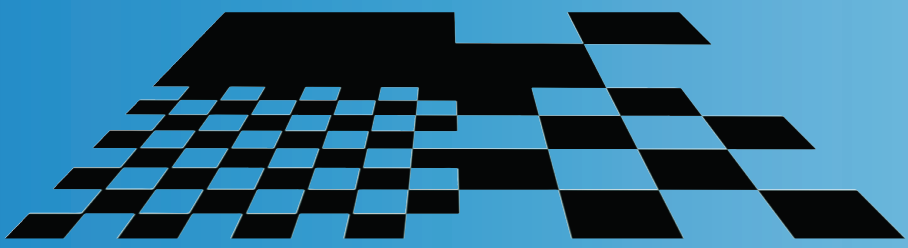
Testing of Modern Semiconductor Memory Structures



Georgi Nedeltchev Gaydadjiev

Stellingen behorende bij het proefschrift

Testing of Modern Semiconductor Memory Structures

van

G. N. Gaydadjiev

Delft, 25 September 2007

1. Both memory testing and university education focus on preventing poorly functioning units from entering industry. The only difference is in the rate of success.

2. In contrast to what many believe, no single memory test exists that is able to detect all possible circuit defects.

3. It is impossible to test asynchronous VLSI circuits using the methods employed for synchronous designs.

4. Moore's law has been a best friend to circuit designers but always the worst enemy of test designers.

5. By using reconfigurable computing as a Trojan horse, software unreliability is infiltrating the hardware domain.

6. In science, the true achievers never fathom their true success.

7. Creating computer architectures is like making music: everybody can make sounds but only few can organize them like Mozart and Beethoven.

8. The day that computers begin to understand and even make jokes will be the day that they are officially declared "intelligent".

9. Nature does not make mistakes, a fact that the human race is continuously trying to disprove.

10. God does not play dice (Albert Einstein): He knows that the devil is in the details.

11. Knowledge brings power (Stamatis Vassiliadis). Power by itself does not create knowledge.

These propositions are considered defendable and opposable and as such have been approved by the supervisor Prof. dr. C.I.M. Beenakker

1. Zowel het testen van geheugens als universitair onderwijs richten zich op het voorkomen van het feit dat slecht functionerende elementen de industrie in gaan. Het enige verschil zit in het slagingspercentage.

2. In tegenstelling tot wat velen geloven is er geen enkele geheugentest die alle mogelijke circuitdefecten kan detecteren.

3. Het is onmogelijk asynchrone VLSI circuits te testen met de methoden die gebruikt worden voor synchrone ontwerpen.

4. De wet van Moore is altijd de beste vriend van circuit ontwerpers geweest, maar altijd de grootste vijand van test ontwerpers.

5. Door gebruik te maken van “reconfigurable computing” als het paard van Troje, infiltreert de onbetrouwbaarheid van software in het hardware domein.

6. De geslaagde wetenschappers zijn zich nooit van eigen succes bewust.

7. Het creëren van computer architecturen lijkt op het schrijven van muziek: iedereen kan geluiden produceren maar weinigen kunnen dit organiseren zoals Mozart en Beethoven.

8. De dag dat computers beginnen te begrijpen en zelfs grappen te maken, zal de dag zijn wanneer zij officieel “intelligent” verklaard zullen worden.

9. De natuur maakt geen fouten, een feit dat de mensheid steeds probeert te weerleggen.

10. God gooit niet met dobbelstenen (Albert Einstein): Hij weet dat “the devil is in the details”.

11. Kennis brengt kracht (Stamatis Vassiliadis). Kracht op zichzelf kan nooit kennis creëren.

Deze stellingen worden verdedigbaar en opponeerbaar geacht en zijn zodanig goedgekeurd door de promotor Prof.dr. C.I.M. Beenakker

Testing of Modern Semiconductor Memory Structures

Testing of Modern Semiconductor Memory Structures

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op dinsdag 25 september 2007 om 10:00 uur

door

Georgi Nedeltchev GAYDADJIEV

elektrotechnisch ingenieur
geboren te Plovdiv, Bulgarije

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. C.I.M. Beenakker

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter	Technische Universiteit Delft
Prof. dr. C.I.M. Beenakker, promotor	Technische Universiteit Delft
Prof. dr. A. Orailoglu	University of California San Diego USA
Prof. dr. L. Carro	Universidade Federal do Rio Grande do Sul, Brazil
Prof. dr. L.K. Nanver	Technische Universiteit Delft
Prof. dr. John Long	Technische Universiteit Delft
Dr. K.L.M. Bertels	Technische Universiteit Delft
Dr. ir. N.P. van de Meijs	Technische Universiteit Delft

This thesis would never be completed without the scientific guidance and inspiration of Stamatis Vassiliadis.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Gaydadjiev, Georgi Nedeltchev

Testing of Modern Semiconductor Memory Structures/Georgi Nedeltchev Gaydadjiev. –
Delft: TU Delft, Faculty of Elektrotechniek, Wiskunde en Informatica – Ill
Thesis Technische Universiteit Delft. – With ref. –
Met samenvatting in het Nederlands.

ISBN 978-90-9022223-3

Subject headings: *memory testing, march tests, fault models, realistic faults, linked faults.*

Printed in The Netherlands

To everybody that understands what science is all about.

Testing of Modern Semiconductor Memory Structures

Georgi Nedeltchev Gaydadjiev

Abstract

In this thesis, we study the problem of faults in modern semiconductor memory structures and their tests. According to the 2005 ITRS, the systems on chip (SoCs) are moving from logic and memory balanced chips to more memory dominated devices in order to cope with the increasing application requirements. The embedded memories are expected to utilize more than 60% of the chip area after 2009. In addition, future SoCs are believed to embed memories of increasing capacities. As a result, the overall SoC yield will be dominated by the memory yield. This trend may make the overall yield unacceptable, unless special measures have been taken.

In this thesis we propose and classify DRAM specific fault models relevant for the state-of-the-art semiconductor technologies. We also define and validate a set of DRAM targeted march tests. In addition, we propose a methodology for deriving conditions and tests for linked memory faults. We also investigate the detection conditions for linked memory faults when one of the faults involved is an address decoder fault. Finally, we propose various optimizations for test time reduction and/or increased fault coverage.

We aimed at high relevancy of the ideas proposed in this thesis. For as far as possible the fault models and the tests presented here are validated using real industrial products. Some of the concepts originally proposed by the author more than 10 years ago are still being widely used by the industry and referred to by the academia. For example, many industrial products did use or are still using March LR, one of the tests derived in this thesis, for testing their (embedded) memories.

Acknowledgements

The work presented in this dissertation was performed in somehow non orthodox way: in small chunks and always in my spare time (for as far as I could have such time). All this started in 1995 when I being enroled as a graduate student in Electrical Engineering in Delft entered prof.dr. Ad van de Goor's office to ask him about master thesis work possibilities. I was looking for a topic in computer architecture but was offered a memory testing one instead. I decided to try this and am grateful to Ad for introducing me to this rather specific but far from boring field. We had a very fruitful collaboration that resulted in several referred publications and a journal paper. The proposed tests and the methodology are still being used by the industry and cited by the academia, a fact that shows the high relevance of what we did back then. Around 1998, two years after my graduation we published the last paper and I decided that my memory testing adventure is finished at that time but as it turned out I was wrong.

In 2002 I got a "conditional" assistant professor position at the Computer Engineering laboratory, TU Delft. The "condition" was indeed completing this PhD thesis, a task that I heavily underestimated back then. At that time, the laboratory was chaired by prof.dr. Stamatis Vassiliadis whom I got to know several years before and who impressed me not only as a scientist but also as a warm hearted human being. Working with Stamatis was always a challenge but challenge I knew about at front. In the five years we worked together, Stamatis and me had many discussions on different topics, a lot of fun, many successes and some disappointments but never had enough time to proceed with all of the ideas and intentions we had in mind. As it turned out, we had much less time than we originally counted on. Exactly at the moment, we had most of the things "under control", e.g. many good quality students, smoothly running master degree program in computer engineering, sufficient financing etc., Stamatis got sick and passed away in a very short period of time. A big loss for the laboratory, the faculty, our university and the scientific community in general. It was an unique experience to work with a person that can understand me from half spoken word and whom I could understand equally easy. I never expected to find someone that had so similar vision about life, work, professional and personal responsibilities. Not surprisingly, in the

last couple of years, we became more than just good colleagues we were also friends. A fact I can be only proud of.

I want to specially thank dr.ir. Said Hamdioui and dr.ir. Zaid Al-Ars who helped me getting back into the field of memory testing after a five years break. They both are formal CE PhD students that became world-class experts in the field of memory testing. So my task seemed not very difficult having Said and Zaid close by. The problem was again in the 24 hours we all have to spend every single day.

Very special thanks to my promotor, prof.dr. Kees Beenakker for giving me the opportunity to officially complete this work. Although memory testing is not exactly his main field of expertise, he significantly contributed to the quality of this thesis and the propositions that accompany it. In addition, Kees made sure that I dedicated adequate amount of time and concentrated on finalizing the thesis.

Special thanks to all my students, graduate and PhD, for their understanding and patience during the last unusually hectic year. I was unable to spend all the time I would like with you. Nevertheless, we succeed to proceed with our valuable work together and publish many good papers.

My immediate thanks go to all my colleagues from the Computer Engineering (CE) laboratory, the ME&CE department, the EWI faculty and the HiPEAC network of excellence. Too many to list here, I would like to specially mention the three famous professors in computer architecture: Mateo Valero, Manolis Katevenis and Per Stenström for the fruitful discussions we had, the inspiration and their continuous support.

Finally, my deepest love and gratitude for my family: my wife Anna my son Alexander, my mother Atanaska and my sister Vesela. I know I have not always been the best husband, father, son and brother one can hope for. You all, however, always believed in me, something I am grateful.

Georgi Delft, The Netherlands, 2007

Contents

List of Acronyms

a	aggressor memory cell
AF	address decoder faults
ATE	automated test equipment
BIST	built in self test
BISR	built in self repair
BL	bit line
CF	coupling fault
CFdr	deceptive read destructive coupling fault
CFds	disturb coupling fault
CFir	incorrect read coupling fault
CFst	state coupling fault
CFtr	transition coupling fault
CFwd	write destructive coupling fault
CMOS	complimentary metal oxide semiconductor
Del	delay time
DRAM	dynamic random access memory
FFM	functional fault model
FP	fault primitive
GND	ground signal line
IFA	inductive fault analysis
IP	intellectual property
ITRS	International Technology Roadmap for Semiconductors
MOS	metal oxide semiconductor
ppm	parts per million
RAM	random access memory
SA	sense amplifier
SAF	stuck at fault
SF	state fault
SoC	system on chip
SRAM	static random access memory
TF	transition fault
v	victim cell
WDF	write destructive fault
WL	word line
Y	yield

Chapter 1

Introduction

The systems on chip (SoC) as known in 2007 are turning into memory hungry devices in order to cope with the continuously increasing application requirements. Another important driver behind this trend is in the significant increase in number of transistors with each technology node and the forthcoming impatience among SoC designers to utilize those resources as on-chip memory. Figure 1.1 shows how embedded memory is expected to dominate the chip area (growing from about 48% in 2004 to more than 60% expected after 2009) according to the 2005 ITRS [1]. In addition, future SoCs are expected to embed memories of increasing sizes, e.g. 256Mbits and more. As a result, the overall SoC yield will be dominated by the memory yield. Due to the fact that memory yield decreases with the increasing memory sizes, the overall yield may become unacceptable, unless special measures are taken. The bottom curve in Figure 1.2 shows the impact increasing memory sizes can have on the yield. For instance, the yield of 24Mbits embedded memory is about 20%; the example assumes a 0.13 micron 12x12mm chip, with a memory defect

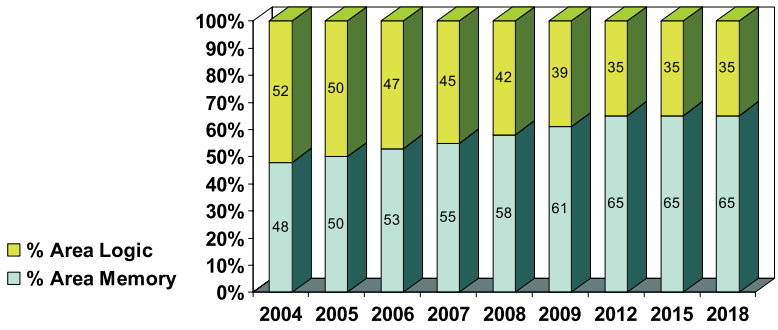


Figure 1.1: The future of embedded memory (ITRS 2005 [1])

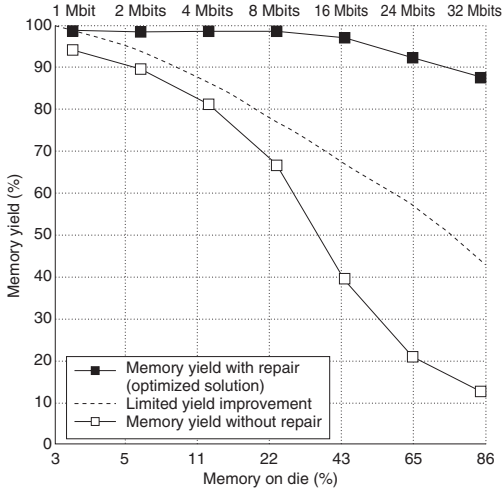


Figure 1.2: Memory sizes versus yield

density of 0.8/square-inch and logic defect density of 0.4/square-inch [2]. To ensure feasible yield levels (the upper curve in Figure 1.2), embedded memories must have repair capabilities; hence "repair" capabilities are considered essential for memory structures as known in 2007 and in future technologies. In complex SoCs; diagnosis and repair algorithms are often required to cope with this. The latter form a challenge since it has been shown to be an NP hard problem [3]. A repair algorithm uses a binary failure bit-map as its input. Such a bit-map is produced by the tests that catch and locate the defective memory cells. For embedded memories, test pattern(s) is generally programmed in a Built In Self Test (BIST) engine. This to cope with the restricted controllability and observability of their inputs and outputs respectively. The memory tests have to guarantee a very high defect coverage, in order to ensure a very low escape rate. The quality of the tests, in terms of the defect coverage and test length, strongly depends on the used fault models. New memory scaling technologies and processes are introducing new defects that were unknown in the past, and therefore novel fault models are emerging.

This all clarifies that the challenges in embedded SoC memory testing will be driven by the following items:

- **Fault modeling:** Novel fault models should be established in order to deal with the new defects introduced by current and future (deep-submicron) technologies. Also the applicability of previously abandoned fault models should be sometimes re-evaluated;
- **Test algorithm design:** Test/diagnosis algorithms that guarantee high defect coverage for the new memory technologies and reduce the DPM level. In addition, those tests should be optimized to allow for low-cost built in implementation;

- **BIST:** Being the only solution allowing at-speed testing for embedded memories BIST engines have to closely follow the trends of complex SoCs. BIST for complex memory structures, e.g. multi-port and content-addressable memories need be addressed. Also issues like power consumption reduction during BIST test and efficient BIST for multiple memories need proper solutions;
- **Built in Self Repair (BISR):** Combining BIST with efficient and low cost repair schemes in order to improve the yield and system reliability is a widely recognized direction to follow. There are, however, challenges related to the optimal (on chip) redundancy calculation and methods to deal with defective redundant memory elements.

In this thesis, only the first two topics will be addressed; the state of art of the fault modeling and the test algorithm design for modern embedded memories. These two problems are generally not considered as the most challenging out of the four above, however they are envisioned to form the sound basis needed to reduce complexity in the latter two. To be more precise, without representative fault models and a simple test set neither BIST or BISR can be efficiently implemented in hardware.

1.1 Fault Modeling

The cost of memory testing keeps increasing with the sizes for every new generation of memory chips as indicated in the past [4]. Precise fault modeling to design efficient tests, in order to keep the test cost and test time within economically acceptable boundaries, is therefore essential. The test quality, in terms of defect coverage, is strongly dependent on the used fault models. Therefore, fault models reflecting the real defects of new memory technologies are essential for developing high defect coverage test algorithms and therefore providing products with low DPM levels.

During the early 1980's many functional fault models for memories have been introduced, allowing the fault coverage of a certain test to be provable while the test time is usually of order $O(n)$; i.e., linear with the size of the memory. Some important fault models introduced in that time are [5]: Stuck-at-Faults and Address-Decoder-Faults. These are abstract fault models and are not based on any real memory design and/or real defects. To reflect the faulty behavior of the real defects in real designs, Inductive Fault Analysis (IFA) was proposed [6]. IFA allows for the establishment of the fault models based on simulated defects at the physical layout level of the design. In addition, IFA is capable of determining the occurrence probability and the importance of each fault model. As a result new fault models were introduced [6]: State-Coupling Fault and Data-Retention Fault. In the late 1990's, experimental results based on DPM screening of a large number of tests applied to a significant population

of memory chips indicated that many detected faults cannot be explained with the well known fault models [7, 8], which suggested the existence of additional faults. This stimulated the introduction of new fault models, based on defect injection and SPICE simulation [9, 10, 11]: Read Destructive Fault, Write Disturb Fault, Transition Coupling Fault, Read Destructive Coupling Fault, and many others.

The published work on memory fault modeling described above focuses on faults sensitized by performing *at most one operation*. For instance, Read Destructive Coupling Fault is sensitized by applying a read operation to the victim cell, while the aggressor cell is put in a certain state (i.e., the required number of operations is **one**). Memory faults sensitized by performing at most one operation are referred as *static faults* [12].

1.1.1 Dynamic Fault Models

Recent publications reveal the existence and the importance of another class of faults in the new memory technologies. It was shown that another kind of faulty behavior can take place in the absence of static faults [13, 14, 15]. This faulty behavior has been attributed to *dynamic faults*; which require *more than one operation* to be performed *sequentially* in time in order for the fault to be sensitized. For example, a write 1 operation followed immediately by a read 1 operation may cause a cell to flip (invert its value) from 1 to 0; however, if only a single write 1 or a single read 1, or a read 1 which is not immediately applied after write 1 operation is performed, then the cell will not flip. In [13] the existence of dynamic faults in the new embedded DRAMs based on defect injection and SPICE simulation was observed. In [14] the presence of dynamic faults was shown in embedded caches of Pentium processors during a detailed analysis of the DPM screening results of a large number of tests. [15] showed the importance of dynamic faults for new SRAM technologies by analyzing DPM screening results of Intel and STMicroelectronics products, and concluded that current and future SRAMs tests need to consider dynamic faults or accept poor DPM numbers.

The majority of the tests currently used in the industry have been designed to target static faults and therefore may not detect dynamic faults. This indicates the importance of dynamic faults for current and future memory technologies. The dynamic fault class, which has been ignored in the past, is now becoming important and has to be taken into consideration. This sets a new direction for further research on memory fault modeling. Items like the following need to be investigated:

- Establishing the complete fault space, the fault framework (based on technology, design and time constraints) and the fault models for dynamic faults;
- Validation based on defect injection and SPICE simulation;

- IFA in order to determine the occurrence probabilities and the importance of each introduced fault model, and provide better understanding of the underlying defects causing dynamic faults.

1.1.2 Other Fault Modeling Aspects

Another special property of memories is that they have signal lines with a very high fan out. Examples of such signals are bit lines, word lines and address decoder pre-select lines. As the memories grow in size and speeds, the lines carrying those signals will have, in addition to a higher load, also higher parasitic capacitance. This increases their sensitivity for delay and timing related faults. Moreover, the significance of the resistive opens is considered to increase in current and future technologies; not only due to the copper wiring but also due to the presence of many, long interconnections and the increasing number of metal layers and vias. Since the partial resistive opens behave as delay and time related faults, these faults will become more important in deep-submicron technologies.

Another aspect that has to be taken into consideration for the deep submicron technologies are the soft errors. The increased operation speed and noise margin reduction that accompany technological scaling, are reducing continuously the reliability of new memories. This process in now approaching a point where it will be infeasible to produce memories that are free of these effects. Contrary to some previous claims that all nanometer memories are becoming so sensitive that even sea level radiation will introduce unacceptable soft errors [16], in [17] it was proven that only SRAM (and peripheral logic) show increasing soft error rates (SER) with each technology node. DRAM system reliability remains constant due to the rapidly decreasing bit SER (1000 times improvement over 7 technology generations) [17]. Designing SRAM soft error tolerant structures is the only way to follow the technological scaling. Among the most efficient techniques are error detecting and error correcting codes, which will not only detect and correct soft errors, but also will compensate for the possible incomplete test/diagnosis coverage. DRAMs, on the other hand, seem to be one of the most robust devices in terms of soft error immunity [17]. This is somehow ironic if we recall that this problem was first discovered in DRAMs.

Other considerations for fault modeling for new technologies are (but not limited to):

- Transistor Short channel effect: lowering the threshold voltage may make the drain leakage contribution significant;
- Cross talk effect and noise from power lines;
- The impact of process variation on the speed failures.

Research on the above topics will be a source of new fault models. This will enable dealing with new defects; hence the development of new, optimal, high

coverage tests and diagnostic algorithms. They will reduce the DPM level and enhance repair capabilities. The greater the fault detection and localization coverage, the higher the repair efficiency; hence the higher the obtained yield.

1.2 Test Algorithm Design

Memory tests and fault detection have experienced a long evolutionary process. The early tests (typically before the 1980's) can be classified as *Ad-Hoc* tests because of the absence of formal fault models and proofs. Tests as Scan, Galpat and Walking 1/0 [18] belong to this class. They have further the property that for a given fault coverage, the test time was excessively long (except for Scan), typically of order of $O(n^2)$, which made them very uneconomical for larger memories.

After the introduction of fault models during the early 1980's, march tests became the dominant type of tests. The advantages of march tests lay in two facts. First, the fault coverage of the considered/known models could be mathematically proven, although one could not have any idea about the correlation between the models and the defects in the real chips. Second, the test time for march tests was linear with the size of the memory, which made them acceptable from an industrial point of view. Some well known march tests, that have been shown to be efficient, are: Mats+ [19], March C- [20], PMOVI [21], IFA 13n [6]. As new fault models have been introduced in the late 1990's, based on defect injection and SPICE simulation, other new march tests have been developed to deal with them. Examples of such tests are March SR [11] and March SS [22].

Conventional memory test algorithms are basically designed to detect static functional faults (that are most likely to occur) in order to determine if the chip is defective or not; in other words, they are pass/fail tests for static faults. As shown in the previous section, the importance of developing new fault models increases with the new memory technologies. In addition, the shrinking technology will be a source of previously unknown defects/faults. The traditional tests are thus becoming insufficient/ inadequate for the today's and the future high speed memories. Therefore, new appropriate test algorithms have to be developed. On the other hand, as the memories occupy a significant part of the SoC, they dominate the overall yield; hence memory fault diagnosis becomes very important. Diagnosis techniques play a key role during the rapid development of semiconductor memories for catching design and/or manufacturing errors and failures; hence improving the yield. Although diagnosis has been widely used for memories, it is considered an expensive process due to long test times and complex fault/failure analysis procedure. Efficient diagnosis algorithms will benefit the industry and will play a more important role in the future as the SoC market grows.

Considering the current situation in test algorithm design and today's in-

dustry needs, it can be concluded that new test/diagnosis algorithms still need to be developed; such algorithms have to take into consideration the following practical issues:

- Optimality in terms of time complexity in order to reduce the overall test time;
- Regularity and symmetry, such that the self-test circuit implementation in silicon can be minimized;
- High defect coverage and diagnosis capability in order to increase the repair capabilities and the overall yield;
- Appropriate stress combinations (voltage, temperature, timing and more) that facilitate the detection of marginal faults.

1.3 Conclusions

To generate a high quality test strategy for new (embedded) memory technologies, a thorough procedure must be pursued. First the memory design, e.g. its cells, pre-charge circuit, sense amplifier, has to be well understood. The circuits need to be investigated not only in the way they are expected to operate, but also in the way each of the circuits behaves in the presence of various defects. These defective and faulty operations need to be mapped into fault models. Once the memory design is understood and the proper fault models are generated, the best test patterns can be developed. Since no single test can achieve an acceptable DPM level, a suite of test patterns is often required. Understanding the design, fault models and tests are required in order to prevent shipping defective parts. Redundancy and repair goes beyond that and are required to guarantee adequate yield on the vast majority of memories. The memory design, fault modeling and test development have to be revisited in the light of redundancy. Redundancy algorithms need to be generated to allocate each redundancy dimension to the appropriate fails, thereby maximizing the yield. Finally, the correct built-in-self testing scheme can be designed (using e.g., a micro-code) while achieving a very low DPM level and boosting the overall yield.

1.3.1 Objectives

This thesis has the following main objectives:

1. Propose and classify DRAM specific fault models relevant for state-of-the-art semiconductor technologies (anno 2007);
2. Define and validate a set of DRAM specific march tests;
3. Propose a methodology for deriving conditions and tests for linked memory faults;

4. Investigate the detection conditions of linked memory faults when one of the linked faults is in the address decoder;

5. Propose various optimizations for test time reduction and/or increased fault coverage.

1.3.2 Thesis Organization

This thesis is organized as follows. Chapter 2 introduces the overall test challenge of modern integrated circuits. In Chapter 3 the theoretical foundation on fault modeling, fault primitives, functional fault models and march tests is given. Chapter 4 deals with single cell dynamic fault tests. Both validation of the fault model and analytical / industrial evaluation of the proposed tests is presented. In Chapter 5 a set of DRAM specific new fault models is introduced. Chapter 6 presents specific test optimizations for transient and soft faults. A methodology to cope with static linked faults is described in Chapter 7. Chapter 8 deals with the linked faults in presence of the address decoder faults. Finally, Chapter 9 concludes the thesis. The main contributions are summarized and a set of direction for future work is listed.

Chapter 2

Deep Sub-Micron testing

As silicon integration continues its relentless pace according to the premise of Moore's law, and as we get ever closer to the nanoscale fabrication domain, new and previously unknown failure mechanisms are being observed that need special analysis and modeling techniques. At the same time, the quality requirements on *integrated circuits (ICs)* have risen significantly in the past few decades to levels approaching zero *DPM (defect per million)* for special mission critical applications, such as those in the aerospace and the automotive industries. As a result, close attention should be given to follow the trends of new failure mechanisms in order to prevent them from becoming the show stoppers for tomorrow's ICs.

In this chapter we identify some of the latest trends observed in the semiconductor industry in terms of testing and failure mechanisms as a result of sensitivities in the manufacturing process. We also analyze these trends and propose ways to deal with them, along with proper methods to address the latest challenges.

The chapter is organized as follows. Section 2.1 starts with a classification of the latest trends in testing and failure mechanisms, and discusses the testing trends developing in the IC production process as a whole. Section 2.2 identifies the testing and failure trends resulting from new directions in IC design. Then, Section 2.3 evaluates how testing is impacted and what new failure mechanisms are expected to result from the latest manufacturing techniques. Section 2.4 shows how the testing practice itself is changing as a result of recent IC developments and requirements. Finally, Section 2.5 ends with the conclusions.

2.1 Trends in IC production

In order to successfully bring an IC to the market, production has to go through a number of important stages that ensure the functionality and quality of the product. Figure 2.1 gives a simplified description of the typical production stages of an IC today.

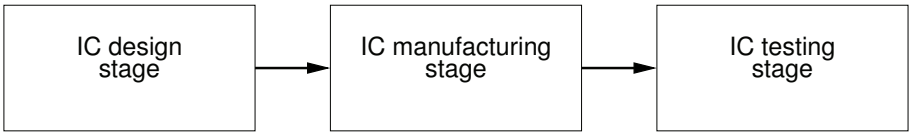


Figure 2.1: Simplified flow of IC production process

The figure shows three main stages: the IC design stage, the IC manufacturing stage and the IC testing stage. As scaling continues and new issues arise in the production process, the challenges for all stages in this figure change, both within each of the three individual blocks, and for the integrated IC production process as a whole. In the following, we describe a couple of trends that involve the production process in general.

2.1.1 Fragmentation of IC production

One of the trends observed for the integrated circuits production process is the gradual fragmentation of the different stages across multiple companies, rather than being carried out by a single semiconductor factory as in the past. This makes it possible to have smaller, specialized and rather flexible companies that closely focus on only one aspect of the semiconductor industrial process.

Many *fabless* design companies are being established, which sell their circuit designs in the form of *intellectual property (IP)* components to other parties, that would in turn integrate them into their complex designs. In addition, so-called *foundries* (such as the Taiwanese Semiconductor Manufacturing Company or shorthand TSMC) are replacing expensive company-owned manufacturing fabs. Also, dedicated test houses are fulfilling the task of ensuring the functionality and quality of the manufactured ICs, thereby allowing in-house testing facilities replacement.

This trend helps companies to cope with the huge investments needed to produce today's top-end semiconductor products. As the cost of the production process continues to increase, this trend is expected to accelerate in two different ways:

- New, more specialized IC production stages will be realized that handle every aspect of the industry from technology research to product specification, design, manufacturing, testing, marketing and sales [23];

- Bigger semiconductor companies will increasingly partner with others in specific fields (research, fabrication, etc.) to muster the heavy investments inherent to future technologies [24].

This trend makes it necessary to have industry-wide standards between the different production stages in order for the different companies to exchange information. In term of testing, standardized test description languages and specialized test data management protocols are needed to facilitate upstream and downstream communication in the IC production flow. In [25] similar observations have been reported.

2.1.2 Value-added testing

The second trend is the usage of testing information not only to screen defective products from reaching the customer, but also to provide feedback to the manufacturing process and/or design process in order to prevent the defects from occurring in the first place (see Figure 2.2). This approach is called *value-added testing* since it provides a way to increase the value of the performed testing by improving manufacturing yield levels and increasing profitability. This, however, comes at an increased price for testing, since test feedback is only possible with the application of more complex diagnostic testing rather than the simpler pass/fail detection testing.

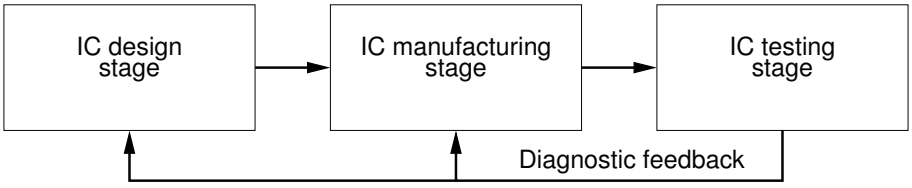


Figure 2.2: Using diagnostic feedback information from the test stage

Value-added testing techniques have been in use by leading silicon manufacturers for a long time [26], where cutting-edge fabrication processes start off with relatively low yields and with many systematic defects. These processes are gradually modified to increase yield (in a procedure referred to as *yield learning*) using diagnostic information from test application [27]. This yield learning process is expected to be increasingly used even in older, well-established manufacturing processes to push yield levels even higher.

In addition to the above mentioned trends that impact the IC production process as a whole, each of the three stages in Figure 2.1 has its own trends, as discussed in the text hereafter.

2.2 Trends in IC design

Market demands coupled with the yet growing number of transistors available for designers today have tilted the design process to adopt novel, challenging design techniques that increase the complexity of both the designs themselves as well as their testing process. In the following text, a number of such new design-related trends are discussed and their impact on testing is identified. Figure 2.3 shows a classification of these trends.

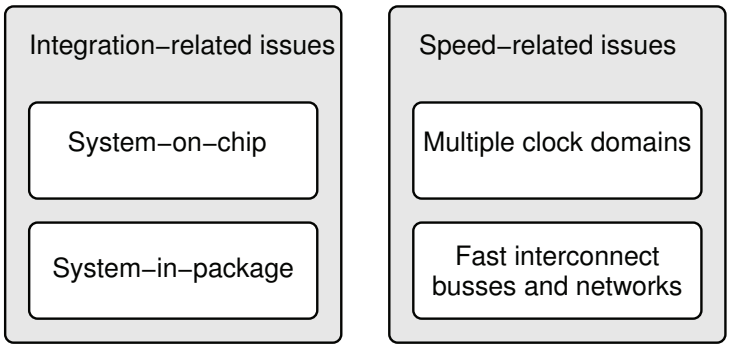


Figure 2.3: Classification of design-related trends

There are two main sub-classes that are driven by the following two factors: the growing device integration and the increasing clock speeds.

2.2.1 System-on-chip integration

The concept of *systems-on-chip (SOCs)* has developed as a result of the need for higher processing power and for hardware integration levels much greater than what could be offered by integrating discrete components on a *printed circuit board (PCB)* [28]. The SOC market is expected to grow significantly in the coming years fueled by the boom of various Internet aware applications in the home and office wireless environments. In general, main circuit types can be distinguished on a typical SOC: logic (e.g., for control and data processing), memory (e.g., for data storage), and analog (e.g., for digital-analog and analog-digital conversions). Each of these components have their own testing requirements, and have dedicated types of *automated test equipment (ATE)* that cater to these requirements. This means that SOCs cannot rely on using a specific kind of ATE optimized to test only one particular type of circuit. There are two different possible solutions: either using *design-for-testability (DFT)* techniques to enable a single tester of testing different core types, or using multiple ATE benches to perform the necessary testing. The first solution costs additional on-chip silicon area, while the second needs investment in

multiple specialized ATE machinery. Depending on the relative cost of each solution and the expected production volume, the most cost effective alternative is chosen. Besides the ATE issue, there is the problem of *electromagnetic compatibility (EMC)* where components should be used that do not electrically interfere with each other. This is one additional side effect of integration that should also be considered in the testing practice.

2.2.2 System-in-package integration

When the manufacturing technology of the different components to be integrated on an SOC are incompatible with each other (e.g. Bipolar and CMOS), the concept of *systems-in-package (SIPs)* offers an attractive alternative. SIPs are made by stacking a number of different chips on top of each other and bonding them into one single package in a technique called *stacked-die packaging*. One famous application for SIPs is stacking a chip manufactured in a technology optimized for logic along with a chip manufactured in a memory tuned (particularly DRAM) technology. These two technologies require incompatible process steps and optimizations, that makes the manufacturing of both circuit types on the same chip a tradeoff between performance and silicon area. The SIPs was successfully used to cope with the above problem as reported for a product targeting the cellphone handset market [29].

The main test challenges faced by SIPs are similar in nature to those faced by SOCs. The integration of components with different test requirements into a single package, means that a number of specialized test techniques have to be used. In addition, SIPs are especially sensitive to edge bonding problems, which require special testing of the behavior at the die interface that goes beyond the simple electrical continuity test. Apart from that, SIPs suffer from a number of unique problems, which require careful attention to power dissipation from multiple chips, capacitive coupling between adjacent dies and incompatible die sizes.

2.2.3 Using multiple clock domains

Due to the hard to deal clock skew and jitter along with the yet increasing power dissipation of the clock tree in modern digital ICs, novel clock-distribution techniques emerge. A number of complex ICs, e.g. in telecommunication products, use several clock signals for functional or performance reasons. Such devices often incorporate many complex clock structures and various clock domain transitions. In these cases, the internal circuitry of the IC is divided into a number of domains, each synchronized by a different clock signal. Testing for such multi-clock circuits can be very challenging due to the "asynchronous" nature of the overall circuit. This also complicates the design of appropriate DFT solutions, which are needed since such devices usually use very high (local) clock frequencies that go beyond the capabilities of the ATE. Special DFT

solutions are needed to overcome the need for a large number of test vectors, which in turn leads to longer test pattern generation and testing times [30].

2.2.4 Fast interconnect busses and networks

There is a growing tendency today toward incorporating fast functional communication interfaces on a chip. These interfaces implement specialized bus protocols with high data communication bandwidth both within the chip and on the interface connecting different chips. Today, speeds for such interfaces range from 1.5 to 3.3 Gbps (giga bits per second), with an expected future increase in bandwidth that may reach 6.4 Gbps and beyond. Testing circuits with fast interfaces involves more than using a test pattern to validate correct functionality. For testing circuits that use wired interfaces, measurements should be made of both voltage and timing, while for testing those using wireless interfaces frequency and power measurements are needed. In addition, ATE-based testing has some limitations when it comes to fast interfaces, and therefore specialized DFT techniques need to be implemented as well [31].

In addition to the above, the aforementioned communication interfaces can be used for transporting test data to the parts of the circuit under test and carry back the test responses. In such a way the traditional test access mechanism hardware overhead (e.g. IEEE1500 test wires) can be avoided [32]. Next generation ATEs are expected to support some of the most widely used communication interfaces.

2.3 Trends in manufacturing

The ever changing nature of the manufacturing process and the on-going research into advanced materials and manufacturing techniques, creates trends that have significant impact on IC testing. Below, we discuss a number of those important trends shown in Figure 2.4. More precisely they are the issues re-

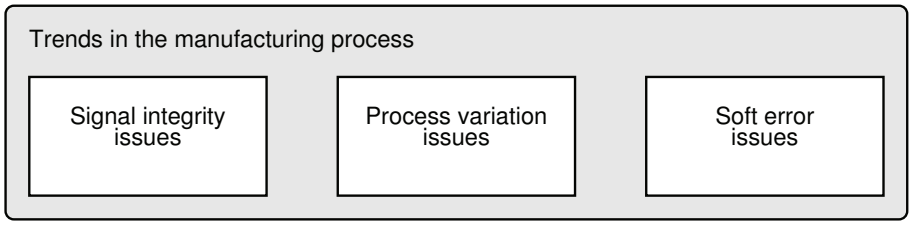


Figure 2.4: List of manufacturing-related trends

lated to the signal integrity, the worsening process variation and the increased contribution of soft errors in modern integrated circuits.

2.3.1 Signal integrity

Signal integrity refers to the general issue of ensuring that the analog voltage present on a given wire correctly reflects the digital signal it represents. Several parasitic effects resulting from the continued scaling down of feature sizes may jeopardize the integrity of digital signals. Problems with signal integrity can be divided into three different types, as indicated in [33]:

- **Propagation delay**—this refers to the time needed for a signal to propagate through a signal line. This delay increases with shrinking line dimensions as a result of the grown in line resistances.
- **Signal interference (crosstalk)**—this refers to the noise introduced on a signal line as a result of a change in the voltage on a neighboring line. This noise increases with decreasing feature sizes as result of the bigger parasitic capacitance between adjacent lines.
- **Crosstalk delay**—this refers to the signal delay induced on a signal line as a result of the simultaneous switching of a neighboring line. This delay is positive when the two lines switch in the opposite directions, while it is negative when they switch in the same direction.

The first effect is inherent to the technology, and should be taken care of during the design stage. The other two effects are design related, and must be tested for to ensure proper functionality. A number of techniques have been proposed to deal with the signal integrity issues in current and future technologies. One of those is the generation of special test patterns that ensure the worst case crosstalk scenarios. This solution, however, requires using a large number of test vectors, which results in a longer test application time. Another solution is to use on-chip noise detection circuitry that signals the development of high noise levels on specific critical signal lines [34].

2.3.2 Process variations

To keep the cost of the manufacturing process low, variations in a number of device parameters are usually tolerated. However, the continued technology scaling has introduced additional variation sources and made process control more difficult. As a result, future technology nodes are expected to suffer from increased process variations and decreased predictability [1]. Process variations can be divided in two main groups, as shown in [35]:

- **Intrinsic (or process) variations**—These variations are always present during circuit operation. They can either be *systematic variations* (i.e., due to known and predictable phenomena) such as changes in transistor channel length, or *random variations* (i.e., due to the inherent unpredictability of the manufacturing process) such as changes in channel doping and gate oxide thickness;

- **Dynamic (or environmental) variations**—These variations only develop temporarily during chip operation. Examples of such variations are temperature distribution or voltage levels on the chip during its normal operation.

The above variations take effect between different chips (inter-die variations), and increasingly within a single chip (intra-die variations) [36]. Solutions to process variations include on-chip sensors that can detect changes in chip behavior and compensate accordingly during chip operation. In addition, statistical timing analysis is becoming an important design tool to model and successfully design high speed circuits in the context of increasing intra-chip process variations [37].

2.3.3 Soft errors

Soft errors are intermittent faults that take place as a result of radiation particle strikes on the chip silicon. This causes a temporary change in the voltage (the local electrical charge) of the effected area. The impact of soft errors becomes increasingly significant with device scaling, as a result of the decreasing node capacitances and reduced supply voltages. *Soft error rate (SER)* values are typically expressed in *FIT (Failure in Time)*, which signifies one error in a billion hours. A FIT rate of 1000 is equivalent to a mean time to failure of 114 years. Currently, the FIT of an SRAM cell is estimated to be around 10^{-4}, while that of logic is estimated to be an order of magnitude lower at 10^{-5}. However, as depicted in Figure 2.5, soft error rate models predict that, by 2011, the contribution of soft errors in the logic will surpass that of SRAM soft errors [38].

There are a number of solutions suggested to reduce SER levels, which can be classified into three different classes [39]:

- **Process technology solutions**—*Silicon-on-insulator (SOI)* is a process technology that uses a much thinner silicon layer than in case of bulk CMOS devices. Therefore, SOI devices collect less charge from an alpha or neutron particle strike compared to their bulk CMOS counterparts. IBM reports a 5x reduction in SER of SRAM devices when SOI technology is used. However, it is unclear whether similar reductions in SER from SOI logic can be achieved.

- **Circuit Solutions**—It is possible to use design techniques to tune the device parameters and create radiation-hardened cells. Examples of such tuning could involve increasing the capacitance and/or supply voltage of a device. It is also possible to design circuits containing redundant states that can recover from a particle strike. However, these designs come with significant area and performance penalties.

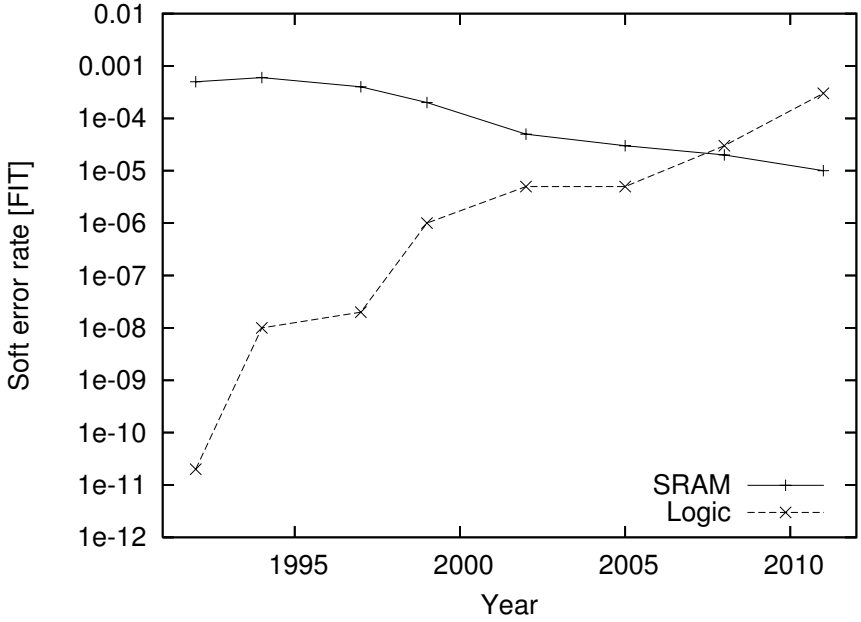


Figure 2.5: Trend in soft error rate for SRAMs and logic

- **Architectural Solutions**—These involve adding specialized components to analyze circuit behavior and detect/compensate for potential failures. Using *error-correction-code (ECC)* circuits is one such example. ECC circuits typically have lower overhead than radiation-hardened cells.[1] Today, ECC circuits are considered by the industry to be the most optimal solution for the soft error problem.

2.4 Trends in testing

As the complexity of the IC test process continues to increase, companies are looking for solutions that reduce the overall cost of manufacturing testing. In the following text, we discuss the current trends in IC testing and the future challenges introduced by the nano-scale technologies.

[1] In a particular example ECC circuits add an overhead of 8 bits per 64 bits of data (i.e., 13%), whereas radiation hardened-cells can have an area penalty of $30 - 100\%$ depending on the aggressiveness.

2.4.1 Test generation

In order to generate the high-quality tests needed for today's circuits, a number of approaches are being explored and adopted by the industry.

- **Extending the existing fault models**—The fault models mainly used to generate test vectors are single stuck-at faults, delay faults and IDDQ faults. Stuck-at faults are mainly caused by bridges as in the example shown in Figure 2.6. In order to increase the effectiveness of stuck-at fault testing, multiple stuck-at faults are being considered, where many faults (not only one) are considered to be present at the same time. Delay faults, on the other hand, are being refined in order to detect small timing deviations that may not always result in an actual failure. IDDQ and $\triangle$IDDQ faults are gradually becoming more analog, where not only a distinction is made between *failing* and *passing* devices, but also the exact value of the failing *current* should be measured [23]. New fault models are also being developed for analog and memory circuits [40].

- **Augmenting the test set**—Test sets (i.e., sequence of test vectors) generated according to the fault coverage requirements of a given fault model are becoming gradually less effective in detecting the faulty behavior observed in modern ICs. One way to overcome this shortcoming of fault models is to augment the model-based test set with a number of hand-picked test vectors, known to detect specific types of faulty behavior. It is rather challenging to come up with a proper augmented test set, as such a set is based more on the understanding of the behavior rather than a specific fault modeling approach.

- **Using specialized tests**—Since SOCs contain memory and analog devices along with logic circuits, it is crucial to include not only logic oriented tests but also specialized tests for the memory as well as the analog parts. Such specialized tests are based on their own set of fault models for each circuit type. Here, there are two challenges to be reckoned with: first is to come up with effective specialized test sets, and second is to apply this test set properly to each circuit despite the restrictions one has with accessing the embedded devices [41].

2.4.2 Test application

In addition to the challenges that stem from generating an effective test set, there is a growing number of limitations on the way these tests are applied to the IC. Below are the main trends with this regard.

- **Tester related challenges**—As the complexity and speed of ICs increase, the complexity and speed of the testers required to test them

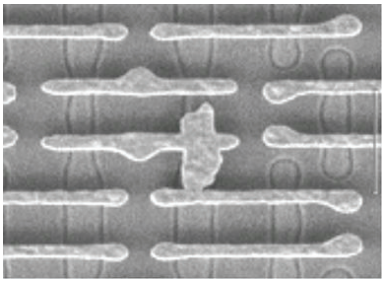


Figure 2.6: Example of a bridge causing stuck-at faults

increase accordingly. The main cause for concern is the overall tester timing accuracy, where rising and falling signal edges must be controlled with continuously decreasing time intervals. With tester accuracy levels as high as tenths of pico seconds, it is unclear how this could be further increased as ICs go to higher frequencies. An accompanying problem is the increasing cost of these specialized high-end testers. The cost aspect has gradually resulted in a move toward so-called *structural testers*, where high accuracy and speed are achieved by embedded DFT circuits, thereby reducing the demands on the external tester [42]. Apart from this, very effective tester-related test approaches widely used in the past, such as using high voltage and high temperature (so-called *burn-in*) are becoming increasingly difficult to apply since they may now cause a failure in the device under test more easily.

- **Challenges with Design for Testability (DFT)**—As the designers are using more and more on-chip resources to facilitate testing (DFT), more implementation related problems appear. Scan chains and *built-in self-test (BIST)* engines are becoming increasingly difficult to implement on modern ICs. Scan chains face a number of problems, ranging from managing heat emitted by dissipated power during test to increasing time needed to perform a scan test with the increasing number of scan flipflops. One solution to this problem is to increase the number of chains, thus decreasing the length of each chain. BIST engines face the problem of performing the correct test set on the device under test. The whole test set is too large to be stored on-chip, while externally loading the test set is difficult due to power, speed and storage related issues. One solution to this problem is using a hybrid of pseudo-random BIST vectors with externally-applied tester seeds. This way, internal tests are partly controlled (or otherwise stated directed) by the tester and driven in the right direction. In this case, it is important to choose effective tester seeds to ensure a high-quality test set [43].

2.5 Conclusions

This chapter presented the trends and challenges of testing and failure mechanisms in deep sub-micron integrated circuits. A classification has been made, based on the flow of the production process of these devices, which results in four different classes of trends: those related to the production process as a whole, those related to design, those in manufacturing and finally those in testing itself. The general trend is toward reducing the cost of the test process by putting as much test circuitry on-chip as possible, rather than focusing on specialized external test equipment. This brings around its own set of challenges that need to be dealt with in the next silicon generations. In this text we will be focusing only on several specific examples of the technology aware memory testing. This considering the fact that each of the challenges presented in this chapter is rather complex to address.

Chapter 3

Memory fault modeling and tests

Considering the fact that memories dominate the SoC area in 2007 and will grow in size in the future, special attention should be paid on their testing challenges. The memory structures' fault behavior is expected to change for new technology nodes and memory BISR is becoming an industrial standard. This argues for the importance of simple but comprehensive fault models; short and efficient memory tests stronger than ever before. In this chapter we will present our memory fault model taxonomy and introduce the concept of march tests.

3.1 Properties of memory fault models

Many *functional fault models (FFMs)* for memories have been introduced in the past; some well known FFMs, which date back to before 1980, are [5]: address decoder faults, stuck-at faults, inversion and idempotent coupling faults, and neighborhood pattern sensitive faults. The following FFMs were introduced later: data retention faults [6], stuck-open faults [6], state coupling faults [6], read disturb faults [44], deceptive read disturb faults [44], and disturb coupling faults [45]. The process of detecting new FFMs has been very ad-hoc and, therefore, slow. Experimental results of applying a large number of tests to a large number of chips [46, 7] indicate that many functional tests do detect faults in memories, which cannot be explained using the current set of known FFMs. This means that additional FFMs do exist.

The fact that many faults in memories do exist for which no FFM has been established yet, together with the fact that the FFMs to be anticipated have to be known in advance, calls for a systematic way to construct potential FFMs and explore the space of all potential FFMs.

Functional faults can be defined as the deviation of the observed memory behavior from the functionally specified one under a set of performed operations. Therefore, two basic ingredients can be identified to any FFM: (1) a list of performed memory operations, and (2) a list of corresponding deviations in the observed behavior from the expected one. Any list of performed operations on the memory is called an *operation sequence*. An operation sequence that results in a difference between the observed and the expected memory behavior is called a *sensitizing operation sequence (S)*[1]. The observed memory behavior that deviates from the expected one is called a *faulty behavior*. A general notation to represent operation sequences is given first, followed by a notation of the faulty behavior.

3.1.1 Representing operation sequences

Any sequence of performed operations on the memory is called an operation sequence. An operation sequence that results in a difference between the observed and the expected memory behavior is called a *sensitizing operation sequence* (S). For example, consider the following operation sequence in a cell $S = 0w1$. It requires the cell to be initialized to 0, followed by an attempt to write a 1 into the cell (this is the operation sequence needed for an up transition fault (TF1) as will be explained next). The observed memory behavior that deviates from the expected one is called a *faulty behavior* or simply a fault. For TF1, the faulty behavior is the inability of the write 1 operation to replace the 0 stored in the cell by a 1.

In order to describe any faulty behavior in the memory, it is important to support any possible operation sequence performed on the memory in the description. A sensitizing operation sequence must list the initial data in the accessed cells and the operations performed on them in order to sensitize the fault. The initial data represents the data in the memory cells prior to the start of a test; this may be random (due to power-on, for example) or deterministic (due to a previously applied test). The operations, on other hand, represent operations performed to sensitize the faulty behavior; these can either be writes w or reads r. Therefore, any operation sequence, expected to result in a faulty behavior, can be represented by the following notation:

$$d_{c_1} \ldots d_{c_i} \ldots d_{c_m}\, Od_{c_1} \ldots Od_{c_j} \ldots Od_{c_n}$$

where c_x: is the cell address used,
O: type of operation on c, $O \in \{w, r\}$,
d: data to be written into c, $d \in \{0, 1\}$,
m: the number of initializations, and

[1] In the literature, some authors make a distinction between operations that sensitize the fault internally and those that result in detecting the fault on the output lines. Since, we are mainly concerned with the internal memory behavior, this distinction is considered unnecessary.

n: the number of operations.

The initial data is described for m cells (denoted as c_i), while the operations are applied to n cells (denoted as c_j). Note that the value of d in a read operation of the form rd_{c_j} represents the expected value of the read operation. This value may be different from the actual read value detected on the output in case of a faulty memory. As an example of the notation, if an operation sequence is denoted by $0_c w1_c r1_c$ then the sequence starts by accessing cell c (which contains a 0) and writing a 1 into it, then reading the written 1. Sometimes, a fault is sensitized because a cell spontaneously loses its stored state, without the need to perform any operation on the memory. Hence, simply setting the cell into a known initial state is enough to sensitize the fault. This situation can also be described using the operation sequence notation above by limiting S to the initial data and eliminating any performed operation. For example, observing the state of cell c which contains a 0 without accessing it can be denoted by 0_c.

3.1.2 Describing faulty behavior

Throughout the 1980s and during the first half of the 1990s, the only functional parameter considered relevant to the faulty behavior was the stored logic state in the memory cell [5]. Recently, another functional parameter, *the output value of a read operation*, has also been considered relevant [47]. Therefore, any difference between the observed and expected memory behavior can be denoted by the following notation $<S/F/R>$ (instead of $<S/F>$ used in the past). S describes the sensitizing operation, F describes the state stored in the faulty cell, $F \in \{0, 1\}$, and R describes the logic output of a read operation, $R \in \{0, 1, -\}$. $R = -$ is used in cases when a *write operation*, and not a read, is sensitizing the fault. The difference between the observed and expected memory behavior denoted by $<S/F/R>$ is referred to as a *fault primitive* (FP). The notion of FPs makes it possible to give a precise definition of an FFM as understood for memory devices. This definition is presented next.

A **functional fault model** is a non-empty set of fault primitives.

The above FFM definition still depends on the selected functional parameters to be observed in the FPs. Yet, this dependence is now precisely known once the FPs are defined. Moreover, it is now possible to distinguish between a specific faulty behavior sensitized by a specific operation sequence and the faulty behavior of the memory sensitized by a number of operation sequences.

3.2 Space of all memory FPs

FPs can be classified according to $\#C$, the number of different cells accessed during an S, according to $\#O$, the number of different operations performed

in an S, and the number of FPs involved in the fault model (see Figure 3.1). For example, if $S = 0_{c_1}\ 0_{c_2}\ w1_{c_1}$ then $\#C = 2$, since two different cells (c_1 and c_2) are accessed by this sequence. On the other hand, $\#O = 1$ for this S since only a single (write 1) operation is performed to c_1.

Depending on $\#C$, FPs can be divided into the following classes:

- If $\#C = 1$ then the FP sensitized by the corresponding SOS is called a *single-cell FP*.
- If $\#C > 1$ then the FP sensitized by the corresponding SOS is called a *coupling FP*. If $\#C = 2$ then it is described as *two-coupling FP* or *two-cell FP*. If $\#C = 3$ then it is described as *3-coupling FP*, etc.

Depending on $\#O$, FPs can be divided into the following classes:

- If $\#O \leq 1$ then the FP sensitized by the corresponding SOS is called a *static FP*.
- If $\#O > 1$ then the FP sensitized by the corresponding SOS is called a *dynamic FP*. If $\#O = 2$ then it is described as *2-operation dynamic FP*. If $\#O = 3$ then it is described as *3-operation dynamic FP*, etc.

In respect to $\#FP$ involved, FPs can be divided into the following classes:

- If $\#FP = 1$ then the FP is called a *simple FP*.
- If $\#FP > 1$ then the FP is called a *linked FP*. If $\#FP = 2$ then it is described as *2-fault linked FP*, etc.

Linked faults, can be identified by inspecting the memory cells associated with the FPs of a given FFM. The linked FFMs are constructed by several FPs and are usually denoted by $\{<S_1/F_1/R_1>\}\ \#\ \{<S_2/F_2/R_2>\}\ \#\ \ldots$.

Figure 3.1 shows a taxonomy of the space of FPs. It is important to note that the three ways to classify FPs are independent, since their definitions are based on independent factors. As a result, a single-cell FP can be static, or dynamic with any number of operations. The same applies to coupling FPs.

Since an FFM is defined as a set of FPs, it is expected that FFMs will inherit the properties of FPs. For example, if an FFM is defined as a collection of single-cell FPs, then the FFM is a single-cell fault model. If an FFM consists of FPs classified into inconsistent classes, single-cell and two-cell FPs for example, it is described as a single-cell and a two-cell fault model.

The taxonomy above does not include such faulty effects as data retention faults and address decoder faults. Considering data retention faults, it is possible to take the retention effects into consideration by introducing a delayed version of an FFM as in $T\{<S/F/R>\}$, where T is the time period we need to wait after performing S to observe the faulty behavior on F and/or R. The address decoder fault are not memory cell array faults and are considered as orthogonal to our taxonomy.

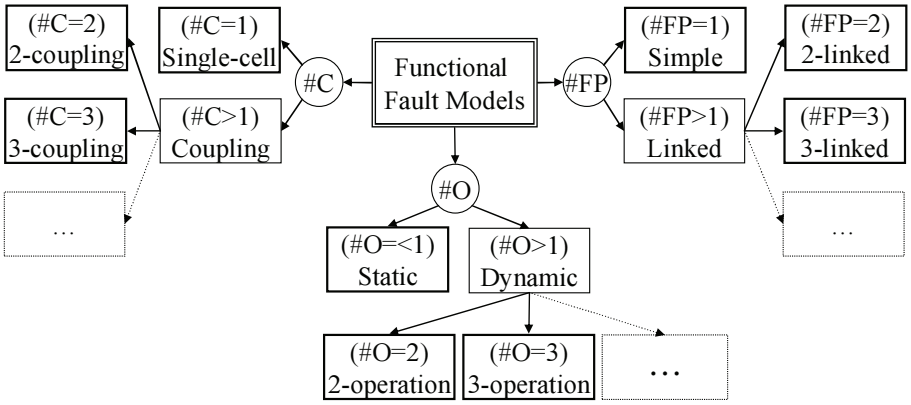


Figure 3.1: Taxonomy of fault primitives.

3.3 Static fault space

Single-cell static FFMs consist of FPs sensitized by performing at most one operation on a faulty cell (i.e., $\#O \leq 1$). As mentioned earlier, a particular FP is denoted by $<S/F/R>$. $S \in \{0, 1, 0w0, 0w1, 1w0, 1w1, 0r0, 1r1\}$ for static FPs, $F \in \{0, 1\}$ while $R \in \{0, 1, -\}$. Table 3.1 lists all single-cell static FFMs and their corresponding FPs using this notation. In total, there are 6 different types of FFMs: state fault (SF), transition fault (TF), write destructive fault (WDF), read destructive fault (RDF), incorrect read fault (IRF), deceptive read destructive fault (DRDF) [44]. The remaining combinations of the S, F and R values do not represent a faulty behavior. For example, $<0w0/0/->$ corresponds to a fault-free $w0$ operation after which the cell contains a 0, as expected.

Table 3.1: Single-cell static FFMs and their corresponding FPs.

#	Fault	FP	Name
1	SF	$<0/1/->$, $<1/0/->$	State fault
2	TF	$<0w1/0/->$, $<1w0/1/->$	Transition fault
3	WDF	$<0w0/1/->$, $<1w1/0/->$	Write destructive fault
4	RDF	$<0r0/1/1>$, $<1r1/0/0>$	Read destructive fault
5	IRF	$<0r0/0/1>$, $<1r1/1/0>$	Incorrect read fault
6	DRDF	$<0r0/1/0>$, $<1r1/0/1>$	Deceptive RDF

Below, a short explanation of a number of well known and new single-cell static FFMs, described in terms of non-empty sets of FPs is provided.

1. **State faults ($\mathbf{SF}_x$)**—A cell is said to have an SF if the logic value of the cell flips before it is accessed, even if no operation is performed on it[2]. Two types of SF exist: $\mathrm{SF}_0 = \{<0/1/->\}$, with FP #1, and $\mathrm{SF}_1 = \{<1/0/->\}$, with FP #2.
2. **Transition faults (TFx)**—A cell is said to have a TF if it fails to undergo a transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) when it is written.
3. **Read disturb faults ($\mathbf{RDF}_x$)** [44]—A cell is said to have an RDF if a read operation performed on the cell changes the data in the cell and returns an incorrect value on the output.
4. **Write disturb faults ($\mathbf{WDF}_x$)**—A cell is said to have a WDF if a non-transition write operation ($0w0$ or $1w1$) causes a transition in the cell.
5. **Incorrect read faults ($\mathbf{IRF}_x$)**—A cell is said to have an IRF if a read operation performed on the cell returns the incorrect logic value, while keeping the correct stored value in the cell.
6. **Deceptive read disturb faults ($\mathbf{DRDF}_x$)** [44]—A cell is said to have a DRDF if a read operation performed on the cell returns the correct logic value, while it results in changing the contents of the cell.
7. **Stuck-at faults ($\mathbf{SAF}_x$)**—A cell is said to have a SAF if it remains always stuck at a given value for all performed operations. Two types of SAF exist: $\mathrm{SAF}_0 = \{<\forall/0/->\}$, and $\mathrm{SAF}_1 = \{<\forall/1/->\}$.

   $\forall$ symbolizes the idea that for all operations the same value remains in the cell. Therefore, $S = \forall$ can be replaced by only those operations that sensitize the fault. This leads to the following equivalent SAF definitions. $\mathrm{SAF}_0 = \{<1/0/->, <0w1/0/->, <1w1/0/->\} = \mathrm{SF}_1 \cup \mathrm{TF}\uparrow \cup \mathrm{WDF}_1$, and $\mathrm{SAF}_1 = \{<0/1/->, <1w0/1/->, <0w0/1/->\} = \mathrm{SF}_0 \cup \mathrm{TF}\downarrow \cup \mathrm{WDF}_0$. The $\cup$ sign is the usual mathematical union sign. In terms of FPs, a $\cup$ connecting a number of sets with FPs means that the FPs are all present in the faulty behavior simultaneously. That is, performing each SOS results in sensitizing the corresponding FP[3].

Two-cell static FFMs (also known as *coupling faults* consist of FPs sensitized by performing at most one operation while considering the faulty effect of two cells. Such FPs can be represented as $<S_a; S_v/F/R>$, where S_a is the sequence performed on the aggressor (a) and S_v is the sequence performed on the victim (v). Table 3.2 lists all two-cell static FFMs and their corresponding

[2]It should be emphasized here that the state fault should be understood in the static sense. That is, the cell should flip in the short time period after initialization and before accessing the cell.

[3]To be precise, stuck-at faults are not strictly static FFMs. SAFs are very general FFMs that describe general dynamic faulty behavior and, therefore, can include many (static and dynamic) single-cell FPs.

FPs. In total, there are 7 different types of two-cell static FFMs: state coupling fault (CFst), disturb coupling fault (CFds), transition coupling fault (CFtr), write destructive coupling fault (CFwd), read destructive coupling fault (CFrd), incorrect read coupling fault (CFir), and deceptive read destructive coupling fault (CFdrd).

Table 3.2: Two-cell static FFMs and their FPs ($x, y \in \{0, 1\}$).

#	Fault	FP	Name
1	CFst	$<0;0/1/->$, $<0;1/0/->$	State coupling
		$<1;1/0/->$, $<1;0/1/->$	fault
2	CFds	$<xwy;0/1/->$, $<xwy;1/0/->$	Disturb coupling
		$<xrx;0/1/->$, $<xrx;1/0/->$	fault
3	CFtr	$<0;0w1/0/->$, $<0;1w0/1/->$	Transition coupling
		$<1;0w1/0/->$, $<1;1w0/1/->$	fault
4	CFwd	$<0;0w0/1/->$, $<0;1w1/0/->$	Write destructive
		$<1;0w0/1/->$, $<1;1w1/0/->$	coupling fault
5	CFrd	$<0;0r0/1/1>$, $<0;1r1/0/0>$	Read destructive
		$<1;0r0/1/1>$, $<1;1r1/0/0>$	coupling fault
6	CFir	$<0;0r0/0/1>$, $<0;1r1/1/0>$	Incorrect read
		$<1;0r0/0/1>$, $<1;1r1/1/0>$	coupling fault
7	CFdrd	$<0;0r0/1/0>$, $<0;1r1/0/1>$	Deceptive read
		$<1;0r0/1/0>$, $<1;1r1/0/1>$	destructive CF

Below, a list of FFMs, some well known and some new, is constructed from the FPs presented in Table 3.2. The new FFMs below are defined in such a way that all FPs are covered by at least one FFM.

1. **State coupling fault (CFst)**—Two cells are said to have a CFst if the victim is forced into a given logic state only if the aggressor is in a given state, without performing any operation on the victim. This fault is special in the sense that no operation is needed to sensitize it and, therefore, it only depends on the initial stored values in the cells. Four types of CFst exist which can be summed up as: $\text{CFst}_{x;y} = \{<x; y/\overline{y}/->\}$, where x, $y \in \{0, 1\}$.
2. **Idempotent coupling fault (CFid)**—Two cells are said to have an CFid if a transition write operation ($0w1$ and $1w0$) on the aggressor forces the victim into a given state. This fault is sensitized by a transition write operation performed on the aggressor. Four types of CFid exist which can be summed up as: $\text{CFid}_{xw\overline{x};y} = \{<xw\overline{x}; y/\overline{y}/->\}$, where x, $y \in \{0, 1\}$.
3. **Inversion coupling fault (CFin)**—Two cells are said to have an CFin if the logic value of the victim is inverted in case a transition write operation is performed on the aggressor. Two types of CFin exist which can be

summed up as: $\text{CFin}_{xw\overline{x}} = \{<xw\overline{x}; y/\overline{y}/->, <xw\overline{x}; \overline{y}/y/->\}$, where x, $y \in \{0, 1\}$.

4. **Non-transition coupling fault (CFnt)**—Two cells are said to suffer from a CFnt if a non-transition write operation ($0w0$ and $1w1$) performed on the aggressor forces the victim into a given state. Four types of CFnt exist which can summed up as: $\text{CFnt}_{xwx;y} = \{<xwx; y/\overline{y}/->\}$, where x, $y \in \{0, 1\}$.
5. **Disturb coupling fault (CFds)**—Two cells are said to have a CFds if an operation (write or read) performed on the aggressor forces the victim into a given logic state. Here, any operation performed on the aggressor is accepted as a sensitizing operation for the fault, be it a read, a transition write or a non-transition write operation. Twelve types of CFds exist which can be summed up as: $\text{CFds}_{xwy;z} = \{<xwy; z/\overline{z}/->\}$ and $\text{CFds}_{xrx;y} = \{<xrx; y/\overline{y}/->\}$, where x, y, $z \in \{0, 1\}$.
6. **Transition coupling fault (CFtr)**—Two cells are said to have a CFtr if a given logic value in the aggressor results in the failure of a transition write operation performed on the victim. This fault is sensitized by a write operation on the victim and setting the aggressor into a given state. Four types of CFtr exist which can be summed up as: $\text{CFtr}_{x;\uparrow} = \{<x; 0w1/0/->\}$ and $\text{CFtr}_{x;\downarrow} = \{<x; 1w0/1/->\}$, where $x \in \{0, 1\}$.
7. **Write disturb coupling fault (CFwd)**—A cell is said to have a CFwd if a non-transition write operation performed on the victim results in a transition when the aggressor is set into a given logic state. Four types of CFwd exist: $\text{CFwd}_{x;y} = \{<x; ywy/\overline{y}/->\}$, where x, $y \in \{0, 1\}$.
8. **Read disturb coupling fault (CFrd)**—Two cells are said to have a CFrd if a read operation performed on the victim destroys the data stored in the victim if a given state is present in the aggressor. Four types of CFrd exist: $\text{CFrd}_{x;y} = \{<x; yry/\overline{y}/\overline{y}>\}$, where x, $y \in \{0, 1\}$.
9. **Incorrect read coupling fault (CFir)**—Two cells are said to have an CFir if a read operation performed on the victim returns the incorrect logic value when the aggressor is set into a given state. Four types of CFir exist: $\text{CFir}_{x;y} = \{<x; yry/y/\overline{y}>\}$, where x, $y \in \{0, 1\}$.
10. **Deceptive read disturb coupling fault (CFdr)**—A cell is said to have a CFdr if a read operation performed on the victim returns the correct logic value and changes the contents of the victim, when the aggressor is set into a given logic state. Four types of CFdr exist: $\text{CFdr}_{x;y} = \{<x; yry/\overline{y}/y>\}$, where x, $y \in \{0, 1\}$.

There is a need to select a collection of the FFMs defined above that would cover all FPs listed in Table 3.2. An analysis of the defined FFMs shows that the FFMs CFst, CFds, CFtr, CFrd, CFir, CFdr and CFwd are necessary and sufficient to cover all two-cell static FPs. Moreover, no other combination of

FFMs may be constructed with this property. Any two-cell static FFM can be represented as the union of two or more of these FFMs. For example, if a defect results in a faulty behavior represented by an incorrect read coupling fault $\{<1; 0r0/0/1>\}$ and a read disturb coupling fault $\{<1; 1r1/0/0>\}$, then the corresponding behavior is presented as: $\{<1; 0r0/0/1>\} \cup \{<1; 1r1/0/0>\}$ $= \{<1; 0r0/0/1>, <1; 1r1/0/0>\}$.

3.4 Dynamic fault space

Dynamic faults can be divided into FPs describing single-cell faults (involving a single-cell), and FPs describing multi-cell faults (involving more than one cell). In this section, we will restrict our analysis to single-cell faults only, because: (a) this is the first attempt for systematic analysis of dynamic faults, and (b) single-cell faults are more dominant than multi-cell faults (as in the case with the well known single stuck-at fault models).

Single-cell dynamic faults consist of FPs sensitized by applying more than one operation to a single cell *sequentially*. We will restrict our analysis to 2-operation dynamic faults because (a) they already have been shown to exist [13, 48, 14], and (b) the probability of dynamic faults decreases as the number of operations increases [49]. As mentioned earlier, a particular FP can be denoted as $< S/F/R >$.

S describes the *sensitizing operation sequence*, which sensitizes a fault F in the cell. Since two operations are considered, there are 18 possible Ss given below; $x, y, z \in \{0, 1\}$ and 'r' denotes a read operation and 'w' denotes a write operation.

- eight Ss have the form '$xwywz$'; e.g., '0w1w0' denotes a write 1 operation applied to a cell whose initial state is 0; the write is followed immediately by another write 0 operation.
- two Ss have the form '$xrxrx$'; e.g., '$0r0r0$' denotes two successive read 0 operations applied to a cell whose initial state is 0.
- four Ss have the form '$xrxwy$'; e.g., '$0r0w1$' denotes a read 0 followed immediately by write 1 applied to a cell whose initial state is 0.
- four Ss have the form '$xwyry$'; e.g., '$1w1r1$ denotes a write 1 followed immediately by read 1 applied to a cell whose initial state is 1.

F describes the value of the *faulty (i.e., victim)* cell (*v-cell*); $F \in \{0, 1\}$. R describes the logical value which appears at the output of the memory if the sensitizing operation applied to the v-cell is a *read* operation: $R \in \{0, 1, -\}$. A '$-$' in R means that the output data is not applicable. For example, $< 0w0w1/0/- >$; $S = 0w0w1$ cause a failing up transition write operation ($F = 0$), no data will appear at the memory output, and therefore R is denoted by '$-$'. Please note that FP shows only the final state/value or the memory cell,

Table 3.3: List of single-cell dynamic FFMs

FFM	FPs
dRDF	$< 0r0r0/1/1 >$, $< 1r1r1/0/0 >$, $< 0w0r0/1/1 >$, $< 0w1r1/0/0 >$, $< 1w0r0/1/1 >$, $< 1w1r1/0/0 >$
dDRDF	$< 0r0r0/1/0 >$, $< 1r1r1/0/1 >$, $< 0w0r0/1/0 >$, $< 0w1r1/0/1 >$, $< 1w0r0/1/0 >$, $< 1w1r1/0/1 >$
dIRF	$< 0r0r0/0/1 >$, $< 1r1r1/1/0 >$, $< 0w0r0/0/1 >$, $< 0w1r1/1/0 >$, $< 1w0r0/0/1 >$, $< 1w1r1/1/0 >$
dTF	$< 0r0w1/0/- >$, $< 1r1w0/1/- >$, $< 0w0w1/0/- >$, $< 1w1w0/1/- >$, $< 1w0w1/0/- >$, $< 0w1w0/1/- >$
dWDF	$< 0r0w0/1/- >$, $< 1r1w1/0/- >$, $< 0w0w0/1/- >$, $< 1w1w1/0/- >$, $< 1w0w0/1/- >$, $< 0w1w1/0/- >$

e.g. the value of the last read operation, all intermediate states or values are not available.

Based on the values of S, F, and R, all detectable single-cell FPs can be enumerated. They consist of a total of 30 FPs. The 30 FPs are compiled into a set of 5 functional fault models (FFMs), and are listed in Table 3.3. The names of the FFMs are chosen in such a way that they represent an extension of the traditional static fault models.

1. *Dynamic Read Destructive Fault (dRDF)*: an operation (i.e., read or write) followed *immediately* by a read operation performed on a single cell changes the data in that cell, and returns an *incorrect* value on the output. The dRDF consists of six FPs; e.g., $< 0w1r1/0/0 >$: applying a 'r1' operation immediately after 'w1' operation to a cell whose initial content was 0, will cause the cell to flip to 0 and the read operation will return a wrong 0 value instead of the expected 1. The first operation involved in the sensitizing operation sequence of dRDF can be a transition write (e.g., write 1 in a cell containing 0), a non-transition write, or a read operation.
2. *Dynamic Deceptive Read Destructive Fault (dDRDF)*: an operation followed *immediately* by a read operation performed on a single cell changes the data in that cell, and returns the *correct* value on the output. The dDRDF consists of six FPs. Here, the operation performed before the read can be a transition write, a non-transition write, or a read operation as in the case of dRDF.
3. *Dynamic Incorrect Read Fault (dIRF)*: a read operation performed *immediately* after another operation (i.e., read, transition, or non-transition

write) on a single cell returns an *incorrect* value on the output, while that cell remains in its correct state. The dIRF consists of six different FPs.

4. *Dynamic Transition Fault (dTF)*: a transition write operation performed *immediately* after an operation (i.e., read, transition, or non-transition write) fails. The dTF consists of six FPs.
5. *Dynamic Write Destructive Fault (dWDF)*: a non-transition write operation applied *immediately* after an operation (i.e., read, transition, or non-transition write) causes that cell to flip. The dWDF consists of six different FPs.

3.5 Simple and linked faults

Simple faults are restricted to one isolated effect per faulty memory cell. On a particular (victim) cell only one S is allowed. Such faults can be a single cell of multiple cells as well as static or dynamic.
A *linked fault* involves two or more simple faults effecting the same (victim) cell. For example, a CF is linked with another CF when both CFs have the same coupled cell (please note that this may cause fault masking). When a static TF is linked with a static CF, and the TF is in the coupling cell, the CF may not be sensitisable; for example, the TF $< 0w1/0/- >$ (or $<\uparrow /0 >$ according to the classic notation) in cell j linked with the CFid $< 0w1; 0/1/- >$ ($<\uparrow;\uparrow>$) whereby cell j is the coupling cell (notation: $< 0w1/0/- >\#< 0w1; 0/1/- >$). When the TF is in the coupled cell, the TF fault effect may be masked by the CF.

Figure 3.2 shows the general form of linked CFs; cell i is coupled to a number a of aggressor cells j all with addresses lower than i, and b aggressor cells k all with addresses higher than i. Masking, e.g., may occur in case of the following two CFids $<0w1; 0/1/->_{j_1 i}$ (old notation $<\uparrow;\uparrow>_{j_1 i}$) and $<0w1; 1/0/->_{j_2 i}$ ($<\uparrow;\downarrow>_{j_2 i}$) because the first CFid causes the cell to be set, which is thereafter reset by the second CFid, assuming linear test following the address order and cell j_1 having lower address than j_2.

Consider the linked fault of Figure 3.2, where the v cell is located at address i. The simple CFs involved in the linked fault can be divided into two *parts*:

- *Low part:* the simple CFs $< x; y/F_p/R_p > j_p i$ (where $j_p < i$)
- *High part:* the simple CFs $< x; y/F_q/R_q > i k_q$ (where $k_q > i$)

Linked faults have a growing importance role in memory testing due to decreasing feature sizes. They can reduce the fault coverage of test due to

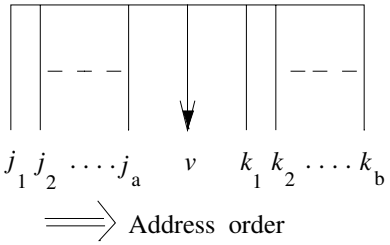


Figure 3.2: General form of linked CFs

masking effects in the victim cell for tests designed with only simple faults in mind. In addition, it is envisioned that the percentage of linked faults will grow when the density of technology nodes increases. Because of their complexity, tests for linked faults require a systematic methodology. Such a methodology will be presented in this thesis based on a set of classical faults without loss of generality.

3.6 Address decoder faults

An *address decoder fault* (AF) can only be present in the address decoder. Address decoder faults are not related to the memory cell array, but can be linked with other simple faults in the latter. Of this fault type the following four different subtypes exist [50];

- Fault 1: with a certain address no cell will be accessed;
- Fault 2: there is no address with which a particular cell can be accessed;
- Fault 3: with a certain address, multiple cells are accessed simultaneously; and
- Fault 4: a certain cell can be accessed with multiple addresses.

A particular subtype cannot occur alone, but only in combination with at least one other subtype, see faults A, B, C and D in Figure 3.3.
The notation used to describe AFs is the following: $< \ldots >$ denotes a particular fault and '...' describes the fault. The capital letters of the alphabet (i.e., A through Z) denote memory addresses and the lower case letters of the alphabet (i.e., a through z) denote memory cells. In a memory consisting of, e.g., 3 cells, and not containing any AFs the following applies: *Aa, Bb, Cc*; i.e., address A is connected with cell a, etc.

Fault A of Figure 3.3 is represented as $< I-, -i >_i$. Fault B is represented as $< Ji, Jj >_{ij}$, where $_{ij}$ represents the address order of the cells i and j. Fault C is represented as $< Ii, Ji >_{ij}$ and fault D as $< Ii, Ji, Jj >_{ij}$.

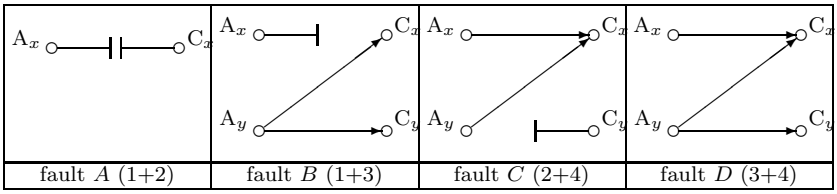


Figure 3.3: Combinations of address decoder faults

3.7 March tests

March tests consist of a family of tests which all have the same structure; they have proven to be superior in terms of short test times, simplicity and high fault coverage.

First we introduce the notation used to describe march tests. Similar to mathematics, a good notation allows for a compact, concise description of the subject matter. Thereafter the relationship between the fault primitives and the march tests to detect them will be briefly described.

3.7.1 March notation

A *march test* consists of a sequence of march elements; a *march element* consists of a sequence of operations which are all applied to a given cell, before proceeding to the next cell. The way one proceeds to the next cell is determined by the *address order* which can be an increasing address order (increasing addresses from cell 0 to cell $n-1$)[4], denoted by the '$\Uparrow$' symbol, or a decreasing address order, denoted by the '$\Downarrow$' symbol. The $\Downarrow$ address order has to be the exact inverse of the $\Uparrow$ address order [5]. For some march elements the address order can be chosen arbitrarily, this will be indicated by the '$\Updownarrow$' symbol. An *operation*, applied to a cell, can be a 'w0' (write '0'), a 'w1', a 'r0' (read '0'), or a 'r1' operation. A complete march test is delimited by the '$\{\ldots\}$' bracket pair; while a march element is delimited by the '$(\ldots)$' bracket pair.

The following march test $\{\Updownarrow\ (w0), \Uparrow\ (r0, w1); \Downarrow\ (r1, w0)\}$ is known as MATS+ [19]. It consists of three march elements, M0, M1 and M2. M2 performs a read one (r1) operation, followed by a write zero (w0) operation on the same cell, after which these two operations are applied to the next cell (in decreasing order). M0 is used to initialize the memory array (all cells set to zero), hence the order of its application is not important.

[4]Any address sequence may be used for the $\Uparrow$ address order; e.g. from 0 to $n-1$, but also a pseudo-randomly determined address sequence; n denotes the total number of addresses.

3.7.2 March tests generation

The FPs are used to identify so-called detection conditions, which describe an incomplete set of march elements specifying the minimum requirements a march test has to fulfill in order to detect the faulty behavior. Detection conditions can easily be used to generate the necessary memory tests to detect the observed faulty behavior. As an example, assume that the fault analysis of a given memory indicates that the memory suffers from an up transition fault dTF $= < 0r0w1/0/- >$. The FP gives a precise description of the way the observed fault can be sensitized. In order to detect this fault, a march test needs to fulfill the detection condition $\Updownarrow (..., w0, ...) \Updownarrow (..., r0, w1, ...) \Updownarrow (..., r1, ...)$, which states that the test has to initialize all cells to 0, then perform an up transition write after read operation on each cell, and finally it has to read the expected (written) 1 from all cells. It is possible to generate many different march tests that satisfy this detection condition. Here are some examples:

- $\{\Updownarrow (w0, r0, w1, r1)\}$
- $\{\Updownarrow (w0) \Updownarrow (r0, w1, r1)\}$
- $\{\Updownarrow (w0) \Updownarrow (r0, w1) \Updownarrow (r1)\}$
- $\{\Uparrow (w0, r0, w1, r1)\}$
- $\{\Uparrow (w0) \Downarrow (r0, w1, r1)\}$
- etc.

In practice, multiple detection conditions for the targeted FPs are used to generate a single march test that can recognize any of the possible fault behaviors.

3.8 Conclusions

This chapter presented the notations needed for the rest of this thesis. A taxonomy of the functional fault models was presented in respect to the number of cells involved, the number of operations needed to sensitize the fault and the number of faults effecting the same faulty cell. The three fault spaces were briefly described. Lastly the march tests and the notation used this work were presented.

Chapter 4

Single-Cell Dynamic Faults

The continued decrease of feature sizes in deep-submicron technology is the source of new defects and faults that strongly depend on stresses and operation sequences for their detection; issues like process variation causing threshold voltage deviation, the increasing influence of parasitics, cross talk, propagation delays, the relative increase in power supply noise and the reduction in the noise margin are just some examples. In this chapter one of the important fault classes for deep-submicron memory technology will be addressed, the *dynamic faults* [13, 48, 14].

Dynamic faults require the application of *more than one operation sequentially* for their sensitization. For example, a write 1 operation followed *immediately* by a read operation causes the cell to flip (change its state to the opposite value) to 0; however, if only a single write or a single read, or a read which does not immediately follow the write is performed, the cell will not flip. The industrial march tests have been mainly designed with static faults (faults that are sensitized by a single operation) in mind, and therefore are expected to perform poorly in devices with dynamic faults. The subject of dynamic faults is only briefly discussed in the literature. In [13] the existence of dynamic faults has been shown for embedded Dynamic Random Access Memories (DRAMs) based on defect injection and SPICE simulation. In [48, 14], the existence of dynamic faults for static RAMs has been proven and a test targeting dynamic faults caused by "read-after-write" has been proposed. This test, however, covers just a very *small subset* of all possible "two-operation" dynamic faults.

This chapter deals with single-cell dynamic faults. It uses a systematic way to model them, and shows the existence of other dynamic faults that have not been addressed in [13, 48, 14]. In addition, it introduces a *complete set* of dynamic fault models based on 'two operations' involving a *single cell*. The chapter shows the shortcomings in the fault coverage of the traditional tests (originally designed for static faults), and thereafter introduces new test primitives for the targeted dynamic faults. The introduced tests are evalu-

ated together with some widely used traditional tests, and the test results are reported.

This chapter is organized as follows. Section 4.1 discusses the validation of dynamic faults for both static and dynamic RAMs. Section 4.2 describes the shortcoming in the fault coverage of traditional tests with respect to dynamic faults. Section 4.3 establishes new test primitives targeting the introduced dynamic faults. Section 4.4 gives the industrial evaluation and discusses the results. Section 4.5 ends with the conclusions.

4.1 Validation of dynamic faults

Currently published work shows the existence of dynamic faulty behavior in the absence of the traditional static faults. The validation of such faults for DRAMs has been shown, based on defect injection and SPICE simulation [13, 14]. For example, the presence of an extra unwanted resistance between the bit line and the memory cell can cause the dynamic faults dRDF, dDRDF and dIRF in the absence of any static faults. That means that the defect can only be detected if the dynamic analysis is considered.

Dynamic faults have also been observed in embedded SRAMs [48]. Further, the wide use of 'hammer tests' (i.e., repeated read or write operations on the same memory cells) in the industry may indicate the existence of the dynamic faults. Furthermore, the 'Holey Shmoo problem' [51] in which the L1 cache of IBM System/390 G6 microprocessor fails to pass consecutive write patterns also indicates that dynamic faults can be caused by 'a write followed immediately by another write' (i.e., dTF or dWDF). It is clear from the above, that this set of fault models for dynamic faults has to be explored, and that the appropriate test algorithms have to be established.

4.2 Effectiveness of the traditional tests

Table 4.1 summarizes the fault coverage of the most well-known memory tests (determined analytically); the test length of each test is also included; n denotes the size of the memory, R denotes the number of rows, and C denotes the number of columns.

In Table 4.1, "a/b" denotes that the test detects 'a' of the 'b' FPs of the corresponding FFM. E.g., March C- detects none of the FPs of the six dRDF, while March SS detects four of them. The last column in the table (i.e., 'FC') gives the total detected FPs for the corresponding test. E.g., March RAW detects 18/30 of single-cell dynamic faults. It is clear from the table that the traditional tests designed with static faults in mind do not detect all targeted dynamic faults. This shows the necessity for new tests targeting dynamic faults specifically.

Table 4.1: DS fault coverage for known tests

#	Tests	Test length	dRDF	dDRDF	dIRF	dTF	dWDF	Total FC
1	SCAN [52]	4n	0/6	0/6	0/6	0/6	0/6	0/30
2	MATS+ [19]	5n	0/6	0/6	0/6	1/6	0/6	1/30
3	MATS++ [18]	6n	1/6	0/6	1/6	2/6	0/6	4/30
4	March A [53]	15n	0/6	0/6	0/6	2/6	0/6	2/30
5	March B [53]	17n	2/6	0/6	2/6	4/6	0/6	8/30
6	March C- [20, 5]	10n	0/6	0/6	0/6	2/6	0/6	2/30
7	March G [53]	23n	2/6	1/6	2/6	4/6	0/6	9/30
8	March LR [54]	14n	2/6	0/6	2/6	2/6	0/6	6/30
9	March RAW [14]	26n	6/6	4/6	6/6	2/6	2/6	20/30
10	March SS [22]	22n	4/6	0/6	4/6	2/6	2/6	12/30
11	PMOVI [21]	13n	2/6	2/6	2/6	2/6	0/6	8/30
12	Galpat [18]	6n+4nRC	0/6	0/6	0/6	2/6	0/6	2/30
13	Walking 1/0 [5]	8n+2nRC	0/6	0/6	0/6	2/6	0/6	2/30

4.3 Test primitives for dynamic faults

This section proposes tests for the introduced single-cell dynamic faults. For each FFM, two tests will be introduced. One is written to facilitate the diagnosis of the FPs during DPM (defect per million) screening, while the other version is optimized in terms of test length. During the industrial evaluation, all test patterns are implemented; this gives more detailed information that can be used in order to establish the importance of each FFM as well as FPs.

Table 4.2 lists the tests designed for each dynamic fault model. The first column gives the name of the test and its test length (n denotes the memory size in cells); e.g., Test dRDF-Diag is the test designed for dRDF for diagnosis purpose, while Test dRDF-Opt is the optimized test for the same fault. The second column of the table gives the description of the test. It can be verified easily that each FP of each FFM is detected with its proposed test. E.g., The Test dRDF-Diag detects all dRDF FPs:

1. FP=$< 0r0r0/1/1 >$ is detected by M2=$\Updownarrow (r0, r0)$ (i.e., the second march element) of the test.
2. FP=$< 1r1r1/0/0 >$ is detected by M5 of the test.
3. FP=$< 0w0r0/1/1 >$ is detected by M1 of the test.
4. FP=$< 0w1r1/0/0 >$ is detected by M3 of the test.
5. FP=$< 1w0r0/1/1 >$ is detected by M6 of the test.
6. FP=$< 1w1r1/0/0 >$ is detected by M4 of the test.

The last test included in Table 4.2 and referred to as *March DS1* is designed to cover all single-cell dynamic faults; this test has a test length of $43n$ and is a superset of all of the above march tests.

Table 4.2: Tests for dynamic single-cell faults

Name (test length)	Test description
dRDF-Diag ($13n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,r0)\ ;\ \Updownarrow(r0,r0)\ ;\ \Updownarrow(w1,r1);\ \Updownarrow(w1,r1)\ ;\ \Updownarrow(r1,r1)\ ;\ \Updownarrow(w0,r0)\}$
dRDF-Opt ($11n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,r0,r0)\ ;\ \Updownarrow(w1,r1)\ ;\ \Updownarrow(w1,r1,r1)\ ;\ \Updownarrow(w0,r0)\}$
dDRDF-Diag ($19n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,r0,r0)\ ;\ \Updownarrow(r0,r0,r0)\ ;\ \Updownarrow(w1,r1,r1)\ ;\ \Updownarrow(w1,r1,r1)\ ;\ \Updownarrow(r1,r1,r1)\ ;\ \Updownarrow(w0,r0,r0)\}$
dDRDF-Opt ($15n$)	$\{\Updownarrow(w0);\ \Updownarrow(w0,r0,r0,r0)\ ;\ \Updownarrow(w1,r1,r1)\ ;\ \Updownarrow(w1,r1,r1,r1)\ ;\ \Updownarrow(w0,r0,r0)\}$
dTF-Diag ($19n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,w1,r1)\ ;\ \Updownarrow(r1,w0,r0)\ ;\ \Updownarrow(w1,w0,r0)\ ;\ \Updownarrow(r0,w1,r1)\ ;\ \Updownarrow(w0,w1,r1)\ ;\ \Updownarrow(w1,w0,r0)\}$
dTF-Opt ($17n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,w1,r1,w0,r0)\ ;\ \Updownarrow(w1,w0,r0,w1,r1)\ ;\ \Updownarrow(w0,w1,r1)\ ;\ \Updownarrow(w1,w0,r0)\}$
dWDF-Diag ($19n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,w0,r0)\ ;\ \Updownarrow(r0,w0,r0)\ ;\ \Updownarrow(w1,w1,r1)\ ;\ \Updownarrow(w1,w1,r1)\ ;\ \Updownarrow(r1,w1,r1)\ ;\ \Updownarrow(w0,w0,r0)\}$
dWDF-Opt ($17n$)	$\{\Updownarrow(w0)\ ;\ \Updownarrow(w0,w0,r0,w0,r0)\ ;\ \Updownarrow(w1,w1,r1)\ ;\ \Updownarrow(w1,w1,r1,w1,r1)\ ;\ \Updownarrow(w0,w0,r0)\}$
March DS1 ($43n$)	$\{\Updownarrow(w0)\ ;\Updownarrow(w0,w0,r0,r0,r0,w0,r0)\ ;\ \Updownarrow(w1,r1,r1,w0,w0,r0,w1,r1)\ ;\ \Updownarrow(w1,w1,r1,r1,r1,w1,r1)\ ;\ \Updownarrow(w0,r0,r0,w1,w1,r1,w0,r0)\ ;\ \Updownarrow(w0,w1,r1,w0,w1,r1)\ ;\ \Updownarrow(w1,w0,r0,w1,w0,r0)\}$

It should be noted that no test is included in the table for dIRF; this is because dIRF and dRDF require the same sensitizing/detection operations. Therefore dIRF can be detected with the same tests as those established for dRDF; i.e., any test detecting dRDF also detects dIRF. Outside of the memory, one cannot distinguish between the two faults since the only difference is that for dIRF the state of the cell is not changed while for dRDF it is changed. Since in this attempt of studying dynamic faults, the diagnosis of the faults will be based on the output signature, it is not possible to distinguish between the two faults.
Table 4.3 summarizes the introduced tests in this section, together with their fault coverage. It can be concluded that dTF and dWDF can be detected only by the tests designed to cope with these specific fault models.

4.4 Industrial evaluation

This section focuses on the industrial evaluation of the traditional tests as well as the tests targeting single-cell dynamic faults. The memory chips considered are 65nm technology 131 Kbytes embedded SRAMs. For all tests used in this experiment (performed at wafer level), the same algorithmic and non-algorithmic stresses have been used.

The non-algorithmic stresses consist of the environmental conditions, exter-

Table 4.3: Summary of single-cell dynamic tests

Test	Fault coverage					
	dRDF	dDRDF	dIRF	dTF	dWDF	Total
dRDF-Diag	6/6	3/6	6/6	0/6	0/6	15/30
dRDF-Opt	6/6	3/6	6/6	0/6	0/6	15/30
dDRDF-Diag	6/6	6/6	6/6	0/6	0/6	18/30
dDRDF-Opt	6/6	6/6	6/6	0/6	0/6	18/30
dTF-Diag	2/6	2/6	2/6	6/6	0/6	12/30
dTF-Opt	2/6	0/6	2/6	6/6	0/6	10/30
dWDF-Diag	2/6	2/6	2/6	0/6	6/6	12/30
dWDF-Opt	2/6	0/6	2/6	0/6	6/6	10/30
March DS1	6/6	6/6	6/6	6/6	6/6	30/30

nally applied to the wafer under test. They do not impact the sequence and/or the type of the memory operations. However, they may have a great impact on the fault coverage. The used non-algorithmic stresses in this experiment consist of: (a) high voltage (1.24V), (b) high speed (2Ghz), and (c) low temperature (-25 C).

The algorithmic stresses specify the way an algorithm is performed, and therefore they influence the sequence and/or the type of the memory operations. All tests have been implemented using similar algorithmic stresses consisting of 'Fast X' address sequence and a 'solid data-background'. Fast X addressing increments or decrements the address in such a way that each step goes to the next memory cells row; while solid data-background means that the initial data used consist of all 0s (i.e., 0000.../0000...) or all 1s for the memory array.

4.4.1 Coverage results

All tests listed in Table 4.1 and in Table 4.2 have been implemented and applied to few millions embedded SRAM chips. To reduce the large data-base for analysis purposes, four classes of tests are defined and presented in Table 4.4.

1. Static Tests (S-Tests). They consist of four tests that mainly target static faults.
2. Diagnosis Dynamic Tests (DiagD-Tests). They consist of four dynamic tests (see Table 4.2) designed for diagnosis purpose to target the single-cell dynamic faults of Table 3.3.
3. Optimal Dynamic Tests (OptD-Tests). They consist of four optimized dynamic tests (see Table 4.2) designed to target the single-cell dynamic faults of Table 3.3.
4. Static and Dynamic Tests (SD-Tests). These are tests which were originally designed to cover static faults; however, due to their structure, they also detect some of the dynamic faults. The four tests with the

most promising fault coverage for single-cell dynamic faults have been selected; see table 4.1.

Table 4.4: Classification of the tests

Static (S-Tests)	Diag. Dynamic (DiagD-Tests)	Opt. Dynamic (OptD-Tests)	Static & dynamic (SD-Tests)
Scan	dRDF-Diag	dRDF-Opt	March RAW
MATS+	dDRDF-Diag	dDRDF-Op	March SS
MATS++	dTF-Diag	dTF-Opt	March G
March C-	dWDF-Diag	dWDF-Opt	PMOVI

Figure 4.1 shows the Venn-diagrams of the failing devices for different test classes, where DiagD-Tests and OptD-Tests are compared with S-Tests and SD-Tests. The fault coverage (FC) of S-Tests is FC=335, of DiagD-Tests is FC=392, of OptD-Tests is FC=247 and that of SD-Tests is FC=484. Interestingly the DiagD-Tests and OptD-Tests detect respectively 121 and 96 faults that are not detected with S-Tests. In addition, they detect respectively 18 and 13 faults that are not detected with SD-Tests. Moreover, they are able to detect 15 and 11 faults not detected by either S-Tests or SD-Tests. It should be noted that an analysis done on all test classes showed that the total faults detected with all test classes is 514, from which 16 faults are detected only with DiagD-Tests and/or OptD-Tests; i.e., 3.1% of the total faults are uniquely detected with DiagD-Tests and/or OptD-Tests. The above clearly indicates the importance of considering dynamic faults in order to achieve high fault coverage and higher product quality. Not considering dynamic faults will translate in unwanted escapes and therefore increase in DPM (Defect-per-Million) level.

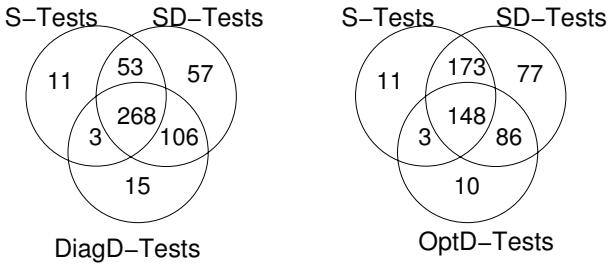


Figure 4.1: Venn-diagram of failing chips

4.4.2 Comparison of dynamic tests

Figure 4.2 shows a comparison of the two test classes targeting dynamic faults (DiagD-Tests and OptD-Tests) and March DS1, which is a test designed to target all single-cell dynamic faults of Table 3.3; see also Table 4.2. It is important to note that the intersection of all dynamic tests is 218 faults; these faults consist of the easy to detect static faults (e.g., stuck-at-fault) and also of the single-cell dynamic faults targeted in this chapter. Moreover, Figure 4.2 shows that DiagD-Tests have the highest FC and detect 128 faults that cannot be detected with OptD-Tests neither with March DS1; and therefore they are **not** single-cell dynamic faults. This indicates that DiagD-Tests have the capability to detect other faults that are not considered in the fault model presented here (e.g., dynamic *coupling faults*, delay faults, ...).

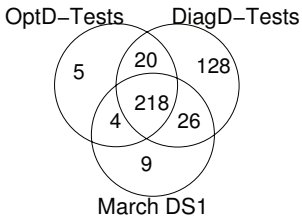


Figure 4.2: Comparison of the dynamic tests

Table 4.5 shows the *union* and the *intersections* of all dynamic tests developed in this chapter; see Table 4.2. A die belongs to the union of two tests if at least one of the two tests detects the fault, and belongs to the intersection if *both* tests detect the faulty die. The first column in each table gives the test number; the second column the name of the test. The column 'FC' lists the fault coverage of the corresponding test; the column 'UFs' gives the number of *unique faults (UFs)* each test detects. Unique faults are faults that are only detected once by a single test; e.g., March DS1 detects 9 UFs that are not detected with any other dynamic test of Table 4.2.

The union and the intersection of each pair of tests is shown in the rest of the table. The numbers in bold (the "main diagonal") give FC of the tests, which are also listed in the column 'FC'. The part above the main diagonal shows the intersection for each test pair, while the part under the diagonal lists the union of each test pair. For example, the union of March SD1 and dTF-Diag Test is 394 while their intersection 198. Based on the table and the Venn-diagram of Figure 4.2, one can conclude the following:

- The total number of faulty chips detected with all dynamic tests is 410.
- The best test, in terms of FC, is dTF-Diag with FC=394, followed by March SD1 with FC=257.

Table 4.5: Intersections and unions of dynamic tests

#	Test	FC	U.F.	1	2	3	4	5	6	9	8	9
1	March DS1	257	9	**257**	166	163	149	157	198	172	119	106
2	dDRDF-Diag	178	2	269	**178**	158	150	152	131	120	116	104
3	dDRDF-Opt	173	2	267	193	**173**	149	154	131	124	113	102
4	dRDF-Diag	161	1	269	181	185	**161**	150	125	115	113	101
5	dRDF-Opt	169	0	269	195	188	180	**169**	137	124	117	109
6	dTF-Diag	335	118	394	382	377	371	367	**335**	172	94	84
7	dTF-Opt	182	2	267	240	231	228	227	345	**182**	85	78
8	dWDF-Diag	134	2	272	196	194	182	186	375	231	**134**	107
9	dWDF-Opt	115	1	266	189	186	175	175	366	239	142	**115**

- The best test, in terms of detecting unique faults which are only detected by a single dynamic test, is dTF-Diag with FC=394 and #UF=118.
- The best union pair in terms of the FC is 394 achieved with dTF-Diag and March DS1.

It is interesting to note that the FC achieved by dTF-Diag test is exceptionally high as compared with the other dynamic tests. Inspecting the nature and the structures of dTF-Diag Test and other dynamic tests (see Table 4.2) reveals that the main property that the dTF-Diag test has is that it consists of *back-to-back* operations with *complementary data values*. E.g., the second march element of dTF-Test $\Updownarrow (w0, w1, r1)$ consist of *write 0 after read 1* back-to-back. Using back-to-back operations along the bit lines (i.e, Fast X addressing) is very powerful in detecting address decoder delay faults, and dynamic/time-related faults in the peripheral circuits of the memory [55]. They are also powerful in detecting *dynamic coupling faults* since they address two different locations with successive operations.

4.5 Conclusions

In this chapter, a systematic approach to analyze dynamic faults has been described. A complete set of two-operation, single-cell dynamic faults has been developed, and appropriate tests have been introduced. The tests have been industrially evaluated together with traditional tests by applying them to real products (Embedded SRAMs) implemented in advanced deep-submicron technology. The results indicated the importance of dynamic faults, their tests, and the exceptional effectiveness of using back-to-back operations during memory testing. The effect of these operations can be increased by using complementary data values. This property makes the newly proposed dTF-Diag test exceptionally effective in achieving very high fault coverage.

Chapter 5

DRAM specific space of memory tests

DRAMs have traditionally played an important role as the main memory of microprocessors, but they gradually find themselves serving in a continuously growing list of applications, in fields ranging from high performance to low power and from networking to graphics [56]. DRAM testing is considered a complex and overly costly activity, requiring a time consuming test development cycle that should be repeated for every new DRAM technology [57, 58]. In order to reduce the cost of DRAM testing, memory tests should be developed specifically for DRAMs to target the specific faults commonly observed in these memories.

In this chapter, we describe a new space of memory tests specifically developed for DRAMs, based on a long term analysis of their faulty behavior in the industry [59]. The tests target all DRAM-specific faults identified to be realistic for their behavior. Three different sets of tests are given: one for hard faults, one for transient faults and one for soft faults. In each set, two tests are given (one for single-cell and one for two-cell faults), making a total of 6 DRAM-specific tests. Some of the tests are also shown to correspond closely to some of the most effective DRAM tests used in practice [60].

The chapter starts with the definition of the different DRAM-specific fault models in Section 5.1. Section 5.2 shows the simulation model used to validate the fault space. The results of the simulation study are discussed in Section 5.3. Section 5.4 derives the space of memory tests suitable to detect the DRAM-specific faulty behavior, while Section 5.5 identifies the similarities these tests have with DRAM tests known to be effective in practice. Section 5.6 ends with the conclusions.

5.1 DRAM-specific faults

DRAM faults in modern CMOS technology can either be attributed to leakage currents (resulting in time dependent faults), or to improperly set voltages (resulting in voltage dependent faults) [61]. Figure 5.1 shows a summary of DRAM-specific faults.

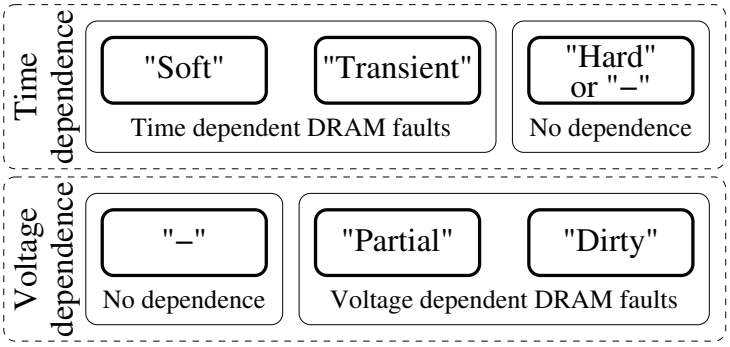


Figure 5.1: Summary of the space of DRAM-specific faults.

5.1.1 Time dependent faults

Time dependent faults are caused by leakage currents in the faulty cells [62]. Time dependence divides all faults into three classes: *soft, transient* and *hard*.

Soft faults—Soft faults (denoted by s) are detectable only after some time from their sensitization. These faults can be detected by adding a *delay* within the test, as it is the case for the *data retention fault*, for example [6]. Soft faults are caused by writing weak voltages into memory cells, that soon get depleted by naturally occurring leakage currents. Soft faults are represented as sFP $=<S_T/F/R>$, where S has an added time parameter T to indicate that some time should elapse before full sensitization. The open defect in Figure 5.2(a) shows an open that may cause soft faults in a DRAM cell. If the open defect has an intermediate resistance value that is not too high (to cause hard faults) and not too low (to result in no fault at all) write operations store a *weak voltage* into the cell. If leakage opposes the weak voltage, the stored information gets lost over an unexpectedly short period of time (e.g. shorter than the DRAM refresh rate).

Transient faults—Transient faults (denoted by t) are memory faults that do not remain sensitized indefinitely, but tend to correct themselves after a period of time. Transient faults are tested for by performing all the operations in the fault in *back-to-back fashion* directly after each other, and directly following them by a detecting read operation. The DRAM open in Figure 5.2(a) may

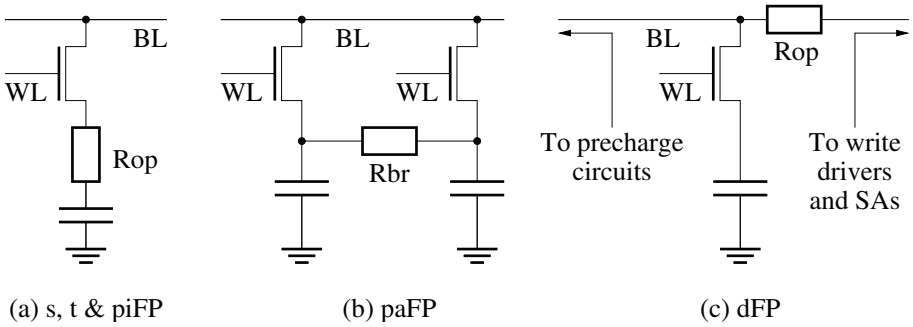


Figure 5.2: Defects causing (a) s, t, p_i, (b) p_a, and (c) causing dirty faults.

cause transient faults. For a specific range of R_{op} values, write operations set a faulty voltage within the cell that is not strong enough to qualify as a hard fault. If leakage tends to correct the weak faulty voltage, the stored voltage gets corrected over time. Transient faults are represented as tFP =$<\underline{S}/F_L/R>$, where the underscore below S means that the operations in S should be performed in back-to-back mode. Furthermore, F has an added time parameter L (*life time*) to indicate that these faults are time bounded. An underscore below operations implies that the operations have to be performed after each other within one march element. For example, if $S = w1\underline{w0}$ then the sensitizing condition should be $\Updownarrow$(..., $w1$, $\underline{w0}$, ...).

Hard faults—Identifying a fault as being hard ("h" or "-") indicates that it is neither soft nor transient (i.e., it is insensitive to time). All the generic faults already described in Chapter 3 belong to this class (hard faults).

5.1.2 Voltage dependent faults

Operations performed on a defective DRAM may set improper voltage levels on memory nodes, thereby causing two types of DRAM faults: partial faults and dirty faults.

Partial faults—Partial faults (denoted with p) are faults that can only be sensitized when a specific memory operation is successively repeated a number of times, either to properly initialize the faulty cell (*partial faults during initialization* p_i), or to properly sensitize the fault in the cell (*partial faults during fault sensitization or activation* p_a). Figure 5.2(a) shows an example of an open (R_{op}) in the cell, causing p_i. R_{op} prevents fully initializing the cell to the required voltage with only a single operation, which means that full initialization requires repeating the operation a number of times. Figure 5.2(b) shows an example of a bridge (R_{br}) between two cells, a causing different type of p_a. These faults are modeled by performing an operation Ox an h (or *hammer*) number of times. For example, if $<xOy/F/R>$ becomes partial during

initialization p_i, it should be modeled as p_iFP $=<x^h Oy/F/R>$.

Dirty faults—Dirty faults (d) assume that after proper initialization or sensitization, the state of the memory (voltages on the bit lines (BLs), the word lines (WLs), or in data buffers) is corrupted, such that subsequent detection is prevented. In order to ensure detectability, additional operations (so called *completing operations*) must be performed to correct the corrupted state of the memory. Figure 5.2(c) shows an example of an open defect (R_{op}) on the BL that causes dirty faults. This defect disconnects memory cells from the write drivers, which prevents the memory controller from writing the cells. This defect also prevents properly precharging the BL. As a result, a $w0$ operation that fails to write 0 in the cell ends up preconditioning the BL to properly sense a 0, thereby causing a dirty fault. These faults are modeled by adding completing operations in square brackets to the FP ($[Ox]$). Detectability of all known dirty faults can be ensured using a completing write operation with data opposite to the data in the victim cell ($[w\overline{y}]$), performed on a cell (b) different from the victim (v) but positioned on the same BL pair. This can be denoted as dFP $=<xO_v y\ [w_b\overline{y}]/\overline{y}/->_{b,v\in \mathrm{BL}}$.

5.1.3 Realistic space of DRAM faults

Any generic memory fault, described in Chapter 3, can represent a DRAM-specific fault by adding DRAM-specific fault attributes to it. First, there are voltage dependent attributes: partial (p), dirty (d), or neither (-). Second, there are time dependent attributes: hard (h), soft (s) and transient (t). Furthermore, the partial attribute can either be initialization related (p_i), or activation (or sensitization) related (p_a).

Based on a detailed analysis of the characteristics of these faults, they represent the full realistic space of DRAM faults that can be constructed for singe-cell faults, as well as two-cell faults [59].

$$\text{Single-cell fault} = \left\{ \begin{array}{c} - \\ p_i \\ d \\ p_i d \end{array} \right\} \left\{ \begin{array}{c} h \\ s \\ t \end{array} \right\} \text{FP} \tag{5.1}$$

$$\text{Two-cell fault} = \left\{ \begin{array}{c} - \\ p \end{array} \right\} \left\{ \begin{array}{c} h \\ s \\ t \end{array} \right\} \text{FP} \tag{5.2}$$

These expressions indicate that any generic single-cell fault can either be regular (-), initialization partial (p_i), dirty (d) or partial dirty (p_id), while being hard (h), soft (s) or transient (t) at the same time. Two-cell faults can be only regular (-) or partial (p), while being hard, soft or transient. Note that some faults classes are considered unrealistic, such as activation partial (p_a) single-cell faults, and therefore they are excluded from the space. More info about these unrealistic faults and their analysis can be found in the literature [59, 61].

For example, a transition 0 fault can be hard (hTF_0), which is the same as the generic TF_0. It can also be partial hard (p_ihTF_0), dirty hard ($dhTF_0$) and partial dirty hard fault (p_idhTF_0). The same combinations apply for soft TF_0 and transient TF_0.

5.2 Fault model validation using Spice

In order to validate the theoretical framework of DRAM faults discussed in Section 5.1, an elaborate spice model based study has been performed based on real industrial memory simulation models, manufactured in a number of different technologies, ranging from 0.35 μm to 0.11 μm feature sizes. This section shows the simulation results performed on a memory model targeting 0.2 μm technology, by injecting resistive defects into the model and subsequently simulating them. This approach has been successfully used in the past to analyze the faulty behavior of both digital and analog devices [63, 64].

5.2.1 Memory simulation model

The model used here is based on a design validation Spice model. The model has been reduced in size to limit simulation time. Figure 5.3 shows the different parts of the memory model. The model has 3 BL pairs[1], one at the top (BTt and BCt), one in the middle (BTm and BCm), and one at the bottom (BTb and BCb). Each BL pair contains 2 functional memory cells, while the rest of the cells are replaced by capacitive and resistive loads. This gives a total of 6 memory cells, three of which are connected to a true BL and controlled by the word line WLt, while the other three are connected to the complement BL and controlled by WLc. The model also contains 3 sense amplifiers (SAs), 3 precharge circuits, access devices, one read buffer (to inspect the output of a read operation), and one write buffer (to enable writing the cells).

This is a representative scenario very close to the real memory devices that allows us to investigate the different layout level defects.

5.2.2 Classification of defects

Depending on the signal lines (where the injected defects are connected to) the defects may be classified into the following three main categories:

- **Open**—Opens represent unwanted resistances on a signal line that is supposed to conduct perfectly.
- **Short**—Shorts are undesired resistive paths between a signal line and power supply (V_{dd} or GND).

[1]Three BL pairs are used to simulate BL coupling effects, but these effects are not discussed further in this text [59].

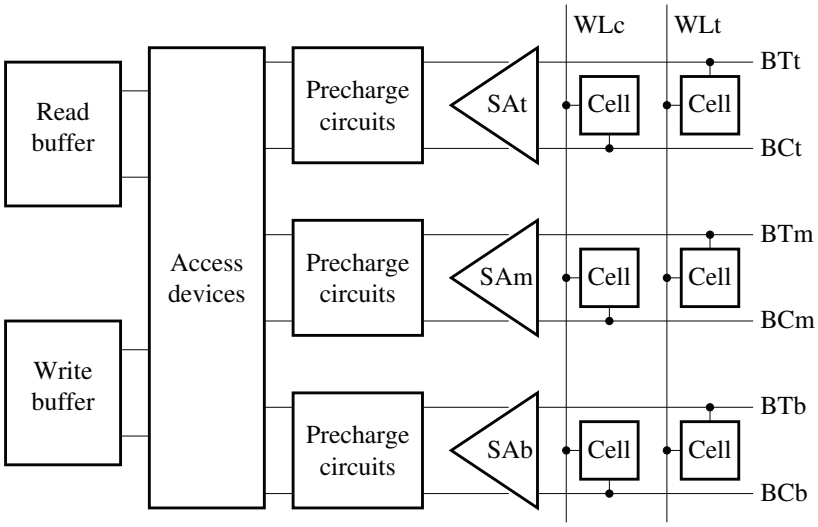


Figure 5.3: Reduced memory simulation model used for fault analysis.

- **Bridge**—Bridges are unwanted resistive paths between two signal lines.

The faulty behavior resulting from many defects can be deduced using symmetrical relationships with other defects. Therefore only symmetrically unrelated defects are simulated. Figure 5.4 shows an overview of the locations of the defects to be simulated. The defects are classified into opens, shorts and bridges. The opens and shorts are divided into defects in the memory cells, on BLs or on WLs, while bridges are divided into defects involving no cells, one cell and two different cells. Table 5.1 gives a summary of the simulated defects and the simulation results. In the following text we will focus on the physical location of opens, shorts and bridges.

Location of opens

At the layout level, opens are usually caused by broken lines, not properly connected vias, or particle contamination that results in increasing line resistivity at the open position. Figure 5.5 shows a layout level example of a BL open caused by particle contamination, resulting in an increase in the BL resistance and inducing some kind of faulty behavior in the memory.

Opens in the memory cell array can be either opens within cells (OC), opens on BLs (OB) or opens on WLs (OW). Figure 5.6 shows all possible electrical positions of these three different types of opens. In addition, there are three classes of OC, on top (t), in the middle (m) and at the bottom (b), all of which (partially) disconnect the cell from the BL and limit the ability of the

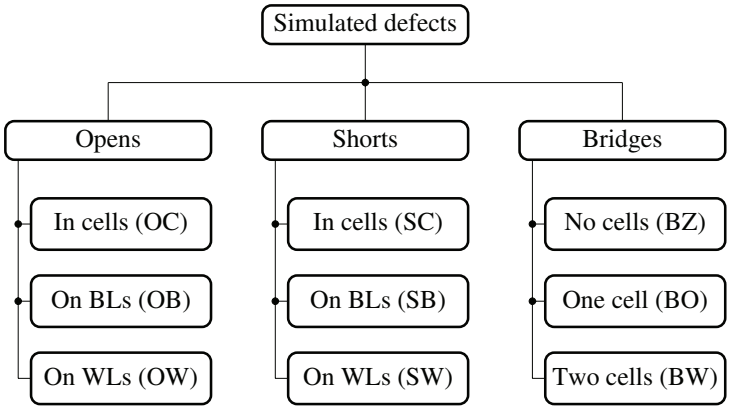


Figure 5.4: Overview of simulated defect locations.

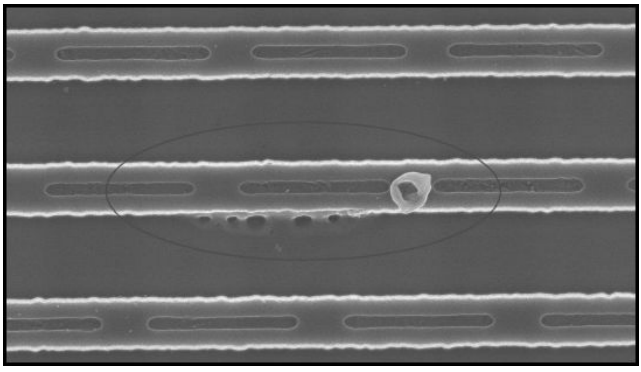


Figure 5.5: Layout level example of a BL open.

memory to write and read the voltage within the cell. There are two types of OB, disconnecting the cell from circuitry on the top of the BL (t), and disconnecting the cell from circuitry at the bottom of a the BL (b). There is only one word line open called OW, which limits the ability of the memory controller to properly access the cell under consideration.

Location of shorts

At the layout level, shorts can be caused by a number of physical failures, such as extra metal deposition or isolation breakdown. This may result in faulty connections between power supply lines and other signal lines in the memory. Figure 5.7 shows a layout example of a power supply short caused by extra metal deposition, resulting in a faulty new connection being formed between

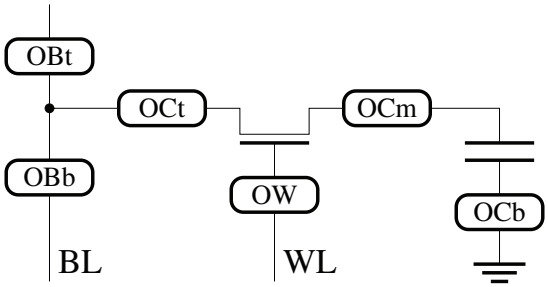


Figure 5.6: Location of opens in the cell array.

two otherwise disconnected lines.

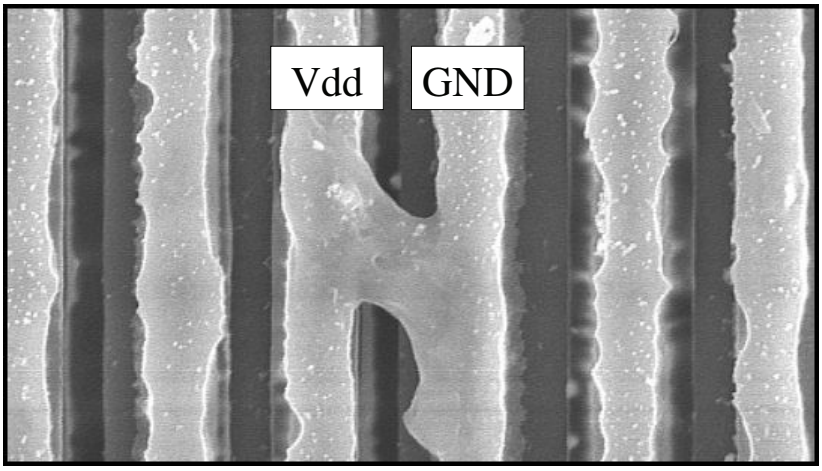


Figure 5.7: Physical example of power supply shorts.

Shorts, similar to opens, can be either within cells (SC), on bit lines (SB), or on word lines (SW). Figure 5.8 shows the short positions of these three different types of shorts. At each position indicated in the figure, a short may connect the line either to V_{dd} or GND. Shorts to V_{dd} are indicated by the letter (v) as in SCv and SBv, while shorts to GND are indicated by the letter (g) as in SCg and SBg. BL in the figure can either be the true bit line (BT), or the complementary bit line (BC).

Location of bridges

At the layout level, bridges can be the result of isolation layer misalignment or mask contamination, resulting in faulty connections between different lines

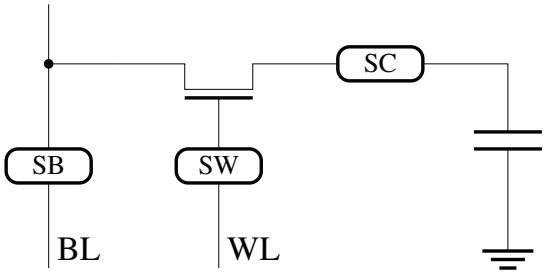


Figure 5.8: Positions of shorts in the cell array.

in the memory. Figure 5.9 shows a layout example of a bridge between the WL and the BL contacts caused by a misalignment in the isolation layer. The small black structure in the middle of the figure represents the WL contact, which is sandwiched between two relatively larger BL contacts. The WL and BL contacts are separated from each other by a thin isolation layer, that is difficult to manufacture. The figure shows that the isolation layer is not created properly, leaving a small space on top, through which the BL contact stretches and connects to the WL.

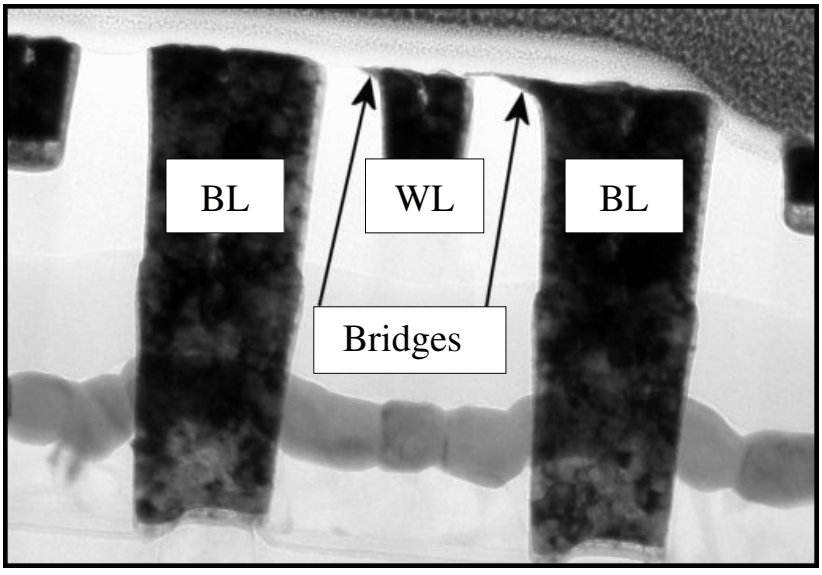


Figure 5.9: Physical example of a BL-WL bridge.

At the electrical level, bridges are resistive connections between two signal lines in the memory. In order to take all cell array bridges into consideration,

we need to consider bridges between any two nodes in the cell array. Since the cell array has a repetitive structure, it is possible to consider only the region surrounding a single cell in the array as a representative of the whole. Such a representative case is shown in Figure 5.10, where all nodes are given different names for later reference.

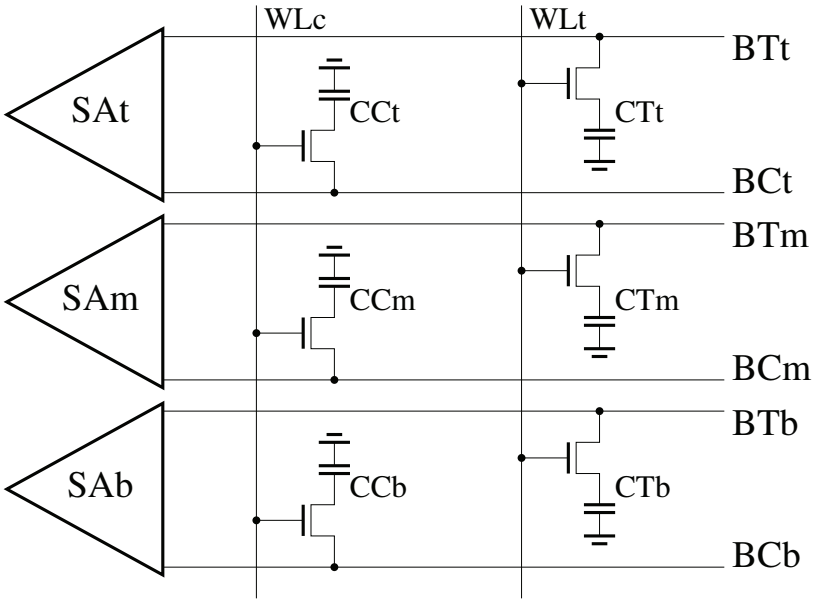


Figure 5.10: Nodes where bridges can take place.

The figure shows three BL pairs, at the top (BTt and BCt), in the middle (BTm and BCm), and at the bottom (BTb and BCb). Each BL pair has two memory cells connected to it, giving a total of six cells, three on BT (with nodes CTt, CTm and CTb) and three on BC (with nodes CCt, CCm and CCb). The cells connected to BT are controlled by the true WL (WLt), while the cells connected to BC are controlled by the complement WL (WLc). The BL lines are terminated by the inputs of the three sense amplifiers (SAt,SAm and SAb).

We can classify bridges into those involving zero cells (BZ), one cell (BO) and two cells (BW). BZ are bridges that connect signal lines outside memory cells, rather than connecting lines within memory cells. BO are bridges that connect nodes within a single memory cell. BW are bridges that connect nodes belonging to two different memory cells. A summary of the simulated defects is given in Table 5.1.

Table 5.1: Definitions of injected defects and the simulation results of the faulty behavior.

Defect (definition)	FP	h	ph	dh	pdh	s	ps	ds	pds	t	pt	dt	pdt	R_c
OCt (Pass transistor disconnected from BL)	$<w1^3w0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	331 kΩ
OCm (Pass transistor disconnected from C)	$<w1^3w0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	323 kΩ
OCb (Cell disconnected from GND)	$<w1^3w0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	323 kΩ
OBt (Cell disconnected from read/write circuits)	$<w0^2w1/0/->$	−	+	−	−	−	+	−	−	−	+	−	−	50 kΩ
OBb* (Cell disconnected from precharge devices*)	$<0w_v1^2[w_b0]/0/->$	−	+	+	+	−	+	+	+	−	+	+	+	8.5 kΩ
OW (WL disconnected from cell)	$<w0/1/->$	+	−	−	−	+	−	−	−	+	−	−	−	24.6 MΩ
SCv (Cell capacitor shorted to V_{dd})	$<0/1/->$	+	−	−	−	+	−	−	−	−	−	−	−	896 kΩ
SCg (Cell capacitor shorted to GND)	$<1/0/->$	+	−	−	−	+	−	−	−	−	−	−	−	693 kΩ
SBv (BL shorted to V_{dd})	$<w1^2w0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	250 kΩ
SBg (BL shorted to GND)	$<0w1/0/->$	+	−	−	−	+	−	−	−	+	−	−	−	190 kΩ
SWv (WL shorted to V_{dd})	$<0/1/->$	+	−	−	−	+	−	−	−	−	−	−	−	0.34 kΩ
SWg (WL shorted to GND)	$<w1^3w0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	8.29 kΩ
BO1 (Cell bridged to own WL)	$<w1^2w0/1/->$	−	+	−	−	−	−	−	−	−	+	−	−	357 kΩ
BO2 (Cell bridged to idle WL)	$<1/0/->$	+	−	−	−	+	−	−	−	−	−	−	−	693 kΩ
BO3 (Cell bridged to own BL)	$<0/1/->$	+	−	−	−	+	−	−	−	−	−	−	−	150 kΩ
BO4 (Cell bridged to own comp. BL)	$<0/1/->$	+	−	−	−	+	−	−	−	−	−	−	−	400 kΩ
BO5 (Cell bridged to another comp. BL)	$<0/1/->$	+	−	−	−	+	−	−	−	−	−	−	−	300 kΩ
BZ1 (Bridge between BTm and BCm)	$<w0/1/->$	+	−	−	−	+	−	−	−	+	−	−	−	1.71 kΩ
BZ2 (Bridge between BTm and BCt)	$<0w1/0/->$	+	−	−	−	+	−	−	−	+	−	−	−	4 kΩ
BZ3 (Bridge between BCm and BTb)	$<0w1/0/->$	+	−	−	−	+	−	−	−	+	−	−	−	0.6 kΩ
BZ4 (Bridge between BTm and WLt)	$<1w0/1/->$	+	−	−	−	+	−	−	−	+	−	−	−	270 kΩ
BZ5 (Bridge between BCm and WLt)	$<0w1/0/->$	+	−	−	−	+	−	−	−	+	−	−	−	250 kΩ
BZ6 (Bridge between WLt and WLc)	$<0w1/0/->$	+	−	−	−	+	−	−	−	+	−	−	−	0.15 kΩ
BW1 (Bridge between CTm and CCm)	$<w1^2;0/1/->$	−	+	−	−	−	+	−	−	−	+	−	−	3 MΩ
BW2 (Bridge between CTm and CTb)	$<w0^2w1/0/->$	−	+	−	−	−	+	−	−	−	+	−	−	460 kΩ

*This defect is injected and simulated in a memory model different from that shown in Figure 5.3. The model used here is shown in Figure 5.2(c), where the precharge circuits are located on one end of the BL, while read/write circuits are located on the other end. This BL design was common in older DRAM technologies [65].

5.3 Simulation results

The simulation and analysis of the faulty behavior of the DRAM has been done using the concept of result planes, as described in the literature [59, 49]. The simulation results for each defect are listed in Table 5.1. The first column in this table lists the name of the simulated defect, followed by the FP notation to describe the resulting faulty behavior in the third column. The following 12 columns indicate the type of the DRAM-specific fault being modeled by the FP. There are five basic types of faults: hard (h), soft (s), transient (t), partial (p) and dirty (d). Together, they combine to make up 12 meaningful fault combinations [see Section 5.1]. For example, (pt) stands for partial-transient fault type. Obviously, some combinations, e.g. (sh) (soft-hard), (ht) and (st), are not realistic. The R_c column depicts the value of the *critical resistance* of the fault, which is the defect resistance at which the fault begins to take effect. The second row of the first column provides a short description of the defect being simulated.

The table shows that *all* the DRAM-specific fault models defined in Section 5.1 do actually represent one defect or another. This indicates that DRAM-specific fault models are realistic and do take place in practice. The table also indicates that, in general, if a defect causes a hard fault to take place, then it might also cause a corresponding soft and transient fault depending on the direction and the strength of the leakage current flowing into the cell. This is however not always true, especially when the defect itself forces current to leak into the cell in a specific direction. This is the case, for example, with cell shorts and cell bridges to BLs and WLs.

Note that the least represented DRAM-specific faults in the table are the dirty (d) and the partial dirty (pd) faults. This is true because in order for these faults to take place, the BLs should be designed in the way shown in Figure 5.2(c), where the precharge circuits are located on one end of the BL, while the read/write circuits are located on the other end. This BL design was common in older DRAM technologies [65]. The simulation results for the defect OBb listed in Table 5.1 correspond to a simulation performed on such an old model that belongs to a 0.35 μm technology. The simulation results of the OBb defect injected into the 0.2 μm memory model used for the rest of table has not resulted in any faulty behavior.

The results in the table make it possible to optimize the test strategy of the memory depending on the DRAM design and the defects that take place there. If, for example, the structure of the BL is known to have both precharge circuits and read/write circuits located on the same side of the BL, then it is not needed to test for dirty faults in the memory at all.

5.4 Space of DRAM tests

In this section, we derive the space of DRAM tests for (single-cell as well as two-cell) hard, then transient, and finally soft faults. The tests are presented in their general form, without any optimization or reduction in test time.

5.4.1 Detecting hard faults

Hard faults are time-independent faults that get sensitized once their sensitizing operation sequence is performed, and they remain sensitized afterwards until they get overwritten (masked) or otherwise detected. Next, we discuss the detection conditions needed, followed by the corresponding tests.

Detection conditions for hard faults

A single-cell hard fault can either be partial with respect to initialization (p_ih), dirty (dh), or both ($p_i$dh). The fault p_ih is modeled by multiple initializing operations, while the fault dh is modeled by performing a write or read operation on a cell along the same BL as the faulty cell, but with opposite data to the sensitization.

Table 5.2 lists all single-cell hard faults, along with the detection conditions needed to detect them [compare with Table 3.1]. The table considers the general form of single-cell hard faults, where both partial, as well as dirty faults take place. For example, the (partial, dirty and hard) transition 0 fault ($p_i$dh TF_0), must first be initialized a multiple number of times ($w1_v^h$). Then, it should be sensitized by $w0$, before a completing operation with data 1 (a value opposite to that of the sensitizing value) must be applied to a different cell along the same BL ($[O1_b]$). The only requirement the completing write operation has to fulfill is to change the state of the BLs connected to the victim cell. The exact written cell address is therefore unimportant, only the fact that it lies along the same BL matters. The detection condition starts with multiple $w1$ operations to initialize the cell, followed by the sensitizing $w0$ operation on the victim. Then, the operation $O1_b$ ensures that the opposite data is present in a cell along the same BL just before the fault is detected by the read operation.

In a similar way, one can derive the detection conditions needed to detect all two-cell, hard faults presented in Table

Tests for hard faults

Based on the detection conditions of single-cell and two-cell hard faults , it is possible to derive memory tests that detect all single-cell and two-cell hard faults. March H1C (for hard, 1-cell) below detects all single-cell hard faults.

Table 5.2: Single-cell, hard FPs and their detection conditions. Ox_b is performed with a value (x) opposite to that in the sensitizing operation and to a cell (b) different from v, but along the same BL as v.

#	Fault	FP (b on same BL as v)	Detection cond., $O \in \{w, r\}$
1	dh SF_0	$<0_v[O1_b]/1/->$	$\Updownarrow(..w0, ..O1_b, ..r0..)$
2	dh SF_1	$<1_v[O0_b]/0/->$	$\Updownarrow(..w1, ..O0_b, ..r1..)$
3	$p_i$dh WDF_0	$<w0_v^h[O1_b]/1/->$	$\Updownarrow(..w0^h, ..O1_b, ..r0..)$
4	$p_i$dh WDF_1	$<w1_v^h[O0_b]/0/->$	$\Updownarrow(..w1^h, ..O0_b, ..r1..)$
5	$p_i$dh TF_1	$<w0_v^h w1_v[O0_b]/0/->$	$\Updownarrow(..w0^h, ..w1, ..O0_b, ..r1..)$
6	$p_i$dh TF_0	$<w1_v^h w0_v[O1_b]/1/->$	$\Updownarrow(..w1^h, ..w0, ..O1_b, ..r0..)$
7	$p_i$dh IRF_0	$<w0_v^h[O1_b]r0_v/0/1>$	$\Updownarrow(..w0^h, ..O1_b, ..r0..)$
8	$p_i$dh IRF_1	$<w1_v^h[O0_b]r1_v/1/0>$	$\Updownarrow(..w1^h, ..O0_b, ..r1..)$
9	$p_i$dh $DRDF_0$	$<w0_v^h r0_v[O1_b]/1/0>$	$\Updownarrow(..w0^h, ..r0, ..O1_b, ..r0..)$
10	$p_i$dh $DRDF_1$	$<w1_v^h r1_v[O0_b]/0/1>$	$\Updownarrow(..w1^h, ..r1, ..O0_b, ..r1..)$
11	$p_i$dh RDF_0	$<w0_v^h[O1_b]r0_v/1/1>$	$\Updownarrow(..w0^h, ..O1_b, ..r0..)$
12	$p_i$dh RDF_1	$<w1_v^h[O0_b]r1_v/0/0>$	$\Updownarrow(..w1^h, ..O0_b, ..r1..)$

March H1C = {
$\Updownarrow(w0^h, r0, w1_b, r0)$; $\Updownarrow(w1^h, r1, w0_b, r1)$;
M0 M1
$\Updownarrow(w0^h, w1, w0_b, r1)$; $\Updownarrow(w1^h, w0, w1_b, r0)$}
M2 M3

This march test has four march elements (M0 through M3), each of which begins with a hammer write operation and ends with a detecting read operation. Each two consecutive march elements represent the exact complement of each other, as they are generated to target complementary FPs. The test substitutes the dirty operation (O) in the detection conditions of Table 5.2 by a write operation, since this choice reduces the length of the test when the completing operation needs to change the data present in b. Note that the test uses a special kind of march operations (Ox_b), where an operation is performed to a different cell within a given march element. The test has a relatively high complexity of $(12 \cdot n + 4 \cdot h \cdot n)$ compared to other single-cell march tests, as a result of the partial and the dirty DRAM-specific faults. In addition, the test requires some knowledge about the exact memory layout needed for the Ox_b operations.

A march test that detects all two-cell hard faults can be represented by March H2C below.

March H2C = {
$\Updownarrow(w0^h)$; $\Uparrow(r0^h, w1^h)$; $\Uparrow(r1^h, w0^h)$;
M0 M1 M2
$\Downarrow(r0^h, w1^h)$; $\Downarrow(r1^h, w0^h)$; $\Updownarrow(r0)$}
M3 M4 M5

This march test has 6 march elements (M0 through M5), many of which begin with a hammer read operation and end with a hammer write operation. These sequences are characteristic for march tests that aim to detect two-cell faults [66]. The march element M1 is the exact complementary of M2, while M3 is the exact complementary of M4. This results from the fact that these march elements are constructed to detect complementary FPs. The sole purpose of M0 is to initialize the memory to a known state before the rest of the operations can be performed. This test has a complexity of $(n + 9 \cdot h \cdot n)$.

5.4.2 Detecting transient faults

Transient FPs mean that, after a fault is sensitized, leakage can result in correcting the fault before it gets detected. Hereafter we derive the tests needed to detect single-cell and two-cell transient faults.

Detection conditions for transient faults

An FP has two components to describe a fault: F (the value of the faulty cell) and R (the output on a read operation). Only F can be transient (get corrected by leakage), whereas R cannot, since it gets sensitized *and* detected on the output at the same time.

Table 5.3: List of single-cell, transient FPs and their detection conditions. The underlined operations must be performed back-to-back.

#	Fault	$<S/F_L/R>$, $O \in \{w, r\}$	Detection cond., $O \in \{w, r\}$
1	dt SF_0	$<0_v[\underline{O1}_b]/1_L/->$	$\Updownarrow(..., w0, \underline{O1}_b, \underline{r0}, ...)$
2	dt SF_1	$<1_v[\underline{O0}_b]/0_L/->$	$\Updownarrow(..., w1, \underline{O0}_b, \underline{r1}, ...)$
3	$p_i$dt WDF_0	$<\underline{w0}^h_v[\underline{O1}_b]/1_L/->$	$\Updownarrow(..., \underline{w0}^h, \underline{O1}_b, \underline{r0}, ...)$
4	$p_i$dt WDF_1	$<\underline{w1}^h_v[\underline{O0}_b]/0_L/->$	$\Updownarrow(..., \underline{w1}^h, \underline{O0}_b, \underline{r1}, ...)$
5	$p_i$dt TF_1	$<\underline{w0}^h_v\underline{w1}_v[\underline{O0}_b]/0_L/->$	$\Updownarrow(..., \underline{w0}^h, \underline{w1}, \underline{O0}_b, \underline{r1}, ...)$
6	$p_i$dt TF_0	$<\underline{w1}^h_v\underline{w0}_v[\underline{O1}_b]/1_L/->$	$\Updownarrow(..., \underline{w1}^h, \underline{w0}, \underline{O1}_b, \underline{r0}, ...)$
7	$p_i$dt IRF_0	$<\underline{w0}^h_v[\underline{O1}_b]\underline{r0}_v/0_L/1>$	$\Updownarrow(..., \underline{w0}^h, \underline{O1}_b, \underline{r0}, ...)$
8	$p_i$dt IRF_1	$<\underline{w1}^h_v[\underline{O0}_b]\underline{r1}_v/1_L/0>$	$\Updownarrow(..., \underline{w1}^h, \underline{O0}_b, \underline{r1}, ...)$
9	$p_i$dt $DRDF_0$	$<\underline{w0}^h_v\underline{r0}_v[\underline{O1}_b]/1_L/0>$	$\Updownarrow(..., \underline{w0}^h, \underline{r0}, \underline{O1}_b, \underline{r0}, ...)$
10	$p_i$dt $DRDF_1$	$<\underline{w1}^h_v\underline{r1}_v[\underline{O0}_b]/0_L/1>$	$\Updownarrow(..., \underline{w1}^h, \underline{r1}, \underline{O0}_b, \underline{r1}, ...)$
11	$p_i$dt RDF_0	$<\underline{w0}^h_v[\underline{O1}_b]\underline{r0}_v/1_L/1>$	$\Updownarrow(..., \underline{w0}^h, \underline{O1}_b, \underline{r0}, ...)$
12	$p_i$dt RDF_1	$<\underline{w1}^h_v[\underline{O0}_b]\underline{r1}_v/0_L/0>$	$\Updownarrow(..., \underline{w1}^h, \underline{O0}_b, \underline{r1}, ...)$

Table 5.3 lists all single-cell transient faults, along with their detection conditions [compare with Table 5.2]. For example, the (partial, dirty and transient) transition 0 fault ($p_i$dt TF_0), must first be initialized by multiple number of operations ($w1^h$). Then, the sensitizing write 0 operation can be performed ($w0$), before a completing operation with data 1 must be applied to a different cell

along the same BL as v ($[O1_b]$). The detection condition starts with multiple $w1$ operations to initialize the cell to 1, directly followed by the sensitizing $w0$, the completing $O1_b$, and a detecting $r0$. Note that this detection condition is not a regular one, since it requires operations to be performed on two different cells (b and v) within a single march element. The fact that the operations in these detection conditions need to be performed back-to-back is indicated by the underline below the corresponding operations.

In a similar way as above, one may derive the detection conditions corresponding to all two-cell, hard faults.

Tests for transient faults

Based on the detection conditions of single-cell and two-cell transient faults, it is possible to derive memory tests that detect all these faults. A march test that detects all single-cell transient faults is March T1C (for transient, 1-cell) as presented below.

March T1C = {

$\Updownarrow(\underline{w0}^h, \underline{w1}_b, \underline{r0});$	$\Updownarrow(\underline{w1}^h, \underline{w0}_b, \underline{r1});$
M0	M1
$\Updownarrow(\underline{w0}^h, \underline{w1}, \underline{w0}_b, \underline{r1});$	$\Updownarrow(\underline{w1}^h, \underline{w0}, \underline{w1}_b, \underline{r0});$
M2	M3
$\Updownarrow(\underline{w0}^h, \underline{r0}, \underline{w1}_b, \underline{r0});$	$\Updownarrow(\underline{w1}^h, \underline{r1}, \underline{w0}_b, \underline{r1})\}$
M4	M5

This march test has six march elements (M0 through M5), each of then begins with a hammer write operation and ends with a detecting read operation. This test has a complexity of $(16 \cdot n + 6 \cdot h \cdot n)$. The march elements have special operations (such as $w1_b$) that are performed on a different cell along the same BL as the current cell under test of the march element. The operations in each march element must be performed back-to-back directly after each other (denoted by the underline below the operations in the test).

A march test that detects all two-cell, transient faults is March T2C presented hereafter.

March T2C = {

$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0}^h_j, \underline{r0}_i, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1}^h_j, \underline{r0}_i, \underline{r0}_i));$
M0	M1
$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0}^h_j, \underline{r1}_i, \underline{r1}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1}^h_j, \underline{r1}_i, \underline{r1}_i));$
M2	M3
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0}_j, \underline{w1}^h_j, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0}_j, \underline{w1}^h_j, \underline{r1}_i));$
M4	M5
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1}_j, \underline{w0}^h_j, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1}_j, \underline{w0}^h_j, \underline{r1}_i));$
M6	M7
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0}_j, \underline{r0}^h_j, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0}_j, \underline{r0}^h_j, \underline{r1}_i));$
M8	M9
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1}_j, \underline{r1}^h_j, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1}_j, \underline{r1}^h_j, \underline{r1}_i));$
M10	M11
$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w0}^h_j, \underline{w0}_i, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(w0_i, \underline{w1}^h_j, \underline{w0}_i, \underline{r0}_i));$
M12	M13
$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0}^h_j, \underline{w1}_i, \underline{r1}_i));$	$\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w1}^h_j, \underline{w1}_i, \underline{r1}_i));$
M14	M15
$\Updownarrow_i(\Updownarrow_j(\underline{w1}^h_i, \underline{w0}^h_j, \underline{w0}_i, \underline{r0}_i));$	$\Updownarrow_i(\Updownarrow_j(\underline{w1}^h_i, \underline{w1}^h_j, \underline{w0}_i, \underline{r0}_i));$
M16	M17
$\Updownarrow_i(\Updownarrow_j(\underline{w0}^h_i, \underline{w1}^h_j, \underline{w1}_i, \underline{r1}_i));$	$\Updownarrow_i(\Updownarrow_j(\underline{w0}^h_i, \underline{w0}^h_j, \underline{w1}_i, \underline{r1}_i))\}$
M18	M19

This test has 20 march elements (M0 through M19), each of them containing a *nested march element*. This test has a complexity of $(56 \cdot n^2 + 24 \cdot h \cdot n^2)$. The reason behind the high computational complexity is the assumption that an aggressor can cause a fault in any victim anywhere in the memory. This assumption is, however, unrealistic. The impact of an aggressor is almost always limited to the adjacent neighboring cells for transient faults. This observation can significantly simplify March T2C, by limiting the value of j to a limited number of adjacent cells. This will reduce the complexity of the test from quadratic to linear with the number of cells.

5.4.3 Detecting soft faults

Soft FPs mean that a correct (but weak) voltage in the cell can gradually be depleted and cause a detectable fault in the cell only after some period of time. In this section, we start by discussing the detection conditions needed to detect all soft faults, and then we generate memory tests for these soft faults.

Detection conditions for soft faults

An FP has two components to describe a fault: F and R. Only F can cause soft faults (fail after some time), whereas R cannot be soft, since its value is read at moment read operation is performed.

Table 5.4 lists all single-cell soft faults, along with the detection conditions needed to detect them. This table is easy to construct based on the detection conditions in Table 5.2, by introducing a delay time T after every sensitizing

Table 5.4: List of single-cell, soft FPs and their detection conditions. Ox_b is performed with a value (x) opposite to that in the sensitizing operation and to a cell (b) different from v, but along the same BL as v.

# Fault	FP (b belongs to BL of v)	Detection cond., $O \in \{w, r\}$
1 ds SF_0	$<0_v[O1_b]_T/1/->$	$\Updownarrow(..w0, ..O1_b, ..T, ..r0..)$
2 ds SF_1	$<1_v[O0_b]_T/0/->$	$\Updownarrow(..w1, ..O0_b, ..T, ..r1..)$
3 $p_i$ds WDF_0	$<w0^h_v[O1_b]_T/1/->$	$\Updownarrow(..w0^h, ..O1_b, ..T, ..r0..)$
4 $p_i$ds WDF_1	$<w1^h_v[O0_b]_T/0/->$	$\Updownarrow(..w1^h, ..O0_b, ..T, ..r1..)$
5 $p_i$ds TF_1	$<w0^h_v w1_v[O0_b]_T/0/->$	$\Updownarrow(..w0^h, ..w1, ..O0_b, ..T, ..r1..)$
6 $p_i$ds TF_0	$<w1^h_v w0_v[O1_b]_T/1/->$	$\Updownarrow(..w1^h, ..w0, ..O1_b, ..T, ..r0..)$
7 $p_i$ds IRF_0	$<w0^h_v[O1_b]r0_v/0/1>$	$\Updownarrow(..w0^h, ..O1_b, ..r0..)$
8 $p_i$ds IRF_1	$<w1^h_v[O0_b]r1_v/1/0>$	$\Updownarrow(..w1^h, ..O0_b, ..r1..)$
9 $p_i$ds $DRDF_0$	$<w0^h_v r0_v[O1_b]_T/1/0>$	$\Updownarrow(..w0^h, ..r0, ..O1_b, ..T, ..r0..)$
10 $p_i$ds $DRDF_1$	$<w1^h_v r1_v[O0_b]_T/0/1>$	$\Updownarrow(..w1^h, ..r1, ..O0_b, ..T, ..r1..)$
11 $p_i$ds RDF_0	$<w0^h_v[O1_b]r0_v/1/1>$	$\Updownarrow(..w0^h, ..O1_b, ..r0..)$
12 $p_i$ds RDF_1	$<w1^h_v[O0_b]r1_v/0/0>$	$\Updownarrow(..w1^h, ..O0_b, ..r1..)$

operation. Note that the detection conditions for soft IRF and RDF do not include the T, since these faults are detected as soon as they get sensitized. As an example, the (partial, dirty and soft) transition 0 fault ($p_i$ds TF_0), must first be initialized a multiple number of times ($w1^h$). Then, the sensitizing write 0 operation can be performed ($w0$), before a completing operation with data 1 is applied to a different cell along the same BL ($[O1_b]$). To ensure the detection of this soft fault, a delay time T must be introduced after the completing operation to allow for sensitization to take place.

In the same way, one can derive the detection conditions corresponding to all two-cell, soft faults.

Tests for soft faults

Based on the detection conditions of single and two-cell soft faults, it is possible to derive memory tests that detect all these faults. A march test that detects all single-cell soft faults can have the form of March S1C below (for all soft, single-cell faults).

March S1C = {
$\Updownarrow(w0^h, r0, w1_b, T, r0)$; $\Updownarrow(w1^h, r1, w0_b, T, r1)$;
M0 M1
$\Updownarrow(w0^h, w1, w0_b, T, r1)$; $\Updownarrow(w1^h, w0, w1_b, T, r0)\}$
M2 M3

This march test has 4 march elements (M0 through M3). This test is similar to the test for hard single-cell DRAM faults (March H1C), which is

expected since the space of soft faults is derived from the space of hard faults. The test uses special nonstandard march elements, where one operation should be performed on a cell b different from the current cell the march element is accessing. This ensures that the BL has the opposite state as compared to the sensitized cell. The test has a complexity of $(12 \cdot n + 4 \cdot h \cdot n + 4 \cdot T \cdot n)$, which is higher than H1C by $(4 \cdot T \cdot n)$. If retention time of $T > 64$ ms and memory size of 1 MBits are assumed, the total idle test time is considered impractical. It is possible to reduce the idle time to as low as $4 \cdot T$ using specific read/write sequences and data backgrounds (such as the checkerboard). It is also important to implement *design-for-testability* (*DFT*) techniques, or use test stresses that force soft faults to become directly detectable hard faults, which in turn do not require any delay time to detect [59, 67].

A march test that detects all two-cell soft faults can be represented by March S2C below.

March S2C = {		
$\Updownarrow(w0^h)$;	$\Uparrow(r0^h, T, r0, w1^h)$;	
M0	M1	
$\Uparrow(r1^h, T, r1, w0^h)$;	$\Downarrow(r0^h, T, r0, w1^h)$;	
M2	M3	
$\Downarrow(r1^h, T, r1, w0^h)$;	T;	$\Updownarrow(r0)\}$
M4	M5	M6

This march test has 7 march elements (M0 through M6). It is based on March H2C for hard faults, with a number of added delays T to a number of march elements in the test (sometimes separating one read operation from a hammer read sequence). Note that M5 is simply a single delay T added to sensitize the faults before the final detecting read operations are performed in M6. This test has a time complexity of $[5 \cdot n + 9 \cdot h \cdot n + (4 \cdot n + 1)T]$, which is prohibitively expensive due to the $(4 \cdot n + 1)T$ term. Restricting the impact of every aggressor to a limited number of adjacent victim cells reduces the complexity of this term to $(4 \cdot j + 1)T$, where j is the number of adjacent victims (typically $3 \leq j \leq 8$). Even with this optimization, this test remains very costly, and a number of additional test time reduction methods need to be implemented, such as test stresses, which force soft faults to become directly detectable hard faults [59, 67].

5.5 Industrial support

The tests presented in this chapter are being industrially evaluated at the moment. In the mean time, it is possible to use previously published work to illustrate the practical strength of our tests. One of the rather effective DRAM tests investigated in the literature is the Gal9R = $\{\Uparrow(w0); \Uparrow_i(w1_i, \Uparrow_j(r0_j, r1_i), w0_i); \Uparrow(w1); \Uparrow_i(w0_i, \Uparrow_j(r1_j, r0_i), w1_i)\}$. In an experiment of 5952 different

tests performed on 800 DRAMs, this tests proved to be far better than any other memory test known in the literature [60].

According to the DRAM test theory presented in this chapter, Gal9R can simply be considered as a reduced form of march elements M9 and M10 of March T2C. More precisely, M9 $=\Updownarrow_i(\Updownarrow_j(w1_i, \underline{w0}_j, \underline{r0}^h_j, \underline{r1}_i))$ detects $\mathrm{p_a t}$ $\mathrm{CFds}_{0r0;1} = <0\underline{r0}^h; 1/0_L/->$, where the initialization of the victim to 1 ($w1_i$) and the aggressor to 0 ($\underline{w0}_j$) are considered to be *transient* and have to be performed back-to-back directly before any other operations. If we assume this not to be necessary, and only the final detecting read operation ($\underline{r1}_i$) has to be performed directly after the sensitizing read ($\underline{r0}^h_j$), then M9 can be rewritten as follows $\Updownarrow(w0)$; $\Updownarrow_i(w1_i, \Updownarrow_j(r0^h_j, \underline{r1}_i), w0_i)$. This form of M9 is equal to the first half of Gal9R (taking $h = 1$). The same transformation is also true for M10, which yields the second half of Gal9R.

In other words, the new DRAM test space can theoretically explain the underlying strength of Gal9R, by indicating that it is nothing more than a test that targets a simplified form of the following two partial transient coupling faults: $\mathrm{p_a t}$ $\mathrm{CFds}_{0r0;1} = <0\underline{r0}^h; 1/0_L/->$ and $\mathrm{p_a t}$ $\mathrm{CFds}_{1r1;0} = <1\underline{r1}^h; 0/1_L/->$.

5.6 Conclusions

This chapter presented the first DRAM-specific test space needed to test for all DRAM-specific fault models. Six different tests have been derived for (both single as well as two-cell) hard, transient and soft DRAM faults. It was also shown how the new DRAM test space can be used to explain the strength of some well-known effective DRAM tests in practice. The chapter also discussed the results of an elaborate simulation-based fault analysis experiment to validate the new memory faults. The results show that *all* newly proposed DRAM faults are realistic. The results also indicate that dirty faults are only possible in older DRAM designs.

Chapter 6

DRAM tests optimizations

Test development for DRAM has commonly been an ad hoc activity, where large numbers of tests are performed on a representative sample and the best performing (in terms of fault coverage) tests are chosen [60]. This approach results in a long and costly test development time, and eventually leads to a non-optimal test flow [58]. The theoretical framework of DRAM tests was proposed in Chapter 5 (see also [68]), based on Spice simulations of DRAM models, that is able to detect all possible DRAM faults [61]. Some of the proposed tests, however, are rather complex, such that they scale super-linearly with the number of cells in the memory under test.

Transient faults are DRAM faults that get temporarily sensitized, and subsequently correct themselves after a limited amount of time [69]. In order to detect these faults properly, the detecting operations should follow the sensitizing operations directly without any delay, before the fault gets corrected. This detection requirement results in a quadratic $O(n^2)$ dependency of test complexity on the number of memory cells. Such test complexity requires a extremely long test application time on current day high-density DRAMs, that render them impractical for a high-volume manufacturing test environment.

This chapter tackles the high complexity issue of DRAM tests generated to detect transient faults. The chapter suggests a number of optimization techniques to reduce test complexity and limit test application time. The concept of physical locality of defects is used, which states that a given defect can only cause a fault in a number of adjacent cells, rather than cells located far apart. The used optimization techniques reduce the test complexity from quadratic to linear, making it suitable for industrial application.

In addition to the above, this chapter discusses a number of methods to optimize the test length of soft fault tests to make them more applicable in practice. Design based optimizations can reduce the length of single-cell tests from linear to constant with respect to the retention time in the memory. The same is true for two-cell tests, where the topological location of the cells with

respect to each other on the layout is used to reduce the test length.

This chapter is organized as follows. Section 6.1 provides test time optimizations for transient faults tests. In Section 6.2 the test length of tests targeting soft faults is evaluated. Finally, Section 6.3 ends with the conclusions.

6.1 Optimizing transient faults tests

Assuming that a test designer has no knowledge of the internal structure of the memory under test, then the most general form of the tests is needed to be able to detect every possible fault that may take place in the memory. It is possible, however, to reduce the test time of the memory by restricting the complexity of memory tests based on information from the design and the layout, as discussed in this section.

The high time complexity of the two-cell march test (March T2C) for transient faults presented in Chapter 5 stems from the assumption that each cell can be coupled to all other cells in the memory. This assumption is unnecessary, since practically a cell can only be coupled to the closest physically neighboring cells on the layout. Once the layout of the memory is known, it is possible to significantly reduce the time complexity of these tests.

The most widely used DRAM layout today is shown in Figure 6.1 [70]. In this figure, the circles represent memory cells, the horizontal lines represent word lines (WLs), while the vertical lines represent bit lines (BLs). BLs are organized in pairs of true (BT) and complementary (BC) bit lines. Note that the order of the WLs is scrambled (so-called *reflected WL organization*), such that WL3 follows WL1 instead of WL2.

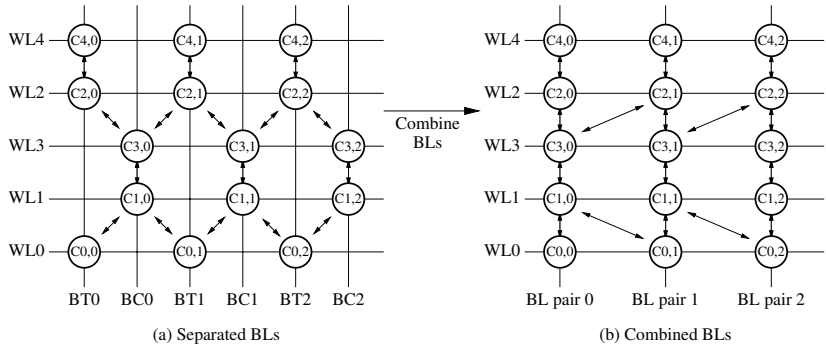


Figure 6.1: Physically neighboring cells (a) on the layout, and (b) with combined BLs (i.e., by combining BT and BC into a single BL pair).

Each memory cell in the figure is indicated by the letter C and a couple of numbers that refer to the WL and BL the cell is connected to. For example, the

cell C3,2 is the memory cell connected to WL3 and BL2. All even numbered WLs (such as WL0, WL2, etc.) access cells connected to BT, while all odd numbered WLs access cells connected to BC. Considering C1,1, for example, the three closest neighboring cells on the layout are C0,1, C0,2 and C3,1.

According to this layout, each cell has three closest physical neighbors. This situation is shown in Figure 6.1(a), where the closest neighbors are highlighted by arrows that connect between them.

When march tests are applied to the memory under test, memory cells are accessed in an increasing, or a decreasing logical address order. In the memory shown in Figure 6.1(a), an increasing logical cell address corresponds for example to the following cell sequence C0,0, C1,0, C2,0, C3,0, C4,0, (then the rest of WLs are accessed), then C1,0, C1,1, etc. In this example, a march test accesses a cell on BT first, then a cell on BC, then again on BT, then BC, etc., according to their logical address and not to their physical position. From a march test point of view (i.e., using logical addressing), there is no difference between a cell connected to BT or to BC. Therefore, it is possible to combine each BT and BC of a given BL pair to inspect the way cell neighbors are organized from a march test point of view. This is done in Figure 6.1(b), which combines each BTx and BCx in Figure 6.1(a) into a single BL pair x line.

In order to identify all logically neighboring cells from Figure 6.1(b), it is only necessary to swap WL2 and WL3, so that a logically ordered WL sequence can be obtained. This is done in Figure 6.2. This figure shows clearly each cell and its closest neighbors, derived from the physical cell proximity in Figure 6.1(a).

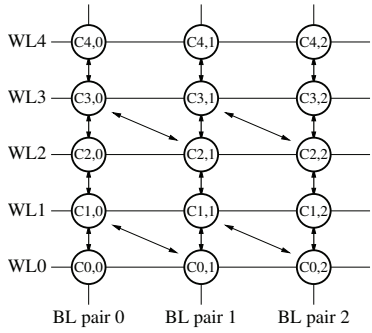


Figure 6.2: Logically neighboring cells.

Based on the information presented in Figure 6.2, it is possible to introduce realistic localized versions of a number of march tests discussed in Chapter 5.

Using this layout information, it is possible to reduce the complexity of March T2C to a more optimized test (called March T2C$_{layout}$), which has a

linear complexity rather than quadratic, by limiting the possible cells that could act as aggressors to only the three cells connected by an arrow in Figure 6.2. The localized version of the test is given below.

March T2C$_{layout}$ = {

$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}^h_j, \underline{r0}_i, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}^h_j, \underline{r0}_i, \underline{r0}_i))$;
M0	M1
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}^h_j, \underline{r1}_i, \underline{r1}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}^h_j, \underline{r1}_i, \underline{r1}_i))$;
M2	M3
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j, \underline{w1}^h_j, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j, \underline{w1}^h_j, \underline{r1}_i))$;
M4	M5
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j, \underline{w0}^h_j, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j, \underline{w0}^h_j, \underline{r1}_i))$;
M6	M7
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}_j, \underline{r0}^h_j, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}_j, \underline{r0}^h_j, \underline{r1}_i))$;
M8	M9
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}_j, \underline{r1}^h_j, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}_j, \underline{r1}^h_j, \underline{r1}_i))$;
M10	M11
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w0}^h_j, \underline{w0}_i, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w0_i, \underline{w1}^h_j, \underline{w0}_i, \underline{r0}_i))$;
M12	M13
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w0}^h_j, \underline{w1}_i, \underline{r1}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(w1_i, \underline{w1}^h_j, \underline{w1}_i, \underline{r1}_i))$;
M14	M15
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w1}^h_i, \underline{w0}^h_j, \underline{w0}_i, \underline{r0}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w1}^h_i, \underline{w1}^h_j, \underline{w0}_i, \underline{r0}_i))$;
M16	M17
$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w0}^h_i, \underline{w1}^h_j, \underline{w1}_i, \underline{r1}_i))$;	$\Updownarrow_i(\Updownarrow_{j=\Delta_i}(\underline{w0}^h_i, \underline{w0}^h_j, \underline{w1}_i, \underline{r1}_i))\}$
M18	M19

The Δ_i in the test refers to the three cells in the neighborhood of the cell i, as shown by the cells connected with an arrow in Figure 6.1(a). This test is similar to March T2C, with the exception that the parameter j does not run through all memory cells, but only the neighboring cells to i. As an example of the test, march element M0 can be understood as follows.

- For every the cell i in the memory start by initializing the cell to 0.
- Directly afterwards, write 0 an h number of times into every cell in the neighborhood of i.
- Directly afterwards, read 0 twice from cell i.

This test has 20 march elements, each of which has a nested march element. This test has a complexity of $(168 \cdot n + 72 \cdot h \cdot n)$, which is of the order $O(n)$. As an example, the complexity of M0 can be calculated as follows: $3 \cdot n(3 + h)$, or # of cells $\times$ # of neighbors per cell $\times$ (# of single operations + # of h repeated operations). Comparing this test with March T2C shows that the complexity of the test was significantly reduced, which makes the test much more suitable for industrial application.

6.2 Optimizing test length of soft faults tests

The tests derived in Section 5.4.3 assume the worst case faulty behavior possible in the memory. In such a situation, the test engineer has no knowledge of

the internal structure, the design or the layout of the memory under test. Therefore, all possible faults should be tested for in order to ensure the best fault coverage. When the internal structure is known, this information can be used to reduce the complexity of the used tests. In this section, we first discuss using electrical design information, then layout information to reduce the tests.

6.2.1 Memory design consideration

The example of dirty faults discussed in Section 5.1.2 shows that, in order for dirty faults to take place, the precharge circuitry should be located on one side of the BL pair, while the sense amplifiers located on the other side [see Figure 5.2(c)]. The memory can, however, be designed in two different ways.

Table 6.1: Different positions of the SA and PR on either side of a BL pair.

#	Upside		Downside		Description
	SA	PR	SA	PR	
1	t	t	b	b	SA and PR on same side of BL
2	t	b	b	t	SA on one side, PR on other

Table 6.1 lists the 4 different combinations of positions for the sense amplifiers (SA) and the precharge circuit (PR) on either side of a BL pair. An entry "t" means that the corresponding circuitry is located at the top of the BL pair, while an entry "b" means that the corresponding circuitry is located at the bottom of the BL pair. Since the top and the bottom of a BL pair are symmetrical, only 2 combinations in Table 6.1 are unique, while the other 2 combinations can be derived by viewing the BL pair from the upside or from the downside.

For example, Figure 6.3 shows one possible position allocation of the SA and PR circuitry. In this configuration, the SA is located on one side of the BL pair and the PR is located on the other side. Considering the upside of the BL pair, the SA is located at the top while the PR is located at the bottom. Considering the downside of the BL pair, the PR is located at the top while the SA is located at the bottom. Obviously, both of these BL organizations are identical.

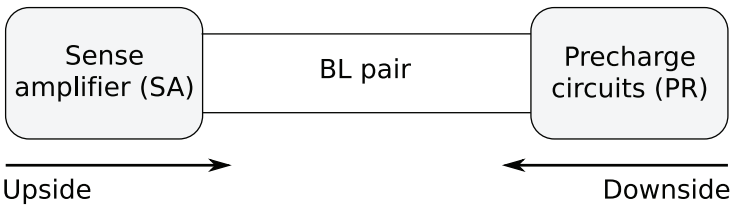


Figure 6.3: Upside and downside symmetry of BL pairs.

Dirty faults only take place when SA and PR are located on different ends of the BL, which is combination #2 in Table 6.1. However, most DRAMs today are designed with SA and PR located on the same side of the BL (combination #1 in Table 6.1), in which case dirty faults are not possible [59]. In other words, this organization reduces the space of single-cell memory faults exhibited as a result of all possible defects from the one described in Expression 5.1 to the following one.

$$\text{Single-cell fault} = \left\{ \begin{array}{c} \text{-} \\ \text{p}_\text{i} \end{array} \right\} \left\{ \begin{array}{c} \text{h} \\ \text{s} \\ \text{t} \end{array} \right\} \text{FP} \tag{6.1}$$

The reduction in the space of single-cell faults implies a corresponding simplification in the tests for these faults, since dirty faults need not be tested for. March S1C$_{part}$ ("*part*" for "only partial") below detects all single-cell soft faults for this special case.

March S1C$_{part}$ = {

$\Updownarrow(w0^h, r0);$	$T;$	$\Updownarrow(r0);$	$\Updownarrow(w1^h, r1);$	$T;$	$\Updownarrow(r1);$
M0	M1	M2	M3	M4	M5
$\Updownarrow(w0^h, w1);$	$T;$	$\Updownarrow(r1);$	$\Updownarrow(w1^h, w0);$	$T;$	$\Updownarrow(r0)\}$
M6	M7	M8	M9	M10	M11

This march test has twelve march elements (M0 through M11). The test can be derived from March S1C by eliminating the operations with the subscript b (meant to detect dirty faults) and separating the delay time T from the march element. Each three consecutive march elements in this test correspond to a single march element in March S1C (for example, M0 in March S1C corresponds to M0, M1 & M2 in March S1C$_{part}$. The test has a complexity of $(8 \cdot n + 4 \cdot h \cdot n + 4 \cdot T)$. When compared to March S1C this test replaces the highly expensive term $(4 \cdot T \cdot n)$ with the term $(4 \cdot T)$, which is constant with respect to the number of cells in the memory. This makes this test more suitable for industrial applications.

6.2.2 Memory layout consideration

March S2C is designed to detect two-cell soft faults. The high complexity of S2C, represented by the term $(4 \cdot n + 1)T$, is caused by the assumption that each cell in the memory can be influenced by every other cell in the whole array. This assumption is not realistic. Cells are mainly influenced by their closest neighbors according to the layout of the memory.

The most widely used DRAM layout today is shown in Figure 6.4. The circles represent the cells in the memory, the horizontal lines represent the word lines, while the vertical lines represent the bit lines. According to this layout, each cell has three closest physical neighbors [70]. This situation is shown in the figure, where the closest neighbors of each cell are highlighted by arrows that connect between them.

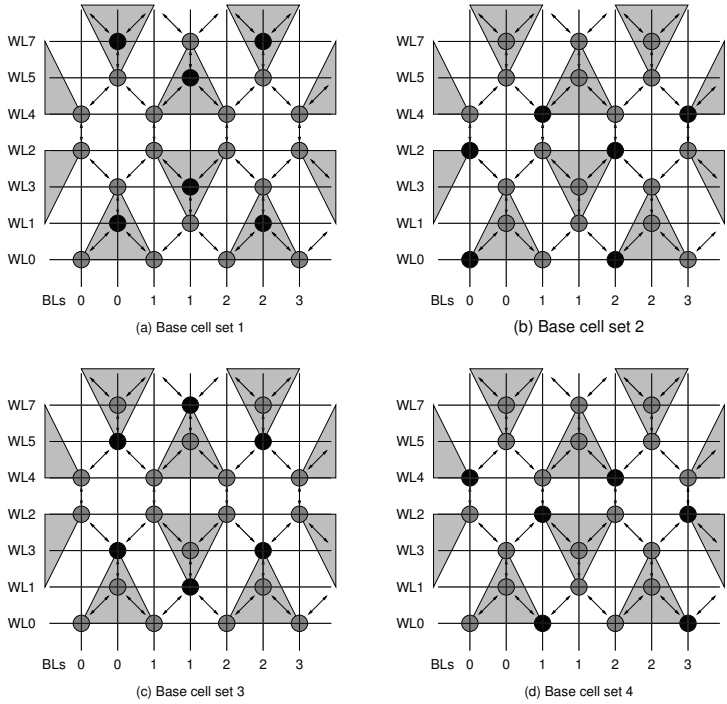


Figure 6.4: Sets of base cells used to accelerate soft fault testing.

Figure 6.4 shows how layout information can be used to accelerate soft fault testing. First, we need to select a set of memory cells that do not influence each other and consider them as **base cells** (each of which is referred to as b) for the test. The base cells are shown in Figure 6.4(a) as black circles. The memory cells that each base cell can influence is called its **neighborhood**, which can be denoted as ($\triangle_b$). Then the memory test can be performed as follows:

1. Select the first set of base cells as indicated by black dots in Figure 6.4(a)
   (a) Initialize base cells to a specific value.
   (b) Access each base cell and its neighborhood to sensitize the fault.
   (c) Wait for idle time T.
   (d) Detect the sensitized faults.
2. Repeat the operations in item 1 for each cell in the neighborhood by considering it as a base cell in itself (as shown in Figures 6.4(b), (c) and (d))

In the memory organization shown in Figure 6.4, there are four possible sets of base cells, which means that the idle time T needed for the test can be

reduced from $(4 \cdot n + 1)T$ to a constant idle time, which is independent of the number of cells in the memory. An optimized test that can detect two-cell soft faults is March C2S$_{\text{layout}}$, as shown below.

March S2C$_{layout}$ = {
$\Updownarrow_i(\Updownarrow_{b_i}(w1^h, w0), \Updownarrow_{b_i}(\Updownarrow_{\triangle_b}(w1, w0^h, r0^h)), \Updownarrow_{b_i}(r0), T, \Updownarrow_{b_i}(r0))$;
M0
$\Updownarrow_i(\Updownarrow_{b_i}(w1^h, w0), \Updownarrow_{b_i}(\Updownarrow_{\triangle_b}(w1^h, r1^h)), \quad \Updownarrow_{b_i}(r0), T, \Updownarrow_{b_i}(r0))$;
M1
$\Updownarrow_i(\Updownarrow_{b_i}(w0^h, w1), \Updownarrow_{b_i}(\Updownarrow_{\triangle_b}(w0, w1^h, r1^h)), \Updownarrow_{b_i}(r1), T, \Updownarrow_{b_i}(r1))$;
M2
$\Updownarrow_i(\Updownarrow_{b_i}(w0^h, w1), \Updownarrow_{b_i}(\Updownarrow_{\triangle_b}(w0^h, r0^h)), \quad \Updownarrow_{b_i}(r1), T, \Updownarrow_{b_i}(r1))\}$
M3

This test has four march elements divided into two complementary parts: M0 and M1 versus M2 and M3. The test divides the memory into sets of base cells, and runs through them one by one using the parameter $i = 0, 1, ...$ In the memory example of Figure 6.4, there are four sets of base cells, which means that i runs from 0 to 3. M0 in the test goes through each set of base cells using ($\Updownarrow_i$) and starts by initializing all base cells ($\Updownarrow_{b_i}$) to 0 using $w1^h, w0$. Then, M0 goes through every aggressor ($\Updownarrow_{\triangle_b}$) of each base cell ($\Updownarrow_{b_i}$) and sensitizes the fault in the base cell by performing $(w1, w0^h, r0^h)$. Then, a sensitizing read operation is performed on every base cell to sensitize all deceptive read disturb coupling faults ($\Updownarrow_{b_i}(r0)$), followed by idle time and a detecting read operation. M1 has the same structure, while M2 and M3 are the complementary of M0 and M1.

The complexity of this test can be calculated as follows $(4 \cdot n(1 + h) + 2 \cdot 3n(1 + 2h) + 2 \cdot 3n(2h) + 4 \cdot n + 4 \cdot (4T) + 4 \cdot n)$, which is equal to $(18 \cdot n + 28h \cdot n + 16T)$. This test, though very costly, has a constant idle time that is independent of the number of cells in the memory ($16T$). This test replaces March S2C derived in Section 5.4.3 to detect all soft coupling faults.

6.3 Conclusions

This chapter discussed two optimization approaches to reduce the test length of tests detecting transient and soft faults in DRAM devices. We showed how to use topological layout information to limit the length of these tests, reducing their complexity from quadratic $O(n^2)$ to linear $O(n)$. This makes them more practical candidates for implementation in the industry. Two new tests for soft faults have been discussed, one that represents an optimized version for single-cell soft faults and another for two-cell faults. The complexity of both tests has been reduced in such a way that delay time does not scale with the number of cells in the memory, making them more suitable for industrial application. Specific optimization examples have been shown that enable test designers to modify the tests according to their own memory design or layout.

Chapter 7

Realistic linked memory faults

Memory devices follow very tightly technology developments and as such are the most vulnerable to new types of defects. This trend requires adequate adjustments in the fault models in order to keep the devices dpm level very low. An interesting fault class is the linked faults, that is envisioned to grow in importance when feature sizes shrink to deep sub-micron dimensions. In this Chapter, we propose a systematic approach of how to deal with static linked faults involving both single-cell and two-cell fault primitive types. Our study is based on a rather classical and well-proven set of simple static faults, however, without loss of generality the same methodology can be applied to any other static simple faults set.

We start our analysis by looking at the conditions required for march tests to detect simple static faults in isolation. Thereafter, we investigate the march test fault coverage for the linked fault models based on the used set of fault primitives. Next, we reduce the universe of linked faults to a realistic sub-set by looking at the internal structure of memory cells and derive a march test (March LR) for it. Two additional March LR variants, March LRD and March LRDD are described to deal with simple and double data retention faults, possibly linked with other faults. Last, theoretical comparison and industrial evaluation of March LR, LRD and LRDD fault coverage is presented.

7.1 Fault coverage of march tests

Depending on the way faults manifest themselves, faults can be classified as *simple faults* (a simple fault is a fault which does not influence the behavior of other faults), and *linked faults*, which do influence the behavior of other faults such that masking may occur as introduced in Section 3.7 and in [71, 5]. In

this chapter the term *fault type* denotes a particular class of faults, while *fault subtype* denotes one of the ways a fault type can manifest itself; e.g., a SAF is a fault type, while the SA0 and SA1 faults are subtypes. The term *fault* is used to denote either a fault type or a subtype. As mentioned earlier we use a rather classical set of fault primitives for our study. More precisely we used the following fault models in our study:

- Address decoder faults (AF);
- Single-cell faults:
  - Stuck-at-faults (SAF);
  - Stuck-open faults (SOF);
  - Transition faults (TF);
  - Data retention faults (DRF).
- Two-cell faults:
  - Coupling faults (CF);
  - Inversion Coupling faults (CFin);
  - Idempotent Coupling faults (CFid);
  - State Coupling faults (CFst);
  - Disturb Coupling faults (CFdst).

Please note that both notations, the one introduced in Section 3.7 and the classical one proposed in [5], are used interchangeably.

Both SAF and SOF can be considered as composite functional fault models. The SAF, denoted as $< \forall/0/- >$ and $< \forall/0/- >$ (classical notation $< \forall/x >$, $x \in \{0, 1\}$), with $\forall$ representing all possible sensitizing operations. Therefore, $S = \forall$ can be replaced by any operation or sequence that sensitizes the fault. This leads to the following two SAF FPs:

1. $< \forall/0/- > = \{< 1/0/- >, < 0w1/0/- >\}$;
2. $< \forall/1/- > = \{< 0/1/- >, < 1w0/0/- >\}$.

More precisely, when the faulty behavior of a cell is said to resemble $< \forall/0/- >$, then the cell exibits the faulty behavior of each of the two FPs: $\{< 1/0/- >, < 0w1/0/- >\}$.

The SOF is defined as the faulty behavior due to an open word line in the memory [5]. An open line is assumed to result in failing $0w1$ and/or $1w0$ operations. Furthermore, it may and may not, depending on the implementation of the sense amplifier, result in an incorrect read value (or repeat the last read operation value). Using the FPs introduced in Chapter 3, the SOF can be defined as follows:
SOF = $\{< 0w1/0/- >, < 1w0/1/- >, < 0r0/0/1 >, < 1w1/1/0 >\}$ =
TF $\cup$ IRF.

7.1.1 Fault coverage of simple faults

In this section, conditions for the march tests to detect different faults are presented. A specific condition should be satisfied by a given march test in order to detect the faults of a particular type.

AFsr *Condition 1sr* [1]: in [72] it has been shown that AFs in SRAMs (AFsrs) are more difficult to detect than AFs in DRAMs. Any AFsr is detectable by a march test which satisfies Condition 1tr and, additionally, contains a march element of the form (which may be one of the march elements of Condition 1tr): $\Updownarrow (rx, \ldots, r\bar{x}, \ldots)$
Condition 1tr (for traditional RAMs) is as follows [50]: a march test has to contain the following march elements: $\Uparrow (rx, \ldots, w\bar{x})$ and $\Downarrow (r\bar{x}, \ldots, wx)$

SOFs *Condition 2*: any SOF is detectable by a march test which contains the following march element: $(\ldots, rx, \ldots, r\bar{x}, \ldots)$.

SAFs and TFs *Condition 3*: SAFs and TFs (even when linked with other faults) can be detected by a march test which contains the following two march elements (or a single march element containing both elements):

1. $(\ldots, w0, r0, \ldots)$ to detect SA1 faults and $<\downarrow /1>$ TFs
2. $(\ldots, w1, r1, \ldots)$ to detect SA0 faults and $<\uparrow /0>$ TFs

When SAFs and unlinked TFs are considered, Condition 3 can be simplified to Condition 3U (U stands for *unlinked*).
Condition 3U: SAF and TFs will be detected by a march test which contains the following operations (which can be distributed over march elements in an arbitrary way): wx, $w\bar{x}$, $r\bar{x}$, wx, rx.

DRFs *Condition 4*: any march test can be extended to detect DRFs [73]. The detection of each of the two DRF subtypes requires that a memory cell be brought into the corresponding logic states; thereafter, certain amount of time must elapse for the DRF to develop (for SRAMs, empirical results show that a wait time of approximately 100ms is sufficient [6]). If we are interested in detecting simple DRFs only, the delay elements can be placed between any two pairs of march elements as follows : $\Updownarrow (\ldots, wx); Del; \Updownarrow (rx, \ldots, w\bar{x}); Del; \Updownarrow (r\bar{x}, \ldots)$ with $x =$ '0' or '1'.

CFs *Condition 5*: a march test which contains one of the two pairs of march elements of Case A and a pair of march elements of Case B (see below) can detect all simple CFs (i.e., all CFins, CFids, CFsts, and all CFdsts). Case A

[1] In this manuscript the following shorthand notations are used: *sr* or *S* for SRAM, *dr* for DRAM and *tr* for traditional RAM. A precise definition of the above three address decoder fault sets and their relationships is presented in Chapter 8.

will detect all $< y; \bar{x} >$ and Case B all $< y; x >$ CFs, with x the expected state of the coupled cell and $y \epsilon \{r0, r1, w0, w1\}$ (This set contains all sensitizing operations for CFs considered in this chapter). Note that $\uparrow$ operation is the same as a $w1$ after the memory is in state '0' (otherwise stated $0w1$).

Condition 5 can be explained as follows: The operations $(rx, \ldots w\overline{x}, r\overline{x}, \ldots, wx)$ in the first march element of Case A.1 will sensitize the CFs, because that march element contains all sensitizing operations; i.e., both states are entered (for CFsts), both transitions write operations take place (for CFins and CFids) and it contains all sensitizing operations for CFdsts. Case A.1 will detect the fault, when the value of the fault effect is $\overline{x}$, by the 'rx' operation of the first march element when the coupled cell has a *higher* address than the coupling cell, and by the 'rx' operation of the second march element when the coupled cell has a *lower* address than the coupling cell. Case A.2 has similar capabilities as Case A.1, except that the fault will be detected by the 'rx' operation of the first march element if the coupled cell has a *lower* address than the coupling cell, etc. Case B is required to sensitize and detect the CFs $< y; x >$.

Case A: 1. $\Uparrow (rx, \ldots, w\bar{x}, r\bar{x}, \ldots, wx); \Updownarrow (rx, \ldots)$
2. $\Downarrow (rx, \ldots, w\bar{x}, r\bar{x}, \ldots, wx); \Updownarrow (rx, \ldots)$
Case B: 1. $\Uparrow (r\bar{x}, \ldots, wx, rx, \ldots, w\bar{x}); \Updownarrow (r\bar{x}, \ldots)$
2. $\Downarrow (r\bar{x}, \ldots, wx, rx, \ldots, w\bar{x}); \Updownarrow (r\bar{x}, \ldots)$

In case of SRAM memory tests the simpler CFdst-w fault type (only write operations can cause the fault effect) can be used instead of the CFdst fault type. This allows Condition 5 to be simplified to Condition 5S (for SRAMs).

Condition 5S: any march test which contains one of the two pairs of march elements of Case A and one pair of march elements of Case B can detect all simple CFs (i.e., all CFins, CFids, CFsts, and CFdst-ws).

Case A: 1. $\Uparrow (rx, \ldots, w\bar{x}, \ldots, wx); \Updownarrow (rx, \ldots)$
2. $\Downarrow (rx, \ldots, w\bar{x}, \ldots, wx); \Updownarrow (rx, \ldots)$
Case B: 1. $\Uparrow (r\bar{x}, \ldots, wx, \ldots, w\bar{x}); \Updownarrow (r\bar{x}, \ldots)$
2. $\Downarrow (r\bar{x}, \ldots, wx, \ldots, w\bar{x}); \Updownarrow (r\bar{x}, \ldots)$

7.1.2 Fault coverage of linked faults

Below, a set of propositions is given for those linked faults that the march tests are not able to detect or for fault models with special properties.

Proposition 1: A CFin linked with *one or more* other CFins (CFin#CFins) is not detectable by any march test if the linked fault does not contain an odd number of CFins of the same subtype (i.e., an even number of $<\uparrow; \updownarrow>$ CFins or

an even number of $<\downarrow;\updownarrow>$ CFins) all with addresses of the coupling cell lower (higher) than the coupled cell "i", while no other types of CFs are present with addresses of the coupling cell lower (higher) than the coupled cell "i". For example, Figure 7.1 depicts two different linked CFins with an even number of faults of the same subtype on one side of the coupled cell; both, the faults of Figure 7.1(a) and 7.1(b) satisfy Proposition 1 and hence are not detectable by any march test with linear address order, because the faults involved *mask* each other.

Figure 7.1: Linked CFins which satisfy Proposition 1.

Proposition 2: CFsts dominate CFins, CFids and CFdsts. Note that in case of CFsts a fault is sensitized by the *state* of the coupling cell, whereas in case of CFids, CFins or CFdsts the fault is sensitized by *operations* on the coupling cell. Therefore, the CFst fault *dominates* the fault types CFid, CFin and CFdst; i.e., the fault effect of the CFst is not caused by a write or a read operation, it is caused by the state of the coupling cell such that it will be present as long as the coupling cell contains the sensitizing value, regardless of the fact whether the coupled cell is influenced by CFins, CFids or CFdsts.

Proposition 3: The fault effect of a CFst linked with one or more other CFsts (notation: CFst # CFsts) can be as follows:

a. The result of the CFst # CFsts is a function independent of the previous operations (deterministic); i.e. given the set '**S**' of m linked CFsts; i.e. $\mathbf{S} = \{< p; e_1 >, < q; e_2 >, \cdots, < z; e_m >\}$, whereby $p, q, z \in \{0, 1\}$, then the resulting fault effect $e_r = F(e_1, e_2, \cdots, e_m)$. This simplifies the set of linked CFsts to a single CFst which is detectable by march tests.

b. The result of reading CFst # CFsts depends on the result of the operation performed before the detect operation (a "random" function). The following three cases can be distinguished:

    1. The number of CFsts with the coupling cell on one side of the coupled cell is 1; i.e., $a = 1$ or $b = 1$ (see Figure 7.2). These faults are

detectable by march tests because one of the linked CFsts can be sensitized and detected in isolation of the rest;

2. The case $a > 1$ (see Figure 7.2) but the CFsts $< \ > j_1 i \cdots < \ > j_a i$ all have the same fault effect; and/or, similarly for the case $b > 1$. These faults are detectable by march tests because the faults on one side of the coupled cell cannot cause a random read result;
3. The case $a > 1$ and $b > 1$, whereby the fault effect of the a CFsts, and of the b CFsts is not the same. These faults are *not detectable* by march tests; a test of $O(n^2)$ is required.

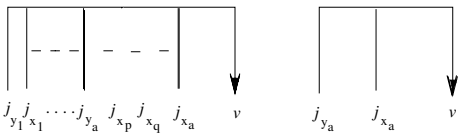


Figure 7.2: Class 2 CFs

7.1.3 Reducing the universe of linked faults to realistic linked faults

Next we will enumerate the set of all linked faults to a smaller set referred to as *realistic linked faults* that consist of all linked faults, excluding the following subclasses:

a. Linked faults which include CFins will not be considered of practical interest because Dekker et al [74] has shown that the empirical evidence does not support the occurrence of simple CFins; linked CFins are therefore even less likely. This subclass will therefore not be considered any further in this chapter.

b. Linked faults which include the following two linked CFids: $<\uparrow;\uparrow>ji \ \# \ <\downarrow;\downarrow>ji$ or $<\uparrow;\uparrow>ik \ \# \ <\downarrow;\downarrow>ik$ are not considered realistic (see Figure 7.3(a)). This is because CFids are caused by capacitive coupling between bitlines, and the above two linked faults can only occur between cells in the same row (because only then the coupling effect on a bitline is passed to the coupled cell). Before a read or a write operation takes place, the bitlines are precharged such that the CFid only can be sensitized when the bitline BL or $\overline{BL}$ of the coupling cell makes a $\downarrow$ transition. Because of the way memories are built, see Figure 7.3(b), cells i and j (i and k respectively) can only be coupled in one of the following ways (for a given word line, and hence for a given cell i): $BL_j \longrightarrow BL_i$ (i.e.,BL_i is coupled to BL_j), $BL_j \longrightarrow \overline{BL}_i$, $\overline{BL}_j \longrightarrow BL_i$ or $\overline{BL}_j \longrightarrow \overline{BL}_i$ (see Figure 7.3).

When an $\uparrow$ transition write operation ($0w1$) takes place, first BL and $\overline{BL}$ are precharged, thereafter $\overline{BL}$ makes a $\downarrow$ transition (similar arguments hold for $\downarrow$ transition write operations). The CFid $<\uparrow;\uparrow> ji$ can only exist when $\overline{BL}_j \rightarrow \overline{BL}_i$; while the CFid $<\downarrow;\downarrow> ji$ can only exist when $BL_j \rightarrow BL_i$; this implies that the linked faults $<\uparrow;\uparrow>ji \ \# \ <\downarrow;\downarrow>ji$ or $<\uparrow;\uparrow>ik \ \# \ <\downarrow;\downarrow>ik$ cannot occur in practice simultaneously. This observation applies to SRAMs as well as to DRAMs.

c. Linked faults which include two linked CFdsts with opposite fault effects: $< x;\downarrow>ji \ \# \ < y;\uparrow>ji$ or $< x;\downarrow>ik \ \# \ < y;\uparrow>ik$, where $x, y \ \epsilon\{r0, r1, w0, w1\}$, are not considered realistic for the same reasons as the CFids above.

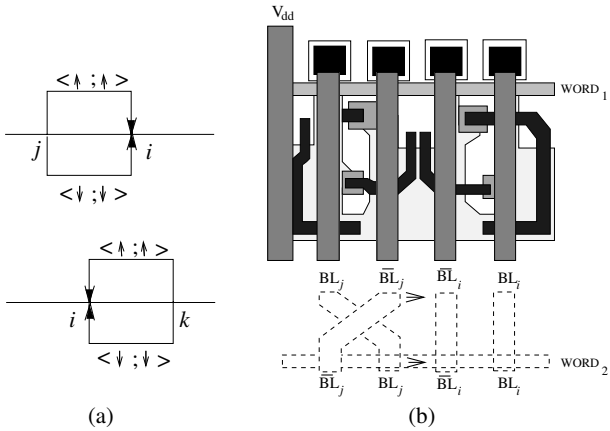


Figure 7.3: CFids between neighboring cells

7.2 March LR: a test for realistic linked faults

The ultimate wish of any test algorithm designer is to get maximum fault coverage with minimal test time. We will try to satisfy these two conflicting goals by designing new tests for realistic linked faults. The design methodology is based on deriving conditions march tests have to satisfy in order to detect faults of certain fault types and subtypes; thereafter these requirements are assembled resulting in the new march test: *March LR*. March LR detects all simple faults used in this chapter as well as all *realistic* linked faults (see Section 7.1.3); excluding those CFins which satisfy Proposition 1 and those CFsts which satisfy Proposition 3(b.3).

This section consists of the following parts: first the universe of linked faults

consisting of two simple faults (excluding CFdsts, and linked CFsts which do not satisfy Proposition 3(b.3)) is divided into sets; thereafter, in Section 7.2.2, the conditions for march tests to detect linked faults, consisting of *two* simple memory cell array faults, are derived. Section 7.2.3 describes March LR together with other derived tests. Section 7.2.4 shows that March LR is able to detect *all realistic* linked faults.

7.2.1 Establishing sets of linked faults

First we divide all *simple* CFs (except CFdsts) into the following sets:

Set B: All subtypes which set the coupled cell to 1
$\mathbf{B} = \{<\uparrow;\uparrow>,<\downarrow;\uparrow>,<0;1>,<1;1>,<\uparrow;\updownarrow>,<\downarrow;\updownarrow>\}$
Note : in case of CFins, the expected value has to be "0".

Set B*: All subtypes which set the coupled cell to 0
$\mathbf{B^*} = \{<\uparrow;\downarrow>,<\downarrow;\downarrow>,<0;0>,<1;0>,<\uparrow;\updownarrow>,<\downarrow;\updownarrow>\}$
Note : in case of CFins, the expected value has to be "1".

Figure 7.4 shows the set of all possible simple CFins, CFids and CFsts, consisting of respectively $2 + 4 + 4 = 10$ faults. It is easy to verify that the universe of all simple CFs (excluding CFdsts) consists of the **B** and **B***. Note: Because of the alternated expected value of the coupled cell of a CFin, the CFins belong to set **B** as well as **B***.

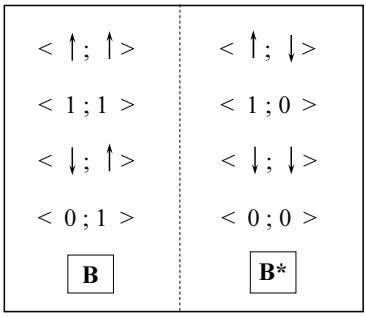


Figure 7.4: Subsets of CFs

The universe of all *linked* faults consisting of *two* simple faults (of type CFin, CFid or CFst) can be divided into the following three *collections*:

1. The collection **B** # **B***. This collection consists of linked faults whereby one of the faults is from set **B** and the other fault is from set **B*** (see Figure 7.4); this collection therefore consists of $5 * 5 = 25$ linked faults. Taking into account the relative address positions of the involved cells (expressed in the three address order classes introduced in the next section), this results into $25 * 6 = 150$ different linked faults;

2. The collection **B** # **B**; and

3. The collection **B*** # **B***.

7.2.2 Conditions for march tests to detect linked faults consisting of two simple faults

In addition to the conditions derived in Section 7.1 (these are the Conditions 1 through 5), the following conditions have to be satisfied in order to detect linked faults consisting of *two* simple faults. Table 7.1 summarizes the conditions derived in this section as a function of the collections of linked faults and of the address order classes. The address orders of all linked faults consisting of two CFs, or an AF and a CF, can be split into the following *address order classes*:

- *Middle class*: $< x; y >ji \#< x; y >ik$ The *coupled* cell address lies between the two coupling addresses;

- *High class*: $< x; y >j_p i \#< x; y >j_q i$ The *coupled* cell has the highest address, and involves two simple faults;

- *Low class*: $< x; y >ik_p \#< x; y >ik_q$ The *coupled* cell has the lowest address, and involves two simple faults.

Table 7.1: Conditions for two linked faults

Address order class	Linked fault collection		
	B # B	B # B*	B* # B*
Low	Condition 6	Condition 6S	Condition 6
Middle	Condition 6	Condition 6	Condition 6
High	Condition 6	Condition 6S	Condition 6

Condition 6 : All linked CFs consisting of two simple CFs (excluding CFdsts, and excluding linked CFsts which do not satisfy Proposition 3(b.3)) of the *Middle class* will be sensitized and detected by the same march element if the march test contains the march element of Case A.1 (in case the expected value

Table 7.2: Requirements for detecting linked CFs

Value of z	$< p; z > ji$ detected	$< q; z > ik$ detected
$z = x$	B.1	B.2
$z = \bar{x}$	A.1	A.2

is x), and the march element of Case B.1 (in case the expected value is $\bar{x}$); or the march element of Case A.2 and the march element of B.2:

Case A: 1. $\Uparrow (rx, w\bar{x}, \ldots, wx)$
2. $\Downarrow (rx, w\bar{x}, \ldots, wx)$
Case B: 1. $\Uparrow (r\bar{x}, wx, \ldots, w\bar{x})$
2. $\Downarrow (r\bar{x}, wx, \ldots, w\bar{x})$

The two write operations $w\bar{x}$ and wx will sensitize all possible CFs (excluding CFdsts because a second sensitizing read operation is needed); and because no masking is possible (the Middle class ensures that the second fault lies at the other side of the coupled cell) the 'rx' operation of Case A will detect the fault when the expected value is x, and the '$r\overline{x}$' operation of Case B when the expected value is $\bar{x}$.

The correct combination of Cases A and B has to be taken as follows: A.1 and B.1, or A.2 and B.2. This can be shown using Figure 7.5 which contains the linked CFs $< p; z > jv \ \# \ < q; z > vk$; $z \ \epsilon \ \{x, \bar{x}\}$ and v the victim cell shared by both CFs (also noted as i in this text). The four detection cases are presented in Table 7.2. The result is that in order to *always* detect fault $< p; z > ji$ march elements B.1 *and* A.1 are required; for fault $< q; z > ik$ march elements A.2 *and* B.2 are required. Therefore, in order to detect the fault of Figure 7.5, march elements A.1 *and* B.1, or A.2 *and* B.2 are required.

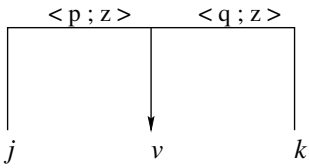


Figure 7.5: Subsets of CFs

Note, one march element of Case A and one of Case B will also sensitize and detect all linked faults of collection **B#B** and of collection **B*#B***, re-

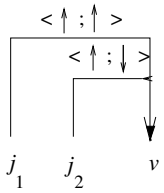


Figure 7.6: The linked fault <↑;↕> # <↑;↓>

gardless of the address order class, because the fault effects of the linking faults are the same; hence, no masking is possible (the first and the third columns of Table 7.1).

For faults of collection **B#B***, regardless of the address order class, we need to make Condition 6 stronger because of the possibility of fault effect masking, hence Condition 6S is required.

Condition 6S : All linked CFs consisting of two simple CFs (excluding CFdsts, and linked CFsts which do not satisfy Proposition 3(b.3)) of collection **B#B***, regardless of the address order class, will be sensitized and detected if the test contains a pair of march elements of Case A.1 and a pair of march elements of Case B.1, or Cases A.2 and B.2 respectively.

Case A: 1. $\Uparrow (rx, w\bar{x}, \ldots, wx); \Uparrow (rx, \ldots)$
2. $\Downarrow (rx, w\bar{x}, \ldots, wx); \Downarrow (rx, \ldots)$
Case B: 1. $\Uparrow (r\bar{x}, wx, \ldots, w\bar{x}); \Uparrow (r\bar{x}, \ldots)$
2. $\Downarrow (r\bar{x}, wx, \ldots, w\bar{x}); \Downarrow (r\bar{x}, \ldots)$

All faults of the High class will be sensitized and detected by the first march element of Case A.1 or Case B.1. This is because these two march elements each contain a $w\overline{x}$ and a wx operation, which implies that they sensitize all considered CFs, while the CF with the coupling cell closest to the coupled cell will be detected by the rx or the $r\overline{x}$ operation of the same march element. For example, consider the linked fault of Figure 7.6. Assume the initial state of memory to be '0', then the pair of march elements of Case A.1 or A.2 will *not* detect that fault; however the pair of Case B.1 or B.2 will.

In case of CFs of the Low class and march elements of Case A.1 and B.1, the read operation of the second march element of each pair is required for detecting the fault (in this case the fault with the larger distance to the coupled cell); in order to prevent masking, the address order of the second march element of each pair has to be identical to that of the first march element of each pair. The first march elements of Cases A.2 and B.2 will detect all faults of the low class, and also sensitize all faults of the high class which will be detected by the second march element of each pair.

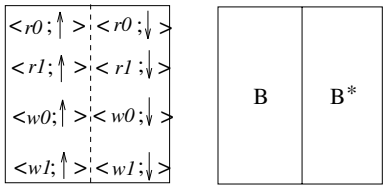


Figure 7.7: Subsets of CFdsts

If, in addition, the detection of CFdsts is required, Condition 6S should be modified into Condition 6SD. The set of CFdsts can be divided into the sets **B** and **B*** (see Figure 7.7), similar to the other CFs, and detection is possible if a march test satisfies Condition 6SD.

Condition 6SD: All linked CFs consisting of two simple CFs, *including CFdsts* (excluding linked CFsts which do not satisfy Proposition 3(b.3)) of collection **B**#**B***, regardless of the address order class, will be sensitized and detected if the test contains a pair of march elements of Case A.1 and a pair of march elements of Case B.1, or of Case A.2 and B.2:

Case A: 1. $\Uparrow (rx, w\bar{x}, r\bar{x}, \ldots, wx); \Uparrow (rx, \ldots)$
2. $\Downarrow (rx, w\bar{x}, r\bar{x}, \ldots, wx); \Downarrow (rx, \ldots)$
Case B: 1. $\Uparrow (r\bar{x}, wx, rx, \ldots, w\bar{x}); \Uparrow (r\bar{x}, \ldots)$
2. $\Downarrow (r\bar{x}, wx, rx, \ldots, w\bar{x}); \Downarrow (r\bar{x}, \ldots)$

The proof of Condition 6SD is similar to that of Condition 6S. Note that the first march element of each pair sensitizes all CFs, including all CFdsts; in addition Condition 6SD is a superset of Condition 5, Condition 6 and Condition 6S. Condition 6SD guarantees detection of the collections **B**#**B** and **B***#**B*** because all simple CFdsts are sensitized, while masking of CFs # CFdsts is not possible because all faults of each collection have the same fault effect.

When one of the two linked faults is an address decoder fault, an additional condition is required. As will be shown in Chapter 8, concerned with linked address decoder faults, any march test will detect an AFsr # CF if the march test satisfies the following condition:
Condition 1L [72]: a march test has to detect SOFs (Condition 2), all simple CFs (Condition 5), and also all simple AFs (Condition 1sr).

In cases when one of the two linked faults is a data retention fault the following conditions hold.
Condition 4LS: Any march test will *additionally* detect the presence of a 'DRF # other (linked) fault(s)' when :

1. that march test is able to detect the other (linked) fault(s) involved in the absence of the DRF;

2. that march test is *extended* with the following sequence of march elements (Note: the state of the cells upon completion of the non extended test is assumed to be x): $Del; \Updownarrow (rx, w\bar{x}); Del; \Updownarrow (r\bar{x})$.

Condition 4LD: Any march test will *additionally* detect the presence of *both* DRF subtypes (i.e. a *double DRF*) in a single SRAM cell (i.e., this will be the case when both pull-up devices are broken), also when linked with any other fault(s), when:

1. that march test detects those other (linked) fault(s) in the absence of double DRFs;

2. that march test is *extended* with the following sequence of march elements (Note: the state of the cells upon completion of the non-extended test is assumed to be x): $Del; \Updownarrow (rx, w\bar{x}, r\bar{x})$.

The presence of double DRFs in a single SRAM cell cause that cell to behave as if it contains a SOF after some time (the time of the 'Del' operation).

Condition 4LSD: Any march will *additionally* detect the presence of simple and double DRFs, possibly linked with other (linked) fault(s), when:

1. that march test detects those other (linked) fault(s) in the absence of double DRFs;

2. that march test is *extended* with the following sequence of march elements (Note: the state of the cells upon completion of the non-extended test is assumed to be x): $Del; \Updownarrow (rx, w\bar{x}, r\bar{x}); Del; \Updownarrow (r\bar{x})$.

7.2.3 March LR

Using the conditions described in the previous section a new march test can be constructed. The test March LR, depicted in Figure 7.8, is able to detect the following faults:

March LR = {		
$\Updownarrow (w0);$	$\Downarrow (r0, w1);$	$\Uparrow (r1, w0, r0, w1);$
M0	M1	M2
$\Uparrow (r1, w0);$	$\Uparrow (r0, w1, r1, w0);$	$\Uparrow (r0)\}$
M3	M4	M5

Figure 7.8: March LR

1. SOFs: Condition 2 is satisfied by M2 and also by M4;
2. SAFs and (linked) TFs: Condition 3 is satisfied by M2 and also by M4;
3. CFs: Condition 5 is satisfied by M2 through M5;
4. Linked CFs consisting of two simple CFs: Condition 6SD is satisfied by M2 together with M3 (Case B.1), and by M4 together with M5 (Case A.1);
5. AFs : Condition 1sr is satisfied by M1 together with M2 and M3;
6. AF # TF: Condition 1sr is satisfied (see above);
7. AF # CF: Condition 1L is satisfied as follows: Condition 2 (satisfied by M2), Condition 5 (because Condition 6SD is satisfied), and Condition 1sr is satisfied.

Figure 7.9 shows the way March LR of Figure 7.8 is modified in order to additionally detect linked DRFs. The linked DRFs are detected because Condition 4LS(1) is satisfied by M0 through M5; and Condition 4LS(2) by M6 through M9.

March LRD = {				
$\Updownarrow (w0);$	$\Downarrow (r0, w1);$	$\Uparrow (r1, w0, r0, w1);$	$\Uparrow (r1, w0);$	$\Uparrow (r0, w1, r1, w0);$
M0	M1	M2	M3	M4
$\Uparrow (r0);$	Del	$\Updownarrow (r0, w1);$	Del	$\Updownarrow (r1)\}$
M5	M6	M7	M8	M9

Figure 7.9: March LRD

The March LR extension for detection of single DRFs and double DRFs in a cell linked with any other fault is depicted in Figure 7.10. DRFs (single and double) linked or unlinked with other faults will be detected because March LRDD satisfies Condition 4LSD(1) by M0 through M5, and Condition 4LSD(2) by M6 through M9.

March LRDD = {				
$\Updownarrow (w0);$	$\Downarrow (r0, w1);$	$\Uparrow (r1, w0, r0, w1);$	$\Uparrow (r1, w0);$	$\Uparrow (r0, w1, r1, w0);$
M0	M1	M2	M3	M4
$\Uparrow (r0);$	Del	$\Updownarrow (r0, w1, r1);$	Del	$\Updownarrow (r1)\}$
M5	M6	M7	M8	M9

Figure 7.10: March LRDD

7.2.4 Comparison of March LR with other tests

This section shows the ability of March LR to detect *all realistic* linked faults, consisting of the linked faults introduced earlier in this chapter. First, a careful theoretical analysis will be presented. Next, the additional fault coverage for specific pattern sensitive faults of March LR will be indicated. And last, an experimental evaluation of March LR fault coverage will be discussed.

Comparison based on theoretical analysis

Because M2 and M4 of March LR contain all sensitizing operations (i.e., $r0, r1, w0, w1$) they are able to sensitize all CFs. Hence, all simple faults involved in the linked fault will be sensitized in an $\Uparrow$ address order (first will be sensitized the simple fault $< \ldots > j_1 i$, and last the simple fault $< \ldots > i k_b$).

March elements M2 and M4 will sensitize and detect the *Low part* of the fault, because M2 and M4 sensitize all CFs involved in the Low part and both have an $\Uparrow$ address order. The fault $< \ldots > j_a i$ will be sensitized as last before the $r0_i$ or $r1_i$ operation on cell 'i' of march element M2 or M4 respectively. Hence, the fault effect of $< \ldots > j_a i$ will differ from the expected value of cell 'i' for march element M2 or M4 and will be detected by the same march element.
If M2 does not detect any faults, one of two alternatives exist:

1. *Alt.1* The fault has no Low part. Then M3 or M5 will detect the fault $< \ldots > i k_b$ which is sensitized last by M2 and M4, and because no Low part exists the read operation of M3 or M5 will detect the fault;

2. *Alt.2* The fault effect of the Low part fault $< \ldots > j_a i$ is a '1' (masking all previous faults).
   In that case the same fault will be detected by M4 (note that CFins are not considered, hence the fault effect of the $< \ldots > j_a i$ is always a '0' or a '1').

Hence, if M2 through M5 do not detect any faults there are no CFs of the considered types (linked or simple).

Table 7.3 shows the fault coverage of March LR and its derivatives (March LRD and LRDD) and the existing march tests shown below, for the realistic linked faults considered in this chapter. The columns 'TF involved' through 'CFdsts involved' together form all the possible faults. The second column 'TF involved' specifies realistic linked faults whereby one of the linked faults is a TF. The linked faults which include CFdsts are not detected by the existing tests because CFdsts are sensitized by read as well as by write operations, which makes it more difficult to avoid masking.

March C [20]=
$\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Updownarrow (r0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$

March C− [5]=
$\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$

March A [53]=
$\{\Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\}$

March B [53]=
$\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\}$

Algorithm B [20]=
$\{\Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, r0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, r1, w0)\}$

MOVI [21]=
$\{\Downarrow (w0); \Uparrow (r0, w1, r1); \Uparrow (r1, w0, r0); \Downarrow (r0, w1, r1); \Downarrow (r1, w0, r0)\}$

March M [75]=
$\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0); \Updownarrow (r0); \Uparrow (r0, w1); \Updownarrow (r1); \Uparrow (r1, w0, r0, w1); \Updownarrow (r1); \Downarrow (r1, w0)\}$

Table 7.3: Fault coverage of March LR and traditional march tests

March test		TF involved	CFid # CFids	CFids involved	CFsts[a] involved	CFdst # CFdsts	CFdsts involved	DRF involved
March C-	$10n$	-	-	-	+	-	-	-
March B	$17n$	+	+	-	+	-	-	-
Algorithm B	$17n$	+	+	-	+	-	-	-
MOVI	$13n$	+	-	-	+	-	-	-
March M	$16n$	+	+	+	+	-	-	-
March LR	$14n$	+	+	+	+	+	+	-
March LRD	$17n+2D$	+	+	+	+	+	+	+
March LRDD	$18n+2D$	+	+	+	+	+	+	+

[a] only those linked CFsts which do not satisfy Proposition 3(b.3)

Additional March LR fault coverage
Existing march tests, such as March A, March B and Algorithm B are designed to detect linked faults consisting of an arbitrary number of simple faults of the same type; in this case of type CFin or CFid. March M, in addition, is able to detect linked faults consisting of an arbitrary number of simple faults which may be of type CFin or CFid, except for the CFins of Proposition 1 and except for the following (complex) linked fault: $<\downarrow;\uparrow> ik_1 \# <\uparrow;\downarrow> ik_2 \# <\uparrow;\uparrow> \& <\downarrow;\downarrow> ik_3$.

Below, a list of linked faults is given which will be detected by March LR but not by either March A, and/or March B, and/or March C. In addition, March LR detects any NPSF 'Neighborhood Pattern Sensitive Fault' (ANPSF, PNPSF, or SNPSF [5]) assuming a *homogeneous neighborhood*: this is a neighborhood whereby all cells, except the base cell, have the same value. This is because of the fact that M2 and M4 do not change the state of the neighborhood while all possible CFs are sensitized for each of the cells, hence the state of the neighborhood does not change (remains activated).

- $<\downarrow /1 >i\#<\uparrow;\updownarrow>ik$ - not detected by March A;
- $<\downarrow;\updownarrow>ik_1\#<\downarrow;1 >ik_2$ - not detected by March A or March B;
- $<\uparrow;\downarrow>j_1i \ \# \ <\uparrow;\updownarrow>j_2i$ - not detected by March A or March B;
- $<\uparrow;\uparrow>j_1i \ \# \ <\uparrow;\updownarrow>j_2i \ \# \ <\downarrow;\updownarrow>j_2i \ \# \ <\uparrow;\updownarrow>ik$ - not detected by March A or March B;
- $<\uparrow;\downarrow>j_1i \ \# \ <\downarrow;\updownarrow>j_2i \ \# \ <\downarrow;\updownarrow>j_3i \ \# \ <\uparrow;\updownarrow>j_3i$ - not detected by March A or March B;
- ANPSF $\langle 0,0,0,\uparrow,0,0,0,0;1\rangle$ - not detected by March A, March B or March C;
- PNPSF $\langle 0,0,0,0,0,0,0,0;\uparrow /0\rangle$ - not detected by March A, March B or March C;
- SNPSF $\langle 1,1,1,1,1,1,1,1;\forall/1\rangle$ - not detected by March A, March B or March C.

Industrial evaluation of March LR

In [46] an experiment has been performed whereby 1896 1Mx4, DRAM chips have been tested at room temperature ($T_r = 25^oC$). In this situation 731 of the chips failed. The passing chips (except for 25 which were mechanically damaged) were subsequently tested at high temperature ($T_h = 70^oC$). In this case only 471 failed. A total of 44 test algorithms were used in this experiment, and the tests were applied with up to 96 different stress conditions (V_{cc} voltage levels, data backgrounds, timing variations, and addressing variations).

Table 7.4 shows some of the test results for comparison purposes. It consists of two 5×5 sub-matrices; one for test results at $T_r = 25^oC$ and one for test results at $T_h = 70^oC$. The diagonal entries of each sub-matrix show the *fault coverage (FC)* of the corresponding tests; e.g., for T_r and matrix element [5,5] FC = 235 means that March LR detects 235 faults when performed at 25^oC (room temperature). The non-diagonal elements represent the intersection of

Table 7.4: DRAM fault coverage

#	Test	Length	$T_r = 25^oC$					$T_h = 70^oC$				
			1	2	3	4	5	1	2	3	4	5
1	March C-	$10n$	234	212	216	144	218	163	148	148	70	150
2	March A	$15n$	212	222	217	146	210	148	157	153	67	145
3	March B	$17n$	216	217	232	147	213	148	153	157	68	144
4	MOVI	$13n$	144	146	147	201	144	70	67	68	144	70
5	March LR	$14n$	218	210	213	144	235	150	145	144	70	173

the corresponding tests: e.g., for T_h and matrix element[5,3] the FC = 144, which means that of the total number of faults detected by March B (157 faults), 144 are also detected by March LR.

From the results presented in Table 7.4 we can conclude that March LR outperforms the other tests. It performs much better than the even longer tests March A ($15n$) and March B ($17n$).

7.3 Conclusions

Many existing march tests have been designed to detect simple as well as linked faults. Linked faults were considered to consist of simple faults of the same type only. This fact is considered invalid in case of modern memory structures. In this chapter a thorough analysis of linked faults was presented resulting in the subclass of realistic linked faults. It has been shown that existing march tests (such as March A and March B) do not detect all linked faults which are considered realistic.

A set of conditions the march tests have to satisfy was derived in order to detect all realistic linked faults. These conditions were used to design the new tests March LR, March LRD and March LRDD; they have the property to detect all realistic faults and have a shorter test length than comparable industry adopted tests. In addition, they can detect NPSFs with a homogenous neighborhood. Industrial results support our claim about the superiority of March LR fault coverage compared to other widely used tests.

Chapter 8

Linked Address Decoder Faults

Many march tests have been designed in the past with the emphasis on faults in the memory cell array; while *address decoder faults* (AFs) have not been analyzed very thoroughly. Only very simple AFs are shown to be detectable by most of the existing memory tests.

Linked faults complicate test design because of the possibility of masking. As indicated earlier, it is expected that their importance will grow when deep sub-micron technologies are used for the implementation of memory structures. Although the statistical significance of this class of faults has not been established yet, one needs industrial (and therefore usually confidential) data for this; the industrial emphasis on low defect levels and utmost fault coverage motivates this study. Many march tests have been designed [5] to detect linked faults in the memory cell array; ignoring the extra conditions a march test has to satisfy in order to detect an AF linked with other AFs, and AFs linked with faults in the memory cell array. This will be the subject of this chapter; it will present a systematic set of conditions for detecting linked faults involving AFs. In addition, the results show that tests can easily be modified to detect this class of faults without any increase, or with only a small increase, in test time. The subject is therefore highly relevant to test designers interested in high fault coverage; in addition, it is of academic interest because the difficult and until now largely neglected problem of linked AFs has been analyzed and solutions are presented.

Section 8.1 gives the analysis of unlinked address decoder faults (*AFs*) and gives the conditions march tests have to satisfy in order to detect unlinked AFs. Section 8.2 derives conditions march tests have to satisfy in order to detect an AF linked with *memory cell array faults* (MCAFs) and an AF linked

with other AFs. Section 8.3 analyzes the detection capabilities of a set of well-known march tests for (linked) AFs. Many tests are shown not to be able to detect all (linked) AFs. Finally, Section 8.4 ends with the conclusions.

8.1 Conditions for detecting unlinked address decoder faults

For the sake of convenience we reintroduce the four fault combinations in the address decoder presented earlier in Chapter 3. They are depicted in Figure 8.1 below.

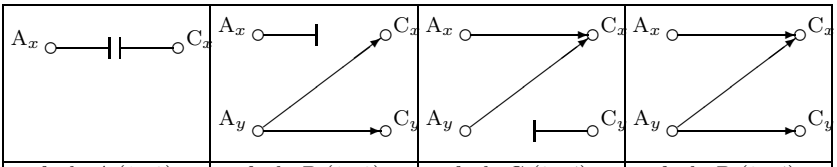


Figure 8.1: Combinations of address decoder faults

Depending on the memory technology, the effect of AFs on memory read operations can be according to the following cases:

Case A: When no cell will be accessed with a certain address (Figure 8.1, fault A and B) the effect of a read operation can be that:

1. the cell behavior is that of a SAF. This applies to DRAMs, and to SRAMs which use a single bitline per column.
2. the cell behavior is that of a SOF. This applies to SRAMs which use differential sense amplifiers.

Case B: When multiple cells, whereby at least one cell contains a '0' and at least one other cell contains a '1' value, are read with a certain address (Figure 8.1, fault D) the result of the read operation can be:

1. deterministic. This will be the case for DRAMs; they produce the AND or the OR of the two read values. This case also applies to SRAMs which use a single bitline per column.
2. not deterministic (random). This applies to SRAMs which use a differential sense amplifier. The detection of this fault cannot be done based on the read result of the multiple cell access; it should be based on reading other cells involved in the fault [5].

Table 8.1 shows the memory types for which the four combinations of the Cases A and B apply; note that the combinations A.1 - B.2 and A.2 - B.1 do not occur in real memories.

For testing AFs for DRAMs Case A.1 and Case B.1 apply, this AF will therefore be called *AFdr*. For testing SRAMs Case A.2 and Case B.2 apply, this AF will be called *AFsr*. The AFs used traditionally in literature, and until this section in this thesis, do not distinguish between SRAM and DRAM and consist of Case A.1 and B.2. This type of traditional AF will be referred to as *AFtr* from hence for. In summary, this means that the fault type AF consists of three subtypes: AFsr, AFdr, and the AFtr; whereby the fault subtype AFdr consists of the following sub-subtypes (as shown below): AFdr-and, AFdr-or, and AFdr-ao.

Table 8.1: Memory types for which Cases A and B apply

Read not connected cell	Read multiple cells	Memory type
A.1: SAF	B.1: Deterministic	AFdr: DRAM (SRAM with 1 bitline)
A.1: SAF	B.2: Random	AFtr: traditional AF *(non existing)*
A.2: SOF	B.1: Deterministic	*(non existing)*
A.2: SOF	B.2: Random	AFsr: SRAM

In order for a march test to detect all *unlinked* address decoder faults, it has to satisfy the following conditions:

AFdr Three cases can be distinguished, depending on the result of reading multiple cells which do not all have the same value.
Condition 1dr-and: The read result is the boolean AND function of the read values. In order to detect unlinked AFdrs of this subsubtype (which are called *AFdr-and*), the march test has to contain the following two pairs of march elements [5]:

1. $\Updownarrow(\ldots, w0); \Updownarrow(r0, \ldots)$
2. $\Updownarrow(\ldots, w1); \Updownarrow(r1, \ldots, w0)$

Condition 1dr-or: The read result is the OR function of the read values. In order to detect unlinked AFdrs of this sub-subtype (which are called *AFdr-or*), the march test has to contain the following two pairs of march elements [5]:

1. $\Updownarrow(\ldots, w0); \Updownarrow(r0, \ldots, w1)$

2. $\Updownarrow(\ldots, w1); \Updownarrow(r1, \ldots)$

Condition 1dr-ao: The read result is the AND *or* OR of the read values (the memory technology is *not* known to the test designer). The march test has to contain the following three march elements to detect the AFdrs of the subsubtype *AFdr-ao*:

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

AFtr *Condition 1tr*: any AFtr is detectable by a march test which contains the following two march elements [50]:
$\Uparrow(rx, \ldots, w\bar{x})$ and $\Downarrow(r\bar{x}, \ldots, wx)$.

AFsr *Condition 1sr*: any AFsr is detectable by a march test which satisfies Condition 1tr and, additionally, contains a march element of the form (which may be one of the march elements of Condition 1tr):
$\Updownarrow(rx, \ldots, r\bar{x}, \ldots)$
From the above conditions one can determine the following hierarchy (1 is the strongest condition) in conditions a march test has to satisfy in order to detect unlinked AFs. If Condition 1sr, for detecting AFsrs, is satisfied, all other AFs (AFtrs, AFdrs, AFdr-ands, AFdr-ors, and AFdr-aos) will be detected; this is the main reason why we used Condition 1sr in Chapter 7.

1. Condition 1sr for AFsrs;
2. Condition 1tr for AFtrs;
3. Condition 1dr-ao for AFdrs, when the memory technology is not known;
4. Condition 1dr-and or Condition 1dr-or for AFdrs, when the memory technology is known at advance.

8.2 Conditions for detecting linked address decoder faults

This section establishes the conditions a march test has to satisfy in order to detect ' AF # faults'; these faults may be AFs or MCAFs. The conditions for detecting simple MCAFs derived in Chapter 7 will be used in this section. A special role is dedicated to Conditions 2 and 5 for detecting simple SOFs and CFs respectively. Section 8.2.1 gives the conditions for march tests to detect 'AF # one fault'; Section 8.2.2 addresses the conditions for 'AF # more than one fault'. The starting point is the AF subtype AFsr, and if a condition can be weaker due to the AFdr fault subtype (in case of DRAM memories), this will be explained additionally.

8.2.1 AF # *one* fault

The conditions march tests have to satisfy in order to detect an 'AF # one fault' (an AF linked with one fault) are derived below for each of the faults.

AF # AF This combination is very realistic (e.g. short between two word lines) (see Figure 8.2). Each of the two march elements of Condition 1tr will detect the fault, due to the its symmetric structure. In case of DRAMs, only Condition 1dr has to be satisfied.

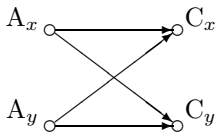


Figure 8.2: AF linked with AF combination

Proposition 4: The AFs A and B of Figure 8.1 dominate any other MCAFs (arbitrary number) because the cell will not be accessed; therefore any other fault in the same cell, in addition to AFs A and B of Figure 8.1, is irrelevant. The combination with other faults will be detected by any march test which detects SOFs (see Condition 2). Therefore, regardless of the conditions for detecting faults C and D of Figure 8.1, the condition for detecting SOFs has to be satisfied.

In the remainder of this section the AFsr C and AFsr D (multiple cell access) of Figure 8.1 will be considered.

AF#SAF This combination can always be detected by a test for AFs, because those tests will also detect the SAFs.

AF#TF This linked fault can be detected by any march test for AFs which in addition satisfies Condition 1T.

Condition 1T: Any march test will detect an AF # TF if it detects the AF and contains a march element of the form:
$(\ldots, wx, rx)$ [50, 5].

AF#DRF The AF linked with a DRF will be detected by a march test which detects the AF (DRF development time is irrelevant in the case that AF is detected). Hence only the detection of the AF is required; for SRAMs this will be Condition 1sr, for DRAMs Condition 1dr.

AF#CF Two alternatives are possible:

1. The AF is linked with the *coupling cell* of the CF, see Figure 8.3. Figure 8.3(a) shows the case whereby the coupled cell of the CF is not involved in the AF, while the coupled cell of the CF of Figure 8.3(b) is involved in the AF. The fault AF # CF of Figure 8.3(a) can be represented as the

linked CF of Figure 8.3(c) whereby the number of linking faults is equal to the number of addresses connected to cell i and the subtype of each of the linking faults is the same. This means that, according to Proposition 1 derived in Chapter 7, the fault of Figure 8.3(a) is not detectable by any march test when the CF is of type CFin and the number of addresses connected to cell i is even. In all other cases, this fault is detectable by a test for CFs or for AFs. The fault of Figure 8.3(b) is always detectable by a test for CFs or for AFs; this is because when a write operation takes place via A_{j2} the effect of the CF will be overruled by the effect of this write operation such that only the AF remains, while in case of writing to address A_i only the CF remains.

2. The AF is linked with the *coupled cell* of the CF. This alternative will be analyzed in the remainder of this section.

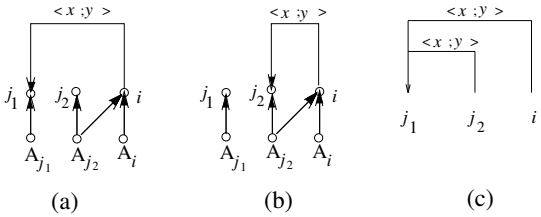


Figure 8.3: AF # CF with the AF in the coupling cell

As already introduced in Chapter 7, the address orders of all linked faults consisting of an AF and a CF (or two CFs) can be split into three address order classes:

- *Low class*: $< x; y > ik_p \ \# < x; y > ik_q$ The *coupled* cell has the lowest address, and involves two simple faults (Figure 8.4(a & b));
- *Middle class*: $< x; y > ji \ \# < x; y > ik$ The *coupled* cell lies between the two coupling addresses (Figure 8.4(c & d));
- *High class*: $< x; y > j_p i \ \# < x; y > j_q i$ The *coupled* cell has the highest address, and involves two simple faults (Figure 8.4(e & f)).

All AF # CF of the Middle class, see Figure 8.4(c & d), will be detected by a march test which can detect AFs or CFs as simple faults, because in this case masking is not possible and one of the faults can be detected before the other will be sensitized.

Condition 1L : Any march test will detect an 'AFsr # CF'; i.e., an AFsr linked with a *single* CF, if this march test satisfies the following three subconditions:

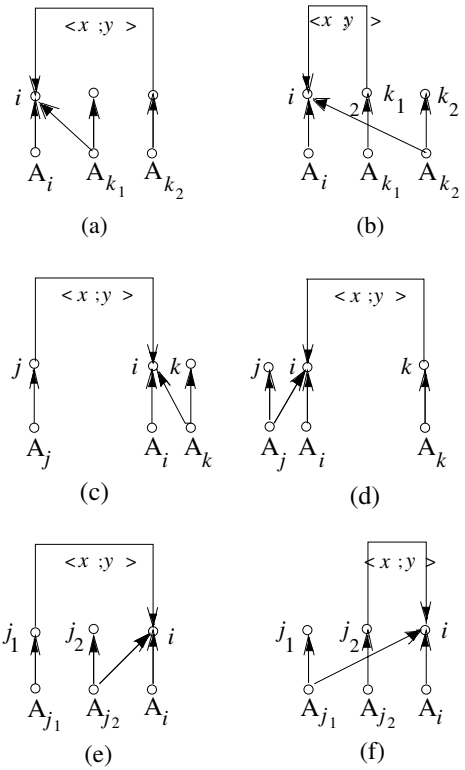


Figure 8.4: AF # CF with the AF in the coupled cell

1. the march test detects SOFs. This is required for faults A and B of Figure 8.1 (see Condition 2);

2. the march test detects all simple CFs (Condition 5);
   Please note that in case the *simple* CF is a CFin, the linked fault will be detected if and only if (iff) the CFin is sensitized and detected by a *single* march element (in order to avoid masking by the AF). This is important for the linked faults of Figure 8.4(b & f) because the CFin and AF may mask each other. The fault of Figure 8.4(b) will be detected by the first march element of Case A.2, in case of the fault $< y; \bar{x} >$; or Case B.2, in case of the fault $< y; x >$ of Condition 5. Only Case A.2 *or* Case B.2 is required because a CFin behaves as a $< y; x >$, when the expected value of the coupled cell is $\bar{x}$, *and* also as $< y; \bar{x} >$. Similarly the fault of Figure 8.4(f) will be detected by the first march element of Case A.1 or B.1 of Condition 5;

3. the march test detects all simple AFs (Condition 1sr).

The faults of Figure 8.4(b & f) will be detected because of Condition 1L(2); this is due to the fact that the CF will be detected because the address of the coupling cell of the CF is closer to the address A_i (of the coupled cell) than the address A_{j_2} or A_{k_2} of the AF. The faults of Figure 8.4(a & e) will be detected because of Condition 1L(3); and the faults of Figure 8.4(c & d) will be detected by any march test which satisfies Condition 1L(2) *or* Condition 1L(3).

When the particular case of AF linked with CFin is considered a special condition can be defined. We would like to comment on the feasibility of such faults considering the concerns related to the validity of the CFin fault model [74].
Condition 5I: A march test which has to detect a CFin # AF has to satisfy Condition 5 in one of the two following ways: Case A.1 together with Case B.2 *or* Case A.2 together with Case B.1 (i.e., both address orders are required; see Condition 1L(2)).

Table 8.2 summarizes the conditions a march test must satisfy in order to detect a certain combination of 'AF # other fault' (C. is a shorthand for Condition). As mentioned earlier the CFin combinations are not considered of practical interest and are not presented in the table. The interested reader can expand the table using the Condition 5I introduced above.

Table 8.2: Conditions for 'AF # one fault'

Type address fault	'Other fault'					
	AF	SAF	SOF	TF	DRF	CF
AFsr	C.1tr	C.1sr	C.1sr	C.1sr	C.1sr	C.1L
AF	C.1tr	C.1tr	C.2	C.1sr	C.1tr	C.1L
AFdr	C.1dr	C.1dr	C.2	C.1sr	C.1dr	C.1L

8.2.2 AF # more than *one* coupling fault

A special case of interest is when more than one CF is involved in the fault ('AF # more than one fault'). For this case the following observations hold:

1. An AFsr *dominates* any MCAFs if the AFsr is of subtype *A* or *B* of Figure 8.1 (Proposition 4);

2. For the subtypes *C* and *D* of Figure 8.1 it is important that the march test be able of detecting the simple AF *and* each of the subtypes of the simple

CFs involved in the linked fault with a *single* march element. Assume the fault of Figure 8.5(a), involving simple faults of the Low part (i.e., all faults with cell addresses which are less than or equal to the coupled cell address). Note that if a march test can detect the AF involved in this fault (thus the test has to have a march element: $\Uparrow (rx, \ldots, w\bar{x})$ in order to detect the AF) the detection of the entire fault is ensured. In case of the fault of Figure 8.5(b), the detection of the last $(< x; y > j_a i)$ CF ensures the detection of the entire fault iff the fault is detectable by a single march element such that masking cannot occur. Similar arguments apply to the simple faults of the High part of the 'AF # more than one fault'.

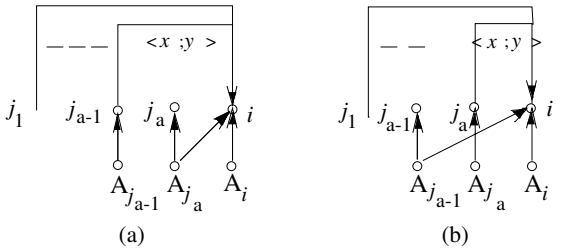


Figure 8.5: AF # (more than one) CFs

Condition 1LA: Any march test will detect 'AF # CFs'; i.e., an AF linked with *any* number of CFs, if the test satisfies the following two subconditions (see above):

1. the march test detects all simple CFs in such a way that faults of each subtype are sensitized and detected by a *single* march element;

2. the march test detects all simple AFs.

8.3 March test coverage for linked address decoder faults

Hereafter we propose a test for all simple faults as well as all linked faults involving an arbitrary number of simple faults.
Theorem 1: Assume a multiple linked fault, let **S**= $\{f_1, f_2, ...\}$ be the set of involved simple coupling faults. If a particular march test **M** guarantees the detection of each of the faults $< y; x > j_a i$ and $< y; x > i k_1$ (j_a and k_1 are the coupling cells with the closest addresses to the coupled cell) of **S** with a *single* march element, then that march test can detect linked faults consisting of any

number of faults of **S**. Exceptions consist of some faults that are not detectable by linear tests [72].
Proof: If a march element of **M** can *sensitize and detect* the simple fault in position $j_a i$ (which means the march element has to have an $\Uparrow$ address order) all faults with lower addresses are irrelevant because the fault in $j_a i$ will always be detected by this march element. If a march element of **M** can *sensitize and detect* the fault in position ik_1 (which implies that the march element has a $\Downarrow$ address order) all faults with higher addresses are irrelevant because the fault in ik_1 will always be detected by this march element.
Below the test March LA will be derived, using MOVI as a starting point. It will be shown that March LA can detect all simple faults as well as all linked faults. More precisely, all linked CFdsts, CFins, CFids, CFsts and TFs as well as all linked faults involving AFs; the main focus of this chapter.

Suppose $\mathbf{S} = \{ < r0; \uparrow >, < r0; \downarrow >, < r1; \uparrow >, < r1; \downarrow >, < w0; \uparrow >, < w0; \downarrow >, < w1; \uparrow >$ and $< w1; \downarrow > \}$, which is the set of all CFdsts. Now we will start with the test MOVI [21, 5] as a starting point, and modify it such that it conforms to Theorem 1 for **S** (the result will be *March LA*, which detects all linked CFdsts, see Figure 8.7). By inspecting MOVI, see Figure 8.6, it is not

MOVI = {		
$\Downarrow (w0);$	$\Uparrow (r0, w1, r1);$	$\Uparrow (r1, w0, r0);$
M0	M1	M2
$\Downarrow (r0, w1, r1);$	$\Downarrow (r1, w0, r0)\}$	
M3	M4	

Figure 8.6: MOVI

difficult to see that some operations required to sensitize all CFdsts subtypes are missing. If M1, in addition, contains the $w0$ operation, then all CFdsts of type $< p; \uparrow > j_a i$, $p \in \{r0, r1, w0, w1\}$, will be sensitized and detected by M1 (see Figure 8.7). The same applies to M2: if M2 in addition contains the $w1$ operation, all faults of type $< p; \downarrow > j_a i$ will be sensitized and detected. M3 and M4 are required to allow sensitization and detection of the faults $< p; \uparrow > ik_1$ and $< p; \downarrow > ik_1$, respectively. From the above discussion it should be obvious that M1 can detect all CFdsts of subtype $< p; \uparrow > j_a i$, M2 can detect all CFdsts of subtype $< p; \downarrow > j_a i$, M3 can detect all CFdsts of subtype $< p; \uparrow > ik_1$, and M4 can detect all CFdsts of subtype $< p; \downarrow > ik_1$, $p \in \{r0, r1, w0, w1\}$. Because March LA satisfies Theorem 1, all linked CFdsts (involving an arbitrary number of CFdsts) will be detected. Below it will be shown that March LA also detects: linked CFins, CFids and CFsts, linked faults whereby the coupling cell influences the coupled cell in more than one way, and AFs linked with AFs or MCAFs.
AFs # AFs (or MCAFs except for DRF) detection: March LA detects

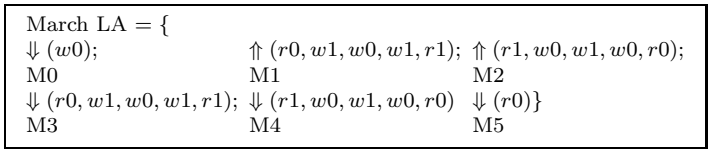


Figure 8.7: March LA

all AFs linked with AFs or MCAFs because March LA detects SOFs and satisfies Condition 1LA as follows:
Condition 1LA(1) is satisfied by March LA via march elements M1 through M4 and Condition 1LA(2) is satisfied via march elements M1 and M4 or via M2 and M3.

Linked CFins, CFids and CFsts detection: In addition to linked CFdsts (involving any number of CFdsts) and linked AFs, March LA (which has a test length of $22n$) also detects linked CFins, CFids, and CFsts, as shown below (a TF involved in a linked fault will be detected because Condition 2 is satisfied by M1, M2, M3 and M4).

Linked CFins which do not satisfy Proposition 1 will be detected : assume the CFin $<\uparrow;\updownarrow> j_a i$. The fault will be sensitized twice by M1 (because M1 contains two w1 operations) such that the fault effect is canceled and therefore cannot be detected (the fault in position $j_{a-1} i$ can be considered as the last fault in this case and is easily detected because it is not another CFin due to Proposition 1); however this fault will be sensitized and detected by M2. The CFin $<\downarrow;\updownarrow> j_a i$ will, similarly, be detected by M1 (and not by M2); the CFin $<\uparrow;\updownarrow> i k_1$ will be detected by M4, and the CFin $<\downarrow;\updownarrow> i k_1$ by M3.

Each of the march elements M1 through M4 sensitizes all CFids. The CFid $< q;\uparrow> j_a i$, $q \epsilon \{\uparrow,\downarrow\}$, will be sensitized and detected by M1; the CFid $< q;\downarrow> j_a i$, will be sensitized and detected by M2, etc.

CFsts dominate other faults (see Proposition 2). First assume a 'CFst # any *other* CFs'; the CFst may have its coupling cell in position j_x, where $x \epsilon \{1 \ldots a\}$ (lower address); or k_y where $y \epsilon \{1 \ldots b\}$ (higher address). Because CFsts cannot be masked (because of the dominance property); they will be detected by March LA if all CFst subtypes are sensitized and detected, for both cases: the coupling cell in position j_x and in position k_y. For example the CFst $< 1; 1 > j_x i$ will be sensitized and detected by M1 (because the coupling cell is left in the sensitized state "1" by the last "$w1$" operation of M1, and thereafter detected by the first "$r0$" operation of M1). The CFst $< 0; 1 > j_x i$ will be sensitized by M1 and detected by M2. Because of the symmetry of the faults and the march elements, all CFsts will be detected; however the CFst $< 0; 1 > i k_y$ requires the extra march element M5: $\Downarrow (r0)$. Linked faults consisting of more than two CFsts will be detected by March LA, iff those linked

faults do not satisfy Proposition 3(b.3).

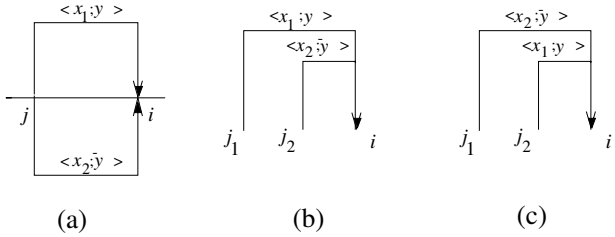


Figure 8.8: Multiple linked fault with single coupling cell

Linked faults whereby the coupling cell influences the coupled cell in more than one way: The linked fault of Figure 8.8(a), i.e. linked faults whereby a single coupling cell influences the coupled cell in more than one way, will also be detected by March LA. In order to show this the address orders of all linked faults consisting of two CFs, or an AF and a CF, can be split into the three already introduced address order classes; *Middle*, *High* and *Low*.

Now, consider the linked fault of Figure 8.8(a) which consists of the fault $< x_1; y >$ (with sensitizing condition x_1 and fault effect y) linked with the fault $< x_2; \bar{y} >$. Because the fault of Figure 8.8(a) is of the High class, it has to be detected by M1 or M2. Assume that when M1 is applied the fault behaves as shown in Figure 8.8(b); i.e., the fault $< x_1; y >$ is sensitized first and thereafter the fault $< x_2; \bar{y} >$. Then, when M2 is applied, the fault will behave according to Figure 8.8(c) because the sequence of sensitizing operations of M2 is the exact inverse of those of M1.

If masking occurs when M1 is applied, this means that the fault effect of $< x_2; \bar{y} > = < x_2; 0 >$ (i.e., $\bar{y} = 0$) in case Figure 8.8(b) applies, the same linked fault will also be masked when M2 is applied because the last sensitized fault $< x_1; y > = < x_1; 1 >$; i.e., $y = 1$ which is the expected value in cell 'i'. However, the linked fault $< x_1; 1 > \# < x_2; 0 >$ will be sensitized by M3 and detected by M4; because the sensitizing operations of M3 are identical of those of M1, but the expected values are different. Hence, the fault effect $< x_2; 0 >$ will be sensitized by M3 and detected by M4.

Faults of the Low class which are not detected by M3 and M4, similarly, will be sensitized by M1 and detected by M2.

March LA [77] was derived to deal with all linked faults based on the fault primitives as known at that time. In respect to linked and unlinked AFs, this march test is compared with a list of well-known memory tests in Table 8.3. This table describes the result of applying the conditions for detecting AFs to

Table 8.3: Test length and fault coverage of (linked) address decoder faults

March test	Length	Unlinked AFs			Linked AFs			
		C 1tr	C 1dr	C 1sr	C 2	C 1T	C 1L	C 1LA
MATS [19]	$4n$	-	-	-	-	-	-	-
MATS+ [19]	$5n$	+	+	-	-	-	-	-
MATS++ [5]	$6n$	+	+	-	-	-	-	-
Marching 1/0 [18]	$14n$	+	+	-	-	-	-	-
March X [5]	$6n$	+	+	-	-	-	-	-
March C [20]	$11n$	+	+	-	-	-	-	-
March C- [5]	$10n$	+	+	-	-	-	-	-
March A [53]	$15n$	+	+	-	-	-	-	-
March Y [5]	$8n$	+	+	-	-	-	-	-
March B [53]	$17n$	+	+	+	+	+	+	-
Algorithm B [20]	$17n$	+	+	+	+	+	+	-
MOVI [21]	$13n$	+	+	+	+	+	-	-
IFA-6 [6]	$6n$	+	+	-	-	-	-	-
IFA-9 [6]	$12n + 2D$	+	+	-	-	-	-	-
IFA-13 [6]	$16n + 2D$	+	+	+	+	+	-	-
March G [73]	$23n + 2D$	+	+	+	+	+	+	-
March GS [73]	$24n + 2D$	+	+	+	+	+	+	-
March M [75]	$16n$	+	+	+	+	+	+	-
March U [76]	$13n$	+	+	+	+	+	+	-
March U- [76]	$12n$	+	+	-	-	+	+	-
March UD [76]	$13n + 2D$	+	+	+	+	+	+	-
March UD- [76]	$12n + 2D$	+	+	-	-	+	+	-
March LR [45]	$14n$	+	+	+	+	+	+	-
March LRD [45]	$15n + 2D$	+	+	+	+	+	+	-
March LRDD [45]	$18n + 2D$	+	+	+	+	+	+	-
March LA [77]	$22n$	+	+	+	+	+	+	+

the list of well-known memory tests. The majority of the existing tests are unable to detect simple AFsrs; i.e., they do not satisfy Condition 1sr. Only a few tests are able to detect an AFsr # CF (i.e., they satisfy Condition 1L); while only March LA, can detect all linked AFs (AFs linked with another AF or with any number of faults of any type). This is because only March LA satisfies Condition 1LA and complies to Theorem 1. In [72] a more elaborate study of the fault coverage of March LA for different MCAF combinations is presented. In addition, three derivatives of it; March LA-, March LAD and March LADD-, are carefully investigated. Those optimizations fall out of the scope of this chapter since they are concerned with specific CF cases and not focusing on the AFs. The results shown in Table 8.3 confirm our claim that when linked AF are targeted only March LA can provide sufficient fault coverage for the cost of increased test length ($22n$).

8.4 Conclusions

A new classification of the AF fault model has been introduced. It has been shown, that when the memory technology is taken into consideration, different requirements exist in order to detect all AFs. The traditionally used AF model (AFtr) has been considered as unrealistic due to the internal structure of SRAMs and DRAMs.

The set of conditions march tests have to satisfy in order to detect AFs was derived for all simple and all linked AFs. Also the list of propositions was augmented to set the margins of abilities of the march tests, in cases when AFs are involved in the linked fault. March LA was derived and its superiority in detecting linked faults when one of the faults is AF was shown.

Chapter 9

Conclusions

In this thesis we proposed and classified DRAM specific fault models relevant for state-of-the-art semiconductor technologies as available in 2007. We defined and validated a set of DRAM specific march tests to cope with the detection of the above models. We also proposed various optimizations for test time reduction and/or increased fault coverage for DRAM. In respect to linked faults, that are increasing in importance with the deep sub-micron technologies, we proposed a systematic methodology for deriving conditions and tests. We also carefully investigated the detection conditions of linked memory faults when one of the faults is an address decoder fault. For both cases appropriate march tests March LR (and its derivatives) and March LA were introduced. The fault coverage of these tests was compared to classic march test widely used by the industry. March LR has been widely adopted by the industry, i.e. HP used it in their Built in Self Test (BIST) engine used to test the cache memory of the PA8500 processor. In addition, Synopsis offers a commercial BIST IP-module that utilizes March LR.

9.1 Major Contributions

The major contributions of this thesis are:

- The definition of extended space of memory faults including linked faults. It is a general taxonomy that describes any possible faulty behavior in modern memories. Three distinct classes are defined with respect to the number of cells, operations and respectively fault primitives involved. This general space is defined in Chapter 3.

- The establishment of DRAM specific fault models and corresponding march tests to detect them. The complete set of two-operation, single-cell dynamic faults has been first evaluated using real industrial products

(Chapter 4). In Chapter 5 the proposed fault models were first evaluated using SPICE simulation. Thereafter six different march tests, March H1C, March H2C, March T1C, March T2C, March S1C and March S2C, were proposed for single and two-cell hard, transient and soft faults respectively. The tests for transient and soft faults were additionally optimized for cases when the memory layout is known in Chapter 6.

- The concept of linked memory faults was thoroughly studied. The huge space of all theoretically possible linked faults was reduced to a much smaller sub set of manageable size, very likely to occur (realistic) linked faults. A set of conditions march tests have to satisfy in order to detect all realistic faults were proposed. March LR and its derivatives were proposed to detect this fault class. Industrial evaluation supported our theoretical claim on this issue. This was the topic of Chapter 7.

- The careful investigation of address decoder faults unlinked and linked with other simple faults. In respect to the unlinked AF faults the precise implementation technology was taken in consideration. A set of conditions to detect all AFs was presented. In addition, the set of conditions was extended with the case when one of the faults involved in a linked fault is AF. March LA was proposed to deal with all linked address decoder faults. The linked address decoder faults were dealt with in Chapter 8.

9.2 Open issues

The following open issues can be considered for future work on the topic:

- Investigation of the case when multiple address decoder faults are involved in a linked fault. With the future decrease of the feature sizes the chance to have memory circuits with more then one AF operating on the same victim cell is expected to grow. This is why it is worth proceeding in this direction.

- Design of a BIST circuits that would generate the tests presented in this thesis. Especially the more recent DRAM specific march tests need more attention. March LR was already implemented and commercialized as a BIST engine by Synopsis.

- Investigation of built in self repair (BISR) algorithms that closely collaborate with the proposed march tests. It is envisioned that using only fault bit map as interface between the test and the repair algorithm, some of the information available while testing is lost. Novel ways for suck close collaboration are expected to increase the quality of the repair action.

Bibliography

[1] "International Technology Roadmap for Semiconductors (ITRS)," 2005. [Online]. Available: http://public.itrs.net

[2] Y. Zorian and S. Shoukourian, "Embedded-memory test and repair: infrastructure IP for SoC yield," in *Design & Test of Computers*, vol. 20, no. 3. IEEE, May-June 2003, pp. 58–66.

[3] S. Kuo and W. Fuchs, "Efficient spare allocation for reconfigurable arrays," in *IEEE Design & Test of Computers*, vol. 4, no. 1, February 1987, pp. 24–31.

[4] M. Inoue, T. Yamada, and A. Fujiwara, "A New Testing Acceleration Chip for Low-Cost Memory Tests," in *IEEE Design & Test of Computers*, vol. 1, 1993, pp. 15–19.

[5] A. van de Goor, *Testing Semiconductor Memories, Theory and Practice*. Gouda, The Netherlands: ComTex Publishing, 1998.

[6] R. Dekker, F. Beenker, and L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memories," in *IEEE Transactions on Computer-Aided Design*, vol. C9, no. 6, June 1990, pp. 567–572.

[7] I. Schanstra and A. van de Goor, "Industrial evaluation of Stress Combinations for March Tests Applied to SRAMs," in *IEEE International Test Conference*, 1999, pp. 983–992.

[8] A. van de Goor and J. de Neef, "Industrial Evaluation of DRAMs Tests," in *Design Automation and Test in Europe*, March 1999, pp. 623–630.

[9] D. Adams and E. Cooley, "Analysis of Deceptive Read Destructive Memory Fault Model and Recommended Testing," in *IEEE North Atlantic Test Workshop*, May 1996.

[10] Z. Al-Ars and A. van de Goor, "Impact of Memory Cell Array Bridges on the Faulty Behavior in Embedded DRAMs," in *Asian Test Symposium*, 2000, pp. 282–289.

[11] S. Hamdioui and A. van de Goor, "Experimental Analysis of Spot Defects in SRAMs: Realistic Fault Models and Tests," in *Ninth Asian Test Symposium*, 2000, pp. 131–138.

[12] A. van de Goor and Z. Al-Ars, "Functional Fault Models: A Formal Notation and Taxonomy," in *Proceedings of the IEEE VLSI Test Symposium*, 2000, pp. 281–289.

[13] Z. Al-Ars and A. J. van de Goor, "Static and Dynamic Behavior of Memory Cell Array Opens and Shorts in Embedded DRAMs," in *Proceedings of Design Automation and Test in Europe*, 2001, pp. 496–503.

[14] S. Hamdioui, Z. Al-ars, and A. van de Goor, "Testing Static and Dynamic Faults in Random Access Memories," in *Proceedings of the IEEE VLSI Test Symposium*, 2002, pp. 395–400.

[15] S. Hamdioui, R. Wadsworth, J. D. Reyes, and A. J. van de Goor, "Importance of Dynamic Faults for New SRAM Technologies," in *European Test Workshop*, 2003, pp. 29–34.

[16] M. Nicolaidis, "Design for Soft Error Robustness To Rescue Deep Submicron Scaling," in *Proceedings of ITC*, 1998, p. 1140.

[17] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," in *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, Sept 2005, pp. 305–316.

[18] M. Breuer and A. Friedman, *Diagnosis and Reliable Design of Digital Systems*. Woodland Hills, CA, USA: Computer Science Press, 1976.

[19] R. Nair, "An Optimal Algorithm for Testing Stuck-at Faults Random Access Memories," in *IEEE Transactions on Computers*, vol. C-28, no. 3, 1979, pp. 258–261.

[20] M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," in *Proceedings of the IEEE International Test Conference*, 1982, pp. 236–239.

[21] J. D. Jonge and A. Smeulders, "Moving Inversions Test Pattern is Thorough, Yet Speedy," in *Computer Design*, 1976, pp. 169–173.

[22] S. Hamdioui, A. van de Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," in *Proceedings of the IEEE International Workshop on Memory Technology, Design and Test*, 2002, pp. 95–100.

[23] B. Vermeulen, C. Hora, B. Kruseman, E. Marinissen, and R. van Rijsinge, "Trends in testing integrated circuits," in *Proceedings of the IEEE International Test Conference*, Oct 2004, pp. 688–697.

[24] S. McGregor, "The Eye of The Storm in Focus," in *EE Times*, 2003. [Online]. Available: http://www.eet.com/story/OEG20030701S0040

[25] B. S. Meyerson, "42nd DAC keynote: How Does One Define "Technology" Now That Classical Scaling Is Dead (and Has Been for Years)?" 2005.

[26] Z. Al-Ars, S. Hamdioui, G. Mueller, and A. van de Goor, "Framework for Fault Analysis and Test Generation in DRAMs," in *Proceedings of Design, Automation and Test in Europe*, 2005, pp. 1020–1021.

[27] J. Jahangiri and D. Abercrombie, "Value-Added Defect Testing Techniques," in *IEEE Design & Test of Computers*, vol. 22, no. 3, May 2005, pp. 224–231.

[28] R. Bergamaschi and J. Cohn, "The A to Z of SoCs," in *Proceedings of IEEE/ACM International Conference CAD*, November 2002, pp. 791–798.

[29] R. Wilson, "Deck stacked against SoC," in *EE Times*, March 2003. [Online]. Available: http://www.eetimes.com/op/showArticle.jhtml?articleID=18308205

[30] J. Schmid and J. Knablein, "Advanced synchronous scan test methodology for multi clock domain ASICs," in *Proceedings of the IEEE VLSI Test Symposium*, April 1999, pp. 106–113.

[31] S. Abdennadher and S. Shaikh, "Challenges in High Speed Interface Testing," in *Proceedings of the Asian Test Symposium*, 2005, pp. 468–468.

[32] A. Amory, K. Goossens, E. J. Marinissen, M. Lubaszewski, and F. Moraes, "Wrapper design for the reuse of networks-on-chip as test access mechanism," in *Proc. European Test Symposium (ETS)*, May 2006, pp. 213–218.

[33] F. Caignet, S. Delmas-Bendhia, and E. Sicard, "The Challenge of Signal Integrity in Deep-Submicrometer CMOS Technology," in *Proceedings of the IEEE*, vol. 89, no. 4, April 2001, pp. 556–573.

[34] M. Nourani and A. Attarha, "Built-In Self-Test for Signal Integrity," in *Proceedings of Design Automation Conference*, 2001, pp. 792–797.

[35] Y. Cao, P. Gupta, A. Kahng, D. Sylvester, and J. Yang, "Design Sensitivities to Variability: Extrapolations and Assessments in Nanometer VLSI," in *Proceedings of the 15-th IEEE International ASIC/SOC Conference*, September 2002, pp. 411–415.

[36] S. Kundu, S. Sengupta, and R. Galivanche, "Test challenges in nanometer technologies," in *Proceedings of the IEEE European Test Workshop*, May 2000, pp. 83–90.

[37] A. Agarwal, V. Zolotov, and D. Blaauw, "Statistical Timing Analysis Using Bounds and Selective Enumeration," in *IEEE Transactions on CAD*, vol. 22, no. 9, September 2003, pp. 1243–1260.

[38] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proceedings of the International Conference Dependable Systems and Networks*, June 2002, pp. 389–398.

[39] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: an architectural perspective," in *Proceedings of the 11-th International Symposium on High-Performance Computer Architecture*, February 2005, pp. 243–247.

[40] S. Hamdioui, G. Gaydadjiev, and A. van de Goor, "The State-of-art and Future Trends in Testing Embedded Memories," in *Proceedings of IEEE International Workshop on Memory Technology, Design and Testing*, August 2004, pp. 54–59.

[41] S. Kundu, T. Mak, and R. Galivanche, "Trends in Manufacturing Test Methods and Their Implications," in *Proceedings of the IEEE International Test Conference*, October 2004, pp. 679–688.

[42] A. Crouch, "Future Trends in Test: The Adoption and Use of Low Cost Structural Testers," in *Proceedings of IEEE International Test Conference*, 2004, pp. 698–703.

[43] D. Das and N. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid Test Patterns," in *Proceedings of IEEE International Test Conference*, October 2000, pp. 115–122.

[44] R. Adams and E. Cooley, "Analysis of a Deceptive Destructive Read Memory Fault Model and Recommended Testing," in *Proceedings of IEEE North Atlantic Test Workshop*, Hanover, NH, June 1996.

[45] A. van de Goor, G. Gaydadjiev, V. Jarmolik, and V. Mikitjuk, "March LR: A Test for Realistic Linked Faults," in *14-th IEEE VLSI Test Symposium*, 1996, pp. 272–280.

[46] A. van de Goor and J. de Neef, "Industrial Evaluation of DRAM Tests," in *Proceedings of the Design, Automation and Test in Europe (DATe '99)*, 1999, pp. 623–630.

[47] A. J. van de Goor and J. Simonse, "Defining sram resistive defects and their simulation stimuli," in *Eighth Asian Test Symposium: proceedings. ATS '99*, November 1999, pp. 33–40.

[48] S. Borri, M. Hage-Hassan, P. Girard, S. Pravossoudovitch, and A. Virazel, "Defect Oriented Dynamic Fault Models for Embedded SRAMs," in *Proceedings of the European Test Symposium*, 2003, pp. 23–28.

[49] Z. Al-Ars and A. van de Goor, "Approximating Infinite Dynamic Behavior for DRAM Cell Defects," in *Proceedings of the IEEE VLSI Test Symposium*, 2002, pp. 401–406.

[50] A. van de Goor and C. Verruijt, "An Overview of Deterministic Functional RAM Chip Testing," in *ACM Computing Surveys*, vol. 22, no. 1, 1995, pp. 5–33.

[51] W. Huott, M. McManus, D. Knebel, S. Steen, D. Manzer, P. Sanda, S. Wilson, Y. Chan, A. Pelella, and S. Polonsky, "The Attack of the 'Holey Shmoos': A Case of the Advanced DFD and Picosecond Imaging Circuit Analysis (PICA)," in *Proceedings of the IEEE International Test Conference*, 1999, pp. 883–891.

[52] M. Abadir and J. Reghbati, "Functional Testing of Semiconductor Random Access Memories," in *ACM Computer Surveys*, vol. 15, no. 3, 1983, pp. 175–198.

[53] D. Suk and S. Reddy, "A March Test for Functional Faults in Semiconductor Random-Access Memories," in *IEEE Transactions on Computers*, vol. C-30, no. 12, 1981, pp. 982–985.

[54] A. van de Goor and G. Gaydadjiev, "March LR: A Memory Test for Realistic Linked Faults," in *Proceedings of the IEEE VLSI Test Symposium*, 1996, pp. 272–280.

[55] A. van de Goor and S. Hamdioui, "Detecting Faults in Peripheral Circuits and an Evaluation of SRAM Tests," in *Proceedings of the IEEE International Test Conference*, USA, October 2004, pp. 114–134.

[56] B. Prince, "Application Specific DRAMs Today," in *Proceedings of the IEEE International Workshop Memory Technology, Design and Testing*, 2003, pp. 7–13.

[57] T. Falter and D. Richter, "Overview of Status and Challenges of System Testing on Chip with Embedded DRAMs," in *Solid-State Electronics*, no. 44, 2000, pp. 761–766.

[58] G. Antonin, H.-D. Oberle, and J. Kolzer, "Electrical Characterization of Megabit DRAMs, 1. External Testing," in *IEEE Design & Test of Computers*, vol. 8, 1991, pp. 36–43.

[59] Z. Al-Ars, "DRAM Fault Analysis and Test Generation," Ph.D. dissertation, Delft University of Technology, Delft, the Netherlands, 2005.

[60] A. van de Goor and A. Paalvast, "Industrial Evaluation of DRAM SIMM Tests," in *Proceedings of the IEEE International Test Conference*, 2000, pp. 426–435.

[61] Z. Al-Ars, A. van de Goor, and S. Hamdioui, "Space of DRAM Fault Models and Corresponding Testing," in *Proceedings of Design, Automation and Test in Europe*, 2006, pp. 1–6.

[62] A. Keshavarzi, K. Roy, and C. Hawkins, "Intrinsic Leakage in Low Power Deep Submicron CMOS ICs," in *Proceedings of the IEEE International Test Conference*, 1997, pp. 146–155.

[63] S. Naik, F. Agricola, and W. Maly, "Failure Analysis of High Density CMOS SRAMs," in *IEEE Design and Test of Computers*, vol. 10, no. 2, 1993, pp. 13–23.

[64] N. Nagi and J. Abraham, "Hierarchical Fault Modeling for Linear Analog Circuits," in *Analog Integrated Circuits and Signal Processing*, vol. 10, no. 1–2, 1996, pp. 89–99.

[65] B. Prince, *Semiconductor Memories: A Handbook of Design Manufacturing and Application*, 2nd ed. West Sussex, UK: John Wiley & Sons, 1991.

[66] G. Harutunyan, V. Vardanian, and Y. Zorian, "Minimal March Tests for Unlinked Static Faults in Random Access Memories," in *Proceedings IEEE VLSI Test Symposium*, 2005, pp. 53–59.

[67] M.-J. Wang, R.-L. Jiang, J.-W. Hsia, C.-H. Wang, and J. E. Chen, "Guardband determination for the detection of off-state and junction leakages in dram testing," in *Proceedingsof the Asian Test Symposium*, November 2001, pp. 151–156.

[68] Z. Al-Ars, S. Hamdioui, A. J. van de Goor, G. N. Gaydadjiev, and J. Vollrath, "Dram-specific space of memory tests," in *Proc. IEEE International Test Conf.*, October 2006.

[69] Z. Al-Ars and A. van de Goor, "Transient Faults in DRAMs: Concept, Analysis and Impact on Tests," in *Proceedings of IEEE International Workshop on Memory Technology, Design and Testing*, 2001, pp. 59–64.

[70] H.-D. Oberle and P. Muhmenthaler, "Test Pattern Development and Evaluation for DRAMs with Fault Simulator RAMSIM," in *Proceedings of the IEEE International Test Conference*, October 1991, pp. 548–555.

[71] C. A. Papachristou and N. Sahgal, "An Improved Method for Detecting Functional Faults in Random-Access Memories," in *IEEE Transactions on Computers*, vol. C-34, no. 2, 1985, pp. 110–116.

[72] G. N. Gaydadjiev and A. van de Goor, "An Analysis of Linked Faults," Delft University of Technology, Tech. Rep. 1- 68340-44(1995)08, 1995.

[73] A. J. van de Goor, "Using March Tests to Test SRAMs," in *IEEE Design & Test of Computers*, March 1993, pp. 8–14.

[74] R. Dekker, F. Beenker, and L. Thijssen, "Fault Modeling and Test Algorithm Development for Static Random Access Memories," in *IEEE International Test Conference*, Washington D.C., 1988, pp. 343–352.

[75] V. Mikitjuk, V. Yarmolik, and A. van de Goor, "RAM Testing Algorithms for Detection Multiple Linked Faults," in *Proceedings of the European Test Conference*, 1996, pp. 435–439.

[76] A. J. van de Goor and G. N. Gaydadjiev, "March U: A Test for Unlinked Memory Faults," *IEEE Proceedings on Circuits, Devices and Systems*, pp. 155–160, June 1997.

[77] A. J. van de Goor, G. N. Gaydadjiev, V. N. Yarmolik, and V. Mikitjuk, "March LA: A Test for All Linked Memory Faults," in *Seventh Asian Test Symposium*, December 1998, pp. 1–8.

Testen van Moderne Halfgeleider Geheugenstructuren

Georgi Nedeltchev Gaydadjiev

Samenvatting

In dit proefschrift bestuderen wij het probleem van fouten in moderne geheugenstructuren. Zoals vermeld in de ITRS 2005, de systemen op chip (SoC) zijn aan het veranderen van systemen die gebalanceerd zijn met betrekking tot logica en geheugen naar systemen die door het geheugen gedomineerd worden. Het ingebedde geheugen zal naar verwachting in 2009 meer dan 60 procent van het totale chipoppervlak in beslag nemen. Verder zullen de capaciteiten van de ingebedde geheugens blijven groeien. Als resultaat zou de productieopbrengst van SoC systemen door de opbrengst van het geheugen gedomineerd raken. Deze trend kan ervoor zorgen dat de algemene opbrengst onacceptabel laag wordt, tenzij bepaalde maatregelen worden genomen.

In dit proefschrift stellen wij DRAM specifieke foutmodellen voor ten behoeve van de recente halfgeleidertechnologieën. Ook definiëren en valideren wij alle DRAM specifieke *march* testen. Verder introduceren wij een methodiek voor het afleiden van condities en testen gericht op verbonden geheugenfouten. Wij onderzoeken de detectiecondities van verbonden fouten wanneer een van de fouten in de adres decodeerder zit. Uiteindelijk stellen wij meerdere optimalisaties voor die de testtijd reduceren en/of de foutdekking verbeteren.

Ons doel in dit proefschrift is om ideeën met hoge relevantie aan te dragen. Voor zover mogelijk zijn onze foutmodellen en testen met behulp van reeële industriële producten gevalideerd. Sommige concepten die meer dan 10 jaar geleden zijn gepresenteerd, worden nog steeds gebruikt in de industrie en geciteerd in de academische wereld. Bijvoorbeeld, veel industriële producten hebben gebruik gemaakt of maken gebruik van March LR, een van de testen afgeleid in dit proefschrift, voor het testen van hun (ingebedde) geheugens.

Bibliography of the author

Journal Publications

1. L. Mhamdi, G. N. Gaydadjiev, S. Vassiliadis, Efficient Multicast Support in High-Speed Packet Switches, Journal of Networks, Vol. 2 No. 3, pp. 28-35, June 2007

2. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, Multimedia Rectangularly Addressable Memory, IEEE Transactions on Multimedia, pp. 315–322, April 2006

3. S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G.K. Kuzmanov, E. Moscu Panainte, The Molen Polymorphic Processor, IEEE Transactions on Computers, pp. 1363- 1375, November 2004, Volume 53, Issue 11

4. A. J. van de Goor, G. N. Gaydadjiev, March U: A Test for Unlinked Memory Faults, Circuits, Devices and Systems, IEEE Proceedings, pp. 155-160, June 1997, vol. 144, no. 3

Conference Publications

1. Z Chang, G. N. Gaydadjiev, S. Vassiliadis, Infrastructure for Cross-Layer Designs Interaction, In Proceedings of the 16th IEEE International Conference on Computer Communications and Networks (IC3N), Honolulu, Hawaii USA, August 2007

2. R. J. Meeuws, Y. D. Yankova, K.L.M. Bertels, G. N. Gaydadjiev, S. Vassiliadis, A Quantitative Prediction Model for Hardware/Software Partitioning, In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07), Amsterdam, The Netherlands, August 2007

3. Y. D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G. N. Gaydadjiev, J. Lu, S. Vassiliadis, DWARV: DelftWorkbench Automated Reconfigurable VHDL Generator, In Proceedings of the 17th International Conference

on Field Programmable Logic and Applications (FPL07), Amsterdam, The Netherlands, August 2007

4. D.R.H. Calderon, G. N. Gaydadjiev, S. Vassiliadis, Reconfigurable Universal Adder, In Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP 07), Montreal, Quebec, Canada, July 2007

5. D.R.H. Calderon, C Galuzzi, G. N. Gaydadjiev, S. Vassiliadis, High-Bandwidth Address Generation Unit, to appear in Proceedings of the Systems Architectures Modeling and Simulation (SAMOS VII) workshop, July 2007

6. N.T. Quach, B. Zafarifar, G. N. Gaydadjiev, Real-time FPGA implementation for blue-sky Detection, (to appear in) Proceedings of the IEEE International conference on Application-Specific Systems, Architectures and Processors (ASAP07), Montreal, Canada, July 2007

7. D. Vermoen, M. Witteman, G. N. Gaydadjiev, Reverse engineering Java Card applets using power analysis, Workshop in Information Security Theory and Practices 2007, pp. 138-149, Heraklion, Crete, Greece, May 2007, LNCS 4462, *Best student paper award*

8. Z. Al-Ars, S. Hamdioui, G. N. Gaydadjiev, Optimizing Test Length for Soft Faults in DRAM Devices, proceedings IEEE VLSI Test Symposium, pp. 59-66, Berkeley, CA, USA, May 2007

9. K.L.M. Bertels, G.K. Kuzmanov, E. Moscu Panainte, G. N. Gaydadjiev, Y. D. Yankova, V.M. Sima, K Sigdel, R. J. Meeuws, S. Vassiliadis, Profiling, Compilation, and HDL Generation within the hArtes Project, FPGAs and Reconfigurable Systems: Adaptive Heterogeneous Systems-on-Chip and European Dimensions (DATE 07 Workshop), pp. 53-62, Nice, France, April 2007

10. P. Ren, A.M. Amory, E. J. Marinissen, K.G.W. Goossens, S.K. Goel, G. N. Gaydadjiev, M. Lubaszewski, F. Moraes, Test wrapper design that allows a core to be tested via a network-on-chip or other functional interconnect, In Diagnostic Services in Networks-on-Chips workshop at Design, Automation and Test in Europe Conference and Exhibition (DATE), April 2007

11. Z. Al-Ars, S. Hamdioui, G. N. Gaydadjiev, Manifestation of Precharge Faults in High Speed DRAM Devices, Proc. IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, April 2007

12. F. J. Bouwens, M. Berekovic, A. Kanstein, G. N. Gaydadjiev, Architectural Exploration of the ADRES Coarse-Grained Reconfigurable Array, proceedings of Int. Workshop on Applied Reconfigurable Computing (ARC 2007), pp. 1-13, Rio de Janiero, Brazil, March 2007, LNCS 4419

13. M. Rieback, G. N. Gaydadjiev, B. Crispo, R. Hofman, A. Tanenbaum, A Platform for RFID Security and Privacy Administration, Proceedings of 20-th USENIX/SAGE Large Installation System Administration conference (LISA 2006), pp. 89-102, Washington DC, USA, December 2006, *Best paper award*

14. Z. Al-Ars, S. Hamdioui, G. N. Gaydadjiev, Using Linear Tests for Transient Faults in DRAMs, Proc. IEEE International Design and Test Workshop, November 2006

15. S. Hamdioui, Z. Al-Ars, G. N. Gaydadjiev, J.D Reyes, Comparison of Static and Dynamic Faults in 65nm Memory Technology, Proc. IEEE International Design and Test Workshop, November 2006

16. B.G.C. de Ruijsscher, G. N. Gaydadjiev, J. Lichtenauer, E. A. Hendriks, FPGA accelerator for real-time skin segmentation, Proceedings of IEEE ESTIMedia 2006. Embedded Systems for Real Time Multimedia, pp. 93-97, Seoul, Korea, October 2006, ISBN 0-7803-9783-5

17. S. Hamdioui, Z. Al-Ars, L. Mhamdi, G. N. Gaydadjiev, S. Vassiliadis, Trends in Tests and Failure Mechanisms in Deep Sub-micron Technologies,, IEEE proceesings of Int. Conference on Design and Test of Integrated Systems in Nanoscale Technology, pp. 216-221, Tunis, September 2006

18. G. N. Gaydadjiev, S. Vassiliadis, SAD Prefetching for MPEG4 Using Flux Caches, Proceedings of the 6th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2006), pp. 248-258, Samos, Greece, July 2006, LNCS 4017

19. S. Hamdioui, Z. Al-Ars, G. N. Gaydadjiev, J.D Reyes, Investigation of Single-Cell Dynamic Faults in Deep-Submicron Memory Technologies, IEEE Proc. European Test Symposium Digest of Papers, May 2006

20. S. Vassiliadis, G.K. Kuzmanov, S. Wong, E. Moscu Panainte, G. N. Gaydadjiev, K.L.M. Bertels, D. Cheresiz, PISC: Polymorphic Instruction Set Computers, Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC 2006), pp. 274-286, Delft, The Netherlands, March 2006, LNCS 3985

21. B. Donchev, G.K. Kuzmanov, G. N. Gaydadjiev, External Memory Controller for Virtex II Pro, in Proceedings of International Symposium on System-on-Chip 2006, pp. 37–40, November 2006

22. Z. Al-Ars, S. Hamdioui, A. J. van de Goor, G. N. Gaydadjiev, J Vollrath, DRAM-Specific Space of Memory Tests, Proc. IEEE International Test Conf., October 2006

23. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, The Molen Media Processor: Design and Evaluation, Proceedings of the International Workshop on Application Specific Processors, WASP 2005, pp. 26–33, New York Metropolitan Area, USA, September 2005

24. G. N. Gaydadjiev, S. Vassiliadis, Flux Caches: What Are They and Are They Useful?, Proceedings of the 5th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2005), pp. 93-102, Samos, Greece, July 2005, LNCS 3553

25. S. Vassiliadis, L. A. Sousa, G. N. Gaydadjiev, The Midlifekicker Microarchitecture Evaluation Metric, Proceedings of the IEEE International conference on Application-Specific Systems, Architectures and Processors (ASAP05), pp. 92-97, Samos, Greece, July 2005

26. Y. Dou, S. Vassiliadis, G.K. Kuzmanov, G. N. Gaydadjiev, 64-bit Floating-Point FPGA Matrix Multiplication, ACM/SIGDA Thirteenth International Symposium on Field Programmable Gate Arrays (FPGA 2005), pp. 86–95, Monterey, CA, USA, February 2005

27. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, Visual Data Rectangular Memory, Proceedings of the 10th International Euro-Par Conference (Euro-Par 2004), pp. 760–767, Pisa, Italy, September 2004, LNCS 3149

28. G. N. Gaydadjiev, S. Vassiliadis, SCISM versus IA-64 Tagging: Differences and Code Density Effects, Proceedings of 10th International Euro-Par Conference, pp. 571-577, Pisa, Italy, August 2004, Springer-Verlag Lecture Notes in Computer Science (LNCS), vol. 3149

29. S. Hamdioui, G. N. Gaydadjiev, A. J. van de Goor, The State-of-the-art and Future Trends in Testing Embedded Memories, Records IEEE International Workshop on Memory Technology, Design, and Testing, pp. 54-59, San Jose, CA, August 2004

30. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, The Virtex II Pro MOLEN Processor, Proceedings of the 4th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2004), pp. 192-202, Samos, Greece, July 2004, LNCS 3133

31. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, The MOLEN Processor Prototype, Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004), pp. 296–299, Napa, CA, USA, April 2004

32. G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, Loading rm-code: Design Considerations, Proceedings of the Third International Workshop on Computer Systems: Architectures, Modeling, and Simulation, pp. 11–19, Samos, Greece, July 2003, LNCS 3133

33. S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, E. Moscu Panainte, The Molen Programming Paradigm, Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation, pp. 1-10, Samos, Greece, July 2003, Springer-Verlag Lecture Notes in Computer Science (LNCS), vol. 3133

34. A. J. van de Goor, G. N. Gaydadjiev, V. N. Yarmolik, V.G. Mikitjuk, March LA: A Test for All Linked Memory Faults, Seventh Asian Test Symposium, pp. 1-8, Singapore, December 1998, ISBN: 0-8186-8277-9

35. A. J. van de Goor, G. N. Gaydadjiev, An Analysis of (Linked) Address Decoder Faults, IEEE MTDT'97, pp. 13-20, San Jose, California, August 1997

36. A. J. van de Goor, G. N. Gaydadjiev, V. N. Yarmolik, V.G. Mikitjuk, March LA: A Test for Linked Memory Faults, Proc. European Design & Test Conference, pp. 627-627, Paris, March 1997

37. A. J. van de Goor, G. N. Gaydadjiev, V. N. Yarmolik, V.G. Mikitjuk, March LR: A Test for Realistic Linked Faults, Proceedings of the 14th IEEE VLSI Test Symposium, pp. 272-280, New Jersey, April 1996

38. A. J. van de Goor, G. N. Gaydadjiev, Realistic Linked Memory Cell Array Faults, Proceedings of the Fifth Asian Test Symposium, pp. 183-188, Taiwan, November 1996

Invited Talks

1. S. Vassiliadis, G.N. Gaydadjiev, SARC: Systematic scalable approach to systems design: From small energy critical embedded systems to large scale networked data servers, Workshop on Directions in FPGAs and Reconfigurable Systems: Design, Programming and Technologies for adaptive heterogeneous Systems-on-Chip and their European Dimensions, DATE Conference 2007, Nice, France, April 2007

2. G.N.Gaydadjiev, Polymorphic Processors: How to Expose Arbitrary Hardware Functionality to Programmers, Guest lecture, Department of Electronic systems, Aalborg Universiteit, Denmark, April 2007

3. S. Vassiliadis, G.N. Gaydadjiev, SARC overview, Computing Architectures and Software Tools for Numerical Embedded Scalable Systems workshop and school, CASTNESS , Rome, Italy, January 2007

4. G.N. Gaydadjiev, Reconfigurable processors and programming paradigms, guest lecture, Technical University of Sofia, Bulgaria, May 2005

5. G.N.Gaydadjiev,Trusted Computing Platform, a quick look under the hood, AEGEE Symposium "What they don't tell us; privacy and security on the computer", Utrecht, Netherlands, April 2004

6. S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, Polymorphic Processors: How to Expose Arbitrary Hardware Functionality to Programmers, IEE FPGA Developer's Forum, London, United Kingdom, October 2003

Editor

1. G. N. Gaydadjiev, C. J. Glossner, J. Takala, S. Vassiliadis, 2006 International Conference on Embedded Systems: Embedded Computer Systems: Architectures, Modeling and Simulation, Proceedings of 2006 International Conference on Embedded Systems: Embedded Computer Systems: Architectures, Modeling and Simulation, pp. 203, Piscataway, NJ, July 2006, ISBN: 1-4244-0155-0, IEEE Catalog Number: 06EX1297

Short biography of the author

Georgi Gaydadjiev was born in Plovdiv, Bulgaria, in 1964. He is currently assistant professor at the Computer Engineering Laboratory, Delft University of Technology, The Netherlands. His research and development industrial experience includes more than 15 years in hardware and software design at System Engineering Ltd. in Pravetz Bulgaria and Pijnenburg Microelectronics and Software B.V. in Vught, the Netherlands. His research interests include: embedded systems design, advanced computer architectures, hardware/software co-design, VLSI design, cryptographic systems and computer systems testing. Georgi has been a member of many conference program committees at different levels, e.g. ISC, ICS, Computing Frontiers, ICCD, HiPC and more. He was program chair of SAMOS in 2006 and is a general chair in 2007. Georgi received the best paper awards at Usenix/SAGE LISA 2006 and WiSTP 2007. He is IEEE and ACM member and a member of the HiPEAC steering committee.

In this thesis we propose and classify DRAM specific fault models relevant for the state-of-the-art semiconductor technologies. We also define and validate a set of DRAM targeted march tests. In addition, we propose a methodology for deriving conditions and tests for linked memory faults. We also investigate the detection conditions for linked memory faults when one of the faults involved is an address decoder fault. Finally, we propose various optimizations for test time reduction and/or increased fault coverage.

TU Delft

ISBN: 978-90-9022223-3