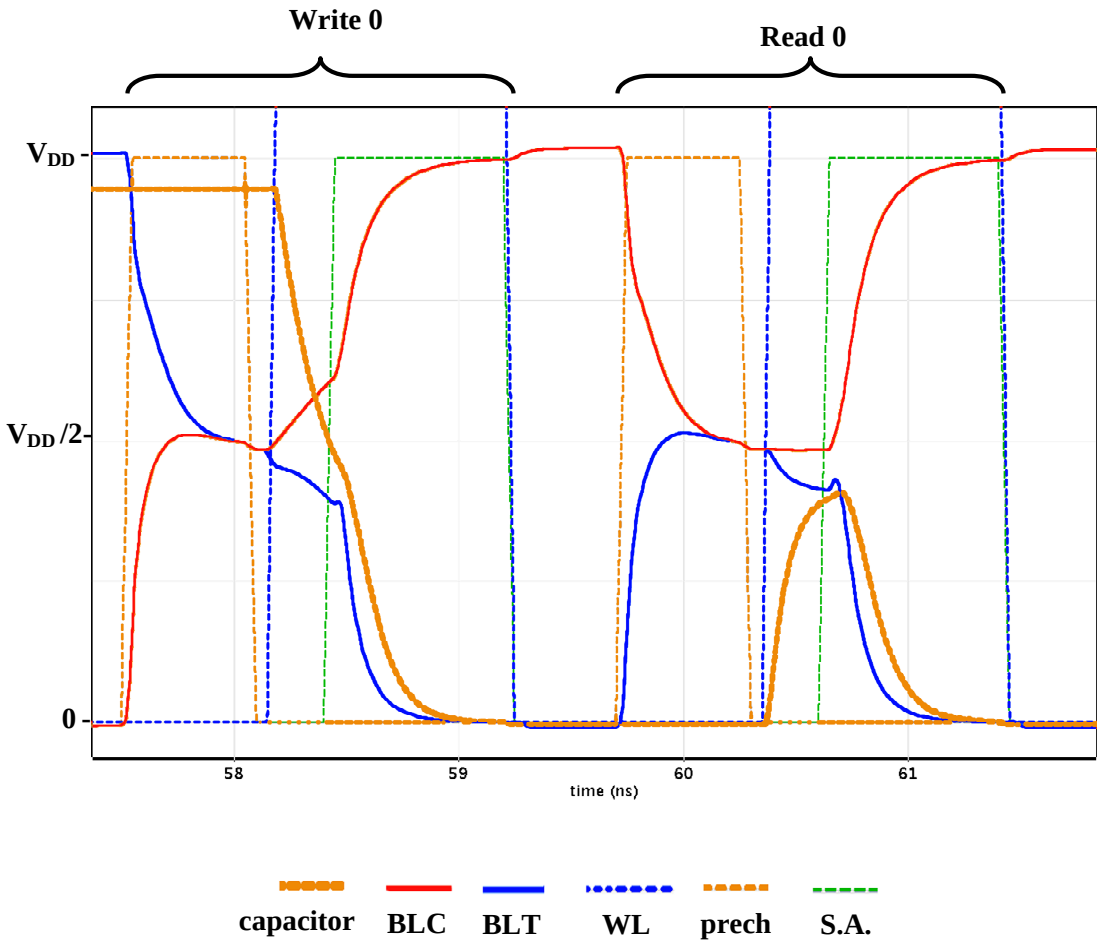


Figure 2.12: DRAM write 0 and read 0 operation waveforms

At this point it is important to note that in the read and write operations described above all the cells attached on the activated Word-Line are involved. When the Word-line voltage is raised to V_{BOOST} all the cells attached to it are activated and their capacitors will exchange charge with the pertinent Bit-Lines, altering the voltage level of the capacitors. Thus, in a read operation, aiming to avoid information loss, all the cells of the activated Word-Line must perform a read and re-write operation, but only the data obtained by the cell we actually wanted to read are transferred to the output buffer. A similar situation occurs in the write operation: except from the cell we want to write, all the other cells that belong to the activated Word-Line will perform a read and re-write operation in order to maintain their data. Consequently, before a

read/write operation all the Bit-Lines of the array will be precharged to $V_{DD}/2$ and during the operation all the sense amplifiers will be activated.

From the discussion above, we can easily see that a read operation on a cell belonging to BLT turns the BLT to the same value as this of the cell and the BLC to the complementary value. On the other hand, a read operation on a cell belonging to BLC turns the BLT to have the complementary value with respect to the value stored at the cell, while BLC will take the same value as the one stored at the cell. The same situation stands for the write operations. Thus, when for example we write or read logic 0 to a cell, the cell's capacitor is charged at 0V if the cell belongs to BLT, or it is charged at 1V if the cell belongs to BLC. Due to this status the names BLT (Bit-Line True) and BLC (Bit-Line Complementary) are derived.

The importance of raising the activated Word-Line to V_{BOOST} instead of V_{DD} can be easily understood if we consider the NMOS transistor operation. From the discussion in subsection 2.2.2 we conclude that if the Word-Line was raised to V_{DD} during the read and write operations then the capacitor would be unable to reach V_{DD} due to the way the NMOS transistor operates; instead, it would only charge up to $V_{DD} - V_T$. Similarly, if instead we used a PMOS transistor for the construction of the cell, the capacitor voltage would not be lower than $|-V_T|$. If both transistors, connected in parallel, are used the area occupied by the cell would be increased significantly. Thus, the adopted solution was to exploit only the NMOS transistor and use a voltage V_{BOOST}, which is at least equal to $V_{DD} + V_T$, as the Word-Line activation voltage. We prefer the NMOS transistor over the PMOS due to its significantly higher conduction factor β_0. The drawback of this approach is that the charging of the capacitor is significantly slower than its discharging.

CHAPTER 3. MEMORY FAULT MODELS

3.1 Abstract

In this chapter we will discuss fundamentals issues on memory fault models. The chapter mainly focuses on memory array faults and presents the most popular fault models.

3.2 General Fault Models

Testing procedures for integrated circuits can be based on various methods. The most common testing approach is the application of a series of combinations at the inputs of the IC and the comparison of the response at the outputs of the IC with the expected values. The input combinations are called *input patterns* or *test patterns*. An erroneous response of the IC, which is called error, manifests the presence of a defect. Obviously the application of all possible input patterns is usually an impossible task, due to the extremely large number of input patterns. Moreover, the ICs which include

memory elements should combine all input patterns and input sequences with every possible state of the memory elements, which is impossible to be done at an acceptable time cost.

A more practical approach is to apply to the IC a suitable subset of selected test patterns which guarantee that in the presence of any fault in it the IC will provide an erroneous response which will allow the detection of the fault.

Some of the most general fault models for digital ICs are the following:

- *Stack-at fault*: a node of the circuit presents always the same value.
- *Transistor Stuck-On* or *Stuck-Open fault*: a transistor of the IC is constantly in conductive or non conductive state respectively.
- *Bridging fault*: two nodes of the circuit are short-circuited and constantly present the same value
- *Open Circuit fault*: a conductive line of the circuit is cut.
- *Delay fault*: the signal delay in one or more paths of the circuit is higher than the nominal.

3.3 Memory Faults

According to Chapter 2, memories consist of memory cells and a number of assisting subcircuits. The most difficult part in memory testing is the detection of defects in the memory array. This difficulty comes as a result of the following reasons:

- Every memory cell is directly involved only in the reading or writing operations performed on itself. This makes testing a time consuming process since in every read or write operation only one cell (or a small number of cells, in a word oriented memory) participate.
- The memory cells interact with each other, mainly due to the small distance between them [19], [23]. This interaction depends on the logic state of the cells and on the operations performed on them.
- Some of the defects that may appear on a cell are so weak that can force them to manifest a faulty behavior only under certain conditions. In other words, the ability to detect a defect depends not only on the operations we

perform on it but also on the combination of operations and states of other cells as well.

- Defects on a sense amplifier, a Word-Line driver or a precharge circuit can manifest themselves by causing faulty behavior when operations are performed on the cells attached on the pertinent Bit-Lines or Word-Lines. Obviously, if the defect is severe all the cells on a Word-Line or a pair of Bit-Lines will present a faulty behavior and the defect will be easily detected. However, weak defects or even small and acceptable deviations from the ideal behavior of these circuits can influence the outcome of a testing procedure, either revealing or masking weak defects at the cells.

For the above reasons, in this dissertation we will focus on the fault models that describe the faulty behavior of the cells, keeping in mind that in many cases a defect or a deviation from the mean performance of a Word-Line driver, a sense amplifier or a precharge circuit can be also responsible for the observed faulty behavior. We will only make a brief description on the address decoder faults since they are completely different than cell related faults.

3.4 Address Decoder Faults

From the assisting circuits we have mentioned so far, only the address decoder can produce a faulty behavior different than these described by cell-related fault models. For this reason, we will now briefly describe the address decoder faults.

As mentioned earlier, the address decoder is responsible for accessing the specific memory cell on which we want to perform a read or write operation. If the address decoder operates correctly then every memory address corresponds to one and only one cell (or a word, if the memory is word-oriented) and every cell (or word) is accessed by only one address. Thus, the relationship between the address space and the set of memory cells is a bijective function. The address decoder faults violate this relationship. There are four cases of address decoder faults [5], [6]:

- A specific address does not give access to any cell.
- A specific address gives access to more than one cell.
- A specific cell cannot be accessed by any address
- A specific cell can be accessed by more than one address.

Note that many of the simplest test algorithms cannot detect the address decoder faults. Thus, various test algorithms that target these specific faults have been developed. Moreover, it is a common practice that if a test algorithm can cover these types of faults, then this fact is explicitly mentioned; otherwise it can be assumed that these faults are not covered.

3.5 Memory Array Faults

This dissertation focuses on memory array faults for the reasons described in Section 3.3. Thus, unless otherwise specified, from now on when we talk about memory faults we mean memory array faults.

Memory faults can be categorized according to:

a) the number of actions (i.e. read or write operations) that must be performed in order to sensitize the fault and alter the data stored at a cell. There are two types [14]:

- *Static Faults*: These are the faults which need at most one action in order to be sensitized.
- *Dynamic Faults*: These are the faults that need more than two actions in order to be sensitized.

b) the number of cells involved for a fault activation, that is the number of influencing cells, either due to the value they store or by the operations performed on them.

Next we will analytically discuss the various fault types.

3.6 Static Faults

Static faults are divided into the following categories [5], [6]:

- Faults in which a single cell is involved *(single cell faults)*. These are the *stuck-at faults* and the *transition faults*.
- Faults in which two cells are involved; these are the *Coupling Faults (CFs)*
- Faults in which k cells are involved. These are divided in two sub-categories:

a) if the cells involved can be located anywhere in the memory array, then we distinguish the k-Coupling Faults, the bridging faults and the state coupling faults.

b) if the cells involved are selected based on their location on the memory array and form a cell neighborhood then we have the *Neighborhood Pattern Sensitive Faults (NPSFs).*

The two-cell and the k-cell fault models make the following assumptions:

- A read operation cannot cause a fault.
- A non transition write cannot cause a fault. Non transition write is a write operation where the value we write on the cell is the same as the one that is already stored at it.
- A transition write can cause a fault. A transition write is a write operation where the value we write on the cell is complementary with respect to the one that is already stored at it.

Next we will study these fault models in more detail.

3.6.1 Single Cell Faults

Single cell faults are the stuck-at faults and the transition faults

- Stuck at faults (SAFs): a cell stores permanently a specific logic value which cannot be changed. A testing algorithm in order to detect these faults must read both logic values 0 and 1 from every cell.
- Transition faults (TFs): a cell is unable to make a transition from one logic state to the other. In order to detect these faults every cell must make both transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ and must be read after each transition to ensure that the operation was successful.

3.6.2 Coupling Faults - CFs

When we are talking about faults in which two or more cells are involved, we frequently use the terms *victim cell* and *aggressor cell*. Victim cell is the cell that presents a faulty behavior, while aggressor cell is the cell which is considered to cause the faulty behavior of the victim cell, either by its state (i.e. the value stored at it) or its transition from one logic state to the other.

The coupling faults can be divided into the following sub-categories:

a) Bridging Faults – BFs

In this category two cells are connected (due a defect on the IC) with a conductive line and as a result both present the same logic value. This logic value may be the logic ‘AND’ of the cell states if there was no bridging fault, a case in which we have the so-called *AND Bridging Fault – ABF*, or may be the logic ‘OR’ of these states, a case in which we have the *OR Bridging Fault – OBF*.

b) *State Coupling Faults – SCFs*

The state coupling fault is defined as follows: the victim cell is forced to a specific value x only when the aggressor cell stores the value y.

c) *Inversion Coupling Faults – CFs*

According to the inversion coupling fault model, a 0 to 1 (↑) or a 1 to 0 (↓) transition of the aggressor cell inverts the data stored at the victim cell.

d) *Idempotent Coupling Faults – CFids*

The idempotent coupling fault model is defined as follows: a ↑ or ↓ transition of the aggressor cell forces the victim cell to a specific logic value, 0 or 1.

3.6.3 k-Coupling Faults

As previously stated, the static fault models in which more than two cells are involved are the k-Coupling Faults and the Neighborhood Pattern Sensitive Faults (NPSF). We should mention here that the most general fault model for the corresponding defects is the *Pattern Sensitive Fault (PSF)* model, which is considered as the most general case of cell interaction faults where all memory cells (N the number) are involved [5], [6].

The k-Coupling Faults can also be seen as an expansion of the coupling faults above. More specifically:

- The bridging faults and the state coupling faults have the same definitions in the case of the k-Coupling Faults with the difference that more than two cells or even the cells of a whole Bit-Line are involved
- The inversion and the idempotent coupling faults in the k-coupling fault model include the same definitions concerning a single aggressor and a single victim cell, with the extra restriction that the faulty behavior occurs only when the rest k-2 cells are in a specific state (their states form a specific pattern)

All the above types of faults have an extremely high test complexity. The PSFs present a complexity of $O(Nx2^N)$ operations and k-coupling faults, if we don't set any restrictions regarding the location of these k cells, where N is the number of cells in the memory array [6]. Even if k is set to 3 the test complexity is at least $O(N \cdot \log_2(N))$ operations. Note that any complexity higher than O(N) is considered as not realistic for testing in today high capacity memories. A restriction regarding the location of the k cells is set at the NPSF model which will be analyzed next.

3.6.4 The Neighborhood Pattern Sensitive Faults - NPSFs

The *Neighborhood Pattern Sensitive Fault (NPSF)* model considers the way a memory cell is influenced by the contents of its neighboring (adjacent) cells in combination with state transitions in a single cell of this neighborhood [5], [6], [15]. According to it, the value of a cell (called *base cell*) or the ability to apply a desired value at that cell is affected by the values and the transitions of k-1 neighboring cells (called *deleted neighborhood*). At this point we should emphasize that when we are referring to 'changes' or 'transitions' in the deleted neighborhood we always mean only one transition of a single cell every time. That's why NPSFs belong to the category of static faults where by definition a fault needs at most one action to be sensitized.

The combination of the base cell and the deleted neighborhood is called *neighborhood* and consists of k cells. The NPSF model is distinguished in three categories:

- *Active NPSF (ANPSF) or Dynamic NPSF*, where the base cell changes its contents due to a change in the deleted neighborhood pattern. In order to test for ANPSFs every cell in the neighborhood should be read in both states (0 and 1) after every possible change, caused by a single cell transition, in the deleted neighborhood. Practically this means that every cell in the neighborhood must be read in both states after every transition (↑ and ↓) of every cell in the deleted neighborhood and under the presence of every possible pattern formed by the other k-2 cells.
- *Passive NPSF (PNPSF)*, where the contents of a cell cannot be changed due to a certain neighborhood pattern. PNPSFs are tested by making a ↑ and a ↓

transition (and confirming that each transition was successful by reading the cell) on every cell in the neighborhood for every possible pattern in the deleted neighborhood.

- *Static NPSF (SNPSF)*, where the contents of a base cell are forced to a certain state due to certain deleted neighborhood pattern. SNPSFs are tested by reading every cell in both states for every possible pattern in the deleted neighborhood.

By covering the combination of Active and Passive NPSFs (APNPSFs) we also cover Static NPSFs because the operations needed for SNPSFs coverage are included in the set of the operations needed in order to cover APNPSFs.

In order to cover the Active NPSFs we need for every base cell $(k-1)*2^k$ test patterns. This comes as a result of the following calculations: for every value stored at the base cell (0 or 1) each one of the other k-1 cells must make both transitions, ↑ and ↓, for every possible pattern formed by the other k-2 cells (2^{k-2} patterns). Thus, the total number of patterns is $2*2*(k-1)*2^{k-2} = (k-1)*2^k$. In other words, every cell in the neighborhood except from the base cell must undergo a ↑ and a ↓ transition for every possible pattern formed by the other k-1 cells of the neighborhood. After every transition the base cell must be read.

Similarly, in order to cover Passive NPSFs we need for every base cell 2^k test patterns, which is calculated as follows: for every pattern in the deleted neighborhood (2^{k-1} patterns) the base cell must perform the two transitions, ↑ and ↓, a total of 2^k patterns.

From the above description we can easily see that in order to cover both Active and Passive NPSFs we need $(k-1)*2^k + 2^k = k*2^k$ patterns. In other words, this means that all the cells of the neighborhood (the base cell included) must make both transitions ↑ and ↓ for every possible pattern formed by the other k-1 cells, and after each transition we must read the base cell. By doing this we also cover the Static NPSFs since we read the base cell in both 0 and 1 states for every possible pattern in the deleted neighborhood. In the next chapter efficient methods in the literature for NPSF testing will be presented.

We should mention that when it is stated that we apply $k*2^k$ patterns, this does not mean that these patterns are all different to each other (this is obvious since a pattern consists of k bits and consequently there are only 2^k different patterns). What

we actually mean is that with those 2^k patterns we construct a pattern sequence of length $k*2^k$.

Despite the fact that the NPSF model is quite old, it is still an important research field since it is the most suitable for describing the faults caused by interactions between cells [16] - [24]. Moreover, faults that appear under the presence of specific data patterns are considered to be among the main reasons for test escapes [25]. Thus, pattern sensitivity is of great importance in modern memories. The most important drawback of the NPSF model is that the pertinent test algorithms are quite expensive in application time cost. Consequently, an important field of research is reducing this cost without compromising the test quality.

Finally, we should mention that various fault models similar to NPSF have been proposed. One of them is the *Disturb Neighborhood Pattern Sensitive Faults* [22], which takes into account not only the transition write operations but also the non-transition write and the read operations. According to this model the data of the base cell are altered due to a read or write operation (not only a write) of another cell in the neighborhood combined with a specific pattern formed by the rest of the cells in the deleted neighborhood. Another model is the *Row / Column Pattern Sensitive Faults* [26] which considers the cell's sensitivity to the data and the transitions of the cells that belong to the same Word-Line and/or the same Bit-Line with it.

3.6.5 Static Faults Hierarchy

In Table 3.1 the static faults hierarchy is presented [6], starting from the most complex and general model (PSF) and moving towards the simplest model (SAF). This hierarchy is important because a test algorithm that covers the highest in hierarchy faults also covers the lowest. For example, a test algorithm that covers CFs or NPSFs includes the actions necessary to cover also the TFs and the SAFs. From this point of view, we can say that the TFs and SAFs are a subset of CFs and NPSFs

Note that the CFs and the NPSFs are independent fault categories, since they are both derived from the more general category, the PSFs, but in a different way. CFs

TABLE 3.1.: Static Faults Hierarchy

1	PSF	Pattern Sensitive Fault
2-a	NPSF	Neighborhood Pattern Sensitive Fault
2-b	CF	Coupling Fault
3	TF	Transition Fault
4	SAF	Stuck – At Fault

allow the aggressor and victim cells to be located anywhere in the memory, while in NPSF the neighborhood of a cell is specific.

3.6.6 Combinations Of Static Faults

In general, it is possible that more than one fault concurrently occur in a memory, which may belong to one or more fault types from those mentioned earlier. Thus, there are two cases [6]:

- The single fault case: there is only one fault in the memory
- The multiple fault case: there are two or more faults in the memory. These faults can be of the same type or may belong to different types. Moreover, they can be *linked* or *unlinked*.

A fault is linked when it influences the behavior of another fault, while it is unlinked when it does not influence the behavior of any other faults. Unlinked faults can be detected independently by simply applying a test algorithm that covers the corresponding fault types. On the other hand, the detection of linked faults is more complex due to their interaction. This interaction may have the following undesired effect: the activation of one fault may alter a cell's data and the subsequent activation of another fault may restore the correct data to the cell. If the cell is not read between the two fault activations, the data loss caused by the first fault activation will not be detected. In this case we have a situation called *fault masking*. Linked faults can either be of the same type or of different types. Thus we have the *linked faults of the same type* and *linked faults of different type* categories.

By definition the single cell faults (SAFs and TFs) are unlinked. On the other hand, the coupling faults are linked and fault masking may occur. The NPSF faults are considered to be unlinked, since the base cell will be read after every change in its neighborhood and it cannot be influenced by changes in other neighborhoods.

Regarding the linked faults of different types, the combinations of stack at faults with other types of faults do not present any difficulty in testing. On the other hand, the combination of transition faults with coupling faults or NPSFs is quite difficult in testing and many times it requires a combination of different test algorithms in order to be detected.

Few publications present test algorithms that cover the NPSFs that are combined with other faults. In [16], [17] the combination of NPSFs with the *Bit-Line Neighborhood Pattern Sensitive Faults – NBLSFs*, which are Bit-Line coupling related faults, is discussed. In [27], [28], the NPSFs are combined with the *Neighborhood Word-Line Sensitive Faults – NWSFs*, which are Word-Line coupling related faults. Also in [28] the influence of the Bit-Line transitions on the cell (*Bit-Line influence*) is explored in combination with the NPSF. The Bit-Line influence and the NWSF combined with NPSF will be analytically discussed in Chapter 5.

3.7 Dynamic Faults

Dynamic faults is a fault category which is intensively studied the last 15 years [14], [29], [30]. This category deals with faults that need more than one consecutive actions (i.e. read or write operations) in order to be activated.

The dynamic faults can be classified in the following categories, depending on:

- the number of cells involved, we have single cell or multi cell dynamic faults. Until now the research is limited to single cell and *two cell dynamic faults*.
- the number of read or write operations needed in order to activate the fault. Until now only *two actions dynamic faults* have been studied.

Like static faults, dynamic faults can be single or multiple and, and in case of multiple faults, these can be further classified as linked or unlinked. Next we will see some basic types of dynamic faults that have been extensively studied:

a) Single cell dynamic faults.

Five types of single cell dynamic faults have been observed:

- *Dynamic Read Destructive Fault (dRDF):* in this type a read or write operation that is immediately followed by a read operation change the data stored at a cell and the outcome of the read operation is an incorrect value.
- *Dynamic Deceptive Read Destructive Fault (dDRDF)*: a read or write operation that is immediately followed by a read operation change the data stored at the cell and the outcome of the read operation is the correct value. It is the same with dRDF with the difference that the read operations returns the correct (the expected, in the fault free case) value as a result.
- *Dynamic Incorrect Read Fault (dIRF)*: a read or write operation that is immediately followed by a read operation which returns the wrong value at the data output while the data stored at the cell remain correct.
- *Dynamic Transition Fault (dTF)*: a read or write operation that is immediately followed by a transition write operation which fails to write the new data to the cell.
- *Dynamic Write Destructive Fault (dWDF)*: a read or write operation that is immediately followed by a non-transition write operation alter the data stored at the cell.

b) Two cell dynamic faults.

The two cell dynamic faults describe the faults that are activated when two consecutive actions are performed on two cells. Like in the coupling faults model, the cell which presents a faulty behavior is called victim cell while the other one is called aggressor cell. Depending on the way that the actions are applied at the two cells, we have the following four cases:

- The two actions are applied on the aggressor cell.
- The two actions are applied on the victim cell
- The first action is applied on the aggressor cell and the second on the victim cell.
- The first action is applied on the victim cell and the second on the aggressor cell.

Since in order to cover all the above cases too many operations are required, the research is so far restricted to the first two cases which will be briefly discussed next.

a) Both actions are applied on the aggressor cell: The fault model that covers this fault category is called *Dynamic Disturb Coupling Fault – dCFd* and is defined as follows: two consecutive actions on the aggressor cell cause data loss on the victim cell.

b) Both actions are applied on the victim cell: In this case the aggressor cell contributes to the activation of the fault only by its logic state. These faults are similar with those discussed in the single cell fault case, but with the difference that the aggressor cell influences the activation of the fault with the value stored at it. Thus, the definitions of these types of faults are quite similar with the pertinent ones of the single cell fault definitions. All the faults described next are considered to be activated only when the aggressor cell stores a specific value. The definitions are as follows:

- *Dynamic Read Destructive Coupling Fault (dCFrd):* in this type a read or write operation that is immediately followed by a read operation change the data stored at the cell and the outcome of the read operation is incorrect value.
- *Dynamic Deceptive Read Destructive Coupling Fault (dCFdrd)*: a read or write operation that is immediately followed by a read operation change the data stored at the cell and the outcome of the read operation is the correct value.
- *Dynamic Incorrect Read Coupling Fault (dCFir)*: a read or write operation that is immediately followed by a read operation which returns the wrong value in the data output while the data stored at the cell remain correct.
- *Dynamic Transition Coupling Fault (dCFtr)*: a read or write operation that is immediately followed by a transition write operation which fails to write the new data to the cell.
- *Dynamic Write Destructive Coupling Fault (dCFwd)*: read or write operation that is immediately followed by a non-transition write operation alter the data stored at the cell.

Despite all those restrictions considered in the faulty behavior, the test algorithms developed so far for dynamic faults are quite expensive in test application time cost compared to the classic test algorithms used in the industry. Moreover, not all of the physical mechanisms that produce this faulty behavior are known. Thus, the area of dynamic faults is an important research field.

3.8 New Trends In Memory Testing

In the previous paragraphs we presented the basic and general information related to fundamental memory fault models. The last 15 years or so the need to adapt the fault models and testing procedures to each specific memory type has become imperative. In the first decades the test algorithms were developed with the intention to use them in both SRAM and DRAM memories. This perception tends to be abandoned, since not only SRAMs and DRAMs are now considered as rather different digital devices, but also the two types of DRAM memories, the embedded DRAM (eDRAM) and the commodity DRAM are treated differently by test researchers. In various publications such as in [31] or [33] we can clearly observe that in order to develop advanced testing techniques we need to take into account the special characteristics and the behavior of each memory device. Towards this direction, we have developed in [27], [28] a new neighborhood type and various test algorithms aiming to effectively test NPSFs in DRAMs with the folded Bit-Line array architecture, as we will see in details in Chapter 5.

Another tendency is that specific known physical mechanisms that can produce or influence a faulty behavior are studied, sometimes in combination with traditional fault models. In these studies either the analytical (theoretical) approach or the electrical simulations are used. An example of an analytical study is given in [16], [17] where NPSF's are tested along with Bit-Line coupling effects. Towards this direction we have developed in [27], [28] test algorithms to deal with NPSFs along with the Bit-Line transition influence and the Word-Line coupling effects in DRAMs. This study is analytically presented in Chapter 5. Moreover, in [34] we propose a new fault model, the Neighborhood Leakage and Transition Fault (NLTF) model, which deals with leakage current and cell transition related interactions between neighbouring cells. This fault model, along with the pertinent test algorithm, is presented in Chapter 6.

Finally, electrical simulations in order to observe and confirm the faulty behavior are widely used during the last decade. In this approach an electrical model of the memory is designed and simulated using a circuit design and simulation tool. In the designed electrical circuit of the memory various defects can be injected, like resistive opens or resistive shorts, and the faulty behavior is observed through simulations. Towards this direction we have studied in [35] the faulty behavior of a DRAM

memory cell having an internal resistive open under the presence of Bit-Line imbalance phenomena. This work is presented in Chapter 7.

CHAPTER 4. FUNCTIONAL MEMORY TESTING ALGORITHMS

4.1 Abstract

In this chapter we present various well known memory testing algorithms. We discuss the operation, the fault coverage and the application time cost of each algorithm. The latter is expressed by the number of required read and write operations as a function of N, where N is the total number of cells in the memory array. It is a common assumption in the open literature that the required time for either a read or a write operation is the same; and also that the memory array is bit-oriented, which means that every read or write operation reads or writes a single cell.

4.2 Traditional Test Algorithms

In this section we briefly present a few test algorithms developed before the early 80s which are usually called ‘traditional algorithms’ [5], [6]. They are not based in a particular fault model but they are very simple and some of them provide quite satisfactory fault coverage if we also consider their low cost in application time. For this reason some of them are still in use today as a first pass in a testing procedure.

a) The Zero – One algorithm.

This very simple test algorithm consists of four stages: i) writes all the memory cells with value 0, ii) reads all cells (expected value 0), iii) writes to all the cells the value 1, iv) reads all cells (expected value 1). Obviously, this algorithm cannot detect address decoder faults (AFs). It can only detect SAFs provided that the address decoder operates properly. It cannot provide full fault cover for any other type of faults. The time application cost is 4N.

b) The Checkerboard algorithm

In this algorithm the memory cells are divided in two groups using the checkerboard pattern (which explains the name of the algorithm). Initially all the cells of the first group are written with the value 0 and all the cells of the second group are written with the value 1 and, afterwards, the whole memory is read. Then the cells of the first group are written with 1 and those of the second group with 0 and the whole memory is read again.

The test application time cost and the fault coverage, based on the previously mentioned fault models, is exactly the same with Zero – One. However, this algorithm also covers the bridging faults between adjacent cells, provided that the address decoder operates correctly. Moreover, the checkerboard pattern used from the algorithm maximizes the leakage currents between adjacent cells. Thus, it can be used for data retention related testing, and this is the reason why it was originally proposed.

In order to apply the algorithm correctly, the *address scrambling table* must be known. Address scrambling is the relationship between the *logical address* of a cell (i.e. the address provided to the address decoder in order to access the cell) and the topological address of the cell, which is the physical location of the cell in the memory array.

4.3 March Algorithms

The *march algorithms* are the most widely used test algorithm category. Although they are mostly used for coupling faults, their usage has been expanded to other fault categories like NPSF and dynamic faults. Their main characteristic is their simplicity.

4.3.1 Definition Of March Tests

A march test algorithm is a finite sequence of *march elements* [5], [6]. A march element consists of a finite sequence of operations applied on a memory cell. These operations are as follows: write the logic value 0 (which is denoted as w0), write the logic value 1 (w1), read with expected logic value 0 (r0) and read with expected logic value 1 (r1). During the application of a march test element two fundamental principles are followed: i) all the operations of a march element are applied to the current cell before we go to the next and do the same operations, and, ii) the current march element is applied to all the cells of the memory before we start applying the next march element.

When applying a march element to the cells of a memory, we can use an increasing cell address sequence, which is denoted with $\Uparrow$, or a decreasing address sequence, which is denoted with $\Downarrow$. If the address order is not important (i.e. it is not considered to play any role to the fault coverage of the specific march element) we can use any of the two address sequences mentioned above, a fact that is denoted with the symbol $\Updownarrow$.

4.3.2 Typical March Tests

In this subsection we will see some well known march test algorithms.

a) March C-

This is probably the simplest march algorithm which covers all the unlinked AFs, SAFs, TFs and CFs. Its application time cost is 10N. It consists of 6 march elements (M_j) which are as follows:

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

$$M_0 \qquad M_1 \qquad M_2 \qquad M_3 \qquad M_4 \qquad M_5$$

b) March A

March A is a very popular test algorithm. It detects AFs, SAFs, TFs and CFs. It also covers linked CFids, which is the main reason it was developed. Its application time cost is 15N.

It consists of the following 5 march elements:

{ ⇳(w0); ⇑(r0, w1, w0, w1); ⇑(r1, w0, w1); ⇓ (r1, w0, w1, w0); ⇓ (r0, w1, w0) }

M_0 M_1 M_2 M_3 M_4

c) March B

The March B algorithm is an extension of March A in order to gain additional fault coverage. Due to the expansion of the march element M_1 it provides coverage for TFs linked with CFins or CFids. As a result of this expansion the application time cost becomes 17N.

{ ⇳(w0); ⇑(r0,w1,r1,w0,r0,w1); ⇑(r1,w0,w1); ⇓ (r1, w0, w1, w0); ⇓ (r0,w1,w0) }

M_0 M_1 M_2 M_3 M_4

Obviously we can build an infinite number of test algorithms using the march test notation. In the early 90's the most expensive march algorithms in test application time cost were March A and March B. However, the evolution of memory technologies raised up the fault coverage demands and, consequently, resulted in the construction of more complex and expensive march algorithms. An example is the march algorithms which target dynamic faults, with a test application cost that reaches up to 70N [36]. Moreover, multi-background march algorithms for NPSF testing have been developed which present a cost of 92N or more.

4.3.3 March Tests For Dynamic Faults

In this subsection we will briefly mention few march type algorithms that target dynamic faults. Among the march algorithms we have seen so far only March B partially covers some dynamic faults. The algorithms that follow target only the dynamic fault types we discussed in details in Section 3.8, where the two consecutive actions that activate the fault are performed on the same cell; either on the aggressor cell or on the victim cell.

a) March RAW1

This algorithm covers the single cell dynamic faults with the restriction that from the two consecutive actions that activate the fault the first action is a write operation

and the second a read operation [30]. The strategy behind the algorithm is simple; it performs a read operation on the cell right after it is written. The cost of the algorithm is 13N operations.

The algorithm consists of 9 march elements which are as follows:

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

$$M_0 \quad M_1 \quad M_2 \quad M_3 \quad M_4 \quad M_5 \quad M_6 \quad M_7 \quad M_8$$

b) March RAW

The March RAW algorithm covers the two cell dynamic faults with the restriction that from the two consecutive actions that activate the fault the first action is a write operation and the second a read operation [30]. At the same time it also detects the same single cell dynamic faults covered by the previous March RAW1 algorithm. The cost of the algorithm is 26N operations.

The algorithm consists of 6 march elements which are as follows:

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

$$M_0 \qquad\qquad M_1 \qquad\qquad M_2$$

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

$$M_3 \qquad\qquad M_4 \qquad\qquad M_5$$

c) March MD2

The March MD2 algorithm covers all single cell and two cell dynamic faults. In the pertinent work [36] it is proven that the cost of the algorithm, which is 70N, is the minimum required for these types of faults.

The algorithm consists of 6 march elements which are as follows:

{ ⇳(w0); ⇑(r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1);
M_0 M_1

⇑(r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w0, w1);
M_2

⇓(r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1);
M_3

⇓(r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); ⇳(r0); }
M_4 M_5

4.4 NPSF Testing Algorithms

In this section we will present the most important algorithms for NPSF testing. We will only discuss the algorithms that detect and locate all the NPSF sub-categories, Static, Passive and Active (Dynamic), because algorithms that deal only with some of these categories and those which only detect and do not locate the fault (i.e. they cannot find which cell is defective) do not provide significant improvement in test application time cost and, therefore, are not of great interest.

4.4.1 Optimum Pattern Sequence

As previously mentioned in sub-section 3.5.4, in order to detect and locate all NPSF faults we need a sequence of $(k-1)*2^k + 2^k = k*2^k$ patterns, where k is the number of cells that form a neighborhood. This is because it is required that all the cells of a neighborhood undergo both transitions, ↑ and ↓, for all the possible patterns formed by the contents of the other k-1 cells. The optimum way to apply these patterns is to follow an *Eulerian sequence* of patterns [5], [6]. An Eulerian sequence is a sequence through an *Eulerian graph* which visits each arc exactly once. An Eulerian graph of k-bit patterns is defined as follows:

- There is one node for each unique k-bit pattern, which gives a total of 2^k nodes. Thus, each node represents one (unique) k-bit pattern.

- Two nodes are connected by two arcs, one for each direction, if and only if the corresponding patterns differ by exactly one bit. The total number of arcs is $k*2^k$.

In Figure 4.1 we see a 3-bit Eulerian graph. Testing for all NPSFs is achieved by crossing the Eulerian graph, visiting each arc exactly once. This crossing requires $k*2^k + 1$ write operations and equal number of read operations: one for initialization and the others to visit the arcs. Usually the initial node is the all zero node (0, 0, ... 0).

The choice of the Eulerian sequence serves another important NPSF requirement as well, which is that between two read operations only one cell in the neighborhood must make a transition. For this reason, there is an arc only between the nodes that differ by exactly one bit. Of course a faulty behavior may occur as a result of two or more actions, but these faults are not covered by the NPSF, since it is a static fault model as previously stated.

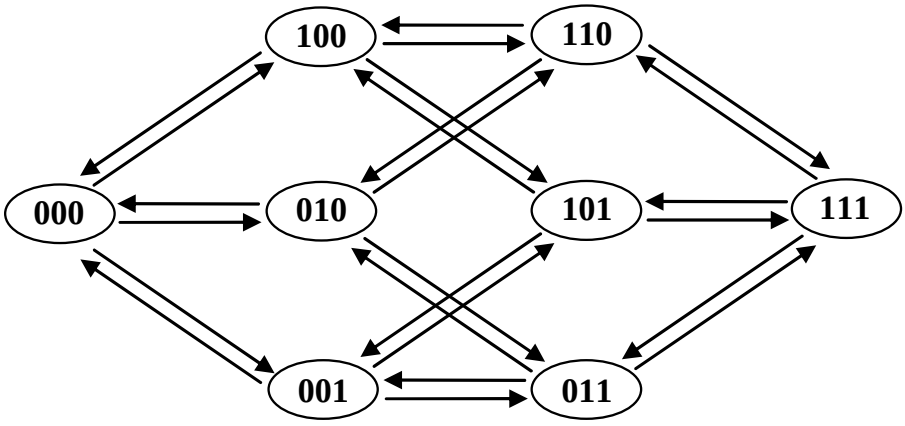


Figure 4.1: A 3-bit Eulerian Graph

4.4.2 NPSF Testing Neighborhoods And Algorithms

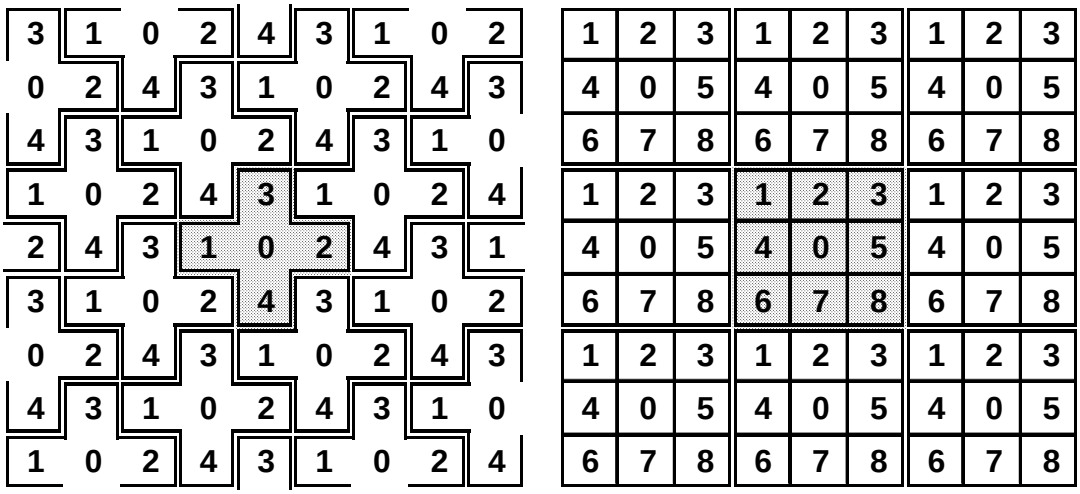
All the characteristics discussed so far are common to the classic NPSF testing algorithms. Their main differences concern the definition of the neighborhood and the method used in order to accelerate the testing process (i.e. minimize the number of write operations). These characteristics will be discussed in this subsection.

The most common neighborhoods are the Type−1 and Type−2 neighborhoods [5], [6]. The Type−1 neighborhood consists of the four adjacent cells to a base cell, these

on the same row and the same column, which form the deleted neighborhood. Thus, this is a five cells neighborhood, as it is shown in Figure 4.2 (a). Consequently, 5-bit patterns are considered and the pertinent pattern sequence consists of $5*2^5 = 160$ patterns.

The Type–2 neighborhood consists of cells within m_1 columns to the west, m_2 rows to the north, m_3 columns to the east and m_4 rows to the south of a base cell. Commonly $m_1=m_2=m_3=m_4=1$ and the neighborhood contains nine cells as it is shown in Figure 4.2 (b). Consequently, the test patterns consist of 9 bits, while the pattern sequence consists of $9*2^9 = 4608$ patterns, a number much larger than the pattern sequence of Type-1 neighborhood.

In Figure 4.2 we also see the *Tiling Method* applied in the Type-1 and Type-2 neighborhoods. According to this method, the whole memory is covered by a group of neighborhoods which do not overlap. Moreover, the numbers from 0 to k-1 are assigned to the cells of each neighborhood in such a way that for every cell number that is considered to correspond to the base cells, the deleted neighborhood of this cell consists of cells where all the rest of the numbers have been assigned. For example, we can clearly see in Figure 4.2(a) that the deleted neighborhood of each cell-0 consists of cells numbered 1, 2, 3 and 4, but we can also easily see that if for example



(a) (b)

Figure 4.2: The Type-1 (a) and Type-2 (b) Neighborhoods

we consider any cell-1 as the base cell, then its deleted neighborhood consists of cells numbered 0, 2, 3, and 4 and so on. The same stands for the Type-2 neighborhood, with cell numbers from 0 to 8.

By using this method we exploit a fundamental principle: every cell is influenced by its neighbouring cells but it is also influencing them. Thus, when for example we make a transition write on all cells numbered 0, we test them for Passive NPSFs but we also test all the other cells for Active NPSFs. In fact, every cell belongs to k neighborhoods: one in which it plays the role of the base cell and k-1 other neighborhoods in which it is a member of the deleted neighborhood.

Based on the above discussion, the application of the test patterns is achieved by assigning each cell number to a bit of the k-bit pattern. Thus, when we visit an arc of the Eulerian graph moving from one node to another, only one cell number makes a transition, since only nodes that differ by exactly one bit are connected with an arc. With this method each pattern application requires only N/k write operations, except from the initialization pattern which writes the whole memory and needs N write operations, where N is the number of cells of the memory. After each new pattern application all the memory is read, which requires N read operations.

The total number of write operations is calculated as follows: N write operations for initialization plus N/k for each pattern, which makes a total of $N + k*2^k *N/k = N*(1+2^k)$ write operations. Similarly, we need N read operations for every one of the $k*2^k +1$ (including the initial) patterns, which makes a total of $N*(k*2^k +1)$ read operations. Thus, the total application cost of a test algorithm using the tiling method with a neighborhood consisting of k cells is $N*(k*2^k+2^k+2)$.

The test algorithm that uses the Type-1 neighborhood and the tiling method is called *TLAPNPSF1T* which stands for Test and Locate Active Passive Neighborhood Sensitive Faults with the Type-1 Neighborhood and the Tiling method [6]. According to the previous discussion the test application time cost of this algorithm is 194N. For the Type-2 neighborhood the pertinent algorithm is called *TLAPNPSF2T* and presents a cost of 5122N operations, which of course is completely prohibitive for use in testing. We should note that even the cost of 194N is considered prohibitive for testing the modern high capacity memories.

The algorithmic description of the above algorithms is rather simple as it is shown in Figure 4.3. It is the same for both algorithms; the only difference is the value of k.

```
write 0 in all cells;
read 0 from all cells;
for  j = 1 to k* 2^k do
{
apply_patern( j );
read_all_cells;
}
```

Figure 4.3: The TLAPNPSF1T and TLAPNPSF2T test algorithms

Except from the tiling method, there is another method for the reduction of the number of write operations which is called *the two group method*. According to this method, we divide the memory in two groups, Group1 and Group2 using the checkerboard pattern. In Figure 4.4 we can see the two group method applied for the Type-1 neighborhood. In each group the base cell is denoted with 'b' while the cells of the deleted neighborhood are A, B, C and D. We see that every cell which is a base cell in Group1, it is a deleted neighborhood cell in Group2 and vice versa. Obviously every deleted neighborhood cell influences four base cells. Every group consists of N/2 base cells and N/2 deleted neighborhood cells; the latter are divided into four sub-groups. Every sub-group consists of N/8 cells and includes either all cells denoted as 'A' in Figure 4.4 or all cells denoted as 'B', or 'C', or 'D'. A test pattern can be applied to all N/2 cells of a group writing all the N/8 cells of a sub-group, and thus reducing the number of write operations by a factor of 4.

The two group method is slightly less effective in terms of test application time than the tiling method, which reduces the number of write operations by a factor of 5. Additionally, it is not a general method since it is not feasible to be applied in all neighborhoods. For instance, it cannot be applied in the Type-2 neighborhood [6]. The pertinent test algorithm for the Type-1 neighborhood is called *TLAPNPSF1G* (Test and Locate Active Passive Neighborhood Sensitive Faults with the Type-1 Neighborhood and the two Group method) and presents a test application time cost of 195.5 N.

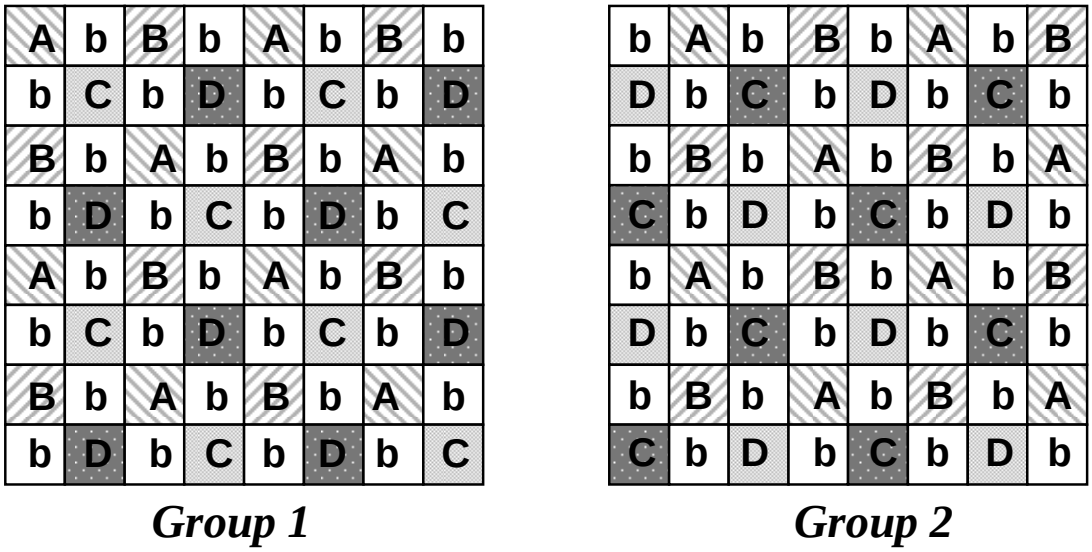


Figure 4.4: The two group method for Type-1 neighborhood..

Finally, except the above early times neighborhoods, a few other neighborhood types have been proposed in the literature. One of them is the *T-Type* which consists of four cells [16][17]. The pertinent test algorithm targets NPSFs combined with the faults described by the *Neighborhood Bit – Line Sensitive Faults – NBLSF*, which are Bit-Line capacitive coupling related faults. Although it is not clearly stated in that work, the cost of the test algorithm for bit-oriented memories is 82N. Recently, we have proposed in [27], [28] another four cell neighborhood, the Delta-Type (Δ-Type) Neighborhood which is dedicated to DRAMs with the folded Bit-Line architecture. This neighborhood is analytically presented in Chapter 5.

4.4.3 Multi-Background March Algorithms For NPSF Testing

Except from the classic NPSF testing algorithms some multi-background march based algorithms have been proposed for NPSF testing. They are called multi-background because as we will see next they create various data patterns (which are also called *data backgrounds*) in the memory. The use of multiple data backgrounds serves the purpose of creating all the necessary neighborhood patterns. We will use the March-12N [20] as an example to illustrate the operation of these algorithms. March-12N is an 8 data background algorithm which uses a march test that requires

12N operations for its application, and which is applied 8 times, one for each data background. Thus, the total cost of March-12N is 96N operations.

The March-12N algorithm consists of the following 6 march elements:

{ ⇳(wa); ⇑ (ra, wb, wa); ⇑ (ra, wb); ⇑ (rb, wa, wb); ⇑ (rb, wa); ⇳(ra) }

M_0 M_1 M_2 M_3 M_4 M_5

Note that the w0/w1 operations are replaced with wa and r0/r1 with ra. The ‘a’ represents the value of the each cell, depending on its location in the memory, for the specific data background. ‘b’ represents the complementary value with respect to ‘a’. Obviously the application of the algorithm for each data background requires the knowledge of ‘a’ for each cell in the specific data background, which is anyway required for the read operations in every test algorithm in order to compare the expected value with the value actually read from each cell. Thus, the M_0 march element performs the initialization of the memory to the actual data background.

Next, we will present the 8 data backgrounds and how they are created by calculating the value ‘a’. The value ‘a’ can be calculated by using the two least significant bits of the row address, denoted by $A_R[0]$ and $A_R[1]$, and the two least significant bits of the column address, $A_C[0]$ and $A_C[1]$. In Figure 4.5 we can see these 8 data backgrounds and the function that calculates the ‘a’. The horizontal pairs (00, 01, 10 and 11) represent the two least significant bits of the column address of each cell while the pertinent vertical pairs (00, 01, 10 and 11) represent the two least significant bits of the row address. The symbol $\oplus$ is the logic XOR operation.

The advantage of these algorithms is that they cover other types of faults in addition to NPSFs. For example, by adding 4N extra operations to the March-12N only for the first data background 1, the new, extended test algorithm covers all address decoder faults (AFs) and all CFs at a cost of 100N operations.

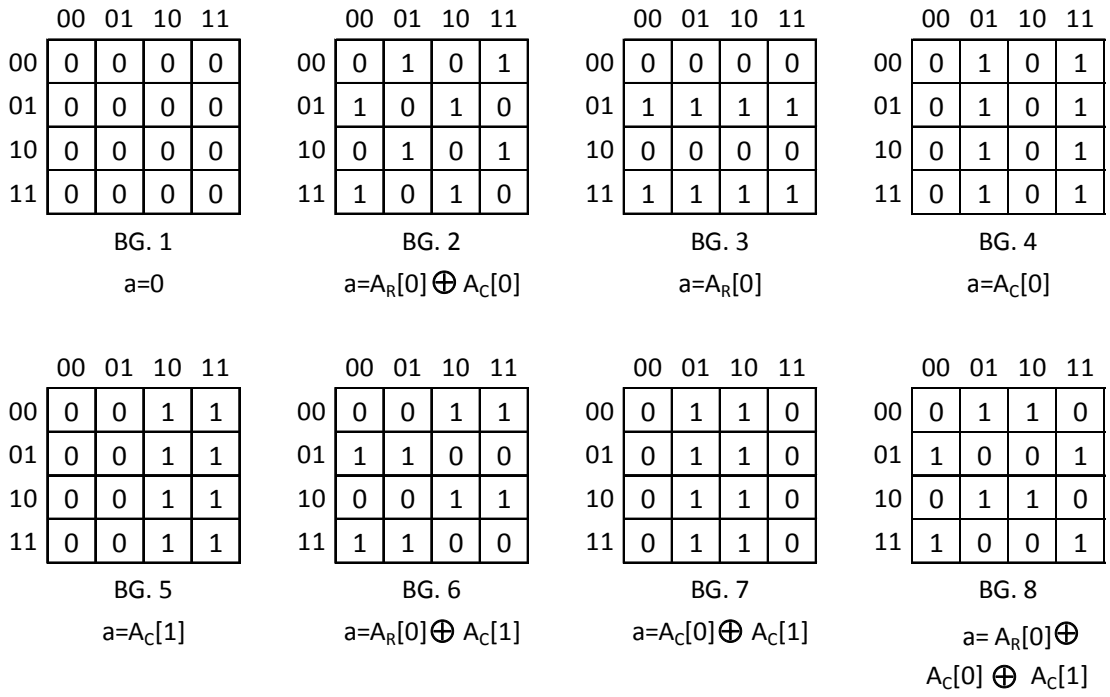


Figure 4.5: The 8 data backgrounds of March-12 Algorithm

However, the impressive cost reduction they present in test application cost compared with the classic TLAPNPSF1T (which has a cost of 194N) comes with a compromise in the quality of NPSFs' testing. Due to the definition of a march test, all march elements are applied to all cells in a row, using either increasing or decreasing address order. For every cell in the memory that is considered as a base cell (unless it belongs to the first or last Word-Line or Bit-Line in the array), there are two cells that have a lower address and two cells that have higher address and belong in the same neighborhood with it. If for example the cell is C[i][j], having a row address i and a column address j, cells C[i-1][j] and C[i][j-1] have a lower address, cells C[i][j+1] and C[i+1][j] have a higher address and all of them belong in the same neighborhood. Thus, if the march element includes at least one write operation, like M_2 in the algorithm March-12N, when it reads the C[i][j] cell both C[i-1][j] and C[i][j-1] will have already make a transition. If more than one write operations are included in the march element, like in M_1, more than two changes in the deleted neighborhood will take place before the cell is read. The same situation would occur if we used a

decreasing address order, where the march element would be applied to cells C[i][j+1] and C[i+1][j] before accessing the base cell, C[i][j].

The transition of two cells in the same neighborhood before the base cell is read may lead to fault masking in case that both transitions activate an Active NPSF. This is because the second activation may restore the correct data to the cell, masking the data loss caused by the first ANPSF activation. Moreover, we should emphasize that the transition of more than one cell in a neighborhood does not comply with the typical definition of the NPSF model. As it is clearly stated in [6], NPSFs are considered to be unlinked by definition because only a single cell transition is allowed to take place in the neighborhood before the cell is read.

Regardless of the masking probability, this difference between multi-background march algorithms and the classic algorithms, like TLAPNPSF1T, for NPSF must be indicated. Note that if in the TLAPNPSF1T algorithm we read the whole memory after the applications of two consecutive patterns, and not after each pattern application, this would reduce the cost of read operations almost to half, from 161N operations to 81N operations, and the total cost of the algorithm would be reduced to 114N.

CHAPTER 5. NPSF TESTING IN FOLDED DRAM ARRAYS

5.1 Abstract

As DRAMs are becoming denser and the technology continues scaling down, more complex fault behaviors emerge; leakage, coupling effects, cell neighborhoods interaction, speed related faults are couple of examples. The Neighborhood Pattern Sensitive Fault (NPSF) model is very suitable to address such faulty behaviors and identify them during the characterization and/or test of new DRAM chips. However, NPSF based test algorithms are extremely time-consuming and therefore economically not affordable for use. In this chapter, we will show how layout/physical-design information can significantly simplify the NPSF fault model and reduce the time complexity of the test algorithms. As a case study, we will use the layout of a folded DRAM array. The NPSF model will be refined and the Δ–Type neighborhood will be introduced. An efficient and low cost test algorithm will be developed; it is more than ×2 cheaper than the traditional NPSF based algorithms.

Even when incorporating the coupling effects of bit lines and word line in the refined model, along with NPSFs, the time complexity of the enhanced version of the proposed test algorithm still remains more than 50% cheaper than the traditional NPSF based algorithms. Therefore, layout driven NPSF testing can become economically affordable and hence suitable for the characterization/test of dense DRAMs in the nano-era.

5.2 Motivation

The continuous technology scaling enables the design of more and more dense DRAMs. This leads to many failure mechanisms; examples are: a) static and dynamic leakage currents [37] like the field-inversion current between adjacent storage cells [5], [31], [32], [38], [39], b) increasing capacitive coupling effect since the spacing between Word-Lines / Bit-Lines decreases [31], [32], [40], [41], c) increasing interaction between adjacent memory cells as their density increase and their size decrease [39][42], etc.

One of the most suitable fault models to deal with the above faults is the well-known *Pattern Sensitive Fault (PSF)* model; it is considered as the most general case of coupling faults where all memory cells are involved [5], [6], [15]. According to this model, the activation of a fault due to a read or write operation in a cell (say a victim-cell) depends on the values stored in all other memory cells (say aggressor-cells) [15]. However, testing DRAMs for PSFs has never become a reality due to the exponential time complexity of the associated test algorithms [16]. Many researchers tried to develop subcategories of the PSF model in order make them testable in practice; such subcategories can be classified as follows: a) the *Neighborhood Pattern Sensitive Fault (NPSF)* model [6] which restricts the aggressor cells to only a limited number of victim-cell neighbors, b) the *Row/Column Pattern Sensitive Fault* model [26], which restricts the aggressor cells to the cells of a single row/column. NPSF seems to be more popular and has been attracted more attention [6], [15] - [24], [43] - [45]; the most important versions of NPSF are based on the Type–1 and Type–2 neighborhoods [6], [15] that have been discussed in Chapter 3. In addition various test algorithms for NPSF in the literature have been presented in Chapter 4.

The study on the state-of-the art in memory testing shows clearly that taking real layout information for NPSF memory test optimization did not get a lot of attention. The reason behind this could be the lack of the layout information by the memory IP developers. This information is considered confidential. However, due to the complexity of the faulty behavior of the DRAMs in the nano-era and the increasing interactions between physically adjacent memory cells [39], [42], it is expected that NPSF may play an important role at least for the characterization (if not also for testing) of the new DRAM chips. Therefore refining the NPSF and developing economically affordable test patters is of great importance.

In this chapter we show how the NPSF can be refined and simplified using the layout information in order to significantly reduce the test time; for our case-study, we use a folded DRAM array layout (which is widely used especially in eDRAMs [31]). A realistic neighborhood, the Δ–Type neighborhood, is introduced. Efficient test and diagnosis algorithms are proposed. These algorithms can be also exploited during the characterization phase of a DRAM generation / product. The characterization of DRAMs require efficient and sophisticated procedures and measurements to predict, analyze and identify all expected performance characteristics (e.g. retention time) of an end product. A comparison with the state-of-the-art reveals that the test time can be reduced at least with a factor of 2, even when the impact of Bit-Line and Word-Line coupling is incorporated in the fault model. Bit-oriented memories are considered in this study. Preliminary publications of this work were in [27] and [28].

5.3 DRAM Memory Array Physical Design

The main advantage of DRAMs is the low area cost due to the simplicity of its one transistor – one capacitor memory cell. Figure 5.1 presents the general layout of a folded DRAM memory array [7], [10], [46], [47]. The memory cell (mcell) size is 4F long (2 lines + 2 spaces) and 2F wide (a line and a space) resulting in a cell area of $8F^2$, where F is the minimum lithographic feature size of the technology defined as one-half of the Word-Line or the Bit-Line pitch. The cells are arranged in pairs back-to-back and share a common Bit-Line contact. The distance between back-to-back storage capacitors is equal to 5F. In our work no assumption is made regarding the

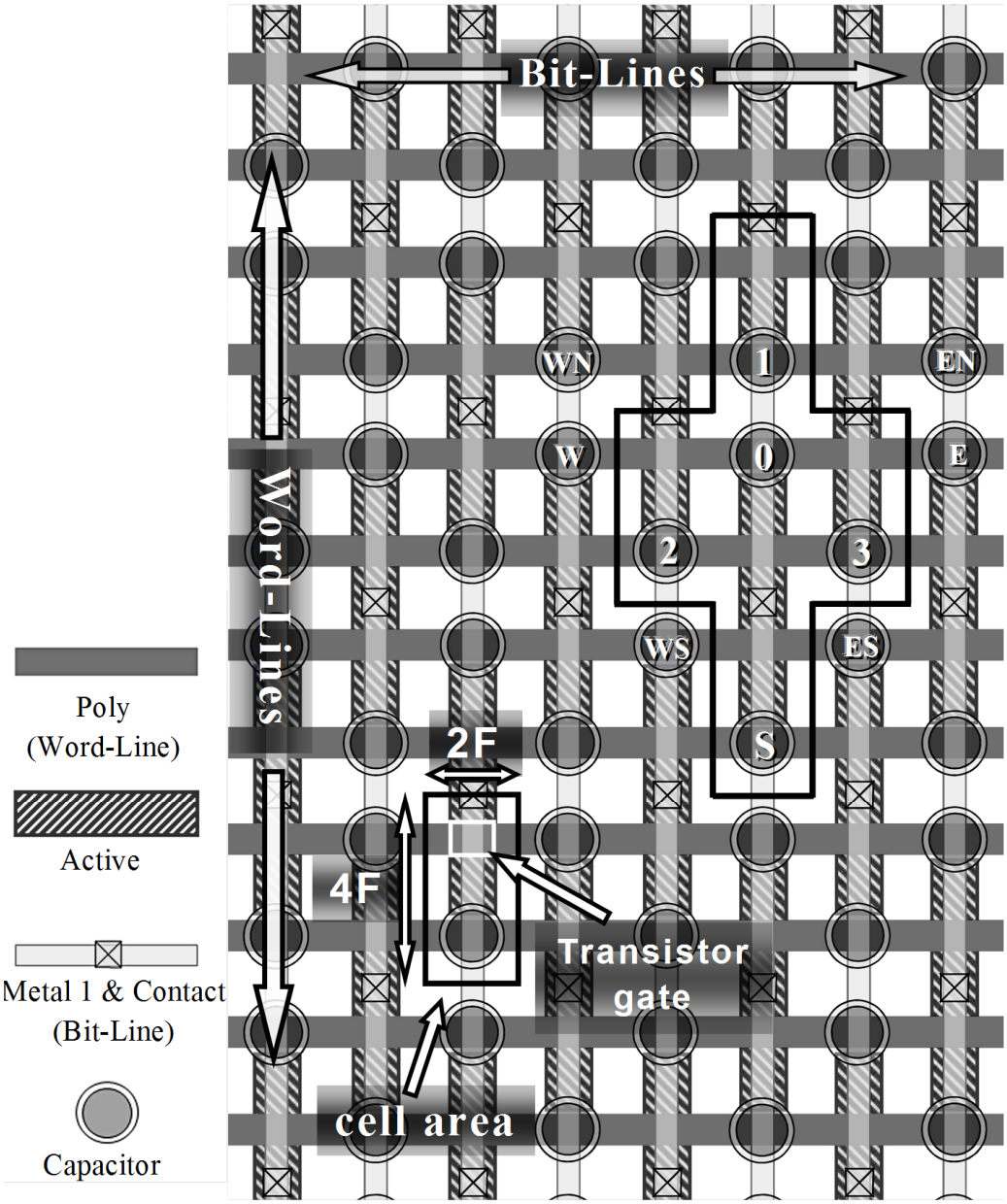


Figure 5.1. Folded memory array DRAM layout

capacitor type; either trench capacitor or stacked capacitor DRAMs can be considered since the capacitor type does not alter the research results.

Due to its folded Bit-Line structure, the $8F^2$ cell layout provides superior signal-to-noise performance (Bit-Line noise rejection) as compared to $6F^2$ [1], [23]. An $8F^2$ architecture uses adjacent data and reference Bit-Lines in read operations providing excellent matching and noise rejection that make them an attractive choice especially in embedded DRAMs [31], [32], [40]. $6F^2$ cell layouts also exist that require two

levels of Bit-Line wiring [1] to achieve equivalent matching and noise rejection. Moreover, in $8F^2$ layouts with folded Bit-Line architecture each sense amplifier serves a total of four Bit-Lines, in contrast to the open Bit-Line architectures used in $6F^2$ layouts where each sense amplifier serves only two Bit-Lines. In the latter memory architecture, the number of sense amplifiers is duplicated and the required silicon area is increased resulting in a significant drawback since sense amplifiers occupy about 10% of the total chip area in modern DRAMs [37].

5.4 Neighborhoods For NPSFs Testing

5.4.1 The Adapted Type−1 Neighborhood

According to the Type−1 neighborhood in Figure 4.2 (a), the deleted neighborhood of a base cell contains two cells sharing the same Word-Line and lie to adjacent Bit-Lines with respect to the base cell and two cells sharing the same Bit-Line and lie to adjacent Word-Lines with respect to the base cell. In Figure 5.1 and considering cell-0 as the base cell, cell-S should belong in the deleted neighborhood since it lies to an adjacent Word-Line and both cells share the same Bit-Line. However, it is not clear which will be the rest three cells of the deleted neighborhood. Cells W and E share the same Word-Line with cell-0 but their Bit-Lines are not adjacent. Moreover, cells 2 and 3 belong to an adjacent Word-Line but do not lie on the same Bit-Line with cell-0. Finally, cell-1 shares the same Bit-Line with cell-0 but their Word-Lines are not adjacent.

A proper deleted neighborhood is formed by the immediate adjacent cells with physical proximity to the base cell, as it is the main idea behind the NPSF model. The classic Type−1 neighborhood fails to fulfill the above requirement since the physical layout of the memory array is not taken into account. According to this, cell-1 is the best candidate since its storage node has the smallest distance of 1F from the storage node of the base cell-0. Afterwards, the storage nodes of cells 2 and 3 have the next smallest distance from the storage node of cell-0, equal to $\sqrt{2}$ F. The other neighboring cells around cell-0 have distances greater than or equal to 2F, so it seems reasonable not to include any of them in the deleted neighborhood. However, since

cell-S shares the same Bit-Line contact with cell-0 and lies to an adjacent Word-Line, we initially decided to include it in the deleted neighborhood although the distance of its storage node is 5F away from the storage node of cell-0. This way a cross-type neighborhood (called adapted Type−1 neighborhood) is formed.

In Figure 5.2 the tiling method for the adapted Type−1 neighborhood is illustrated. As we mentioned in Chapter 4, a common assumption in the open literature is that the memory read and write operations are of equal time cost. Thus, applying the TLAPNPSF1T (Test and Locate APNPS Fault in Type−1 neighborhoods) algorithm analyzed in [6], we can detect and locate all active, passive and static NPSFs related to this neighborhood with a cost of 194N operations, where N is the number of cells in the memory array. This cost is equal to the cost of the traditional Type−1 neighborhood.

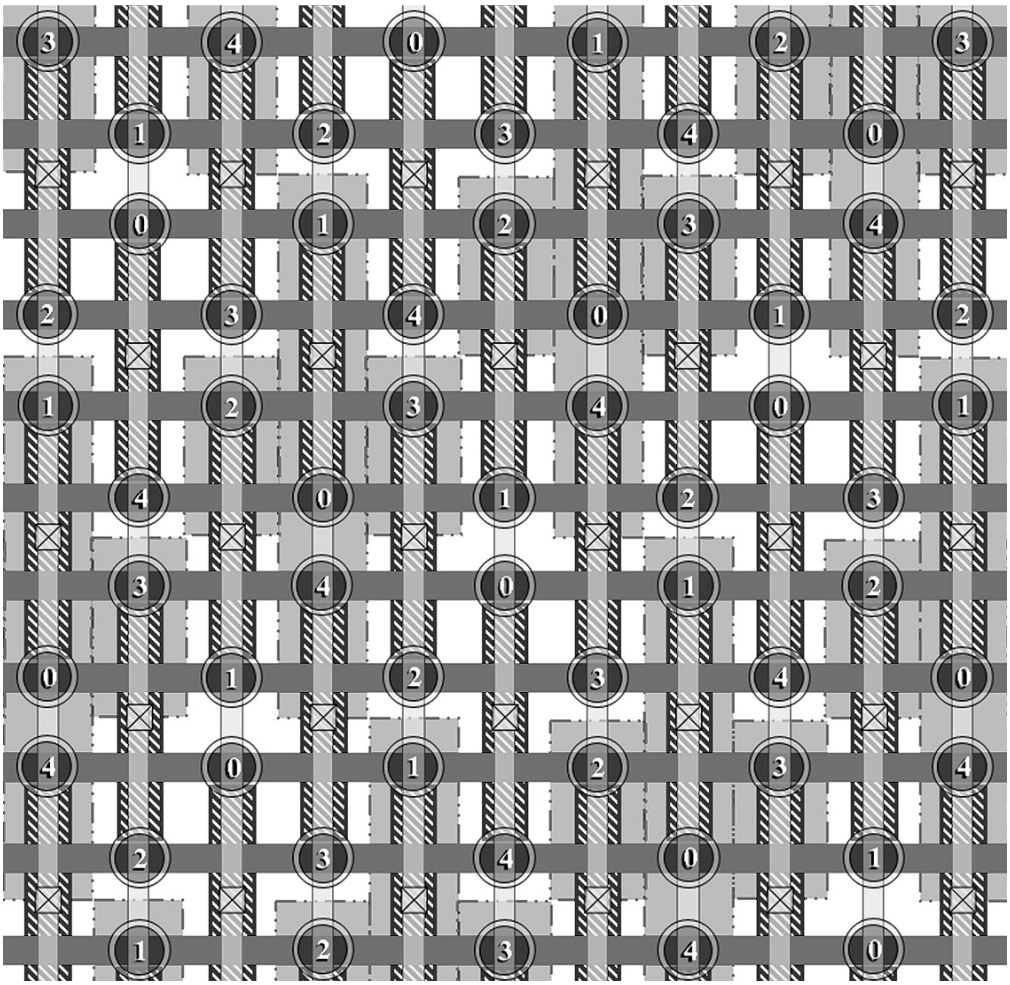


Figure 5.2. The tiling method for the adapted Type−1 neighborhood

5.4.2 The Δ–Type Neighborhood

In the adapted Type–1 neighborhood we have included cell-S in the neighborhood of cell-0. However, the distance of the storage node of cell-S from the corresponding storage node of cell-0 is equal to 5F (see Figure 5.1). This does not sustain the assumption of physical proximity between the two cells. Moreover, the common Bit-Line contact comes in between the two storage nodes at a distance of 2F. Therefore, a possible susceptibility of cell-0 to nearby electrical loads towards the direction of cell-S will be related to the common Bit-Line contact and not to cell-S. Consequently, it is not expected that cell-S will have any realistic contribution in case that it will be included into the deleted neighborhood of the base cell-0.

The above observation motivated us to consider a new neighborhood for NPSF testing consisting of four-cells. This neighborhood is bounded by the triangle in Figure 5.3, where cell-0 is the base cell while cells 1, 2 and 3 form the deleted neighborhood [27], [28], [48], [49]. We call this the *Triangle–Type* or *Delta–Type (Δ–Type)* neighborhood. Consequently, as the Type–1 neighborhood was adopted instead of Type–2 aiming the reduction of the test application cost, in the same

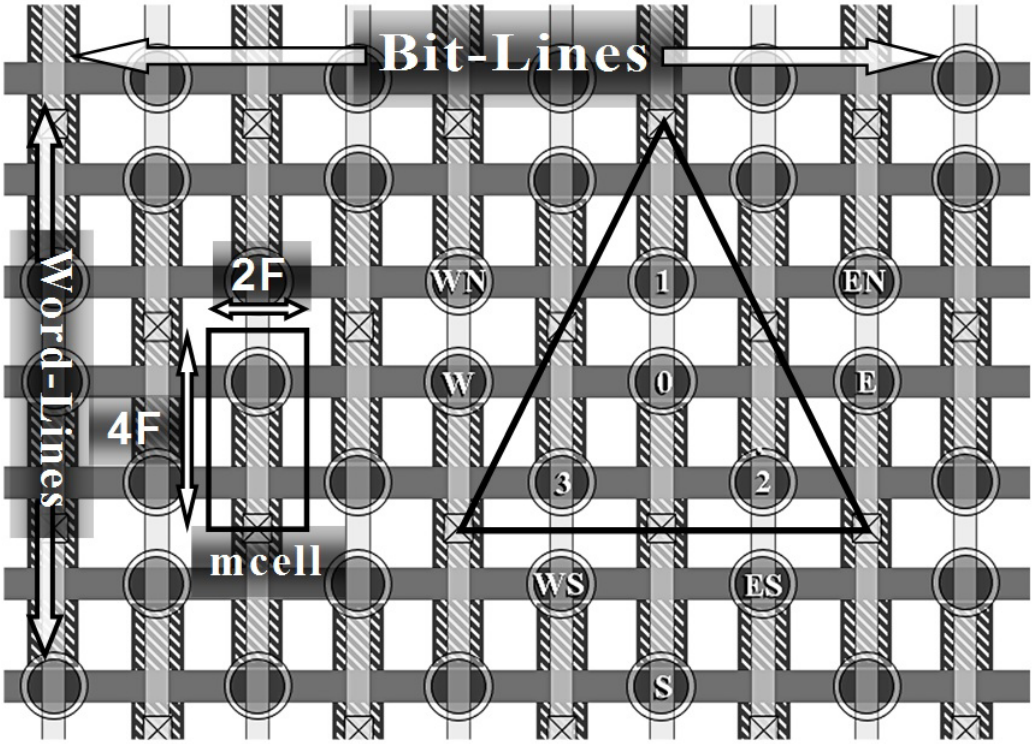


Figure 5.3. The Δ–Type neighborhood

reasoning and based on the earlier discussion we propose the Δ–Type neighborhood instead of Type–1 for NPSF testing. The layout based definition of the Δ–Type neighborhood and the reduced number of deleted neighborhood cells implies realistic fault coverage at reduced test complexity and test application time with respect to neighborhood types (like Type–1 and Type–2) proposed earlier in the open literature.

It is easy to realize that the tiling method is applicable to the Δ–Type neighborhood. In Figure 5.4 the tiling method is illustrated. According to this, the memory is tiled by non-overlapping triangle neighborhoods. There are two kinds of triangle neighborhoods, *up oriented triangles* where the “top” cell-1 lies at the top of the neighborhood, and *down oriented triangles* where the “top” cell-1 lies at the bottom of the neighborhood.

5.4.3 Test Algorithm For The Δ–Type Neighborhood

According to the previous discussion, the Δ–Type neighborhood is formed by

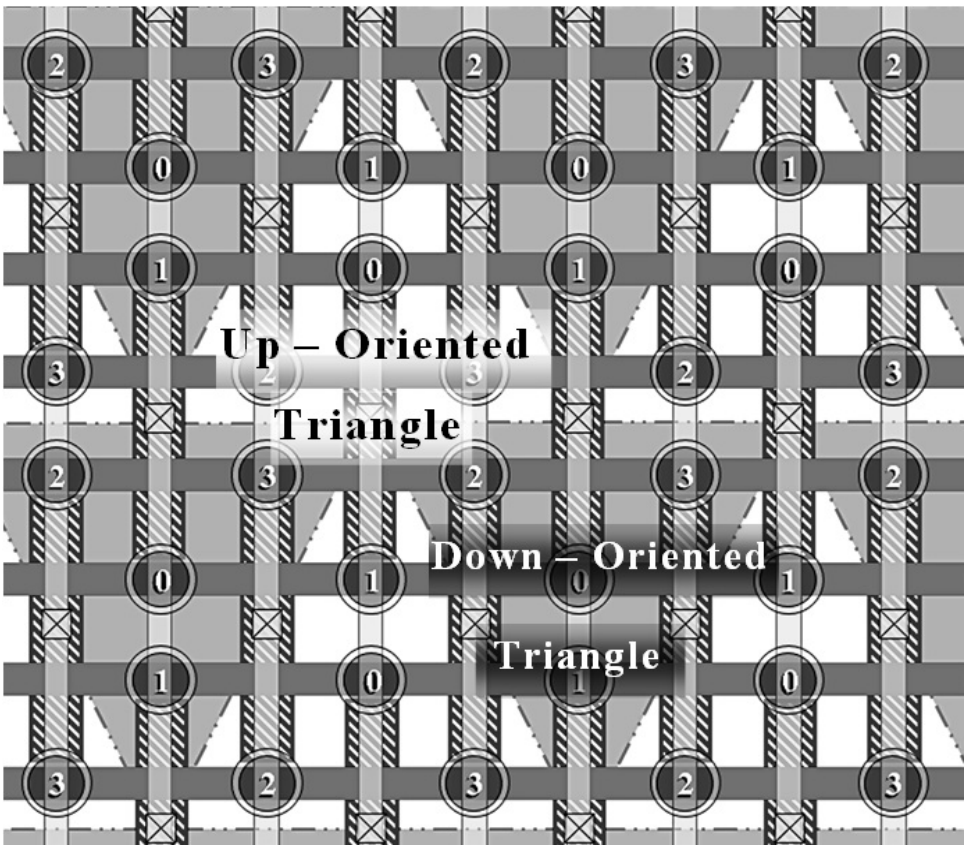


Figure 5.4. Tiling method for the Δ–Type neighborhood

four cells (k=4). Thus, the Eulerian sequence for testing Active Passive and Static NPSF in it consists of $k2^k+1=65$ test patterns and is presented in TABLE 5.1.

The tiling method illustrated in Figure 5.4 is exploited in order to reduce the complexity of the write operation, during the application of the Eulerian test sequence for NPSF testing. Applying an algorithm similar to the TLAPNPSF1T [6], it is feasible to detect and locate all active, passive and static NPSFs related to the Δ–Type neighborhood. This algorithm is shown in Figure 5.5 and we will call it the TLAPNPSFΔT algorithm (Test and Locate Active and Passive NPSFs in Δ–Type neighborhoods). In the notation of Figure 5.5, the write(j) ($1 \le j \le k2^k$) procedure, writes the j-th pattern of the 4-bit Eulerian sequence to the neighborhoods of the memory array.

Note that for each pattern only half of the Word-Lines are involved in the write operations and in each of these Word-Lines half of the cells are written. This is due to

TABLE 5.1. A 4-bit Eulerian Sequence for Δ–Type NPSF Testing

$B_3B_2B_1B_0$	$B_3B_2B_1B_0$	$B_3B_2B_1B_0$	$B_3B_2B_1B_0$
0000	1001	0101	0011
0001	0001	0001	0001
0011	0000	1001	0101
0010	1000	1101	0111
0110	1010	1100	1111
0111	0010	1000	1101
0101	0011	0000	1001
0100	1011	0100	1011
1100	1111	0110	1010
1101	0111	0010	1000
1111	0110	1010	1100
1110	1110	1110	1110
1010	1100	1111	0110
1011	0100	1011	0100
1001	0101	0011	0000
1000	1101	0111	0010
			0000

(1) Initialize all cells with 0; read 0 from all cells;
(2) For j:=1 to 64 do
begin
write(j);
read all cells
end;

Figure 5.5. The TLAPNPSFΔT algorithm

the fact that each Word-Line contains only conjugate cells and only one bit changes between subsequent patterns in the Eulerian sequence. The Eulerian sequence ensures that the required patterns are generated optimally, since the requirement for single bit transition between subsequent patterns without repeating previously generated subsequences is fulfilled [6]. Moreover, a read operation on all cells follows each *write(j)* operation. The algorithm detects and locates all NPSFs because each execution of *write(j)* applies a new pattern in each neighborhood for Active or Passive NPSF testing, while finally all possible 4-bit patterns are written in it for Static NPSF testing.

The algorithm cost is analysed as follows: a) in step (1) there are N write and N read operations, where N is the number of memory cells, and b) in step (2) there are $Nk2^k/k=N2^k$ write operations to apply the patterns of the Eulerian sequence and $Nk2^k$ read operations, where k is the number of cells in the neighbourhood. Thus, there is a total of $N[2+(k+1)2^k]$ operations and since k=4 for the Δ–Type neighbourhood, the cost turns to be 82N operations. This is a significant test application time reduction with respect to the TLAPNPSF1T algorithm for the classic and the adapted Type–1 neighbourhoods discussed in Section III.A, where the corresponding cost is equal to 194N operations [6]. The test cost reduction is 57.7%.

Obviously, the application of the new NPSF testing procedures requires the knowledge of layout information. This information is not always available to IP integrators and system developers. However, built-in self test (BIST) techniques, commonly used in embedded memories, can be exploited by DRAM design houses to address this issue and provide a suitable test solution.

5.4.4 Matrix-Like Representation And Useful Definitions

In order to make the discussion that follows easier, next we will make some observations and provide some definitions regarding the Δ–Type neighborhood. In Figure 5.6, two types of triangle neighborhoods are observed: *up oriented triangles* where the "top" cell-1 lies at the top of the neighborhood, and *down oriented triangles* where the "top" cell-1 lies at the bottom of the neighborhood. Moreover, every Word-Line or Bit-Line contains cells assigned with only two numbers; either 0 and 1 or 2 and 3. A Word-Line (or Bit-Line) that contains cells numbered 0 and 1 will be called a *0-1 Word-Line (Bit-Line)*, while a Word-Line (Bit-Line) that contains cells 2 and 3 will be called *2-3 Word-Line (Bit-Line)*. In the same figure we observe that on every Sense Amplifier one 0-1 and one 2-3 Bit-Line are connected.

Additionally, cells 0 and 1 are defined as *conjugates* to each other and the same stands for cells 2 and 3. The previous definition can be expanded to Word-Lines. Thus, *conjugate Word-Lines* are the Word-Lines which activate conjugate cells. Also, note that the conjugate Word-Lines appear in neighboring pairs.

A pair of neighboring and conjugate Word-Lines will be called *adjoining Word-Lines*. For each pair of adjoining 0-1 Word-Lines, the top Word-Line will be called *upper 0-1 Word-Line* and the bottom will be called *lower 0-1 Word-Line*. The same definition stands for the 2-3 Word-Lines (Figure 5.6). In addition, the cells that share a common bit-line contact will be called *adjoining cells* since they are activated by adjoining Word-Lines. We will also define as *Word-Line coupled cells* the cells that have their Bit-Lines connected to the same SA and their Word-Lines are neighboring and non-adjoining.

Moreover, we will transform the memory array in Figure 5.6 into an array that looks more like a matrix as it is shown in Figure 5.7. According to Figure 5.6, in the folded memory array architecture, for each Word-Line only half of its intersections with the Bit-Lines correspond to a memory cell. Thus, in Figure 5.7 the dark non-enumerated squares of the matrix do not correspond to memory cells.

In Figure 5.7 we can observe our previous statement that each Word-Line activates cells that belong either to 0-1 Bit-Lines or to the 2-3 Bit-Lines. Moreover, we indicate two types of Δ–Type neighborhoods (considering cell-0 as base cell) which are not triangles anymore but cross-like shapes (up-oriented / down oriented).

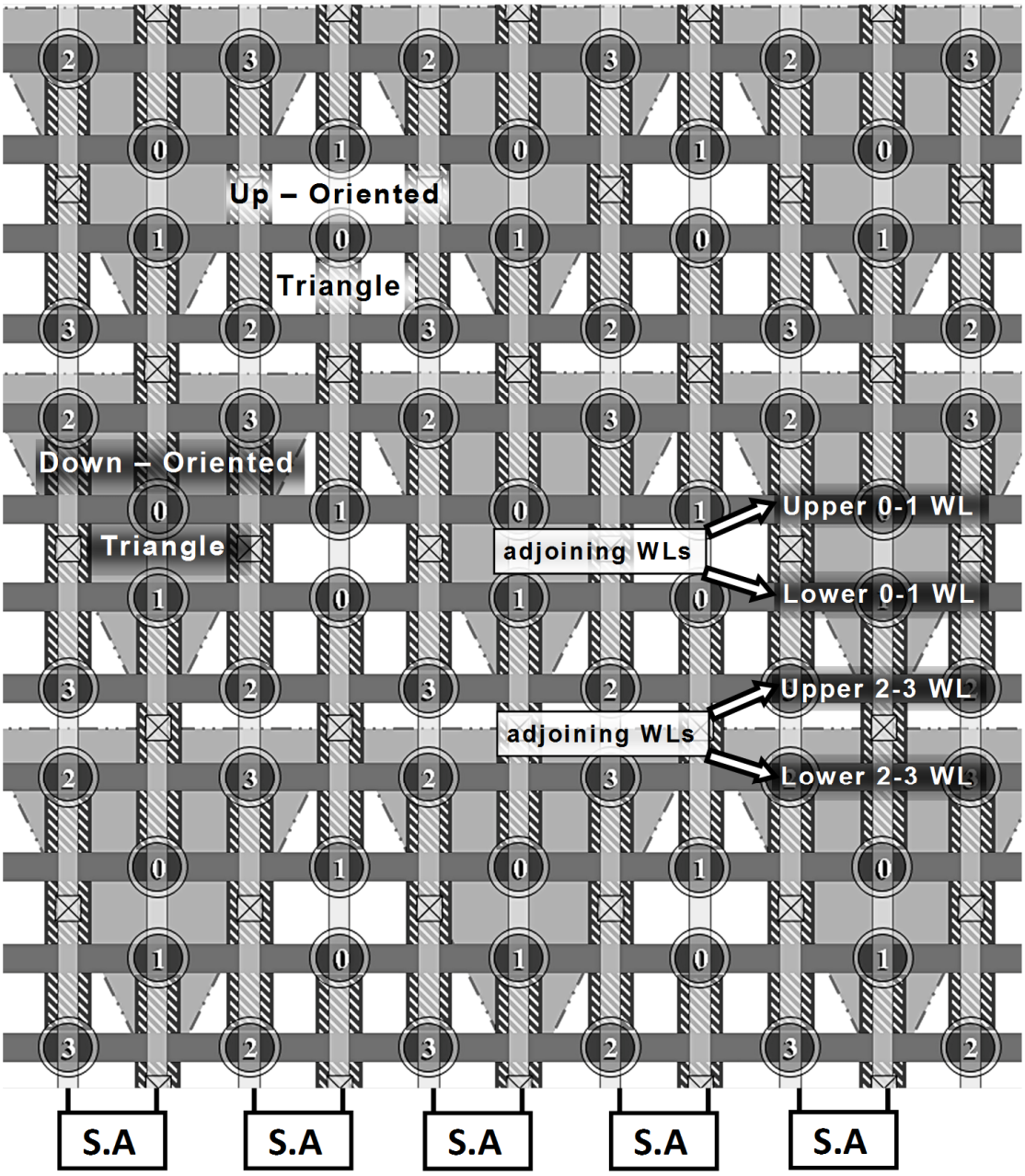


Figure 5.6. Δ-Type neighborhood definitions

In the same figure we can also see how the definitions given in the previous paragraph apply in the matrix-like representation.

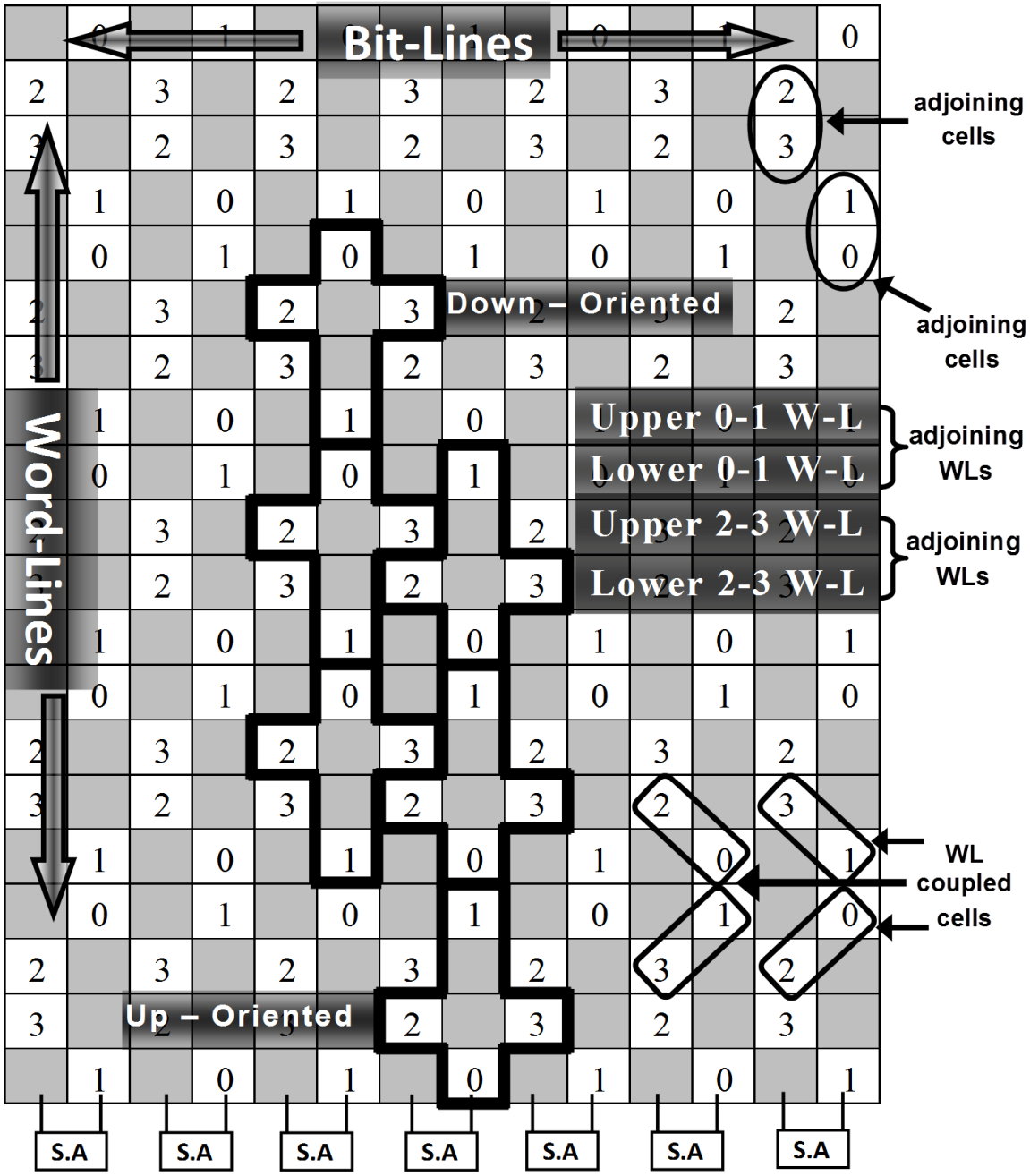


Figure 5.7. Matrix like representation of a folded memory array with the corresponding Δ–Type neighborhoods as cross-like shapes

5.5 Δ–Type Neighborhood And Bit-Line Influence

In the previous subsection we have excluded cell-S (see Figure 5.3) from the deleted neighborhood that affects the base cell-0, since it is not expected to have any realistic contribution as member of this neighborhood. Instead, in case of an interaction of cell-0 towards the direction of cell-S, this will be with the in-between Bit-Line (through the common Bit-Line contact) and not with cell-S. Excessive leakage currents through the access transistor of cell-0 can be responsible for such an interaction.

Various transistor leakage current mechanisms exist [31], [50]. The leakage current of a cell through its access transistor, depends exponentially on the voltage difference between the cell's capacitor and the cell's Bit-Line (weak inversion leakage current) according to the following expression [50]:

$$I_{OFF} = I_0 \cdot e^{\frac{V_{GS} - V_{th\,0} + \eta V_{DS} + \gamma V_{BS}}{n \cdot \upsilon_T}} \cdot \left(1 - e^{-\frac{V_{DS}}{\upsilon_T}} \right) \qquad (5.1)$$

where I_0 is a parameter that depends on the used technology and the size of the transistor, V_{GS}, V_{DS} and V_{BS} denote the transistor's gate, drain and bulk to source voltages respectively, V_{th0} is the zero bias threshold voltage, η is the DIBL coefficient γ is the linearized body effect coefficient, n is the subthreshold swing coefficient and υ_T is the thermal voltage. This leakage current is zero when $V_{DS} = 0$ and is maximized when V_{DS} has its maximum value, V_{DD}. In other words, the leakage current is maximized when the Bit-Line makes a transition to a logic value that is complementary to the logic value stored at the base cell. In case of excessive source-drain leakage current phenomena, due to manufacturing defects or local parameter variations, cell-0 will present a malfunctioning behavior and, therefore, signal transitions on the Bit-Line will influence the contents of cell-0. Consequently, it is essential for a test procedure to cover possible effects related to the Bit-Line activity.

Note that for the adapted Type–1 neighbourhood, the standard TLAPNPSF1T algorithm is inherently capable to cover the Bit-Line influence on cell-0, since it includes cell-S in the deleted neighbourhood. Thus, read/write operations on cell-S during the application of this algorithm provide the necessary Bit-Line transitions to

detect the Bit-Line influence. However, as we have already mentioned, the cost of the TLAPNPSF1T algorithm in test application time is undesirably high. In the following subsection we will see that it is possible to modify the proposed TLAPNPSFΔT algorithm, with a negligible cost increase, and make it capable of covering these faults.

5.5.1 Bit-Line Influence Coverage

As we previously stated, the leakage current is maximized when the Bit-Line makes a transition to a logic value that is complementary to the logic value stored at the base cell. Therefore, the Bit-Line influence can generate an error only when the involved cell has a different value than the final value of the Bit-Line. Consequently, we accept the reasonable assumption that a Bit-Line transition can force a cell to a value which is the same as the value of the Bit-Line at the end of the transition. For example, a Bit-Line making a transition from the precharge voltage (about $V_{DD}/2$) to V_{DD} can only force a cell attached to it to the logic value “1”.

Next, we will make some observations about the Bit-Line transitions. According to the discussion in section 2.4 on the way a DRAM memory operates, whenever a Word-Line is activated for a read operation all the cells attached to it have to make a read – rewrite operation in order to maintain their data. Thus, if we perform a read operation on any 0-1 Word-Line, all 0-1 Bit-Lines will make a transition to the values of the pertinent cells attached to them. Meanwhile, each one of the 2-3 Bit-Lines will make a transition to the complementary value with respect to the value of the pertinent 0-1 bit line that is connected in the same Sense Amplifier (SA).

Similar transitions occur during the write operations, with the difference that the Bit-Line of the cell we are writing will make a transition to the new value (the one we are writing to that cell) while the 2-3 Bit-Line that is connected to the same Sense Amplifier with the cell’s Bit-Line will make a transition to the complementary value. The same observations apply when we perform a read/write operation on a cell belonging to a 2-3 Word-Line, with the difference that the 1-2 and 2-3 Bit-Lines mutually exchange roles.

In order to cover the Bit-Line influence on a cell, we must make sure that the last transition (or series of transitions) its Bit-Line performs before that cell is read is to

the complementary value with respect to the value stored at the cell (the expected value, in the fault free case). Obviously the above situation must be ensured for both possible values stored at the cell (0 and 1). We will now show that the proposed algorithm in Figure 5.5 fulfils the above requirement and covers the Bit-Line influence related faults, provided that it incorporates the following additional features regarding the read operations:

i) Perform the read operations on all cells of each Word-Line before start reading cells of the next Word-Line, and

ii) Access the Word-Lines successively (in order), either from top to bottom or from bottom to top.

The above features are fulfilled by simply using an increasing or decreasing address order for the read operations, which is a common practice for most test algorithms.

The effect of feature (i) is analysed as follows: Assume that we read all the cells of Word-Line k-1 and, thus, all the Bit-Lines make transitions to the pertinent values, depending on the values stored at the cells of this Word-Line as previously described. Next, we perform, the first read operation on Word-Line k, which forces all the cells of this Word-Line to be read for the first time. Assume now that the Word-Line k is a 0-1 Word-Line. From Figure 5.6, and according to the way the Bit-Lines are connected to the sense amplifiers, we can distinguish the following cases:

a) The k-1 Word-Line is also a 0-1 Word-Line. In this case, for every cell of k Word-Line the previous transition of its Bit-Line was to the value of its conjugate cell. Thus, for every cell numbered 0 the previous transition of its Bit-Line was to the value of cell-1 and vice versa.

b) The k-1 Word-Line is a 2-3 Word-Line. In this case, for every cell numbered 0 the previous transition of its Bit-Line is to the complementary value with respect to the value stored at cell 2. Similarly, for every cell numbered 1 the previous transition of its Bit-Line is to the complementary value with respect to the value stored at cell 3.

Similar situations occur if the k Word-Line is a 2-3 Word-Line. Thus, the requirements in order to cover the Bit-Line influence faults are as follows: a) conjugate cells must have complementary values with respect to each other, and b) cells numbered 0 and 2 must store the same value and also cells numbered 1 and 3 must have the same value. Obviously these requirements are met in those patterns that

cells numbered 0 to 3 have respectively the following values: 0, 1, 0, 1 and 1, 0, 1, 0. Note that for every cell number, both possible values 0 and 1 are included in these two patterns.

Of course from this first read operation on Word-Line k we will obtain data from only one cell; the cell we are actually reading. However, if any other cell of the same Word-Line loses its data due to Bit-Line influence, it is easy to realize that the erroneous value will be preserved until the cell is actually read. This is true because due to the faulty read operation of the cell its Bit-Line will make a transition to the erroneous value, which of course is complementary with respect to the expected value. As long as we continue performing operations on the same Word-Line, all the transitions of the faulty cell's Bit-Line will be to the erroneous value. On the other hand, if we perform a read operation on another (random) Word-Line in-between, before the cell is actually read, that read operation may force the Bit-Line of the faulty cell to make a transition towards the correct value, which may lead to fault masking through a possible second activation of the Bit-Line influence mechanism. For this reason it is important to continue the read operations on the same Word-Line until the cell is actually read.

To summarise, the algorithm of Figure 5.5 covers the Bit-Line influence related faults since the above requirements are fulfilled for two of the Eulerian sequence patterns.

5.5.2 Bit-Line Influence And Neighborhood Pattern Interaction

The analysis of the previous subsection does not take into account a possible assistance of the neighborhood pattern on a weak Bit-Line influence mechanism for an error generation. In order to achieve higher fault coverage, we may wish to test the influence of the Bit-Line transitions for every possible combination in the deleted neighborhood of the base cell. In other words, our aim is to ensure that before the first read operation on a specific Word-Line, all pertinent Bit-Lines will make a transition to the complementary value with respect to the value stored on the corresponding cells, for every possible test pattern. Next, we will make the appropriate modifications to the TLAPNPSFΔT algorithm to enhance its fault coverage.

As stated in the previous subsection, only two patterns fulfill the requirements for Bit-Line influence coverage. These are the patterns in which cells numbered 0 to 3 have respectively the following values: 0, 1, 0, 1 and 1, 0, 1, 0. However, by making a slight modification to the algorithm in Figure 5.5 we can ensure that these requirements are fulfilled for more patterns. The modification of the algorithm is as follows: read the 0-1 Word-Lines and the 2-3 Word-Lines separately. In other words, chose one pair of conjugate Word-Lines (0-1 or 2-3), read these Word-Lines first and then read the others. In all these read operations the Word-Lines must be accessed in order, from top to bottom or vice versa.

Due to the above modification the fulfilment of the requirements for Bit-Line influence coverage depends only on the values of conjugate cells. This is true due to the fact that when for example we are reading only 0-1 Word-Lines, then for every cell we are reading the previously read cell on the same Bit-Line was its conjugate according to Figure 5.7 (which dictates the Bit-Line transition direction). Consequently, the Bit-Line influence mechanism can be activated for cells 0 and 1 whenever those cells have complementary values, regardless of the values carried by the cells numbered 2 and 3. Thus, the mechanism can be activated for every possible pattern formed by cells 2 and 3. The same situation occurs when we are reading the 2-3 Bit-Lines.

This first modification of the algorithm ensures Bit-Line influence coverage for the patterns in which the conjugate cells have complementary values. In other words, for every memory cell the Bit-Line influence will be covered for every pattern in which its conjugate cell carries the complementary value. On the other hand, for every pattern in which the conjugate cells have the same value, the Bit-Line will only make transitions from the precharge state to that value during the read operations. For these particular patterns, the Bit-Line influence coverage is inadequate when we wish to also take into account the neighborhood pattern.

In order to achieve the desired fault coverage for those patterns where any pair of conjugate cells (0 and 1 or 2 and 3) takes the same value X, we must add some extra operations to the test algorithm. Next we will describe these extra operations.

Let us define as T-cells the cells that make a transition during the application of the new pattern and C-cells their conjugates. We also define as T-C-Word-Line every Word-Line that contains T and C cells and the same stands for a T-C-Bit-Line. Due to

the fact that in every pattern application all the T-C-Word-Lines are either 0-1 or 2-3 Word-Lines, the first modification of the algorithm can be equivalently described as follows: read first the T-C-Word-Lines and then the non T-C-Word-Lines (or the opposite).

Now consider a new pattern to be written in the memory array where T and C cells will have the same value X. As we mentioned earlier this is the type of patterns for which the Bit-Line influence mechanism is not activated during the testing process (we call them *inert patterns*). Obviously, from the previous pattern application T-cells have the value $\overline{X}$ and C-cells the value X, as it is illustrated in Figure 5.8 (a). In Figure 5.8, A and B are don't care values for the second pair of conjugate cells, since for these cells every combination is feasible.

The operations that will be added to the TLAPNPSFΔT algorithm aim to force all T-C-Bit-Lines to make a transition to the $\overline{X}$ value during the read process and before reading the first cell of each T-C-Word-Line, for each inert pattern, in order to activate the Bit-Line influence mechanism. This transition can be achieved by making a dummy read operation to a predetermined T-C-Word-Line (e.g. the first or the last Word-Line) whose cells have intentionally been written with the $\overline{X}$ value. Consider for example that the memory array consist of M Word-Lines numbered from 1 (the top Word-Line) to M (the bottom Word-Line) In the new algorithm the reading operations are performed in two phases; a) initially we read the half *upper* Word-Lines (Word-Lines numbered from 1 to M/2) using the last T-C-Word-Line for these dummy read operations and b) we read the half *lower* Word-Lines (numbered from M/2+1 to M) using the first T-C-Word-Line for the dummy read operations. These extra operations are performed only for the inert patterns indicated above. The algorithm is presented in Figure 5.9 under the name TLAPNPSFBLIΔT and discussed next in more details.

Step 1: Apply the new pattern to the whole memory except the last two T-C-Word-Lines (Figure 5.8 b). The last T-C Word-Line is excluded because it will be used for the dummy read operations.

Step 2: Write the C-cells cells in the last T-C-Word-Line with the $\overline{X}$ value (the T-cells have already that value – Figure 5.8 c).

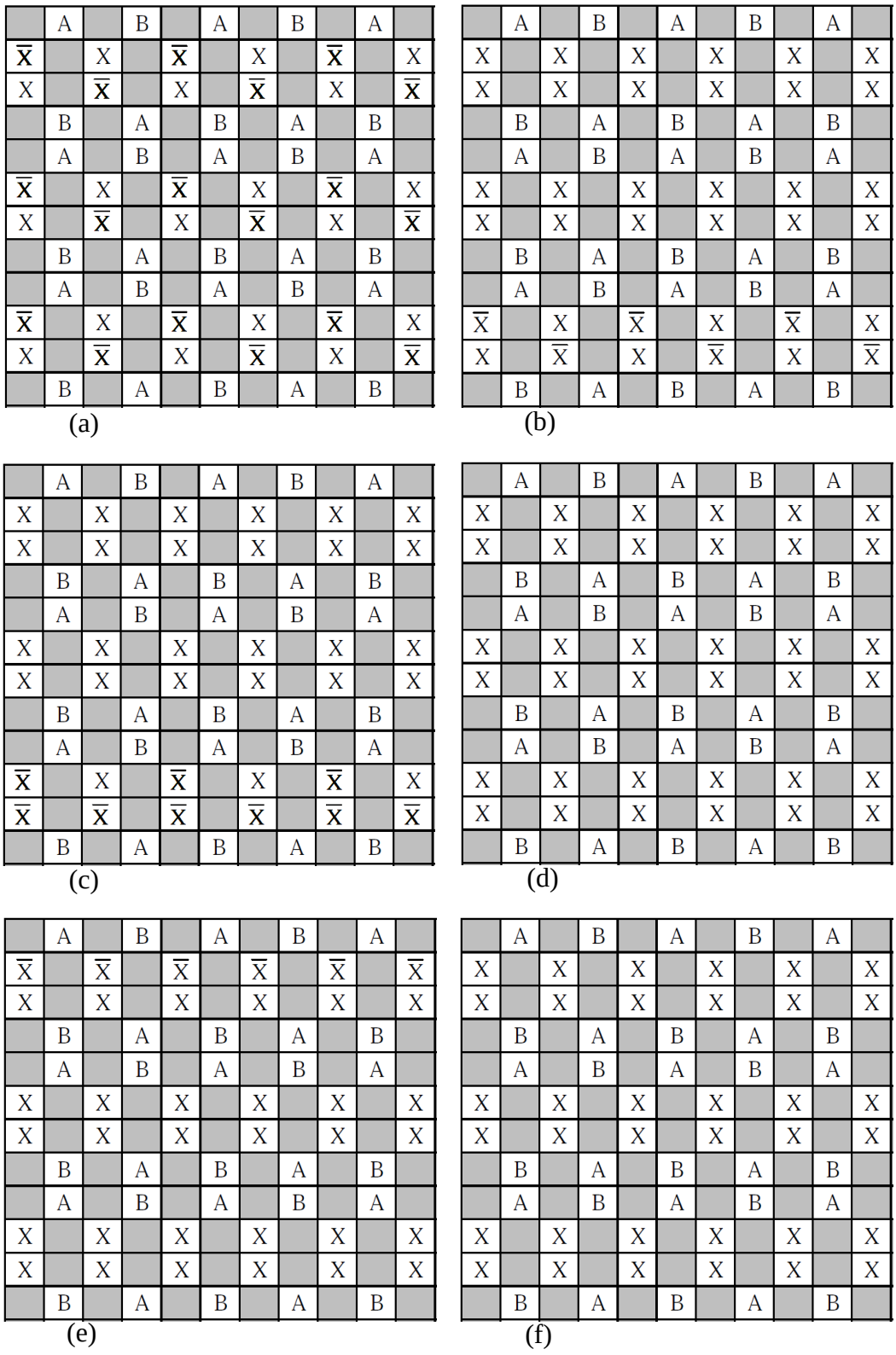


Figure 5.8. Algorithm steps a) initial state, b) step 1, c) step 2, d) step 5, e) step 6, f) step 8

Step 3: Read the half upper Word-Lines (numbered from 1 to M/2). During this reading process, before the first read operation to a T-C-Word-Line make a single read operation to any cell of the last T-C- Word-Line. Thus, we ensure that all T-C-Bit-Lines make a transition to $\overline{X}$ before the first read operation on any cell of the T-C-Word-Lines.

Step 4: Write the C-cells of the last T-C Word-Line to their initial value X. (the pertinent neighborhoods return to the state of Figure 5.8 b).

Step 5: Apply the new pattern to the last two T-C Word-Lines (Figure 5.8 d).

Step 6: Write the first T-C Word-Line to the value $\overline{X}$ (Figure 5.8 e).

Step 7: Read the half lower Word-Lines of the memory (from M/2+1 to M). During this reading process, before the first read operation on a T-C-Word-Line, make a single read operation on a cell of the first T-C- Word-Line.

Step 8: Write back the cells of the first T-C-Word-Line to their initial value X (in order to proceed with the next pattern – Figure 5.8 f).

The cost of the additional operations is calculated as follows, considering that the memory array consists of M Word-Lines and L Bit-Line pairs (thus, a Word-Line contains L cells and a Bit-Line pair contains M cells). For every pattern in which the conjugate cells have the same value (*inert pattern*) we need L/2 (step2) + M/2 (step3) + L/2 (step4) + L (step6) + M/2 (step7) + L (step8) = 3L + M operations. The operations of step 1 and step 5 are not additional to these of the earlier TLAPNPSFΔT algorithm. The number of inert patterns is ($k2^k$ / 2) + 1 = 33 (for k=4) and, therefore, the number of additional operations is 33×(3L + M). Assuming that L = M = $\sqrt{N}$, the number of additional operations is $132\sqrt{N}$ and, therefore, the total cost of the algorithm is $82N+132\sqrt{N}$ operations.

The extra cost of $132\sqrt{N}$ operations to cover the Bit-Line influence mechanism is obviously very small. For example, for a N=256x256 cell memory array, the $\sqrt{N}$ factor is equal to 0.004N). Therefore, the $\sqrt{N}$ factor turns to be negligible and the cost of testing remains almost equal to 82N operations. The new algorithm achieves a cost reduction of 57.7% with respect to the TLAPNPSF1T algorithm, while it retains the same capability of covering the NPSF's and the Bit-Line influence faults.

```
(1) Initialize all cells to 0; read 0 from all cells;
(2) For j:=1 to 64 do
 Begin
  if (in the new pattern T and C cells have the same value X) then
          begin
          write(j) except the last two T-C word-lines;
          write to cells C of the last T-C word-line the X̄ value;
          for (the upper half  word-lines) do
             begin
             read the non T-C word-lines;
             read the T-C-word-lines;
               { before the first reading operation on a
               T-C word-line, read one cell of the last word-line }
             end;
          write to cells C of the last word-line the X value;
          write to cells T of the last 2 word-lines the X value;
          write to cells of the first T-C word-line the X̄ value;
          for (the lower half  word-lines) do
             begin
             read the non T-C word-lines;
             read the T-C-word-lines;
               { before the first reading operation on a
               T-C word-line, read one cell of the first word-line }
             end;
          write to cells of the first T-C word-line the X value;
          end;
else
          begin
          write(j);
          read the non  T-C word-lines;
          read the T-C word-lines;
          end;
 end if;
end;
```

Figure 5.9. The TLAPNPSFBLIΔT algorithm

Finally we should not that the new TLAPNPSFBLIΔT algorithm is capable to cover the occurrence of multiple faults, where a cell is affected by both the NPSF and the Bit-Line influence fault mechanism. Initially, this is true due to the fact that if a pattern in a neighbourhood can force a cell to an erroneous data value (this is a Static NPSF), this pattern will be still present after the Bit-Line influence activation

mechanism (that may correct the data) to re-affect the victim cell before it is read. Therefore, the Bit-Line influence cannot mask the Static NPSF faults. Moreover, the Bit-Line influence faults cannot be masked by Active and Passive NPSF's, since no transition write operations are performed during steps 3 and 7 of the algorithm, at the half memory segment under consideration. Following the above observations, we conclude that: i) if a cell can lose its data due to a Static NPSF, this fault will be detected and located independently of the presence or not of any Bit-Line influence fault, ii) in case that no Static NPSF faults are present, any Bit-Line influence related faults will be detected and located and iii) if no Static NPSF or Bit-Line influence faults are present, all Active and Passive NPSF's will be detected and located. Therefore, the TLAPNPSFBLIΔT algorithm covers the case in which a cell is affected by multiple faults of the above categories.

5.6 Neighborhood Word-Line Sensitive Faults (NWSFs)

In the previous section, we have discussed the Bit-Line transition influence to the base cell during read and write operations. We will now focus on a special case of Bit-Line influence in which the activated Word-Line during a read or write operation is one of the two Word-Lines that are adjacent to the word line of the base cell. This condition is capable to boost a weak (and thus not easily detectable) Bit-Line influence mechanism and produce errors only under specific circumstances, as analyzed next.

5.6.1 NWSF Coverage

It is known that the capacitive coupling between two adjacent Word-Lines is expected to be significantly high, since they are close to each other, they are located on the same conducting layer and are routed side by side for a long distance. The activation of a Word-Line may raise the voltage level of its adjacent Word-Lines, temporarily increasing the leakage current of the cells attached to these adjacent Word-Lines [32], [40], [41], [51], [52]. This leakage current increment depends exponentially both on the induced voltage level to the cell's Word-Line and on the voltage difference between the cell's capacitor and the cell's Bit-Line (weak inversion

leakage current) according to expression (5.1). In other words, the Bit-Line influence is expected to be more effective when an adjacent Word-Line is activated. We will call this type of faults *Neighborhood Word-Line Sensitive Faults (NWSFs)*.

Let us consider cell 0 as the base cell; it can be influenced by two possible NWSFs: a) an operation on the adjoining Word-Line, given that the adjoining cell and the base cell carry complementary values, and b) an operation on the neighbouring 2-3 Word-Line, given that the base cell carries the same value with the cell-2 that belongs to the same neighborhood (assuming, without loss of generality, that the 0-1 and 2-3 Bit-Lines are connected to the SAs as shown in Figure 5.7). Cells 0 and 2 are called *Word-Line coupled cells*. Generally, Word-Line coupled cells have their Bit-Lines connected to the same SA and their Word-Lines are neighbouring and non-adjoining. The other pair of Word-Line coupled cells in a neighborhood is the cells 1 and 3.

It is easy to see that the TLAPNPSFΔT algorithm covers the NWSF faults, provided that both the read and the write operations on the memory are performed as described in Section 5.5.1.: finishing all operations in the current Word-Line before accessing the next and accessing all Word-Lines in order. Next, we will show in details why this is true.

Initially, note that in the patterns where the T and C cells have complementary values the NWSF due to the adjoining Word-Line will be activated. Also, in the patterns where the Word-Line Coupled cells carry the same value the NWSF will be activated due to the non-adjoining Word-Line activation.

However, it is also essential to ensure that there will be an NWSF activation before the Word-Line under consideration is read, because any NWSF activated afterwards will not be detected. In order to be more specific, assume that in the read and write operations the Word-Lines are accessed from top to bottom. If the Word-Line under consideration is an upper (0-1 or 2-3) Word-Line, it will be accessed before its adjoining Word-Line and after its non adjoining neighbouring Word-Line in both the read and write operations. Thus, in this case during the read operations the NWSF (due to the adjoining Word-Line activation) will occur after the Word-Line under consideration is already read. However, this poses no problem because the NWSF mechanism is also activated during the write operations that are performed prior to the read operations.

To summarize, under the presence of the appropriate pattern for an upper (0-1 or 2-3) Word-Line, the read operations can detect a) the NWSF due to the adjoining Word-Line activation which occurred during the write operations and b) the NWSF due to the non-adjoining Word-Line activation which occurred during the read operations while reading the previous Word-Line. The complementary situation occurs in the case of a lower (0-1 or 2-3) Word-Line. Thus, the NWSF is covered by the TLAPNPSFΔT and the cost of testing remains 82N operations in order to detect both NPSFs and NWSFs.

5.6.2 NWSF And Neighborhood Pattern Interaction

According to the above discussion, the TLAPNPSFΔT algorithm effectively covers the NWSF faults. However, in order to cover extreme conditions, we may wish to test NWSFs for every possible combination in the deleted neighborhood of the base cell. In this section we will provide solutions for testing these types of faults.

For the non-adjoining neighbouring Word-Line the NWSF will be activated for every possible pattern, since all related cells are assigned with different numbers. Of course, as stated in the previous section, in this case the NWSF can only cause a fault only when the Word-Line coupled cells have the same value (e.g. cells 0 and 2 in Figure 5.7). Thus, no NWSF can occur due to the non-adjoining neighbouring Word-Line activation when Word-Line coupled cells have complementary values, regardless of the sequence of operations applied by a test algorithm.

On the other hand, the NWSF caused by the adjoining Word-Line activation is related to the adjoining cell, which has the same number as the top cell in the neighborhood. Thus, the NWSF cannot be activated for the patterns where these cells have the same values. The adapted Type-1 neighborhood does not present this problem since it assigns to these cells different numbers.

In order to face this situation we will modify the TLAPNPSFΔT algorithm and enhance it with a small number of proper additional operations (whenever is needed) to ensure the detection of NWSFs (along with NPSFs) for every possible pattern. This way we can achieve the same fault coverage as in the adapted Type−1 neighbourhood but at a considerably lower cost.

Initially, we should note that the only case where two cells with the same number will have a different value is during the write operations that change the value of the cells with that particular number. Thus, in the cases where in the new pattern application the top and the adjoining cell have to make a transition (they play the role of T-cell), they will temporarily have different values since they cannot be written simultaneously. For this reason we focus on the cases where the top and the adjoining cell are T-cells.

Next, the new testing approach is presented. Initially, with A|BT> we denote a down-oriented triangle along with its adjoining cell. In this notation, a read or write operation is performed on the adjoining cell, with value A, while the base cell is carrying the logic value B and the top cell is carrying the logic value T {A, B, T$\in$[0, 1]}. Similarly with <TB|A we refer to an up-oriented triangle along with its adjoining cell. Note that the values A, B and T are the expected values, in a fault-free memory. When the write operations follow the top-down direction, then in up-oriented triangles the top cell is written first with respect to the adjoining cell, while in down oriented triangles the adjoining cell is written first. The opposite situations occur if we chose the down-top direction.

Let us now assume that in the previously applied pattern the adjoining and the top cell carry the value $\overline{X}$ and in the new pattern their value is X while the write operations are performed in the top-down direction. In this case, the only conditions during the write operations are X|Y$\overline{X}$> and <XY|X. Choosing the opposite direction the complementary situations X|YX> and <$\overline{X}$Y|X appear. Thus, by performing the write operations on the T-C-Word-Lines in two phases; a) writing the upper T-C-Word-Lines using the top-down direction and b) writing the lower T-C-Word-Lines using the down-top direction, the X|Y$\overline{X}$> and <$\overline{X}$Y|X conditions are covered.

Moreover, the rest two conditions X|YX> and <XY|X appear during the read operations. However, if, for example, we chose the top-down direction for the read operations, a possible NWSF mechanism activation for a cell in an upper T-C-Word-Line (due to a read operation to the pertinent lower Word-Line) will take place after the read operations on its cells. An earlier activation of the NWSF mechanism, by introducing a single dummy read operation to one cell of every lower T-C-Word-Line before the read operations on the memory, can solve this problem.

The discussion of the previous paragraphs can be summarized in the following points:

- The testing for NWSFs at the C-cell (due to the adjoining neighbouring Word-Line activation) takes place only in the patterns which T and C cells have complementary values. However, by exploiting the fact that the T cells are not written simultaneously, the NWSF mechanism is activated for both values of the top cell (which is also a T-cell) in the neighbourhood of the C-cell. These NWSF activations take place once during the write and once during the read operations. During the test algorithm application all cells will play the role of C-cell; therefore, all NWSFs for every possible pattern and for every cell in the neighbourhood will occur.
- The NWSF activation caused by the non adjoining neighbouring Word-Line activation will occur during the read operations on the non T-C Word-Lines. For this reason during the read operations we first read the non T-C Word-Lines to activate the mechanism and then the T-C Word-Lines.

The extra operations described above ensure the activation of both NWSFs that can affect the C-cell (due to both the adjoining and the non-adjoining Word-Line activation) for every possible neighbourhood pattern.

Based on the previous observations we construct a new algorithm that incorporates the modifications described above. The write and read operations for each pattern of the new algorithm (with the exception of the initial pattern application and the corresponding memory reading operation) are described in the following steps:

Step 1: apply the new pattern to the upper T-C Word-Lines starting from the top T-C Word-Line and moving towards the bottom T-C Word-Line in order.

Step 2: apply the new pattern to the lower T-C Word-Lines starting from the bottom T-C Word-Line and moving towards the top T-C Word-Line in order.

Step 3: (only) if in the new pattern the group of T-cells and the group of C-cells have complementary values, read one cell of every lower T-C Word-Line.

Step 4: read all the cells of the non T-C Word-Lines.

Step 5: read all the cells of the T-C Word-Lines.

Next, in Figure 5.10, the above algorithm is presented under the name TLAPNPWSFΔT. The cost of the extra operations of the new algorithm, considering

again that M is the number of Word-Lines, is calculated as follows; the number of the lower 0-1 Word-Lines is M/4. From the total number of patterns $k2^k + 1 = 65$, the group of T and C cells have complementary values only in 32 patterns. Therefore, the total additional cost is $32 \times M/4 = 8M$. Assuming that $M = \sqrt{N}$, the total cost of the algorithm is $82N + 8\sqrt{N} \approx 82N$ operations for realistic N values. Thus, the cost remains the same as this of the initial TLAPNPSFΔT algorithm for the NPSF testing in the Δ–Type neighborhood. The test application time reduction with respect to the pertinent algorithm presented in [27], where the cost was 114N, is 28.1%.

```
(1)Initialize all cells with 0; read 0 from all cells;
(2)For j:=1 to 64 do
  begin
     write(j) to the upper T-C word-lines;     (top-down direction)
     write(j) to the lower T-C word-lines;     (down-top direction)
     if  (in the new pattern T and C cells have complementary values) then
          read one cell of every lower T-C  word-line;
     end if;
     read the non T-C word-lines;   (top-down direction)
    read the T-C word-lines;          (top-down direction)
  end;
```

Figure 5.10. The TLAPNPWSFΔT algorithm

5.6.3 Multiple Fault Assumption

In the above discussion, we treated NPSFs and NWSFs as independent fault models and this seems to be a realistic approach. Consequently, a possible worst case scenario is the simultaneous occurrence of both NWSF and NPSF faults in a neighborhood which may lead to fault masking. For this reason in this subsection we will construct a new algorithm (which is actually a modified version of the algorithm in Figure 5.10) that can avoid the fault masking situations.

Initially it is easy to see that the static NPSF faults cannot be masked by NWSFs, since if a certain pattern forces a cell to an erroneous data value, the same pattern will be present (and therefore force the cell to that value) until the cell is read. Also, it can be shown that passive NPSF faults cannot be masked by NWSFs. Let us be reminded that a Passive NPSF will not permit the base cell to make a transition write to a value

under a certain deleted neighborhood pattern. In this case, fault masking will occur in case of an NWSF activation that will force the base cell to the correct value, which is the value it would have if the write operation was successful. However, it is reasonable to assume that if the write operation fails to set the cell to the correct value, the NWSF mechanism will also be unable to force the base cell to that value under the same neighborhood pattern. This is true since the capacitive Word-Line coupling can only make the base cell's Word-Line to rise to a value slightly higher than 0V, while in the write operation the cell's Word-Line rises to V_{BOOST} as explained in Section 2.4.

Since static and passive NPSFs cannot be masked by NWSF (it is guaranteed that if they are present they will be detected) there is no need to deal with the case that NWSF is masked by static or passive NPSFs. This is due to the fact that when a static or a passive NPSF is present, it will be detected and the cell will be characterized as faulty independently of any NWSF presence or not. In case that no static or passive NPSF is present, this means that any NWSF will not be masked. Thus, the cell will be correctly characterized as faulty or fault free.

With the same reasoning, it is adequate to ensure that either the active NPSFs (ANPSFs) or the NWSFs (for every possible pattern) are properly detected and that no masking will invalidate the test. As we will see next, it is more convenient to ensure that the NWSF is not masked by the ANPSF. Moreover, we must also ensure that a NWSF activation is not masked by a second NWSF activation.

As stated in the previous section, the full NWSF coverage for every cell is achieved in the test algorithm of Figure 5.10 by considering only the cases in which that cell plays the role of the C-cell. Thus, it is adequate to ensure that for the cell that each time plays the role of the C-cell, no NWSF is masked by either an ANPSF activation or a second NWSF activation.

During the write operations the only NWSF that can occur is the one due to the adjoining cell (which in this case is the T-cell), since only the T-C Word-Lines are accessed. Without loss of generality we assume that the T-cell is the cell numbered 1 and thus the C-cell (which in that case is the base cell) is the cell-0.

We should initially emphasize on the fact that due to the sequence that the write operations are performed, the NWSF activation is always prior to the ANPSF activation for every C-cell in the memory array. Thus the following extra operations

of the algorithm aim to avoid the cases where the ANPSF activation can mask the error caused by a prior NWSF activation.

In more details, as we are writing the upper Word-Lines from top to bottom the base cell of each down oriented triangle is subjected to two influence mechanisms (see Figure 5.7). The first activated mechanism is the NWSF due to the activation of the adjoining cell's Word-Line, provided that in the new pattern the T and C cells have complementary values. The second activated mechanism is the ANPSF due to the transition write on the top cell. Since the latter takes place afterwards, it can possibly mask an activation of the NWSF mechanism by correcting the erroneous data. As we can see in the figure the Word-Line of the base cell is the lower T-C (0-1 in our example) Word-Line between the Word-Lines of the adjoining cell and the top cell. Thus, the solution to this masking problem would be as follows: after writing an upper T-C Word-Line, read the C cells of the neighboring lower T-C Word-Line. A similar solution applies when writing the lower T-C Word-Lines; after writing each lower T-C Word-Line, read the C-cell of the neighboring upper T-C Word-Line. These extra read operations are performed only in the patterns where T and C cells have complementary values.

Obviously during the read operations no ANPSFs may occur. Thus, the only fault masking situation that may occur is due to a second NWSF which is caused by the activation of either the adjoining or the non adjoining neighboring Word-Line and may correct the erroneous data due to an earlier NWSF activation. Moreover, since in the patterns under consideration T and C cells have complementary values, the NWSF due to the adjoining Word-Line activation can only force the C-cell to the erroneous data value and not to the correct one. Thus, we only need to consider the NWSFs caused by the non adjoining neighboring Word-Line activation.

The NWSF influence to the C-cell by the non-adjoining neighboring Word-Line activation can be easily avoided when it can cause fault masking. If during the read operations we read first the T-C and afterwards the non T-C Word-Lines (the latter group includes the non-adjoining Word-Line) the second NWSF influence caused by the non-adjoining neighboring Word-Line activation will occur after the C-cell is already read. This approach will be used in the cases where this NWSF influence can force the C-cell to the correct data value; that is the case where the C-cell and its Word-Line coupled cell have complementary values. On the other hand, when the C-

cell and its Word-Line coupled cell have the same value, the NWSF influence caused by the non-adjoining neighboring Word-Line activation can only force the C-cell to the erroneous data value. In this case, we will read the non T-C Word-Lines before the T-C Word-Lines.

There is also another fault masking situation that must be considered. In order to test the C-cell for NWSFs by the non-adjoining neighboring Word-Line and for every possible pattern (that includes all patterns in which the C-cell and its Word-Line coupled cell carry the same value) we also need to test for the patterns in which T and C cells carry the same value. In this case the NWSF caused by the adjoining Word-Line has the tendency to force the C-cell to the correct data value. Thus, a fault masking situation that may occur is as follows: the C-cell loses its data due to an NWSF caused by the non-adjoining neighboring Word-Line (and thus carries the complementary value with respect to the T-cell) during the read operations on the non T-C Word-Lines and, afterwards, during the read operations on the T-C Word-Lines an NWSF related to the adjoining Word-Line corrects that data.

However, since our new algorithm has the capability to test the NWSF related to the adjoining Word-Line for every possible pattern, no further action is needed to avoid this fault masking situation. This is true due to the fact that if the NWSF related to the adjoining Word-Line has the ability to correct the erroneous data, which means to change the C-cell's value from the complementary with respect to the T-cell value to the same value, then it will also have the ability to produce a faulty behavior in the pattern where the T-cell and C-cell have complementary values, forcing the C-cell to make exactly the same transition. Thus, in every case one faulty behavior will be detected.

The write and read operations for each pattern of the new algorithm (with the exception of the initial pattern application and corresponding memory reading) can be described in the following steps:

Step 1: apply the new pattern to the upper T-C Word-Lines starting from the top T-C Word-Line and moving towards the bottom T-C Word-Line in order. If in the new pattern the groups of T and C cells have complementary values with respect to each other, then after the application of the new pattern to each upper T-C Word-Line, read the C cells of the neighboring lower T-C Word-Line.

Step 2: apply the new pattern to the lower T-C Word-Lines starting from the bottom T-C Word-Line and moving towards the top T-C Word-Line in order. If in the new pattern the group T and C cells have complementary values, then after the application of the new pattern to each lower T-C Word-Line, read the C cells of the neighboring upper T-C Word-Line.

Step 3: if in the new pattern the T-cells group and the C-cells group have complementary values, read one cell of every lower T-C Word-Line.

Step 4: if the C-cells group and its Word-Line coupled cells have the same value, read first the non T-C Word-Lines and then the T-C Word-Lines; else, read first the T-C Word-Lines and then the non T-C Word-Lines.

The new algorithm, under the name TLAPNPWSMFΔT (Test and Locate Active and Passive Neighborhood Pattern and Word–line Sensitive Multiple Faults using the Δ – neighborhood and the Tiling method) is presented in Figure 5.11. The cost of the extra read operations is calculated as follows. From the total number of patterns $k2^k+1 = 65$ (k=4), the pair of T-cells and C-cells have complementary values in 32 patterns. In each one of these 32 patterns we have to read all the C-cells, which represent the 1/4 of the total number of memory cells and thus they are N/4. Thus, the number of extra operations with respect to the TLAPNPWSFΔT algorithm is 32×N/4=8N and the total cost of the new algorithm is $90N + 8\sqrt{N} \approx 90N$. The test application time reduction of the new algorithm with respect to the pertinent algorithm presented in [27], where the cost was 130N, is 30.8%. Moreover, its test application time reduction with respect to the TLAPNPSF1T algorithm, for the classic and the adapted Type–1 neighborhoods, remains significantly high and equal to 53.6%.

5.7 Conclusions

New neighborhood types for NPSF based characterization and/or testing of folded DRAMs are presented in this chapter, where the physical design of the memory array is taken into account. Initially, we introduce a new topology for the well known Type–1 neighborhood, which is adapted to the layout of a folded memory array. Next, a new neighborhood, the Δ–Type neighborhood, consisting of four

memory cells in a triangle fashion, is proposed. For this neighborhood, the coverage of the NPSF faults along with faults related to the influence of the bit–line transitions on the contents of a memory cell are considered. In addition, the neighborhood Word-Line sensitive fault model (NWSF) is introduced and possible strategies to cover NWSFs along with NPSFs are discussed. Four new testing algorithms are presented for NPSF, Bit-Line influence and NWSF faults detection and location, with test application time cost equal to 82N, $82N+132\sqrt{N}$, $82N+8\sqrt{N}$ and $90N+8\sqrt{N}$ respectively. The cost reduction, with respect to the well known TLAPNPSF1T testing algorithm that is used for NPSF testing in Type–1 neighborhoods, is 57.7% for the first algorithm, almost 57.7% for the next two algorithms and 53.6% for the last one.

```
(1) Initialize all cells with 0; read 0 from all cells;
(2) For j:=1 to 64 do
   write(j) to the upper T-C word-lines in each pair;
                                (top-down direction)
      { if in the new pattern the T and C cells have
      complementary values, then during write(j), after the
      write operations on a word-line, read the C cells which
      belong to the neighboring lower T-C word-line }

   write(j) to the lower word-lines in each pair;
                                (down-top direction)
      { if in the new pattern the T and C cells have
      complementary values, then during write(j), after the
      write operations on a word-line, read the C cells which
      belong to the  neighboring upper T-C word-line }

   if (in the new pattern the T and C cells have
                                complementary values) then
            read one cell of the lower word-lines in each pair;
   end if;

   if (in the new pattern the C cells their Word-Line Coupled have
                                complementary values) then
            read the T-C Word-Lines;
            read the non T-C Word-Lines;
 else
            read the non T-C Word-Lines;
            read the T-C Word-Lines;
   end if;
end;
```

Figure 5.11. The TLAPNPWSMFΔT algorithm

CHAPTER 6. THE NEIGHBORHOOD LEAKAGE AND TRANSITION FAULT MODEL (NLTF)

6.1 Abstract

Due to their high density, modern DRAMs are very susceptible to the interactions between adjacent cells, which in turn increases the difficulty and complexity of memory testing. In this chapter we study the interaction mechanisms among neighboring DRAM cells in order to provide an efficient testing solution. According to the open literature, there are two mechanisms responsible for this interaction: leakage currents and cell state transitions. The frequently used Coupling Fault (CF) model is inadequate to model the combined effect of these mechanisms, while testing procedures using the Neighborhood Pattern Sensitive Fault (NPSF) model are not time efficient solutions. Towards this direction, we propose a new fault model, the Neighborhood Leakage and Transition Fault (NLTF) model for DRAMs, which effectively models the faulty behavior related to neighboring cell interference. In addition, we developed a new test algorithm which is based on the NLTF model, and

provides test application time reductions ranging from 68% to 87% with respect to the existing testing algorithms in the literature that are capable to cover the NLTFs. Finally, the proposed algorithm is extended to cover NLTFs in word-oriented DRAMs

6.2 Motivation

Modern nanometer technology DRAMs offer extremely high capacity at elevated performance but also present increased difficulty in testing due to the complexity of the related failure mechanisms. A crucial failure mechanism that makes testing hard is the severe susceptibility of a memory cell to the contents (pattern) and state transitions of the other cells in the memory array. This susceptibility is due to various failure generation mechanisms that are present in high density memory arrays and are related to static and dynamic leakage currents [37] (like the field-inversion current between two adjacent storage cells [5], [38], [39], [53]), as well as cell-to-cell couplings due to cell state transitions [5], [6]. Therefore, it is essential to develop test algorithms which efficiently cover the combined or individual effect of the above failure mechanisms. Note that in nanometer DRAMs, failures related to cell-to-cell couplings that are activated only in the context of very specific data patterns are among the main reasons for test escapes [25]. Also note that, although the importance of these fault generation mechanisms is well known for over thirty years, no cost-effective solution for testing the related faults has been developed until now.

The general fault model for these defects is the *Pattern Sensitive Fault (PSF)* model, which is considered as the most general case of cell interaction faults where all memory cells (N the number) are involved [5], [6]. However, testing DRAMs for PSFs is impractical due to the prohibitive test application time that is required [16].

Alternatively, a more practical and well established memory fault model is the *Neighborhood Pattern Sensitive Fault (NPSF)* model [5], [6], [15]. The NPSF model considers the influence of a memory cell from the contents of its neighboring (adjacent) cells in combination with state transitions in a single cell of this neighborhood. The test application time cost of the algorithms proposed in the literature for NPSF testing makes them almost prohibitive for the high capacity modern DRAMs.

An attractive (from the test application time point of view) and commonly used fault model is the *Coupling Fault (CF)* model. It models interactions between any pair of cells in the memory array; however it is inadequate to cover the combined interactions among all cells in a neighborhood.

In our work we propose a new fault model, the *Neighborhood Leakage and Transition Fault (NLTF)* model. This model considers the nature of realistic interactions among neighboring cells in a DRAM memory array that are related to leakage currents and cell state transitions as well as their combination. Since the CF model is not adequate for neighborhood-related faults and NPSF is not applicable, the NLTF is an attractive solution for efficiently testing faults induced by the neighboring cell interaction. Due to the targeted description of known interactions by the NLTF model, high quality and reduced complexity DRAM testing procedures can be developed. Towards this direction, a suitable low test application time NLTF testing algorithm is presented. Test application time reductions ranging from 68% up to 87%, with respect to test algorithms in the literature that are also capable to detect NLTFs, are reported.

6.3 NPSF And Coupling Faults

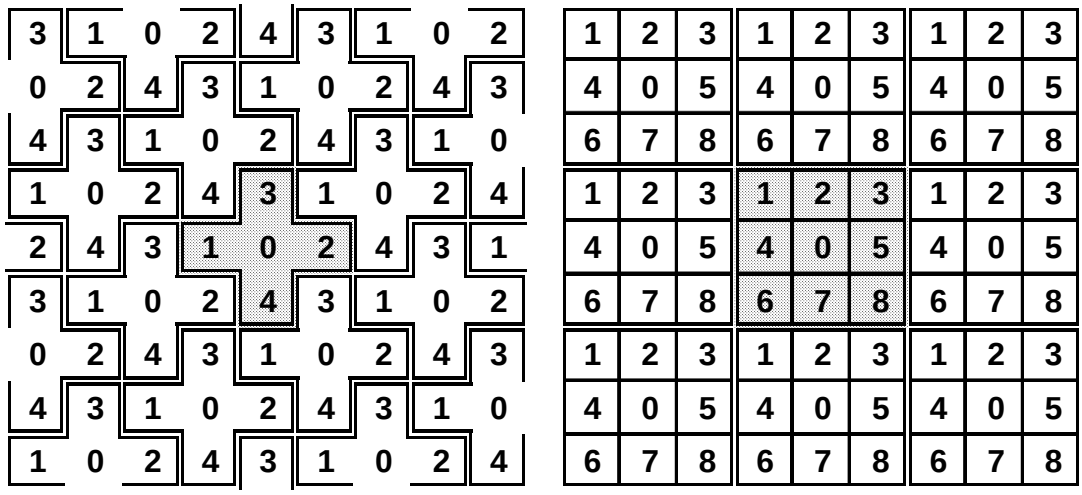
The interaction between neighboring cells is a great concern in memory testing since 80’s. As technology scales down the size and the distances between the memory cells, this interaction turns out to be a significant source of influence for the faulty behavior of a memory. Consequently, one of the main reasons that memory testing becomes harder is the existence of failures related to cell-to-cell couplings that are activated only in the context of very specific data patterns. These pattern–depended failures are among the main reasons for test escapes [25]. In other words, a cell can be influenced by the combination of the data stored at other cells (data pattern) and other cell transitions. Despite the fact that the existence of such a faulty behavior is well known and that the technology scaling makes its test coverage imperative, no viable solution is provided so far by the existing fault models and testing algorithms, as we will see next.

The cell interaction is considered by two well-known fault models: the Coupling Fault (CF) model and the Neighborhood Pattern Sensitive Fault (NPSF) model. The 2

– cell Coupling Fault model [5], [6] is a very popular fault model which deals with the interactions between two cells that each one can be located anywhere in the memory array. The pertinent faults, called 2-cell Coupling Faults, are divided into four sub-categories [5], [6]: a) *Bridging Faults – BFs*, b) *State Coupling Faults – SCF*, c) *Inversion Coupling Faults* – CFin d) *Idempotent Coupling Faults – CFid*. The first two sub-categories, Bridging and State Coupling faults deal with the interaction caused by the cells' states (i.e. the values stored at the cells), while the other two, Inversion and Idempotent coupling faults, deal with the interaction caused by cell's transitions.

The other fault model, the NPSF, covers the influence on a memory cell (called base cell) from the contents of the neighboring cells (neighborhood pattern) combined or not with a single transition write operation on one of these neighboring cells [5], [6]. The cells which are considered to be neighboring to the base cell form the deleted neighborhood while the combination of the base cell and the deleted neighborhood is simply called neighborhood. The NPSF model is divided into three sub-categories: *Static NPSF (SNPSF)*, *Passive NPSF (PNPSF)*, and Active *NPSF (ANPSF)*. The first two sub-categories cover the neighborhood pattern influence on the base (victim) cell while the third (ANPSF) covers the influence of a change in the neighborhood pattern (that is caused by a single-cell transition). Therefore, the ANPSF covers the base cell's susceptibility to a transition write on a neighboring cell combined with the pattern that is formed by the contents of the other neighboring cells.

As we mentioned in Chapter 4, the most common neighborhoods are the Type–1 and Type–2 neighborhoods [5], [6]. The Type–1 neighborhood consists of the four adjacent cells to a base cell, these on the same row and the same column, which form the deleted neighborhood. Thus, this is a five cells neighborhood, as it is depicted in Figure 6.1 (a). The Type–2 neighborhood consists of nine cells as it is shown in Figure 6.1 (b). Aiming to detect in common and at an optimum test application time, active, passive and static NPSFs, every possible neighborhood with base cell every cell of the memory array should be written with the patterns of an *Eulerian* sequence [6]. Moreover, in order to accelerate the test application time, the tiling and the two-group methods have been adopted [5], [6].



(a) (b)

Figure 6.1: The Type-1 (a) and Type-2 (b) Neighborhoods

By its definition, the 2-Cell Coupling fault model does not cover the combined influence of all neighboring cells. In other words, this model cannot take into account the data pattern formed by the contents of all cells that are likely to influence the victim cell. On the other hand, NPSF based testing algorithms that take into account the influence of all neighboring cells are characterized by excessive test application times which make them unattractive for high capacity memory testing. Even considering the Type–1 neighborhood, which requires only 161 test patterns, the classic TLAPNPSF1T and TLAPNPSF1G algorithms have a test application time cost of 194N and 195.5N operations respectively, where N is the number of cells in the memory array [5], [6].

Various efforts to reduce the NPSF test application time cost are presented in the open literature. In Chapter 5 and in [27], [28] a four cells layout-based neighborhood (the Δ–Type neighborhood) is proposed. In [16], [17] another four cell neighborhood (the T–Type neighborhood) is discussed. In both cases above, the pertinent test algorithms present a test application time cost equal to 82N operations.

Moreover, various March-like multi-background tests have been proposed to detect NPSFs. In [44] a 96N (12N x 8 data backgrounds) March test is proposed, while in [45] a 92N (23N x 4 data backgrounds) algorithm is presented. An alternative 96N March algorithm is discussed in [20], [21]. In the application cost of the above

algorithms we should also consider an extra cost in order to write the data backgrounds.

However, the typical March algorithms are inherently not compliant with the NPSF test procedures According to its classic definition, which can be found in [5], [6] and in Chapter 4, a March test algorithm consists of a finite sequence of march elements. Each march element is formed by a series of read and write operations and is applied to all memory cells following an incrementing or a decrementing address direction, without skipping any cells. As a consequence, multiple write operations are performed in a neighborhood before the base cell is read. Let us consider the Type–1 Neighborhood in Figure 6.1 (a). If a march element is applied following the up to down direction, it will access both cells numbered 3 and 1 before accessing cell-0 (which in our example is considered as the base cell). Therefore, if the march element includes a transition write operation, both cells 3 and 1 will make a transition before cell 0 is read. Such a testing procedure is incompliant with the ANPSF testing, which requires the base cell to be read after each single-cell transition in the neighborhood. The possible consequence of this incompliance is a fault masking occurrence due to multiple ANPSF activation. Consequently, all typical March test algorithms, including the multi-background ones for NPSF coverage, are not compliant with the NPSF testing procedures.

The analysis of the previous paragraphs leads us to the conclusion that the existing test algorithms are either inadequate for the type of faults discussed here (the CF-oriented test algorithms) or too expensive in test time application (the NPSF-oriented algorithms). In the sections that follow we will present a viable solution for testing this type of faults.

6.4 Neighborhood Leakage And Transition Faults

The susceptibility of a memory cell to the activity of its neighboring cells can be divided to two sub-categories: a) the susceptibility to the contents of the neighboring cells and b) the susceptibility to the neighboring cell transitions. Next we will analyze separately these sub-categories in order to find a viable solution for the testing of faults related to interactions between adjacent cells.

It is well known that a large portion of memory failures can be attributed to excessive leakage currents, resistive open circuits and improperly set voltages, [6], [37]. The DRAM failures examined in [54] are classified into five defect categories: i) interconnected cells, ii) Word-Line to Word-Line shorts, iii) Word-Line to Bit-Line shorts, iv) Bit-Line to Bit-Line shorts, v) interrupted Bit-Lines. Obviously, among these defect types, the defect mechanism that is related to the interaction between adjacent cells is their resistive interconnection. According to this mechanism, the state of a cell is influenced by the contents of the neighboring cells (i.e. the *Neighborhood Pattern*) due to leakage currents. In [49] the interaction between adjacent cells was modeled using resistive interconnections among the storage nodes of these cells, which result in leakage currents generation between them. This interaction is modeled in [37] using two parasitic transistors, an NPN bipolar and a JFET, which connect the storage nodes of two cells and is called “static leakage mechanism”. Moreover, leakage current mechanisms between adjacent cells, like the field inversion current, are discussed in [5], [6], [38], [39], [53]. Also, in [42] the leakage paths that may occur in a memory are described in details and the leakage path that can cause a direct cell-to-cell interaction is the one that can be established between the storage nodes of the two cells. Note that leakage currents between adjacent cells are always present even in a defect-free memory (due to the high density of cells) and that their strength depends on the neighborhood pattern [31], [55].

To the best of our knowledge, no interaction mechanism related to the neighborhood pattern other than the leakage currents is mentioned in the open literature. From the physics point of view, it also seems reasonable that this is the only interaction mechanism, since the DRAM memory cell is merely a capacitor charged to a certain voltage. Thus, when a memory cell is idle (i.e. no read or write operations are performed on it), the only way to influence another cell is by exchanging charge.

However, the susceptibility of a cell’s contents to the transitions of neighboring cells, is much more complicated, due to the various electromagnetic disturbances that can be induced. As previously stated, the pertinent faulty behavior is modeled by many popular fault models (e.g. CFs, NPSFs) [5], [6]. However, the exact interaction mechanism is not known so far. According to the CF and the NPSF models, all possible combinations between the transition direction of the aggressor cell and the value stored at the victim cell should be considered for testing [5], [6]. In other words,

the cell that is considered as the aggressor must make a low to high (↑) and a high to low (↓) transition for every possible value of the cell that is considered as the victim. This is due to the fact that there is not enough evidence that some of these combinations cannot cause faulty behavior and, therefore, can be excluded from the testing procedure. The NPSF model introduces an extra requirement for the coverage of these faults, as the transitions of the aggressor cell must take place not only for every value of the victim cell (base cell) but also for every possible pattern formed by values stored at the cells in the deleted neighborhood (adjacent cells). By taking into account all possible combinations we ensure complete fault coverage for the transition faults, despite the fact that the exact nature of the interaction is not known.

Considering the above discussion on the susceptibility of a cell to the transitions of its neighboring cells and to the leakage currents with its neighboring cells, we introduce a unified fault model that covers faults that can occur by each one as well as by the combination of these two mechanisms. The latter case turns to be of great importance in nanometer DRAMs [25]. The new fault model is called *Neighborhood Leakage and Transition Fault (NLTF)* model. According to this model, faults in a memory cell are generated either by the leakage currents with the neighboring cells or a neighboring cell transition or the combination of these two mechanisms. Note that the combination of these two mechanisms is crucial when each one of them is quite weak to be independently detectable [25]. Such a combination cannot be ignored since weak leakage currents always exist even in fault free memory arrays and turn out to be of great importance in high density DRAMs.

As it is well known, in testing two fundamental approaches exist: i) a known faulty behavior is described by using a reduced functional fault model [6] and ii) a specific physical mechanism (defect) that is known to cause faulty behavior is studied through simulations (defect oriented testing like in [56]). Actually, the NLTF adopts both approaches, since it combines a physical mechanism, which is the leakage currents between neighboring cells, and a faulty behavior that is caused by neighboring cells' transitions (as it is also described by the reduced functional models CF and NPSF).

It is reasonable to assume, like in the NPSF and the CF models, that also in the NLTF model the read operations and the non-transition write operations cannot

produce a faulty behavior. In addition, only single-cell transition faults are considered, as it is the case for the NPSF and the CF models.

For the detailed definition of the NLTF model it is convenient to follow a similar approach and terminology as in the case of the NPSF (see Chapter 4 and [5], [6]), since it is the only model that takes into account the combined influence of all neighboring cells. Thus, the following definitions for the NLTFs are introduced:

- *Static NLTF (SNLTF)*, where the contents of the base cell are forced to a certain value due to the combination of the leakage currents caused by the pattern that is formed by the contents of all cells in the deleted neighborhood.
- *Passive NLTF (PNLTF)*, where the base cell fails to make a transition to a value due to the combination of the leakage currents caused by the pattern that is formed by the contents of all cells in the deleted neighborhood.
- *Active NLTF (ANLTF)*, where the base cell changes its contents due to the combined influence of a transition write on a cell in the deleted neighborhood and the leakage currents caused by the pattern that is formed by the contents of the other cells in the deleted neighborhood.

The above definitions will enable us in the sections that follow to construct an efficient and low cost test algorithm.

6.5 Neighborhood Max Leakage Patterns

In the previous section we introduced the NLTF model, based on the neighboring cell leakage and cell state transition failure generation mechanisms. Moreover, according to the references presented in that section, the leakage paths between adjacent cells in a DRAM memory array are located between their storage nodes. Obviously, leakage current between two nodes can only exist if there is a voltage difference between these two nodes (adjacent cells in our case), as it is clearly stated in [37], [42], [53]. Consequently, considering the DRAM leakage mechanisms, leakage current between two neighboring cells may exist only if these cells carry complementary data values.

The observations above lead to the reasonable conclusion that the leakage currents affecting a specific cell are maximized when the data stored at all neighboring cells have complementary values with respect to the data stored at the

victim cell [38], [39]. This observation is also considered in the well known and still widely used Checkerboard test algorithm. The Checkerboard algorithm maximizes the leakage currents by assigning complementary values to the neighborhood of a cell [5], [6], [31], [39], [53], [55]. The neighborhood in the Checkerboard algorithm is identical to the Type−1 neighborhood. However, the Checkerboard algorithm is not capable to detect all possible cell state transition related faults and the combined influence of cell state transitions and leakage currents [6].

From now and on, the pertinent neighborhood patterns that maximize the leakage currents from/towards the base cell will be called *Neighborhood MAX Leakage Patterns (NMLPs).* Obviously for a given neighborhood there are two possible NMLPs:

a) the base cell carries the logic value 0 and the deleted neighborhood cells carry the logic value 1 and

b) the base cell carries the logic value 1 and the deleted neighborhood cells carry the logic value 0.

Assume now that in the Type-1 neighborhood the base cell carries the logic value 0. In that case there are $2^4 = 16$ possible patterns for the four cells that form the deleted neighborhood. Under the presence of the first NMLP (a), which is one of these 16 patterns, the leakage currents between the base cell and the deleted neighborhood are greater or at least equal to the leakage currents under the presence of any of the 15 other (non-NMLP) patterns. In other words, the influence of the deleted neighborhood on the base cell under the presence of the NMLP will be greater or at least equal to the influence caused by any other pattern. Therefore, if any of the 15 non-NMLP patterns is capable to activate a faulty behavior, the same faulty behavior will also be activated by the first NMLP at a greater or at least equal strength. A similar observation applies to the second NMLP (b), with respect to the other 15 possible patterns, where the base cell carries the logic value 1.

From the observations discussed in the previous paragraph, we conclude that the non-NMLP patterns can be excluded from an NLTF testing procedure, since any pattern related faulty behavior that can be activated by a non-NMLP pattern will be also activated by the pertinent NMLP pattern. Consequently, an appropriate sequence of NMLP patterns always sensitizes an NLTF fault in case that this fault is sensitized by the standard Eulerian sequence of the patterns used for NPSF testing. In practice,

this is a basic difference on the testing requirements between the NPSF and NLTF models; NPSF demands the use of all possible patterns, while NLTF requires only the use of NMLP patterns for testing. The reduced number of NMLP patterns will allow us to develop a test algorithm that requires a significantly smaller number of operations, compared to existing NPSF algorithms, in order to effectively cover the NLTFs.

It is easy to realize that NPSF testing algorithms can cover the NLTFs, since NPSF uses all possible neighborhood patterns including the NMLPs, but at a prohibitive test application time. However, note that the multi-background March test algorithms [20], [21], [44], [45] which are not completely compliant with the NPSF testing procedures as stated in the previous section, are not suitable for NLTF testing. This is true due to the fact that the new NLTF fault model considers only a single-cell transition in the neighborhood before the base cell is read. This requirement is not fulfilled by the multi-background March test algorithms. Moreover, as earlier stated, the standard March algorithms used for the 2–cell Coupling Faults are not appropriate for NLTF testing.

An efficient testing algorithm for NLTFs, which is based on NMLP patterns, is presented in the section that follows.

6.6 NLTF Test And Locate Algorithm

In order to simplify and make clear and easy the presentation of the proposed algorithm, we introduce the following notations which describe the operations that take place in the memory and the values carried by the base cell and the cells of the deleted neighborhood:

- [B, D]: the base cell carries the logic value B while all cells of its deleted neighborhood carry the logic value D, where B, D $\in \{0, 1\}$ and $B = \overline{D}$.
- [↑, D] (or [↓, D]): the base cell makes a low-to-high transition (↑) (or a high-to-low transition (↓)) while all cells of the deleted neighborhood carry the logic value D.
- [B, D, ↑] (or [B, D, ↓]): one of the cells in the deleted neighborhood makes a ↑ transition (or a ↓ transition) while the rest of the cells of the deleted

neighborhood carry the logic value D and the base cell the value B, where $B = \overline{D}$.

According to the definition of the static, passive and active NLTFs and considering that the NMLP patterns of interest are those where the base cell and the deleted neighborhood carry complementary values, the corresponding test procedure should include the following cases:

- for Static NLTFs: [0, 1] and [1, 0],
- for Passive NLTFs: [↑, 0] and [↓, 1] and
- for Active NLTFs: [0, 1, ↑], [0, 1, ↓], [1, 0, ↑], [1, 0, ↓]

Note that in the case of ANLTFs and in order to cover all pertinent faults, every cell in the deleted neighborhood must make the defined transitions.

The neighborhood under consideration in this work is the commonly used Type–1 neighborhood. Also, the common assumption that the memory layout is a 2-D matrix is used here. This assumption is true for the open Bit-Line architecture, while for the folded Bit-Line architecture an adaptation of the Type-1 neighborhood to the actual layout, like in Chapter 5 and in [27], [28], is needed. Aiming the development of the new test algorithm we will utilize a variation of the well known two-group method that is applicable to the Type–1 neighborhood. According to the two-group method, the memory cells are divided in two groups, group A and group B, using a checkerboard structure, as it is shown in Figure 6.2 Consequently, the deleted neighborhood of a cell in group A consists of four cells that belong to group B and vice versa. Additionally, the cells of group A (B) are numbered from 0 to 3 in such a way that for every cell that belongs to group B (A), its deleted neighborhood consists of cells A0, A1, A2 and A3 (B0, B1, B2 and B3), one of each number.

Following the common notation from [6], the new test algorithm is called TLNLTF1G (Test and Locate Neighborhood Leakage and Transition Faults with the use of the Type–1 neighborhood and the two-group method) and consists of two phases. In Table 6.1 we present the test patterns that are applied in each phase. The first column of each phase indicates the pattern number (p/n). Thus, each phase of the algorithm consists of 17 patterns. The patterns are applied in order according to the pattern sequence in this table. The next four columns of each phase show the values of group A cells (A0, A1, A2 and A3) for every pattern, while the last four columns show the pertinent values of group B cells (B0, B1, B2 and B3).

B0	A3	B2	A1	B0	A3	B2	A1	B0
A0	B1	A2	B3	A0	B1	A2	B3	A0
B2	A1	B0	A3	B2	A1	B0	A3	B2
A2	B3	A0	B1	A2	B3	A0	B1	A2
B0	A3	B2	A1	B0	A3	B2	A1	B0
A0	B1	A2	B3	A0	B1	A2	B3	A0
B2	A1	B0	A3	B2	A1	B0	A3	B2
A2	B3	A0	B1	A2	B3	A0	B1	A2

Figure 6.2. The two-group method

In the first phase (Phase-1) the cells of group A are initialized with the logic value 0 and the cells of group B with the logic value 1. During Phase-1 the cells of group A will make successively a pair of up (↑) and down (↓) transitions in two subsequent patterns, while the cells of group B will make successively a pair of down (↓) and up (↑) transitions in two subsequent patterns. In the second phase (Phase-2) the cells in group A and in group B exchange mutually their roles. The cells of group A are initialized to the logic value 1 while the cells of group B are initialized to the logic value 0 and the pertinent transition write operations follow. In addition, after the application of a pattern proper read operations in the memory array follow in order to detect and locate any activated faults. The two phases of the proposed algorithm are analyzed in more details next.

In Phase-1, after the initialization (application of the first pattern) the whole memory is read. By doing this, the [0, 1] SNLTFs are covered for the group A cells while the [1, 0] SNLTFs are covered for the group B cells. During the second and the third pattern application, cells B0 will make a ↓ and a ↑ transition respectively. After each pattern application, the cells in group A are read. Additionally, after the third pattern application we read the cells B0 as well. As a result, a quarter of the [0, 1, ↓] and [0, 1, ↑] ANLTFs are covered for group A cells, since only B0 cells make a ↓ and

a ↑ transition. Moreover, the PNLTFs [↑, 0] are covered for cells B0 due to the third pattern application.

During the application of the test patterns from 4 to 9, all the rest group B cells (B1, B2 and B3) will make a ↓ and a ↑ transition, while the group A cells carry the logic value 0. After each pattern application all cells in group A are read. Thus, the rest [0, 1, ↓] and [0, 1, ↑] ANLTFs for the cells in group A are covered. Additionally, after each odd pattern application (patterns 5, 7, 9) we read the cells in group B that made a transition (B1, B2 and B3 respectively), since only in these patterns the [↑, 0] PNLTFs are covered for the pertinent cells. Therefore, by applying the patterns 1 to 9, the SNLTFs [0, 1] and [1, 0] are covered for the cells in group A and in group B respectively, the PNLTFs [↑, 0] are covered for the cells in group B and the ANLTFs

TABLE 6.1.: Test Patterns for Each Phase of the Algorithm

	PHASE 1							
	A				B			
p/n	0	1	2	3	0	1	2	3
1	0	0	0	0	1	1	1	1
2	0	0	0	0	**0**	1	1	1
3	0	0	0	0	1	1	1	1
4	0	0	0	0	1	**0**	1	1
5	0	0	0	0	1	1	1	1
6	0	0	0	0	1	1	**0**	1
7	0	0	0	0	1	1	1	1
8	0	0	0	0	1	1	1	**0**
9	0	0	0	0	1	1	1	1
10	**1**	0	0	0	1	1	1	1
11	0	0	0	0	1	1	1	1
12	0	**1**	0	0	1	1	1	1
13	0	0	0	0	1	1	1	1
14	0	0	**1**	0	1	1	1	1
15	0	0	0	0	1	1	1	1
16	0	0	0	**1**	1	1	1	1
17	0	0	0	0	1	1	1	1

	PHASE 2							
	A				B			
p/n	0	1	2	3	0	1	2	3
1	1	1	1	1	0	0	0	0
2	1	1	1	1	**1**	0	0	0
3	1	1	1	1	0	0	0	0
4	1	1	1	1	0	**1**	0	0
5	1	1	1	1	0	0	0	0
6	1	1	1	1	0	0	**1**	0
7	1	1	1	1	0	0	0	0
8	1	1	1	1	0	0	0	**1**
9	1	1	1	1	0	0	0	0
10	**0**	1	1	1	0	0	0	0
11	1	1	1	1	0	0	0	0
12	1	**0**	1	1	0	0	0	0
13	1	1	1	1	0	0	0	0
14	1	1	**0**	1	0	0	0	0
15	1	1	1	1	0	0	0	0
16	1	1	1	**0**	0	0	0	0
17	1	1	1	1	0	0	0	0

[0, 1, ↑] and [0, 1, ↓] are covered for the cells in group A.

Finally, during the application of the test patterns from 10 to 17, all cells in group A will make a ↑ and a ↓ transition while the cells in group B carry the logic value 1. After each pattern application we read all the cells in group B and in the odd patterns (11, 13, 15, 17) the cells in group A that made a transition. The faults covered during the application of this pattern sequence are the ANLTFs [1, 0, ↑] and [1, 0, ↓] for the cells in group B and the PNLTFs [↓, 1] for the cells in group A.

The test operations in Phase-2 are exactly the same with those in Phase-1, with the difference that the cells in group A and in group B exchange roles mutually. Group A cells are initialized to the logic value 1 and group B cells to the logic value 0. The whole memory is read afterwards, so that the SNLTFs [1, 0] and [0, 1] are covered for the group A and the group B cells respectively. During the application of the test patterns from 2 to 9, the cells in group B make a ↑ and a ↓ transition, while the cells in group A carry the logic value 1.

After each pattern application all cells in group A are read. Additionally, after each odd pattern application (patterns 5, 7, 9) we read the cells in group B that made a transition. The faults covered by the sequence of these patterns are the ANLTFs [1, 0, ↑] and [1, 0, ↓] for the cells in group A and the PNLTFs [↓, 1] for the cells in group B.

During the application of the test patterns from 10 to 17 the cells in group A will make a ↓ and a ↑ transition while the cells in group B carry the logic value 0. After each pattern application we read all the cells in group B and in the odd patterns (11, 13, 15, 17) the cells in group A that made a transition. Consequently, the ANLTFs [0, 1, ↑] and [0, 1, ↓] are covered for the cells in group B and the PNLTFs [↑, 0] are covered for the cells in group A. The complete NLTF test algorithm is synopsized in Figure 6.3. The analysis of the fault coverage provided by the sequence of patterns in both phases is presented in Table 6.2. Considering the common assumption that the read and write operations in DRAMs require more or less equal time [6], the test application time cost of the proposed test algorithm is calculated as follows:

i) Write operations: Phase-1 requires N write operations for the initialization (first pattern application) and N/8 write operations for the application of each one of the rest 16 patterns since in each pattern application only one of the 8 cell types (A0,

TABLE 6.2: Fault Coverage Analysis of the Proposed Test Algorithm

PATTERN NUMBER		FAULT COVERAGE	
		GROUP A	GROUP B
PHASE 1	1	static [0, 1]	static [1, 0]
	2-9	active [0, 1, ↑], [0, 1, ↓]	passive [↑, 0]
	10 - 17	passive [↓, 1]	active [1, 0, ↑], [1, 0, ↓]
PHASE 2	1	static [1, 0]	static [0, 1]
	2-9	active [1, 0, ↑], [1, 0, ↓]	passive [↓, 1]
	10 - 17	passive [↑, 0]	active [0, 1, ↑], [0, 1, ↓]

```
{Phase-1}
write 0 to group A cells
write 1 to group B cells
read all the memory

{ operations for patterns 2 to 9}
for i:=0 to 3 do
    begin
    write 0 to cells Bi
    read group A cells
    write 1 to cells Bi
    read group A cells
    read cells Bi
    end;

{ operations for patterns 10 to 17}
for i:=0 to 3 do
    begin
    write 1 to cells Ai
    read group B cells
    write 0 to cells Ai
    read group B cells
    read cells Ai
    end;
```

```
{Phase-2}
write 1 to group A cells
write 0 to group B cells
read all the memory

{ operations for patterns 2 to 9}
for i:=0 to 3 do
    begin
    write 1 to cells Bi
    read group A cells
    write 0 to cells Bi
    read group A cells
    read cells Bi
    end;

{ operations for patterns 10 to 17}
for i:=0 to 3 do
    begin
    write 0 to cells Ai
    read group B cells
    write 1 to cells Ai
    read group B cells
    read cells Ai
    end;
```

Figure 6.3. The proposed NLTF testing algorithm

A1, A2, A3, B0, B1, B2 or B3) is written. This makes a total of N + N×(16/8) = 3N write operations.

ii) Read operations: After the initialization we read the whole memory, which requires N read operations. Then, after the application of each one of the rest 16 patterns we must read all the cells of the group that did not participate in the write operations (N/2 operations) and in the odd patterns all the cells that made a transition (N/8 operations for each pattern, for 8 patterns). This makes a total of N + 16×(N/2) +8×(N/8) = 10N operations. Therefore, the total cost of read and write operations in Phase-1 is 13N.

According to the previous analysis, Phase-2 has exactly the same number of read/write operations (13N). Consequently, the total test application time cost of the proposed NLTF testing algorithm is 26N operations.

The new algorithm provides a significant reduction in the NLTF test application time cost, compared to existing NPSF related algorithms in the literature which can be also exploited for the same purpose. The cost reduction is 86.6%, with respect to the classic TLAPNPSF1T algorithm for the Type–1 neighborhood, where the corresponding cost is equal to 194N operations [6]. Moreover, the cost reduction is 71.7% with regard to the 92N operations of the March-like multi-background algorithm presented in [45]. Finally, the test application time cost is reduced by 68.3% compared to the 82N operation algorithms proposed in [16],[27] and [28]. The above comparisons are synopsized in Table 6.3. Generally, from this table it is obvious that existing algorithms which cover NLTFs, present a prohibitive test time cost that makes them not applicable for high capacity DRAMs. On the other hand, the proposed algorithm provides a test time cost well within the range of commonly used test algorithms for DRAM testing.

TABLE 6.3: Test Application Cost Comparisons

	Test Algorithm				
	Proposed	[6]	[45]	[27][28]	[16]
Required Operations	26N	194N	92N	82N	82N
Reduction	-	86.6%	71.7%	68.3%	68.3%

6.7 Word Oriented Memories

Next we deal with the case of word-oriented memories. According to the usual structure of word-oriented memories, both the number of cells in a Word-Line and the number of cells that form a word are even and perfectly divided between each other, while the cells of every word are distributed in equal distances throughout the whole Word-Line [9]. Consequently, in the discussion that follows we consider the above memory cell array architecture, where the Word-Line length is W and the length of each word in a Word-Line is L.

The new algorithm in Figure 6.3 can be easily modified to cover the NLTFs in word oriented memories. The application of the algorithm in a word-oriented memory, must follow three guidelines: a) the patterns and cell transitions will be exactly the same as in the bit-oriented case, which ensures that the fault coverage of the algorithm is not affected, b) read and write operations of each step are performed only on the words that contain the cells that must be read or written each time, and c) during a new pattern application, if a word contains both, cells that need to make a transition write and cells that must not make a transition, the latter cells must be re-written with the corresponding values according to the used pattern (non transition write in the fault free case).

Due to the requirement of guideline (c), we have to consider the scenario where the re-write operation may lead to fault masking since it actually re-writes the correct data to the pertinent cells. If one of these cells has been affected by the activation of a

fault prior to this re-writing, then this fault will not be detected. Next we will show that in our algorithm no such fault masking can occur.

It is easy to observe that when L≤W/2, no fault masking can occur, since each word consists of either cells in group A or cells in group B. As a result, a transition write operation on a group B cell will not re-write any cell in its neighborhood since the latter are group A cells.

The above observation applies to all cases except from the special case where the word consists of all cells in a Word-Line (L=W). In that case, the word contains both group A and B cells and, therefore, cells that belong in the same neighborhood have to be written simultaneously. For example, in Figure 6.2 if a transition write is performed on cells B0 during the current pattern application, then cells A1 and A3 must make a non transition write and, therefore, will be rewritten with the correct data. Next, assume that the transition write on a B0 cell causes an Active NLTF (ANLTF) on its neighboring A1 (or A3) cell. If the ANLTF is stronger than the non-transition write, then the A1 cell will be written with erroneous data and the fault will be detected during the read operations. If the write operation is stronger, then the cell will always appear to have the correct data and this fault will never manifest itself, either during testing or the normal operation of the memory. Thus, no fault masking can occur. In addition, considering the previous example, note that the A0 and A2 cells that can be also influenced by the transition write on B0 cell, do not belong in the same Word-Line (nor the same word) with B0 and therefore will not be re-written with the correct data during the write operations. Thus, also in that case no fault masking can occur. Note that for the SNLTFs and PNLTFs fault masking is not feasible.

Next, in order to calculate the cost of the algorithm we will examine how the length of the word modifies the number of the required operations. Initially, we assume that L≤W/4. In that case, it is easy to realize (see for example Figure 6.2) that all the cells that form a word have the same number assignment, Ai or Bi (where i=0, 1, 2 or 3). Therefore, during the application of the algorithm described in Figure 6.3 and for every word in the memory, either all the cells of the word participate to the read or write operations of each step, or the whole word does not participate at all. Consequently, no redundant operations appear due to the fact that we have to read or write a whole word instead of a single cell. Thus, the cost of the algorithm is reduced

by a factor of L and is equal to 26N/L. The above discussion for L≤W/4 does not cover the two cases where L=W/2 or L=W.

When L=W/2 we observe (see also Figure 6.2) that each word consists of either group A or group B cells. In that case some redundant write operations are detected in the steps described as “write 0/1 to cells Ai/Bi” and “read cells Ai/Bi” in the algorithm of Figure 6.3. This is true due to the fact that in each pattern application half of the cells that belong to a word must not make a transition write; the exception is for the first pattern (initialization of the memory) in each phase. This means that the number of operations required to apply a pattern is reduced by a factor of L for the first pattern and by a factor of L/2 for all the other patterns. Consequently, the cost of the write operations in each phase of the algorithm is N/L for the initialization pattern, plus 2N/(L/2) = 4N/L for the rest of the 16 patterns, which totals to 5N/L. The number of read operations in each phase is N/L for the first pattern, (N/2)/L for each one of the rest 16 patterns and (N/8)/(L/2) for each one of the 8 odd patterns. Thus, the cost of reading operations is N/L + 16×(N/2)/L + 2×8×(N/8)/L = 11N/L. The total number of operations for each phase is 16N/L. Therefore, the total cost of the algorithm when L=W/2 is 32N/L = 64N/W.

Finally, we calculate the test time cost when L=W. For the application of the patterns of Phase 1 we need N/L operations for the first pattern plus 16×(N/8)/(L/4) = 8N/L operations for the rest of the 16 patterns, a total of 9N/L write operations. Similarly, the number of read operations is N/L for the first pattern, plus 16×(N/2)/(L/2) = 16N/L for the rest of the 16 patterns. No extra reading operations for the odd patterns are required in that case since the cells in both groups A and B are read anyway after every pattern application. Thus, the total number of operations for each phase is 9N/L + 17N/L = 26N/L and the total cost of the algorithm is 52N/L = 52N/W.

Considering the two cases, where L=W/2 or L=W, the cost of the algorithm can be expressed with respect to $\sqrt{N}$. Towards this direction we assume for simplicity that the number of Word-Lines in the memory array is B so that N=B×W (this is a realistic assumption); thus when B=W it results that N/W = $\sqrt{N}$. Considering the above observations, the test application cost in word oriented memories is summarized in Table 6.4.

To the best of our knowledge no word oriented versions of the algorithms that cover NPSF faults exist in the open literature for comparisons.

TABLE 6.4: Test Application Cost For Word-Oriented Memories

L vs W	APPLICATION COST
L≤W/4	26N/L
L=W/2	32N/L = 64N/W = $64\sqrt{N}$ *
L=W	52N/L = 52N/W = $52\sqrt{N}$ *

L = length of the word, W = length of the word-line
* provided that the number of word-lines is also W

6.8 Conclusions

Adjacent memory cell interactions are a reliability threat in high density nanometer DRAMs. The mechanisms underneath these interactions are the neighborhood leakage currents and the cell state transitions. Traditional fault models, like the Coupling Faults or the Neighborhood Pattern Sensitive Faults, are either inadequate or require test time hungry algorithms respectively to cover cases where the combined influence of these mechanisms results in a failure generation.

In this chapter, a new fault model the *Neighborhood Leakage and Transition Fault* (*NLTF*) model, is introduced to deal with the individual or combined effect of the above failure mechanisms on the DRAMs operation. In addition, a low test application time algorithm is proposed, which requires only 26N operations for the detection and location of all NLTF faults in a bit-oriented DRAM memory array (N is the number of cells in the memory array). The achieved test application time reduction, with respect to well known algorithms in the literature that are also capable to detect NLTFs, ranges from 68% to 87%.

The proposed algorithm is easily extended to cover word-oriented memories and in that case the test application time cost ranges from $52\sqrt{N}$ to 26N/L operations (where L is the word length).

The above test application time costs are similar to the test time cost of the well known, but ineffective for NLTFs, March algorithms that are massively used in DRAM testing. Consequently, the proposed testing approach provides a viable solution for nanometer technology, high density and extremely high capacity DRAMs.

CHAPTER 7. RESISTIVE OPEN DEFECTS IN DRAMS: THE CHARGE ACCUMULATION EFFECT

7.1 Abstract

The test complexity of high density DRAMs increases with technology evolution, due to the larger impact of process variations and weak defects. In particular, resistive open defects turn to be a major concern in DRAMs. Our analysis and simulation results show that an important phenomenon exists, we call it *charge accumulation*, which currently is not considered in DRAM testing. Charge accumulation occurs in DRAM cells that suffer from internal resistive opens; a weak value stored at such cells is strengthened when a sequence of read operations is applied to them. Typical DRAM testing procedures (like March tests) fail to provide enhanced coverage of resistive open defects, since they do not consider charge accumulation. In this chapter we provide an effective test algorithm that targets resistive open defects, while considering the Bit-Line imbalance and the charge accumulation mechanisms.

7.2 Motivation

Resistive opens are among the most common defects in integrated circuits (digital or analogue). These defects increase the resistance of conductive lines beyond their expected value, and have been intensively studied in memory circuits, especially in SRAMS [57]. In DRAM memory arrays resistive opens may occur either inside a cell, on a Bit-Line or on a Word-Line. A typical case of an internal resistive open is the STRAP problem, which is a resistive open that occurs in the connection between the trench capacitor and the pass transistor of the cell (the strap connection) [31], [56]. Resistive opens inside a cell are a major problem that is getting harder with technology scaling (e.g. due to thinner bit-line contacts). The problem is even worse in stacked capacitor cells where the Bit-Line contact aspect ratio is higher.

The presence of an internal resistive open in a memory cell affects both write and read operations. A write operation on a faulty DRAM cell will fail to charge the cell capacitor to the desired voltage, leaving the cell with a weak or erroneous value. Moreover, the charge transfer between the cell capacitor and the Bit-Line slows down during read operations, and inevitably, causes incomplete charge sharing. Similar behavior occurs when Bit-Lines contain resistive opens, with the difference that these defects affect more than one cell.

The combined impact of a weak value and an incomplete charge sharing during read operations makes the result of such read operations sensitive to various influence mechanisms. These mechanisms include Bit-Line imbalance (i.e. an imperfect voltage equalization of the Bit-Lines during the precharge operation), the cell's capacitor voltage and the Bit-Line coupling [31], [55], [56], [58]. These factors must be taken into account when enhanced testing algorithms for the detection of resistive open defects are developed.

Various publications addressed the impact of resistive open defects in the presence of the above influence mechanisms. In [49] resistive open and short circuit defects are examined along with imperfect precharging. The STRAP problem along with Bit-Line coupling and process variations is studied in [56], whereas [29], [33] deal with resistive opens, shorts and precharge faults.

However, these works overlooked an important phenomenon. According to our study, the initial weak logic value stored at a cell with an internal resistive open will be significantly enhanced after a successful read operation on this cell. In other words,

after a successful read operation the capacitor's voltage (which was initially close to $V_{DD}/2$ after a write operation – this is a weak logic value) will be significantly moved towards the direction of the correct voltage level (which corresponds to the fault free logic value). Consequently, given that the first read operation is successful, even marginally, the next read operation has a higher probability to succeed because the capacitor's voltage is much closer to the appropriate voltage. After a relatively small number of read operations, the voltage on the capacitor converges to a maximum value for read-1, or to a minimum value for read-0, and, as a result the cell obtains a logic value much stronger than the initially written value. Due to this phenomenon, the increasing or decreasing address order of read/write sequences in commonly used test algorithms tend to mask the faulty behavior. In normal mode operation however, where the operations are performed in a random address sequence, the faulty behavior will eventually manifest itself.

The main contributions of our work in this chapter are as follows.

- Impact analysis of the charge accumulation in DRAM cells with resistive open defects.
- Analysis of defect coverage loss of commonly used March elements when the charge accumulation and Bit-Line imbalance are considered.
- Development of an enhanced low cost test algorithm to detect such defects. This algorithm also detects the defects covered by the widely used Checkerboard test algorithm, and due to its low test application time can effectively replace it.

7.3 Resistive Opens

Resistive open defects in a memory array may occur either inside a cell, on a Bit-Line or on a Word-Line. Within a cell, resistive opens may occur between the transistor and the Bit-Line contact, between the transistor and the capacitor or between the capacitor and the ground [55], as shown in Figure 7.1. A similar effect may occur if the transistor itself is defective or if it cannot be properly turned 'ON' due to a resistive open between its gate and the Word-Line. It is proven that all opens inside a cell cause the same faulty behavior [29].

A resistive open inside a cell decreases the current from/towards the capacitor during read/write operations. Due to this reduction the write operation fails to set the capacitor's voltage at the appropriate level within the available time period. In other words, a write-1 operation will fail to set the capacitor's voltage to the V_{DD} level (supply voltage) and a write-0 operation will fail to set the voltage to 0V. Instead, the capacitor's voltage after the write operation will have a value between 0V and V_{DD}, depending on its initial voltage level (before the write operation) and on the resistance of the resistive open [29].

Moreover, during the read operations, a resistive open slows down charge transfer between the capacitor and the Bit-Line. As a result, the Bit-Line's voltage shift caused by the charge transfer within the available time period is reduced. Additionally, the Bit-Line's voltage shift becomes even smaller due to the weak voltage value in the capacitor. All the effects discussed above can be also caused by a Bit-Line resistive open, with the difference that the latter would influence the behavior of more than one cell.

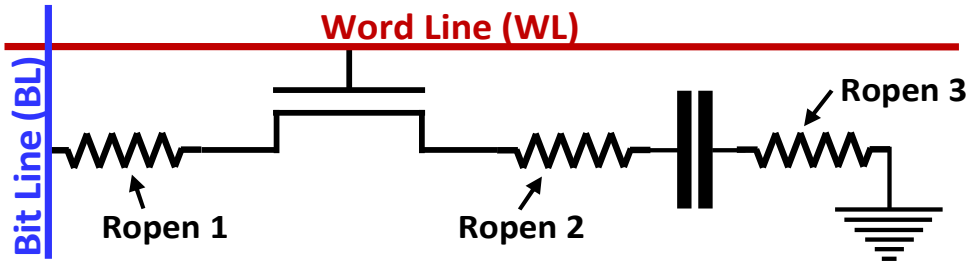


Figure 7.1.: DRAM cell structure and possible locations of resistive opens

7.4 Bit-Line Imbalance

As described in Chapter 2, before reading or writing on a DRAM cell, the Bit-Line attached to it (the cell Bit-Line or in short BLc) and the Bit-Line used for reference (BLr) are (ideally) precharged to the same voltage level, the precharge voltage, which is close to $V_{DD}/2$. If the precharge procedure is not perfect, the two Bit-Lines will present a voltage difference ΔV (*Bit-Line imbalance*) which depends on the previous state of the Bit-Lines.

Consider for example the pair of a cell's Bit-Lines, BLc and BLr, with a Bit-Line imbalance after the precharging phase. If we perform a read-1 or write-1 operation on this cell, BLc will be forced to V_{DD} and BLr will be forced to 0V. At the beginning of the next operation, the precharge circuits will set both Bit-Lines to the precharge voltage. If the precharging procedure (which includes voltage equalization of the two Bit-Lines) is not perfect, the two Bit-Lines will present a voltage difference ΔV with the BLc having a higher voltage than the BLr. Due to this voltage difference, the next read operation on any cell that is connected to BLc will have a tendency to sense '1'. In general, under Bit-Line imbalance a read operation tends to sense the same value that was previously read or written to a cell on the same Bit-Line.

The above tendency can prevent testing algorithms from detecting faults. For example, if a cell is written and immediately read afterwards, the correct value may be read even if the cell's capacitor stored an erroneous value after the write operation; the read operation is predisposed to sense the correct value due to the previous write operation. In order to ensure the detection of the fault, the solution proposed in [49] is to perform (after writing and before reading the cell under consideration) a write operation to another cell in the same Bit-Line with the complementary value. Thus, the memory is biased to sense the wrong value when the weak cell is read, ensuring the activation and detection of the fault. This extra write operation is referred in [49] as a *completing action*.

Although [49] suggests only write operations as completing actions, our simulations show that read operations can be used as completing actions as well. This makes sense because whether we read or write a value to a cell, the involved Bit-Lines make the same transition. This observation will be exploited next in order to reduce the test time of the test algorithms that will be proposed in this chapter.

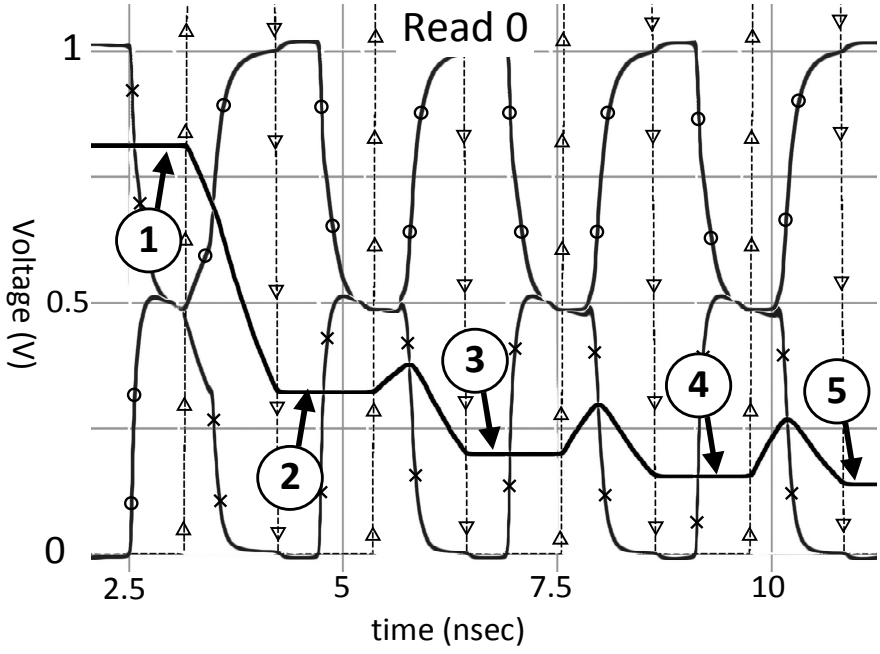
7.5 Charge Accumulation

The defects described in the previous sections are studied through simulation experiments. Without loss of generality, our DRAM simulation model consists of six bit-lines and eight word-lines. Each word-line activates cells attached to three bit-lines. In addition, the capacitance of each bit-line is set to the equivalent of 512 cells, the cell capacitor is set to 30fF and the resistive open is placed between the transistor

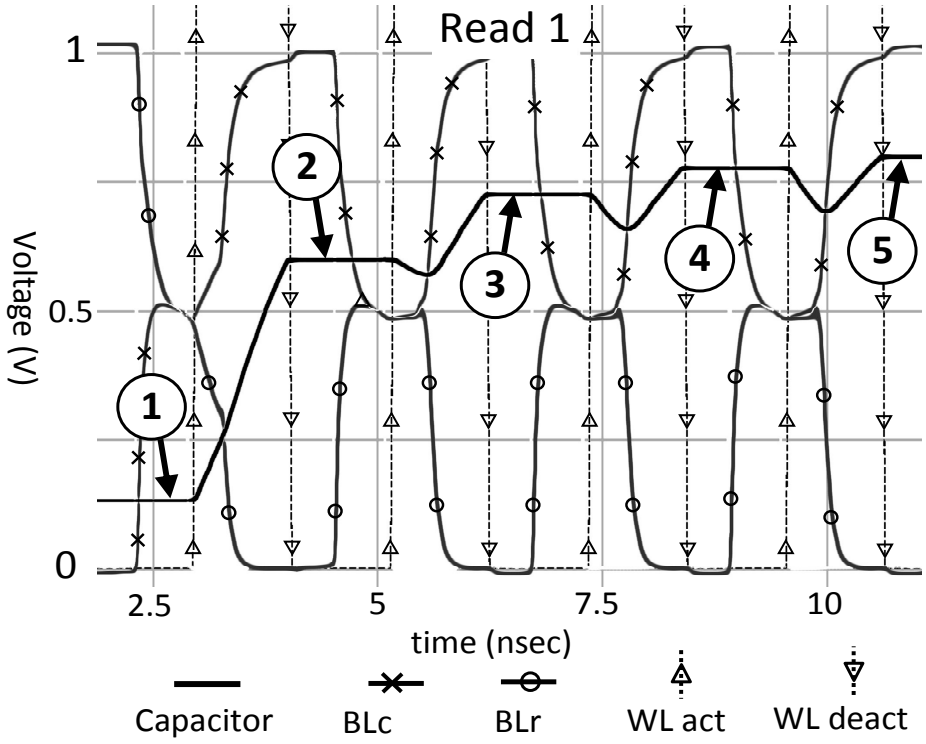
and the capacitor (see Figure 7.1). The 90nm technology of UMC is used with V_{DD}=1V.

The main target of our work in this chapter is to introduce and study the charge accumulation phenomenon, which is the strengthening of an initially weak logic value stored in a cell with a resistive open by applying a series of successful read operations. Note that the Bit-Lines are perfectly balanced in the experiments that are presented in this section. The charge accumulation is illustrated in Figure 7.2a (Figure 7.2b), which shows the cell capacitor voltage after a write-0 (write-1) and three read-0 (read-1) operations using a cell with a 30kΩ resistive open. In both figures the numbered arrows 1-5 show five states of the capacitor's voltage: before the write operation (#1), after the write operation (#2), and after each read operation (#3-5). The figures also show the voltage level of the cell's Bit-Line (BLc) and the reference Bit-Line (BLr), and the activation deactivation transitions of the Word-Line signal (WL act-deact). Note that the Word-Line swing is 1.5V.

As we can see from Figure 7.2, the capacitor's voltage after the write operation is very close to $V_{DD}/2$. In case of read-0 (Figure 7.2a), the capacitor's voltage moves towards 0 after each read operation converging to a minimum voltage value (close to 0.15V in this figure). Similarly, in case of read-1 (Figure 7.2b), the capacitor's voltage moves towards V_{DD} after each read operation converging to a maximum voltage value (close to 0.8V in this figure).



(a)



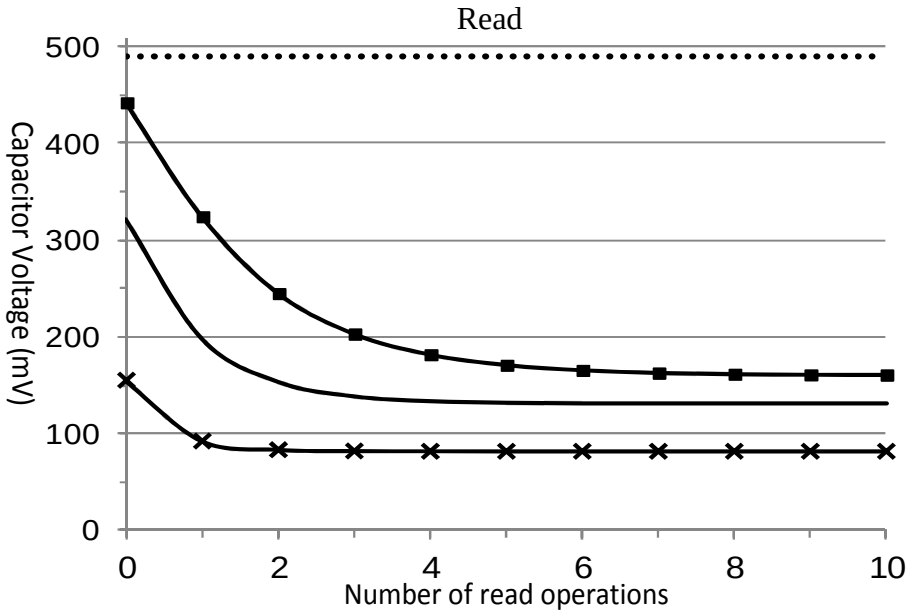
(b)

Figure 7.2: Simulation waveforms showing the charge accumulation for the read-0 (a) and the read-1 (b) cases

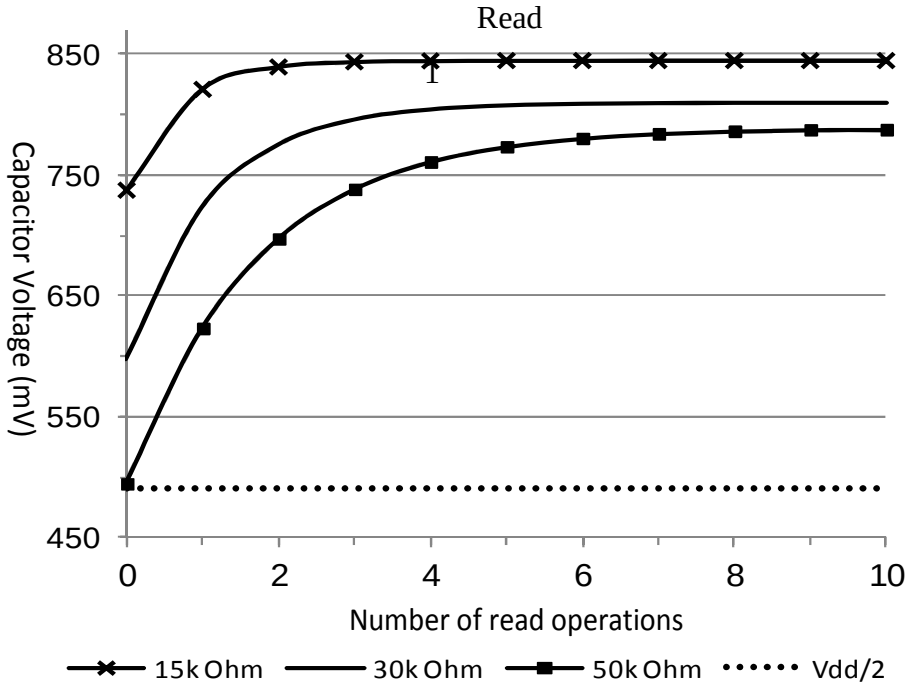
The impact of the charge accumulation phenomenon depends on the value of the resistive open. Figure 7.4a presents simulation results for various resistive open values up to 70 kΩ. The line ‘weak 1’ (weak 0) shows the cell capacitor’s voltage right after a write-1 (write-0) operation, performed on a cell that is initialized with a robust 0 (robust 1). The line robust 1 (0) shows the capacitor’s voltage when a write-1 (0) operation is performed to a cell initialized with a robust 0 (1) and subsequently this write operation is followed by 10 read-1 (0) operations.

Note that read-0 operations always fail for resistive open values of approximately 50 kΩ and higher, while read-1 operations never fail, even when the capacitor’s voltage is below $V_{DD}/2$; note that the Bit-Lines are perfectly balanced. This is due to the fact that the capacitive coupling between the cell’s Word-Line and Bit-Line through the access transistor increases the Bit-Line voltage about 2mV when the word- line is activated. The same trend is observed in [56]. For this reason we cannot set the cell to a robust 0 state for resistive open values above 50 kΩ. This is the reason why in Figure 7.4a no measurements exist for the robust 0 case above 50 kΩ. Nevertheless, a read-0 operation for resistive opens over 50 kΩ may succeed under the presence of a small Bit-Line imbalance, as we will see in the sections that follow.

The importance of the charge accumulation phenomenon becomes more obvious if instead of the capacitor’s voltage, the voltage difference between the cell’s Bit-Line and the reference Bit-Line is considered (see Figure 7.4b). We observe that a cell with a robust value is always able to produce a significant voltage shift on the Bit-Line, in contrast to a cell with a weak value.

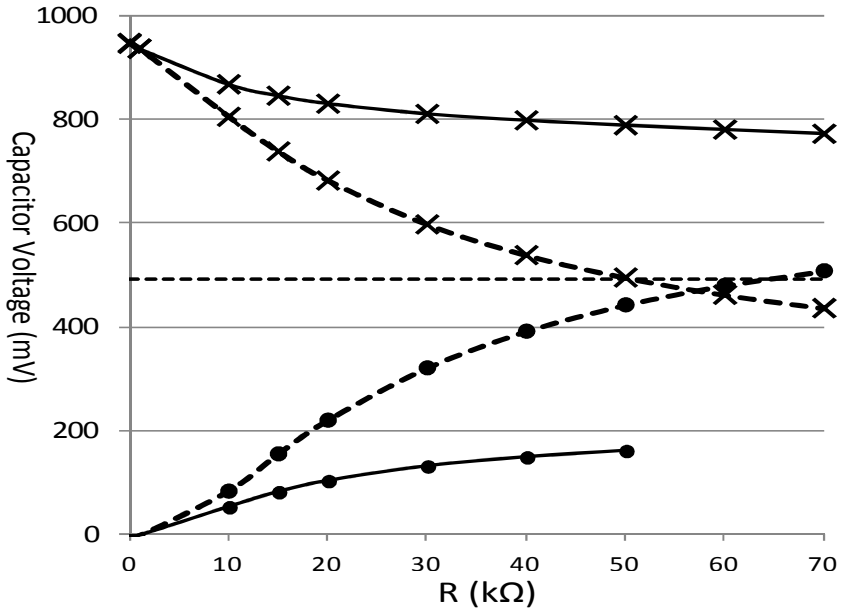


(a)

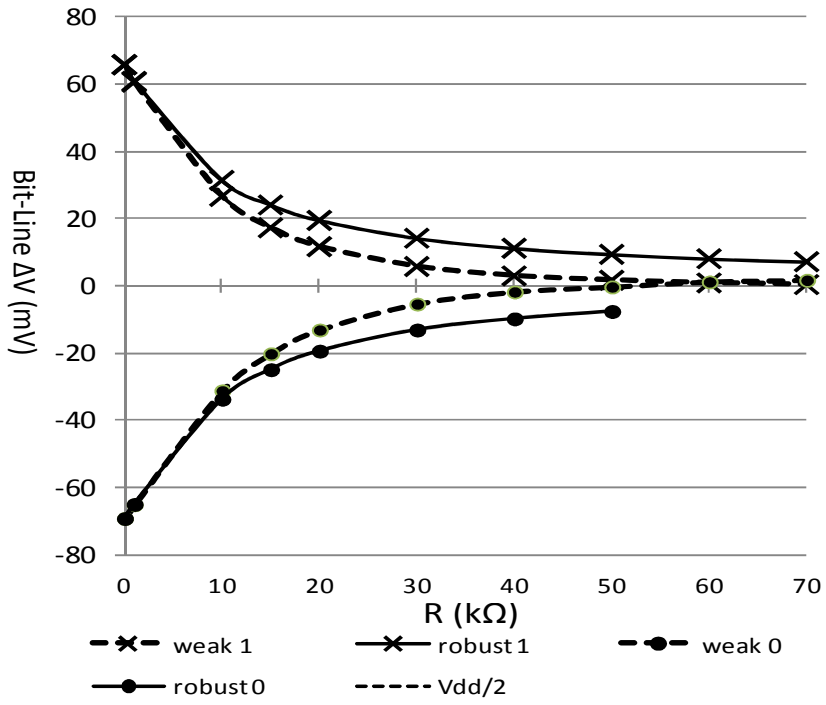


(b)

Figure 7.3: Charge accumulation for successive read-0 (a) and read-1 (b) operations for three values of the resistive open defect



(a)



(b)

Figure 7.4: Cell capacitor voltage (a) and Bit-Lines voltage difference (b) with and without the influence of charge accumulation for various resistive open values

Large numbers of read operations between write operations are very common in DRAMs. Each time a Word-Line is accessed for a read or write operation, all the cells attached to this Word-Line must perform a “dummy read” operation in order to maintain their data. This read operation is dummy because no data are obtained. However, from the cell’s point of view, a dummy read operation is identical to a normal read operation and it is equally influenced by the charge accumulation phenomenon. Therefore, charge accumulation is rather common during the operation of a defective DRAM. Depending on the address sequence of the operations, the combination of charge accumulation and Bit-Line imbalance, presented in section 7.6, can either mask or reveal the fault.

According to the simulations, under the presence of a resistive open, a write (or read) operation fails to set the appropriate voltage level to the cell capacitor. Consequently, when we attempt to perform a write operation to a faulty cell, we do not know the initial voltage of the capacitor before the write operation, even if we know the logic value stored at that cell. As a result, a write-1/0 operation on a faulty cell storing a weak 0/1 has a higher success probability with respect to the case where the cell stores a robust 0/1. Equivalently, a write-1/0 operation on a faulty cell, which initially stores a robust 0/1 has a higher probability to fail with respect to the case where the cell stores a weak 0/1. To enhance the fault coverage of a testing algorithm, write operations must be performed over robust values that are stored at the pertinent cell. The charge accumulation phenomenon can be exploited by performing read operations (dummy or not) in order to set a robust value at a cell before writing new data to it.

7.6 Impact Of Bit-Line Imbalance On Resistive Open Detection

In the previous section we mentioned that the cell’s ability to create a voltage difference between its Bit-Line and the reference Bit-Line diminishes during a read operation when a resistive open is present. Consequently, the outcome of a read operation will be very sensitive to various influence mechanisms, such as the Bit-Line imbalance. Note that Bit-Line imbalance is present even in fault-free memories. Process variations, device mismatches, transistor aging and clock disturbances, which impact the precharge time, are the main reasons for Bit-Line imbalance. However,

according to the simulation results in Figure 7.4b, the charge sharing between the capacitor and the Bit-Line can cause a shift of more than 60mV in the fault free case. Therefore, a Bit-Line imbalance up to a few tens of mV will eventually not cause a faulty behavior in the absence of a resistive open. However, under the presence of a resistive open, a bit-line imbalance of even a few mV can influence the fault activation and therefore the ability to detect it.

Most test algorithms, especially the March-based ones, perform successive write and read operations on the memory cells using either an increasing or a decreasing address order. By doing this, almost every read operation is predisposed to sense the correct value due to the tendency induced by the previous operation. Consider for example a March element like (… w0 r0 …) that is performed on a cell with a resistive open defect. In this case, after the write-0 (w0) operation, the Bit-Lines' state predispose the read-0 (r0) operation to succeed (sense a 0 value). Consequently, in case the Bit-Line imbalance is large enough, then the r0 operation will succeed even if the cell capacitor has a voltage higher than $V_{DD}/2$ after the write operation.

As mentioned earlier, completing actions are performed before a read operation in order to predispose it to sense the wrong value. To the best of our knowledge, the concept of the completing actions is not incorporated in any test algorithm so far. However, we have proven through simulations that even the application of a completing action may not provide sufficient fault coverage if the charge accumulation is not taken into account. For example, consider two successive march elements (… w0) and (r0 …). After the application of the element w0 to the defective cell, the algorithm will start applying it to the next cell, which most likely is on the same Word-Line as the defective cell. Therefore, a number of dummy read operations are applied to the defective cell. The probability that the the first dummy read operation successfully reads a 0 increases as the memory is predisposed to sense a 0 due to the w0 operation on the defective cell. Finally, the rest of the dummy read operations that follow will turn the initial weak 0 at the defective cell into a robust 0. When the r0 operation of the second element will be applied to the defective cell, the fault may not be activated as the cell's capacitor has a voltage level much closer to 0 than it had right after the w0 operation, even if the memory is predisposed to sense the faulty value (1 in our case) due to the application of a proper completing action.

The importance of the above observations is illustrated in Figure 7.5. The figure shows the success or failure areas of a read-0 (Figure 7.5a) and a read-1 operation (Figure 7.5b) performed after a write-0 and a write-1 operation respectively, for various values of the resistive defect and Bit-Line imbalance. In both cases the write operation is performed over a robust complementary value. The graph shows the resistive open detection areas (failure areas of read-0 and read-1 operations) in three cases:

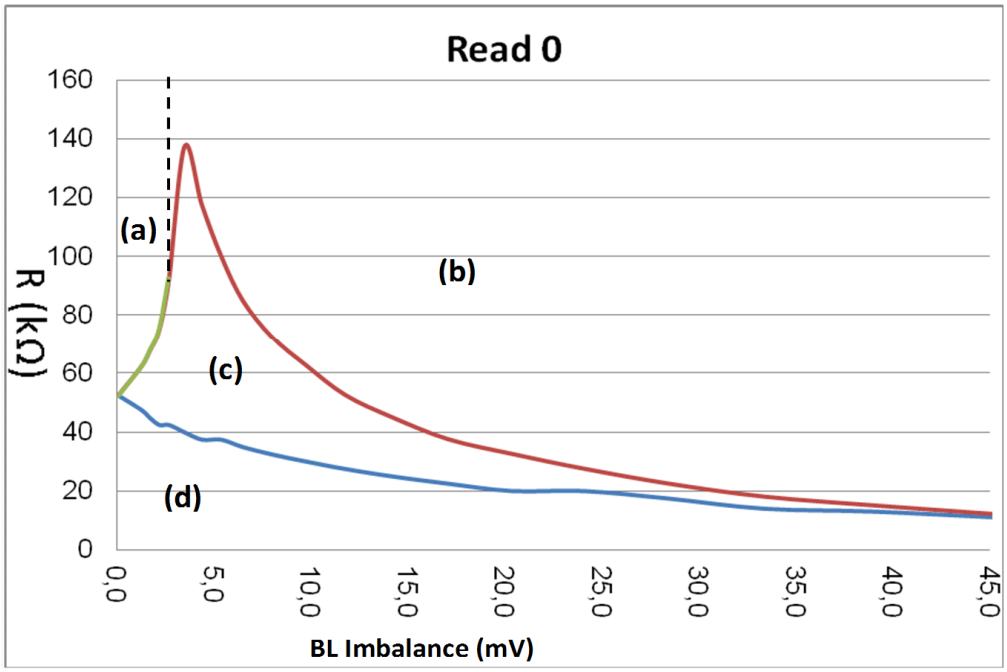
i) without a completing action (area (a) in Figure 7.5),

ii) with a completing action just before the read operation; given that earlier the data stored at the cell is enhanced through the charge accumulation mechanism (areas (a) and (b) in Figure 7.5), and

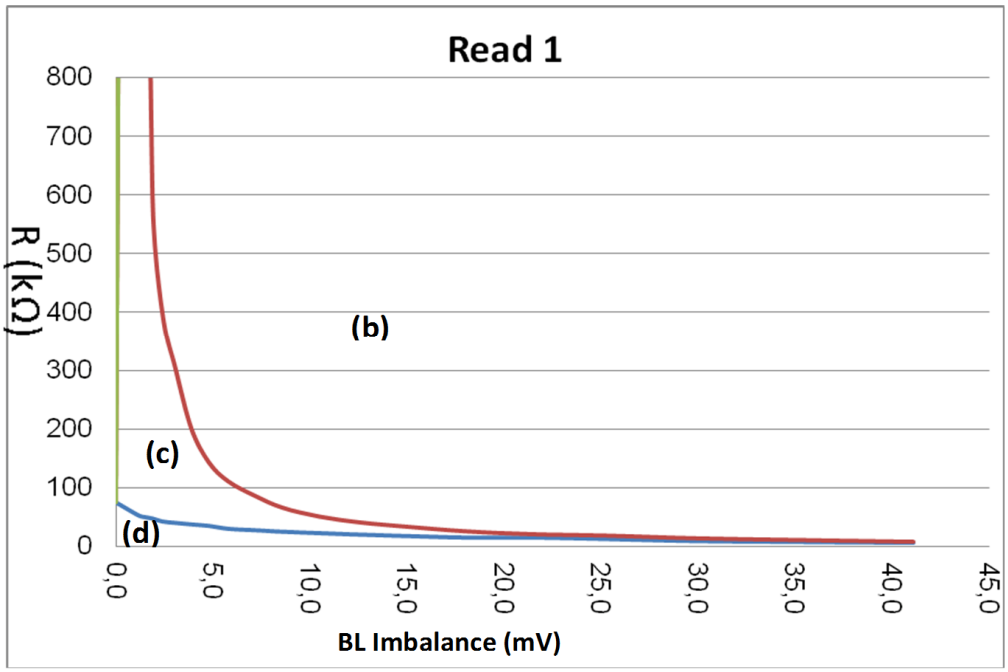
iii) with a completing action but the read operation is performed immediately after the write operation in order to avoid the charge accumulation caused by the dummy read operations, (areas (a), (b) and (c) in Figure 7.5).

The rest of the area in the graphs (area (d)) corresponds to successful read operations.

The majority of the well-known test algorithms in the literature can only detect a faulty behavior in area (a) of Figure 7.5; they use either an increasing or a decreasing address order, without completing actions and without considering the charge accumulation phenomenon. Also note that the graphs of Figure 7.5 come out as a result of a write operation over a robust complementary value (i.e. w0 over a robust 1 and w1 over a robust 0). In case the test algorithm does not follow this guideline, it is expected that it will provide a much smaller fault coverage. For example, consider a typical March element (w0 r0 w1 r1 …). After the w0 only a single read 0 operation (r0) is performed, which is not adequate to achieve a robust 0. Therefore, the w1 operation that follows is not performed over a robust 0 and has a higher chance to succeed.



(a)



(b)

Figure 7.5: The effect on the resistive open detection of the completing action and the charge accumulation for the read-0 and read-1operations, considering the presence of Bit-Line imbalance

7.7 The Proposed Test Algorithm

In order to achieve adequate fault coverage in the areas (a), (b) and (c) of Figure 7.5, a DRAM test algorithm must satisfy the following requirements:

i) a write operation of a logic value to a cell (0 or 1) must be performed while the cell stores a robust complementary value and

ii) after the write operation, initially a completing action must take place and thereafter the cell must be immediately read. Alternatively, if a number of dummy read operations are performed between the write and the read operation, they should all have a negative predisposition (i.e., a predisposition to sense the wrong value).

Next, we will construct the required test algorithm. Initially, we divide the memory cells into two groups, A and B, using a checkerboard pattern as depicted in Figure 7.6. Furthermore, we consider that neighbouring Word-Lines (rows) are grouped in pairs. In each pair of neighbouring Word-Lines, the cells of the A and B groups are numbered from 1 to k, where k is the number of Bit-Lines of the memory array.

The algorithm uses two different address orders to access the cells of group A or

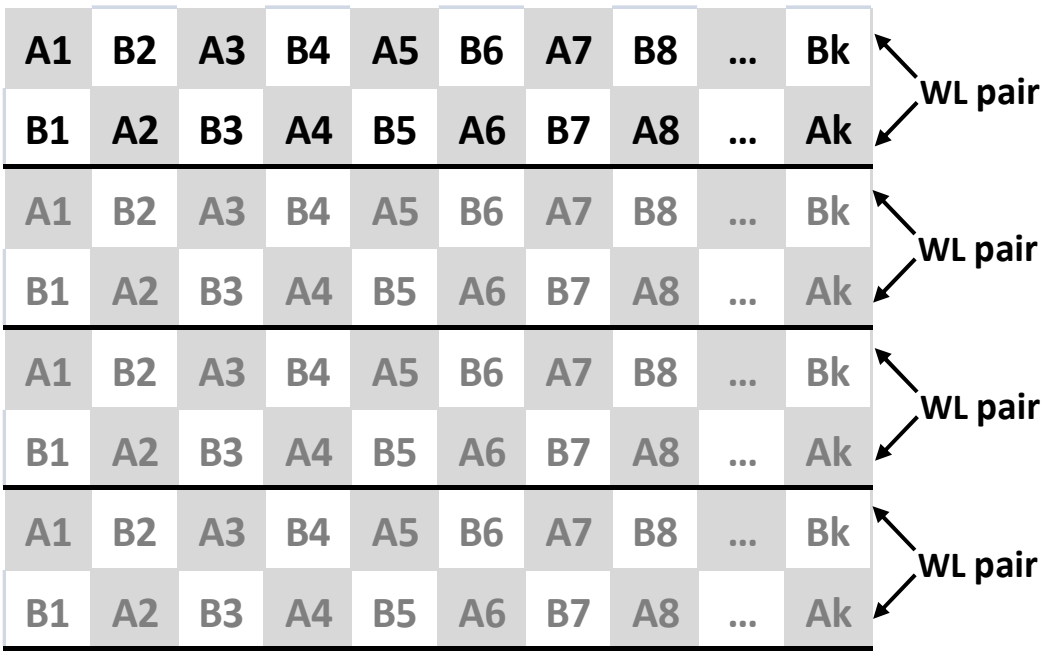


Figure 7.6: Group formation and cells' number assignment in a memory array

B in a pair of Word-Lines, which are:

i) an increasing address order in which all cells of the first Word-Line are accessed before we move to the second one, i.e. (A1, B2, A3 B4, … Bk) denoted as address order (i), or

ii) an increasing address order in which the operations follow a zigzag course between the cells of a Word-Line pair, i.e. (A1, B1, B2, A2, A3, B3, B4, A4 … Bk, Ak,) denoted as address order (ii).

Address order (i) initializes the memory to a known state; during these write operations, the dummy read operations have a positive predisposition (a predisposition to sense the correct value), optimizing in this way the conditions for applying the pattern correctly. During address order (ii), the dummy read operations have a negative predisposition (a predisposition to sense the wrong value), setting the proper conditions to activate the fault. The algorithm is presented in Figure 7.7

The algorithm writes the checkerboard pattern C_p and its complementary $\overline{C_p}$. The five steps of the algorithm can be summarized as follows:

where U is the (unknown) initial data of the memory and R symbolizes reading the whole memory array.

$$U \xrightarrow{\text{step 1}} C_p \xrightarrow{\text{step 2}} \overline{C_p} \xrightarrow{\text{step 3}} R \xrightarrow{\text{step 4}} C_p \xrightarrow{\text{step 5}} R$$

Step 1initializes the memory to a known state, the C_p. Note that in Step 1 the address order (i) is used to access the cells, while for the remaining steps we only use the address order (ii). Step 2 writes the $\overline{C_p}$, while Step 3 reads the cells, covering the write-0/read-0 case for group A cells and write-1/read-1 for group B cells. Step 4 writes the C_p pattern and Step 5 reads the cells, covering the write 1/read 1 for group A cells and write-0/read-0 for group B cells. After Step1 it is not necessary to read the memory, since the pertinent pattern, C_p, is read at step 5.

The test application cost of the algorithm is N operations for each step, a total of 5N operations, where N is the number of memory cells. This cost is very close to the well-known and widely used Checkerboard test algorithm, which has a cost of 4N operations but without the ability to provide sufficient levels of resistive open defect coverage. On the other hand the proposed algorithm is capable to cover the faults covered by the Checkerboard algorithm, since the read operations of steps 3 and 5 are

performed under the presence of both checkerboard patterns in the memory array. Consequently, the proposed test algorithm can be exploited instead of the Checkerboard one.

```
1. Write 1 to group A cells and 0 to group B
       cells using increasing address order.
2. For each pair of neighbouring word-lines
       { for i =1 to k step=2
              { write 0 to A(i) cell
              write 1 to B(i) cell
              write 1 to B(i+1) cell
              write 0 to A(i+1) cell } }
3. For each pair of neighbouring word-lines
       { for i =1 to k step=2
              { read 0 from A(i) cell
              read 1 from B(i) cell
              read 1 from B(i+1) cell
              read 0 from A(i+1) cell } }
4. For each pair of neighbouring word-lines
       { for i =1 to k step=2
              { write 1 to A(i) cell
              write 0 to B(i) cell
              write 0 to B(i+1) cell
              write 1 to A(i+1) cell } }
5. For each pair of neighbouring word-lines
       { for i =1 to k step=2
              { read 1 from A(i) cell
              read 0 from B(i) cell
              read 0 from B(i+1) cell
              read 1 from A(i+1) cell } }
```

Figure 7.7: The proposed test algorithm for resistive open defect detection

7.8 Conclusions

Resistive open defects are of great importance in DRAMs. In this paper, we introduce the charge accumulation phenomenon, which affects the operation of a DRAM when resistive opens are present. The charge accumulation is extensively studied through simulations. The simulation results show that the understanding of the charge accumulation phenomenon is crucial to effectively detect resistive opens in DRAMs. Existing and widely used test algorithms (like March algorithms) fail to provide high defect coverage of resistive opens, since they do not consider charge accumulation. Based on the outcome of our experiments, a new, fast test algorithm is

proposed that enhances the coverage of resistive open defects under the presence of Bit-Line imbalance by taking into account the charge accumulation effect.

CHAPTER 8. A BIST CIRCUIT FOR NLTF TESTING

8.1 Abstract

In this chapter a brief introduction to memory Built-In Self-Test (BIST) circuits is presented and then, as a feasibility case study, we illustrate the design of a DRAM BIST circuit that implements the NLTF test algorithm proposed in Chapter 6.

8.2 The Memory Built-in Self Test Concept

A popular method to accelerate and simplify memory testing procedures is to embedded them in the IC, with the use of a proper circuit (either a *Built-In Self Test – BIST* circuit or an existing processing unit / microcontroller in the chip). The advantages of using BIST circuits are very important and they are synopsized as follows: a) the acceleration of the read-write operations, since we no longer need to transfer test data and addresses between the IC and outside testing units, since the whole testing procedure is executed inside the IC, b) the ability to perform many operations in parallel, since we are no longer restrained by the interface of the memory and c) the testing is performed without the need of an expensive external

tester, thus the testing cost is reduced or the testing throughput is increased with the use of more, simple and low cost testers.

The built in self test procedure can be implemented in two basic ways:

i) In standalone DRAM circuits, usually called *Commodity DRAMs*, a BIST circuit can be incorporated inside the IC of the memory [16], [17], [26], [43]. This circuit can perform the testing using one or more testing algorithms.

ii) In *embedded memories* [23], [24], [37], [46], like eDRAMs, where the memory is either in the same IC or in the same package with a processor, except from the use of a BIST circuit it is feasible to execute the testing procedure for the memory on this processor.

Another significant advantage of built-in self test is that the testing procedure can be performed outside the factory environment, in the field of operation, where the memory is incorporated in a computing system. This is very important because a faulty behavior may occur also in the field of operation. Since a system consists of many ICs (and many memory ICs as well), in case of a system failure it is very difficult to detect the defective IC that caused this failure, unless these ICs support built in self test options.

Main drawbacks of a BIST solution are the silicon area requirements and the need of extra I/O pins, which increase the cost of the IC.

8.3 The Proposed BIST Circuit For NLTF DRAM Testing

In this chapter we present the implementation of a BIST circuit that is implementing the NLTF test algorithm presented in Chapter 6. The bit-oriented case is considered. Also a simplified DRAM interface is assumed; however, this can be easily adapted to any existing DRAM interface.

8.3.1 Overview

The proposed BIST circuit consists of two basic building blocks (see Figure 8.1): the Controller and the Address Generator, each of one of them consisting of several sub-blocks as we will see next. Both devices have two common input signals: the clock (*clk*) and the reset (*Reset*) signals (the latter is active at low).

The Controller controls the test flow, generates test data and sends the appropriate signals that control the Address Generator and the memory depending on the current step of the algorithm each time. The test procedure starts when the *TEST_ENABLE* signal of the Controller is set to high. The signals of the Controller that directly feed the memory are: *READ_ENABLE*, *WRITE_ENABLE* and *BIT*. The latter corresponds to the bit value to be written in the memory at a write operation; it also corresponds to the expected value during a read operation. Thus, the *BIT* signal along with the *BIT_OUT* signal of the memory, which corresponds to the output of a read operation, feed a XOR gate for comparison. If the *BIT* and *BIT_OUT* signals are not the same, the XOR gate's output (which is called *ERROR*) is set to 1, indicating that the value read from the memory was not the expected one. At the same time the current Row and Column addresses (signals ROW_ADDRESS[0-K-1] and COLUMN_ADDRESS[0-L-1]) are available to indicate the address in which the error was detected (fault location option).

The Address Generator is controlled by the signals of the Controller and generates the row address (port *R_A[0..K]*) and the column address (port *C_A[0..L]*) of the cell to be accessed each time. As long as the count enable signal (*addr_count_en*) is high, the Address Generator provides an address of the required address sequence on every positive clock edge.

Next, the functionality of the signals controlling the Address Generator are presented. The signals *mode*, *cell_gr* and *cell_no* determine which memory cells will be accessed at the current step of the executed NLTF algorithm. Let us be reminded that in the NLTF test algorithm the memory cells are divided into two groups, A and B and in each group cells are numbered from 0 to 3 (see Figure 6.2). In each step of the algorithm either all the cells of a particular group are accessed (A or B) or the cells of a group having a particular number assignment (A_i or B_i , i=0 – 3) are accessed. When the signal *mode* is set to 0, the addresses generated correspond to either the group A or group B cells, depending on the value of the signal *cell_gr* (the name stands for "cell group"); if *cell_gr*=0 then group A cells will be accessed, otherwise group B cells will be accessed.

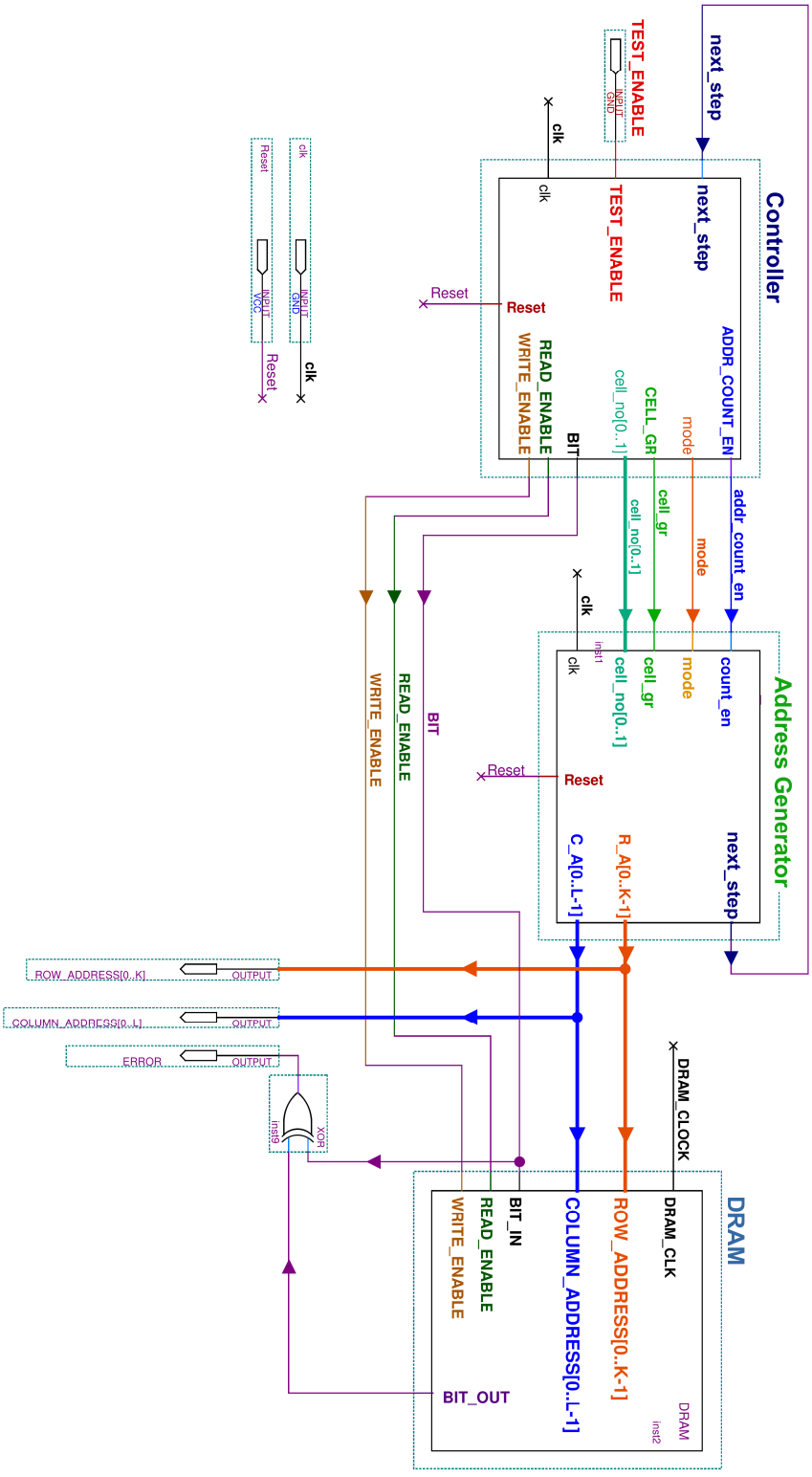


Figure 8.1: The proposed BIST for NLTF testing

When the cells to be accessed are the ones with a specific number assignment, either A_i or B_i , (i=0 –3), then *mode* is set to 1. In that case the accessed cells are those determined by the pertinent values of *cell_gr* and *cell_no*. The *cell_no* signal is a two bit signal and, thus, the values it takes correspond to the decimal numbers 0, 1, 2 and 3 which are the cell numbers that appear in Figure 6.2. The above description is synopsized in Table 8.1, in which the X values are "don't care" values.

The Address Generator on the other hand provides to the Controller a signal called *next_step* which turns to high when the current address is the last address of the current step and, therefore, the Controller must proceed to the next step.

Next, the Address Generator and the Controller will be presented in details.

TABLE 8.1: Address Generator control inputs

mode	cell_gr	cell_no (i=0-3)	cells accessed
0	0	X	group A
0	1	X	group B
1	0	i	Ai
1	1	i	Bi

8.3.2 The Address Generator.

The Address Generator is a simple circuit since it is a modification of the classic counter that is based on JK flip-flops. This counter provides addresses that follow the increasing address order sequence. The generated address consists of L+K bits, where L and K are the lengths (in number of bits) of the column address and row address respectively, while the L bits are the least significant bits of the counter. The state table of a JK flip flop is shown in Table 8.2 (where A´ is the complementary of A) [59]. In Figure 8.2, we present as an example a simple 4-bit counter, where the output bits C0 and C1 correspond to the column address and the bits R0, R1 correspond to the row address. Each flip flop provides one bit of the address (L=K=2).

TABLE 8.2: The JK flip flop state table

Q(t)	J	K	Q(t+1)
A	0	0	A
A	0	1	0
A	1	0	1
A	1	1	A′

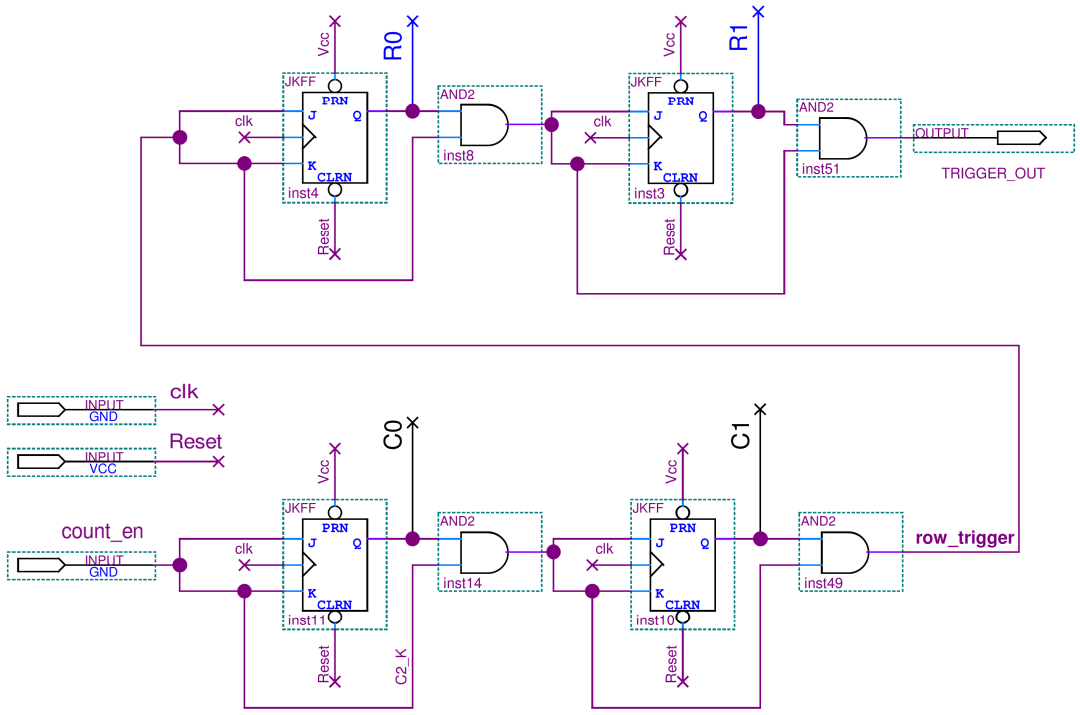


Figure 8.2 A simple JK flip-flop counter

In order to access the cells according to the test algorithm under consideration, we must modify the way that the last two bits (i.e. the least significant bits) of the column address and the row address are generated. Thus, we must initially identify how the last two digits of the row and column address are related to the cell group and the cell number assignments. In Figure 8.3, the relation of the cell group and number

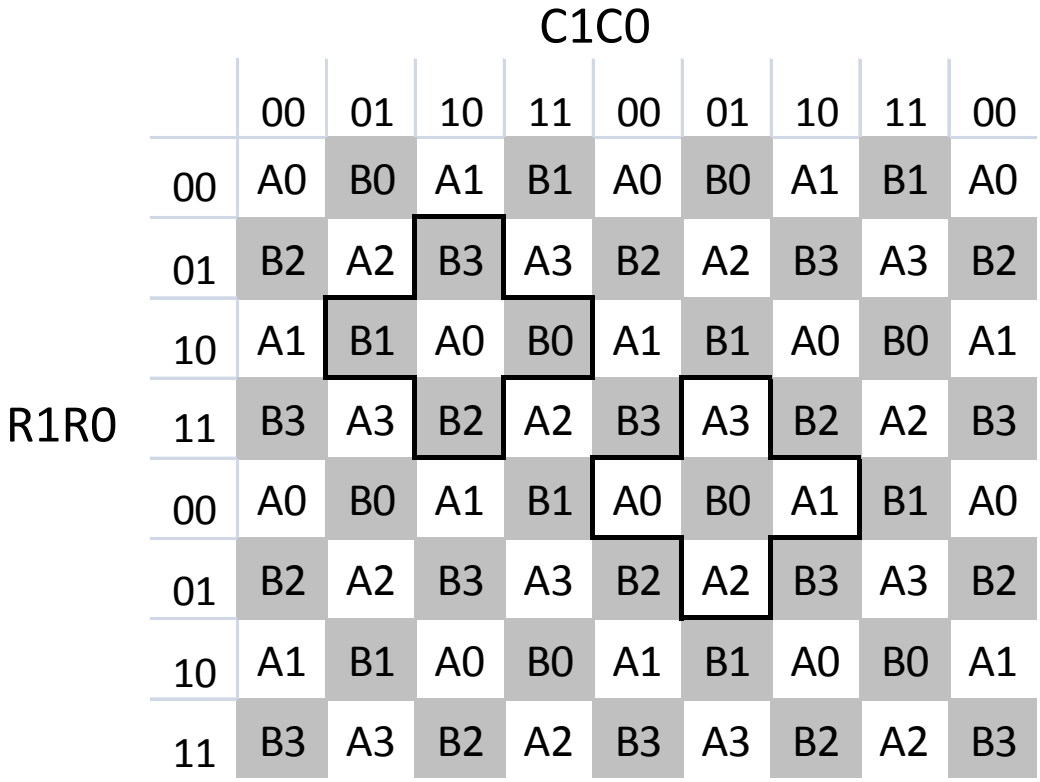


Figure 8.3: Last two digits of row and column address for each memory cell

TABLE 8.3: Boolean expressions for the last two bits of row and column address of each cell group and cell number assignment

mode=0	mode=1			
group A	A0	A1	A2	A3
cell_gr=0	cell_no=00	cell_no=01	cell_no=10	cell_no=11
C0′·R0′ + C0·R0	C0′·R0′·(C1′·R1′ + C1·R1)	C0′·R0′·(C1·R1′ + C1′·R1)	C0·R0·(C1′·R1′ + C1·R1)	C0·R0·(C1·R1′ + C1′·R1)
group B	B0	B1	B2	B3
cell_gr=1	cell_no=00	cell_no=01	cell_no=10	cell_no=11
C0·R0′ + C0′·R0	C0·R0′·(C1′·R1′ + C1·R1)	C0·R0′·(C1·R1′ + C1′·R1)	C0′·R0·(C1′·R1′ + C1·R1)	C0′·R0·(C1·R1′ + C1′·R1)

assignment with the last two digits of the pertinent row and column address are illustrated. The two least significant digits of the column address are denoted as C0

and C1 while R0 and R1 are the two least significant digits of the row address.

We observe that group A cells have addresses where (C0=0 and R0=0) or (C0=1 and R0=1). Using the boolean formalism, this can be denoted as (C0´·R0´) + (C0·R0), where C0´ is the complementary of C0. Similarly, for group B cells the corresponding expression is (C0·R0´)+(C0´·R0). The expressions for all possible cases are presented in Table 8.3. On the same table we can also see the corresponding values of the input signals *mode, cell_gr* and *cell_no*.

From the Figure 8.3 and the Table 8.3 the following observations arise:

a) When *mode*=0, where all cells of group A or group B are accessed, the C0 does not change as we move from cell to cell on the same word-line. Moreover, when *mode*=1, C0 has a constant value which depends on the group and the number of the cell. Thus, in both modes the values of C0 does not follow the counter sequence and the JK flip-flop for C0 can be replaced with a combinational circuit.

b) When *mode*=0, the values of C1 and R1 follow the counter sequence. However, when *mode*=1, C1 does not change as we are moving on the same word-line while R0 has a constant value that depends on the group and the cell number. Therefore, when *mode*=1, R1 and C1 do not follow the counter sequence.

Aiming to simplify the design of the address generator we exploit the JK flip-flop in order to construct the *Count – Set flip flop* (*CS-flip flop*). The CS-flip flop has two operation modes set and count, which are selected by the signal *mode*. When *mode* =0 the output Q toggles on every positive edge of the clock, provided that the signal *count_en* is high, regardless on the value of the input signal *set*. When *mode*=1, the input *set* passes to the output on every positive edge of the clock (like a D-flip flop).

Based on the CS flip-flop and the previous observations we constructed the Address Generator of Figure 8.5.

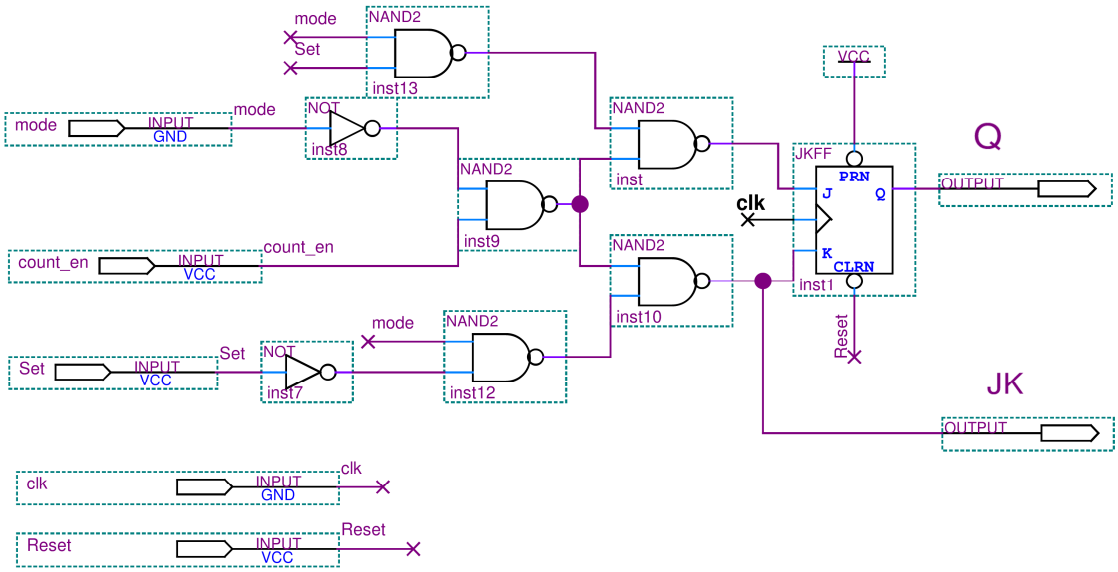


Figure 8.4: The Count-Set (CS) flip flop

As we can see in Figure 8.5 the CS flip flop is used for bits C1 and R0, while bit C0 is generated by a single XOR gate.

Note that as soon as the input signals *mode*, *cell_gr* and *cell_no* are set to the desired value, the first address of the pertinent address sequence is available on the next clock edge, regardless of the *addr_count_en* signal's state. The *addr_count_en* signal is only responsible to control whether the Address Generator will move to the next address on the next positive edge of the clock.

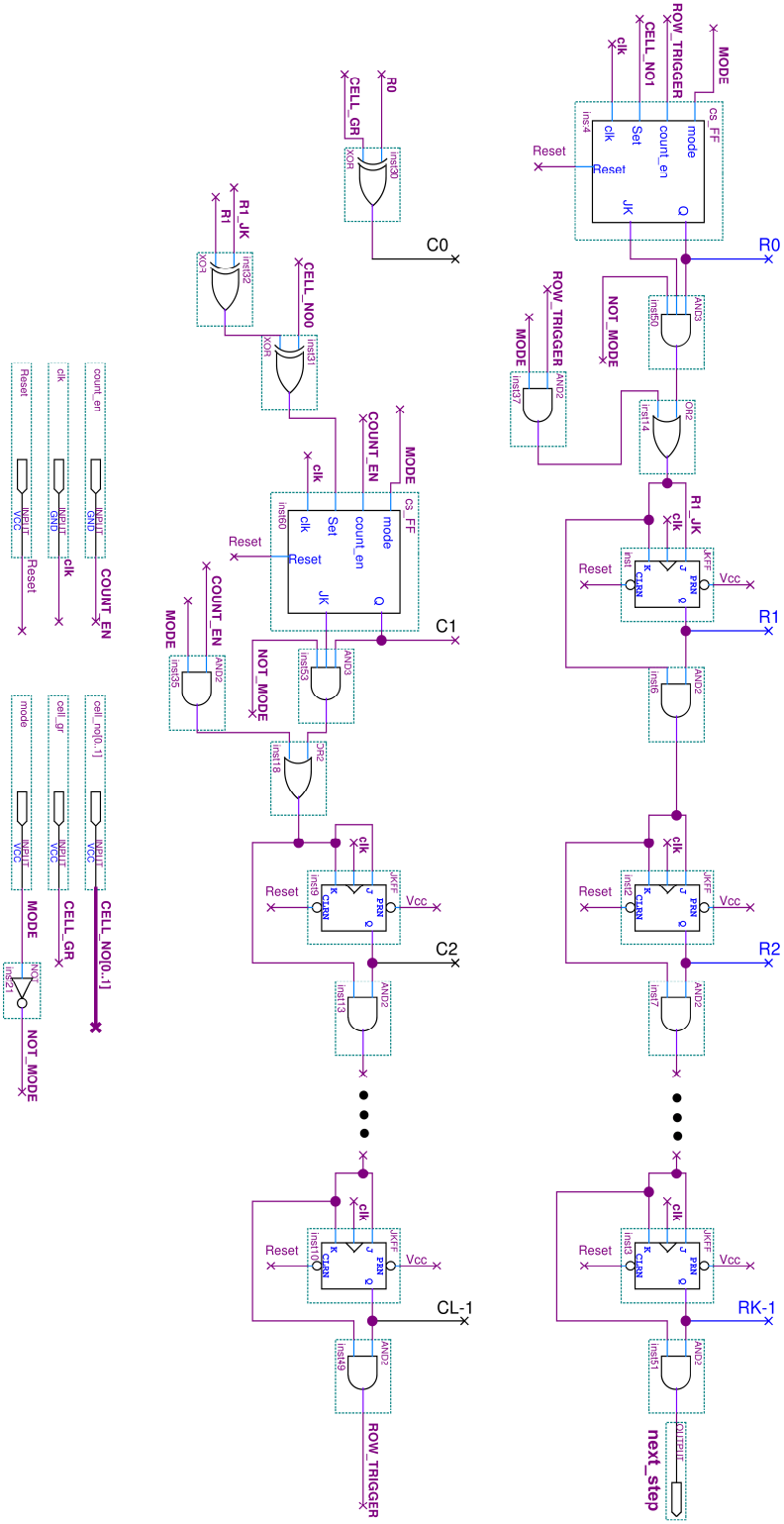


Figure 8.5: The Address Generator

8.3.3 The Controller

As previously stated, the Controller controls the test flow and provides the appropriate signals to control the Address Generator and the memory according to the current step of the executed algorithm each time. It consists of two basic sub-blocks: the Program Counter (PC) and the Signal Generator, as it is illustrated in Figure 8.6.

The Program Counter (PC) is a state machine with 28 states, which correspond to the number of steps of the test algorithm in both phases. The states of these five flip flops are presented as a 5-bit signal *C* which has a unique value depending on the state of the PC. The signal *C* is used by the Signal Generator to produce the required control signals, as will be explained later. The PC also generates the control signals *cell_no* and *addr_count_en*.

The Signal Generator is a combinational circuit that generates the appropriate control signals *mode*, *R/W*, *cell_gr* and *BIT*. The signal *R/W* indicates the type of memory operation, which is ‘write’ if *R/W* =0 and ‘read’ if *R/W* =1. From this signal the *READ_EN* and *WRITE_EN* signals are generated.

In Table 8.4, for each step of the test algorithm we present the pertinent PC state signal *C* and the output signals of the Signal Generator.

Before we get into more details about the construction of the PC and the Signal Generator, we will make some observations regarding the functionality of the Controller. As explained in paragraph 8.3.1, the Controller receives the signal *next_step* from the Address Generator which indicates that the current address sequence is completed and we must proceed to the next step. Thus, the *next_step* signal feeds the Program Counter in order to induce a state change. In addition,

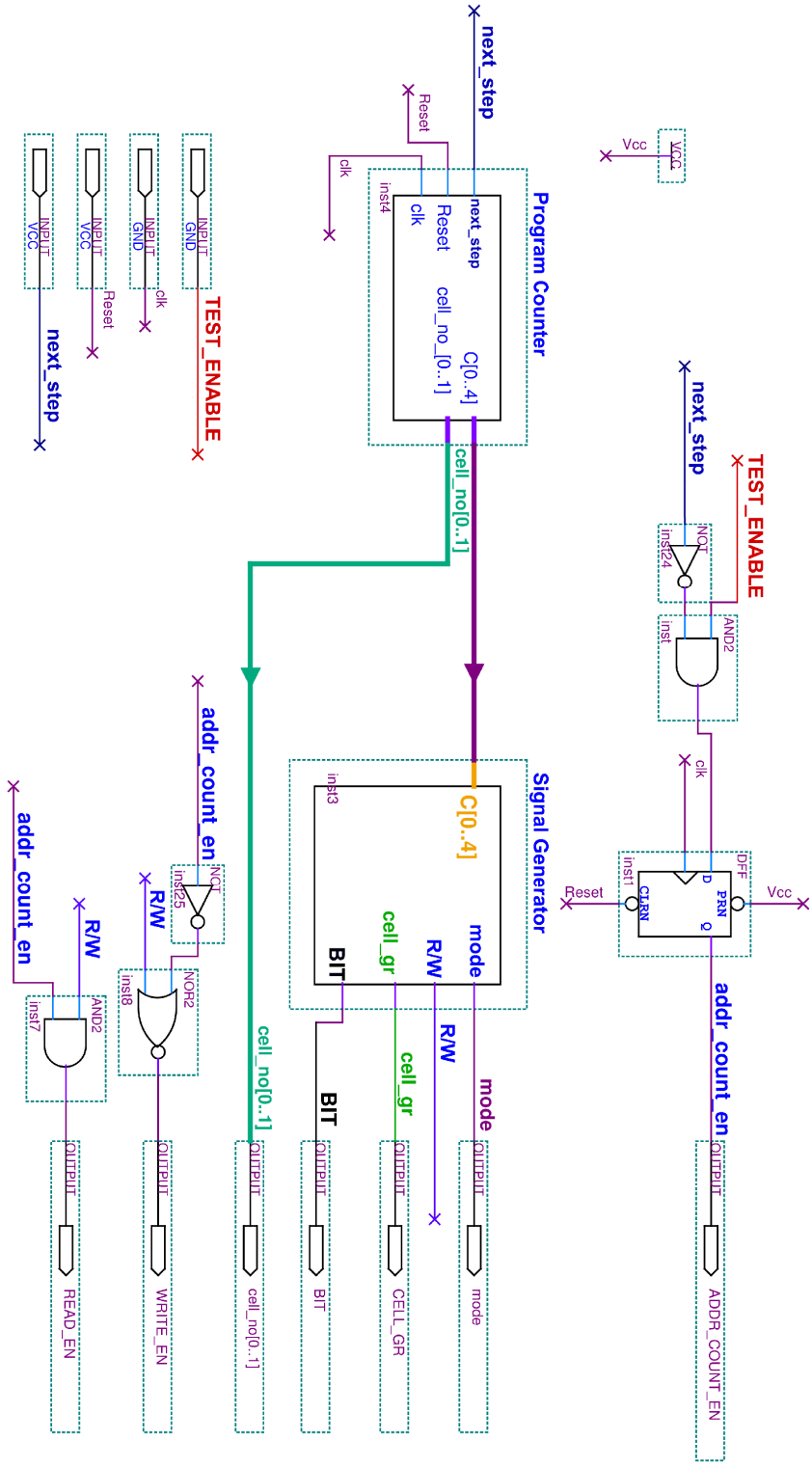


Figure 8.6: The Controller

TABLE 8.4: The Controller's states and control signals at each step of the test algorithm

PHASE 1										
		PC state					Control Signals			
s/n	Algorithm Step	4	3	2	1	0	mode	R/W	cell_gr	BIT
1	*write 0 to group A cells*	0	0	0	0	0	0	0	0	0
2	*write 1 to group B cells*	0	0	0	0	1	0	0	1	1
3	*read group A cells*	0	0	0	1	0	0	1	0	0
4	*read group B cells*	0	0	0	1	1	0	1	1	1
5	*write 0 to cells Bi*	0	0	1	0	0	1	0	1	0
6	*read group A cells*	0	0	1	0	1	0	1	0	0
7	*write 1 to cells Bi*	0	0	1	1	0	1	0	1	1
8	*read group A cells*	0	0	1	1	1	0	1	0	0
9	*read cells Bi*	0	1	0	0	0	1	1	1	1
10	*write 1 to cells Ai*	0	1	0	1	0	1	0	0	1
11	*read group B cells*	0	1	0	1	1	0	1	1	1
12	*write 0 to cells Ai*	0	1	1	0	0	1	0	0	0
13	*read group B cells*	0	1	1	0	1	0	1	1	1
14	*read cells Ai*	0	1	1	1	0	1	1	0	0

Repeat for cell_no= 0 to 3

Repeat for cell_no= 0 to 3

PHASE 2										
		PC state					Control Signals			
s/n	Algorithm Step	4	3	2	1	0	mode	R/W	cell_gr	BIT
1	*write 1 to group A cells*	1	0	0	0	0	0	0	0	1
2	*write 0 to group B cells*	1	0	0	0	1	0	0	1	0
3	*read group A cells*	1	0	0	1	0	0	1	0	1
4	*read group B cells*	1	0	0	1	1	0	1	1	0
5	*write 1 to cells Bi*	1	0	1	0	0	1	0	1	1
6	*read group A cells*	1	0	1	0	1	0	1	0	1
7	*write 0 to cells Bi*	1	0	1	1	0	1	0	1	0
8	*read group A cells*	1	0	1	1	1	0	1	0	1
9	*read cells Bi*	1	1	0	0	0	1	1	1	0
10	*write 0 to cells Ai*	1	1	0	1	0	1	0	0	0
11	*read group B cells*	1	1	0	1	1	0	1	1	0
12	*write 1 to cells Ai*	1	1	1	0	0	1	0	0	1
13	*read group B cells*	1	1	1	0	1	0	1	1	0
14	*read cells Ai*	1	1	1	1	0	1	1	0	1

Repeat for cell_no= 0 to 3

Repeat for cell_no= 0 to 3

when *next_step* is set to 1 the signal *addr_count_en* is set to 0 (within the next clock cycle). This is essential because the Address Generator must receive the new signals (corresponding to the new step under execution) from the Controller before it starts generating the new address sequence. For the same reason, when *addr_count_en* is set to 0 the *READ_EN* and *WRITE_EN* signals are also set to 0, prohibiting any operation in the memory, because there is not a valid address available by the Address Generator yet.

8.3.4 The Program Counter And The Signal Generator

The Program Counter mainly consists of seven JK flip-flops as we can see in Figure 8.7; five of them are related to the state of the PC while the other two form a 2-bit counter which determines the current cell number (signal *cell_no*). The latter is necessary to be included in the PC circuit because steps 5 to 9 and 10 to 14 of the NLTF algorithm are repeated for every value of the signal *cell_no*, as it is presented in Table 8.4.

Next, some observations will be made in order to understand more easily the PC circuit. The five flip-flops that determine the 28 states of the PC behave differently than a normal 5-bit counter for the following two reasons:

Note that the PC does not behave like a typical 5-bit counter, for the following two reasons:

a) A 5bit counter has 32 possible states, which means that four of this states must be skipped in order to take the required 28 states. This is achieved by the internal signal *jump*, which is set to high when the next state (considering the states as numbers in increasing order) must be skipped. This skip operation occurs when the PC state is moving from step 9 to 10 and from 14 to 1.

b) Steps 5 to 9 and 10 to 14 are repeated for every value of the signal *cell_no*. Thus, the internal signal *REPEAT* is set to high when the PC state must return to 5 (if it is currently at state 9) or to 10 (if it is currently at state 14).

The Signal Generator, as we can see in Figure 8.8, is a simple combinational circuit. It receives the 5-bit signal C from the PC as input and generates the signals *mode, R/W, cell_gr* and *BIT*.

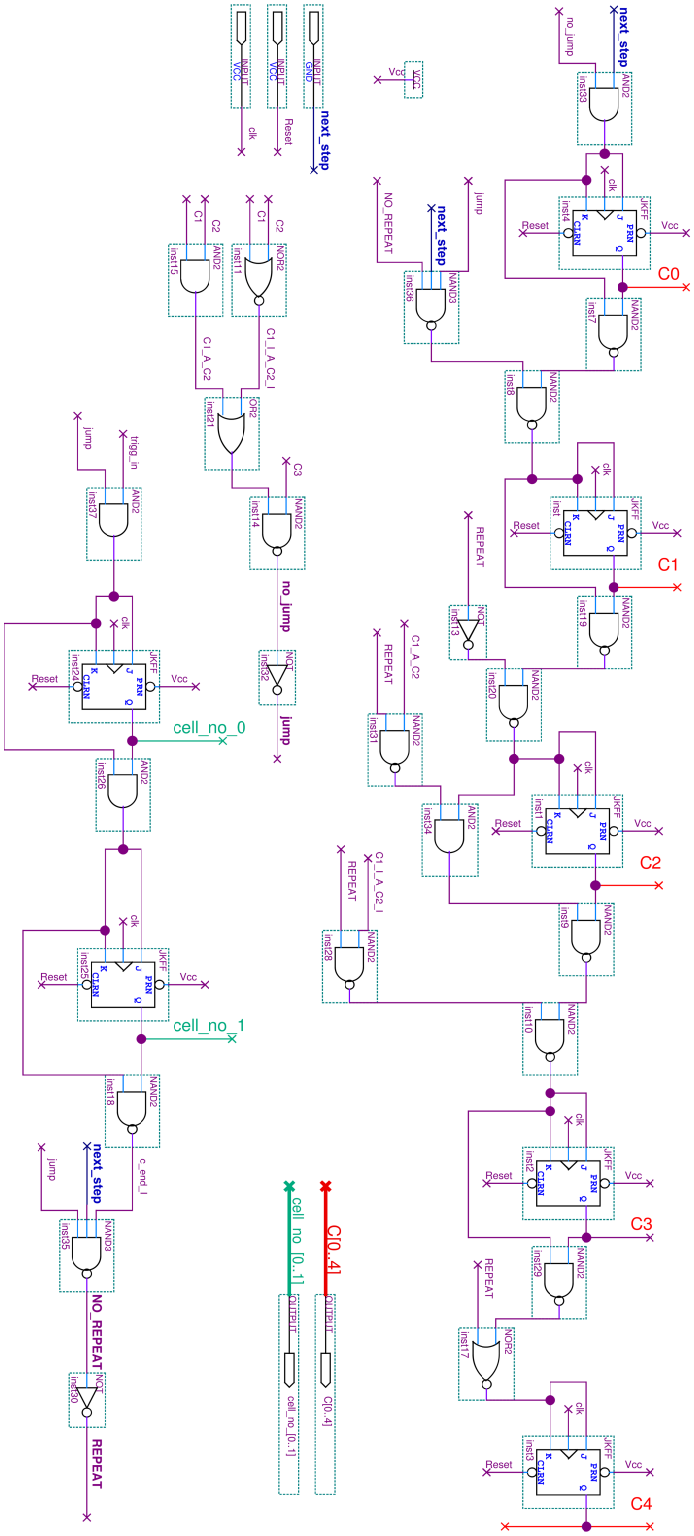


Figure 8.7: The Program Counter

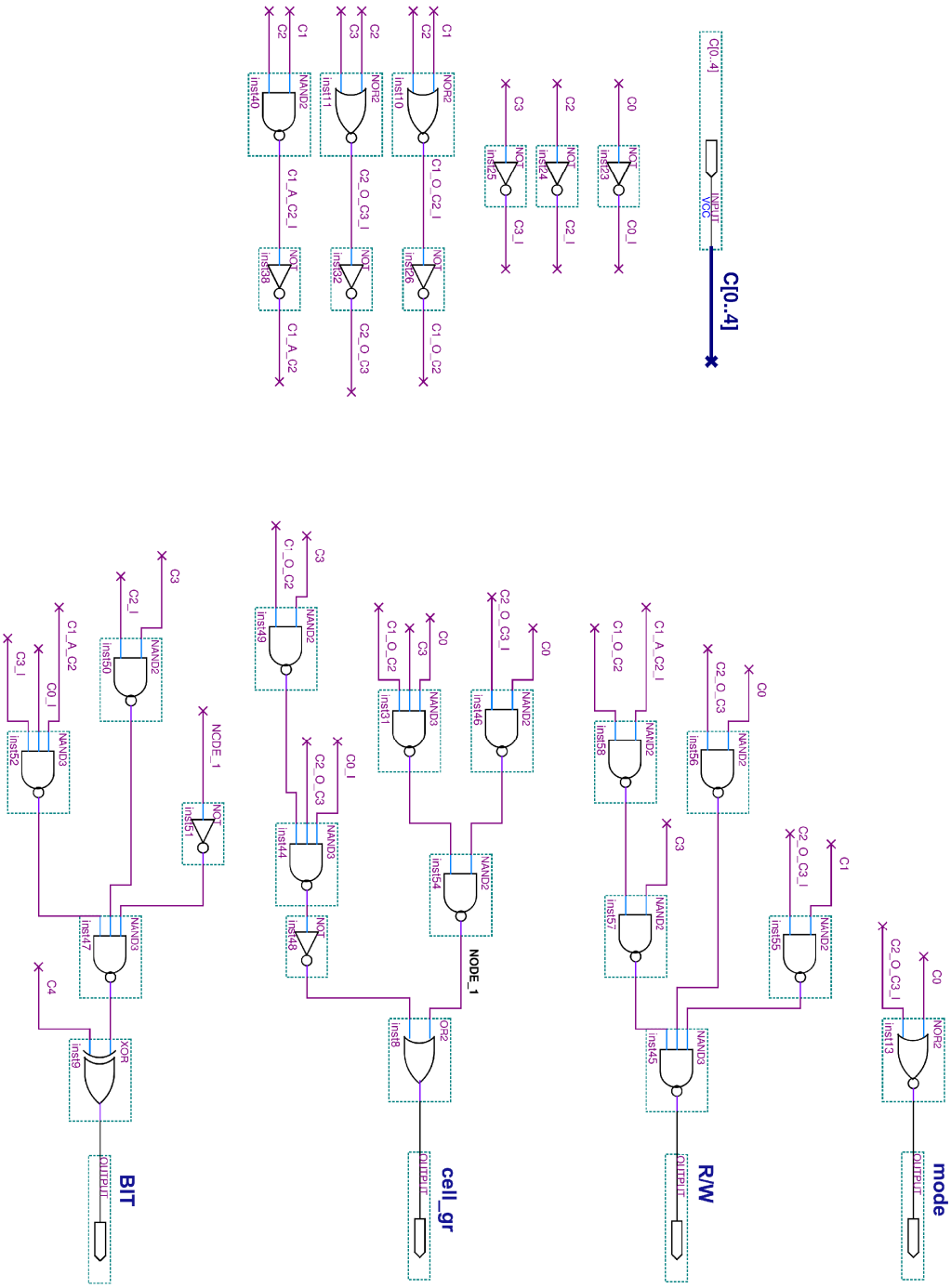


Figure 8.8: The Signal Generator

8.4 BIST Circuit Validation

Aiming to validate its functionality, the proposed BIST circuit was designed and simulated in the Altera Quartus II environment. Next, indicative simulation waveforms are presented in order to demonstrate the operation of the proposed BIST circuit. The waveforms are depicted in Figures 8.9 and 8.10. In order to provide as much information as possible, we present four pictures (two in each figure) at different zoom levels. The main signals of the BIST circuit discussed in the previous sections are presented. Note that for this validation task a simple 16x16 memory array is considered.

8.5 Conclusions

In this Chapter we presented the design of a Built-In Self Test (BIST) circuit that implements the NLTF test algorithm presented in Chapter 6. At a first sense, the algorithm appears to be rather complex, especially due to the fact that it divides the memory cells in groups and each time requests access only to the cells of one group. However, according to the discussion above, the actual implementation of such a BIST circuit is feasible and quite simple while the silicon area cost is very low.

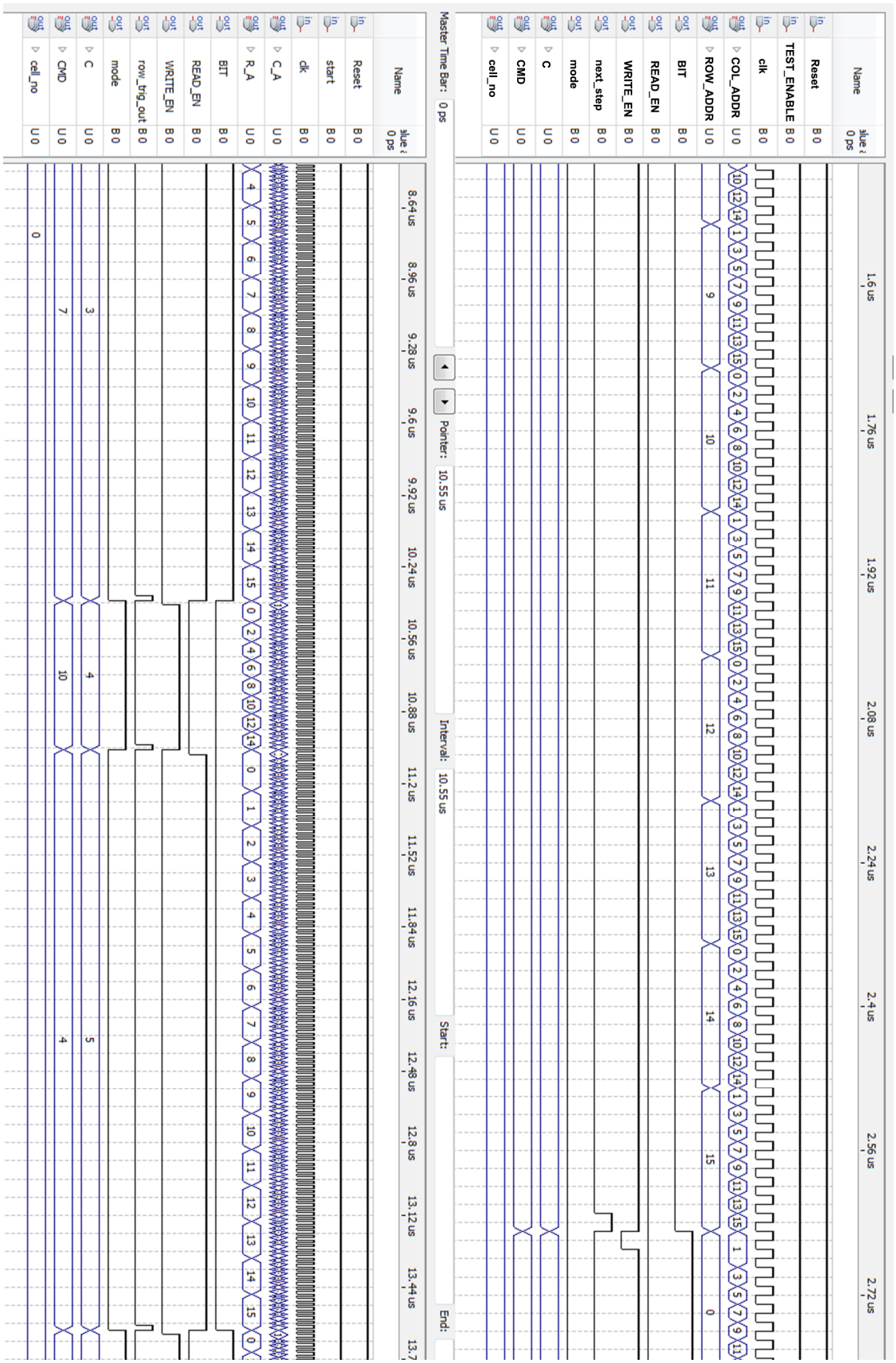


Figure 8.9: BIST Simulation Waveforms 1

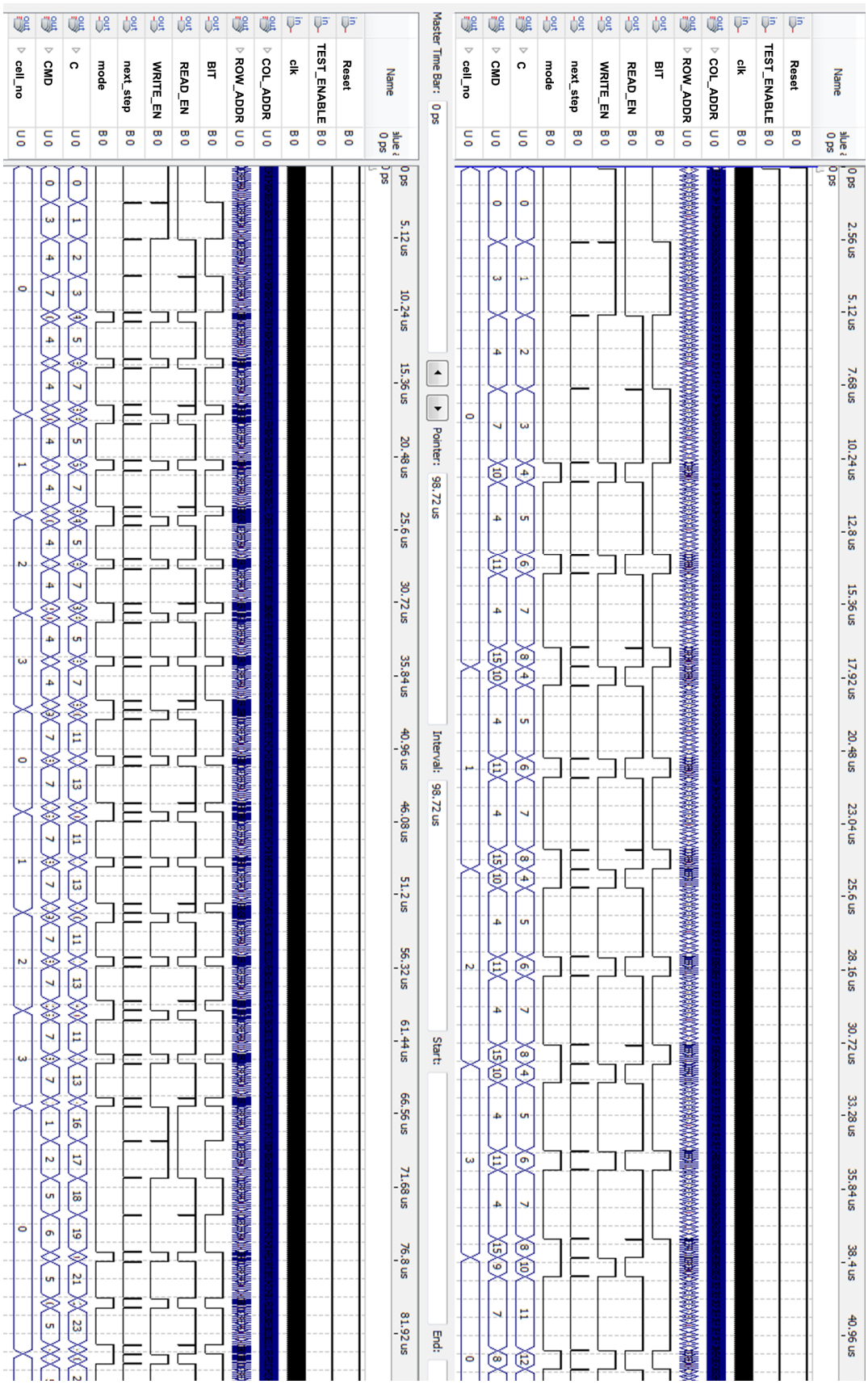


Figure 8.10: BIST Simulation Waveforms 2

CHAPTER 9. CONCLUSIONS

The rapid evolution of semiconductor nanotechnologies the last 50 years resulted in the development of very powerful digital integrated circuits (ICs) which nowadays are present in almost any electronic device used in everyday life. DRAMs are among the most important components of the digital systems. Due to the evolution of the corresponding technologies, the storage capacity and the performance of DRAMs has significantly increased the last decades.

However, like in all other IC types, modern DRAMs suffer from increased difficulty in testing. Due to the continuous increase in density and the memory cell's size shrinking, existing defect mechanisms are reinforced and combined with new ones, affecting modern DRAMs reliability. Thus, new testing solutions are essential in order to keep the testing procedure reliable and at an affordable time cost.

The contributions of this dissertation in the area of DRAM memory testing are summarized as follows:

1) The development of a new neighborhood type for characterization and/or testing of folded DRAMs with the NPSF fault model, by taking into account the physical design of the memory array. Due to the proposed neighborhood, that is called Δ–Type neighborhood, a new test algorithm is constructed that achieves a cost reduction of 57.7% in test application time with respect to the well known TLAPNPSF1T test algorithm that uses the classic Type-1 Neighborhood. Moreover, enhanced versions of the new algorithm, with practically the same cost in test application time, can cover the NPSF faults combined with faults caused by the Bit-Line influence and by the Word-Line capacitive coupling; the latter are called NWSF faults. Finally, a slightly more expensive version of the proposed test algorithm, which is still cheaper by 53.6% compared to the TLAPNPSF1T test algorithm, is able

to cover NPSFs combined with NWSFs even in the case where both fault types appear simultaneously.

2) The introduction of a new fault model, the NLTF, which targets the adjacent memory cell interactions that are a reliability threat in high density nanometer DRAMs. The new fault model is established by considering the two well-known mechanisms underneath these interactions: the neighborhood leakage currents and the cell state transitions. Regarding the traditional fault models, the detection of faults due to the combined influence of these fault generation mechanisms is either beyond their capabilities (like in the case of the Coupling Faults model), or turns out to be an excessively time consuming task (like in the case of the NPSF model and the pertinent test algorithms). On the other hand, the proposed NLTF fault model enabled us to develop a new test algorithm which covers these faults with a test application time reduction that ranges from 68% to 87% with respect to well known algorithms in the literature that are also capable to detect them.

3) The study of the resistive open defects inside the memory cells that are quite frequent in DRAM memories. Our research on this defect type, through electrical simulations, revealed the existence of an important phenomenon, the charge accumulation, which significantly affects the faulty behavior of a cell with a resistive open and, thus, plays an important role in testing procedures. Existing and widely used test algorithms (like March algorithms) fail to provide high defect coverage of resistive opens, since they do not consider charge accumulation. Based on the outcome of our experiments, a new, fast test algorithm is proposed that enhances the coverage of resistive open defects, under the presence of Bit-Line imbalance phenomena, by taking into account the charge accumulation effect.

4) The development of a Built-In Self-Test (BIST) circuit that implements the NLTF test algorithm. BIST circuits are a very attractive solution in memory testing. The circuit has been designed and simulated in order to validate its functionality and prove the ability to embed a quite complex test algorithm in a BIST solution at a low silicon area cost.

Our future plans are mainly focused towards two directions: i) expand our research on resistive opens in order to take into account the Bit-Line capacitive coupling effect as well, which significantly affects the faulty behavior of the memory cells, and ii) examine the ability to apply the new fault models, test algorithms and

methodologies on new memory types, like Magnetoresistive RAMs (MRAMs) or Resistive RAMs (RRAMs) that have gained attention as the next step in memory technologies.

REFERENCES

[1] M. L. Bushnell, V. D. Agrawal, “Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits,” Kluwer Academic Publishers, 2000.

[2] S. Mourad, Y. Zorian, “Principles of Testing Electronic Systems,” John Wiley & Sons Inc., 2000.

[3] L-T Wang, C-W Wu, X. Wen, “VLSI Test Principles and Architectures: Design for Testability,” Morgan Kaufmann Publishers, 2006.

[4] L-T Wang, C. E. Stroud, N. A. Touba, “System-On-Chip Test Architectures: Nanometer Design For Testability,” Morgan Kaufmann Publishers, 2008.

[5] P. Mazumder and K. Chakraborty, “Testing and Testable Design of High-Density Random-Access Memories,” Kluwer Academic Publishers, 1996.

[6] A. J. van de Goor, “Testing Semiconductor Memories-Theory and Practice,” John Wiley & Sons Ltd., 1991.

[7] Weste, D. Harris, “CMOS VLSI Design: A Circuits and Systems Perspective, ” Addison-Wesley, 2011.

[8] J. M. Rabaey, A. Chandrakasan, B. Nikolic, “Digital Integrated Circuits: A Design Perspective,” Prentice Hall, 2003.

[9] B. Jacob, Spencer W. NG, D.T. Wang, “Memory Systems, Cache, DRAM, Disk”, Elsevier Inc., 2008.

[10] B. Keeth and R.J. Baker, “DRAM Circuit Design: A Tutorial,” IEEE Press, Series on Microelectronic Systems, 2001.

[11] A. K. Sharma, “Semiconductor Memories: Technology, Testing, and Reliability,” IEEE PRESS, 1997.

[12] B. Prince, “Semiconductor Memories: A Handbook of Design, Manufacture and Application,” John Wiley & Sons Ltd., 1996.

[13] K. Itoh, “VLSI Memory Chip Design,” Springer-Verlag, Berlin, Germany,2001.

[14] S. Hamdioui, G. N. Gaydadjiev A.J.van de Goor, “A Fault Primitive Based Analysis of Dynamic Memory Faults”, IEEE 14th Annual Workshop on Circuits, Systems and Signal Processing, Veldhoven, the Netherlands, 2003.

[15] J. P. Hayes, "Testing Memories for Single-Cell Pattern-Sensitive Faults," IEEE Transactions on Computers, vol. 29, no. 3, pp. 249-254, 1980.

[16] D-C. Kang, S.M. Park and S-B Cho, “An Efficient Built-In Self Test Algorithm for Neighborhood Pattern and Bit-Line-Sensitive Faults in High Density Memories,” ETRI Journal, vol. 26, no. 6, pp. 520-534, 2004.

[17] D-C. Kang and S-B Cho, “An Efficient Built-In Self Test Algorithm for Neighborhood Pattern and Bit-Line-Sensitive Faults in High Density Memories,” IEEE KORUS, pp. 218-223, 2000.

[18] J.Y. Kim, S.J. Hong and j. Kim, “Parallely Testable Design for Detection of Neighborhood Pattern Sensitive Faults in High Density DRAMs,” IEEE Int. Symposium on Circuits and Systems, pp. 5854-5857, 2005.

[19] Y-J. Huang and J-F. Li, “Testing Active Neighborhood Pattern-Sensitive Faults of Ternary Content Addressable Memories,” IEEE European Test Symposium, pp. 55-60, 2006.

[20] K-L. Cheng, M-F. Tsai and C-W Wu, “Efficient Neighborhood Pattern-Sensitive Fault Test Algorithms for Semiconductor Memories,” IEEE VLSI Test Symposium, pp. 225-230, 2001.

[21] K-L. Cheng, M-F. Tsai and C-W Wu, “Neighborhood pattern sensitive fault testing and diagnostics for random access memories,” IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 21, no. 11, pp. 1328–1336, 2002.

[22] A. J. van de Goor and I.B.S. Tlili, “Disturb Neighborhood Pattern Sensitive Fault,” IEEE Int. VLSI Test Symposium, pp. 37-45, 1997.

[23] S. Banerjee, D.R. Chowdhury and B.B. Bhattacharya, “A Programmable Built-In Self-Test for Embedded DRAMs,” IEEE Int. Workshop on Memory Technology Design and Testing, pp. 58-63, 2005.

[24] S. Banerjee, D.R. Chowdhury and B.B. Bhattacharya, “A Programmable Built-In Self-Test for Embedded Memory Cores,” IETE Technical Review, vol. 24, no. 4, pp. 287-311, 2007.

[25] C. Wilkerson, A. Alameldeen, Z. Chishti, “Scaling the Memory Reliability Wall,” Intel Technology Journal, vol. 17, no. 1, pp. 18-34, 2013.

[26] M. Franklin, K. Saluja and K. Kinoshita, “A Built In Self Test Algorithm for Row/Column Pattern Sensitive Faults in RAMs,” IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 514-524, 1990.

[27] Y. Sfikas and Y. Tsiatouhas, “Physical Design Oriented DRAM Neighborhood Pattern Sensitive Fault Testing,” IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 108-113, 2009.

[28] Y. Sfikas, Y. Tsiatouhas, and S. Hamdioui, “Layout-Based Refined NPSF Model for DRAM Characterization and Testing,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 6, pp. 1446-1450, 2014.

[29] Z. Al-Ars, Ad J. van de Goor, “Static and Dynamic Behavior of Memory Cell Array Opens and Shorts in Embedded DRAMs,” Design Automation and Test in Europe, pp. 496-503, 2001.

[30] S. Hamdioui, Z. Al-ars and A.J. van de Goor, Testing Static and Dynamic Faults in Random Access Memories", In Proc.of IEEE VLSI Test Symposium, pp. 395-400, 2002.

[31] H.-Y. Yang, C-M Chang, M. C.-T. Chao, R.-F. Huang and S.-C. Lin, “Testing Methodology of Embedded DRAMs,” IEEE Transactions on Very Large scale Integration (VLSI) Systems, vol. 20, no 9, pp. 1715-1727, 2012.

[32] C-M. Chang, M. C-T. Chao, R-F. Huang and D-Y Chen, “Testing Methodology of Embedded DRAMs,” IEEE International Test Conference, p. 25.3, 2008.

[33] Z. Al-Ars, S. Hamdioui, Ad J. van de Goor, G. Gaydadjiev, and J. Vollrath, “DRAM-Specific Space of Memory Tests,” IEEE International Test Conference, p3.3, 2006.

[34] Y. Sfikas and Y. Tsiatouhas, “Testing Neighbouring Cell Leakage and Transition Induced Faults in DRAMs,” IEEE Transactions on Computers (accepted for publication), online access: DOI: 10.1109/TC.2015.2479606, 2015.

[35] Y. Sfikas, Y. Tsiatouhas, M. Taouil and S. Hamdioui, “On Resistive Open Detection in DRAMs: The Charge Accumulation Effect,” IEEE European Testing Symposium (ETS) 2015.

[36] G. Harutunyan, V. Vardanian and Y. Zorian, “Minimal March Tests for Dynamic Faults in Random Access Memories,” IEEE European Test Symposium, pp. 43-48, 2006.

[37] J.A. Mandelman, R.H. Bennard, G.B Bronner, J.K. DeBrosse, R. Divakaruni, Y. Li and C.J. Radens, “Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM),” IBM Journal of Research and Development, vol. 46, no. 2/3, pp. 187-212, 2002.

[38] P.K. Chatterjee, G.W. Taylor and A.F. Tasch, “Leakage studies in high-density dynamic MOS memory devices,” IEEE Journal of Solid-State Circuits, vol. 14, no. 2, pp. 486-497, 1979.

[39] E. Gizdarski, “Built-in self-test for folded Bit-Line Mbit DRAMs”, Elsevier Integration, the VLSI Journal 21 , pp. 95-112, 1996.

[40] M. C-T. Chao, H-Y Yang, R-F. Huang, S-C. Lin and C-Y. Chin, “Fault Models for Embedded DRAM Macros,” ACM/IEEE Design Automation Conference, pp. 714-719, 2009.

[41] Z. Yang and S. Mourad, “Crosstalk in Deep Submicron DRAMS,” IEEE International Workshop on Memory Technology, Design and Testing, pp. 125-129, 2000.

[42] W. Jeong, I. Kim, S. Lee, O. Kwon, E. Kal, K. Lee, H. Bae, K. Lee and S. Kang, “A Novel Screen-Ability Estimation Methodology for DRAM with a Test Algorithm Simulator: FS5,” International Technical Conference on Circuits/Systems, Computers and Communication, pp. 927-929, 2010.

[43] S. Boutobza, M. Nicolaidis, K. M. Lamara and A. Costa, “A Transparent Based Programmable Memory BIST,” IEEE European Test Symposium, pp. 89-94, 2006.

[44] R.R. Julie, W.H. Wan Zuha and R.M. Sidek, “12N Test Procedure for NPSF Testing and Diagnosis for SRAMs,” Proc. IEEE International Conference on Semiconductor Electronics, pp. 430-435, 2008.

[45] V. Yarmolik, Yu. Klimets and S. Demidenko, “March PS(23N)Test for DRAM Pattern-Sensitive Faults,” Proc. IEEE Asian Test Symposium, pp. 354-357, 1998.

[46] Y. Matsubara, et al, “Fully Compatible Integration of High Density Embedded DRAM with 65nm CMOS Technology (CMOS5),” IEEE Electron Devices Meeting, pp. 423-426, 2003.

[47] C-H. Chung and J-W. Chien, “Memories Having Charge Storage Node at Least Partially Located in a Trench in a Semiconductor Substrate and Electrically Coupled to a Source/Drain Region Formed in the Substrate,” US Patent 7,348,622 B2, 2008.

[48] H.D. Oberle and P. Muhmenthaler, “Test Patten Development and Evaluation for DRAMs with Fault Simulator RAMSIM,” IEEE Int. Test Conference, pp. 548-555, 1991.

[49] Z. Al-Ars, S. Hamdioui, G. Gaydadjiev, “Optimizing Test Length for Soft Faults in DRAM Devices,” IEEE VLSI Test Sym., pp. 59-66, 2007.

[50] S. Henzler, “Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies,” Springer, 2010.

[51] F. Karimi, S. Irrinki, T. Crosby, N. Park and F. Lombardi, “Parallel Testing of Multi-Port Static Random Access Memories,” Microelectronics Journal, vol. 34, pp. 3-21, 2003.

[52] D-S. Min and D. Langer, “Multiple Twisted Data Line Techniques for Coupling Noise Reduction in Embedded DRAMS,” IEEE Custom Integrated Circuits Conference, pp. 231-234, 1999.

[53] P. Mazumder, “Parallel Testing of Parametric Faults in a Three-Dimensional Dynamic Random-Access Memory,” IEEE Journal of Solid-State Circuits, vol. 23, no. 4, pp. 933-941, 1988

[54] H.-D. Oberle, M. Maue, and E. Muhmenthaler, “Enhanced Fault Modeling for DRAM Test and Analysis,” Dig. 1991 IEEE VLSI Test Syrup., Atlantic City, NJ, pp. 149-154, April 15-17, 1991.

[55] M. C.-T. Chao, H-Y Yang, R-F Huang, S-C Lin and C-Y Chin, “Fault Models for Embedded-DRAM Macros,” ACM Design Automation Conference (DAC), pp. 714-719, 2009.

[56] Z. Al-Ars, S. Hamdioui, Ad J. van de Goor, G. Mueller, “Defect Oriented Testing of the Strap Problem Under Process Variations in DRAMs,” IEEE International Test Conference, p 30.1, 2008.

[57] L. Dilillo, P. Girard, S. Pravossoudovitch and A. Virazel, "Resistive-Open Defects in Embedded-SRAM Core Cells: Analysis and March Test Solution," IEEE Asian Test Symp., pp. 266-271, 2004.

[58] H. Shin, Y. Park, G. Lee, J. Park, and S. Kang, “Interleaving Test Algorithm for Subthreshold Leakage-Current Defects in DRAM Considering the Equal Bit Line Stress, ” IEEE Tran. On VLSI Systems, Vol. 22, No. 4, pp. 803-812, 2014.

[59] M. M. Mano, “Digital Design,” Prentice Hall, 2002.

AUTHOR'S PUBLICATIONS

- Y. Sfikas and Y. Tsiatouhas, "Physical Design Oriented DRAM Neighborhood Pattern Sensitive Fault Testing," IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 108-113, 2009 (Best Paper Award).
- Y. Sfikas, Y. Tsiatouhas, and S. Hamdioui, "Layout-Based Refined NPSF Model for DRAM Characterization and Testing," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 6, pp. 1446-1450, 2014.
- Y. Sfikas and Y. Tsiatouhas, "Testing Neighbouring Cell Leakage and Transition Induced Faults in DRAMs," IEEE Transactions on Computers (accepted for publication), online access: DOI: 10.1109/TC.2015.2479606 , 2015.
- Y. Sfikas, Y. Tsiatouhas, M. Taouil and S. Hamdioui, "On Resistive Open Detection in DRAMs: The Charge Accumulation Effect," IEEE European Testing Symposium (ETS) 2015.
- Y. Sfikas and Y. Tsiatouhas, "Efficient DRAM Memory Testing Algorithms," Panhellenic Conference on Electronics and Telecommunications, p. P2.1, March 2012.
- Y. Sfikas and Y. Tsiatouhas, "Testing High Density Nanometer Technology DRAMs," 2nd Workshop on Modern Circuits and Systems Technologies, 2013.

SHORT VITA

Yiorgos Sfikas received the B.S. degree in physics in 1998, the B.S. degree in computer science in 2006 and the M.S. degree in 2009 in computer science, all from the University of Ioannina, Greece. Currently he is a Ph.D. candidate at the Computer Science Department of the University of Ioannina, Greece. From 2001 to 2009 he was with the Epirus Institute of Technology as a visiting professor. He is currently working as a second grade school teacher on computer science. He received the best paper award of the 2009 IEEE International Symposium on Design and Diagnostics of Electronic Circuit and Systems. His research interests include logic and memory integrated circuit design and design for testability and low power design.

GRANT ACKNOWLEDGEMENT

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.