UNIVERSITY OF ALBERTA

A Hierarchy of DRAM Testing Problems

BY

Lin Shen ©

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In partial fulfillment of the requirements for the degree

of Master of Science

Department of Electrical Engineering

EDMONTON, ALBERTA

Spring, 1994

ISBN 0-612-11370-1

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: **Lin Shen**

TITLE OF THESIS: **A Hierarchy of DRAM Testing Problems**

DEGREE: **Master of Science**

YEAR THIS THESIS GRANTED: **1994**

[illegible]

Permanent Address:
3B
8913, 112 St.
Edmonton, Alberta,
Canada

Date

Dec 15, 1993

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Hierarchy of DRAM Testing Problems** submitted by **Lin Shen** in partial fulfillment of the requirements for the degree of **Master of Science.**

[illegible]

Supervisor: Dr. B.F. Cockburn

[illegible]

Dr. J.T. Mowchenko

[illegible]

Dr. P.G. van Beek

Date

Nov. 23, 1993

Abstract

In this thesis, we apply a formal methodology for memory testing and diagnosis using deterministic Mealy automata to the case of a 4 Mbit DRAM. Based on an informal description of the faulty behaviours, a realistic formal fault model is developed that differs substantially from traditional fault models that contain only stuck-at faults, coupling faults, and pattern sensitive faults. Based on this fault model, we propose a hierarchy of testing problems that includes fault detection, faulty cell location, fault type location, and fault diagnosis. For the problem of fault detection, we derive lower bounds of $5n$ and $5n - 2$ on the lengths of any march test and any unrestricted test, respectively, where n is the number of single bit memory words. An optimal march test of length $5n$ is also proposed for this problem and proved correct. For the problem of faulty cell location, we derive a lower bound of $8n$ on the length of any march test that achieves the location. Again, an optimal march test of length $8n$ is proposed and proved correct. For the problem of fault type location, we derive a lower bound of $8n$ on the length of any march test, and we propose a near-optimal march test of length $13n$ that, in the best case, can locate all of the fault types in the fault model. For the problem of fault diagnosis, we report some preliminary work and conjecture on the existence of a lower bound proportional to $n \log_2 n$.

Acknowledgements

I wish to express my sincere gratitude to my supervisor, Dr. Bruce F. Cockburn, for his support, patience and insightful guidance throughout this thesis work. Dr. Cockburn's encouragement and valuable contribution have been essential to the progress of this thesis. I would also like to thank the other members of my thesis committee for their helpful comments.

I would also like to thank the many other faculty, staff, and fellows students for their criticism, suggestions, discussion, and all the good time we had together.

Finally, I would like to acknowledge the financial support of the National Science and Engineering Research Council under grant OGP105567.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation for Testing and Diagnosing DRAMs

Semiconductor memory technology continues to make impressive advances. Multi-megabit memory chips are now available on the market, and gigabit memory chips seem likely to appear by the end of this decade [17]. Accompanying this rapid growth in memory chip density, *i.e.*, the number of bits per chip, is increased concern about testing and reliability.

Conventionally, semiconductor memory chips are divided into five different types according to their functionalities and architectures: 1) Dynamic Random Access Memories (DRAMs); 2) Static Random Access Memories (SRAMs); 3) Read-Only Memories (ROMs); 4) Erasable and Programmable Read-Only Memories (EPROMs); and 5) Electrically Erasable and Programmable, Read-Only Memories (EEPROMs). Of these memory types, DRAMs have the largest production volume. Since the beginning of the 1970s, the dynamic RAM has been the main technology driver for the

Table 1.1: Past and Projected Memory Market Share [18]

	1988	1990	1994
DRAMs	56%	54%	58%
SRAMs	17	22	21
ROMs	8	8	6
EPROMs	17	14	12
EEPROMs	2	2	3
Total	100%	100%	100%

semiconductor industry. Table 1.1 shows the market shares for each of the above five types of memory chips [18].

Ongoing developments in DRAM technology have resulted in continually increasing memory chip densities. As shown in Fig. 1.1, DRAM die size has grown by 40% to 50% every generation; at the same time, memory cell size has decreased by 60% to 65% every generation. The growth in die size and the shrinking of memory cell size are expected to continue as DRAMs approach the gigabit level [12].

Along with increasing die sizes and shrinking cell sizes, major cell parameters such as feature width, chip area, and pitch width have dramatically decreased. As a result of the decrease, DRAM chips have tended to become more vulnerable to faults for at least the following two reasons [36, 15]:

- The growing die size produces larger metal resistance, which leads to such circuit problems as wiring delay, crosstalk, noise, and V_{ss} bounce.

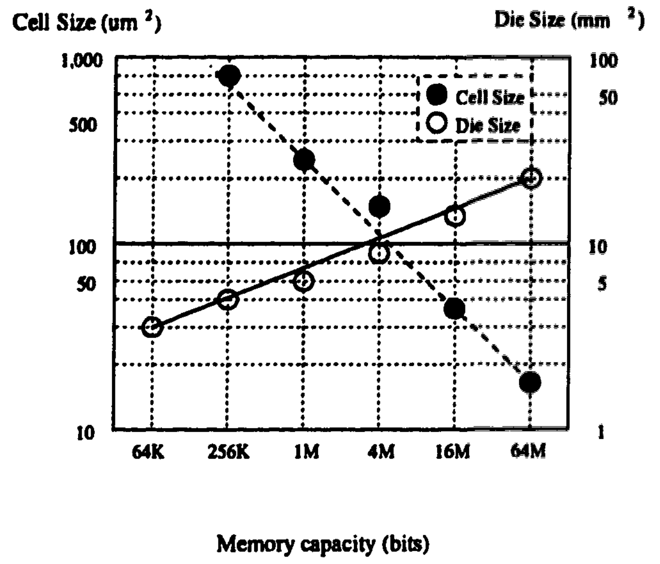


Figure 1.1: Trends in DRAM die size and memory cell size [12].

- Crosstalk problems get worse as signal wires are packed more closely together.

Major factors contributing to the chip cost of DRAMs are process complexity, die size, equipment cost, test cost, and yield. A consequence of the increasing capacity and susceptibility to defects of DRAMs is that the test cost is growing at an especially rapid rate. This situation is illustrated by Fig. 1.2 [12], which shows the ratio of test cost to total chip cost as a function of memory capacity.

Economic considerations have been playing a progressively larger role in DRAM chip design and manufacturing. Test cost is directly related to test time. Thus to keep the test cost low, novel methods that can speed up the DRAM testing process and improve its quality are of interest to test engineers [18].

As memory chip density increases, the production of chips without defects becomes increasingly difficult, and the yield has a tendency to decrease. Rejection of

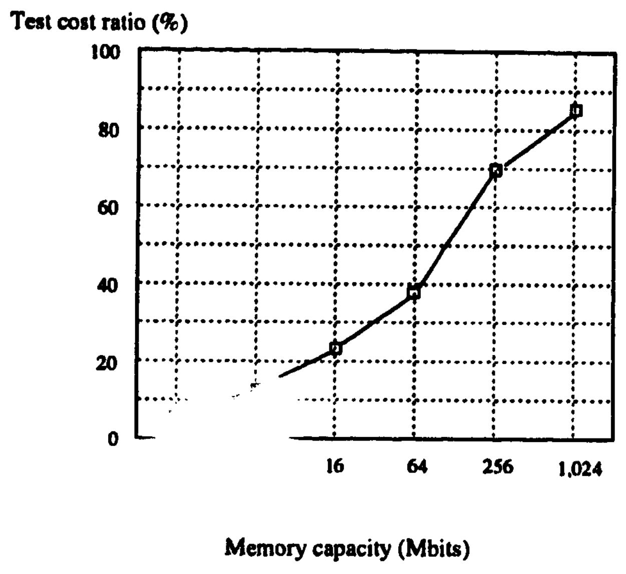


Figure 1.2: Ratio of test cost to total chip cost [12].

a chip, and thus an overall lower production yield, can be caused by only a small number of faults. Therefore, to obtain economically acceptable yields, semiconductor manufacturers have employed fault-tolerant memory designs. The main idea of these designs is to add spare rows and columns of memory cells. Faulty memory cells can be replaced with either a spare row or spare column by reprogramming the modified row and/or column decoders using laser or electrical fusing [31, 14, 3]. This technique, which was originally used as a yield improvement aid for the early stages of memory development, has become an essential activity in the design and manufacture of today's multi-megabit memories. To perform memory repair, production tests are required not only to detect at least one fault in a faulty memory (fault detection), but also to locate all of the cells affected by all of the faults (faulty cell location), to determine which cells are involved in each type of fault (fault type location), and to determine all of the faults present, including the type and set of cells associated with each fault component (fault diagnosis) [7].

1.2 Thesis Overview

The three specific problems studied in this thesis are 1) *fault detection*, 2) *faulty cell location*, and 3) *fault type location* under a realistic DRAM fault model. The fault model is based on the physical defects and resulting faulty behaviours observed in 4 Mbit DRAMs manufactured by Siemens [21].

March tests are a family of RAM tests that are relatively simple in structure and have linear time complexity. They are also very attractive from an on-chip testing point of view because of their regularity and hence ease of implementation. Therefore, in this thesis we will usually restrict our discussion to march tests.

In Chapter 2, we review the structure and operation of a typical DRAM. Some unique features of the Siemens DRAM are also described.

In Chapter 3, we provide background on memory testing and diagnosis.

Chapter 4 describes the fundamental definitions, notation and methodology used in this thesis. Brzozowski and Cockburn developed a formal methodology for RAM testing using the theory of finite automata [5]. This methodology allows us to derive fault models that are suitable for rigorous analysis. Our lower bound proofs and fault coverage analysis are all based on this methodology. Hence a description of this methodology is also included in Chapter 4.

Oberle *et al.* developed a realistic fault model based on the physical defects and resulting faulty behaviours that were observed in 4M $\times 1$ DRAMs manufactured by Siemens [21]. In addition to stuck-at faults, the fault model also includes more complex faults such as interconnected cells, word line shorts, bit line shorts, world line to bit line shorts, and interrupted bit lines. However, the fault model is specified informally and hence is not suitable for applying mathematical proof techniques. In Chapter 5, we present a slightly generalized formal version of the same fault model. The formal fault model forms the basis for the remaining chapters of this thesis.

Chapter 6 addresses the fault detection problem. To detect a fault we mean that at least one of the faulty cells in a faulty memory must be identified. We derive a lower bound of $5n$ on the length of any march test that detects all faults in the fault model, where n is the number of memory cells. We also derive a lower bound of $5n - 2$ on the length of any boolean test of arbitrary structure that detects all faults in the fault model. A march test whose length matchs the $5n$ lower bound is also provided in this chapter. We then prove that the march test has 100% fault coverage.

In Chapter 7, we investigate the faulty cell location problem. To locate the faulty cells of a fault we mean that all of the faulty cells involved in a fault must be determined. We derive a lower bound of $8n$ on the length of any march test that locates all faulty cells in the fault model. We also propose a march test whose length matchs the lower bound and then prove that it indeed locates all faulty cells in the fault model.

In Chapter 8, we investigate the fault type location problem. To locate all the fault types we mean that all of the faulty cells involved in all faults of each fault type must be detected. Since equivalent faults cannot be distinguished by pure boolean tests, we have to eliminate all redundant faults in the fault model before considering fault type location. An irredundant fault model is therefore first derived from the original fault model. Then we propose a march test of length $13n$ and prove that this test locates all fault types in the irredundant fault model. We do not provide a greater lower bound than $8n$ due to the increased difficulty of the fault type location problem (Note: any test that performs fault type location must at least perform faulty cell location).

Finally in Chapter 9, the results obtained in the preceding chapters are summarized and the main contributions of this thesis are enumerated. Discussions of potential areas of future research are also included.

Chapter 2

Review of DRAM Structure and Operation

A basic knowledge of DRAM structure and operation is required to better understand the following chapters, especially the descriptions of the fault mechanisms present in the realistic DRAM fault model. The DRAM structure discussed here is meant to be representative of the many different DRAM designs that have been published; usually each DRAM manufacturer uses a different design. However, the fundamental principles of all DRAM designs are similar. Where it is appropriate, we will describe the unique features of the Siemens DRAM.

Dynamic random-access memories store binary data as charges on an array of capacitors. Each storage cell uses one capacitor to store one bit of data. The DRAM is random-access because data can be stored to and fetched from all cells at approximately the same speed. DRAMs are volatile, which means that once the power to the DRAMs is shut off, the stored data will be lost. However, compared with other semiconductor memories, DRAMs have some advantages. The basic memory cell of a DRAM consists of only one transistor and one capacitor, which means that very dense DRAMs can be made. Other memories, such as static random access memories (SRAMs), require more circuitry per bit (4 or 6 transistors per bit in the case of SRAMs). Therefore DRAMs have a lower cost per bit than other memories. Because DRAMs consume only transient current, the power dissipation can be quite low. The main disadvantage of DRAMs is that the memory cells need to be refreshed. Because of the finite reverse-biased junction leakage current of the cell transistor, the amount of stored charge will decrease with time. In order to guarantee that a read operation will still produce the logic value originally stored in the cell, the charge must be periodically sensed and restored to its original voltage; otherwise, the data will be irretrievably lost. The need for refreshing makes the supporting circuitry for DRAMs more complicated and hence their access times tend to be slower than some other memories such as SRAMs. In spite of this disadvantage, DRAMs are the most widely used semiconductor random access memories.

2.1 DRAM Structure

Figure 2.1 depicts a simplified view of the architecture of a typical $n \times 1$ DRAM, where n is the number of storage cells and the "$\times 1$" indicates that one storage cell is associated with each of n distinct addresses. If $k > 1$ bits are associated with each address then we have an $n \times k$ architecture. In this thesis, as in most RAM

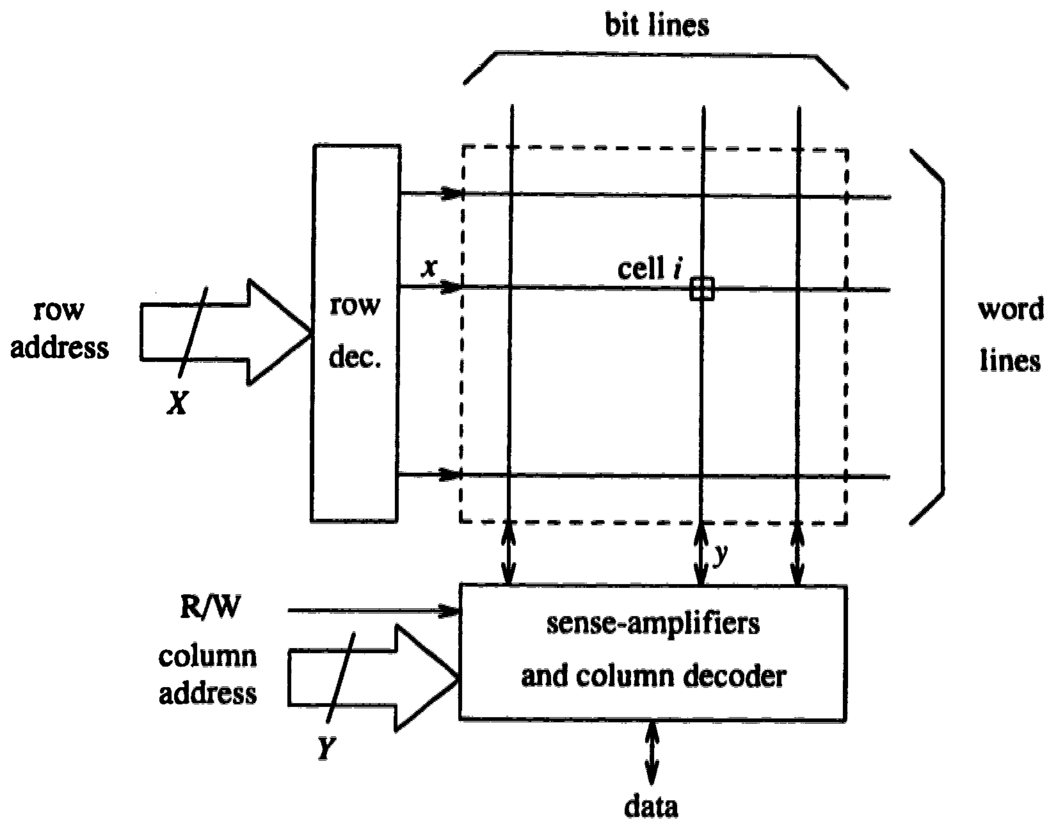


Figure 2.1: The architecture of a DRAM.

testing work, we will only consider $n \times 1$ RAMs.

As shown in Fig. 2.1, a DRAM consists of the following main components:

- Memory cell array: The cell array, where the information is stored, comprises a group of n storage cells. There are several possible implementations of DRAM cells. Among them, the single-device DRAM cell, which consists of an enhancement mode transistor and a separate capacitor, is the most commonly used design because it provides for a high-density, low cost per bit memory cell with a reasonable performance. For a good summary of single-device designs see [25].

In addition to storage cells, the cell array contains a group of parallel bit lines and a group of parallel word lines. The two groups of lines cross each other to form a rectangular grid. Every cell is located at an intersection of a bit line and a word line. By selecting a specific bit line and a specific word line we can uniquely address each cell. Figure 2.2 depicts a cell array in more detail. The horizontal lines labelled "WL" are word lines; the vertical lines labelled "BL" are bit lines. Here, as in most modern DRAMs, the storage cells are implemented as single-device cells. To access cell (3,1), for example, WL3 is activated and data is written or read via BL1.

Note that in Fig. 2.2, each sense-amplifier is connected to two adjacent bit lines. This is called a "folded bit line" layout and is typical of many modern memory designs and is used by the Siemens DRAM. As described in [21], in the Siemens DRAM's cell array, the rectangular grid is distorted in such a way that each cell is actually surrounded by three immediate neighbours.

- Decoders: These are used to access a particular cell or a group of cells (in word-oriented RAMs) in the memory cell array. Row and column decoders are used to activate one word line and one bit line, respectively. There are several popular implementations of row and column decoders. The schematics of a typical row decoder and a typical column decoder are shown in Figs. 2.3 and 2.4, respectively [24]. In a complete row decoder, each word line is driven by a separate circuit as in Fig. 2.3. The row address connections are different

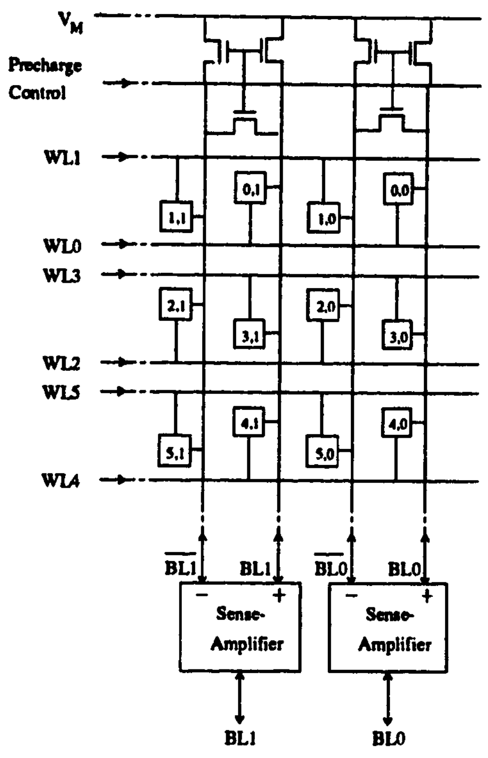


Figure 2.2: Detail of a DRAM cell array.

for each word line so that only one word line is activated for each possible row address. In the small column decoder shown in Fig. 2.4, each possible column address sets up a conducting path from the data line at the top to exactly one bit line at the bottom. Note that the data flows in both directions (upwards and downwards) through the column decoder.

- Sense-amplifier: A sense-amplifier is basically a combined differential amplifier and driver that operates together with clock signals and precharge circuitry. It is used to sense the voltage across the two connected bit lines and to thereby recover the value stored in the memory cell that is being read.

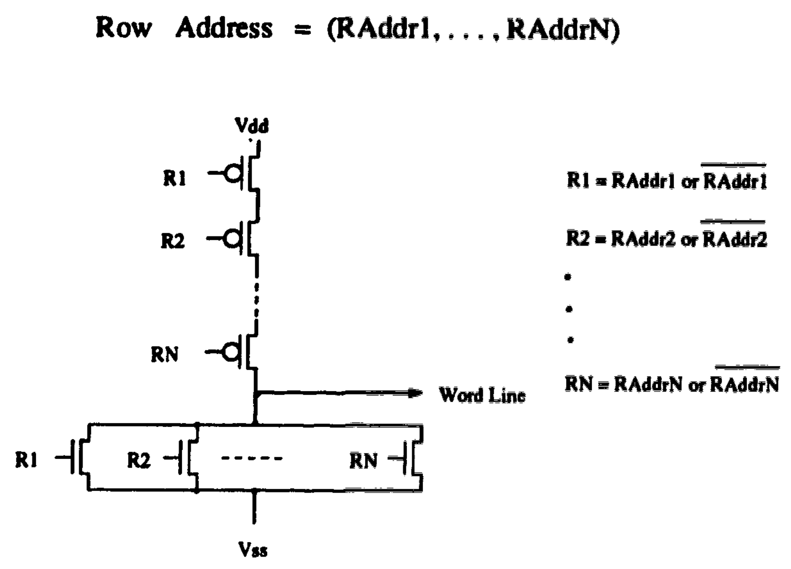

Row Address = (RAddr1, . . . , RAddrN)
Vdd
R1
R2
RN
Word Line
R1 = RAddr1 or $\overline{\text{RAddr1}}$
R2 = RAddr2 or $\overline{\text{RAddr2}}$
RN = RAddrN or $\overline{\text{RAddrN}}$
R1
R2
RN
Vss

Figure 2.3: The schematic of a row decoder.

Not shown in Fig. 2.1 are the following circuits:

- Refresh logic: Binary data are represented physically in a DRAM cell as either a high or low voltage charge stored dynamically on the *storage node* of a cell capacitor. This charge must be restored periodically, or else the data will be lost due to the finite cell leakage current. Basically, the refresh operation reads an entire row of cells at one time in parallel, then passes the data to the corresponding sense-amplifiers to determine the logic value stored in each cell; slightly later, the sense-amplifiers write-back the data such that the original charge is restored to all cells of that row. The sequence of events in the refresh operation is controlled by the refresh logic.

- Data latches: A group of latches in which input and output data words are stored.

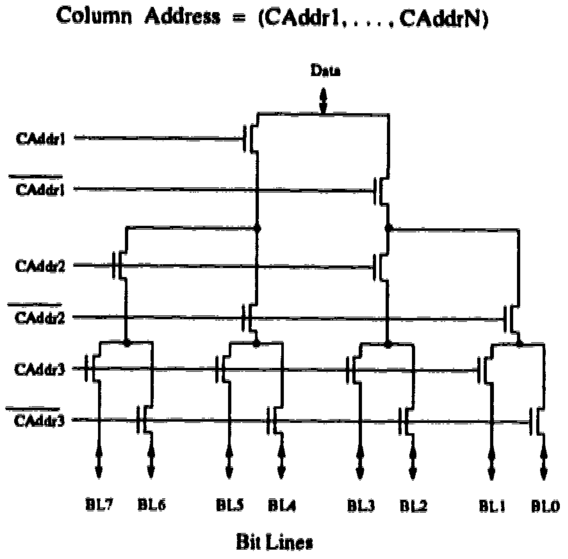

Column Address = (CAddr1, . . . , CAddrN)
Data
CAddr1
$\overline{\text{CAddr1}}$
CAddr2
$\overline{\text{CAddr2}}$
CAddr3
$\overline{\text{CAddr3}}$
BL7
BL6
BL5
BL4
BL3
BL2
BL1
BL0
Bit Lines

Figure 2.4: The schematic of a column decoder.

- Address latches: A group of latches in which the address of the memory cell to be accessed is stored. The row and column addresses for an operation are usually provided at different times on the same address lines to reduce the number of external connections. The falling edges of two signals, called RAS and CAS, are used to indicate the distinct times when the row and column addresses, respectively, are valid on the shared address lines.

2.2 DRAM Operation

In addition to read and write operations, it is possible for a DRAM to provide some other specialized operations such as read-modify-write, fast page mode, static column mode, *etc.*. However, usually only read and write operations are considered when DRAM testing and diagnosis are concerned. We therefore only describe these two operations in detail. To illustrate the two basic operations more clearly, consider read and write operations to cell i in Fig. 2.1.

A read operation to cell i does four things: 1) addresses cell i; 2) evaluates the charge stored in the capacitor of cell i; 3) routes the appropriate logic signal, either 0 or 1, to the output pin of the DRAM; and 4) writes the same logic signal back along the bit line to restore the stored change. We now describe these events in more detail. First, all pairs of bit lines are precharged to half of the positive supply voltage V_{DD}. This is achieved by precharge circuits at one end of the bit lines. Then, the word line to which the transistor of cell i is connected is activated (word line x in Fig. 2.1). This turns on the cell's transistor and thus causes a conducting path to be formed between the capacitor of cell i and the bit line to which the transistor of cell i is connected (bit line y). As a result, the voltage of bit line y is pulled down or raised up slightly depending on whether a valid high or a valid low voltage is stored in cell i. At the same time, since the other bit line that is connected to the same sense-amplifier as bit line y is not connected to any capacitor, its voltage remains unchanged, *i.e.,* half of V_{DD}. The slight difference in the voltage across the two bit lines is detected and amplified by the sense-amplifier to which they are connected so that a full-strength logic signal corresponding to the bit stored in cell i is produced at the output of the sense-amplifier. At the same time, the resulting voltage signal from the sense-amplifier is then routed through the column selector to the data latches and hence to the output pin of the DRAM.

A write operation to cell i does three things: 1) addresses cell i; 2) routes the input data from the input pin of the DRAM through the column decoder and along one bit line to the capacitor of cell i; and 3) charges or discharges the capacitor of cell i. As with read operations, a write operation has to first precharge the bit line pairs, and then form a conducting path between bit line y and cell i. Since the driven input signal is much stronger than the signals on the bit lines connected to the sense-amplifier, the voltage from the input pin of the DRAM is routed through bit line y and charges the capacitor of cell i.

If no cell on a particular row is accessed for a sufficiently long time (approaching 10ns), then all of the cells on that row can be refreshed by simply reading any one cell in the row. If refreshing is not done soon enough, then the stored data will start to be lost.

Chapter 3

DRAM Testing Background

3.1 Basic Types of Memory Tests

DRAM testing includes functional testing and parametric testing. *Functional testing* verifies whether the DRAM chip under test is fault-free with respect to its logical behaviour. Two types of DRAM functional tests, deterministic and pseudo-random tests, are commonly used. By *deterministic tests* we mean that the test patterns and reference data for the memories under test are predetermined before the tests are executed. By *pseudo-random tests* we mean that the test patterns for the memories under test are determined pseudo-randomly during the test. *Parametric tests* verify whether the DRAM under test works properly with respect to the specified parameters of the chip such as voltage/current levels and propagation delays from the input pins to the output pins of the chip. Two types of parametric tests, DC parametric and dynamic tests, are commonly used. In thesis we will only be concerned with functional tests. All of the DRAM tests that are proposed in this thesis are deterministic functional tests. If we assume that a test can store intermediate results and make use of those results for further testing, then the test is said to be an *adaptive* test; otherwise, the test is said to be a *nonadaptive* test. In this thesis, as with most previous research, we assume that the tests are nonadaptive.

3.2 Fault Modelling

According to the way the faults affect DRAMs, we can divide them into two different types: permanent and nonpermanent faults. *Permanent faults* cause DRAMs to malfunction all of the time. An example of a permanent fault is a broken or incorrect connection between components such as memory cells, word lines, and bit lines. *Nonpermanent* faults randomly cause a DRAM to malfunction for finite periods of time. A common example of a nonpermanent fault is a loose connection between two components. Due to the uncertain characteristics of nonpermanent faults, it is very hard to model them with functional fault models. Also, to detect some of the nonpermanent faults, external equipment and detailed knowledge of the fabrication technology must be used. In this thesis, we only consider permanent faults.

It has been shown that the the length of any functional test that can detect all possible functional faults in a memory is $O(2^n)$, where n is the size of the memory under test [11]. Therefore, it is not feasible in practice to consider all possible faults that may occur in DRAMs when designing tests. As a result, all practical tests restrict themselves to a subset of all possible faults, *i.e.*, by choosing a fault model.

A DRAM chip typically consists of many functional blocks including the cell array, row and column decoders, read/write logic, and timing and enabling circuits (including refresh logic). Conventionally, however, most testing procedures only deal with faults that occur in the memory cell array. This is because the frequency of faults occurring in the field of peripheral circuits is much lower than in the cell array and, as we show later, most of the faults that occur in decoders and read/write logic circuits can be modelled as faults that occur in the cell array [20, 23]. Defects in the timing and refresh circuits usually induce time-dependent faults that are quite different from the faults in cell array. Normally, dynamic tests are used to detect these faults. In this thesis we will not discuss defects that occur in timing and refresh circuits.

The following faults are commonly used to model the functional behaviour of faulty memory cells [33]:

- Stuck-at fault (SAF): The logic value of the affected cell is always 0 (stuck-at-0 fault) or 1 (stuck-at-1 fault). That is, the content of the cell can never be changed to the opposite logic value.

- Transition fault (TF): A memory cell fails to undergo either 0 to 1 or 1 to 0 transition when a write operation is applied to the cell. In other words, a cell may make one transition, but then will forever appear like a stuck-at fault.

- Coupling fault (CF): A write operation to a memory cell causes an unexpected transition in other memory cell. If k memory cells are coupled together, then we call the fault a k-coupling fault, where $1 \leq k \leq n$, where n is the capacity of the memory. In practice, k is usually much smaller than n. The 2-coupling fault has appeared in many fault models and has been extensively studied [20, 23, 5]. According to the faulty behaviours, there can be quite a few coupling relations between a pair of coupled cells. For example, for a 2-coupling fault that consists of two coupling cells i and j, there are four different relations: 1) inversion coupling (CFin), 2) idempotent coupling (CFid), 3) bridging coupling (BF), and 4) state coupling (SF). For the definition of these coupling relations, refer to [33].

- Neighborhood pattern sensitive fault (NPSF): The content of a cell, or transitions in the content of a cell, are influenced by the contents of some other cells in the memory. For example, consider four distinct cells i, j, k, and l. One possible pattern sensitive fault involving these four cells is the situation when, if cells j and k contain 1, and cell l contains 0, then cell i cannot make a 1 to 0 transition when a 0 is written to cell i. As in coupling faults, an arbitrary number of cells can be involved in a pattern sensitive fault. In practice, however, the number is usually restricted to be less than or equal to 5. Several different pattern sensitive faults have been studied [33]: 1) active neighbourhood pattern sensitive fault, 2) passive neighbourhood pattern sensitive fault (PNPSF), and 3) static neighbourhood pattern sensitive fault (SNPSF). Refer to [33] for the definitions of these different types of pattern sensitive faults.

An instance of any of the above listed faults is called a *single fault*. It is also possible for several such faults to appear simultaneously. Such a fault is called a *multiple fault*. If the component faults in a multiple fault involve disjoint sets of cells, then the component faults are said to be *unlinked.* If two component faults have at least one faulty cell in common, then those faults are said to be *linked.* In this thesis we will make the simplifying assumption that the component faults in any multiple fault are unlinked.

As we mentioned earlier, in addition to faults in the cell array, faults can occur in other functional blocks of the DRAM. However, those faults are usually modelled as faults in the cell array.

Assuming that the decoders are not changed into sequential circuits, then any failure occurring in the decoders can be divided into one of the three following faulty behaviours:

1. No cell in the memory cell array is accessed.

2. Cells other than the addressed cell are accessed.

3. The addressed cell and some other cell(s) are accessed.

In the case of faulty behaviour 1, a valid but unchanging voltage signal will be generated by the peripheral circuitry of the memory and then read out. Therefore we can consider the addressed cell as a stuck bit. In the case of faulty behaviour 2, we can still consider the addressed cell as a stuck at bit. Note, however, that at least one other cell will be affected. Faulty behaviour 3 is assumed to be modelled as a multiple coupling fault [20].

The most common failures occurring in the read/write logic circuits are either that the data input and data output lines are stuck-at-0 (1), or that some of the input (output) lines are shorted together. For the former case, we can consider the failures as stuck-at faults; for the latter case, we can consider the failures as coupling faults [20].

In this way, we can model faults occurring in decoders and read/write logic circuits as faults in the memory cell array. Therefore, in order to test a memory, we need to concentrate on the faults in the memory cell array only.

3.3 Functional Tests

Various functional tests have been proposed and used in industry since the 1970's. The earlier tests were not based on high-level functional fault models. Four well-known traditional tests are:

- Zero-one [1]: It detects all SAFs. The test is very simple in structure and has a length of only $4n$.
- Checkerboard [4]: It detects all SAFs and shorts between adjacent cells, under the condition that the address decoder functions correctly. The test first writes 1 and 0 alternatively into all of the cells in either ascending or descending address order, reads out all of the cells; then it repeats the two steps by exchanging 1 and 0. Obviously, this test has a length of only $4n$ too.
- GALPAT [4]: It detects all SAFs, TFs, and CFs. This test has a length of $2(n + 2n^2)$. Therefore, it is not feasible for testing modern multi-megabit RAMs because the test times would be several days.
- Sliding Diagonal [33]: It detects all SAFs and TFs. This test has a length of $6n + 2n\sqrt{n}$.

Today DRAM functional testing is dominated by a family of tests called march tests. March tests are relatively simple and have linear complexity. Many march tests have been proposed to detect and locate faults in different fault models with different fault coverages and often different complexities. An excellent survey by van de Goor [33] describes many different march tests. Here, we only describe three typical march tests [34]:

- MAST+: Detects all SAFs and has a length of $5n$.
- March C-: Detects all unlinked SAFs, TFs, and some of CFs. This test has a length of $10n$.
- March B: Detects most linked SAFs, TFs, and CFs. This test has a length of $17n$.

In addition to these tests, built-in self-tests, parallel tests, and concurrent tests are also used for DRAM functional testing. However, they are beyond the scope of this thesis, so we omit describing them here. Detailed descriptions are available in [26, 32].

Chapter 4

Definitions and Notation[†]

4.1 Finite Automata

An *alphabet* is a finite, nonempty set of symbols. By convention, in this thesis we use letters in italics to denote symbols, and bold letters to denote sequences of symbols; e.g., $\mathbf{s} = abcd$. The *length* of a sequence $\mathbf{s}$, denoted by $|\mathbf{s}|$, is the number of symbols in $\mathbf{s}$; e.g., $|abcd| = 4$. If X denotes an alphabet, then X^* denotes the set of all sequences composed of zero or more symbols from X. We use ϵ to denote the null sequence, that is, the sequence of length 0.

A *prefix* of a sequence $\mathbf{s}$ is a sequence comprising 0 or more leading symbols of $\mathbf{s}$. A *suffix* of $\mathbf{s}$ is a sequence comprising 0 or more trailing symbols of $\mathbf{s}$. A *segment* of $\mathbf{s}$ is a prefix of a suffix of $\mathbf{s}$. A *subsequence* of $\mathbf{s}$ is a sequence obtained from $\mathbf{s}$ by deleting 0 or more symbols. For example, if $\mathbf{s} = abcacbaa$, then ϵ and $abca$ are prefixes, baa and ϵ are suffixes, ϵ and $cacb$ are segments, and $baca$ and ϵ are

[†]Many of these definitions appeared previously in [5].

subsequences of $\mathbf{s}$. The *concatenation* of two sequences $\mathbf{s}$ and $\mathbf{t}$, written $\mathbf{st}$, is a sequence comprising the symbols of $\mathbf{s}$ followed immediately by the symbols of $\mathbf{t}$. For example, if $\mathbf{s} = acd$ and $\mathbf{t} = cdef$, then the concatenation of $\mathbf{s}$ and $\mathbf{t}$ is $acdcdef$.

If S is a set, then 2^S denotes all possible subsets of S. For example, if $S = \{a, b\}$, then $2^S = \{\epsilon, a, b, ab\}$. A *nondeterministic finite Mealy automaton* is a quadruple $A = (Q, X, Y, \delta)$, where Q is the finite nonempty set of states, X is the input alphabet, Y is the output alphabet, $\delta : Q \times X \rightarrow 2^{Q \times Y}$ is the combined *transition and output function.* A *deterministic finite Mealy automaton* is a quintuple $A = (Q, X, Y, \delta, \lambda)$, where Q is the finite nonempty set of states, X is the input alphabet, Y is the output alphabet, $\delta : Q \times X \rightarrow Q$ is the *transition function*, and $\lambda : Q \times X \rightarrow Y$ is the *output function.* A deterministic finite Mealy automaton is just the special case of a nondeterministic finite Mealy automaton in which there is only one next state and only one output for each present state and input pair.

4.2 A Methodology for RAM Testing

In order to perform rigorous analysis for RAM testing and diagnosis, we need a formal mathematical framework on which the analysis can be based. Brzozowski and Jürgensen developed a mathematical model for general sequential circuit testing and diagnosis using deterministic Mealy automata [6]. By noting that memories are a specific family of sequential circuit, Brzozowski and Cockburn [5] adapted this model to a formal methodology for RAM testing and diagnosis. In this methodology, which we now describe, the behaviour of both good and faulty memories is represented unambiguously using finite automata.

A *binary value* is either 0 or 1. The *complement* of 0 is 1, and of 1 is 0. The complement of a binary value $b \in \{0,1\}$ is denoted by $\bar{b}$. Formally, a memory is a sequential circuit containing a group of storage cells; each cell is capable of storing one binary value which can be evaluated and changed when necessary. We use r_b to denote a read operation for which the expected value of is b, r^i to denote a read operation applied to cell i, and use r_b^i to denote that the expected value of a r^i is $b \in \{0,1\}$. Similarly, we use w_b to denote the operation of writing the value b, w^i to denote a write operation applied to cell i, and use w_b^i to denote the operation of writing b to cell i. The contents of a cell i is denoted by c^i, where $0 \leq i \leq n-1$. A *good memory* of size $n \geq 1$ is a deterministic Mealy automaton $M_0 = (Q_0, X, Y_0, \delta_0, \lambda_0)$, where $Q_0 = \{0,1\}^n$ is the set of memory states, $X = \{r^i, w_0^i, w_1^i \mid 0 \leq i \leq n-1\}$ is the input alphabet corresponding to all possible "operations" that can be performed on the memory, $Y_0 = \{0, 1, \#\}$ is the output alphabet, and $\delta_0 : Q \times X \rightarrow Q$ and $\lambda_0 : Q \times X \rightarrow Y$ are the "good" transition and "good" output functions, respectively, which are defined in Table 4.1. (Note that, in the table, we represent an arbitrary memory state $q \in Q_0$ using the notation $[c^0, \ldots, c^{n-1}]$, where $c^0, \ldots, c^{n-1}$ denote the contents of cells $0, \ldots, n-1$, respectively.) The value $\#$ is a convenient place-holding symbol used to denote the null output produced by a write operation. Note that the initial state of a memory is left unspecified; this reflects our assumption that each cell can power up containing randomly either 0 or 1.

By a faulty memory, we mean a memory that has different logic behaviour from that of a good memory. We assume that all faulty memories can be represented

Table 4.1: Transition and Output Functions for M_0

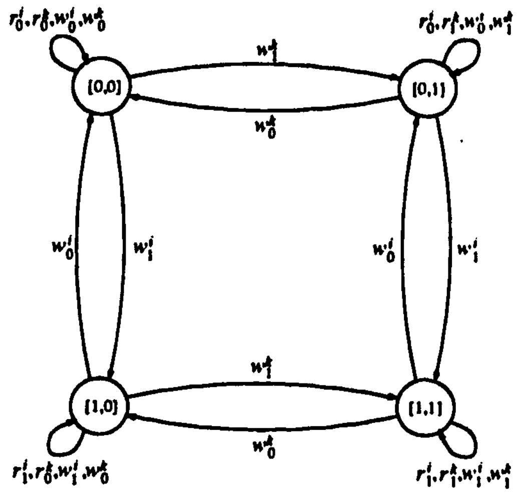
Input, x	Next State, $\delta_0(q, x)$	Output, $\lambda_0(q, x)$
r^i	$[c^0, \ldots, c^{i-1}, c^i, c^{i+1}, \ldots, c^{n-1}]$	c^i
w_0^i	$[c^0, \ldots, c^{i-1}, 0, c^{i+1}, \ldots, c^{n-1}]$	#
w_1^i	$[c^0, \ldots, c^{i-1}, 1, c^{i+1}, \ldots, c^{n-1}]$	#

as nondeterministic Mealy automata. A *faulty memory* or *fault* of size $n \geq 1$ is a deterministic Mealy automaton $M_i = (Q_i, X, Y_i, \delta_i, \lambda_i)$, where $Q_i \subseteq Q_0$ and $Y_i \subseteq Y_0$. Both M_0 and M_i have the same input alphabet X. However, the Q_i, Y_i, δ_i, and λ_i could be different from Q_0, Y_0, δ_0, and λ_0, respectively.

To better understand the definitions, we now provide two examples that specify, using finite automata, the behaviours of a pair of cells in both good and faulty memories.

Let $M_0 = (Q_0, X, Y_0, \delta_0, \lambda_0)$ be a deterministic Mealy automaton that describes the behaviour of a pair (i, k) of memory cells in a good memory. We use a transition diagram as shown in Fig 4.1, instead of the formal description in Table 4.1, to describe the transition function of M_0. Transition diagrams are often more convenient to use when describing memory cell behaviour. In Chapter 5, when we describe faults in DRAMs, we will also use transition diagrams to describe the transition functions.

The pair of numbers in each circle of the diagram represents the states of cells i and k. For example, "[0,0]" (which we will abbreviate 00) in the upper-left circle



$r_0^i, r_0^k, w_0^i, w_0^k$
$r_0^i, r_1^k, w_0^i, w_1^k$
w_1^k
[0,0]
[0,1]
w_0^k
w_0^i
w_1^i
w_0^i
w_1^i
w_1^k
[1,0]
[1,1]
w_0^k
$r_1^i, r_0^k, w_1^i, w_0^k$
$r_1^i, r_1^k, w_1^i, w_1^k$

Figure 4.1: The transition diagram of a good memory M_0.

indicates that both cells i and k contain logic 0. Here $Q_0 = \{00, 01, 10, 11\}$ is the set of all four possible joint states of the two cells. Since the only operations that can be applied to a memory are read and write operations, the input alphabet is $X = \{w_0^i, w_1^i, w_0^j, w_1^j, r^i, r^j\}$. The output alphabet is still $Y_0 = \{0, 1, \#\}$. The transition function is implied by the transition diagram in Fig 4.1. For example, if cells i and k are in state "00" and a read operation is applied to cell i (r^i), the state of cell i remains 0 and hence the joint states of cells i and k will remain "00". If cells i and k are in joint state "00" and a 1 is written to cell i (w_1^i), then the joint state of cell i will be changed to 1 and hence the state of cells i and k will be changed to "10".

Let $M_s = (Q_s, X, Y_s, \delta_s, \lambda_s)$ denote the deterministic Mealy automaton that describes the behaviour of a pair (i, k) of memory cells in a faulty memory in which cell i is stuck-at-0. Since cell i has only one state, $Q_s = \{00, 01\}$ has only two possible states instead of four. The input and output alphabets of M_s, however, are the same as M_0. The transition function for this fault is depicted by the transition diagram in Fig. 4.2. Since there is a fault in the memory, the transition diagram of M_s is different from that of M_0. For example, if cells i and j are in state "00" and a 1 is written to cell i, then the states of cells i and k will not be changed because cell i cannot be changed to 1. The output function λ_s for this fault is the same as λ_0 over the restricted domain $Q_s \times X$.

4.3 Testing Terminology

We give basic definitions for testing terms using the methodology described in the preceding section. Similar definitions of testing terminology have appeared in other references [33, 10, 13].

A *physical defect* is a deviation beyond the specified physical structure or composition of a device. A physical defect can be introduced during the manufacturing process by inappropriate operations such as dust particle contamination, over-etching, and mask misalignments. We assume that the erroneous behaviour of memories containing one or more physical defects can be specified mathematically using faults. Cell i is said to have an *error* if there exists no initialization (M_0, q) of a good memory, where $q \in Q$, such that cell i contains $b \in \{0, 1\}$ after a sequence of operations has been applied.

An *erroneous transition* is a transition from a state $q \in Q_i$ under an input $x \in X$ to a state other than $\delta_0(q, x)$, that is, a transition that differs from the corresponding transition in the good memory. For example in Fig. 4.2, applying an operation w_1^i to state 00, the resulting state is still 00. In a good memory, applying an operation w_1^i to state 00, causes a transition to 10. Thus, the transition in the fault is erroneous. If an output of M_i, $\lambda_i(q,x)$, is different from the output of M_0, $\lambda_0(q,x)$, then the output of M_i is said to be an *erroneous output.* A *nontrivial fault* is a fault that contains at least one erroneous transition. For convenience, we will sometimes refer to a *fault type* by which we mean a class of faults that share similar definitions. An example of a fault type is the set of all $2n$ possible single stuck-at faults affecting cells $0, \cdots, n-1$. The simultaneous presence of two or more faults in a memory is called a *multiple fault.* For example, if we assume that both cells i and k are stuck-at-0, then we have a multiple fault. Let $[q]_i$ denote the content of cell i in state q. With respect to a fault M, a *faulty cell* is a cell i such that there exists at least one test which causes an error to arise in cell i. A test t *locates* a faulty cell i in a memory M_f if $\lambda_f(q, r^i)$ is an erroneous output, where $q \in Q_f$. A *faulty cell address* is the address of a cell with an error.

In dynamic RAMs, a boolean signal is stored as a charge on a small capacitor. As we described in Chapter 2, the act of reading the logical value stored in a cell destroys the charge. To avoid data loss, read operations must be followed by write operations that restore the original data. Usually, the restore operation is performed automatically by the DRAM. The write operation that automatically follows a read operation is called a *write-back cycle.* For some types of faulty memories, it is possible for the write-back cycle that follows every read operation to cause an erroneous state transition. A state in a transition diagram is said to be *stable* if no read operation can cause further transitions from this state. Otherwise, the state is said to be *unstable.* Clearly all states in a good memory must be stable. Examples of stable and unstable states will be given in Chapter 5.

A *fault model* is a set of faults that represents all of the expected faulty behaviours. Let $M_0 = (Q_0, X, Y_0, \delta_0, \lambda_0)$ be a good memory. A *fault model* for M_0 is a finite family $\mathsf{F}_{M_0} = \{M_i = (Q_i, X, Y_i, \delta_i, \lambda_i) | i \in I\}$ of faults of M_0, where $I \in \{1, 2, 3, 4, ...\}$. Note that a fault model may contain multiple faults if it is convenient. For example, it is common to use multiple faults to represent the behaviour of memories containing multiple defects.

A *boolean test*, or simply a *test*, is a sequence $\mathbf{t} \in X^*$ composed of read and write operations. An *initialization* (M, q) of a memory M is a copy of M started in state $q \in Q$. The *response* $\rho(M, q)$ of an initialization (M, q) to a test $\mathbf{t} = x_1 \cdots x_p$ is the sequence $y_1 \cdots y_p$ of outputs produced when $\mathbf{t}$ is applied to (M, q). Two faults M and M' are said to be *equivalent* if, for all initial states $p, q \in Q$, if (M, p) and (M', q) have the same response for all tests $\mathbf{t} \in X^*$. It follows that no boolean test can be used to distinguish between two equivalent faults. Let F' denote a fault model obtained from fault model F by including one fault from each fault equivalence class. A fault model, such as F', that contains no equivalent faults is called *irredundant.* A test $\mathbf{t}$ *decides* between (M_0, q) and (M, p), where M is a fault of M_0 and $q, p \in Q$, if $\rho(M_0, q)$ differs from $\rho(M, p)$. A test $\mathbf{t}$ *detects* a fault M if $\mathbf{t}$ decides between (M_0, q) and (M, p), for all initial states $q, p \in Q$. For example, consider the fault M_s whose transition diagram is illustrated in Fig. 4.2. We assume, for simplicity, that the memory under test is only composed of two cells. Therefore, the cells i and k in Fig. 4.2 correspond to cell 0 and 1, respectively. Assume that we have a test $\mathbf{t} = w_0^0 w_1^0 r_0^0 r_1^1$, and two initial states $q = [0,0]$ and $p = [0,1]$. Then, the responses of a good memory M_0 and a faulty memory M_s to $\mathbf{t}$ are $\rho(M_0, q) = \#\#11$ and $\rho(M_s, p) = \#\#01$, respectively. Note that the two responses are different. Therefore, $\mathbf{t}$ decides between (M_0, q) and (M_s, p). Note also that no matter what initial states that M_0 and M_s start with, their responses to $\mathbf{t}$ are ##11 and ##01, respectively. Therefore, test $\mathbf{t}$ detects M_s. If a fault cannot be detected by any boolean test, then the fault is said to be *unclean*.

If $D \subseteq \{0, \ldots, n-1\}$ denotes a set of cell addresses, then we shall write $S_D \subseteq \mathsf{F}$ to denote the set of all (single or multiple) faults whose sets of faulty cell addresses are exactly D. Any irredundant finite fault model F can be partitioned into a finite set of classes $S_{D_1}, \ldots, S_{D_v}$, where $D_1, \ldots, D_v$ are distinct sets of faulty cell addresses. Given a set S_D of faults and a test $\mathbf{t}$, we shall write $R(S_D)$ to denote the set of all possible responses of faults in S_D to $\mathbf{t}$, for all possible initial states.

Consider a fault model $\mathsf{F} = S_{D_1} \cup \cdots \cup S_{D_v}$, where $S_{D_1}, \ldots, S_{D_v}$ are disjoint classes corresponding to the distinct sets $D_1, \ldots, D_v$ of faulty cell addresses. A test $\mathbf{t}$ is said to *locate the faulty cells* of an unknown fault $M \in \mathsf{F}$ if there exists exactly one set S_{D_i} such that, for all initial states $q \in Q$, the response of (M, q) to $\mathbf{t}$ appears in $R(S_{D_i})$ and does not appear in any other $R(S_{D_j})$, for $j \in \{1, \ldots, v\} - \{i\}$. A test $\mathbf{t}$ is said to *locate the faulty cells* of a fault model F if $\mathbf{t}$ locates the faulty cells of M, for all $M \in \mathsf{F}$.

Consider a set S_{D_i} of faults that share the same set D_i of faulty cell addresses. Given a fault $M \in S_{D_i}$, the set D_i can be partitioned into disjoint subsets $D_{i1}, \ldots, D_{is}$ such that each D_{ij}, for $1 \leq j \leq s$, contains only addresses of faulty cells caused by fault components of the same fault type. Any set S_{D_i} can therefore be partitioned into disjoint classes $S'_1, \ldots, S'_h$ of faults that have the same sets of faulty cell addresses for each fault type.

Consider a fault model $\mathsf{F} = S'_1 \cup \cdots \cup S'_m$, where $S'_1, \ldots, S'_m$ are disjoint classes corresponding to sets of faults that share the same sets of faulty cell addresses for each fault type. A test $\mathbf{t}$ is said to *locate the fault types* of a fault M if there exists exactly one set S'_i such that, for all initial states $q \in Q$, the response of (M, q) to $\mathbf{t}$ appears in $R(S'_i)$ and does not appear in any other $R(S'_j)$, for $j \in \{1, \ldots, v\} - \{i\}$. A test $\mathbf{t}$ is said to *locate the fault types* of a fault model F if $\mathbf{t}$ locates the fault types of M, for all $M \in \mathsf{F}$. It follows from the preceding definitions that any test that locates the fault types of M (F) must also locate the faulty cells of M (F).

Fault detection, faulty cell location and fault type location are key problems studied in this thesis. To better understand the definitions of these problems, we present the following example.

Figure 4.3 illustrates a memory with eight faulty cells A, B, C, D, E, F, G, and H. Among them, cells A and B, and C and D are involved in two different faults of the same fault type M_f; cells E and F, G, and H are involved in three different faults of different fault types M_m, M_g, and M_n, respectively.

For the faulty memory shown in Fig. 4.3, by fault detection we mean that a test $\mathbf{t}$ must detect at least one of the faulty cells A, B, C, D, E, F, G, and H. In other words, any such test $\mathbf{t}$ determines whether the memory under test is good or faulty. By faulty cell location we mean that a test $\mathbf{t}$ must detect all of the faulty cells A, B, C, D, E, F, G, and H. By fault type location we mean that a test $\mathbf{t}$ must not only detect all of A, B, C, D, E, F, G, and H, but also separate the faulty cells into the different fault types, *i.e.*, the set of faulty cells must be divided into four different subsets {A, B, C, D}, {E, F}, {G}, and {H}. By fault diagnosis, which we will not define formally, we mean that the test t determines the five subsets {A,B}, {C,D}, {E,F}, {G}, and {H}.

An *ascending (descending) march element* is a finite sequence $s \in \{r_0, r_1, w_0, w_1\}^*$ of operations repeated to each memory cell, in ascending (descending) address order. In this thesis we will use the notation $\Uparrow(s)$ ($\Downarrow(s)$) to denote an ascending (descending) march element, where $s \in \{r_0, r_1, w_0, w_1\}^*$. For example, $\Uparrow(r_0 w_1) = r_0^0 w_1^0 r_0^1 w_1^1 \cdots r_0^{n-1} w_1^{n-1}$ and $\Downarrow(w_0) = w_0^{n-1} w_0^{n-2} \cdots w_0^0$ are ascending and descending march elements, respectively, where n is the size of a memory. Naturally, $\Updownarrow(s)$ denotes a march element that can be either descending or ascending. A *march test* is a test that is composed of a series of zero or more ascending or descending march elements. For example, $\Updownarrow(w_0) \Uparrow(r_0 w_1) \Downarrow(r_1)$ is a march test that contains three march elements.

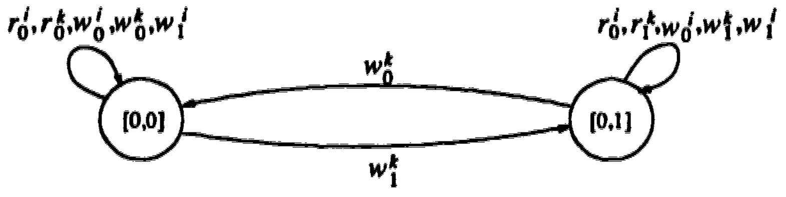

$r_0^i, r_0^k, w_0^i, w_0^k, w_1^i$
$r_0^i, r_1^k, w_0^i, w_1^k, w_1^i$
w_0^k
[0,0]
[0,1]
w_1^k

Figure 4.2: The transition diagram of a faulty memory M_s.

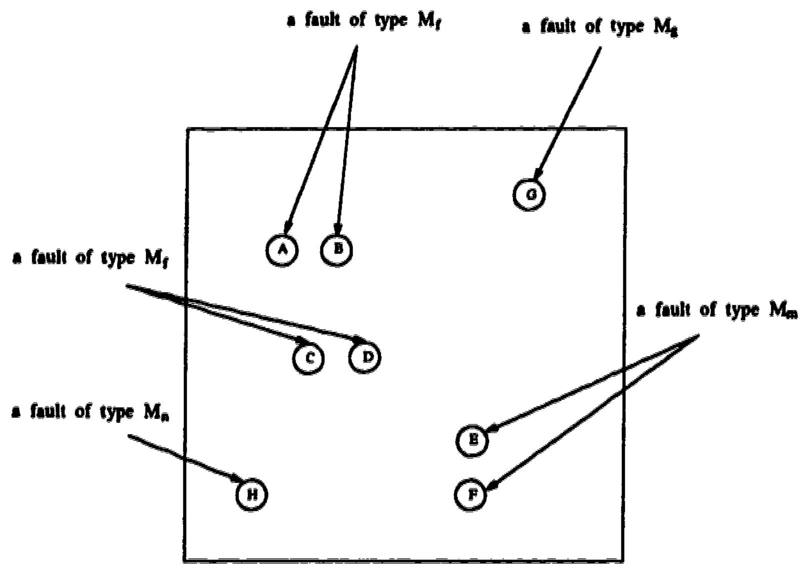


Figure 4.3: A faulty memory

Chapter 5

A Realistic DRAM Fault Model

Many of the fault models that have appeared in the literature appear to have been proposed without analyzing the manufacturing defects that actually occur in RAM chips [33]. As a result, those fault models may not be able to accurately represent the failure mechanisms that occur in real RAMs. With the increasing density of RAM chips and the use of new manufacturing technologies, new types of defects can be expected to occur in RAM chips. Traditional fault models (stuck-at, transition, coupling, and pattern sensitive fault models [33]) are inadequate for describing the faulty behaviours of some defects. Therefore, new fault models and new methodologies for developing accurate fault models are needed.

Defect-oriented analysis is a recently proposed way of developing realistic fault models [9, 16, 28]. Usually, systematic manual or automated procedures are used to predict and analyze all possible defects likely to occur in a RAM chip, and then the behaviour of each defect is modelled as a functional fault. A fault model is obtained by including all of the functional faults that are derived from the defects. Fault models developed in this way have been shown to be realistic and hence more accurate than traditional ones [21]. Two major methods of defect-oriented analysis are inductive fault analysis (IFA), in which the defects are predicted from simulation, and physical defect analysis where the defects are determined from actual failed devices.

Oberle *et al.* used IFA to develop a fault model that they found to be an accurate description of the faulty behaviours observed in real DRAMs [21]. The cell arrangement in this particular DRAM was depicted in Fig. 2.2. In [21], the faulty behaviours observed in the DRAMs were described informally and are not suitable for rigorous analysis. For example, it is difficult to derive lower bounds and prove fault coverage. In this chapter, by using the mathematical methodology introduced in Chapter 4, we introduce a slightly generalized formal fault model for the same DRAM.

Eleven physical defects were reported by Oberle *et al.* [21] to be the most frequently occurring defects in a Siemens 4M $\times 1$ DRAM. The defects fall into two categories: those that create stuck bits and those that create more complex faults. The defects are listed in Table 5.1.

In the following sections, we model the functional behaviours of those defects using the methodology introduced in Chapter 4. We assume that refresh operations may be triggered to any particular cell(s) whenever any other cell is accessed. In

Table 5.1: Categories of Defects

Defects resulting in stuck bits:

- a short from a word line to a cell capacitor
- a short from a bit line to a cell capacitor
- interrupted word lines
- no connection between a bit line and a cell
- no connection between a word line and a transfer transistor
- excessive leakage current across a cell capacitor

Defects resulting in more complex behaviour:

- a short between adjacent word lines
- a short between adjacent bit lines
- a short between a word line and a bit line
- the capacitors of two adjacent cells are shorted together
- interrupted bit lines

the case of a faulty DRAM, it is possible for refresh operations to induce new errors or to mask existing errors. Therefore, we have to consider the effects of refresh operations when modelling the functional behaviours of the defects.

5.1 Stuck Bits

Stuck bits, including single and multiple stuck bits, can be modelled formally as single and multiple stuck-at faults, respectively. A ***single stuck-at-b fault affecting cell i***, where $b \in \{0,1\}$ and $0 \leq i \leq n-1$, is defined to be a deterministic Mealy automaton of the form $M_{ib} = (Q_{ib}, X, Y, \delta_{ib}, \lambda)$, where $Q_{ib} = \{q \in \{0,1\}^n \mid [q]_i = b\}$ and δ_{ib} is identical to δ_0 on the restricted domain except for erroneous transitions of the form:

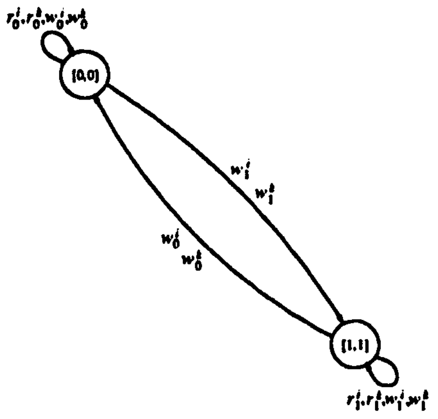
$$\delta_{ib}(w^i_{\bar{b}}, [c^0, \ldots, c^{i-1}, b, c^{i+1}, \ldots, c^{n-1}]) = [c^0, \ldots, c^{i-1}, b, c^{i+1}, \ldots, c^{n-1}].$$

We define a *multiple stuck-at fault* to be recursively the composition of a single stuck-at fault and a single or multiple stuck-at fault. Let M'', M', and M_{ib} denote a multiple stuck-at fault, a single or multiple stuck-at fault, and a single stuck-at fault affecting cell i, respectively. Assume that $M' = (Q', X, Y, \delta', \lambda)$. Then M'' is defined as $M'' = M' \circ M_{ib}$, provided $Q' \cap Q_{ib} \neq \{\}$, such that $Q'' = Q' \cap Q_{ib}$ and δ'' is derived from δ' on the restricted domain such that if $\delta'(g, q) = [c^0, \ldots, c^{i-1}, c^i, c^{i+1}, \ldots, c^{n-1}]$, then $\delta''(g, q) = [c^0, \ldots, c^{i-1}, b, c^{i+1}, \ldots, c^{n-1}]$, for all $g \in X$ and $q \in Q''$.

5.2 Interconnected Cells

A defect in the mask for field oxidation may cause the absence of lateral isolation between two or more cells, in which case those cells are erroneously interconnected. In practice, only physically adjacent cells are likely to be interconnected. However, to know which cells are physically adjacent, we need detailed knowledge of the chip layout. In this generalized DRAM fault model, we assume that such detailed knowledge of the chip layout is not available. Therefore, we generalize and permit any two cells in the DRAM to be shorted together.

The effect of a pair of interconnected cells is that, when one of the two inter-



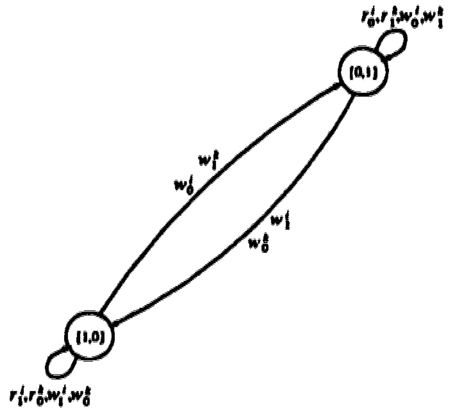
Interconnected cells, same encoding.

Figure 5.1: Transition diagram for a fault M_1 of type T_1 $(n = 2)$.

connected cells is addressed for a write operation, the same physical voltage level, either high (H) or low (L), is written into both cells.

Note that if a H is stored in a DRAM cell, it does not necessarily mean that a logic 1 is stored. The same situation is true for a L. The corresponding logic value of a H (L) in a particular cell is determined by the encoding scheme that the cell is using. In the Siemens DRAM, two encoding schemes are used in the same device, namely positive and negative encodings. The cells whose bit lines are connected to the "+" ("-") terminals of the sense-amplifiers use positive (negative) encoding. By *positive encoding*, we mean that a physical H represents a logic 1 and a physical L represents a logic 0. By *negative encoding*, we mean that a H represents a 0 and a L represents a 1.

Let cells i and k denote the addresses of two interconnected cells, where $i < k$. If the two cells use the same encoding scheme, then writing a logic value to one



Interconnected cells, different encoding.

Figure 5.2: Transition diagram for a fault M_2 of type T_2 ($n = 2$).

cell will cause the same logic value to be written into the other cell; otherwise, if the two cells are using different encoding schemes, then writing a logic value to one cell will cause the opposite logic value to be written into the other cell. Therefore, two possible types of faults, namely T_1 and T_2, can be induced by the same kind of physical defect. They can be formally defined as follows:

Type T_1. (cells i and k use the same encoding) The behaviour of the fault, in which only cells i and k are involved, is given by a deterministic Mealy automaton of the form $M_1 = (Q_1, X, Y, \delta_1, \lambda)$, where $Q_1 = \{q \in \{0,1\}^n \mid [q]_i = [q]_k\}$ and all transitions in δ_1 are identical to the corresponding transitions in δ_0 except for erroneous transitions of the form:

$$\delta_1(x, [c^0, \ldots, c^{i-1}, \bar{b}, c^{i+1}, \ldots, c^{k-1}, \bar{b}, c^{k+1}, \ldots, c^{n-1}]) =$$
$$[c^0, \ldots, c^{i-1}, b, c^{i+1}, \ldots, c^{k-1}, b, c^{k+1}, \ldots, c^{n-1}],$$

where $x \in \{w_b^i, w_b^k\}$ and $b \in \{0,1\}$. Figure 5.1 gives the transition diagram for M_1. Note that states $[0,1]$ and $[1,0]$ have been omitted from the diagram because once the memory is powered up, the two cells must contain the same value. According to the transition diagram, any read operations cannot cause states [0,0] and [1,1] to make further transitions. Therefore, both states are stable states.

Type T_2. (cells i and k use different encodings) The behaviour of the fault, in which only cells i and k are involved, is given by a deterministic Mealy automaton of the form $M_2 = (Q_2, X, Y, \delta_2, \lambda)$, where $Q_2 = \{q \in \{0,1\}^n \mid [q]_i \neq [q]_k\}$ and all transitions in δ_2 are identical to the corresponding transitions in δ_0 except for erroneous transitions of the form:

$$\delta_2(w_b^i, [c^0, \ldots, c^{i-1}, \bar{b}, c^{i+1}, \ldots, c^{k-1}, b, c^{k+1}, \ldots, c^{n-1}]) =$$
$$[c^0, \ldots, c^{i-1}, b, c^{i+1}, \ldots, c^{k-1}, \bar{b}, c^{k+1}, \ldots, c^{n-1}],$$

where $x \in \{w_b^i, w_{\bar{b}}^k\}$ and $b \in \{0,1\}$. Figure 5.2 gives the transition diagram for M_2. Note that states $[0,0]$ and $[1,1]$ have been omitted because once the memory is powered up, the two cells must contain different values. According to the transition diagram, any read operations will not cause states [1,0] and [0,1] to make further transitions. Therefore, these are stable states.

5.3 Shorts Between Word Lines

Additional or missing metal can cause two word lines to be shorted together. In practice, only physically adjacent word lines are likely to be shorted together. The effect of this physical defect is that writing to a cell along one of the word lines will cause that one cell and another cell along the other affected word line to be written. Let cells i and k denote two cells that are connected to the two shorted word lines, respectively. In the Siemens DRAM cell array layout shown in Fig. 2.2, we note that the cells that are along two adjacent word lines can either use the same encoding, like the cells along word lines $WL0$ and $WL3$, or use different encodings, like the cells along word lines $WL0$ and $WL1$.

It is easier for L signals to pass through the n-channel of MOS transistors than for H signals [35]. It is therefore possible for the resistance of the short between the two word lines to be high enough to stop the transfer of H signals through the transistor (but not affect the transfer of L signals), so that writing a H signal to a cell along one word line will not affect the second cell along the other word line.

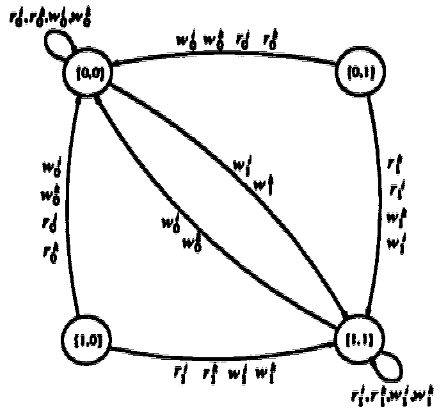
Considering all possible combinations of encoding scheme and short resistance, there are five types of faults, namely T_3, T_4, T_5, T_6, and T_7, that can be created by the physical defect. The transition diagrams in Figs. 5.3-5.7 define the functional behaviours of faults of types $T_3, ..., T_7$ by only considering how c^i and c^k respond to operations addressed to cells i and k. However, a real fault that is induced by a short between word lines defect contains all of the cells along the two shorted word lines. For example, in the Siemens DRAM, 2048 cells are affected by a word line to word line short. Therefore, we must represent a single inter-word line short defect as a multiple fault that is composed of a certain number of single faults. For example, a low resistance inter-word line short is composed of 1024 instances of faults of type T_3. These multiple faults operate together in parallel so that if one cell in one component is addressed for a read or write operation, all of the other faults receive read operations (to simulate the automatic write-back mechanism).

The behaviours of the five types of faults can be specified formally as follows:

Type T_3.(low short resistance; cells use the same or different encodings) Since either H or L signals can be easily transferred as a result of a low resistance short, whenever a physical level is written to either cell, the corresponding physical level will be written to the other cell. If both cells i and k use the same encoding, then writing a $H(L)$ to cell i (k) will cause a $H(L)$ to be written to cell k (i). That is, writing a logic value b, where $b \in \{0, 1\}$, to cell i (k), causes a b to be written to cell k (i). If cells i and k use different encodings, then writing a $H(L)$ to cell i (k) will cause a $L(H)$ to be written to cell k (i). Note that since different encodings are being used by cells i and k, writing a logic value b, where $b \in \{0, 1\}$, to cell i (k), will still cause a b to be written to cell k (i).

As before, a fault M_3 of type T_3 can be formally defined as a finite nondeterministic automata. However, since the functional behaviour of a fault is specified adequately using a transition diagram only, we omit formal definitions in terms of alphabets and transition functions. The transition diagram of M_3 is depicted in Fig. 5.3.

It is possible, as shown in the figure, for c^i and c^k to have different values after the memory is powered up. However, once any operation is applied to cells i or k, the values stored in cells i and k must become the same and will never be different during the remaining period that the memory is on. Therefore, after the first operation to cells i or k, M_3 and M_1 have the same functional


$r_0^i, r_0^k, w_0^i, w_0^k$
$w_0^i\ w_0^k\ r_0^i\ r_0^k$
[0,0]
[0,1]
w_0^i w_0^k r_0^i r_0^k
w_1^i w_1^k
w_0^i w_0^k
r_1^k r_1^i w_1^k w_1^i
[1,0]
[1,1]
$r_1^i\ r_1^k\ w_1^i\ w_1^k$
$r_1^i, r_1^k, w_1^i, w_1^k$

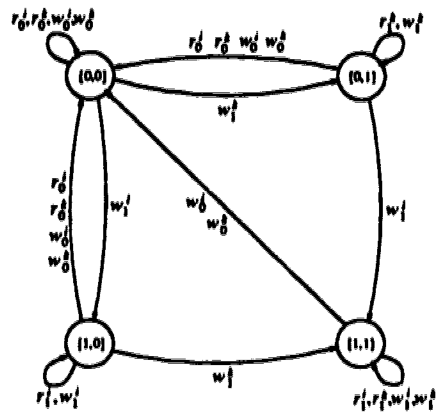
Low resistance inter-word line short.

All possible encodings for cells i and k.

Figure 5.3: Transition diagram for a fault M_3 of type T_3 ($n = 2$).

behaviours. According to the transition diagram, any refresh operations will not cause states [0,0] and [1,1] to make further transitions. Therefore, these are stable states. Note, however, that states [1,0] and [0,1] are unstable. For example, a read operation to cell i will cause state [0,1] to change to state [0,0]. Note that a fault M_3 of type T_3 is actually a nondeterministic Mealy automata due to nondeterministic transitions exiting states 01 and 10; these transitions reflect the uncertainty as to how the sense-amplifiers will interpret slightly corrupted bit line signals.

Type T_4.(high short resistance; both cells use positive encoding) When a L is written to either cell, the other cell will be loaded with a L too. However, since H signals can not pass through the weakly activated channel of a transistor well, when a H is written to one of the two cells, the other one will not be


$r_0^i, r_0^k, w_0^i, w_0^k$
$r_0^i \; r_0^k \; w_0^i \; w_0^k$
r_1^k, w_1^k
[0,0]
[0,1]
w_1^k
r_0^i r_0^k w_0^i w_0^k
w_1^i
w_0^i w_0^k
w_1^i
[1,0]
[1,1]
w_1^k
r_1^i, w_1^i
$r_1^i, r_1^k, w_1^i, w_1^k$

High resistance inter-word line short.

Encoding for cell i: $0 \leftrightarrow L$ and $1 \leftrightarrow H$;

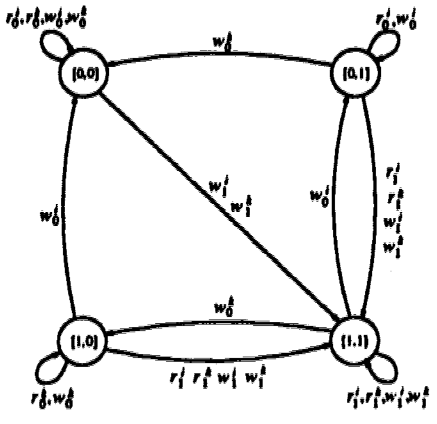
Encoding for cell k: $0 \leftrightarrow L$ and $1 \leftrightarrow H$.

Figure 5.4: Transition diagram for a fault M_4 of type T_4 ($n = 2$).

affected. Note that positive encoding is used by both cells. The behaviour of M_4 is depicted in Fig. 5.4. States [0,0] and [1,1] are stable states. Reading cell k will change the state from [1,0] to [0,0] and reading cell i will change state [0,1] to [0,0]. Thus states [1,0] and [0,1] are unstable. Note that a fault M_4 of type T_4 is a nondeterministic Mealy automaton.

Type T_5.(high short resistance; both cells use negative encoding) Figure 5.5 depicts the transition diagram of a fault M_5 of type T_5. It is interesting to note that by interchanging 0's and 1's in Fig. 5.4, we can obtain Fig. 5.5. Once again states [0,0] and [1,1] are stable and states [1,0] and [0,1] are unstable. Note that a fault M_5 of type T_5 is a nondeterministic Mealy automaton.

Type T_6.(high short resistance; different encodings, cell i uses positive encoding



High resistance inter-word line short.

Encoding for cell i: $0 \leftrightarrow H$ and $1 \leftrightarrow L$;

Encoding for cell k: $0 \leftrightarrow H$ and $1 \leftrightarrow L$.

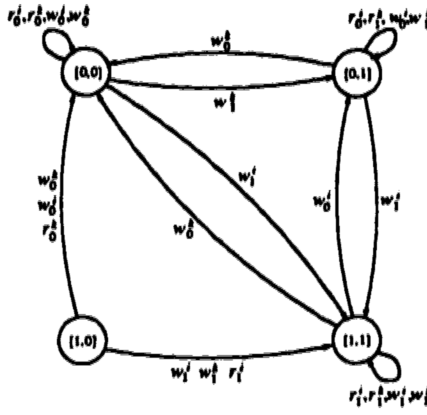
Figure 5.5: Transition diagram for a fault M_5 of type T_5 ($n = 2$).

and cell k uses negative encoding)

When either cell is loaded with a H, the other cell is simultaneously loaded with a L; however, when one of the cells is loaded with a L, the other cell is not affected. The behaviour of a fault M_6 of type T_6 is depicted in Fig. 5.6. All of the possible states, except [1,0], are stable. However, after any first operation is applied to either cell, state [1,0] will never be re-entered.

Type T_7.(high short resistance; different encodings, cell i uses negative encoding and cell k uses positive encoding)

Figure 5.7 depicts the transition diagram of a fault M_7 of type T_7. It is interesting to note that by interchanging 0's and 1's in Fig. 5.6, we can obtain Fig. 5.7. All of the possible states, except [0,1], are stable. However, after any



High resistance inter-word line short.

Encoding for cell i: $0 \leftrightarrow L$ and $1 \leftrightarrow H$;

Encoding for cell k: $0 \leftrightarrow H$ and $1 \leftrightarrow L$.

Figure 5.6: Transition diagram for a fault M_6 of type T_6 ($n = 2$).

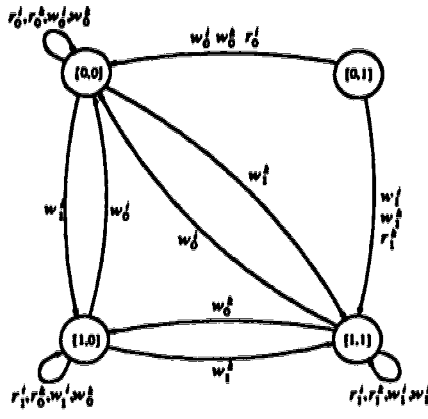
first operation is applied to the memory, state [0,1] will never be re-entered.

5.4 Shorts Between Bit Lines

Due to a structuring defect similar to the one that induces shorts between word lines, two bit lines can be shorted together. In practice, only adjacent bit lines are likely to be shorted together. The two cells at the intersection of a word line and the two bit lines are affected by faulty behaviour. Referring back to the cell array layout in Fig. 2.2, we have two types of shorts to consider:

Case 1. (The shorted bit lines are connected to the same sense-amplifier.)

Assume that cell i is connected to one of the two shorted bit lines. During a write operation to cell i, either a full-strength or a reduced-strength voltage



High resistance inter-word line short.

Encoding for cell i: $0 \leftrightarrow H$ and $1 \leftrightarrow L$;

Encoding for cell k: $0 \leftrightarrow L$ and $1 \leftrightarrow H$.

Figure 5.7: Transition diagram for a fault M_7 of type T_7 ($n = 2$).

signal will be written to cell i depending on the relative positions of cell i, the short, and the sense-amplifier. During a read operation, the short will further reduce the already weak voltage signal stored in cell i. As a result, the output of a read operation to cell i is likely to produce a weak voltage signal. Therefore, this fault will be classified as untestable because it is not reliably detected by a boolean test.

Case 2. (The shorted bit lines are connected to different sense-amplifiers.)

Depending on the stored and written data, one of two situations can occur: either the data for the two cells will correspond to opposite physical levels, or the data will correspond to the same physical level. In the first sub-case, as in Case 1, weak signals may result, so the fault is not testable. In the second sub-case, no faulty behaviour is observed, so again the fault is untestable.

As a consequence of Case 1 and 2, the fault that is induced by the shorts between bit lines is unclean. Therefore, this fault will not be included in the fault model.

5.5 Shorts Between Word and Bit Lines

Due to a structuring defect, it is possible for a word line and a bit line to be shorted together. Let BL and $\overline{BL}$ denote a pair of bit lines that are connected to the same sense-amplifier. Under this defect, during a read operation that is applied to any cell along BL or $\overline{BL}$, the enable or disable signal on the shorted word line will be transferred to BL or $\overline{BL}$. Since the enable or disable signal is much stronger than the voltage stored in the cell to be read, the latter one will be overridden. The resulting behaviours of those cells along BL and $\overline{BL}$ are the same as for stuck-at faults. Depending on whether the bit line is connected to "+" or "-" terminals, the stuck-at bit can be either stuck-at-1 or stuck-at-0 bit. The fault that is induced by the physical defect is therefore equivalent to a multiple stuck-at fault affecting all cells on the shorted bit line.

5.6 Interrupted Bit Lines

If a bit line contains a break somewhere between the sense-amplifier and the precharge circuitry, then one bit line of a bit line pair can no longer be precharged to midlevel M. That means that the voltage level from the last write or refresh operation remains on the bit line due to the significant capacitance of the bit line. As a result, the next read operation to any cell along the bit line pair will sense this voltage level instead of the voltage level stored in the cell. The functional behaviour of this fault is that following the first operation, all of the cells that are along the bit line pair appear to be connected together logically, much like multiple interconnected cells with the same encoding. This fault can be formally defined as follows: Let $i_1, \ldots, i_\sigma$ denote the addresses of all the cells that are connected to the two affected bit lines. We define a *single interrupted bit line fault involving cells* $i_1, \ldots, i_\sigma$ to be a fault of the form $M_8 = (Q, X, Y, \delta_8, \lambda_8)$. Erroneous transitions due to write operations have the form $\delta_8(w_b^j, q) = q'$, for all $q \in Q$, where $[q']_j = b$ if $j \in \{i_1, \ldots, i_\sigma\}$, and $[q']_j = [q]_j$ otherwise. (Note that some nonerroneous transitions will also have this form.) Erroneous transitions and erroneous outputs are produced nondeterministically from some initial states; the nondeterminism models the random signal that the sense-amplifier detects initially on the affected bit line pair. Define $Q_8 = \{q \in \{0,1\}^n \mid [q]_{i_1} = \cdots = [q]_{i_\sigma}\}$ to be the set of states in which all cells on the two affected bit lines, namely cells $i_1, \cdots, i_\sigma$, contain the same logical value. For all states $q \in (Q - Q_8)$, $\delta_8(r^j, q) \in \{q_1, q_2\}$, where $q_1, q_2 \in Q_8$ such that $[q_1]_j = b$ and $[q_2]_j = \bar{b}$ if $j \in \{i_1, \ldots, i_\sigma\}$, and $[q_1]_j = [q_2]_j = [q]_j$ otherwise. The corresponding erroneous outputs are $\lambda_8(r^j, q) = b$ when $\delta_8(r^j, q) = q_1$, and $\lambda_8(r^j, q) = \bar{b}$ when $\delta_8(r^j, q) = q_2$. We define fault type T_8 to be the set of all faults of the same form as M_8.

5.7 Fault Model Overview

In this chapter we described a realistic DRAM fault model that assumes that a finite number of unlinked instances of the following faults can occur simultaneously in the memory:

Stuck Bits: A single or multiple stuck-at fault.

Interconnected Cells: A fault of type T_1 or T_2.

Inter-Word Line Short: A multiple fault equal to the appropriate composition of m faults of the same type $T \in \{T_3, \ldots, T_7\}$, where m is the number of cells that are along one word line (in the Siemens DRAM, $m = 1024$).

Word Line to Bit Line Short: A multiple stuck-at fault.

Interrupted Bit Line: A fault of type T_8.

This new fault model differs substantially from traditional fault models that contained stuck-at faults, coupling faults, transition faults, and pattern sensitive faults.

Chapter 6

Fault Detection

Fault detection, faulty cell location, fault type location and fault diagnosis form a hierarchy of increasingly difficult memory testing problems. Fault detection is the easiest problem in the hierarchy while fault diagnosis is the most difficult problem. The more difficult the problem, the more information that must be obtained about the faults in the memory (at the cost of increased test length). If memory repair and manufacturing-process analysis are not of concern, then fault detection probably is the appropriate test level to use.

6.1 Lower Bound Results

In algorithm design, lower bound derivations are often an effective way of gaining deeper insight into the problem being studied. A tight lower bound is also a good criterion for judging the efficiency of an algorithm. In this section, we derive two lower bounds on the length of any march test and any test of unrestricted structure that accomplish fault detection for the DRAM fault model, respectively. The effect of refresh operations must of course be considered.

Proposition 6.1 *If the first operation to a cell i is a read operation, then that operation can be deleted with no loss in detection.*

Proof: After the memory is powered up, cell i can contain either a 1 or a 0. Note that the tester cannot adapt the test by storing any logic value read earlier on from the memory under test and then comparing it with values read later. Therefore, the first read operation does not help to detect any fault in the memory and can thus be safely deleted. □

Corollary 6.1 *Proposition 6.1 implies that in our lower bound derivation the first operation to each cell can be assumed to be a write operation.*

Proposition 6.2 *If a test* **t** *detects all possible stuck-at faults, then* **t** *must contain at least one of the subsequences $r_0^i w_1^i r_1^i$ or $r_1^i w_0^i r_0^i$, for all n addresses i.*

Proof: To detect a stuck-at-1 fault at cell i, a r_0^i must be included in **t**. Similarly, a r_1^i must be included in **t** in order to detect a stuck-at-0 fault at cell i. Therefore, to detect both stuck-at faults at cell i, **t** must contain one of the subsequences $r_0^i r_1^i$ or $r_1^i r_0^i$. According to the definition of a good memory, at least one write operation must be inserted between the two required read operations to change the content of cell i. Thus the proposition is proved. □

Let a^i denote the number of operations applied to cell i in $\mathbf{t}$. Then we have the following proposition.

Proposition 6.3 *If a test* $\mathbf{t}$ *detects all single stuck-at faults, then* $\mathbf{t}$ *must contain all four of* w_0^i, w_1^i, r_0^i *and* r_1^i *and hence* $a^i \geq 4$, *for* $0 \leq i \leq n-1$.

Proof: Follows from Corollary 6.1 and Proposition 6.2. □

According to Proposition 6.3, we have obtained a lower bound of $4n$. However, since we have only considered stuck-at faults, it may be possible to increase the lower bound by including some other faults. In the following, we improve the lower bound by focusing on the increased difficulty for a test to also detect single faults M_6 and M_7 of types T_6 and T_7, respectively. Except for state [1,0] in M_6 and [0,1] in M_7, all of the other states in the transition diagrams of M_6 and M_7 are stable states. Note that [1,0] of M_6 and [0,1] of M_7 cannot be re-entered after the first operation has been applied to either cells i or k. Thus no subsequent refresh operations can change the state of M_6 and M_7. Therefore, we need not consider the effect of refresh operations when deriving the lower bound.

6.1.1 A Lower Bound for March Tests

Proposition 6.4 *If there exist two cells* i *and* k *such that* $a^i = a^k = 4$ *in a given march test* $\mathbf{t}$, *then* $\mathbf{t}$ *cannot detect separate single instances of* M_6 *and* M_7 *involving only cells* i *and* k.

Proof: Assume that there exists a test **t** that detects single instances of faults M_6 and M_7, and that $a^i = a^k = 4$. According to Proposition 6.3, if **t** is to detect stuck-at faults, then the four operations w_0^i, w_1^i, w_0^k, and w_1^k must all be included in **t**. Consider the subsequence **s** of **t** containing all the write operations to cells i and k. Without loss of generality, assume that the first write operation to cell i occurs before the first write operation to cell k. Given that **t** is a march test, we have the following six cases to consider:

Case 1. ($\mathbf{s} = w_0^i w_1^i w_0^k w_1^k$)

Refer to the transition diagrams of M_0 and M_7 in Figs. 4.1 and 5.7. After the w_0^i is applied, M_7 must be in state [0,0]; M_0, however, could be in either [0,0] or [0,1]. After the w_1^i is applied, M_7 must be in state [1,0]; M_0, however, could be in either [1,0] or [1,1]. The w_0^k causes both M_7 and M_0 to enter state [1,0]. Finally, the w_1^k causes both M_7 and M_0 to enter state [1,1]. An error is never created in M_7, so the test cannot detect the fault.

Case 2. ($\mathbf{s} = w_0^i w_0^k w_1^i w_1^k$)

Similar to Case 1.

Case 3. ($\mathbf{s} = w_0^i w_0^k w_1^k w_1^i$)

Refer to the transition diagrams of M_0 and M_6 in Figs. 4.1 and 5.6. After the w_0^i is applied, both M_6 and M_0 can be in states [0,0] or [0,1]. After the w_0^k is applied, both M_6 and M_0 must be in state [0,0]. The w_1^k causes both M_6 and M_0 to enter state [0,1]. Finally, the w_1^i causes both M_6 and M_0 to enter state [1,1]. An error is never created in M_6, so the test cannot detect the fault.

Case 4. ($s = w_1^i w_0^i w_1^k w_0^k$)

Refer to the transition diagrams of M_0 and M_6 in Figs. 4.1 and 5.6. After the w_1^i is applied, M_6 must be in state [1,1]; M_0, however, could be in either [1,0] or [1,1]. After the w_0^i is applied, M_6 must be in state [0,1]; M_0, however, could be in either [0,0] or [0,1]. The w_1^k causes both M_6 and M_0 to enter state [0,1]. Finally, the w_0^k causes both M_6 and M_0 to enter state [0,0]. An error is never created in M_6, so the test cannot detect the fault.

Case 5. ($s = w_1^i w_1^k w_0^i w_0^k$)

Similar to Case 4.

Case 6. ($s = w_1^i w_1^k w_0^k w_0^i$)

Refer to the transition diagrams of M_0 and M_7 in Figs. 4.1 and 5.7. After the w_1^i is applied, both M_7 and M_0 can be in states [1,0] or [1,1]. After the w_1^k is applied, both M_7 and M_0 must be in state [1,1]. The w_0^k causes both M_7 and M_0 to enter state [1,0]. Finally, the w_0^i causes both M_7 and M_0 to enter state [0,0]. An error is never created in M_7, so the test cannot detect the fault.

All of the above six cases lead to contradictions with the assumption. Therefore, the proposition must be true. □

Proposition 6.5 *If, for all interacting pairs of cells on two adjacent word lines WLi and WLj, a test* t *does not detect the corresponding instance of the same one fault type $T \in \{T_3, \ldots, T_7\}$, then* t *does not detect all possible shorts between WLi*

and WLj.

Proof: As described earlier, the short between two word lines can be defined as the composition of faults of one particular type $T \in \{T_3, ..., T_7\}$. Note that any errors only affect cells on WLi and WLj. We can conclude that if no errors are detected on any cells on WLi and WLj for a particular fault type T, then at least one type of short between word lines WLi and WLj cannot be detected. □

From the above propositions, we now obtain the lower bound for march tests.

Theorem 6.1 *Any march test* **t** *that detects the fault model has a length of at least* $5n$.

Proof: According to Proposition 6.3, if **t** detects all stuck-at faults in the fault model, the length of **t** must be at least $4n$. However, Propositions 6.4 and 6.5 further imply that **t** cannot detect all possible shorts between word lines if the test only has a length of $4n$. Hence the theorem is proved. □

6.1.2 A Lower Bound for Unrestricted Tests

Refer back to the proof of Proposition 6.4. If we do not restrict our consideration to march tests when considering the subsequence **s** of **t** containing all of the write operations to cells i and k that are required for detecting stuck-at faults, then there are six more cases to consider, namely, 1) $w_0^i w_1^i w_1^k w_0^k$, 2) $w_0^i w_1^k w_1^i w_0^k$, 3) $w_0^i w_1^k w_0^k w_1^i$, 4) $w_1^i w_0^i w_0^k w_1^k$, 5) $w_1^i w_0^k w_0^i w_1^k$, and 6) $w_1^i w_0^k w_1^k w_0^i$.

Proposition 6.6 *Cases 2), 3), 5), and 6) can detect separate single instances of both M_6 and M_7; cases 1) and 4), however, cannot.*

Proof:

Case 1. ($w_0^i w_1^i w_1^k w_0^k$)

Refer to the transition diagrams of M_0 and M_7 in Figs. 4.1 and 5.7. After the w_0^i is applied, M_7 must be in state [0,0]; M_0, however, could be in either [0,0] or [0,1]. After the w_1^i is applied, M_7 must be state [1,0]; M_0, however, could be in either [1,0] or [1,1]. The w_1^k causes both M_7 and M_0 to enter state [1,1]. Finally, the w_0^k causes both M_7 and M_0 to enter state [1,0]. An error is never created in M_7, so the test cannot detect the fault.

Case 2. ($w_0^i w_1^k w_1^i w_0^k$)

Refer to the transition diagrams of M_0, M_6, and M_7 in Figs. 4.1, 5.6, and 5.7. After the w_0^i is applied, M_7 must be in state [0,0]; M_6 and M_0, however, could be in either [0,0] or [0,1]. The w_1^k causes M_7 to enter [1,1]; however, causes M_6 and M_0 to enter [0,1]. Note that an error is created in M_7. Thus, M_7 is detected. After w_1^i is applied, both M_6 and M_0 must be in state [1,1]. The w_0^k causes M_6 to enter [0,0]; however, causes M_0 to enter [1,0]. Note that an error is created in M_6. Thus, M_6 is detected.

Case 3. ($w_0^i w_1^k w_0^k w_1^i$)

Similar to Case 2.

Case 4. ($w_1^i w_0^i w_0^k w_1^k$)

Similar to Case 1.

Case 5. $(w_1^i w_0^k w_0^i w_1^k)$

Similar to Case 2.

Case 6. $(w_1^i w_0^k w_1^k w_0^i)$

Similar to Case 2.

□

Proposition 6.6 implies that it is possible for a test to detect M_6 and M_7 involving only cells i and k when $a^i = a^k = 4$. However, as we show in the following proposition, the other cells in the memory must then be accessed at least five times.

Proposition 6.7 *If a test* **t** *detects both M_6 and M_7 for all pairs of cells and $a^i = a^k = 4$ for two particular distinct address i and k, then $a^j \geq 5$ for all $j \in \{0, \ldots, n-1\} - \{i, k\}$.*

Proof: Assume that there exists a test **t** that detects both M_6 and M_7 such that $a^i = a^k = a^j = 4$. Note that an instance of M_6 or M_7 may involve any pair of cells among i, k, and j. Thus, in order to detect both M_6 and M_7 in the fault model, **t** must be able to detect all possible pairs of cells among i, k, and j, *i.e.*, pairs (i, k), (i, j), and (k, j). As specified in Proposition 6.6, there exist four forms of tests that can detect both M_6 and M_7 involving cells i and k such that $a^i = a^k = 4$. Therefore, we have four cases to consider:

Case 1. ($w_0^i w_1^k w_1^i w_0^k$) Again, according to Proposition 6.6, four forms of tests can detect both M_6 and M_7 that involve cells i and j such that $a^i = a^j = 4$. Therefore, we have four subcases to consider:

Subcase 1.1. ($w_0^i w_1^j w_1^i w_0^j$) Any test that detects both M_6 and M_7 involving cells i and k, or i and j must be one of the forms $w_0^i w_1^k w_1^j w_1^i w_0^k w_0^j$, $w_0^i w_1^k w_1^j w_1^i w_0^j w_0^k$, $w_0^i w_1^j w_1^k w_1^i w_0^k w_0^j$, or $w_0^i w_1^j w_1^k w_1^i w_0^j w_0^k$. Note that none of the four forms can detect both M_6 and M_7 involving cells k and j because the first write operations to cells k and j write the same value, and this situation does not appear in any of the four allowed subsequences. This leads to a contradiction.

Subcase 1.2. ($w_0^i w_1^j w_0^j w_1^i$)

Similar to Subcase 1.1.

Subcase 1.3. ($w_1^i w_0^j w_0^i w_1^j$)

Similar to Subcase 1.1.

Subcase 1.4. ($w_1^i w_0^j w_1^j w_0^i$)

Similar to Subcase 1.1.

Case 2. ($w_0^i w_1^k w_0^k w_1^i$)

Similar to Case 1.

Case 3. ($w_1^i w_0^k w_0^i w_1^k$)

Similar to Case 1.

Case 4. ($w_1^i w_0^k w_1^k w_0^i$)

Similar to Case 1.

All of the four cases lead to a contradiction with the initial assumption. Thus, the proposition is proved. □

We now prove the lower bound for unrestricted tests.

Theorem 6.2 *Any test* **t** *that detects the DRAM fault model has a length of at least* $5n - 2$.

Proof: Let i be the address of the cell with smallest value of a^i. Proposition 6.3 implies that $a^i \geq 4$. We therefore have the following two cases to consider:

Case 1. ($a^i = 4$) Let k be the address of the cell which, apart from cell i, has the smallest value of a^k. We have two subcases to consider:

Subcase 1.1. ($a^k = 4$) Proposition 6.7 implies that $a^j \geq 5$ for any third cell j such that $j \in \{0, \cdots, n-1\} - \{i, k\}$. Therefore, $|\mathbf{t}| \geq 4 + 4 + 5(n-2) = 5n - 2$.

Subcase 1.2. ($a^k \geq 5$) Clearly, $a^j \geq 5$ for all cells $j \in \{0, \cdots, n-1\} - \{i, k\}$. Therefore, $|\mathbf{t}| \geq 4 + 5(n-1) = 5n - 1$.

Case 2. ($a^i \geq 5$) Clearly, $a^k \geq 5$ for all cells $k \in \{0, \cdots, n-1\} - \{i\}$. Therefore, $|\mathbf{t}| \geq 5n$.

The lowest lower bound in all possible cases is $5n - 2$. □

	$\Uparrow(w_0)$	$\Uparrow(r_0 w_1)$	$\Uparrow(r_1 w_0)$
Addr	Seq. A	Seq. B	Seq. C
0	w_0	$r_0 w_1$	$r_1 w_0$
1	w_0	$r_0 w_1$	$r_1 w_0$
2	w_0	$r_0 w_1$	$r_1 w_0$
↓	↘	↘	↘
n-1	w_0	$r_0 w_1$	$r_1 w_0$

Figure 6.1: Specification for MARCH5N.

6.2 An Optimal March Test for Fault Detection

A march test, called MARCH5N, that detects all of the faults in the fault model is proposed. As sepcified in Fig. 6.1, MARCH5N consists of three march elements, which we will call $\mathbf{e}_A, \mathbf{e}_B$, and $\mathbf{e}_C$, respectively. All of the states in the transition diagrams for M_1, M_2, M_3, M_6, and M_7 are stable states (except for those states that cannot be re-entered after the first operation has been applied). States [0,0] and [1,1] in both M_4 and M_5 are also stable states. Therefore, we need not consider the effect of refresh operations on those states.

Proposition 6.8 *MARCH5N detects all single and multiple stuck-at faults.*

Proof: For each cell i, $\mathbf{e}_B$ contains a r_0^i operation and $\mathbf{e}_C$ contains a r_1^i operation. Therefore, any stuck bit must be detected in either $\mathbf{e}_B$ (stuck-at-1) or $\mathbf{e}_C$ (stuck-at-0). □

Corollary 6.2 *MARCH5N detects all single shorts between word lines and bit lines.*

Proof: The claim follows because such shorts are equivalent to multiple stuck-at faults. □

In the following proofs, we use i and k to denote two distinct cell addresses. Without loss of generality, we assume that $i < k$.

Proposition 6.9 *MARCH5N detects all single interconnected cell faults of type T_1 and T_2.*

Proof: Let cells i and k be the two cells involved in an arbitrary single interconnected cell fault M. Let M be either a fault M_1 of type T_1, or a fault M_2 of type T_2. We have two cases to consider:

Case 1. ($M = M_1$) Refer to the transition diagrams in Figs. 5.1 and 4.1. After $\mathbf{e}_A$ has been applied, both M_1 and M_0 must be in state [0,0]. The w_1^i in $\mathbf{e}_B$ causes M_1 and M_0 to enter states [1,1] and [1,0], respectively. The following read operation, r_0^k, to cell k will detect the fault.

Case 2. ($M = M_2$) Refer to the transition diagrams in Figs. 5.2 and 4.1. After $\mathbf{e}_A$ has been applied, M_1 and M_0 must be in states [1,0] and [0,0], respectively. Thus next operation to cell i, the r_0^i in $\mathbf{e}_B$, will detect the fault. □

Corollary 6.3 *MARCH5N detects all interrupted bit lines.*

Proof: Follows because an interrupted bit line fault is composed of a certain number of faults of type T_1. □

Proposition 6.10 *MARCH5N detects all single shorts between word lines.*

Proof: As described earlier, shorts between word lines are modelled as the composition of single faults of types $T_3, ..., T_7$. Let cells i and k be the two cells involved in an arbitrary fault $M \in \{M_3, ..., M_7\}$ of type $T \in \{T_3, ..., T_7\}$, respectively. If a test can detect all single faults $M_3, ..., M_7$ of types $T_3, ..., T_7$, then we claim it will detect shorts between word lines. We have five cases to consider:

Case 1. ($M = M_3$) Refer to the transition diagrams in Figs. 5.3 and 4.1. After $\mathbf{e}_A$ has been applied, both M_3 and M_0 will be in state [0,0]. Note also that M_3 is equivalent to M_1 after $\mathbf{e}_A$. Thus M_3 is detected by the test.

Case 2. ($M = M_4$) Refer to the transition diagrams in Figs. 5.4 and 4.1. After $\mathbf{e}_A$ has been applied, both M_4 and M_0 must be in state [0,0]. The r_1^i in $\mathbf{e}_B$ will cause both M_4 and M_0 to enter state [1,0]. However, refresh operations to cells i and k can cause M_4 to make further transitions to states [1,1] or [0,0]. If M_4 was in state [1,1], then the following read operation to cell k, r_0^k, will detect the fault. If M_4 was in state [0,0] or [1,0], the fault will not be detected at this moment. The following read and write operations to cell k will cause M_4 to enter state [0,1]. Note again that refresh operations may cause state [0,1] to change to [0,0] or [1,1]. At this moment, M_0 is in state [1,1]. If M_4 was in states [0,0] or [0,1], the the fault is detected by the following read operation to cell i, r_1^i, in $\mathbf{e}_C$. Otherwise, the fault will not be detected at this time. However, the w_0^i in $\mathbf{e}_C$ will cause M_4 to enter state [0,0], and cause M_0 to enter state [0,1]. Note that [0,0] is a stable state. Therefore, the following read operation to cell k, a r_1^k, will detect the fault.

Case 3. ($M = M_5$) Refer to the transition diagrams in Figs. 5.5 and 4.1. After the w_0^i in $\mathbf{e}_A$ has been applied, both M_5 and M_0 can be in states [0,0] or [0,1]. However, the following refresh operations can cause the state of M_5 to make a further transition to state [1,1]. Thus after the w_0^k in $\mathbf{e}_A$, M_5 will be in states [0,0] or [1,0] while M_0 must be in state [0,0]. Since the state [1,0] of M_5 is not stable, subsequent refresh operations will lead it to states [0,0] or [1,1]. If M_5 is in states [1,0] or [1,1], the following read operation to cell i, r_0^i, in $\mathbf{e}_B$ will detect the fault. Otherwise, if the state of M_5 was in [0,0], then after the write operation to cell i, w_1^i, in $\mathbf{e}_B$ will force M_5 to enter state [1,1], which is a stable state. The state of M_0, however, will be [1,0]. Thus the next read operation, r_0^k, in $\mathbf{e}_B$ will detect the fault.

Case 4. ($M = M_6$) Refer to the transition diagrams in Figs. 5.6 and 4.1. After $\mathbf{e}_A$ has been applied, M_6 must be in states [0,0], [0,1], or [1,1]. Note that all of the three states are stable. Thus by applying a similar argument as that of Case 1, we can prove that the fault is detected.

Case 5. ($M = M_7$) Refer to the transition diagrams in Figs. 5.7 and 4.1. After $\mathbf{e}_A$ has been applied, M_7 must be in states [0,0], [1,0], or [1,1]. Note that all of the three states are stable. Thus by applying a similar argument as that of

Case 2, we can prove that the fault is detected.

From the above argument, we know that all single faults of types $T_3, ..., T_7$ are detected, which means that all single shorts between word lines faults are also detected. □

Theorem 6.3 *MARCH5N is an optimal march test for detecting the DRAM fault model.*

Proof: Follows from Theorem 6.1 and Propositions 6.6-6.8 together with their corollaries. □

MARCH5N is n operations (13%) shorter than the march test described in [21] and the new test detects a more general fault model. The existence of the $5n - 2$ lower bound for unrestricted tests implies that the march test is near optimal for unrestricted tests as well.

Chapter 7

Faulty Cell Location

In order to repair a RAM chip using redundancy, not only must at least one fault be detected (fault detection), it is also necessary to locate all of the cells affected by all of the faults (faulty cell location).

In [22], the faulty cell location problem was investigated for the Siemens DRAM fault model. Two march tests, namely MARCH6N and MARCH9N, were proposed to locate faulty cells in the DRAM fault model as follows:

- MARCH6N: $\Uparrow (w_0) \Uparrow (r_0 w_1) \Downarrow (r_1 w_0) \Downarrow (r_0)$.
- MARCH9N: $\Uparrow (w_0) \Uparrow (r_0 w_1) \Downarrow (r_1 w_0) \Downarrow (r_0 w_1) \Uparrow (r_1 w_0)$.

MARCH6N was claimed to be able to locate 98% of all the faulty cells and MARCH9N was claimed to have 100% fault coverage. However, in [22] the fault coverages of the two march tests were only evaluated with simulation data; a provably optimal test was not provided for the faulty cell location problem for this particular DRAM. In this chapter, we first derive a lower bound of $8n$ on the length of any march test that locates all of the faulty cells in our formalized version of the Siemens DRAM fault model. We then propose an optimal march test of length $8n$ that locates all of the faulty cells in faulty DRAMs.

7.1 Lower Bound Result

We derive the lower bound by focusing on the difficulty of locating simultaneous unlinked instances of arbitrary faults M_6 and M_7 of types T_6 and T_7, respectively. Consider a double fault that includes M_6 and M_7 as components. Let cells i and k, where $i < k$, denote the two faulty cells created by M_6, and let cells i' and k', where $i' < k'$, denote the two faulty cells created by M_7. In the following, we derive a lower bound on the length of any march test that locates all four of cells i, k, i', and k'.

We first investigate the problem of locating each cell separately. Propositions 7.1 to 7.4 describe necessary and sufficient conditions on march tests that locate each of the four cells. Before we describe the propositions, we introduce a notation $\{\Updownarrow(r_b^*)\}^*$, where $b \in \{0,1\}$, which will be used in the following propositions. The notation $\{\Updownarrow(r_b^*)\}^*$ denotes zero or more march elements each of which is composed of zero or more r_b operations to each cell.

Proposition 7.1 *Any march test that locates cell i must contain a segment of the form $\Updownarrow(\mathbf{a}w_1\mathbf{b})\mathbf{u} \Downarrow(r_1\mathbf{c}w_0\mathbf{d})$ or $\Uparrow(\mathbf{a}w_0\mathbf{c}w_1\mathbf{b}) \Updownarrow(r_1\mathbf{d})$, where $\mathbf{a}, \mathbf{c}, \mathbf{d} \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b} \in \{r_1, w_1\}^*$, and $\mathbf{u} \in \{\Updownarrow(r_1^*)\}^*$.*

Proof: Refer to the state diagrams in Figs. 4.1 and 5.6. According to the transition diagram, errors can be caused in cell i by only three operations: (1) a r_0^k when $[c^i, c^k] = [1,0]$, (2) a w_0^k when $[c^i, c^k] = [1,0]$, and (3) a w_0^k when $[c^i, c^k] = [1,1]$. Situations (1) and (2) can only detect the error if the memory happens to power up in $[c^i, c^k] = [1,0]$ because, after the first march element, it is impossible to reach $[1,0]$. Therefore, any test that locates i must at least use erroneous transition (3). To detect the error, and hence locate cell i, a r_1^i must be applied after the w_0^k in (3). After the memory is powered up, the content of cell i is assumed to be unknown. To ensure that $[c^i, c^k]$ is in state $[1,1]$, a w_1^i must be applied before the w_0^k. Note that, because we are restricted to march tests, every cell undergoes the same operations. Therefore, any march test that can locate cell i must apply w_1, w_0 and r_1 to each memory cell. To guarantee that the content of cell i is 1 when the w_0^k is applied, we can either separate the required w_1 and w_0 into two march elements such that the w_1 is in a preceding element, or we can put them in the same ascending march element such that the w_0 precedes the w_1. Thus we have two cases to consider:

Case 1. (the w_1 and the w_0 are in different march elements) The r_1 and the w_0 must be in the same march element. Otherwise, the error in cell i would be over-written and the r_1^i operation would not detect the error. For the same reason, the r_1 must precede the w_0. To preserve generality, for the first march element we allow a finite number of operations (segment **a**) to precede the w_1. However, only r_1 and w_1 operations (segment **b**) can be allowed to follow the w_1; otherwise, the error in cell i would be over-written. The first march element can be either ascending or descending. For the second march element, we allow a finite number of operations (segment **c**) to appear between the r_1

and the w_0 operations, and a finite number of operations (segment **d**) to follow the w_0 operation. However, no operation can precede the r_1: a preceding write operation would overwrite the error, and a preceding r_0 would not recognize the error. Therefore, the second march element must start with a r_1. Note that the w_0 must be applied to cell k before cell i, and cell k has a higher address than cell i. Thus the second march element must be descending. Finally, a sequence **u** comprising a finite number of march elements containing only r_1 operations can be inserted, without loss of fault location, between the two march elements.

Case 2. (the w_0 and the w_1 are in the same march element) The w_0^k must be followed by the r_1^i to excite and then detect the fault. The march element that includes the w_1^i and w_0^k must be an ascending march element; therefore, the r_1 must be in a different march element. To preserve generality, for the first march element we allow a finite number of operations (segment **a**) to precede the w_0, and a finite number of operations (segment **c**) to appear between the w_0 and w_1. However, only r_1 and w_1 operations (segment **b**) can be allowed to follow the w_1; otherwise, the error in error in cell i would be over-written. For the second march element, we allow a finite number of operations (segment **d**) to follow the r_1. However, no operation can precede the r_1 in the march element: a preceding write operation would overwrite the error, and a preceding r_0 would not recognize the error. Therefore the second march element must start with a r_1. Finally, the second march element can be either ascending or descending because, as noted at the start of the proof, operations to cell k cannot affect cell i if $c^i = 0$. Unlike Case 1, the two march elements must be adjacent. □

Propositions 7.2, 7.3, and 7.4 below are proved in the same way as Proposition 7.1.

Proposition 7.2 *Any march test that locates cell k must contain two adjacent march elements of the form* $\Updownarrow(\mathbf{a}w_0\mathbf{b})\mathbf{u}\ \Uparrow(r_0\mathbf{c}w_1\mathbf{d})$ *or* $\Downarrow(\mathbf{a}w_1\mathbf{c}w_0\mathbf{b})\ \Updownarrow(r_0\mathbf{d})$, *where* $\mathbf{a}, \mathbf{c}, \mathbf{d} \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b} \in \{r_0, w_0\}^*$, *and* $\mathbf{u} \in \{\Updownarrow(r_0^*)\}^*$.

Proposition 7.3 *Any march test that locates cell i' must contain two adjacent march elements of the form* $\Updownarrow(\mathbf{a}w_0\mathbf{b})\mathbf{u}\ \Downarrow(r_0\mathbf{c}w_1\mathbf{d})$ *or* $\Uparrow(\mathbf{a}w_1\mathbf{c}w_0\mathbf{b})\ \Updownarrow(r_0\mathbf{d})$, *where* $\mathbf{a}, \mathbf{c}, \mathbf{d} \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b} \in \{r_0, w_0\}^*$, *and* $\mathbf{u} \in \{\Updownarrow(r_0^*)\}^*$.

Proposition 7.4 *Any march test that locates cell k' must contain two adjacent march elements of the form* $\Updownarrow(\mathbf{a}w_1\mathbf{b})\mathbf{u}\ \Uparrow(r_1\mathbf{c}w_0\mathbf{d})$ *or* $\Downarrow(\mathbf{a}w_0\mathbf{c}w_1\mathbf{b})\ \Updownarrow(r_1\mathbf{d})$, *where* $\mathbf{a}, \mathbf{c}, \mathbf{d} \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b} \in \{r_1, w_1\}^*$, *and* $\mathbf{u} \in \{\Updownarrow(r_1^*)\}^*$.

Let S_1, S_2, S_3, and S_4 denote four march elements of the forms $\Updownarrow(\mathbf{a_1}w_1\mathbf{b_1})\mathbf{u_1}\ \Downarrow(r_1\mathbf{c_1}w_0\mathbf{d_1})$, $\Updownarrow(\mathbf{a_2}w_0\mathbf{b_2})\mathbf{u_2}\ \Uparrow(r_0\mathbf{c_2}w_1\mathbf{d_2})$, $\Updownarrow(\mathbf{a_3}w_0\mathbf{b_3})\mathbf{u_3}\ \Downarrow(r_0\mathbf{c_3}w_1\mathbf{d_3})$, and $\Updownarrow(\mathbf{a_4}w_1\mathbf{b_4})\mathbf{u_4}\ \Uparrow(r_1\mathbf{c_4}w_0\mathbf{d_4})$, respectively, where $\mathbf{a_1}, ..., \mathbf{a_4}, \mathbf{c_1}, ..., \mathbf{c_4}, \mathbf{d_1}, ..., \mathbf{d_4} \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b_1}, \mathbf{b_4} \in \{r_1, w_1\}^*$, $\mathbf{b_2}, \mathbf{b_3} \in \{r_0, w_0\}^*$, $\mathbf{u_1}, \mathbf{u_4} \in \{\Updownarrow(r_1^*)\}^*$, and $\mathbf{u_2}, \mathbf{u_3} \in \{\Updownarrow(r_0^*)\}^*$.

Let S_1', S_2', S_3', and S_4' denote four march elements of the forms $\Uparrow(\mathbf{a}_1'w_0\mathbf{c}_1'w_1\mathbf{b}_1')\ \Updownarrow(r_1\mathbf{d}_1')$, $\Downarrow(\mathbf{a}_2'w_1\mathbf{c}_2'w_0\mathbf{b}_2')\ \Updownarrow(r_0\mathbf{d}_2')$, $\Uparrow(\mathbf{a}_3'w_1\mathbf{c}_3'w_0\mathbf{b}_3')\ \Updownarrow(r_0\mathbf{d}_3')$, and $\Downarrow(\mathbf{a}_4'w_0\mathbf{c}_4'w_1\mathbf{b}_4')\ \Updownarrow(r_1\mathbf{d}_4')$, respectively, where $\mathbf{a}_1', ..., \mathbf{a}_4', \mathbf{c}_1', ..., \mathbf{c}_4', \mathbf{d}_1', ..., \mathbf{d}_4' \in \{r_1, r_0, w_1, w_0\}^*$, $\mathbf{b}_1', \mathbf{b}_4' \in \{r_1, w_1\}^*$, and $\mathbf{b}_2', \mathbf{b}_3' \in \{r_0, w_0\}^*$.

Note that there are at least two march elements in every S_i or S'_i (the **u** element can be absent). In the following, we use $\mathbf{e_i}$ to represent an element from S_i or S'_i, and use $\mathbf{e_{ij}}$ to represent the jth march element in $\mathbf{e_i}$, where $1 \leq i \leq 4$, and $1 \leq j \leq 2$. For example, $\mathbf{e_1} = \mathbf{e_{11}e_{12}}$ represents an element from S_1 or S'_1.

Proposition 7.5 *Any march test* **t** *that locates cells* i, k, i', *and* k' *either must have a length of at least* $8n$, *or must have one of the forms (i)* $\Downarrow(w_0w_1) \Downarrow(r_1w_1w_0) \Downarrow(r_0w_1)$; *(ii)* $\Uparrow(w_1w_0) \Uparrow(r_0w_0w_1) \Uparrow(r_1w_0)$; *(iii)* $\Uparrow(w_0w_1) \Uparrow(r_1w_1w_0) \Uparrow(r_0w_1)$; *or (iv)* $\Downarrow(w_1w_0) \Downarrow(r_0w_0w_1) \Downarrow(r_1w_0)$.

Proof: According to Propositions 7.1–7.4, **t** must contain at least one element from each of $S_i \cup S'_i$, where $1 \leq i \leq 4$. Every element has a length of at least $3n$, so one might expect a trivial lower bound of $12n$ on the length of **t**. However, as we will see later, adjacent march elements may sometimes be overlapped such that the resulting march element contains all of the original march elements and, at the same time, has a length that is shorter than the sum of the lengths of all the original march elements. In order to derive a correct lower bound, we need to consider all $2^4 = 16$ possible combinations of the four elements $(\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}, \mathbf{e_4})$ from $S_i \cup S'_i$, where $1 \leq i \leq 4$:

Case 1. $(\mathbf{e_1} \in S_1, \mathbf{e_2} \in S_2, \mathbf{e_3} \in S_3, \mathbf{e_4} \in S_4)$ Note that $\mathbf{e_{12}}$ and $\mathbf{e_{32}}$ are descending march elements, while $\mathbf{e_{22}}$ and $\mathbf{e_{42}}$ are ascending march elements. Therefore, the only possible mergings between those march elements are $\mathbf{e_{12}}$ with $\mathbf{e_{32}}$, and $\mathbf{e_{22}}$ with $\mathbf{e_{42}}$. However, since $\mathbf{e_{12}}$ must start with r_1 while $\mathbf{e_{32}}$ must start with r_0, we cannot combine $\mathbf{e_{12}}$ with $\mathbf{e_{32}}$ such that the resulting march element contains both $\mathbf{e}_{12}$ and $\mathbf{e}_{32}$. Similarly, we cannot combine $\mathbf{e}_{22}$ with $\mathbf{e}_{42}$. Thus $\mathbf{t}$ must contain at least four march elements $\mathbf{e}_{12}$, $\mathbf{e}_{22}$, $\mathbf{e}_{32}$, and $\mathbf{e}_{42}$, each of which has a length of at least $2n$. The length of $\mathbf{t}$ is therefore at least $8n$.

Case 2. ($\mathbf{e}_1 \in S_1'$,$\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4'$) Similar to Case 1, considering march elements $\mathbf{e}_{11}$, $\mathbf{e}_{21}$, $\mathbf{e}_{31}$, and $\mathbf{e}_{41}$.

Case 3. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3$, $\mathbf{e}_4 \in S_4'$) Applying an argument similar to that used in Case 1, we can prove that $\mathbf{e}_{12}$, $\mathbf{e}_{22}$, and $\mathbf{e}_{32}$ must stay in different march elements. Since $\mathbf{e}_{41}$ is descending and $\mathbf{e}_{22}$ is ascending, they also must stay in different march elements. However, it may still be possible to combine $\mathbf{e}_{41}$ with $\mathbf{e}_{12}$, or $\mathbf{e}_{41}$ with $\mathbf{e}_{32}$. Therefore, we have three possibilities to consider:

Subcase 1. ($\mathbf{e}_{41}$ is not combined with $\mathbf{e}_{12}$ or $\mathbf{e}_{32}$) Test $\mathbf{t}$ will contain at least four march elements $\mathbf{e}_{12}$, $\mathbf{e}_{22}$, $\mathbf{e}_{32}$, and $\mathbf{e}_{41}$, each of which has a length of at least $2n$. Thus $\mathbf{t}$ has a length of at least $8n$.

Subcase 2. ($\mathbf{e}_{41}$ is combined with $\mathbf{e}_{12}$) If the combination, called $\mathbf{s}$, contains both $\mathbf{e}_{41}$ and $\mathbf{e}_{12}$ and has a length that is shorter than $4n$, then $\mathbf{s}$ must have the form $\Downarrow(r_1 w_0 w_1)$. Note that $\mathbf{e}_{42}$ must start with r_1. Thus $\mathbf{s}$ must be followed by a march element that starts with r_1. But both $\mathbf{e}_{22}$ and $\mathbf{e}_{32}$ start with r_0, so neither of them can be the required march element. Therefore, in addition to $\mathbf{e}_{22}$, $\mathbf{e}_{32}$, and $\mathbf{s}$, $\mathbf{t}$ must contain at least one other march element that starts with r_1 and follows $\mathbf{s}$ immediately. Thus, the length of t is still at least $8n$.

Subcase 3. ($\mathbf{e}_{41}$ is combined with $\mathbf{e}_{32}$) If the combination, called $\mathbf{s}'$, contains both $\mathbf{e}_{41}$ and $\mathbf{e}_{32}$ and has a length that is shorter than $4n$, then $\mathbf{s}'$ must have the form $\Downarrow(r_0 w_0 w_1)$. Note that $\mathbf{e}_{42}$ must start with r_1. Thus $\mathbf{s}'$ must be followed by a march element that starts with r_1. Of $\mathbf{e}_{12}$ and $\mathbf{e}_{22}$, only $\mathbf{e}_{12}$ starts with r_1. So only $\mathbf{e}_{12}$ could be the required march element. Note, also, that $\mathbf{e}_{31}$ must end with w_0 or r_0. This implies that $\mathbf{s}'$ must be preceded by a march element that ends with w_0 or r_0. Again, only $\mathbf{e}_{12}$ satisfies this condition. Thus, we reach the contradiction that $\mathbf{e}_{12}$ must precede and, at the same time, follow $\mathbf{s}'$. Therefore, t must contain at least one other march element that either starts with r_1 or ends with w_0 or r_0 and the length of t is still at least $8n$.

Case 4. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4$)

Similar to Case 3.

Case 5. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3$, $\mathbf{e}_4 \in S_4$)

Similar to Case 3.

Case 6. ($\mathbf{e}_1 \in S_1'$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3$, $\mathbf{e}_4 \in S_4$)

Similar to Case 3.

Case 7. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4'$)

Note that $\mathbf{e}_{12}$ and $\mathbf{e}_{41}$ are descending while $\mathbf{e}_{22}$ and $\mathbf{e}_{31}$ are ascending. The possible mergings are $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, and $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$. Therefore, we have four possibilities to consider:

Subcase 1. (Do not combine $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, nor $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$) Test **t** will contain at least four march elements $\mathbf{e}_{12}$, $\mathbf{e}_{41}$, $\mathbf{e}_{22}$, and $\mathbf{e}_{31}$, each of which has a length of at least $2n$. Therefore, **t** has a length of at least $8n$.

Subcase 2. (Combine $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, and $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$) Let **s** and **s**$'$ denote the merging of $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, and $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$, respectively. If **s** and **s**$'$ have lengths that are shorter than $4n$, they must have forms $\Downarrow(r_1 w_0 w_1)$ and $\Uparrow(r_0 w_0 w_1)$, respectively. Note that $\mathbf{e}_{22}$ must be preceded by a march element that ends with w_0 or r_0. However, **s** ends with w_1 or r_1. Therefore, in addition to **s** and **s**$'$, **t** must contain at least one other march element that ends with w_0 or r_0, say **m**. Similarly, **s** must be followed by a march element that starts with r_1. However, neither **s**$'$ nor **m** starts with r_1. Therefore, **t** must contain **s**, **s**$'$, **m**, and at least one other march element that starts with r_1. So, the total length of **t** is at least $8n$.

Subcase 3. (Combine $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, do not combine $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$) Let **s** represent the combination of $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$. If **s** has a length that is shorter than $4n$, then **s** must have the form $\Downarrow(r_1 w_0 w_1)$. Note that $\mathbf{e}_{41}$ must be followed by a march element that starts with r_1. Thus **s** must be followed by a march element that starts with r_1. But neither $\mathbf{e}_{22}$ nor $\mathbf{e}_{31}$ starts with r_1. Therefore, in addition to $\mathbf{e}_{22}$, $\mathbf{e}_{31}$, and **s**, **t** must contain at least one other march element that starts with r_1. Thus, the length of **t** is still at least $8n$.

Subcase 4. (Combine $\mathbf{e}_{22}$ with $\mathbf{e}_{31}$, do not combine $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$)

Similar to Subcase 3.

Case 8. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3$, $\mathbf{e}_4 \in S_4'$)

Since all of the four march elements are descending. Thus the possible mergings are $\mathbf{e}_{12}$ with $\mathbf{e}_{21}$, $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, $\mathbf{e}_{32}$ with $\mathbf{e}_{21}$, and $\mathbf{e}_{32}$ with $\mathbf{e}_{41}$. We have seven possibilities to consider:

Subcase 1. (Do not combine any of them) Test $\mathbf{t}$ will contain at least four march elements $\mathbf{e}_{12}$, $\mathbf{e}_{21}$, $\mathbf{e}_{32}$, and $\mathbf{e}_{41}$, each of which has a length of at least $2n$. Then, $\mathbf{t}$ has a length of at least $8n$.

Subcase 2. (Combine $\mathbf{e}_{12}$ with $\mathbf{e}_{21}$, and $\mathbf{e}_{32}$ with $\mathbf{e}_{41}$)

Similar to Subcase 2 of Case 7.

Subcase 3. (Combine $\mathbf{e}_{12}$ with $\mathbf{e}_{41}$, and $\mathbf{e}_{32}$ with $\mathbf{e}_{21}$)

Similar to Subcase 2 of Case 7.

Subcase 4. (Combine $\mathbf{e}_{12}$ with $\mathbf{e}_{21}$, do not combine $\mathbf{e}_{32}$ with $\mathbf{e}_{41}$) If the combination of $\mathbf{e}_{12}$ with $\mathbf{e}_{21}$, called $\mathbf{s}$, contains both $\mathbf{e}_{12}$ and $\mathbf{e}_{21}$ and has a length that is shorter than $4n$, then $\mathbf{s}$ must have the form $\Downarrow(r_1 w_1 w_0)$. Note that $\mathbf{e}_{22}$ starts with r_0. Thus $\mathbf{s}$ must be followed by a march element that starts with r_0. Only $\mathbf{e}_{32}$ satisfies this condition. Note that $\mathbf{e}_{31}$ can be combined with $\mathbf{s}$. Thus we have a segment, $\Downarrow(r_1 w_1 w_0) \Downarrow (r_0 w_1)$, that contains $\mathbf{e}_{12}$, $\mathbf{e}_{21}$, and $\mathbf{e}_{32}$. Note, also, that $\mathbf{e}_{11}$ must end with w_1 or r_1. Thus $\mathbf{s}$ must follow a march element that ends with w_1 or r_1. Only $\mathbf{e}_{41}$ satisfies this condition. Note that $\mathbf{e}_{42}$ can be combined with $\mathbf{s}$. So we end in a segment of length $7n$, $\Downarrow(w_0w_1)\Downarrow(r_1w_1w_0)\Downarrow(r_0w_1)$, that contains $\mathbf{e_{12}}$, $\mathbf{e_{21}}$, $\mathbf{e_{32}}$, and $\mathbf{e_{41}}$.

Subcase 5. (Combine $\mathbf{e_{32}}$ with $\mathbf{e_{41}}$, do not combine $\mathbf{e_{12}}$ with $\mathbf{e_{21}}$) Similar to Subcase 4. The segment of length $7n$ has the form $\Downarrow(w_1w_0)\Downarrow(r_0w_0w_1)\Downarrow(r_1w_0)$.

Subcase 6. (Combine $\mathbf{e_{12}}$ with $\mathbf{e_{41}}$, do not combine $\mathbf{e_{32}}$ with $\mathbf{e_{21}}$) If the combination of $\mathbf{e_{12}}$ with $\mathbf{e_{41}}$, called $\mathbf{s}'$, contains both $\mathbf{e_{12}}$ and $\mathbf{e_{41}}$, and has a length that is shorter than $4n$, then $\mathbf{s}$ must have the form $\Downarrow(r_1w_0w_1)$. Note, however, that $\mathbf{e_{42}}$ must start with r_1, which means that $\mathbf{s}'$ must be followed by a march element that starts with r_1. However neither $\mathbf{e_{32}}$ nor $\mathbf{e_{21}}$ starts with r_1. Thus, in addition to $\mathbf{e_{32}}$, $\mathbf{e_{21}}$, and $\mathbf{s}'$, $\mathbf{t}$ must contain at least one other march element that starts with r_1. Thus the length of $\mathbf{t}$ is still at least $8n$.

Subcase 7. (Combine $\mathbf{e_{32}}$ with $\mathbf{e_{21}}$, do not combine $\mathbf{e_{12}}$ with $\mathbf{e_{41}}$)

Similar to Subcase 6.

Case 9. ($\mathbf{e_1} \in S_1$, $\mathbf{e_2} \in S_2'$, $\mathbf{e_3} \in S_3'$, $\mathbf{e_4} \in S_4$)

Similar to Case 7.

Case 10. ($\mathbf{e_1} \in S_1'$, $\mathbf{e_2} \in S_2$, $\mathbf{e_3} \in S_3$, $\mathbf{e_4} \in S_4'$)

Similar to Case 7.

Case 11. ($\mathbf{e_1} \in S_1'$, $\mathbf{e_2} \in S_2$, $\mathbf{e_3} \in S_3'$, $\mathbf{e_4} \in S_4$)

Applying a similar argument as in Case 8, we can prove that if $\mathbf{t}$ locates all faulty cells and has a length that is shorter than $8n$, then **t** must have length $7n$ and must have the form $\Uparrow(w_1w_0)\Uparrow(r_0w_0w_1)\Uparrow(r_1w_0)$ or $\Uparrow(w_0w_1)\Uparrow(r_1w_1w_0)\Uparrow(r_0w_1)$.

Case 12. ($\mathbf{e}_1 \in S_1'$, $\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3$, $\mathbf{e}_4 \in S_4$)

Similar to Case 7.

Case 13. ($\mathbf{e}_1 \in S_1'$, $\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4$)

Similar to Case 3.

Case 14. ($\mathbf{e}_1 \in S_1'$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4'$)

Similar to Case 3.

Case 15. ($\mathbf{e}_1 \in S_1'$, $\mathbf{e}_2 \in S_2$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4'$)

Similar to Case 3.

Case 16. ($\mathbf{e}_1 \in S_1$, $\mathbf{e}_2 \in S_2'$, $\mathbf{e}_3 \in S_3'$, $\mathbf{e}_4 \in S_4'$)

Similar to Case 3. □

Proposition 7.6 *A march test* **t**$'$ *of the form (a)* $\Downarrow(w_0w_1)\Downarrow(r_1w_1w_0)\Downarrow(r_0w_1)$, *(b)* $\Uparrow(w_0w_1)\Uparrow(r_1w_1w_0)\Uparrow(r_0w_1)$, *(c)* $\Downarrow(w_1w_0)\Downarrow(r_0w_0w_1)\Downarrow(r_1w_0)$, *or (d)* $\Uparrow(w_1w_0)\Uparrow(r_0w_0w_1)\Uparrow(r_1w_0)$ *cannot locate fault* M_2.

Proof: Let cells i and k be two faulty cells created by an arbitrary instance of a fault M_2 of type T_2. Without loss of generality, assume that $i < k$. Assume that **t**$'$ has form (**a**). After the memory is powered up, $[c^i, c^k]$ is $[1,0]$ or $[0,1]$. The first march element changes $[c^i, c^k]$ to $[1, 0]$, which means that cell k is changed to 0 erroneously. Thus the r_1 in the second march element detects the fault and locates cell k. Since the following operations $w_1^i w_0^i$ restore a 1 to cell i, the r_1 cannot detect and locate cell i. Note that in the third march element, w_1^k will keep cell i in 0. Thus r_0 in the third march element still cannot detect and locate cell i. As a result, $\mathbf{t}'$ cannot locate cell i and hence cannot locate M_2. Forms (b), (c), and (d) are proved in the same way as (a). □

Theorem 7.1 *Any march test that locates all faults in the DRAM fault model must have a length of at least* $8n$.

Proof: Follows from Propositions 7.5-7.6. □

7.2 An Optimal March Test for Faulty Cell Location

A march test, called MARCH8N, that locates all of the faulty cells in the DRAM fault model is proposed. As specified in Fig. 7.1, MARCH8N consists of four march elements, which we will call $\mathbf{e}_A, \mathbf{e}_B, \mathbf{e}_C$, and $\mathbf{e}_D$. Note that sequence $\mathbf{e}_A\mathbf{e}_B$ is an element from both sets S_1 and S_4'; in addition, sequence $\mathbf{e}_C\mathbf{e}_D$ is an element from both S_2 and S_3'.

We now prove that MARCH8N locates all of the faulty cells in the DRAM fault model. Write-back cycles are an additional consideration in DRAMs. Write-back cycles cannot disturb the contents of a good memory; however, they can trigger

	$\Uparrow(w_0 w_1)$	$\Uparrow(r_1 w_0)$	$\Downarrow(w_1 w_0)$	$\Downarrow(r_0 w_1)$
Addr	Seq. A	Seq. B	Seq. C	Seq. D
0	$w_0 w_1$	$r_1 w_0$	$w_1 w_0$	$r_0 w_1$
1	$w_0 w_1$	$r_1 w_0$		
2	$w_0 w_1$	$r_1 w_0$	↗	↗
↓	↘	↘	$w_1 w_0$	$r_0 w_1$
			$w_1 w_0$	$r_0 w_1$
n-1	$w_0 w_1$	$r_1 w_0$	$w_1 w_0$	$r_0 w_1$

Figure 7.1: Specification for MARCH8N.

errors in a faulty memory. Thus the effect of refresh operations must be considered in our proof. However, as discussed before, only faults of types T_4 and T_5 contain unstable states that may affect the location. Thus we only consider the effect of refresh operations in the cases of T_4 and T_5.

Proposition 7.7 *MARCH8N locates all of the faulty cells involved in single and multiple stuck-at faults.*

Proof: Let cell i be a stuck bit. If cell i is stuck-at-0, then it is located by the r_1^i in $\mathbf{e}_B$. Similarly, if cell i is stuck-at-1, then it is located by the r_0^i in $\mathbf{e}_D$. Finally, we note that write-back cycles to cell i will not change its content and thus do not prevent location. □

Corollary 7.1 *MARCH8N locates all of the faulty cells created by a word line to bit line short.*

Proposition 7.8 *MARCH8N locates all single interconnected cells.*

Proof: Let cells i and k be two interconnected cells whose behaviour is governed by a fault of types T_1 or T_2. Without loss of generality, assume that $i < k$. We have two cases to consider:

Case 1. ($M = M_1$) Consider the transition diagram in Figs. 5.1 and 4.1. After sequence $\mathbf{e}_A$, both M and M_0 must be in state $[1,1]$. The w_0^k in $\mathbf{e}_B$ causes $[c^i, c^k]$ to become $[0,0]$ in the fault and $[1,0]$ in a good memory. Hence, the r_1^i in $\mathbf{e}_B$ detects the fault and thereby locates cell i. Sequence $\mathbf{e}_C$ causes $[c^i, c^k]$ to become $[0,0]$ in both the faulty and the good memory. The w_1^i in $\mathbf{e}_D$ causes $[c^i, c^k]$ to become $[1,1]$ in the fault and $[1,0]$ in a good memory. Hence, the r_0^k in $\mathbf{e}_D$ detects the fault and thereby locates cell k.

Case 2. ($M = M_2$) Consider the transition diagram in Figs. 5.2 and 4.1. The w_1^i in $\mathbf{e}_A$ causes $[c^i, c^k]$ to become $[1,0]$ in the fault and $[1,1]$ in a good memory. Hence, the r_1^k in $\mathbf{e}_B$ detects the fault and locates cell k. The w_0^k in $\mathbf{e}_C$ causes $[c^i, c^k]$ to become $[1,0]$ in the fault and $[0,0]$ in a good memory. Hence, the r_0^i in $\mathbf{e}_D$ detects the fault and locates cell i. □

Proposition 7.9 *MARCH8N locates all faulty cells caused by shorts between word lines.*

Proof: Let i and k be two cells that interact erroneously according to a fault $M \in \{T_3 \cup \cdots \cup T_7\}$ because of a short between two word lines. Without loss of generality, assume that $i < k$. We have five cases to consider:

Case 1. ($M = M_3$) Consider the state diagrams in Figs. 5.3 and 4.1. After $\mathbf{e}_A$ has been applied, both M and M_0 must be in state $[1,1]$. The fault is located by Proposition 7.8 because M_1 and M_3 are equivalent faults following the application of $\mathbf{e}_A$.

Case 2. ($M = M_4$) Consider the state diagrams in Figs. 5.4 and 4.1. The w_0^k in $\mathbf{e}_A$ causes $[c^i, c^k]$ to become $[0,0]$ in the faulty memory and state $[0,0]$ or $[1,0]$ in a good memory. The following w_1^k causes M to enter state $[0,1]$ while causing M_0 to enter state $[0,1]$ or $[1,1]$. The w_0^i in $\mathbf{e}_A$ causes $[c^i, c^k]$ to become $[0,0]$ again in the faulty memory and $[0,1]$ in a good memory. Any subsequent write-back operations to cells i and k cannot change cell k from 0 to 1. Thus the r_1^k in $\mathbf{e}_B$ detects the fault and locates cell k. Note that after $\mathbf{e}_A$ is applied, M_0 must be in state $[1,1]$. The w_0^k in $\mathbf{e}_B$ causes M to enter state $[0,0]$ while causing M_0 to enter $[1,0]$. Any subsequent write-back operations to cells i and k cannot change cell i in M from 0 to 1. Hence, the r_1^i in $\mathbf{e}_B$ detects the fault and locates cell i.

Case 3. ($M = M_5$) Refer to the state diagrams in Figs. 5.5 and 4.1, and then apply a similar argument to the one used in Case 2. Cells i and k are located by $\mathbf{e}_C$ and $\mathbf{e}_D$.

Case 4. ($M = M_6$) Refer to the state diagrams in Figs. 5.6 and 4.1. After $\mathbf{e}_A$ is applied, both M and M_0 must be in state $[1,1]$. Thus w_0^k in $\mathbf{e}_B$ causes M to enter state $[0,0]$ while M_0 enters state $[1,0]$. The r_1^i in $\mathbf{e}_B$ therefore detects the fault and locates cell i. After $\mathbf{e}_C$ is applied, M_0 must be in state $[0,0]$.

Any write-back operations do not affect the good memory. Thus M_0 will stay in state [0,0]. No matter what state M is in, the w_1^i in $\mathbf{e}_D$ will change M to state [1,1]. Hence the r_0^k in $\mathbf{e}_D$ detects the fault and locates cell k.

Case 5. ($M = M_7$) Refer to the state diagrams in Figs. 5.7 and 4.1. The w_1^k in $\mathbf{e}_A$ causes M to enter [1,1] and causes M_0 to enter [1,1] or [0,1]. The w_0^i in the same sequence causes M to enter [0,0] while causing M_0 to enter [0,1]. This means that cell k has been changed from 1 to 0 erroneously. The following w_1^i will not change cell k from 0 to 1. Thus the r_1^k in $\mathbf{e}_B$ detects the fault and locates cell k. The w_0^i in $\mathbf{e}_C$ causes both M and M_0 to enter [0,0]. The w_1^k in the same sequence causes M to enter [1,1] while causing M_0 to enter [0,1]. This means that cell i has been changed from 0 to 1 erroneously. The following w_0^k won't change cell i from 1 ot 0. Thus the r_0^i in $\mathbf{e}_D$ detects the fault and locates cell i. □

Theorem 7.2 *MARCH8N is an optimal march test for locating all faulty cells caused in the DRAM fault model.*

Proof: Follows from Theorem 7.1 and Propositions 7.7, 7.8, and 7.9. □

MARCH8N is n operations shorter (11%) than the shortest test for fault location described in [22] and the new test locates all faulty cells for a more general fault model.

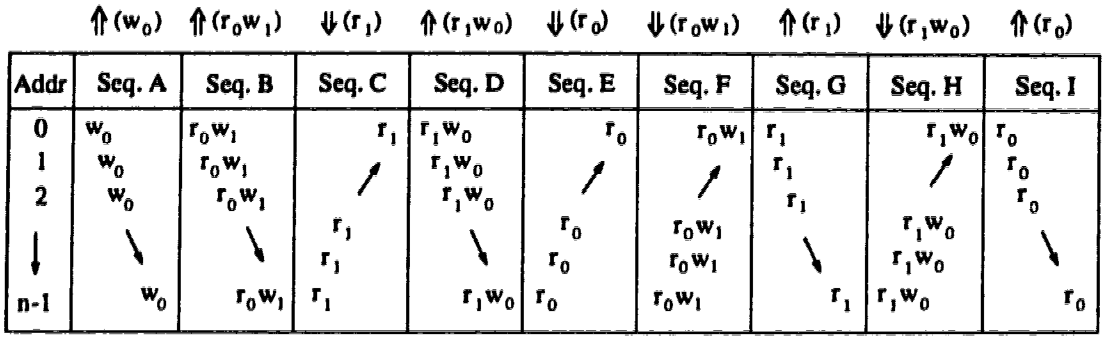
Chapter 8

Fault Type Location

Failure analysis and memory repair are two key activities that are required to improve the yield of newly manufactured DRAMs. In order to perform failure analysis, and more efficiently repair faulty memories, we need to know more about the faults in the memories beyond just locating the faulty cells (faulty cell location) [7]. Specifically, we prefer to know which faulty cell is created by which type of fault; that is, we would like to perform fault type location.

8.1 An Irredundant DRAM Fault Model

As shown in Chapter 4, a short between a bit line and a word line is equivalent to multiple stuck bits. Also, following the application of an initialization sequence, a fault of type T_1 is equivalent to a fault of type T_3. By selecting multiple stuck bits and a fault M_1 of type T_1 as canonical faults for the preceding two fault equivalence classes, we obtain an irredundant fault model whose fault types can be located as



	$\Uparrow(w_0)$	$\Uparrow(r_0w_1)$	$\Downarrow(r_1)$	$\Uparrow(r_1w_0)$	$\Downarrow(r_0)$	$\Downarrow(r_0w_1)$	$\Uparrow(r_1)$	$\Downarrow(r_1w_0)$	$\Uparrow(r_0)$
Addr	**Seq. A**	**Seq. B**	**Seq. C**	**Seq. D**	**Seq. E**	**Seq. F**	**Seq. G**	**Seq. H**	**Seq. I**
0	w_0	r_0w_1	r_1	r_1w_0	r_0	r_0w_1	r_1	r_1w_0	r_0
1	w_0	r_0w_1	$\nearrow$	r_1w_0	$\nearrow$	$\nearrow$	r_1	$\nearrow$	r_0
2	w_0	r_0w_1		r_1w_0			r_1		r_0
$\downarrow$	$\searrow$	$\searrow$	r_1 r_1	$\searrow$	r_0 r_0	r_0w_1 r_0w_1	$\searrow$	r_1w_0 r_1w_0	$\searrow$
n-1	w_0	r_0w_1	r_1	r_1w_0	r_0	r_0w_1	r_1	r_1w_0	r_0

Figure 8.1: Specification for MARCH13N.

compositions of component faults of the following eight types: T_{s0} (Stuck-at-0 bits), T_{s1} (Stuck-at-1 bits), T_1, T_2, T_4, T_5, T_6, and T_7.

8.2 A March Test for Fault Type Location

Figure 8.1 specifies a march test of length $13n$, which we will call MARCH13N, that locates the fault types for all possible combinations of fault instances of the above listed eight fault types. MARCH13N contains 9 march elements. We will refer to the first,..., ninth march elements by $\mathbf{e}_A, ..., \mathbf{e}_I$, respectively. We assume that the addresses of all faulty cells that are located by each march element are stored in separate data files. In general each data file contains the addresses of cells caused by faults of more than one fault type. A post-processing step is therefore required in which set operations are applied to the data files to completely determine which faulty cells are created by each fault type.

Clearly, all of the stuck-at-0 (1) bits will be located by march elements that contain at least one r_1 (r_0) operation. Therefore, we omit discussing the location of stuck bits.

Let cells i and k, where $i < k$, be the addresses of the two faulty cells created by one instance M_1, M_2, M_4, M_5, M_6, or M_7 of fault types T_1, T_2, T_4, T_5, T_6, or T_7, respectively. The following propositions specify how each march element of MARCH13N affects cells in a good memory M_0 and single instances of each of the eight fault types.

We assume, after the memory is powered up, that all of the faults are in arbitrary states, *i.e.,* one of the four possible states 00, 01, 10, or 11.

Proposition 8.1 *After* $\mathbf{e}_A$ *has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{00, 01, 10, 11\} \rightarrow \{00\}$

$M_1 : \{00, 01, 10, 11\} \rightarrow \{00\}$

$M_2 : \{00, 01, 10, 11\} \rightarrow \{10\}$

$M_5 : \{00, 01, 10, 11\} \rightarrow \{00, 10, 11\}$

$M_6 : \{00, 01, 10, 11\} \rightarrow \{00\}$

$M_7 : \{00, 01, 10, 11\} \rightarrow \{00\}$

$M_4 : \{00, 01, 10, 11\} \rightarrow \{00\}$

Proof: Since every state in the transition diagrams of M_0, M_1, and M_2 is stable, it is straightforward to determine the states that M_0, M_1, and M_2 will be in, after a certain march element is applied, by referring to the corresponding transition diagrams.

Some of the states in $M_3, ..., M_7$ are unstable. However, note that after the initializing march element $\mathbf{e}_A$ has been applied, the unstable states in M_3, M_6 and M_7 can never be re-entered. Hence we can still treat M_3, M_6 and M_7 in the same way as M_0, M_1, and M_2.

Whatever state M_4 is in, once a w_0^i is applied, it will enter state 00. Note that state 00 is a stable state and that write-back cycles cannot cause a further transition. Also, the r_0^k will not cause the state to change. Thus after march element $\mathbf{e}_A$ has been applied, M_4 will be in state 00.

After a w_0^i is applied, M_5 can be either in state 00 or 01. Note that state 01 is unstable, so possible write-back cycles will change it to 11. As a result, we have three possible states, 00, 01, and 11, that M_5 may be in. Applying a r_0^k to the three states, we obtain the two possible states, 00 and 10, that M_5 may be in. Again, since state 10 is unstable, we have to consider the effect of write-back cycles. After write-back cycles, 10 will change to 11. Therefore, after $\mathbf{e}_A$ has been applied, M_5 will be in one of the three states 00, 10, or 11. □

Propositions 8.2-8.9 below are proved in the same way as Proposition 8.1.

Proposition 8.2 *After $\mathbf{e}_B$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{00\} \rightarrow \{11\}$

$M_1 : \{00\} \rightarrow \{11\}$, *cell k located*

$M_2 : \{10\} \rightarrow \{01\}$, *cell i located*

$M_4 : \{00\} \rightarrow \{00, 01\}$

$M_5 : \{00, 10, 11\} \rightarrow \{11\}$, *cell k located, cell i possibly located*

$M_6 : \{00\} \rightarrow \{11\}$, *cell k located*

$M_7 : \{00\} \rightarrow \{11\}$

Note that in Proposition 8.2, we have stated that cell i of M_5 is only possibly located. The reason cell i is only possibly located is that, as we already knew, M_5 contains some unstable states, and those unstable states may be further changed depending on the refresh operations. For instance, after $\mathbf{e}_A$ has been applied to M_5, there are three possible states in which M_5 may be. When a r_0^i is applied to cell i and M_5 is in state 00, then cell i is not located. However, if M_5 is in states 10 or 11, then it is located.

An unstable state will be further changed if a write-back cycle is applied to a cell that is on the same word line as cell i or k. Therefore, if we have detailed knowledge of the DRAM layout, then we can uniquely determine whether an unstable state state will be further changed by write-back cycles. That means that with detailed knowledge of the DRAM layout, the ambiguity can be avoided.

Proposition 8.3 *After* $\mathbf{e}_C$ *has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{11\} \rightarrow \{11\}$

$M_1 : \{11\} \rightarrow \{11\}$

$M_2 : \{01\} \rightarrow \{01\}$, *cell i located*

$M_4 : \{00\} \rightarrow \{00, 01\}$, *cell i located, cell k possibly located*

$M_5 : \{11\} \rightarrow \{11\}$

$M_6 : \{11\} \rightarrow \{11\}$

$M_7 : \{11\} \rightarrow \{11\}$

Proposition 8.4 *After $\mathbf{e}_D$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{11\} \rightarrow \{00\}$

$M_1 : \{11\} \rightarrow \{00\}$, *cell k located*

$M_2 : \{01\} \rightarrow \{10\}$, *cell i located*

$M_4 : \{00\} \rightarrow \{00\}$, *cells i and k located*

$M_5 : \{11\} \rightarrow \{10, 11\}$

$M_6 : \{11\} \rightarrow \{00\}$

$M_7 : \{11\} \rightarrow \{00\}$, *cell k located*

Proposition 8.5 *After $\mathbf{e}_E$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{00\} \rightarrow \{00\}$

$M_1 : \{00\} \rightarrow \{00\}$

$M_2 : \{10\} \rightarrow \{10\}$, *cell i located*

$M_4 : \{00\} \rightarrow \{00\}$

$M_5 : \{10, 11\} \rightarrow \{11\}$, *cell i located, cell k possibly located*

$M_6 : \{00\} \rightarrow \{00\}$

$M_7 : \{00\} \rightarrow \{00\}$

Proposition 8.6 *After $\mathbf{e}_F$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{00\} \rightarrow \{11\}$

$M_1 : \{00\} \rightarrow \{11\}$, *cell i located*

$M_2 : \{10\} \rightarrow \{10\}$

$M_4 : \{00\} \rightarrow \{00, 10\}$

$M_5 : \{11\} \rightarrow \{11\}$, *cells i and k located*

$M_6 : \{00\} \rightarrow \{11\}$

$M_7 : \{00\} \rightarrow \{11\}$, *cell i located*

Proposition 8.7 *After $\mathbf{e}_G$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{11\} \rightarrow \{11\}$

$M_1 : \{11\} \rightarrow \{11\}$

$M_2 : \{10\} \rightarrow \{10\}$, *cell k located*

$M_4 : \{00, 10\} \rightarrow \{00\}$, *cell k located, cell i possibly located*

$M_5 : \{11\} \rightarrow \{11\}$

$M_6 : \{11\} \rightarrow \{11\}$

$M_7 : \{11\} \rightarrow \{11\}$

Proposition 8.8 *After $\mathbf{e}_H$ has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{11\} \rightarrow \{00\}$

$M_1 : \{11\} \rightarrow \{00\}$, *cell i located*

$M_2 : \{10\} \rightarrow \{01\}$, *cell k located*

$M_4 : \{00\} \rightarrow \{00\}$, *cells i and k located*

$M_5 : \{11\} \rightarrow \{01, 11\}$

$M_6 : \{11\} \rightarrow \{00\}$, *cell i located*

$M_7 : \{11\} \rightarrow \{00\}$

Proposition 8.9 *After* $\mathbf{e}_I$ *has been applied, the good and faulty memories will change states as follows:*

$M_0 : \{00\} \rightarrow \{00\}$

$M_1 : \{00\} \rightarrow \{00\}$

$M_2 : \{01\} \rightarrow \{01\}$, *cell k located*

$M_4 : \{00\} \rightarrow \{00\}$

$M_5 : \{01, 11\} \rightarrow \{01, 11\}$, *cell k located, cell i possibly located*

$M_6 : \{00\} \rightarrow \{00\}$

$M_7 : \{00\} \rightarrow \{00\}$

Table 8.1 summarizes the faulty cell information obtained by march elements $\mathbf{e}_B, ..., \mathbf{e}_I$ according to Propositions 8.1-8.9. In Table 8.1, each row represents a march element, and each column represents a type of faulty cell. For example M_3^i denotes all cell i's created by M_3. If a faulty cell, say M_5^i, is located by a march element, say $\mathbf{e}_E$, then the entry at (E,M_5^i) is assigned a "⋈". If a faulty cell, say M_5^i, is possibly located by a march element, say $\mathbf{e}_B$, then the entry at (B,M_5^i) is assigned a "*". If a faulty cell is neither located or possibly located by a march element, then a "-" appears at the corresponding entry.

Let $S_B, ..., S_I$ denote the eight sets of faulty cells included in the data files that are generated in march elements $\mathbf{e}_B, ..., \mathbf{e}_I$, respectively. Note that some of the

Table 8.1: Response Summaries

March Elements	M_{s0}	M_{s1}	M_1^i	M_1^k	M_2^i	M_2^k	M_4^i	M_4^k	M_5^i	M_5^k	M_6^i	M_6^k	M_7^i	M_7^k
B	-	$\bowtie$	-	$\bowtie$	$\bowtie$	-	-	-	*	$\bowtie$	-	$\bowtie$	-	-
C	$\bowtie$	-	-	-	$\bowtie$	-	$\bowtie$	*	-	-	-	-	-	-
D	$\bowtie$	-	-	$\bowtie$	$\bowtie$	-	$\bowtie$	$\bowtie$	-	-	-	-	-	$\bowtie$
E	-	$\bowtie$	-	-	$\bowtie$	-	-	-	$\bowtie$	*	-	-	-	-
F	-	$\bowtie$	$\bowtie$	-	-	-	-	-	$\bowtie$	$\bowtie$	-	-	$\bowtie$	-
G	$\bowtie$	-	-	-	-	$\bowtie$	*	$\bowtie$	-	-	-	-	-	-
H	$\bowtie$	-	$\bowtie$	-	-	$\bowtie$	$\bowtie$	$\bowtie$	-	-	$\bowtie$	-	-	-
I	-	$\bowtie$	-	-	-	$\bowtie$	-	-	*	$\bowtie$	-	-	-	-

entries in Table 8.1 denote possible locations. We have the following two theorems.

Theorem 8.1 *Assuming that all of the possible locations do not happen, then all of the eight types of faults in the irredundant fault model can be located as follows:*

Proof: To show the correctness of these equations, we prove the first one as an example. The others can be proved in the same way.

Note that S_C only contains all of the faulty cells created by M_{s0}, M_2^i, and M_4^i; S_G only contains all of the faulty cells created by M_{s0}, M_2^k, and M_4^k. Since one one faulty cell can be involved in one type of fault, by applying an "AND" operation to S_C and S_G, the resulting set will contain exactly all of the faulty cells created by M_{s0}. Thus the stuck-at-0 fault type is located. □

Theorem 8.2 *Assuming that all of the possible locations do happen, then faults of types T_{s0} and T_4, T_{s1} and T_5 cannot be distinguished. The remaining fault types,*

$$T_{s0} = S_C \cap S_G \qquad T_{s1} = S_I \cap S_F$$

$$T_1^i = S_F \cap S_H \qquad T_1^k = (S_B \cap S_D) - T_2^i$$

$$T_2^i = S_B \cap S_C \qquad T_2^k = S_G \cap S_I$$

$$T_4^i = (S_C \cap S_H) - T_{s0} \qquad T_4^k = (S_D \cap S_G) - T_{s0}$$

$$T_5^i = (S_E \cap S_F) - T_{s1} \qquad T_5^k = (S_B \cap S_I) - T_{s1}$$

$$T_6^i = S_H - T_{s0} - T_1^i - T_2^k - T_4^i - T_4^k \qquad T_6^k = S_B - T_{s1} - T_1^k - T_2^i - T_5^k$$

$$T_7^i = S_F - T_{s1} - T_1^i - T_5^i - T_5^k \qquad T_7^k = S_D - T_{s0} - T_1^k - T_2^i - T_4^i - T_4^k$$

however, still can be located.

Proof: Note that a data file contains faulty cells created by M_{s0} iff it contains faulty cells created by M_4 (M_4^i and M_4^k). Thus there is no way to distinguish them. The same situation happens to faulty cells created by M_{s1} and M_5 (M_5^i and M_5^k). If we treat the two pairs of indistinguishable fault types as two new types of faults. Using an argument similar to that of Theorem 8.1, we can prove the theorem. □

Chapter 9

Conclusions

9.1 Summary of Results

Fault detection, faulty cell location, fault type location, and fault diagnosis are important problems in DRAM testing and diagnosis. Together they form a hierarchy of problems in terms of test lengths and fault coverages: *fault diagnosis* $\rightarrow$ *fault type location* $\rightarrow$ *faulty cell location* $\rightarrow$ *fault detection.* In the hierarchy, fault diagnosis has the highest level, while fault detection has the lowest level. The relationship among the problems is that the higher level a problem is in the hierarchy, the more information is obtained from faulty memories under test. However, the corresponding march test must be longer.

The main results obtained in this thesis are as follows:

- A realistic formal fault model for a 4 Mbit Siemens DRAM was proposed using the mathematical methodology for RAM testing developed by Brzozowski and Cockburn. This fault model allows us to perform rigorous analysis such as lower bound derivation and proof of fault coverage.

- A lower bound of $5n$ was derived on the length of any march test that detects all of the faults in the fault model, where n is the size of the memory under test. Also, a lower bound of $5n - 2$ was derived on the length of any arbitrary test that detects all of the faults in the fault model. A march test whose length matches the $5n$ lower bound was described and shown to detect the fault model.

- A lower bound of $8n$ was derived on the length of any march test that locates all of the faulty cells in the fault model. A march test, whose length matches the $8n$ lower bound, was proposed and shown to locate all of the faulty cells in the fault model.

- An irredundant fault model was derived from the original fault model by identifying and removing all of the equivalent faults. A march test of length $13n$ was proposed to locate the fault types in the fault model. In the best case, the march test can locate all eight types of faults in the fault model; in the worst case, however, it still can locate four types of faults, and separate the other four types of faults into two groups each of which contains exactly two types of faults.

Table 9.1 gives the hierarchy of DRAM testing problems studied in this thesis. Every problem and its corresponding lower bound and the length of its march test are given in the table. Note that the test length and upper bound for fault diagnosis are currently only conjectures supported by preliminary work.

Table 9.1: A Hierarchy of DRAM Testing Problems.

Problem	Lower Bound	Test Length
Fault Diagnosis	$O(n \log n)$?	$O(n \log n)$?
Fault Type Location	$\geq 8n$ [30]	$13n$ [30]
Faulty Cell Location	$8n$ [29]	$8n$ [29]
Fault Detection	$5n$ [8]	$5n$ [8]

Although a particular DRAM was used in this thesis, the design of the DRAM appears to be representative of modern DRAM technology. Thus, given suitable descriptions of the faulty behaviours that can be caused by physical defects, it should not be hard to extend the methodology that was used in this thesis to the testing and diagnosis of other DRAMs.

9.2 Future Research

In the following subsections we describe three areas that should be investigated further as a consequence of this thesis.

9.2.1 The Lower Bound for Fault Type Location

In Chapter 8 we described a march test of length $13n$ that, in the best case, locates all of the fault types in the fault model. Fault type location appears at a higher level in the hierarchy of problems than faulty cell location, so any march test that locates the fault types of the fault model must also locate all of the faulty cells. In Chapter 6 we showed that any march test that locates all faulty cells must have a length of at least $8n$. Thus $8n$ is a lower bound on the length of any march test for fault type location. We conjecture that it is possible to improve the $8n$ lower bound to closer to $13n$. The derivation, however, could be very long because many cases need to be considered.

9.2.2 Fault Diagnosis

The fault diagnosis problem, the highest level in the hierarchy, was not studied in this thesis. However, in order to efficiently repair memory, it is sometimes necessary to distinguish each fault component in multiple faults. For example, to repair a pair of interconnected cells, only one of the two faulty cells needs to be replaced by redundancy. Assume that there are x pairs of interconnected cells. If we do not separate each fault component from the others, then we have to replace all of the $2x$ faulty cells. In the worst case, in which each faulty cell is on a distinct word line, $2x$ redundant word lines are needed to repair the memory. If we separate each fault component from the others, in the same worst case, only x redundant word lines are needed. Further research can be done to propose a march test that diagnoses all of the faults in the fault model. Early work on this problem suggests that by using a divide-and-conquer strategy [2], we can design an algorithm to solve this problem in a complexity of $O(n \log n)$, where n is the number of faulty cells. It may be possible to derive a lower bound of $O(n \log n)$ for an algorithm that can solve this problem. The essential idea of the derivation of the lower bound is to prove that this problem is equivalent to sorting, which has a lower bound of $O(n \log n)$.

9.2.3 Detection of Linked Multiple Faults

As mentioned in Chapter 5, the fault model that is used in this thesis is restricted to unlinked faults. That is, any memory cell can be involved in at most one fault component. However, in practice, more than one fault component of the same or different fault types may be linked together. Therefore, the problems of fault detection, faulty cell location, fault type location, and fault diagnosis for an unrestricted fault model remain a subject for future research.

Bibliography

[1] M.S. Abadir and J.K. Reghbati, "Functional Testing of Semiconductor Random Access Memories", *AMC Computing Surveys*, 15(3), pp. 175-198, 1983.

[2] S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, 2nd ed., Addison-Wesley, Reading, MA, USA, 1988.

[3] C.A. Benvit, J.M. Cassard, and K.J. Dimmler, "A 256K dynamic random-access memory", *IEEE J. Solid-State Circuits,* v. SC-17, no. 5, pp.857-862, Oct. 1982.

[4] M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Inc., Woodland Hills, CA, USA.

[5] J.A. Brzozowski and B.F. Cockburn, "Detection of Coupling Faults in RAMs", *J. of Electronic Testing: Theory and Applications*, v. 1, no. 2, May 1990, pp. 151-162.

[6] J.A. Brzozowski and H. Jürgensen, "A Model for Sequential Machine Testing and Diagnosis", *J. of Electronic Testing: Theory and Applications*, vol. 3, no. 3, pp. 219-234, 1992.

[7] M.F. Chang, W. K. Fuchs, and J.H. Patel, "Diagnosis and Repair of Memory with Coupling Faults", *IEEE Trans. Computers*, v. 38, no. 4, pp. 493-500, April 1989.

[8] B.F. Cockburn and L. Shen, "An Optimal Test for a Realistic DRAM Fault Model", *Sixth Workshop on New Directions for Testing*, Montréal, Canada, May 20-22, 1992.

[9] R.Dekker et al., "A Realistic Fault Model and Test Algorithms for Static Random Access Memories", *IEEE Trans. Computers*, vol. c-9, no. 6, pp. 567-572, 1990.

[10] A.D. Friedman and P.R. Menon, *Fault Detection in Digital Circuits*, Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1971.

[11] J.P. Hayes, "Detection of pattern-sensitive faults in random-access memories", *IEEE Trans. Comput.*, v. c-24, pp. 150-157, Feb. 1975.

[12] M. Inoue, T. Yamada, and A. Fujiwara, "A New Testing Acceleration Chip for Low-Cost Memory Tests", *IEEE Design & Test of Computers*, vol. 3, pp. 15-19, March 1993.

[13] B.W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, Don Mills, Canada, 1989.

[14] K. Kokkonen, P.O. Sharp, and R. Albers, "Redundancy techniques for fast static RAMs", *ISSCC Dig. Tech. Papers*, pp.80-81, Feb. 1981.

[15] L.L. Lewyn and J.D. Meindl, "Physical Limits of VLSI DRAM's", *IEEE J. of Solid-State Circuits,* SC-20(1), pp. 231-241, 1985.

[16] W.Maly, "Modelling of Lithography Related Yield Losses for CAD of VLSI circuits", *IEEE Trans. CAD,* vol. CAD-4, no. 3, pp. 166-177, 1985.

[17] P. Mazumder and J.P. Hayes, "Testing and Improving the Testability of Multimegabit Memories", *IEEE Design & Test of Computers* pp. 6-7, March 1993.

[18] W.J. McClean, *A Report On the Integrated Circuit Industry,* Integrated Circuit Engineering Corporation, Scotsdale, AZ, USA, 1990.

[19] W.R. Moore, "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield", *Proc. IEEE,* vol. 74, pp. 684-697, May 1986.

[20] R. Nair and S. M. Thatte, "Efficient Algorithms for Testing Semiconductor Random-Access Memories", *IEEE Trans. on Comput.,* v. c-27, no. 6, pp. 572-576, June 1978.

[21] H.-D. Oberle, M. Maue, and P. Muhmenthaler, "Enhanced Fault Modeling for DRAM Test and Analysis", *Digest of the 1991 IEEE VLSI Test Symp.,* Atlantic City, NJ, U.S.A., April 15-17, 1991, IEEE Comp. Soc., Washington, 1991, pp. 149-154

[22] H.-D. Oberle and P. Muhmenthaler, "Test Pattern Development and Evaluation for DRAMs with Fault Simulator RAMSIM", *Proc. Int. Test Conf. 1991,* pp. 548-555.

[23] C. A. Papachristou and N. B. Sahgal, "An Improved Method for Detecting Functional Faults in Semiconductor Random Access Memories", *IEEE trans. on Comput.*, vol. c-34, no. 2, pp. 110-116, Feb. 1985.

[24] B. Prince, *Semiconductor Memories*, 2nd ed., John Wiley and Sons, 1991.

[25] V.L. Rideout, "One-Device Cells for Dynamic Random-Access Memories", *IEEE Trans. on Electron. Devices*, ED-26 (6), pp. 839-852, 1979.

[26] K.K. Saluja, S.H. Sng, and K. Kinoshita, "Built-In Self Testing RAM: A Practical Alternative", *IEEE Design & Test of Computers*, vol. 4, no. 1, pp. 42-51, Feb. 1987.

[27] S.E. Schuster, "Multiple Word/Bit Line Redundancy for Semiconductor Memories", *IEEE J. Solid-State Circuits*, vol. SC-13, pp. 698-702, Oct. 1978.

[28] J.P. Shen et al., "Inductive Fault Analysis of CMOS Intergrated Circuits", *IEEE Design & Test of Computers*, pp. 13-26, Dec., 1985.

[29] L. Shen and B.F. Cockburn, "An Optimal March Test for Locating Faults in DRAMs", *Proc. IEEE Int. Workshop on Memory Testing*, Aug. 9-10, San Jose, 1993.

[30] L. Shen and B.F. Cockburn, "A March Test for Fault Type Location in DRAMs", *Proc. CCVLSI'93*, Banff, Canada, 1993.

[31] R.T. Smith, J.D. Chlipala, and J.F.M. Bindels, "Laser programmable redundancy and yield improvement in a 64K DRAM", *IEEE J. Solid-State Circuits*, vol. SC-16, no. 5, pp.506-514, Oct. 1981.

[32] T. Sridhar, "A New Parallel Test Approach for Large Memories," *IEEE Design & Test of Computers,* vol. 3, no. 4, pp. 15-22, Aug. 1986.

[33] A.J. van de Goor, *Testing Semiconductor Memories, Theory and Practice,* John Wiley and Sons, 1991.

[34] A.J. van De Goor, "Using March Tests to Test SRAMs", *IEEE Design & Test of Computers,* vol. 10, no. 1, pp. 8-14, March 1993.

[35] N. Weste and K. Eshraghian, "Principles of CMOS VLSI Design", Addison-Wesley, Reading, MA, USA, 1895.

[36] T. Yamada et al., "A 64-Mb DRAM with Meshed Power Line", *IEEE J. Solid-State Circuits,* vol. 26, pp. 1506-1510, Nov. 1991.