NTNU – Trondheim
Norwegian University of
Science and Technology

Low-Cost MemBIST for Micro-Controllers

Hossein Atashi

Embedded Computing Systems
Submission date: June 2012
Supervisor: Einar Johan Aas, IET
Co-supervisor: Kai Kristian Amundsen, Atmel Corporation

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Low-Cost MemBIST for Micro-Controllers

by
Hossein Atashi

June, 2012

Abstract

The challenge of testing SRAM memories consists in providing realistic fault models and test solutions with minimal application time. While classical memory tests cover the static faults, they are not sufficient to cover dynamic faults which have emerged in VDSM technologies. The purpose of this thesis is implementation of a memory BIST that targets static faults as well as dynamic faults while maintaining an acceptable test time and area overhead.

At first, and as a semester project [1], the functional fault models (FFMs) associated with state-of-the-art SRAM technologies have been studied and state-of-the-art memory testing algorithms, targeting these FFMs have been presented.

Next, and as part of this master's thesis, a combination of March LR and March AB memory testing algorithms is selected and modified to support testing word-oriented memories. Furthermore, this algorithm is extended to provide support for detecting Data-Retention Faults. This algorithm is then implemented using Verilog HDL in Register-Transfer Level of abstraction.

The implemented MemBIST is then evaluated with respect to area, performance and fault coverage. A bit-oriented March LR-based MemBIST, currently in use on Atmel® AVR® micro-controllers, is used as a reference for benchmarking purposes. All target fault primitives (FPs) have been implemented using behavioral Verilog HDL and simulated with both MemBISTs.

Our evaluations show that our word-oriented MemBIST can provide a 500% performance advantage (due to the word-oriented execution) for 32-bit memories and at the same time has a better fault coverage compared to the reference MemBIST. The implemented algorithm can detect all static and realistic dynamic inter-word memory faults as well as most static and realistic dynamic intra-word faults. The implemented MemBIST also maintains a very small area overhead due to sharing the required registers with existing system components.

Keywords: MemBIST, Built-In Self Test, Memory Testing, March Test, Fault Model, Fault Coverage, Fault Detection

Problem Description

Micro-controllers are produced in very high volumes, and are sensitive to high test cost (long test time) as well has high Design-For-Test cost (increased silicon area). This conflicting requirement poses a particular challenge for production testing of RAMs. Each micro-controller typically contains multiple RAM blocks, which require a memory built-in self test (MemBIST) module to reduce test cost. This can only be justified if the added cost of the MemBIST module itself is very low. At the same time, test coverage should be as high as possible, both for traditional stuck-at faults, as well as memory-specific faults such as retention faults.

The main assignment in this thesis was to:

1. Describe different fault mechanisms in embedded RAMs.
2. Find a BIST algorithm that covers these mechanisms for RAMs of arbitrary bit size.
3. Find a way to support retention testing of RAMs using the above algorithm.
4. Suggest how this algorithm might be implemented in an actual 32-bit micro-controller.

Note: This research was carried out in two separate steps. At first, and as part of a semester project, a comprehensive literature review was performed, the results of which are presented in chapters 2 and 3. Afterwards, and as part of a master's thesis, the actual implementation and evaluation of the MemBIST was done, the results of which are presented in chapters 4 and 5. For the sake of completeness and readability, the results of both steps are joined in this report. The results of the semester project were also separately published as [1].

Preface

The work presented in this thesis contains the results of my research as a student in European Master in Embedded Computing Systems (EMECS) program at the Department of Electronics and Telecommunications at Norwegian University of Science and Technology (NTNU), and at Atmel Norway AS in Trondheim, Norway.

This thesis would not have been possible without the contribution of numerous people in several ways. I would like to take this opportunity to thank them.

First, I would like to thank Prof. Einar Johan Aas for giving me the opportunity to work on this research. His guidance in every step of the project, from the start to writing the thesis report was of great help to me without which, this research would not have been possible.

Furthermore, I would like to thank the involved people at Atmel Norway AS at Trondheim, Norway, especially Kai Kristian Amundsen, and Tor Erik Leistad who also helped me in every step of the project, from problem definition to writing the report, with their valuable feedbacks, technical guidance, and providing the required equipment and tools for implementation and evaluation of the project.

Last, but not least, I want to thank my family and friends who always stood by me and provided me with the moral support. In particular, I want to thank my father who never stopped believing in me, and whom I can never thank enough for his support.

Hossein Atashi

Trondheim, June, 2012
Norway

Contents

1 Introduction **9**

2 Literature Review **11**

2.1 Memory Fault Models . . . 11
2.1.1 Concept of Fault Primitive . . . 11
2.1.2 Classification of Fault Primitives . . . 13
2.1.2.1 Simple versus Linked Faults . . . 14
2.1.2.2 Static versus Dynamic Faults . . . 15
2.1.2.3 Single-Port versus Multi-Port Faults . . . 15
2.1.2.4 Single-Cell versus Multi-Cell Faults . . . 16
2.1.3 Single-Port Static Faults . . . 16
2.1.3.1 Single-Cell Fault Primitives . . . 16
2.1.3.2 Single-Cell Functional Fault Models . . . 17
2.1.3.3 Two-Cell Fault Primitives . . . 20
2.1.3.4 Two-Cell Functional Fault Models . . . 20
2.1.4 Single-Port Dynamic Faults . . . 24
2.1.4.1 Single-Cell Fault Primitives . . . 25
2.1.4.2 Single-Cell Functional Fault Models . . . 25
2.1.4.3 Two-Cell Fault Primitives . . . 27
2.1.4.4 Two-Cell Functional Fault Models . . . 27
2.2 Memory Test Algorithms . . . 28
2.2.1 March Test Algorithms . . . 29
2.2.2 March LR: A Test for "Realistic" Linked Faults . . . 29
2.2.3 March SS: A Test for All Static Simple RAM Faults . . . 31
2.2.4 March 13N and March 9N . . . 32
2.2.5 March AB: A State-of-the-Art March Test for Realistic Static Linked Faults and Dynamic Faults . . . 33

2.2.5.1 Dynamic Faults 34
2.2.5.2 Static Linked Faults 34
2.2.5.3 March AB Test Algorithm 36
2.2.6 March RAW: Testing Static and Dynamic Faults in Random Access Memories 36
2.2.7 Effect of Address Ordering 37
2.2.8 Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories 37
2.2.8.1 Detection of Single-Cell and Inter-Word Faults . 38
2.2.8.2 Detection of Intra-Word Faults 39
2.3 Conclusion 47

3 Comparison of MemBIST Algorithms 49
3.1 Effectiveness of Memory Test Algorithms and Analysis of Fault Distribution in SRAMs: A Case Study Based on Industrial Test Results 50
3.2 Testing Static and Dynamic Faults in Random Access Memories 51
3.3 March AB, A State-of-the-Art March Test for Realistic Static Linked Faults and Dynamic Faults in SRAMs 52
3.4 Conclusion 53

4 Implementation of the MemBIST 55
4.1 Proposed Architecture 55
4.2 Proposed MemBIST Algorithm 58
4.2.1 Selected BOM Test 58
4.2.2 Extension to WOM Test 60
4.2.3 Theoretical Performance Analysis 61
4.2.4 Extension to Retention Testing 62
4.3 Implementation of the MemBIST Unit 63
4.3.1 Implementation of March AB+LR 63
4.3.2 Implementation of Data Retention Testing 64
4.4 Conclusion 65

5 Evaluation and Experimental Results 66
5.1 Performance Evaluation 67
5.1.1 Discussion 68
5.2 Area Evaluation 69
5.2.1 Discussion 69
5.3 Fault Coverage Evaluation 70

5.3.1 Inter-word Faults 71
5.3.1.1 Discussion 71
5.3.2 Intra-word Faults 73
5.3.2.1 Discussion 73
5.3.3 Interpretation of the Results and Translation to Physical Defect Coverage 79
5.4 Summary 80

6 Summary and Conclusions **81**

Bibliography **82**

List of Figures

2.1 Classification of Fault Primitives 13
2.2 General Form of Linked Coupling Faults 14

3.1 Results of the Comparison Performed in [2] 50

4.1 A Block Diagram of Relevant Interconnection Structures in a Typical Atmel® AVR32® Micro-Controller 56

List of Tables

2.1 The Complete Set of 1PF1 Fault Primitives 18
2.2 List of 1PF1 FFMs; $x \in \{0, 1\}$ 18
2.3 The Complete Set of 1PF2 FPs; $x \in \{0, 1\}$ 21
2.4 List of 1PF2 FFMs; $x, y \in \{0, 1\}$ 22
2.5 The Complete Set of Single-Cell 2-Operation Dynamic Fault Primitives 26
2.6 List of Single-Cell Two-Operation Dynamic Functional Fault Models 26
2.7 List of Two-Cell Two-Operation Dynamic Functional Fault Models 27
2.8 Realistic Static Linked Faults Targeted by March AB 35
2.9 DBs for uCFsts (B=8) 40
2.10 DBS S_8 for uCFids (B=8) 41
2.11 DBOS for CFdsts (B=4) 42
2.12 Set of 8-bit DBs for rCFsts 45
2.13 Set of 8-Bit DBs for crCFsts 46
2.14 8-Bit DBS for crCFids 46
2.15 DBOS for crCFdsts, B=4 47
2.16 Number of DBs and TLs for Different CF Types 48

3.1 Result of the Fault Coverage Estimation Performed in [3] 51
3.2 Comparison of March Tests with Respect to Dynamic Faults as Presented in [4] 52
3.3 Comparison of March Tests with Respect to Linked Faults as Presented in [4] 53

5.1 Run-time of MemBIST algorithms on an $8 \times 32bit$ reference memory 68
5.2 Area Report for Synthesis of Memory Service Unit 69

5.3 Fault Simulation Results for Inter-Word Static Single-Cell Faults 71
5.4 Fault Simulation Results for Intra-Word Static Coupling Faults . 72
5.5 Fault Simulation Results for Inter-Word Dynamic Single-Cell Faults 73
5.6 Fault Simulation Results for Inter-Word Dynamic Coupling Faults 74
5.7 Fault Coverage for Inter-Word Faults 75
5.8 Fault Simulation Results for Intra-Word Static Faults 76
5.9 Fault Simulation Results for Intra-Word Dynamic Faults 77
5.10 Fault Coverage for Targeted Intra-Word Faults 78
5.11 Fault Coverage for Untargeted Intra-Word Faults 78

Chapter 1

Introduction

Memories are designed to exploit the technology limits to reach the highest storage density and high speed access. The main consequence is that memory devices are statistically more likely to be affected by manufacturing defects. The challenge of testing SRAM memories consists in providing realistic fault models and test solutions with minimal application time. Due to the complexity of the memory device, fault modeling is not trivial. Classical memory test solutions cover the so-called *static faults*, such as stuck-at, transition, and coupling faults, but are not sufficient to cover faults that have emerged in latest VDSM[1] technologies, referred to as *dynamic faults* [5].

The goal of this thesis is to implement an efficient MemBIST[2] for production testing of embedded SRAMs on a micro-controller. In particular we are interested in selecting a suitable test algorithm for the SRAMs embedded on Atmel® AVR® family micro-controllers.

As the users are generally not willing to pay the price for testing costs while expecting the final product to be defect-free, the desired algorithm should satisfy the following criteria:

Performance: The ATE[3] devices used for testing semiconductor devices are typically very expensive. Therefore the required time for testing a device has a direct impact on the production costs. A good MemBIST should be able to test the device in a reasonable amount of time, typically in the

[1] Very Deep Sub-Micron
[2] Memory Built-In Self Test
[3] Automatic Test Equipment

order of seconds.

Area: Since MemBIST is a built-in unit, it will be implemented in every single device. Therefore, the silicon area of each device will be increased by the MemBIST which will result in increased production costs. A good MemBIST algorithm should impose a relatively low area overhead.

Fault coverage: The main objective in implementation of a MemBIST is to detect as many faulty units as possible before shipping to the field. This requires a high fault coverage to be provided by the algorithm.

In the rest of this research, an efficient MemBIST algorithm is chosen, implemented and evaluated with respect to the above criteria.

This report is organized as follows: at first, and in chapter 2, a comprehensive literature review will be performed, and state-of-the-art memory fault models and memory test algorithms will be discussed. Next, in chapter 3, we will compare the studied memory test algorithms based on the fault coverage results published in the literature. In chapter 4, an appropriate algorithm will be selected and implemented for the target devices. In chapter 5, the designed MemBIST will be evaluated with respect to the above criteria and compared to a reference MemBIST currently in use in Atmel AVR micro-controllers. Finally, in chapter 6, a summary of the research will be given and conclusions will be made.

Chapter 2

Literature Review

In this chapter an overview of previous research on memory test will be given.

In section 2.1, different memory fault models will be studied and in section 2.2, existing memory test algorithms will be reviewed.

2.1 Memory Fault Models

In this section, different memory fault models will be introduced and categorized. First the concept of *fault primitive* will be defined based on which the memory faults are categorized. Afterwards different faults will be classified based on these fault primitives. Most of the fault models and classifications in this section are based on the study done in [6].

In general, the functional model of a memory depends on its specific implementation. However for testing purposes, a so-called *reduced functional memory model* is used that only consists of three subsystems: the address decoder, the memory cell array and the read/write logic. Since the vast majority of mainstream memory devices contain these three subsystems, the reduced functional fault model is, to a large extent, independent of specific memory implementations [6].

2.1.1 Concept of Fault Primitive

Intuitively, a functional fault model is defined as a description of the failure of the memory to fulfill its functional specifications. This definition of a fault is

not precise since it does not indicate which functional specifications should be taken into account.

By performing a number of memory operations and observing the behavior of any component functionally modeled in the memory, functional faults can be defined as the deviation of the observed behavior from the specified one under the performed operation(s). Therefore the two basic ingredients to any fault model are:

1. A list of performed memory operations.
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 (SOS)*. The observed memory behavior that deviates from the expected one is called a *faulty behavior* [6].

In order to specify a certain fault, a SOS together with the corresponding faulty behavior should be specified. This combination for a single fault behavior is called a *fault primitive (FP)* [7], and is denoted as $< S/F/R >$. S describes the SOS that sensitizes the fault. F describes the value or the behavior of the faulty cell (e.g., the cell flips from 0 to 1), while R describes the logic output level of a read operation (e.g., 0) [6].

The concept of a FP allows for establishing a complete framework of all memory faults, since for all allowed operational sequences in the memory, one can derive all possible faulty behaviors. In addition, the concept of a FP makes it possible to give a precise definition of *functional fault model (FFM)* as it has to be understood for memory devices [7, 6]: A *functional fault model* is a non-empty set of fault primitives.

Since a fault model is defined as a set of FPs, it is expected that FFMs would inherit the properties of FPs. For example, if a FFM is defined as a collection of single cell FPs, then the FFM is a single cell fault. If a FFM is defined as a collection of 2-operation (i.e., the SOS consists of two sequential operations) FPs, then the FFM is also called a 2-operation fault. If a FFM consists of FPs classified into inconsistent classes (e.g., single cell and two-cell FPs) the FFM is described by the classes of its consistent FPs. Therefore a FFM that consists of single cell and two-cell FPs, for example, is described as a single and two-cell FFM [6].

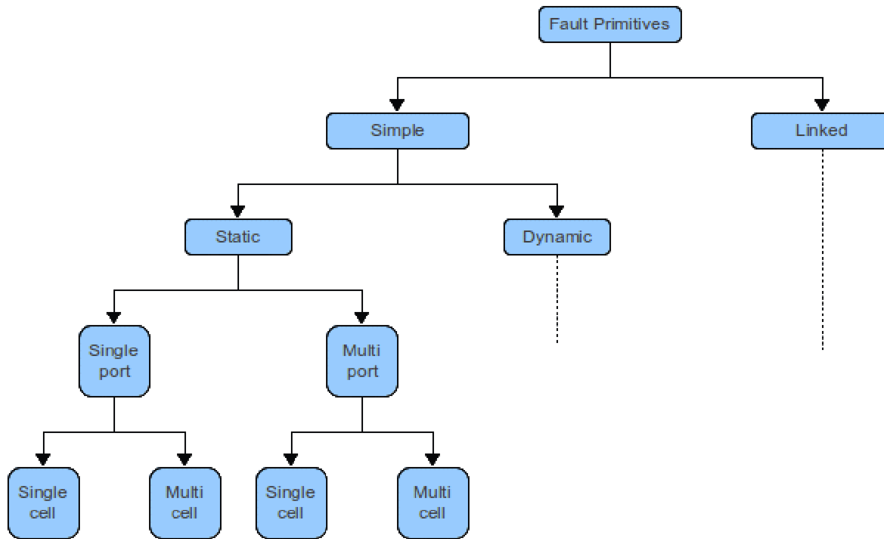


Figure 2.1: Classification of Fault Primitives

2.1.2 Classification of Fault Primitives

In this section, a classification of different fault primitives is given. This classification is based on the classification done by Hamdioui in [6].

Figure 2.1 shows the different classifications of the FPs. They can be classified based on:

1. The way the FPs manifest themselves, into *simple* and *linked* faults
2. The number of *sequential* operations required in the SOS, into *static* and *dynamic* faults
3. The number of *simultaneous* operations required in the SOS, into *single-port* and *multi-port* faults
4. The number of different cells the FPs involve, into *single-cell* and *multi-cell* faults

Note that the four ways of classifying fault primitives are independent since their definition is based on independent factors of the SOS. Therefore a dynamic fault

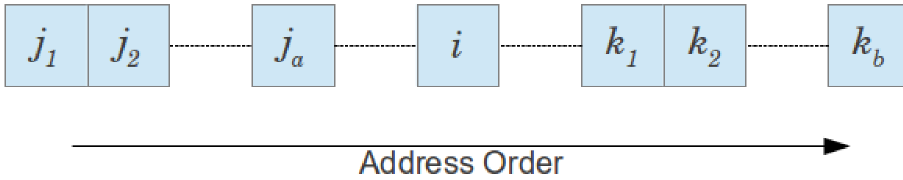


Figure 2.2: General Form of Linked Coupling Faults

primitive can be single-port or multi-port, single-cell or multi-cell. Similarly a linked fault can be static or dynamic, and each of them can be single-port or multi-port, single-cell or multi-cell [6].

In the rest of this section, this classification is described in more detail.

2.1.2.1 Simple versus Linked Faults

Depending on the way FPs manifest themselves, they can be categorized into *simple faults* and *linked faults*.

Simple faults are those faults which cannot influence the behavior of each other. In other words, the behavior of a simple fault does not influence the behavior of another one; therefore *masking* cannot occur.

Linked faults are those faults that do influence the behavior of each other. This means that the behavior of a certain fault can change the behavior of another one such that *masking* can occur [8, 9]. Linked faults consist of two or more simple faults. As an example, assume that performing an operation to a cell c_1 will cause a fault in a cell c_v (i.e., the cell flips). Also assume that application of an operation to a cell c_2 will cause a fault in the same cell c_v, but with a fault effect opposite to the fault caused by c_1. Now if an operation is applied to cell c_1, followed by an operation on c_2, the fault effect of c_1 is masked by the fault effect of c_2. Therefore no fault effect is visible in c_v [6].

An example for linked coupling faults is shown in figure 2.2. Cell i is coupled to a cells j all with addresses lower than i and b cells k all with addresses higher than i. Masking can for example occur in case of the following two coupling faults: $<\uparrow;\uparrow>_{j_1,i}$ and $<\uparrow;\downarrow>_{j_2,i}$; because the first CF causes the cell to be set which is thereafter reset by the second CF [10].

2.1.2.2 Static versus Dynamic Faults

Fault primitives can be classified based on the number of operations required to sensitize the corresponding faults. This criterion will classify the FPs into *static* and *dynamic* faults [6].

Static faults are those FPs which can be sensitized by performing *at most* one operation. For example if the state of a cell is stuck at one, no operation is needed to sensitize the fault. As another example, if a read operation on a cell causes that cell to flip, one operation will be required to sensitize this fault. Therefore, these two faults are classified as static faults.

Dynamic faults are those FPs that can only be sensitized by at least two *sequential* operations. Depending on the number of sequential operations required to sensitize these faults, these FPs can be further categorized to *2-operation dynamic FPs, 3-operation dynamic FPs,* etc.

2.1.2.3 Single-Port versus Multi-Port Faults

Fault primitives can be classified based on the number of ports required *simultaneously* to apply a SOS. This criterion will classify the FPs into *single-port* and *multi-port* faults [6].

Single-port faults are those faults that require at most one port to be sensitized. For example if a single read operation on a memory cell causes that cell to flip, this fault will be classified as a single-port fault. Note that single-port faults can be sensitized in single-port memories as well as in multi-port memories.

Multi-port faults are those faults that can only be sensitized by multiple simultaneous operations through different ports. For example, if two *simultaneous* read operations cause a memory cell to flip, this will be classified as a *multi-port* fault. Based on the number of ports required to sensitize a fault, this class can be further categorized into *two-port faults, three-port faults,* etc.

Since the primary objective of this project is testing single-port memories, we have focused on single-port faults in the rest of this report.

2.1.2.4 Single-Cell versus Multi-Cell Faults

Based on the number of cells accessed during a SOS, FPs can be classified to *single-cell* and *multi-cell* faults. Multi-cells faults are also known as *coupling faults* [6].

Single-cell faults are the FPs which only involve a single cell. In these FPs, the cell used for sensitizing the faults is the same as the cell in which the fault appears (i.e., the victim cell). For example, if reading a cell c_1 causes the same cell c_1 to flip, the corresponding FP will be a single-cell fault primitive.

Coupling faults are the FPs which involve more than one cell. These FPs have the property that the cell(s) which sensitizes (or contributes for sensitizing) the fault is different from the cell where the fault appears. For example, if reading a cell c_1 causes a different cell c_2 to flip, the corresponding FP will be a *coupling fault.* Based on the number of cells involved in a FP, these faults can be further classified into *two-coupling fault primitives, 3-coupling fault primitives,* etc.

2.1.3 Single-Port Static Faults

Since the main focus of this project is on single-port memories, the faults associated to this type of memories (i.e., single-port faults), are described in more detail. The descriptions in this section are mostly based on the overview made in [6, 5].

Single-port faults can occur in both single-port and multi-port memories. These faults can in turn be simple or linked, static or dynamic and single-cell or multi-cell. In the literature, among multi-cell faults, mostly 2-cell coupling faults have been considered. Since these faults are an important class in SRAM faults [6], we have also focused only on this type of coupling faults in this project.

2.1.3.1 Single-Cell Fault Primitives

A single port, single cell fault primitive (*1PF1*) involving a single cell $c_v$is denoted by $< S/F/R >$ (or $< S/F/R >_v$) . In this type of fault, the cell used to sensitize the fault is the same as the cell in which the fault appears.

In this notation, S describes the sensitizing value or operation; $S \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$ whereby 0 (1) denotes a zero (one) value, $0w0$ ($1w1$) denotes a write 0 (1) operation to a cell which contains a 0 (1) , $0w1$ ($1w0$) denotes

an up (down) transition write operation, and $r0$ ($r1$) denotes a read 0 (1) operation. If the fault effect of S appears after time T, then the sensitizing operation is given as S_T [6].

F describes the value of the *faulty cell* (victim cell); $F \in \{0, 1, \uparrow, \downarrow, ?\}$, whereby $\uparrow(\downarrow)$ denotes an up (down) transition due to a certain sensitizing operation, and ? denotes an *undefined* state of the cell (e.g., the voltage of the true and the false node of the cell are almost the same).

R describes the logical value which appears at the output of the SRAM if the sensitizing operation applied to the v-cell is a read operation: $R \in \{0, 1, ?, -\}$, whereby ? denotes a *random* value. A random logic value can occur if the voltage difference between the bit lines is very small. A '-' in R means that the output data is not applicable; for example if $S = w0$, then no data will appear at the memory output, and for that reason R is replaced by a '-'. It is worth noting here that the word *undefined* is used for the state of the cell ($F =?$) and *random* is used for the read data value ($R =?$).

Using these possible values for S, F, and R, it is possible to list all 1PF1s. This list is illustrated in table 2.3. Other possible combinations of S, F, and R do not represent a faulty behavior. For example, $< 1w0/0/- >$ demonstrates a correct $w0$ operation in the memory. The 'FFM' column describes the functional fault model corresponding to each fault primitives. These FFMs will be discussed in detail in section 2.1.4.2.

One can see that in total there are 28 single-cells FPs, 8 of which are sensitized by write operations and 16 by read operations.

2.1.3.2 Single-Cell Functional Fault Models

The list of all possible single-cell FPs can be compiled to a set of FFMs. Assigning each FP to a FFM is rather arbitrary and is mainly determined by historical arguments. Again, this classification is based on the classification done in [6]. The list of FFMs and their corresponding FPs is shown in table 2.2.

Each of these FFMs is described in the following.

1. State Fault (SF): A cell has a *state fault* if its logic value flips before it is accessed, even if no operation is performed on it[1]. This fault does not need any operation for sensitization and therefore only depends on the initial stored value in the cell.

[1] It should be noted that here the cell should flip in short time period after initialization and before accessing the cell

Table 2.1: The Complete Set of 1PF1 Fault Primitives

#	S	F	R	$< S/F/R >$	FFM	#	S	F	R	$< S/F/R >$	FFM
1	0	1	$-$	$< 0/1/- >$	SF	2	0	?	$-$	$< 0/?/- >$	USF
3	1	0	$-$	$< 1/0/- >$	SF	4	1	?	$-$	$< 1/?/- >$	USF
5	$0w0$	$\uparrow$	$-$	$< 0w0/\uparrow/- >$	WDF	6	$0w0$	?	$-$	$< 0w0/?/- >$	UWF
7	$1w1$	$\downarrow$	$-$	$< 1w1/\downarrow/- >$	WDF	8	$1w1$	?	$-$	$< 1w1/?/- >$	UWF
9	$0w1$	0	$-$	$< 0w1/0/- >$	TF	10	$0w1$	?	$-$	$< 0w1/?/- >$	UWF
11	$1w0$	1	$-$	$< 1w0/1/- >$	TF	12	$1w0$	?	$-$	$< 1w0/?/- >$	UWF
13	$r0$	0	1	$< r0/0/1 >$	IRF	14	$r0$	0	?	$< r0/0/? >$	RRF
15	$r0$	$\uparrow$	0	$< r0/\uparrow/0 >$	DRDF	16	$r0$	$\uparrow$	1	$< r0/\uparrow/1 >$	RDF
17	$r0$	$\uparrow$	?	$< r0/\uparrow/? >$	RRDF	18	$r0$	?	0	$< r0/?/0 >$	URF
19	$r0$	?	1	$< r0/?/1 >$	URF	20	$r0$	?	?	$< r0/?/? >$	URF
21	$r1$	1	0	$< r1/1/0 >$	IRF	22	$r1$	1	?	$< r1/1/? >$	RRF
23	$r1$	$\downarrow$	0	$< r1/\downarrow/0 >$	RDF	24	$r1$	$\downarrow$	1	$< r1/\downarrow/1 >$	DRDF
25	$r1$	$\downarrow$	?	$< r1/\downarrow/? >$	RRDF	26	$r1$	?	0	$< r1/?/0 >$	URF
27	$r1$	?	1	$< r1/?/1 >$	URF	28	$r1$	?	?	$< r1/?/? >$	URF

Table 2.2: List of 1PF1 FFMs; $x \in \{0, 1\}$

#	FFM	Fault Primitives
1	SF	$< 1/0/- >$, $< 0/1/- >$
2	TF	$< 0w1/0/- >$, $< 1w0/1/- >$
3	WDF	$< 0w0/\uparrow/- >$, $< 1w1/\downarrow/- >$
4	RDF	$< r0/\uparrow/1 >$, $< r1/\downarrow/0 >$
5	DRDF	$< r0/\uparrow/0 >$, $< r1/\downarrow/1 >$
6	RRDF	$< r0/\uparrow/? >$, $< r1/\downarrow/? >$
7	IRF	$< r0/0/1 >$, $< r1/1/0 >$
8	RRF	$< r0/0/? >$, $< r1/1/? >$
9	USF	$< 1/?/- >$, $< 0/?/- >$
10	UWF	$< 0w0/?/- >$, $< 0w1/?/- >$, $< 1w0/?/- >$, $< 1w1/?/- >$
11	URF	$< rx/?/0 >$, $< rx/?/1 >$, $< rx/?/? >$
12	SAF	$< \forall/0/- >$,$< \forall/1/- >$
13	NAF	$\{< 0w1/0/- >, < 1w0/1/- >, < rx/x/? >\}$
14	DRF	$< 1_T/\downarrow/- >$, $< 0_T/\uparrow/- >$, $< x_T/?/- >$

2. Transition Fault (TF): A cell has a *transition fault* if it fails to undergo a transition in a write operation. This FFM depends both on the initial stored value and the type of the operation.

3. Write Destructive Fault (WDF): A cell suffers from a *write destructive fault* if a non-transition write operation causes a transition in the cell.

4. Read Destructive Fault (RDF): A cell is said to have a *read destructive fault* if a read operation performed on the cell changes the data in the cell, and returns an *incorrect* value on the output.

5. Deceptive Read Destructive Fault (DRDF): A cell suffers from a *deceptive read destructive fault* if a read operation performed on the cell changes the logic value of the cell, while returning the correct value as the output.

6. Random Read Destructive Fault (RRDF): A cell has a *random read destructive fault* if a read operation performed on the cell, changes the logic value of the cell and returns a random value.

7. Incorrect Read Fault (IRF): A cell is said to have an *incorrect read fault* if a read operation performed on the cell returns the incorrect logic value while keeping the correct value in the cell.

8. Random Read Fault (RRF): A cell suffers from a *random read fault* if a read operation returns a random value while keeping the correct value in the cell.

9. Undefined State Fault (USF): A cell has an *undefined state fault* if the logic value of the cell flips to an undefined state before the cell is accessed, even if no operation is performed on it[2]. As in SF, this fault does not need any operation for sensitization and therefore only depends on the initial stored value in the cell.

10. Undefined Write Fault (UWF): A cell is said to have an *undefined write fault* if the cell is brought in an undefined state by a write operation.

11. Undefined Read Fault (URF): A cell is said to have an *undefined read fault* if the cell is brought in an undefined state by a read operation. The returned data value during this operation can be correct, incorrect, or random.

[2]It should be noted that here the cell should flip in short time period after initialization and before accessing the cell

12. Stuck-At Fault (SAF): A cell is said to have a *stuck-at fault* if it remains always stuck at a given value for all performed operations.

13. No Access Fault (NAF): A cell suffers from a *no access fault* if the cell is not accessible. In this case the state of the cell cannot be changed by write operations, and any read operation applied to the cell returns a random data value. The NAF consists of four FPs which occur *simultaneously*: $\{< 0w1/0/- >, < 1w0/1/- >, < r0/0/? >, < r1/1/? >\}$. Note that NAF is a more general form of the *Stuck-Open Fault* which is defined as an inaccessible cell due to an open word line.

14. Data Retention Fault (DRF): A cell is said to have a *data retention fault* if the state of the cell changes after a certain time T, and without accessing the cell. T here should be longer than the duration of the pre-charge cycle in SRAMs, because if the cell flips within the pre-charge cycle then the sensitized fault would be a state fault.

Note that the first 11 FFMs defined above, contain 28 possible FPs of table 2.2. The SAF and NAF are FFMs which require more than one FP to be present due to the same single defect; therefore they are called *composite* FFMs.

2.1.3.3 Two-Cell Fault Primitives

A two-cell FP is denoted by $< S_a; S_v/F/R >$ (or $< S_a; S_v/F/R >_{a,v}$). In this type of fault, the sensitizing cell is different from the cell in which the fault occurs.

In this notation, S_a describes the sensitizing operation or state of the aggressor cell (a-cell); while S_v describes the sensitizing operation or state of the victim cell (v-cell). Here, the set S_i is defined as: $S_i \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$ $(i \in \{a, v\})$, $F \in \{0, 1, \uparrow, \downarrow, ?\}$, and $R \in \{0, 1, ?, -\}$.

All possible combinations of the values for $< S_a; S_v/F/R >$ notation are listed in table 2.3. The column 'FFM' shows the functional fault model associated with each FP. These FFMs will be discussed in more detail in section 2.1.3.4.

The completeness of table 2.3 is proven in [6].

2.1.3.4 Two-Cell Functional Fault Models

The list of all possible two-cell FPs can be compiled to a set of FFMs. Assigning each FP to a FFM is rather arbitrary and is mainly determined by historical

Table 2.3: The Complete Set of 1PF2 FPs; $x \in \{0, 1\}$

#	S_a	S_v	F	R	$< S_a; S_v/F/R >$	FFM	#	S_a	S_v	F	R	$< S_a; S_v/F/R >$	FFM
1	x	0	1	–	$< x; 0/1/- >$	CFst	2	x	0	?	–	$< x; 0/?/- >$	CFus
3	x	1	0	–	$< x; 1/0/- >$	CFst	4	x	1	?	–	$< x; 1/?/- >$	CFus
5	x	$0w0$	$\uparrow$	–	$< x; 0w0/\uparrow/- >$	CFwd	6	x	$0w0$	?	–	$< x; 0w0/?/- >$	CFuw
7	x	$1w1$	$\downarrow$	–	$< x; 1w1/\downarrow/- >$	CFwd	8	x	$1w1$	?	–	$< x; 1w1/?/- >$	CFuw
9	x	$0w1$	0	–	$< x; 0w1/0/- >$	CFtr	10	x	$0w1$	?	–	$< x; 0w1/?/- >$	CFuw
11	x	$1w0$	1	–	$< x; 1w0/1/- >$	CFtr	12	x	$1w0$	?	–	$< x; 1w0/?/- >$	CFuw
13	x	$r0$	0	1	$< x; r0/0/1 >$	CFir	14	x	$r0$	0	?	$< x; r0/0/? >$	CFrr
15	x	$r0$	$\uparrow$	0	$< x; r0/\uparrow/0 >$	CFdrd	16	x	$r0$	$\uparrow$	1	$< x; r0/\uparrow/1 >$	CFrd
17	x	$r0$	$\uparrow$	?	$< x; r0/\uparrow/? >$	CFrrd	18	x	$r0$	?	0	$< x; r0/?/0 >$	CFur
19	x	$r0$	?	1	$< x; r0/?/1 >$	CFur	20	x	$r0$	?	?	$< x; r0/?/? >$	CFur
21	x	$r1$	1	0	$< x; r1/1/0 >$	CFir	22	x	$r1$	1	?	$< x; r1/1/? >$	CFrr
23	x	$r1$	$\downarrow$	0	$< x; r1/\downarrow/0 >$	CFrd	24	x	$r1$	$\downarrow$	1	$< x; r1/\downarrow/1 >$	CFdrd
25	x	$r1$	$\downarrow$	?	$< x; r1/\downarrow/? >$	CFrrd	26	x	$r1$	?	0	$< x; r1/?/0 >$	CFur
27	x	$r1$	?	1	$< x; r1/?/1 >$	CFur	28	x	$r1$	?	?	$< x; r1/?/? >$	CFur
29	$0w0$	0	$\uparrow$	–	$< 0w0; 0/\uparrow/- >$	CFds	30	$0w0$	0	?	–	$< 0w0; 0/?/- >$	CFud
31	$1w1$	0	$\uparrow$	–	$< 1w1; 0/\uparrow/- >$	CFds	32	$1w1$	0	?	–	$< 1w1; 0/?/- >$	CFud
33	$0w1$	0	$\uparrow$	–	$< 0w1; 0/\uparrow/- >$	CFds	34	$0w1$	0	?	–	$< 0w1; 0/?/- >$	CFud
35	$1w0$	0	$\uparrow$	–	$< 1w0; 0/\uparrow/- >$	CFds	36	$1w0$	0	?	–	$< 1w0; 0/?/- >$	CFud
37	$r0$	0	$\uparrow$	–	$< r0; 0/\uparrow/- >$	CFds	38	$r0$	0	?	–	$< r0; 0/?/- >$	CFud
39	$r1$	0	$\uparrow$	–	$< r1; 0/\uparrow/- >$	CFds	40	$r1$	0	?	–	$< r1; 0/?/- >$	CFud
41	$0w0$	1	$\downarrow$	–	$< 0w0; 1/\downarrow/- >$	CFds	42	$0w0$	1	?	–	$< 0w0; 1/?/- >$	CFud
43	$1w1$	1	$\downarrow$	–	$< 1w1; 1/\downarrow/- >$	CFds	44	$1w1$	1	?	–	$< 1w1; 1/?/- >$	CFud
45	$0w1$	1	$\downarrow$	–	$< 0w1; 1/\downarrow/- >$	CFds	46	$0w1$	1	?	–	$< 0w1; 1/?/- >$	CFud
47	$1w0$	1	$\downarrow$	–	$< 1w0; 1/\downarrow/- >$	CFds	48	$1w0$	1	?	–	$< 1w0; 1/?/- >$	CFud
49	$r0$	1	$\downarrow$	–	$< r0; 1/\downarrow/- >$	CFds	50	$r0$	1	?	–	$< r0; 1/?/- >$	CFud
51	$r1$	1	$\downarrow$	–	$< r1; 1/\downarrow/- >$	CFds	52	$r1$	1	?	–	$< r1; 1/?/- >$	CFud

Table 2.4: List of 1PF2 FFMs; $x, y \in \{0, 1\}$

#	FFM	Fault Primitives
		$1PF2_s$
1	CFst	$< 0; 0/1/- >, < 0; 1/0/- >, < 1; 0/1/- >, < 1; 1/0/- >$
2	CFus	$< 0; 0/?/- >, < 0; 1/?/- >, < 1; 0/?/- >, < 1; 1/?/- >$
		$1PF2_a$
3	CFds	$< xwy; 0/\uparrow/- >, < xwy; 1/\downarrow/- >, < rx; 0/\uparrow/- >, < rx; 1/\downarrow/- >$
4	CFud	$< xwy; 0/?/- >, < xwy; 1/?/- >, < rx; 0/?/- >, < rx; 1/?/- >$
5	CFid	$< 0w1; 0/\uparrow/- >, < 0w1; 1/\downarrow/- >, < 1w0; 0/\uparrow/- >, < 1w0; 1/\downarrow/- >$
6	CFin	$\{< 0w1; 0/\uparrow/- >, < 0w1; 1/\downarrow/- >\}, \{< 1w0; 0/\uparrow/- >, < 1w0; 1/\downarrow/- >\}$
		$1PF2_v$
7	CFtr	$< 0; 0w1/0/- >, < 1; 0w1/0/- >, < 0; 1w0/1/- >, < 1; 1w0/1/- >$
8	CFwd	$< 0; 0w0/\uparrow/- >, < 1; 0w0/\uparrow/- >, < 0; 1w1/\downarrow/- >, < 1; 1w1/\downarrow/- >$
9	CFrd	$< 0; r0/\uparrow/1 >, < 1; r0/\uparrow/1 >, < 0; r1/\downarrow/0 >, < 1; r1/\downarrow/0 >$
10	CFdrd	$< 0; r0/\uparrow/0 >, < 1; r0/\uparrow/0 >, < 0; r1/\downarrow/1 >, < 1; r1/\downarrow/1 >$
11	CFrrd	$< 0; r0/\uparrow/? >, < 1; r0/\uparrow/? >, < 0; r1/\downarrow/? >, < 1; r1/\downarrow/? >$
12	CFir	$< 0; r0/0/1 >, < 1; r0/0/1 >, < 0; r1/1/0 >, < 1; r1/1/0 >$
13	CFrr	$< 0; r0/0/? >, < 1; r0/0/? >, < 0; r1/1/? >, < 1; r1/1/? >$
14	CFuw	$< x; 0w0/?/- >, < x; 0w1/0/- >, < x; 1w0/?/- >, < x; 1w1/?/- >$
15	CFur	$< x; r0/?/0 >, < x; r0/?/1 >, < x; r0/?/? >, < x; r1/?/0 >, < x; r1/?/1 >, < x; r1/?/? >$

arguments. This classification is based on the classification done in [6]. The list of FFMs and their corresponding FPs is shown in table 2.4. Here the FFMs are divided into three types: $1PF2_s$, $1PF2_a$ and $1PF2_v$.

$1PF2_s$ FFMs In this type of fault, the *state* of the a-cell sensitizes a fault in the v-cell. This type of fault consists of two FFMs:

1. State Coupling Fault (CFst): Two cells are said to have a *state coupling fault* if the v-cell is forced into a given logic state only if the a-cell is in a given state, without performing any operation on the v-cell or the a-cell. This fault is special in the sense that no operation is needed to sensitize it and it only depends on the initial stored values in the cells.

2. Undefined State Coupling Fault (CFus): Two cells are said to have an *undefined state coupling fault* if the v-cell is forced into an undefined logic

state only if the a-cell is in a given state, without performing any operation on the v-cell or the a-cell.

$1PF2_a$ FFMs In this type of fault, performing a single-port operation to the a-cell sensitizes a fault in the v-cell. It consists of the following FFMs:

1. Disturb Coupling Fault (CFds): Two cells are said to have a *disturb coupling fault*, if application of an operation (read, transition write or non-transition write) performed on the a-cell, causes the v-cell to flip.

2. Undefined Disturb Coupling Fault (CFud): Two cells are said to have an *undefined disturb coupling fault*, if application of an operation (read, transition write or non-transition write) performed on the a-cell, forces the v-cell to an undefined state.

3. Idempotent Coupling Fault (CFid): Two cells are said to have an *idempotent coupling fault*, if a transition write operation performed on the a-cell, causes the v-cell to flip.

4. Inversion Coupling Fault (CFin): Two cells are said to have an *inversion coupling fault*, if a transition write operation performed on the a-cell, causes the inversion of the v-cell. The CFin consists of two pairs of FPs; the two FPs of each pair have to be present simultaneously: $\{<0w1;0/\uparrow/->, <0w1;1/\downarrow/->\}$, $\{<1w0;0/\uparrow/->, <1w0;1/\downarrow/->\}$.

$1PF2_v$ FFMs This type of fault has the property that the application of a single-port operation to the v-cell, with the a-cell in a certain state, sensitizes a fault in the v-cell. It consists of the following FFMs:

1. Transition Coupling Fault (CFtr): Two cells are said to have a *transition coupling fault* if a given logic value in the aggressor cell prevents a transition write operation on the victim.

2. Write Destructive Coupling Fault (CFwd): Two cells are said to have a *write destructive coupling fault* if a non-transition write operation performed on the v-cell results in a transition when the a-cell is in a given logic state.

3. Read Destructive Coupling Fault (CFrd): Two cells are said to have a *read destructive coupling fault* if a read operation performed on the v-cell changes the data in the v-cell and returns an *incorrect* value on the output, if the a-cell is in a given logic state.

4. Deceptive Read Destructive Coupling Fault (CFdrd): Two cells are said to have a *deceptive read destructive coupling fault* if a read operation performed on the v-cell changes the data in the v-cell and returns a *correct* value on the output, if the a-cell is in a given logic state.

5. Random Read Destructive Coupling Fault (CFrrd): Two cells are said to have a *random read destructive coupling fault* if a read operation performed on the v-cell changes the data in the v-cell and returns a *random* value on the output, if the a-cell is in a given logic state.

6. Incorrect Read Coupling Fault (CFir): Two cells are said to have an *incorrect read coupling fault* if a read operation performed on the v-cell returns the incorrect value on the output, when the a-cell is in a given logic state. Note that here the state of the v-cell is not changed.

7. Random Read Coupling Fault (CFrr): Two cells are said to have an *random read coupling fault* if a read operation performed on the v-cell returns a *random* value on the output, when the a-cell is in a given logic state. Note that here the state of the v-cell is not changed.

8. Undefined Write Coupling Fault (CFuw): Two cells are said to have an *undefined write coupling fault* if the v-cell is brought in an undefined state by a write operation performed on the v-cell, when the a-cell is in a given state.

9. Undefined Read Coupling Fault (CFur): Two cells are said to have an *undefined read coupling fault* if the v-cell is brought in an undefined state by a read operation performed on the v-cell, when the a-cell is in a given state.

A subset of above FFMs that covers all the FPs listed in table 2.3 needs to be selected. An analysis of the defined FFMs shows that all introduced FFMs are necessary except the CFid and CFin. These two FFMs have been introduced for historical reasons [6].

2.1.4 Single-Port Dynamic Faults

Dynamic faults are those faults that require more than one operation (read/write) in order to be sensitized. All parts of memory can be subject to these faults. For example, Address Decoder Open Faults (ADOF) are related to dynamic faults in the address decoder, dynamic Read Destructive Faults (dRDFs) are

dynamic faults linked to failures in the core-cell and Un-Restored Destructive Write Faults (URDWFs) are dynamic faults due to failures either in the precharge circuit or in the write driver [5].

Dynamic faults can be categorized based on the number of operations needed to sensitize each fault, as well as the number of memory cells involved in the faulty behavior. In the literature, the single and two-cell two-operation dynamic faults are the only investigated type of dynamic faults [7, 5].

In the rest of this section, single-port dynamic faults will be classified. This classification is mostly based on the study done in [7].

2.1.4.1 Single-Cell Fault Primitives

Each particular FP is denoted by $< S/F/R >$, where $S \in \{i(Od)_1(Od)_2 : O \in \{r, w\}, i \in \{0,1\}\ , d \in \{0,1\}\}$ for two-operation dynamic FPs, $F \in \{0,1\}$, and $R \in \{0,1,-\}$. Based on the values of S, F, and R, a list of 30 detectable single-cell 2-operation dynamic FPs can be compiled. This list is shown in table 2.5.

Out of these 30 FPs, only a subset of 12 FPs has been demonstrated to be realistic [5, 7]. These 12 FPs are assigned to three functional fault models which will be described in section 2.1.4.2.

2.1.4.2 Single-Cell Functional Fault Models

As mentioned in section 2.1.4.1, a subset of 12 single-cell dynamic fault primitives are shown to be realistic. These fault primitives are associated to three functional fault models as shown in table 2.6.

Each of these FFMs is described in the following:

1. Dynamic Read Disturb Fault (dRDF): In this fault, a write operation immediately followed by a read operation, changes the logical value stored in the memory cell and returns an *incorrect* output.

2. Dynamic Deceptive Read Disturb Fault (dDRDF): In this fault, a write operation immediately followed by a read operation, changes the logical value stored in the memory cell, but returns the *correct* output.

3. Dynamic Incorrect Read Fault (dIRF): In this fault, a write operation immediately followed by a read operation, does not change the logical value stored in the memory cell but returns an *incorrect* output.

Table 2.5: The Complete Set of Single-Cell 2-Operation Dynamic Fault Primitives

#	S	F	R	FP	#	S	F	R	FP
1	$0w0w0$	1	-	$< 0w0w0/1/- >$	2	$0w0w1$	0	-	$< 0w0w1/0/- >$
3	$0w0r0$	0	1	$< 0w0r0/0/1 >$	4	$0w0r0$	1	0	$< 0w0r0/1/0 >$
5	$0w0r0$	1	1	$< 0w0r0/1/1 >$					
6	$0w1w0$	1	-	$< 0w1w0/1/- >$	7	$0w1w1$	0	-	$< 0w1w1/0/- >$
8	$0w1r1$	0	0	$< 0w1r0/0/0 >$	9	$0w1r1$	0	1	$< 0w1r1/0/1 >$
10	$0w1r1$	1	0	$< 0w1r1/1/0 >$					
11	$1w0w0$	1	-	$< 1w0w0/1/- >$	12	$1w0w1$	0	-	$< 1w0w1/0/- >$
13	$1w0r0$	0	1	$< 1w0r0/0/1 >$	14	$1w0r0$	0	0	$< 1w0r0/0/0 >$
15	$1w0r0$	1	1	$< 1w0r0/1/1 >$					
16	$1w1w0$	1	-	$< 1w1w0/1/- >$	17	$1w1w1$	0	-	$< 1w1w1/0/- >$
18	$1w1r1$	0	0	$< 1w1r1/0/0 >$	19	$1w1r1$	0	1	$< 1w1r1/0/1 >$
20	$1w1r1$	1	0	$< 1w1r1/1/0 >$					
21	$0r0w0$	1	-	$< 0r0w0/1/- >$	22	$0r0w1$	0	-	$< 0r0w1/0/- >$
23	$0r0r0$	0	1	$< 0r0r0/0/1 >$	24	$0r0r0$	1	0	$< 0r0r0/1/0 >$
25	$0r0r0$	1	1	$< 0r0r0/1/1 >$					
26	$1r1w0$	1	-	$< 1r1w0/1/- >$	27	$1r1w1$	0	-	$< 1r1w1/0/- >$
28	$1r1r1$	0	0	$< 1r1r1/0/0 >$	29	$1r1r1$	0	1	$< 1r1r1/0/1 >$
30	$1r1r1$	1	0	$< 1r1r1/1/0 >$					

Table 2.6: List of Single-Cell Two-Operation Dynamic Functional Fault Models

#	FFM	Fault Primitives
1	dRDF	$< 0w0r0/1/1 >$, $< 1w1r1/0/0 >$, $< 0w1r1/0/0 >$, $< 1w0r0/1/1 >$
2	dDRDF	$< 0w0r0/1/0 >$, $< 1w1r1/0/1 >$, $< 0w1r1/0/1 >$, $< 1w0r0/1/0 >$
3	dIRF	$< 0w0r0/0/1 >$, $< 1w1r1/1/0 >$, $< 0w1r1/1/0 >$, $< 1w0r0/0/1 >$

Table 2.7: List of Two-Cell Two-Operation Dynamic Functional Fault Models

#	FFM	Fault Primitives
1	dCFds	$< 0w0r0, 0/1/- >$, $< 0w0r0, 1/0/- >$, $< 1w1r1, 1/0/- >$, $< 1w1r1, 0/1/- >$, $< 0w1r1, 0/1/- >$, $< 1w0r0, 1/0/- >$, $< 0w1r1, 1/0/- >$, $< 1w0r0, 0/1/- >$
2	dCFrd	$< 0, 0w0r0/1/1 >$, $< 1, 0w0r0/1/1 >$, $< 1, 1w1r1/0/0 >$, $< 0, 1w1r1/0/0 >$, $< 0, 0w1r1/0/0 >$, $< 1, 0w1r1/0/0 >$, $< 1, 1w0r0/1/1 >$, $< 0, 1w0r0/1/1 >$
3	dCFdrd	$< 0, 0w0r0/1/0 >$, $< 1, 0w0r0/1/0 >$, $< 1, 1w1r1/0/1 >$, $< 0, 1w1r1/0/1 >$, $< 0, 0w1r1/0/1 >$, $< 1, 0w1r1/0/1 >$, $< 1, 1w0r0/1/0 >$, $< 0, 1w0r0/1/0 >$
4	dCFir	$< 0, 0w0r0/0/1 >$, $< 1, 0w0r0/0/1 >$, $< 1, 1w1r1/1/0 >$, $< 0, 1w1r1/1/0 >$, $< 0, 0w1r1/1/0 >$, $< 1, 0w1r1/1/0 >$, $< 1, 1w0r0/0/1 >$, $< 0, 1w0r0/0/1 >$

2.1.4.3 Two-Cell Fault Primitives

Each two-cell two-operation dynamic fault primitive can be denoted by $< S/F/R >$ where S can be one of the following:

$$S_{aa} = a(iO_1d_1O_2d_2)v(j)$$
$$S_{av} = a(iO_1d_1)v(jO_2d_2)$$
$$S_{va} = v(jO_1d_1)a(iO_2d_2)$$
$$S_{vv} = a(i)v(jO_1d_1O_2d_2)$$

The subscripts a and v in the SOS names indicate whether each of the two operations is performed on the aggressor or the victim, respectively. For example, $< v(0r0)a(1r1)/1/- >$ stands for an FP sensitized by performing a $0r0$ on the victim first and then performing a $1r1$ on the aggressor. After performing the sensitizing sequence, a 1 is detected in the victim instead of the expected 0.

Based on possible combinations of S, F, and R, an exhaustive list of all 192 possible two-cell two-operation dynamic FPs is compiled and given in [7]. Out of these 192 FPs, a subset of 32 FPs has been demonstrated to be realistic [7, 5] and is associated to a list of four functional fault models. These functional fault models, considered to be realistic, are described in section 2.1.4.4.

2.1.4.4 Two-Cell Functional Fault Models

The list of realistic two-cell two-operation dynamic fault primitives can be compiled to a set of four FFMs. The list of these FFMs is given table 2.7.

Each of the these FFMs is described in the following:

1. Dynamic Disturb Coupling Fault (dCFds): In this type of fault, a write operation, immediately followed by a read operation on the aggressor cell, causes the victim cell to flip.
2. Dynamic Read Disturb Coupling Fault (dCFrd): Two cells are said to have a *dynamic read disturb coupling fault* if a write operation immediately followed by a read operation performed on the victim cell when the aggressor is in a certain state, changes the logical value stored in the memory and returns an *incorrect* output.
3. Dynamic Deceptive Read Disturb Coupling Fault (dCFrd): Two cells are said to have a *dynamic deceptive read disturb coupling fault* if a write operation immediately followed by a read operation performed on the victim cell when the aggressor is in a certain state, changes the logical value stored in the memory but returns the *correct* output.
4. Dynamic Incorrect Read Disturb Coupling Fault (dCFir): Two cells are said to have a *dynamic incorrect read disturb coupling fault* if a write operation immediately followed by a read operation performed on the victim cell when the aggressor is in a certain state, *does not* affect the logical value stored in the memory but returns an *incorrect* output.

As mentioned before, the n-cell m-operation dynamic faults where $n \geq 2$ and $m \geq 2$ have never been considered in the literature. From a theoretical point of view, it is possible to compute the FPs to model these types of dynamic faults, but in practice, it has not been considered. Indeed, up to now, it has never been demonstrated that these types of dynamic faults are a realistic set of dynamic faults [5].

2.2 Memory Test Algorithms

There have been many test algorithms proposed for testing memories. The first tests proposed in the literature were *ad-hoc* test algorithms. They were developed without resorting to formal fault models and proofs. Tests such as *Scan*, *Galpat*, and *Walking 1/0* belong to this class. The main drawback of these algorithms is the high complexity, generally not linear with respect to memory size, that makes them not applicable to nowadays large memories[5].

To overcome the complexity issues of ad-hoc test solutions, the class of *march tests* has been introduced and defined in [11]. In section 2.2.1, some of the well known algorithms of this class have been studied.

Traditionally, most of the memory test algorithms have been targeted against bit-oriented memories while most of the memories currently used in the industry are word-oriented. In [12], a systematic method has been presented to convert bit-oriented tests to word-oriented algorithms. In section 2.2.8, we will review this method for converting bit-oriented march tests to word-oriented algorithms.

2.2.1 March Test Algorithms

A march test is a test algorithm composed of a sequence of *march elements.* Each march element (ME) is a sequence of memory operations applied sequentially on a certain memory cell before proceeding to the next one. The way in which one moves from a certain address to another is called *address order* (AO). The AO characterizes the ME and can be done in either one of two address orders: an increasing ($\Uparrow$) address order (e.g., from address 0 to $n-1$) or a decreasing ($\Downarrow$) address order which is the opposite of $\Uparrow$ address order. When the address order is irrelevant the symbol $\Updownarrow$ is used. Hereinafter, a march test is denoted by '{...}' bracket and a march element using a '(...)' bracket. The ith operation is defined as op_i where $op_i \in \{wd, rd\}, d \in \{0, 1\}$ in which 'wd' means 'write logic value d in the memory cell' and 'rd' means 'read the content of the memory cell and verify that its value is equal to d.' [5]

In general, the complexity of a March test is in the order of $O(n)$, i.e., linear with respect to the size of the memory under test. More precisely, it is defined as the number of memory operations composing the different March elements multiplied by the size of the memory (n).

March tests are nowadays the dominant type of memory tests used in the industry due to their low complexity [5].

In the following sections, we will concentrate on this class of algorithms for SRAM testing, and some of the well known algorithms will be introduced and studied.

2.2.2 March LR: A Test for "Realistic" Linked Faults

March LR algorithm was proposed by van de Goor et al. in [10] in 1996. In the reference article, first the universe of linked faults is reduced to *realistic* linked faults. Then it has been shown that March LR is able to detect all static simple faults as well as realistic static linked faults. As in the time when March LR was proposed dynamic faults had not become a serious problem in SRAMs, dynamic faults have not been taken into account.

In [10], the following linked faults have been assumed to be unrealistic:

1. Linked faults which include CFins.
2. Linked faults which include the following two linked CFids: $<\uparrow;\uparrow>_{j,i}$ # $<\downarrow;\downarrow>_{j,i}$ or $<\uparrow;\uparrow>_{i,k}$ # $<\downarrow;\downarrow>_{i,k}$; The authors have argued that this types of linked faults are unrealistic due to the structure of SRAMs and DRAMs.
3. Linked faults which include two linked CFdsts with opposite fault effects: $< x;\downarrow>_{j,i}$ # $< y;\uparrow>_{j,i}$ or $< x;\downarrow>_{i,k}$ # $< y;\uparrow>_{i,k}$ where $x, y \in \{r0, r1, w0, w1\}$. These faults are also considered unrealistic due to the structure of SRAMs and DRAMs.

March LR algorithm has a complexity of O(14n) and is as follows:

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

Simple March LR is not able to detect data retention faults (DRFs). Two extensions have been proposed by the authors to make it able to detect DRFs as well.

March LRD is able to detect single DRFs (where the cell fails to retain '1' or '0' state, but not both). This algorithm is an extension of March LR as follows:

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

In this notation, *Del* refers to a delay to let the data retention fault affect the victim cell.

March LRDD is able to detect single and double DRFs (where the cell fails to retain both '0' and '1' logic values). This algorithm is an extension of March LR as follows:

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

In [10], it has been shown that March LR outperforms the following older algorithms in terms of fault coverage:

- March C: $\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Updownarrow (r0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$ [13]
- March C-: $\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$ [14]

- March A: $\{\Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\}$ [15]
- March B: $\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\}$ [15]
- Algorithm B: $\{\Updownarrow (w0); \Uparrow (r0, w1, w0, w1); \Uparrow (r1, w0, r0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, r1, w0)\}$ [13]
- MOVI: $\{\Downarrow (w0); \Uparrow (r0, w1, r1); \Uparrow (r1, w0, r0); \Downarrow (r0, w1, r1); \Downarrow (r1, w0, r0)\}$ [16]
- March M: $\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0); \Updownarrow (r0); \Uparrow (r0, w1); \Updownarrow (r1); \Uparrow (r1, w0, r0, w1); \Updownarrow (r1); \Downarrow (r1, w0)\}$ [17]

Based on these results, and considering that the above mentioned algorithms are mostly very old and not suited to new technologies and corresponding faults, we will not review these algorithms and focus on the more advanced and up-to-date algorithms.

2.2.3 March SS: A Test for All Static Simple RAM Faults

March SS algorithm was proposed by Hamdioui et al. in [18] in 2002. This algorithm targets simple static fault models that have been shown to exist for Random Access Memories. In the reference article, it has been shown that this algorithm is able to detect all of these fault models.

March SS has a complexity of O(22n) and is as follows:

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

In [18], it has been shown that March SS outperforms the following older algorithms in terms of fault detection capabilities:

- MATS+: $\{\Updownarrow (w0); \Uparrow (r0, w1); \Downarrow (r1, w0)\}$[11]
- March C-: $\{\Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0)\}$ [14]
- March B: $\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0, r0, w1); \Uparrow (r1, w0, w1); \Downarrow (r1, w0, w1, w0); \Downarrow (r0, w1, w0)\}$ [15]

- PMOVI: $\{\Downarrow (w0); \Uparrow (r0, w1, r1); \Uparrow (r1, w0, r0); \Downarrow (r0, w1, r1); \Downarrow (r1, w0, r0)\}$ [16]
- March U: $\{\Updownarrow (w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0, w1); \Downarrow (r1, w0, r0, w1); \Downarrow (r1, w0)\}$ [19]
- March LR: $\{\Updownarrow (w0); \Downarrow (r0, w1); \Uparrow (r1, w0, r0, w1); \Uparrow (r1, w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0)\}$ [10]
- March SR: $\{\Downarrow (w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0, r0); \Uparrow (w1); \Downarrow (r1, w0, r0, w1); \Downarrow (r1, r1)\}$ [20]

It should be noted that while March SS outperforms all the above mentioned algorithms in terms of fault detection capabilities, in terms of complexity it is the slowest. While March SS has a complexity of O(22n), the second slowest algorithm of the above is March B with a complexity of O(17n).

2.2.4 March 13N and March 9N

March 13N and March 9N were introduced by Dekker et al. in [21] in 1988. March 9N is proposed for SRAMs with combinational R/W logic, while March 13N is targeted for memories with sequential R/W logic. For each of these algorithms, a data retention test extension is also proposed.

March 9N, as the name suggests, has a complexity of O(9n) and is as follows:

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

March 9N with data retention test is as follows:

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

March 13N, as the name suggests, has a complexity of O(13n) and is as follows:

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

March 13N with data retention test is as follows:

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

March 9N and March 13N have been shown to detect the following fault models:

- Stuck-at Faults (SAFs)
- Stuck-Open Faults[3]
- Transition Faults (TFs)
- State Coupling Faults (CFsts)
- Multiple Access Faults: Multiple Access Fault is a special case of Disturb Coupling Faults (CFds), in which case, a *write* operation on the a-cell with value $x(x \in \{0, 1\})$, forces a write operation in the v-cell with the same value x.
- Data retention faults

It is worth emphasizing that March 9N and March 13N target *the same set of faults* and the only difference between them is that March 9N is designed for SRAMs with combinational R/W logic while March 13N is designed for SRAMs with sequential R/W logic.

In the reference paper, March 9N and March 13N have also been extended to support testing of word-oriented memories. But since in the following sections of this chapter, we are going to present a general methodology for this type of extension, these extensions are not presented here.

2.2.5 March AB: A State-of-the-Art March Test for Realistic Static Linked Faults and Dynamic Faults

March AB was proposed by Bosio et al. in [4] in 2006. This march test targets all realistic memory static linked faults and dynamic unlinked faults as well as all static simple faults targeted by March SS.

The authors have first defined a subset of all possible static linked faults and dynamic faults as *realistic*. This subset is defined in the following.

[3] As mentioned in section 2.1.4.2, Stuck-Open Fault is a special case of No-Access Faults (NAFs)

2.2.5.1 Dynamic Faults

Dynamic faults are in turn categorized into single-cell two operation dynamic faults and two-cell two operation dynamic faults. Dynamic faults which need more than two operations to be sensitized have not been considered since there has been no evidence of their existence in the real world.

March AB targets the following single-cell two-operation dynamic FFMs:

- Dynamic Read Disturb Faults (dRDF)
- Dynamic Deceptive Read Disturb Faults (dDRDF)
- Dynamic Incorrect Read Disturb Faults (dIRF)

In addition, March AB targets the following two-cell two-operation dynamic FFMs:

- Dynamic Disturb Coupling Faults (dCFds)
- Dynamic Read Disturb Coupling Faults (dCFrd)
- Dynamic Deceptive Read Disturb Coupling Faults (dCFdrd)
- Dynamic Incorrect Read Disturb Coupling Faults (dCFir)

A comparison between the fault models mentioned above and the fault models in tables 2.6 and 2.7 shows that March AB essentially targets all realistic two-operation dynamic fault models.

2.2.5.2 Static Linked Faults

As mentioned in section 2.1.2.1, static linked faults are two or more static faults that share the same aggressor and/or victim cell. The static linked faults considered to be realistic and targeted by March AB are summarized in table 2.8.

As can be seen in table 2.8, two-cell static LFs have been divided into $LF2_{aa}$, $LF2_{av}$, and $LF2_{va}$ classes. Each of these classes is described in the following:

$LF2_{aa}s$ are the linked faults that share both the a-cell and the v-cell.

$LF2_{av}s$ are the linked faults in which FP_1 is a two-cell FP and FP_2 is a single-cell FP.

$LF2_{va}s$ are the linked faults in which FP_1 is a single-cell FP and FP_2 is a two-cell FP.

Table 2.8: Realistic Static Linked Faults Targeted by March AB

LF Type	FFM
Single-Cell Static LFs	$TF \rightarrow WDF$
	$WDF \rightarrow WDF$
	$DRDF \rightarrow WDF$
	$TF \rightarrow RDF$
	$WDF \rightarrow RDF$
	$DRDF \rightarrow RDF$
Two-Cell Static $LF2_{aa}s$	$CFds \rightarrow CFds$
	$CFtr \rightarrow CFds$
	$CFwd \rightarrow CFds$
	$CFdr \rightarrow CFds$
	$CFds \rightarrow CFwd$
	$CFtr \rightarrow CFwd$
	$CFwd \rightarrow CFwd$
	$CFdr \rightarrow CFwd$
	$CFds \rightarrow CFrd$
	$CFtr \rightarrow CFrd$
	$CFwd \rightarrow CFrd$
	$CFdr \rightarrow CFrd$
Two-Cell Static $LF2_{av}s$	$CFds \rightarrow WDF$
	$CFtr \rightarrow WDF$
	$CFwd \rightarrow WDF$
	$CFdr \rightarrow WDF$
	$CFds \rightarrow RDF$
	$CFtr \rightarrow RDF$
	$CFwd \rightarrow RDF$
	$CFdr \rightarrow RDF$
Two-cell Static $LF2_{va}s$	$WDF \rightarrow CFds$
	$TF \rightarrow CFds$
	$DRDF \rightarrow CFds$
	$WDF \rightarrow CFwd$
	$TF \rightarrow CFwd$
	$DRDF \rightarrow CFwd$
	$WDF \rightarrow CFrd$
	$TF \rightarrow CFrd$
	$DRDF \rightarrow CFrd$

It has also been shown that *realistic* three-cell linked faults are composed of two two-cell fault models that share at least one cell; therefore realistic three-cell linked faults can be represented by the same fault primitives used to represent two-cell linked faults. Therefore these types of faults are targeted by March AB as well.

2.2.5.3 March AB Test Algorithm

March AB has a complexity of O(22n) and is as follows:

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

In [22], another variant of March AB, called AB1, is proposed which with a complexity of O(11n) is able to detect single-cell two-operation dynamic faults. Unfortunately, no claim has been made by the authors with respect to coverage of static faults (simple or linked). March AB1 is as follows:

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

To the extent of our knowledge, the only other algorithms in the literature that target the same set of faults as March AB and March AB1, are March RAW with complexity of O(26n) and March RAW1 with complexity of O(13n) respectively. For the sake of completeness, we will review these tests in section 2.2.6.

2.2.6 March RAW: Testing Static and Dynamic Faults in Random Access Memories

March RAW was proposed by Hamdioui et al. in [3] in 2002. Like March AB, March RAW also has two variants: March RAW and March RAW1.

March RAW and March RAW1 target the same fault set as March AB and March AB1 that were described in sections 2.2.5.1 and 2.2.5.2.

March RAW1 has a complexity of O(13n) and targets realistic single-cell dynamic faults as March AB1. This march test is as follows:

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

March RAW has a complexity of O(26n) and targets realistic two-cell dynamic fault as March AB. This march test is as follows:

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

It should be mentioned again that March RAW and March RAW1 are outperformed by the newer March AB and March AB1 respectively in terms of complexity while targeting the same fault set. Since these march tests target the same dynamic memory fault set of March AB and March AB1, they are presented here for the sake of completeness.

2.2.7 Effect of Address Ordering

In the literature, there have been several attempts for targeting dynamic memory faults. Some of these attempts involve using a specific address order for the march test, which should be derived based on the memory layout and the dynamic fault set being targeted. As an example, an address order has been used in [23] that enables March C- algorithm to detect dynamic read destructive faults (dRDFs) on a specific SRAM layout from Infineon Technologies.

However, March RAW and March AB consider the memory and the associated fault models from a functional point of view. This means that the address ordering is not important in performing these march tests, and these tests are designed to detect the targeted faults regardless of the memory layout and physical location of memory cells. Therefore, as shown in [4], March AB and March RAW can achieve a high fault coverage for dynamic faults in a functional model of an SRAM while the fault coverage of March C- for dynamic faults in such a memory is marginal.

Since address ordering does not affect the fault coverage of the presented march tests for their targeted fault set, in our implementation we will use a sequential address ordering to minimize the resources required for implementation of the MemBIST. This type of address ordering can be implemented using a simple counter while specific hardware should be designed to implement other address orderings.

2.2.8 Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories

Almost all the march tests reviewed in this chapter were designed for testing bit-oriented memories (BOMs). In the real world, on the other hand, most of the memories are word-oriented (WOMs). This means a read or write operation

on a single bit in the memory cannot be performed and all the bits in a memory word are addressed simultaneously.

The *traditional* method for testing WOMs consists of the repeated application of a test for bit-oriented memories, whereby a different *data background (DB)* is used during each application. This means that the length of the BOM test will be *multiplied* by the number of DBs used. The disadvantages of these methods are test time inefficiency and limited fault coverage for *intra-word* (faults within words) coupling faults[12]. Due to these disadvantages, we will not go into details about the traditional method of WOM testing. Instead, we will discuss a more efficient and up-to-date method, proposed by van de Goor et al. in [12] in 2003. This systematic method can be used to convert all march tests designed for BOMs into WOM tests. All subsequent parts of this section are based on this article.

The conversion consists of concatenating to the march test for *inter-word faults* (fault between words), a march test designed for intra-word faults. This means that the length of the intra-word march test will be *added* to the length of the BOM test. For construction of the test for intra-word faults, a minimal *data background sequence (DBS)*, capable of sensitizing the targeted CFs has to be established.

In the reference article, first the fault models for WOMs are categorized as follows:

1. *Single-cell faults:* These are the classical *stuck-at faults (SAFs), transition faults (TFs), data retention faults (DRFs),* and *read disturb faults*
2. *Fault between memory cells:* This class of faults consists of coupling faults (CFs). These can be further divided into two subclasses:
   (a) *Inter-word faults:* These are classical CFs whereby aggressor and victim cells belong to *different words.* They will be detected by properly designed BOM tests.
   (b) *Intra-word faults:* These are CFs whereby the aggressor and victim cells belong to the *same word.* A special *intra-word test* has to be designed to detect this class of faults.

2.2.8.1 Detection of Single-Cell and Inter-Word Faults

Any BOM test can be converted into a WOM test to detect single-cell and inter-word faults as follows:

1. The $w0$ operation should be replaced with a *"w-data background"* denoted as "w_D" whereby *any* DB is acceptable. For example "$w0...0$" (an all-zero DB), "$w01...01$" (a DB pattern of a repeated sequence of "01"), etc., whereby the length of bit string is B bits. The $w1$ operation should be replaced with a write operation which writes *inverted data background, i.e., "$w_{\bar{D}}$".*

2. The $r0$ and $r1$ operations should be replaced with *"r-data background" (r_D)* and *"r-inverted-data background" ($r_{\bar{D}}$).*

3. In the equations expressing the required number of operations for a test, n (representing the number of bits in the chip) has to be replaced by n/B (representing the number of words in the chip).

2.2.8.2 Detection of Intra-Word Faults

The DBs to be used for intra-word faults are determined by used intra-word coupling fault models. If the layout of the memory is not known, fault models called *unrestricted CF models (uCFs)* should be applied. If the layout of the memory is known, *restricted CF models (rCFs)* and *concurrent restricted CF models (crCFs)* can be used to simplify the march tests, improving the performance. Each of these fault models is explained in more detail in the following sections.

Tests for Unrestricted Intra-Word Faults In unrestricted fault model, there can be a fault between any two cells in a word. There are three coupling fault types for which DBs have been proposed in the reference article, in order to detect intra-word types:

1. Unrestricted intra-word CFsts (uCFsts)

2. Unrestricted intra-word CFids (uCFids)

3. Unrestricted intra-word CFdsts (uCFdsts)

Detection of these faults is discussed in the following sections.

Data Backgrounds for Unrestricted Intra-Word CFsts (uCFsts) In order to detect uCFsts, all states of two arbitrary cells c_i and c_j should be checked, i.e., the states of $(c_i, c_j) \in \{(0,0),(0,1),(1,0),(1,1)\}$.

Table 2.9: DBs for uCFsts (B=8)

#	Normal	#	Inverse
0	00000000	1	11111111
2	01010101	3	10101010
4	00110011	5	11001100
6	00001111	7	11110000

One way to test all these states is that for every DB, to start by an all-zero word, in the second DB, set all bits b_i such that $(i\,mod\,2) \geq 1$ to 1, in the third DB, set all b_i such that $(i\,mod\,4) \geq 2$ and so on. To clarify this, a set of DBs for B=8 is shown in table 2.9.

The nature of uCFsts is such that only the *states* of the cells are relevant for sensitizing and detecting a fault; therefore, the DBs of the set can be applied in any order.

Data Backgrounds for Unrestricted Intra-Word CFids (uCFids) In order to detect uCFids, the intra-word test has to use a set of DBs, which have to be applied to the memory in a particular sequence. This set of DBs is called a *data background sequence (DBS).* For a two-bit memory the DBS $S_2 = 00, 11, 00, 01, 10, 01$ has been derived which can be split into: $S_2' = 00, 11, 00$ and $S"_2 = 01, 10, 01$.

As an example, the method to extend this DBS for an 8-bit memory is presented which can easily be extended to wider memories.

In order to extend the DBS to WOMs with 8-bit words, $W_8 = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$ the following steps should be performed:

1. *Level 0:* For each cell-pair (c_i, c_{i+1}), we apply the DBS S_2 for 2-bit words, replicated four times to fill the 8-bit word.
2. *Level 1:* For each cell-pair (c_i, c_{i+2}) we apply only the DBS $S"_2 = 01, 10, 01$; this is sufficient because the DBS S_2' has already been applied in *Level 0.*
3. *Level 2:* For each cell pair (c_i, c_{i+4}) we apply the DBS $S"_2$.

After *Level 2*, all uCFids for an 8-bit WOM are sensitized. The DBS for 8-bit WOM is shown in table 2.10.

Table 2.10: DBS S_8 for uCFids (B=8)

#	Data background	Level
	$c_0c_1c_2c_3c_4c_5c_6c_7$	
0	00000000	
1	11111111	0
2	00000000	0
3	01010101	0
4	10101010	0
5	01010101	0
6	00110011	1
7	11001100	1
8	00110011	1
9	00001111	2
10	11110000	2
11	00001111	2

Data Background for Unrestricted CFdsts (uCFdsts) An intra-word test for uCFdsts uses a sequence of data background operations; referred to as the *DB operation sequence (DBOS).*

In order to derive the DBOS, first a data background sequence needs to be derived. It has been shown that the DBS for uCFdsts is identical to that of uCFids which was derived in the previous section.

Based on this DBS, the DBOS can be derived. First the DBOS for a 2-bit memory has been derived:

$$\Omega_{t2} = w11, r11, r11, w00, r00, r00, w01, w10, r10, r10, w01, r01, r01$$

This DBOS can be split into:

$$\Omega'_{t2} = w11, r11, r11, w00, r00, r00$$

$$\Omega"_{t2} = w01, w10, r10, r10, w01, r01, r01$$

In order to derive the DBOS for an 8-bit memory the following steps should be taken:

1. *Level 0:* For each cell-pair (c_i, c_{i+1}), we generate the DBOS found for 2-bit words (Ω_{t2}).

Table 2.11: DBOS for CFdsts (B=4)

#	Op.	DB	Level	#	Op.	DB	Level
		$c_0c_1c_2c_3$				$c_0c_1c_2c_3$	
0		0000	0	11	w	0101	0
1	w	1111	0	12	r	0101	0
2	r	1111	0	13	r	0101	0
3	r	1111	0	14	w	0011	1
4	w	0000	0	15	w	1100	1
5	r	0000	0	16	r	1100	1
6	r	0000	0	17	r	1100	1
7	w	0101	0	18	w	0011	1
8	w	1010	0	19	r	0011	1
9	r	1010	0	20	r	0011	1
10	r	1010	0				

2. *Level 1:* For each cell-pair (c_i, c_{i+2}), we apply only the DBOS $\Omega"_{t2}$. This is sufficient, because the DBOS Ω'_{t2} has already been applied in *Level 0.* The first operation $w01$ operation in $\Omega"_{t2}$ does not need to be followed by a read operation because it is only used to connect the DBOS of *Level 0* and *Level 1* while producing a transition write operation on both bits.

3. *Level 2:* For each cell-pair (c_i, c_{i+4}) we apply the the DBOS $\Omega"_{t2}$.

This method can easily be extended to higher memory widths. The DBOS for a 4-bit memory is shown in table 2.11.

WOM March Tests for Unrestricted Intra-Word Coupling Faults Now that DB sequences and DB operation sequences have been derived, they can be used to convert any given BOM test to a WOM test which additionally covers intra-word CFs (uCFsts, uCFids, and/or uCFdsts).

Such a WOM march test is a concatenation of two march tests: {*inter-word* march test} {*intra-word* march test}. The *inter-word* march test consists of a traditional BOM test modified such that the bit-operations "$r0$," "$r1$," "$w0$," "$w1$" are replaced with the word operations "r_D," "$r_{\bar{D}}$," "w_D," "$w_{\bar{D}}$," whereby the all 0s DB value has been chosen for D to optimize ground bounce. The *intra-word* march test is used to detect the intra-word CFs. It consists of a single march element of the following form:

1. For uCFsts: $\Updownarrow (w_{D_0}, r_{D_0}, ..., w_{D_{d-1}}, r_{D_{d-1}})$ whereby D_0 through D_{d-1} are taken from the set of DBs of table 2.9 (for B=8), such that both the normal and inverse values are covered. These DBs can be applied in any order.

2. For uCFids: $\Updownarrow (w_{D_0}, r_{D_0}, w_{D_1}, r_{D_1}, ..., r_{D_{d-2}}, w_{D_{d-1}}, r_{D_{d-1}})$, whereby D_0 through D_{d-1} represents the DBS of table 2.10 (for B=8).

3. For uCFdsts: $\Updownarrow (w_{D_0}, r_{D_0}, r_{D_0}, ..., w_{D_{d-1}}, r_{D_{d-1}}, r_{D_{d-1}})$, whereby the DBOS (consisting of the operations together with the DBs) is taken from table 2.11 (for B=4).

The above intra-word test may be modified as follows, without any impact on the fault coverage:

1. Extra read operations may be added, for example to make the test more symmetric and/or to detect possible faults of other fault models.

2. The single march element may be divided into any number of march elements and, for each march element, the address can be chosen freely.

The above freedom to modify the intra-word test allows for:

1. Test time reduction. If march elements of the intra-word test can be made identical to march elements of the inter-word test, those intra-word march elements can be removed.

2. Extra fault coverage for unanticipated faults. This can be optimized when the intra-word march test has the following properties:

   (a) It consists of several march elements because each march element performs a sweep over the memory.

   (b) All march elements start with a read; this allows for detection of CFs.

   (c) The address orders of the march elements of the intra-word march test should vary as much as possible. This maximizes the probability of detecting dynamic faults.

Relationships of WOM March Tests and The Impact of Memory Organizations It can be shown that:

1. WOM tests for uCFsts, uCFids, and uCFdsts cover SAFs, TFs, and RDFs.

2. WOM tests for uCFids cover all uCFsts.

3. WOM tests for uCFdsts cover all uCFids.

In addition, it can be shown that based on the organization of the word-oriented memory, only a subset of fault models may be sufficient to cover all intra-word CFs. A memory can be organized in the following manners:

1. *Adjacent:* A q-bit row in a sub-array contains $w \times B$ bits. The B bits of a word are adjacent and, therefore, the proposed WOM tests for uCFsts, uCFids, and uCFdsts have to be applied.

2. *Interleaved* (a.k.a *folded*): A q-bit row in a sub-array contains $w \times B$ bits. The B bits of a word are spread across B groups in such a way that the bits of a B-bit word are interleaved with $w - 1$ bits of the other B-bit words in that row. Therefore, only the test for uCFsts has to be applied to one word of each fold in order to verify the I/O data path of each fold. The uCF fault model is adequate for checking CFs in data paths.

3. *Sub-arrays:* Each bit of a B-bit word is taken from a different sub-array, while all B bits have the same address in each sub-array. Similarly to the interleaved case, only a test for uCFsts has to be applied to one word in each sub-array, in order to verify the I/O data path.

Tests for Restricted Intra-Word Faults One reasonable restriction of intra-word CFs is that an a-cell can only influence its left or its right physical neighbor. These *restricted CFs*, denoted as *rCFs,* apply to the following fault types: rCFsts, rCFids, and rCFdsts. The way BOM tests can be converted into WOM tests for rCFs depends on the physical memory organization:

1. Adjacent

   - rCFsts: In order to detect rCFsts in an adjacent memory organization, all states of adjacent cells should be checked. Table 2.12 lists the required DBs to perform such a test for a memory with a word width of 8. These DBs can be extended to any arbitrary word width and the number of DBs will be a constant 4.

   - rCFids: In order to sensitize all rCFids in an adjacent memory organization, only the DBS for *Level 0* of section 2.2.8.2 is required. This reduces the number of DBs to 6 and the length of the DBS to 10 for any given word width.

Table 2.12: Set of 8-bit DBs for rCFsts

#	Normal	#	Inverse
0	00000000	1	11111111
2	01010101	3	10101010

- rCFdsts: In order to sensitize all rCFdsts in an adjacent memory organization, only the DBOS for *Level 0* of section 2.2.8.2 is required. This reduces the number of DBs to 6 and the length of the DBOS to 13, for any given word width.

2. Interleaved (folded): In case of interleaved memory organization, the B bits of a word are spread across B groups such that there are no neighboring cells belonging to the same word. This means that there are no adjacent cells within a word. One only has to check the data I/O paths of each fold by applying the test for rCFsts to one word in each fold.

Tests for Concurrent-Restricted Intra-Word Faults WOMs are read and written B bits at a time, i.e., B bits concurrently. Conceptually, B-1 bits in a word may concurrently act as a-cells for a given v-cell. When the physical layout of the cells in a row is known, tests can be simplified significantly when the a-cells are restricted to be the two physical neighbors of the v-cell, which are the most likely a-cells. This coupling fault model will be called *concurrent-restricted CF model (crCF model)*. The crCF model has to be analyzed for the fault types: *crCFst*, *crCFid*, and *crCFdst*.

In order to detect crCFsts, all states of three adjacent cells $(i-1, i, i+1)$ should be checked:

$$(i-1, i, i+1) \in \{(0,0,0), (0,0,1), (0,1,0), \\ (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$$

The set of required DBs to sensitize crCFsts in an 8-bit-wide memory has been shown in table 2.13. It can be extended to any number of bits, and the number of DBs is 8 and independent of the memory width.

The DBS required to sensitize the crCFids in an 8-bit memory is shown in table 2.14. This can be extended to any word width, and the length of the DBS is 22 and independent of the word width.

Table 2.13: Set of 8-Bit DBs for crCFsts

#	Normal	#	Inverse
0	00000000	1	11111111
2	00100100	3	11011011
4	01001001	5	10110110
6	01101101	7	10010010

Table 2.14: 8-Bit DBS for crCFids

#	DB $c_0c_1c_2c_3c_4c_5c_6c_7$	Operation
0	00000000	-
1	11111111	w,r
2	00000000	w,r
3	00100100	w
4	11011011	w,r
5	00100100	w,r
6	10110110	w
7	01001001	w,r
8	10110110	w,r
9	01101101	w
10	10010010	w,r
11	01101101	w,r

Table 2.15: DBOS for crCFdsts, B=4

#	Op.	DB $c_0c_1c_2c_3$	#	Op.	DB $c_0c_1c_2c_3$
0		0000	14	w	0100
1	w	1111	15	w	1011
2	r	1111	16	r	1011
3	r	1111	17	r	1011
4	w	0000	18	w	0100
5	r	0000	19	r	0100
6	r	0000	20	r	0100
7	w	0010	21	w	0110
8	w	1101	22	w	1001
9	r	1101	23	r	1001
10	r	1101	24	r	1001
11	w	0010	25	w	0110
12	r	0010	26	r	0110
13	r	0010	27	r	0110

The DBOS for sensitizing crCFdsts in a 4-bit-wide memory is shown in table 2.15. It can be extended to any word width and the length of the DBOS is 27, independent of the word width.

For an adjacent memory organization, the proposed tests for crCFsts, crCFids, and/or crCFdsts have to be applied. For interleaved (folded) and sub-array memory organizations, only the data I/O paths of each fold/sub-array have to be verified using a test for crCFsts.

Complexity of Intra-Word Tests The complexity of the proposed intra-word fault tests is summarized in table 2.16. In this table, *TL* denotes the required test length for each fault model. Note that the complexity of rCF and crCF fault models is independent of the word width.

2.3 Conclusion

In this chapter, first different functional fault models were introduced, defined and classified.

Table 2.16: Number of DBs and TLs for Different CF Types

CF Type	# of DBs	TL
uCFst	$2 \times \beta + 2$	$4 \times \beta$
rCFst	4	4
crCFst	8	12
uCFid	$3 \times \beta + 3$	$6 \times \beta + 4$
rCFid	6	10
crCFid	12	22
uCFdst	$3 \times \beta + 3$	$7 \times \beta + 6$
rCFdst	6	13
crCFdst	12	27

Note: $\beta = \lceil log_2 B \rceil$.

After that, some state-of-the-art memory test algorithms were reviewed. These algorithms included March LR, March SS, March 13N, March 9N, March RAW and March AB. While the older algorithms (March LR, March SS, March 13N and March 9N) mostly targeted static memory faults, newer algorithms (March RAW and March AB) targeted both static and dynamic faults.

At the end of this chapter a systematic method for conversion of bit-oriented memory tests to word-oriented memory tests was reviewed in detail, and realistic simplifications to reduce the complexity of the test were discussed.

Chapter 3

Comparison of MemBIST Algorithms

The objective of this chapter is to compare different MemBIST algorithms reviewed in chapter 2. The information presented in this chapter will be used to select the appropriate MemBIST algorithm for implementation in hardware.

This chapter is based on the comparisons performed in the literature to compare fault detection capabilities of different algorithms. Unfortunately, some of such studies are very old, and do not cover the algorithms evaluated in this report such as March LR, March RAW, and March AB. Neither do they cover the new fault models present in new embedded SRAMS. Therefore we will not consider these studies and will confine our attention to more up-to-date articles which cover the new algorithms and fault models.

It should be noted that these studies do not cover all the algorithms presented in section 2.2 at the same time; each study usually covers a subset of them. Nevertheless, these comparisons provide valuable information for selection of a suitable algorithm for implementation in hardware.

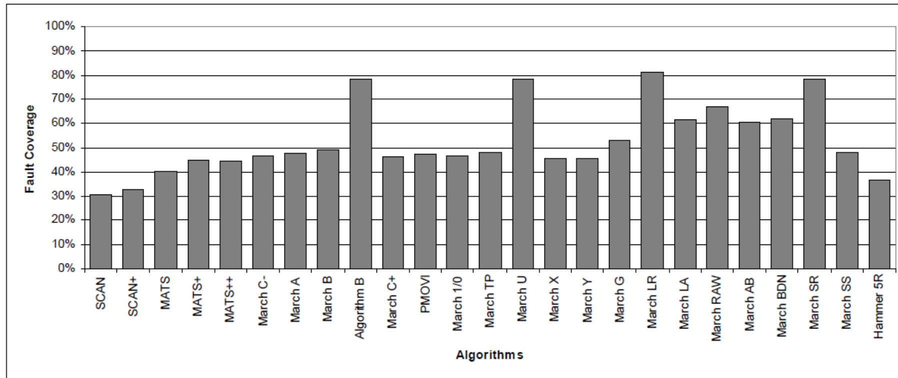


Figure 3.1: Results of the Comparison Performed in [2]

3.1 Effectiveness of Memory Test Algorithms and Analysis of Fault Distribution in SRAMs: A Case Study Based on Industrial Test Results

A study done by Linder et al., and presented in [2], has compared the fault coverage of 25 different march tests based on industrial test results. This study is particularly interesting, because the results reflect the physical defect coverage instead of functional fault coverage. This means that the faults which happen more often in reality have a higher weight in determining the fault coverage, compared to the faults that rarely happen. In other words, the fault coverage here is not based on any functional fault model set and instead is based on the coverage of actual physical defects in SRAMs produced by Infineon Technologies.

The results of this comparison are shown in figure 3.1. As can be seen, March LR provides the highest fault coverage among the 25 evaluated algorithms. This is due to March LR's high capabilities in detecting linked faults, and the fact that these types of faults were very common in the particular set of memories, tested in this study. Dynamic faults on the other hand, for which March AB and March RAW were designed, have not occurred as often in this particular set of memories.

It should be noted however, that even though dynamic faults did not happen as often as static linked faults, they were present, and probably account for a considerable portion of the faults that were not detected by March LR. A fault

Table 3.1: Result of the Fault Coverage Estimation Performed in [3]

FFM	March Tests							
	MATS+ (5n)	March C- (10n)	March B (17n)	PMOVI (13n)	March U (13n)	March SR (14n)	March LA (22n)	March LR (14n)
$dRDF$	0%	0%	50%	50%	50%	50%	50%	50%
$dDRDF$	0%	0%	0%	50%	0%	0%	50%	0%
$dIRF$	0%	0%	50%	50%	50%	50%	50%	50%
$dCFds_{xw\bar{x}r\bar{x}}$	0%	0%	50%	87.5%	50%	50%	100%	50%
$dCFds_{xwxrx}$	0%	0%	0%	0%	0%	0%	0%	0%
$dCFrd$	0%	0%	25%	50%	25%	25%	50%	25%
$dCFdrd$	0%	0%	0%	37.5%	0%	0%	50%	0%
$dCFir$	0%	0%	25%	50%	25%	25%	50%	25%

distribution extracted in the study, shows that linked faults accounted for 9% of the total faults in the memories, while dynamic faults accounted for 7%.

This shows that in order to achieve a low defect-per-million (DPM), it is necessary to address both types of faults. The authors conclude that a combination of different march tests, with different capabilities (e.g., March LR and March RAW to target both linked faults and dynamic faults) is necessary to achieve a high fault coverage.

3.2 Testing Static and Dynamic Faults in Random Access Memories

Another comparison of different MemBIST algorithms has been performed in [3], the study in which March RAW was proposed to address the inability of other algorithms in detection of dynamic faults. As March AB was proposed after March RAW, it is not present in this comparison.

In this study, functional fault models are used to estimate the fault coverage of march tests. Therefore, all FFMs contribute equally to the fault coverage, regardless of their actual probability in the real world. The result of this comparison is shown in table 3.1.

As can be seen in the table, none of the studied march tests can provide a high fault coverage with respect to dynamic faults. March RAW and March RAW1 are then proposed to address this problem. March RAW1 can detect

Table 3.2: Comparison of March Tests with Respect to Dynamic Faults as Presented in [4]

FFM	March Tests				
	March C- 10n	March RAW1 13n	March SS 22n	March AB 22n	March RAW 26n
dRDF	0.39%	100%	25%	100%	100%
dDRDF	0.39%	100%	0.39%	100%	100%
dIRF	0.39%	100%	25%	100%	100%
dCFds	0.83%	0%	50%	100%	100%
dCFrd	0.39%	50%	50%	100%	100%
dCFrdr	0.39%	50%	0.39%	100%	100%
dCFir	0.39%	50%	50%	100%	100%

all 1-cell dynamic FFMs considered in table 3.1 (dRDFs, dDRDFs and dIRFs), while March RAW can detect all 2-cell dynamic FFMs in this table (these results are not reflected in the table).

3.3 March AB, A State-of-the-Art March Test for Realistic Static Linked Faults and Dynamic Faults in SRAMs

Another interesting comparison of fault coverage of different march tests has been done in [4], the article in which March AB was proposed. As March RAW was introduced before March AB, it is included in this study as well.

The comparison performed in [4] consists of two parts; in the first part algorithms are compared with respect to dynamic faults and in the second part, they are compared with respect to linked faults.

The result of comparison with respect to dynamic faults is presented in table 3.2.

Of particular interest in this table, is comparison with March SS and March RAW. The results show that March AB provides a much better fault coverage with respect to dynamic faults compared to March SS, while having the same complexity of O(22n). It is also claimed that March AB covers the same set of static faults as March SS. It can also be seen that March AB provides the same fault coverage with respect to dynamic faults as March RAW, while maintaining

Table 3.3: Comparison of March Tests with Respect to Linked Faults as Presented in [4]

FFM	March Tests							
	LR 14n	A 15n	B 17n	LA 22n	AB 22n	MSL 23n	RAW 26n	SL 41n
Single-cell	75%	66%	75%	83%	100%	100%	100%	100%
$LF2_{aa}$	82%	75%	70%	87%	100%	100%	100%	100%
$LF2_{av}$	75%	60%	64%	83%	100%	100%	100%	100%
$LF2_{va}$	80%	73%	73%	86%	100%	100%	100%	100%
All	80%	69%	70%	86%	100%	100%	100%	100%

a lower complexity.

The result of comparison with respect to linked faults is presented in table 3.3.

It can easily be seen that among the algorithms that provide 100% linked fault coverage, March AB provides the lowest complexity with a complexity of O(22n).

3.4 Conclusion

In this chapter, a few articles on comparison of different march tests were reviewed.

Based on these comparisons, it appears that some disagreement exists with respect to the best algorithm for detection of linked faults. While the results of [2] can be interpreted to the conclusion that March LR provides the best linked fault coverage, results of [4] indicate that March AB and March RAW are able to detect 100% of two-cell linked faults. After a more detailed look into these algorithms, as mentioned in [2], it seems that the superior results of March LR in that study can be associated to the following march elements in March LR that is not covered in March AB or March RAW:

$$...w1); \Uparrow (r1, w0, r0, w1); \Uparrow (r1...$$

and

$$...w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0...$$

With respect to dynamic faults however, without any disagreement, it seems that March RAW and March AB provide the best fault coverage, which is 100% for two-cell two-operation faults. It should be noted however that March AB provides a lower complexity compared to March RAW.

It seems that a combination of March LR with either March AB or March RAW can provide a good fault coverage with respect to all realistic faults existing in current SRAMs.

Chapter 4

Implementation of the MemBIST

In this chapter the implementation of the MemBIST unit will be discussed. At first, the architecture of a typical Atmel AVR micro-controller will be presented and different architectural choices for implementation of the MemBIST will be discussed. Afterwards, an appropriate MemBIST algorithm will be chosen for implementation. At the end the technical details of implementation will be explained.

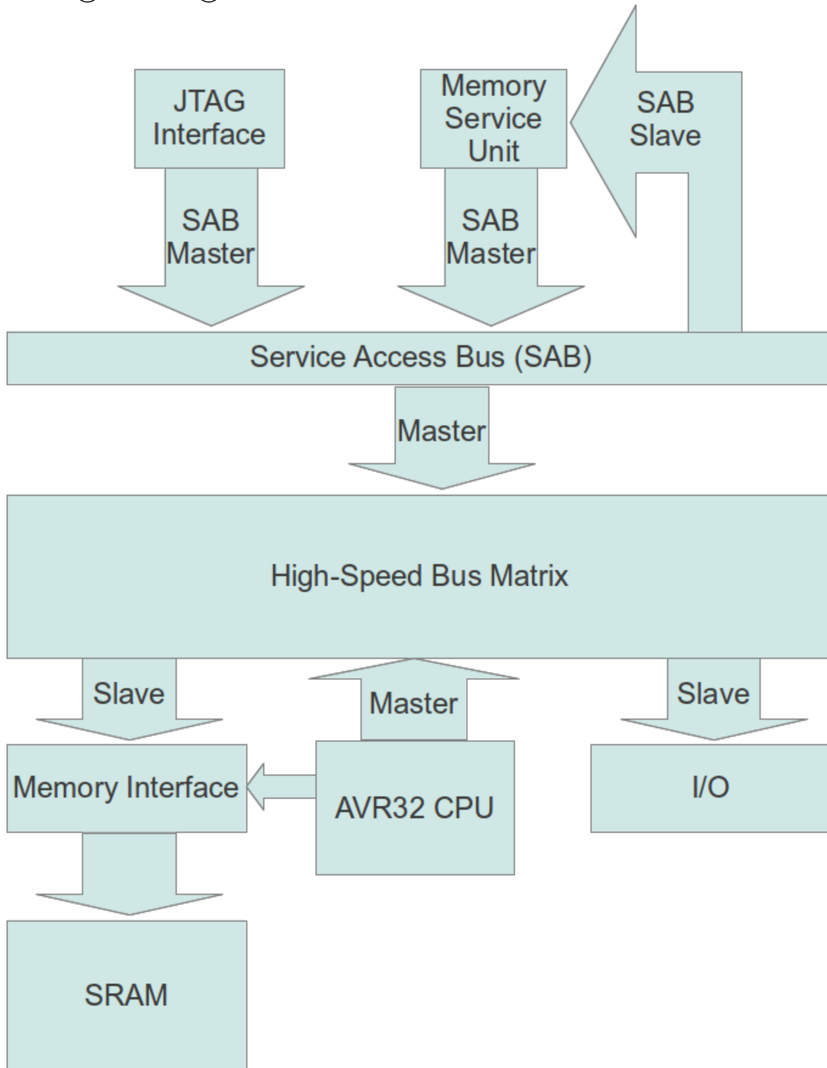
4.1 Proposed Architecture

The first step in implementation of the MemBIST unit is finding a suitable architecture, providing efficient access to memories for the MemBIST unit.

Figure 4.1 shows a block diagram of interconnection structures in a typical Atmel AVR32® micro-controller. Note that in order to keep simplicity, only the basic blocks which are relevant for this project are shown in the diagram. The SRAM shown in the diagram represents CPU's main memory, connected to the CPU using a Memory Interface unit. Additional SRAM units may exist in a design which may or may not be connected to the High-Speed Bus Matrix.

Two possible concepts were considered for the block in which the MemBIST module could be implemented. Each of these choices has its own pros, cons, and implications. These two choices have been described in the following.

Figure 4.1: A Block Diagram of Relevant Interconnection Structures in a Typical Atmel® AVR32® Micro-Controller



Centralized MemBIST in Memory Service Unit In this scheme, MemBIST unit is implemented inside Memory Service Unit and accesses the SRAM blocks through Service Access Bus and High-Speed Bus Matrix.
In this architecture, all SRAMs in a chip are tested using the same MemBIST unit which leads to reduced area overhead. Furthermore, using this architecture, the functional path to the SRAM will be tested in addition to the SRAM itself.
On the other hand, all memory transactions need to go through Service Access Bus, High-Speed Bus Matrix and possibly Memory Interface which can lead to longer testing times due to communication delay. Furthermore, since more blocks are involved in the MemBIST operation, this can potentially lead to higher power consumption during testing.
Another important implication of this architecture is that MemBIST can only access the memory using words as wide as the Service Access Bus which is 32-bits wide. Therefore, if a wider memory is to be tested, each transaction needs to be segmented into smaller transactions and reassembled in the target SRAM. This will in turn require a segmentation-and-reassembly protocol which potentially results in communication overhead.

Specific MemBIST for Each SRAM Block In this scheme, every SRAM block will have a MemBIST unit, directly attached to it, using a dedicated test port.
Using a separate MemBIST unit for each SRAM block will lead to a higher area overhead. Furthermore, this approach only provides coverage for the SRAM and the functional path to the SRAM will not be tested.
On the other hand, the memory access time will be shorter which in turn leads to a shorter testing time. Furthermore, multiple blocks can be tested at the same time which can further shorten the testing time.
Another, and possibly the most important implication of this architecture is that the MemBIST can access the memory using the full bus-width of the memory in a single cycle. This shortens the access time and furthermore simplifies the communication protocol and the MemBIST.

As described above, each of these architectures has its pros and cons. Based on the number of SRAM blocks in a design, the second approach can lead to an unacceptable area overhead, while the first architecture can be prohibitively slow for wide memories.

Based on these observations, it was decided that a combination of both architectures will be most suitable for our application. In our proposed architecture,

the centralized approach will be used to test SRAMs which have a width equal to or lower than that of the Service Access Bus, while a specific MemBIST will be attached to every wider memory. Since typically there are not many such memories in each design, this will not impose a very high area overhead on the design, while keeping the testing time to an acceptable level.

4.2 Proposed MemBIST Algorithm

The next step in implementation of the MemBIST, is the selection of the MemBIST algorithm. Based on the study done in chapter 3, it was concluded that a combination of March LR and March AB can provide an acceptable fault coverage. Such an algorithm should then be converted to a WOM test in order to test the word-oriented embedded SRAMs. In this section, this algorithm will be derived and optimized and then a theoretical performance analysis will be given. Practical simulation-based performance evaluation is carried out in chapter 5.

4.2.1 Selected BOM Test

As concluded in section 3.4, a combination of *March LR* and *March AB* was found suitable for the embedded SRAMs targeted in this research. March LR and March AB are described in more detail in section 2.2 and are as follows:

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

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

Without any optimization, a combined test algorithm consisting of March LR ($O(14n)$) and March AB ($O(22n)$) will have a complexity of $O(36n)$ and will be as follows:

$$\{\Updownarrow(w0)[M_0]; \Downarrow(r0,w1,r1,w1,r1)[M_1]; \Downarrow(r1,w0,r0,w0,r0)[M_2]; \Uparrow(r0,w1,r1,w1,r1)[M_3]; \Uparrow(r1,w0,r0,w0,r0)[M_4]; \Updownarrow(r0)[M_5]; \Updownarrow(w0)[M_0']; \Downarrow(r0,w1)[M_1']; \Uparrow(r1,w0,r0,w1)[M_2']; \Uparrow(r1,w0)[M_3']; \Uparrow(r0,w1,r1,w0)[M_4']; \Uparrow(r0)[M_5']\}$$

It can be seen that the march element M_1' and M_3' in March LR are covered by M_1 and M_4 in March AB. Therefore M_1' and M_3' are redundant and do

not serve any purpose with regard to fault coverage. However they do set up the memory values for M'_2 and M'_4. In order to achieve a better performance, M'_1 can be replaced with simple $\Updownarrow (w1)$. M'_0 is also unnecessary, because it is completely covered by M_0, and therefore can be removed.

After these optimizations, the combined march test will have a complexity of $O(33n)$ and be as follows:

$$March\,AB + LR: \{\Updownarrow (w0); \Downarrow (r0, w1, r1, w1, r1); \Downarrow (r1, w0, r0, w0, r0); \Uparrow (r0, w1, r1, w1, r1); \Uparrow (r1, w0, r0, w0, r0); \Updownarrow (r0); \Updownarrow (w1); \Uparrow (r1, w0, r0, w1); \Uparrow (r1, w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0)\}$$

This march test can even be further optimized. Note that the only reason why we need to replace M'_1 with a $\Updownarrow (w1)$ march element, is that the final values of memory, after performing March AB will be set to zero. However, since March AB is a symmetric test algorithm (for example both stuck-at-0 and stuck-at-1 are detected), all the values written and read from the cells can be inverted without affecting the fault coverage [5]. This means that the following test can be used instead of March AB without any impact on the fault coverage:

$$March\,ABi: \{\Updownarrow (w1)[M"_0]; \Downarrow (r1, w0, r0, w0, r0)[M"_1]; \Downarrow (r0, w1, r1, w1, r1)[M"_2]; \Uparrow (r1, w0, r0, w0, r0)[M"_3]; \Uparrow (r0, w1, r1, w1, r1)[M"_4]; \Updownarrow (r1)[M"_5]\}$$

At the end of this march test, the values in the memory are preset to 1. It should be noted that M'_1 and M'_3 are still redundant, because they are covered by $M"_2$, $M"_4$ respectively. M'_0 is also unnecessary because after removal of M'_1, it does not serve any testing purpose. After making these modifications, $M"_5$ will also be unnecessary, because the $r1$ operation of M'_2 can be used to check the result of $M"_4$.

After these optimizations, the final optimized march test will have a complexity of $O(32n)$ and will be as follows:

$$March\,ABi + LR: \{\Updownarrow (w1); \Downarrow (r1, w0, r0, w0, r0); \Downarrow (r0, w1, r1, w1, r1); \Uparrow (r1, w0, r0, w0, r0); \Uparrow (r0, w1, r1, w1, r1); \Uparrow (r1, w0, r0, w1); \Uparrow (r1, w0); \Uparrow (r0, w1, r1, w0); \Uparrow (r0)\}$$

It should be noted that this algorithm will be the base BOM test which should be extended to a WOM test before being used in the MemBIST. Since this algorithm is a combination of March LR and March AB, hereinafter we will refer to it simply as *March AB+LR*.

4.2.2 Extension to WOM Test

As mentioned before, the specific targets of this research are embedded SRAMs on Atmel AVR32 micro-controllers. This means that the memory layout is known prior to implementation of the SRAM (this would not be true if the MemBIST was expected to test the external memories as well). Therefore the neighbors of each memory cell are known and a test designed for *concurrent-restricted intra-word fault model* will provide enough fault coverage for the target memories.

Among concurrent-restricted intra-word fault models, *crCFdst* fault model is the most general fault model which requires 12 different data backgrounds (DBs) in 27 operations in order to be tested thoroughly.

In this section we will use a Hexadecimal 32-bit notation to describe these DBs and operations. But it should be noted that the number of required DBs and operations are independent of memory width and the test can easily be extended to any arbitrary word size.

The inter-word march test, as derived in section 4.2.1, is as follows:

$$\begin{gathered}
March\,ABi + LR:\ \{\Updownarrow (w_{FFFFFFFF});\\
\Downarrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{00000000}, r_{00000000});\\
\Downarrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{FFFFFFFF}, r_{FFFFFFFF});\\
\Uparrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{00000000}, r_{00000000});\\
\Uparrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{FFFFFFFF}, r_{FFFFFFFF});\\
\Uparrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{FFFFFFFF});\\
\Uparrow (r_{FFFFFFFF}, w_{00000000}); \Uparrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{00000000});\\
\Uparrow (r_{00000000})\}
\end{gathered}$$

Assuming the initial value of memory cells is set to zero at the beginning of intra-word test (which is true at the end of the inter-word test derived above), the intra-word march test consists of a single march element as follows:

$$\begin{gathered}
\Updownarrow (w_{FFFFFFFF}, r_{FFFFFFFF}, r_{FFFFFFFF}, w_{00000000},\\
r_{00000000}, r_{00000000}, w_{24924924}, w_{DB6DB6DB},\\
r_{DB6DB6DB}, r_{DB6DB6DB}, w_{24924924}, r_{24924924},\\
r_{24924924}, w_{49249249}, w_{B6DB6DB6}, r_{B6DB6DB6},\\
r_{B6DB6DB6}, w_{49249249}, r_{49249249}, r_{49249249},\\
w_{6DB6DB6D}, w_{92492492}, r_{92492492}, r_{92492492},\\
w_{6DB6DB6D}, r_{6DB6DB6D}, r_{6DB6DB6D})
\end{gathered}$$

This march element is derived using the method demonstrated in section 2.2.8 and specifically the DBOS shown in table 2.15. For optimization purposes,

theoretically, the intra-word march element can be modified in the following ways without any impact on fault coverage [12]:

1. Extra read operations may be added, for example to make the test more symmetric and/or to detect possible faults of other fault models.

2. The single march element may be divided into any number of march elements and, for each march element, the address can be chosen freely.

However, based on the inter-word and intra-word tests mentioned above, even considering these degrees of freedom, it doesn't seem possible to modify the intra-word test in a way that some march elements become similar to those in the inter-word test. Therefore the final WOM test will be a simple concatenation of the inter-word and intra-word test and will have a complexity of $O(59n)$:

$$\{\Updownarrow (w_{FFFFFFFF}); \Downarrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{00000000}, r_{00000000});$$
$$\Downarrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{FFFFFFFF}, r_{FFFFFFFF});$$
$$\Uparrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{00000000}, r_{00000000});$$
$$\Uparrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{FFFFFFFF}, r_{FFFFFFFF});$$
$$\Uparrow (r_{FFFFFFFF}, w_{00000000}, r_{00000000}, w_{FFFFFFFF});$$
$$\Uparrow (r_{FFFFFFFF}, w_{00000000}); \Uparrow$$
$$(r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}, w_{00000000}); \Uparrow (r_{00000000});$$
$$\Updownarrow (w_{FFFFFFFF}, r_{FFFFFFFF}, r_{FFFFFFFF}, w_{00000000},$$
$$r_{00000000}, r_{00000000}, w_{24924924}, w_{DB6DB6DB},$$
$$r_{DB6DB6DB}, r_{DB6DB6DB}, w_{24924924}, r_{24924924},$$
$$r_{24924924}, w_{49249249}, w_{B6DB6DB6}, r_{B6DB6DB6},$$
$$r_{B6DB6DB6}, w_{49249249}, r_{49249249}, r_{49249249},$$
$$w_{6DB6DB6D}, w_{92492492}, r_{92492492}, r_{92492492},$$
$$w_{6DB6DB6D}, r_{6DB6DB6D}, r_{6DB6DB6D})\}$$

4.2.3 Theoretical Performance Analysis

As currently implemented in Atmel AVR micro-controllers, March LR algorithm has been used in a bit-oriented manner to test the embedded SRAMs on the micro-controllers. It is worth mentioning that this was an effective algorithm prior to DSM technologies when static simple and linked faults were the dominant types of memory faults. In this thesis, we use this MemBIST as a reference for benchmarking purposes and evaluation of our proposed MemBIST.

For a 32-bit wide memory of depth n, our optimized combination of March LR and March AB, when converted to WOM with concurrent-restricted model,

has a complexity of $O(59n)$ while the reference MemBIST implementation has a complexity of:

$$O((14 \times 32)n) = O(448n)$$

This shows that the proposed MemBIST implementation can theoretically provide up to 659% improvement in test time, while providing additional fault coverage for dynamic faults, at least for inter-word fault models.

This improvement will be practically verified using simulation in chapter 5.

4.2.4 Extension to Retention Testing

Extension to data retention testing can easily be done by concatenating the WOM test with the following march elements to any march test:

$$\Updownarrow (w_{00000000}, r_{00000000}); Del; \Updownarrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}); Del; \Updownarrow (r_{FFFFFFFF})$$

where Del represents a delay for a specified time. This march element provides coverage for both single DRFs (where the faulty cell fails to maintain a 0 or 1 state but not both), and double DRFs (where the faulty cell fails to maintain both states) [10].

Although it may be possible to reduce the test complexity by combining this march element with the march elements in the main WOM test, it is better to separate the retention testing from the rest of the tests because this way, the retention testing can be enabled or disabled on demand.

In theory, it is also possible to take advantage of the existing values in memory after performing the WOM test. In this case the memory words will contain $0x6DB6DB6D$ value. We could use this value to optimize the test further by using the following march elements for retention testing instead:

$$Del; \Updownarrow (r_{6DB6DB6D}, w_{92492492}, r_{92492492}); Del; \Updownarrow (r_{92492492})$$

However, this approach would eliminate the independence of the two march tests (March AB+LR and retention testing). Therefore, we decided to use the non-optimized retention testing. This way, either test can be used independently, and data retention testing can even be used with other WOM tests without any for modifications.

With this extension, the final march test will be as follows:

$$\{\Updownarrow(w_{FFFFFFFF});\Downarrow(r_{FFFFFFFF},w_{00000000},r_{00000000},w_{00000000},r_{00000000});$$
$$\Downarrow(r_{00000000},w_{FFFFFFFF},r_{FFFFFFFF},w_{FFFFFFFF},r_{FFFFFFFF});$$
$$\Uparrow(r_{FFFFFFFF},w_{00000000},r_{00000000},w_{00000000},r_{00000000});$$
$$\Uparrow(r_{00000000},w_{FFFFFFFF},r_{FFFFFFFF},w_{FFFFFFFF},r_{FFFFFFFF});$$
$$\Uparrow(r_{FFFFFFFF},w_{00000000},r_{00000000},w_{FFFFFFFF});$$
$$\Uparrow(r_{FFFFFFFF},w_{00000000});\Uparrow$$
$$(r_{00000000},w_{FFFFFFFF},r_{FFFFFFFF},w_{00000000});\Uparrow(r_{00000000});$$
$$\Updownarrow(w_{FFFFFFFF},r_{FFFFFFFF},r_{FFFFFFFF},w_{00000000},$$
$$r_{00000000},r_{00000000},w_{24924924},w_{DB6DB6DB},$$
$$r_{DB6DB6DB},r_{DB6DB6DB},w_{24924924},r_{24924924},$$
$$r_{24924924},w_{49249249},w_{B6DB6DB6},r_{B6DB6DB6},$$
$$r_{B6DB6DB6},w_{49249249},r_{49249249},r_{49249249},$$
$$w_{6DB6DB6D},w_{92492492},r_{92492492},r_{92492492},$$
$$w_{6DB6DB6D},r_{6DB6DB6D},r_{6DB6DB6D});\Updownarrow(w_{00000000},r_{00000000});$$
$$Del;\Updownarrow(r_{00000000},w_{FFFFFFFF},r_{FFFFFFFF});Del;\Updownarrow(r_{FFFFFFFF})\}$$

4.3 Implementation of the MemBIST Unit

Implementation of the MemBIST was done using RTL[1] Verilog. To minimize the area overhead, the MemBIST was embedded into Memory Service Unit and existing registers were reused in implementation of the MemBIST.

The implementation was done in two phases. In the first phase, March AB+LR algorithm was implemented and extended for WOM using crCFdst intra-word fault model. In the second phase, necessary extension was made to enable detection of data retention faults. Each of these steps is described in more detail in the following sections.

4.3.1 Implementation of March AB+LR

To implement the March AB+LR part of the MemBIST, a new command was added to Memory Service Unit to activate the execution of this algorithm. The MemBIST was implemented as a finite state machine with two sets of registers as state registers.

The first set of registers was used to point to the march element being executed and the second set was used to point to the active memory operation within the march element. This was done to maximize the reuse of the registers in the Memory Service Unit. The state register of the old bit-oriented MemBIST

[1] Register Transfer Level

was reused to point to march elements and the bit-counter of the bit-oriented test was reused to point to each operation within march elements. This helps to minimize the area overhead when it is desired to have both MemBISTs implemented in the system (for example during transition period from the old MemBIST to the new MemBIST). Since the longest march element (intra-word march element) has 27 operations, this 5-bit counter was long enough and no changes were necessary to adapt it for the new purpose.

After initiation of a MemBIST operation, a status register will be set to indicate that MemBIST unit is busy. Upon successful completion of a MemBIST operation, the status register will be set accordingly, indicating a successful MemBIST operation. In case of a failure, the status register will indicate an error and the march element, memory operation, and memory address which led to the failure will be saved. This information can help diagnosis algorithms to locate the fault and identify the type of fault. It should be noted, however, that fault diagnosis is beyond the scope of this thesis.

4.3.2 Implementation of Data Retention Testing

As mentioned in section 4.2.4, data retention testing can be done by performing the following march elements:

$$\Updownarrow (w_{00000000}, r_{00000000}); Del; \Updownarrow (r_{00000000}, w_{FFFFFFFF}, r_{FFFFFFFF}); Del; \Updownarrow (r_{FFFFFFFF})$$

In the above march element *Del* indicates a delay which is usually chosen as equal to a few milliseconds. This is a huge amount of time compared to most testing operations during MemBIST. Furthermore, during this delay, the MemBIST is not doing anything, and it's simply waiting for data retention faults to be sensitized. The testing efficiency can be improved by allowing the ATE to perform other useful tasks during this period.

To achieve this goal, it was decided to use three different commands to activate each of the three march elements. This way, the MemBIST will return the control to the tester between the march elements. The tester will then reactivate MemBIST after the *Del* amount of time has passed. In the meantime, the tester can apply test vectors to other parts of the chip, as long as they do not change the values inside the memory being tested.

Another advantage of separating data retention testing from March AB+LR is that it can be used with any previous or future MemBISTs. For example it can be used in conjunction with the March LR MemBIST to enable DRF testing for that algorithm.

4.4 Conclusion

In this chapter the actual implementation of the MemBIST unit was discussed. At first, based on the results from chapter 3, a combination of March LR and March AB was derived and optimized which is expected to provide coverage for both static and dynamic, simple and linked faults.

This algorithm was then extended for word-oriented memories using the concurrent-restricted fault model described in section 2.2.8. It was estimated that the proposed algorithm can provide up to 659% performance improvement over the current bit-oriented March LR-based MemBIST.

Afterwards, the proposed MemBIST algorithm was extended to provide coverage for data retention faults.

At the end, the relevant technical implementation details of the MemBIST inside Memory Service Unit were discussed.

Chapter 5

Evaluation and Experimental Results

As explained in chapter 1, the main requirements that an efficient MemBIST should fulfill are the following:

Performance: The equipment used for testing semiconductor devices are typically very expensive. Therefore the required time for testing a device has a direct impact on the production costs. A good MemBIST algorithm should be able to test the device in a reasonable amount of time. This will lead to lower production costs and increase the profit margin of the product.

Area: Since MemBIST is a built-in unit, it will be implemented in every single device. Therefore, the silicon area of each device will be increased by the MemBIST which will result in increased production costs. A good MemBIST algorithm should impose a relatively low area overhead on the target chip. Since most of the device resources (e.g., registers) are not in use during MemBIST operation, it is a good idea to reuse these resources in implementation of the MemBIST.

Fault coverage: The main objective in implementation of any BIST unit is to detect as many faulty units as possible before shipping to the field. This requires a high fault coverage to be provided by the algorithm.

The production cost of a semiconductor can typically be calculated according to the following equation:

$$Cost = C_{area} + C_{test-time} + const.$$

In this equation, *const.* represents other production costs which are not related to test circuitry (such as packaging). C_{area} is the silicon cost and can be calculated as follows:

$$C_{area} = Area \times a$$

where a represents the cost of the silicon per area unit. Here, area unit can be considered, for example, as the number of gates in the design. Furthermore, $C_{test-time}$ is the cost of the ATE equipment and can be calculated as follows:

$$C_{test-time} = TestTime \times t$$

where t represents the cost of the ATE per time unit.

With the advance of semiconductor technology, the cost of silicon per area unit is becoming cheaper and cheaper, while the ATE equipment are becoming more expensive. This means that in the long run, testing time will be a more important factor compared to area.

In this chapter, we will evaluate the implemented MemBIST with respect to the above mentioned criteria. The implemented MemBIST will be compared to the current implementation of bit-oriented March LR as a reference. The results are normalized in such a way that the results of the reference MemBIST will be equal to 1. This is done to compensate for dependence of the results on technology, clock frequency, and synthesis parameters (e.g. optimization effort). After each evaluation, a discussion is made to explain the result in more detail. At the end, a explanation will be given with regard to how these results should be interpreted.

5.1 Performance Evaluation

The implemented MemBIST was simulated on an $8word \times 32bit$ embedded SRAM of an AVR micro-controller using Synopsys® VCS®.

The reference MemBIST was simulated using the same test-bench on the same memory.

The normalized run-times of the MemBIST units are shown in table 5.1. Note that this run-time does not include retention testing and only corresponds to WOM March AB+LR algorithm. This is due to three reasons:

Table 5.1: Run-time of MemBIST algorithms on an $8 \times 32bit$ reference memory

MemBIST Unit	MemBIST Algorithm	Normalized Run-Time
Newly Implemented MemBIST	Word-oriented March AB+LR with crCFdst intra-word extension	0.16
Reference MemBIST	Bit-oriented March LR	1.00

1. Data retention testing consists mostly of huge delays during which other useful tasks can be performed (e.g. testing other parts of the chip), therefore the run-time of data retention testing is not entirely an *overhead* in its real sense.

2. The reference MemBIST does not perform retention testing and therefore such a comparison would not be a fair one, since the MemBISTs would be targeting two completely different types of fault.

3. The data retention testing is really a separate test which can be used in conjunction with either MemBIST. Therefore it should not be considered as a part of WOM March AB+LR.

5.1.1 Discussion

As can be seen in table 5.1, the required testing time of the MemBIST is shortened by 84% which translates to 525% performance improvement over this particular implementation of bit-oriented March LR. This performance improvement is less than the estimated improvement in section 4.2.3. Our more detailed analysis showed that this difference is due to the fact that this particular March LR implementation is not completely bit-oriented and the initial $w0$ and the final $r0$ operations are done in a word-oriented manner.

It should be noted that even though this test is run on an $8 - word$ memory, due to the regular structure of SRAMs and MemBIST algorithm, the same improvement can be expected on all $32 - bit$ wide memories. In other words, since test times of both test algorithms (as well as any other march test) increase linearly with the memory size, the timing improvement should remain almost constant.

With reduction of memory width, this improvement becomes smaller, and eventually the bit-oriented test will become faster than the word-oriented test due to the long intra-word march element which will be run on every memory

Table 5.2: Area Report for Synthesis of Memory Service Unit

Synthesized Configuration	Normalized Area
Without any MemBIST	0.9994
With reference MemBIST	1.0000
With new MemBIST	1.0007
With both MemBISTs	1.0013

word. On the other hand, with increasing the memory width, this improvement will become even more significant due to the fact that the number of memory operations is independent from the memory width, thanks to crCFdst intra-word fault model. Based on the current trend in microprocessor technology (e.g., 64-bit architectures becoming more and more popular), it seems that the latter case will be more likely in the foreseeable future and therefore even more significant improvements can be expected over the bit-oriented march tests.

5.2 Area Evaluation

The micro-controller device was synthesized using Synopsys® Design Compiler®. The synthesis was done for 4 different configurations:

1. Neither MemBIST was implemented
2. Only the reference MemBIST implemented
3. Only the new MemBIST implemented
4. Both MemBISTs implemented

Similar synthesis parameters were used for all mentioned configurations. The area report of the synthesis of micro-controller are shown in table 5.2.

The device used here as a reference, was a typical Atmel 32-bit AVR micro-controller.

5.2.1 Discussion

As can be seen in table 5.2, either MemBIST imposes a marginal area overhead when compared to the micro-controller area. Although the area overhead of the proposed MemBIST is 100% more than the reference MemBIST, this amount

of area overhead is still too little to cause any considerable increase in the production costs.

This very low area overhead is due to the fact that all the registers used for implementation of MemBIST are reused from existing registers in the Memory Service Unit. Therefore this area overhead mostly consists of the control logic of the MemBIST state machine. This is expected to be higher than the reference MemBIST, considering that the proposed MemBIST has many more states (59 memory operations) compared to the reference MemBIST (14 memory operations).

5.3 Fault Coverage Evaluation

As mentioned in section 2.2.5, the WOM March AB+LR algorithm is expected to detect the following inter-word faults:

- All single-cell static faults (1PF1s, table 2.2)
- All two-cell static faults (1PF2s, table 2.4)
- Realistic single-cell two-operation dynamic faults (table 2.6)
- Realistic two-cell two-operation dynamic faults (table 2.7)

Additionally, as mentioned in section 2.2.8, this algorithm is expected to detect the following intra-word faults:

- State Coupling Faults (CFsts)
- Idempotent Faults (CFids)
- Disturb Coupling Faults (CFdsts)

In order to evaluate fault coverage, all static faults and realistic dynamic fault models (except the ones which involve a random output or undefined state) were implemented using behavioral Verilog and injected to the simulated memory. All of the fault primitives of each FFM from tables 2.2, 2.4, 2.6 and 2.7 were separately implemented and simulated.

Coupling faults were injected 2 times, once on cells from different memory words (to simulate inter-word faults) and once on cells within the same memory word (to simulate intra-word faults). For injection of intra-word faults, the aggressor and victim cells were chosen as two adjacent memory bits, due to the assumptions of concurrent-restricted fault model.

Table 5.3: Fault Simulation Results for Inter-Word Static Single-Cell Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
SF	$< 1/0/- >$	Yes	Yes
	$< 0/1/- >$	Yes	Yes
TF	$< 0w1/0/- >$	Yes	Yes
	$< 1w0/1/- >$	Yes	Yes
WDF	$< 0w0/\uparrow/- >$	Yes	No
	$< 1w1/\downarrow/- >$	Yes	No
RDF	$< r0/\uparrow/1 >$	Yes	Yes
	$< r1/\downarrow/0 >$	Yes	Yes
DRDF	$< r0/\uparrow/0 >$	Yes	Yes
	$< r1/\downarrow/1 >$	Yes	Yes
IRF	$< r0/0/1 >$	Yes	Yes
	$< r1/1/0 >$	Yes	Yes

In the following subsections, the results for inter-word and intra-word faults will be separately presented and discussed.

5.3.1 Inter-word Faults

The fault simulation results for inter-word faults are shown in tables 5.3, 5.4, 5.5, and 5.6.

5.3.1.1 Discussion

The fault coverage for inter-word FFMs is extracted from tables 5.3, 5.4, 5.5, and 5.6 and shown in table 5.7.

As shown in table 5.7, the proposed MemBIST successfully manages to detect all targeted inter-word faults, including all static faults and realistic dynamic faults.

The reference MemBIST, as expected from a March LR-based MemBIST, provides a reasonable fault coverage for static simple faults and static coupling faults. However it provides a poor coverage for dynamic faults. Since the reference MemBIST does not have a data retention fault testing extension, it was not evaluated for these fault models.

Table 5.4: Fault Simulation Results for Intra-Word Static Coupling Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
CFst	$< 0; 0/1/- >$	Yes	Yes
	$< 0; 1/0/- >$	Yes	Yes
	$< 1; 0/1/- >$	Yes	Yes
	$< 1; 1/0/- >$	Yes	Yes
CFds	$< xwy; 0/\uparrow/- >$	Yes	Yes
	$< xwy; 1/\downarrow/- >$	Yes	Yes
	$< rx; 0/\uparrow/- >$	Yes	Yes
	$< rx; 1/\downarrow/- >$	Yes	Yes
CFid	$< 0w1; 0/\uparrow/- >$	Yes	Yes
	$< 0w1; 1/\downarrow/- >$	Yes	Yes
	$< 1w0; 0/\uparrow/- >$	Yes	Yes
	$< 1w0; 1/\downarrow/- >$	Yes	Yes
CFin	$\{< 0w1; 0/\uparrow/- >, < 0w1; 1/\downarrow/- >\}$	Yes	Yes
	$\{< 1w0; 0/\uparrow/- >, < 1w0; 1/\downarrow/- >\}$	Yes	Yes
CFwd	$< 0; 0w1/0/- >$	Yes	Yes
	$< 1; 0w1/0/- >$	Yes	Yes
	$< 0; 1w0/1/- >$	Yes	Yes
	$< 1; 1w0/1/- >$	Yes	Yes
CFrd	$< 0; 0w0/\uparrow/- >$	Yes	No
	$< 1; 0w0/\uparrow/- >$	Yes	No
	$< 0; 1w1/\downarrow/- >$	Yes	No
	$< 1; 1w1/\downarrow/- >$	Yes	No
CFdrd	$< 0; r0/\uparrow/1 >$	Yes	Yes
	$< 1; r0/\uparrow/1 >$	Yes	Yes
	$< 0; r1/\downarrow/0 >$	Yes	Yes
	$< 1; r1/\downarrow/0 >$	Yes	Yes
CFir	$< 0; r0/\uparrow/0 >$	Yes	Yes
	$< 1; r0/\uparrow/0 >$	Yes	No
	$< 0; r1/\downarrow/1 >$	Yes	Yes
	$< 1; r1/\downarrow/1 >$	Yes	Yes

Table 5.5: Fault Simulation Results for Inter-Word Dynamic Single-Cell Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
dRDF	$< 0w0r0/1/1 >$	Yes	No
	$< 1w1r1/0/0 >$	Yes	No
	$< 0w1r1/0/0 >$	Yes	Yes
	$< 1w0r0/1/1 >$	Yes	Yes
dDRDF	$< 0w0r0/1/0 >$	Yes	No
	$< 1w1r1/0/1 >$	Yes	No
	$< 0w1r1/0/1 >$	Yes	No
	$< 1w0r0/1/0 >$	Yes	No
dIRF	$< 0w0r0/0/1 >$	Yes	No
	$< 1w1r1/1/0 >$	Yes	No
	$< 0w1r1/1/0 >$	Yes	Yes
	$< 1w0r0/0/1 >$	Yes	Yes

5.3.2 Intra-word Faults

The fault simulation results for inter-word faults are shown in tables 5.8 and 5.9.

5.3.2.1 Discussion

Since the proposed algorithm is not targeted against all simulated faults of table 5.8 and 5.9, we divide these faults into *targeted* and *untargeted* faults and discuss each category separately. Of course, the targeted fault set of the proposed algorithm is different from that of the reference MemBIST. Therefore, in this classification, "targeted fault" means faults that are targeted by the proposed algorithm, and untargeted fault means the faults that are not targeted by the proposed algorithm.

Targeted Fault Coverage As mentioned in section 5.3, the proposed algorithm only targets the following intra-word faults:

- State Coupling Faults (CFsts)
- Idempotent Coupling Faults (CFids)
- Disturb Coupling Faults (CFdsts)

Table 5.6: Fault Simulation Results for Inter-Word Dynamic Coupling Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
dCFds	$< 0w0r0, 0/1/- >$	Yes	No
	$< 0w0r0, 1/0/- >$	Yes	No
	$< 1w1r1, 1/0/- >$	Yes	No
	$< 1w1r1, 0/1/- >$	Yes	No
	$< 0w1r1, 0/1/- >$	Yes	Yes
	$< 1w0r0, 1/0/- >$	Yes	Yes
	$< 0w1r1, 1/0/- >$	Yes	No
	$< 1w0r0, 0/1/- >$	Yes	No
dCFrd	$< 0, 0w0r0/1/1 >$	Yes	No
	$< 1, 0w0r0/1/1 >$	Yes	No
	$< 1, 1w1r1/0/0 >$	Yes	No
	$< 0, 1w1r1/0/0 >$	Yes	No
	$< 0, 0w1r1/0/0 >$	Yes	Yes
	$< 1, 0w1r1/0/0 >$	Yes	No
	$< 1, 1w0r0/1/1 >$	Yes	Yes
	$< 0, 1w0r0/1/1 >$	Yes	No
dCFdrd	$< 0, 0w0r0/1/0 >$	Yes	No
	$< 1, 0w0r0/1/0 >$	Yes	No
	$< 1, 1w1r1/0/1 >$	Yes	No
	$< 0, 1w1r1/0/1 >$	Yes	No
	$< 0, 0w1r1/0/1 >$	Yes	No
	$< 1, 0w1r1/0/1 >$	Yes	No
	$< 1, 1w0r0/1/0 >$	Yes	No
	$< 0, 1w0r0/1/0 >$	Yes	No
dCFir	$< 0, 0w0r0/0/1 >$	Yes	No
	$< 1, 0w0r0/0/1 >$	Yes	No
	$< 1, 1w1r1/1/0 >$	Yes	No
	$< 0, 1w1r1/1/0 >$	Yes	No
	$< 0, 0w1r1/1/0 >$	Yes	Yes
	$< 1, 0w1r1/1/0 >$	Yes	No
	$< 1, 1w0r0/0/1 >$	Yes	Yes
	$< 0, 1w0r0/0/1 >$	Yes	No

Table 5.7: Fault Coverage for Inter-Word Faults

FFM	Fault Coverage	
	Proposed MemBIST	Reference MemBIST
SF	100%	100%
TF	100%	100%
WDF	100%	0%
RDF	100%	100%
DRDF	100%	100%
IRF	100%	100%
CFst	100%	100%
CFds	100%	100%
CFid	100%	100%
CFin	100%	100%
CFtr	100%	100%
CFwd	100%	0%
CFrd	100%	100%
CFdrd	100%	75%
CFir	100%	100%
dRDF	100%	50%
dDRDF	100%	0%
dIRF	100%	50%
dCFds	100%	25%
dCFrd	100%	25%
dCFdrd	100%	0%
dCFir	100%	25%
DRFs	100%	-

Table 5.8: Fault Simulation Results for Intra-Word Static Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
CFst	$< 0; 0/1/- >$	Yes	Yes
	$< 0; 1/0/- >$	Yes	Yes
	$< 1; 0/1/- >$	Yes	Yes
	$< 1; 1/0/- >$	Yes	Yes
CFds	$< xwy; 0/\uparrow/- >$	Yes	Yes
	$< xwy; 1/\downarrow/- >$	Yes	Yes
	$< rx; 0/\uparrow/- >$	Yes	Yes
	$< rx; 1/\downarrow/- >$	Yes	Yes
CFid	$< 0w1; 0/\uparrow/- >$	Yes	Yes
	$< 0w1; 1/\downarrow/- >$	Yes	Yes
	$< 1w0; 0/\uparrow/- >$	Yes	Yes
	$< 1w0; 1/\downarrow/- >$	Yes	Yes
CFin	$\{< 0w1; 0/\uparrow/- >, < 0w1; 1/\downarrow/- >\}$	Yes	Yes
	$\{< 1w0; 0/\uparrow/- >, < 1w0; 1/\downarrow/- >\}$	Yes	Yes
CFtr	$< 0; 0w1/0/- >$	Yes	Yes
	$< 1; 0w1/0/- >$	Yes	Yes
	$< 0; 1w0/1/- >$	Yes	Yes
	$< 1; 1w0/1/- >$	Yes	Yes
CFwd	$< 0; 0w0/\uparrow/- >$	Yes	No
	$< 1; 0w0/\uparrow/- >$	Yes	No
	$< 0; 1w1/\downarrow/- >$	No	No
	$< 1; 1w1/\downarrow/- >$	Yes	No
CFrd	$< 0; r0/\uparrow/1 >$	Yes	Yes
	$< 1; r0/\uparrow/1 >$	Yes	Yes
	$< 0; r1/\downarrow/0 >$	Yes	Yes
	$< 1; r1/\downarrow/0 >$	Yes	Yes
CFdrd	$< 0; r0/\uparrow/0 >$	Yes	Yes
	$< 1; r0/\uparrow/0 >$	Yes	No
	$< 0; r1/\downarrow/1 >$	Yes	Yes
	$< 1; r1/\downarrow/1 >$	Yes	Yes
CFir	$< 0; r0/0/1 >$	Yes	Yes
	$< 1; r0/0/1 >$	Yes	Yes
	$< 0; r1/1/0 >$	Yes	Yes
	$< 1; r1/1/0 >$	Yes	Yes

Table 5.9: Fault Simulation Results for Intra-Word Dynamic Faults

FFM	FP	Detected By	
		Proposed MemBIST	Reference MemBIST
dCFds	$< 0w0r0, 0/1/- >$	No	No
	$< 0w0r0, 1/0/- >$	Yes	No
	$< 1w1r1, 1/0/- >$	Yes	No
	$< 1w1r1, 0/1/- >$	Yes	No
	$< 0w1r1, 0/1/- >$	Yes	Yes
	$< 1w0r0, 1/0/- >$	Yes	Yes
	$< 0w1r1, 1/0/- >$	Yes	No
	$< 1w0r0, 0/1/- >$	Yes	No
dCFrd	$< 0, 0w0r0/1/1 >$	Yes	No
	$< 1, 0w0r0/1/1 >$	Yes	No
	$< 1, 1w1r1/0/0 >$	Yes	No
	$< 0, 1w1r1/0/0 >$	No	No
	$< 0, 0w1r1/0/0 >$	Yes	Yes
	$< 1, 0w1r1/0/0 >$	Yes	No
	$< 1, 1w0r0/1/1 >$	Yes	Yes
	$< 0, 1w0r0/1/1 >$	Yes	No
dCFdrd	$< 0, 0w0r0/1/0 >$	Yes	No
	$< 1, 0w0r0/1/0 >$	Yes	No
	$< 1, 1w1r1/0/1 >$	Yes	No
	$< 0, 1w1r1/0/1 >$	No	No
	$< 0, 0w1r1/0/1 >$	Yes	No
	$< 1, 0w1r1/0/1 >$	Yes	No
	$< 1, 1w0r0/1/0 >$	Yes	No
	$< 0, 1w0r0/1/0 >$	Yes	No
dCFir	$< 0, 0w0r0/0/1 >$	Yes	No
	$< 1, 0w0r0/0/1 >$	Yes	No
	$< 1, 1w1r1/1/0 >$	Yes	No
	$< 0, 1w1r1/1/0 >$	No	No
	$< 0, 0w1r1/1/0 >$	Yes	Yes
	$< 1, 0w1r1/1/0 >$	Yes	No
	$< 1, 1w0r0/0/1 >$	Yes	Yes
	$< 0, 1w0r0/0/1 >$	Yes	No

Table 5.10: Fault Coverage for Targeted Intra-Word Faults

FFM	Fault Coverage	
	Proposed MemBIST	Reference MemBIST
CFst	100%	100%
CFid	100%	100%
CFdst	100%	100%

Table 5.11: Fault Coverage for Untargeted Intra-Word Faults

FFM	Fault Coverage	
	Proposed MemBIST	Reference MemBIST
CFin	100%	100%
CFtr	100%	100%
CFwd	75%	0%
CFrd	100%	100%
CFdrd	100%	75%
CFir	100%	100%
dCFds	87.5%	25%
dCFrd	87.5%	25%
dCFdrd	87.5%	0%
dCFir	87.5%	25%

In other words, the algorithm only targets a subset of intra-word static faults. No intra-word dynamic faults are targeted by this algorithm.

As shown in table 5.8, the implemented MemBIST manages to detect all targeted faults successfully. The fault coverage for targeted intra-word faults is shown in table 5.10.

Since the reference MemBIST is a bit-oriented test, it provides the same coverage for inter-word and intra-word faults of the same fault model. As shown in table 5.8, this coverage is 100% for all targeted fault models.

Untargeted Fault Coverage The fault coverage for untargeted FFMs is presented table 5.11.

It can be seen from table 5.11 that the implemented MemBIST provides a good fault coverage for untargeted faults as well. Again, the reference MemBIST provides a fault coverage similar to inter-word faults, as expected.

The reason why some fault primitives are not covered by the proposed algorithm is that there are no sensitizing sequences corresponding to these fault primitives in the intra-word march test.

For example, the following CFwd is not covered by the proposed MemBIST: $< 0; 1w1/ \downarrow /- >$. This is due to the fact that based on the DBOS used for crCFdsts (table 2.15), there are no operations which overwrite the victim cell with 1, while the aggressor cell is set to 1 and the old value for the victim cell is 0. The same reason is true for other uncovered fault primitives as well.

It should also be noted that since the designed MemBIST operates at the system's target clock frequency, the memory test will be run at-speed. This will enable the MemBIST to provide some coverage for timing-related faults, such as path delay faults, as well. However since our simulations were done in a functional level, the exact coverage for these faults was not determined.

5.3.3 Interpretation of the Results and Translation to Physical Defect Coverage

While the fault coverages estimated in this section can give an overview of how good each MemBIST will perform, they should not be interpreted as the actual defect coverages.

In order to estimate the actual physical defect coverage, a statistical distribution showing the probabilities of each functional fault model in the real world should be available. Only after that, the actual defect coverage can be estimated as follows:

$$Defect\,Coverage = \frac{\sum_i [P(FFM_i) \times Fault\,Coverage(FFM_i)]}{\sum P(FFM_i)}$$

In the above formula, $P(FFM_i)$ is the probability of functional fault model i occurring in reality and $Fault\,Coverage(FFM_i)$ is the fault coverage for that functional fault model provided by the algorithm.

This means that the fault primitives that have a lower probability of occurring in real life should have a lower weight in calculation of defect coverage.

For example, prior to introduction of DSM technologies, dynamic FFMs had a very low probability of happening in real life. That's why algorithms such as March LR were able to provide a very high defect coverage without targeting these fault models.

Unfortunately, at the time of this research, statistical distribution of fault models for Atmel's process was not available. This is due to the fact that the chips produced at Atmel typically have had a high yield and it has not been

necessary to identify the type of the faults that led to very few chips failing the MemBIST.

That said, considering the fact that the proposed MemBIST covers all the fault primitives covered by the reference MemBIST, it is safe to assume that the defect coverage of the proposed MemBIST will be at least as high as the reference MemBIST and any fault detected by the reference MemBIST will be detected by the proposed MemBIST as well.

5.4 Summary

In this chapter the implemented MemBIST was evaluated with respect to area overhead, performance, and fault coverage. In each evaluation, the designed MemBIST was compared to a bit-oriented March LR-based MemBIST currently in use in Atmel AVR family micro-controllers.

Our evaluation results show that the designed MemBIST provides a very high fault coverage for static and realistic dynamic faults and at the same time it performs 400% faster compared to the reference MemBIST. This all comes at the cost of a small area overhead which is neglectable when compared to the huge savings in testing cost.

Chapter 6

Summary and Conclusions

In this research, a state-of-the-art Memory Built-In Self Test unit was implemented and evaluated for use on Atmel AVR family micro-controllers.

We started by studying state-of-the-art memory fault models for deep sum-micron SRAM technologies. A comprehensive summary of these fault models was given in section 2.1. We then continued by studying the state-of-the-art memory fault-detection algorithms. A summary of up-to-date algorithms (at the time of this research) was given in section 2.2.

In chapter 3, these state-of-the-art MemBIST algorithms were compared, mostly based on their fault detection capabilities, and a combination of March AB and March LR was selected for implementation.

In chapter 4, the combined MemBIST algorithm was derived from March AB and March LR, and was extended to WOM testing using concurrent-restricted intra-word memory fault model. This algorithm was then extended to provide support for data retention testing and implemented in Verilog HDL in register-transfer level.

Finally, in chapter 5, the designed MemBIST was evaluated for area overhead, performance, and fault coverage. The bit-oriented March LR-based MemBIST, currently in use in Atmel AVR micro-controllers, was used as a reference MemBIST for benchmarking purposes. Our results show that the proposed MemBIST provides a significant performance improvement and at the same time provides a better fault coverage compared to the reference MemBIST. This comes at the price of a small area overhead compared to the reference MemBIST.

Note: This research was carried out in two separate steps. At first, and as

part of a semester project, a comprehensive literature review was performed, the results of which are presented in chapters 2 and 3. Afterwards, and as part of a master's thesis, the actual implementation and evaluation of the MemBIST was done, the results of which are presented in chapters 4 and 5. For the sake of completeness and readability, the results of both steps are joined in this report. The results of the semester project were also separately published as [1].

Bibliography

[1] H. Atashi, "Low-cost membist for embedded srams," 2012.

[2] M. Linder, A. Eder, K. Oberlaender, and M. Huch, "Effectiveness of memory test algortihms and analysis of fault distribution in srams: A case study based on industrial results," in *European Test Symposium, 2011. (ETS 2011). 16th IEEE*, May 2011.

[3] S. Hamdioui, Z. Al-Ars, and A. van de Goor, "Testing static and dynamic faults in random access memories," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, pp. 395 – 400, 2002.

[4] A. Bosio, S. Di Carlo, G. Di Natale, and P. Prinetto, "March ab, a state-of-the-art march test for realistic static linked faults and dynamic faults in srams," *Computers Digital Techniques, IET*, vol. 1, pp. 237 –245, may 2007.

[5] A. Bosio, P. Girard, L. Dilillo, S. Pravossoudovitch, and A. Virazel, *Advanced Test Methods for SRAMs: Effective Solutions for Dynamic Fault Detection in Nanoscaled Technologies.* Springer, 2009.

[6] S. Hamdioui, *Testing Multi-Port Memories: Theory and Practice.* PhD thesis, January 2001.

[7] A. van de Goor and Z. Al-Ars, "Functional memory faults: a formal notation and a taxonomy," in *VLSI Test Symposium, 2000. Proceedings. 18th IEEE*, pp. 281 –289, 2000.

[8] A. J. van de Goor, *Testing semiconductor memories: theory and practice.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[9] C. Papachristou and N. Sahgal, "An improved method for detecting functional faults in semiconductor random access memories," *Computers, IEEE Transactions on*, vol. C-34, pp. 110 –116, feb. 1985.

[10] A. van de Goor, G. Gaydadjiev, V. Mikitjuk, and V. Yarmolik, "March lr: a test for realistic linked faults," *VLSI Test Symposium, IEEE*, vol. 0, p. 272, 1996.

[11] A. J. van de Goor, *Testing semiconductor memories: theory and practice.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[12] A. van de Goor and I. Tlili, "A systematic method for modifying march tests for bit-oriented memories into tests for word-oriented memories," *Computers, IEEE Transactions on*, vol. 52, pp. 1320 – 1331, oct. 2003.

[13] M. Marinescu, "Simple and efficient algorithms for functional ram testing," in *ITC*, pp. 236–239, 1982.

[14] A. J. van de Goor, *Testing semiconductor memories: theory and practice.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[15] D. Suk and S. Reddy, "A march test for functional faults in semiconductor random access memories," *Computers, IEEE Transactions on*, vol. C-30, pp. 982 –985, dec. 1981.

[16] J. de Jonge and A. Smeulders, "Moving inversions test pattern is thorough, yet speedy," *Computer Design*, pp. 169 –173, may 1976.

[17] V. G. Mikitjuk, V. N. Yarmolik, and A. J. V. D. Goor, "Ram testing algorithms for detection multiple linked faults," 1996.

[18] S. Hamdioui, A. van de Goor, and M. Rodgers, "March ss: a test for all static simple ram faults," in *Memory Technology, Design and Testing, 2002. (MTDT 2002). Proceedings of the 2002 IEEE International Workshop on*, pp. 95 – 100, 2002.

[19] A. van de Goer and G. Gaydadjiev, "March u: a test for unlinked memory faults," *Circuits, Devices and Systems, IEE Proceedings* -, vol. 144, pp. 155 –160, jun 1997.

[20] S. Hamdioui and A. Van De Goor, "An experimental analysis of spot defects in srams: realistic fault models and tests," in *Test Symposium, 2000. (ATS 2000). Proceedings of the Ninth Asian*, pp. 131 –138, 2000.

[21] R. Dekker, F. Beenker, and L. Thijssen, "Fault modeling and test algorithm development for static random access memories," in *Test Conference, 1988. Proceedings. New Frontiers in Testing, International*, pp. 343 –352, sep 1988.

[22] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, and P. Prinetto, "March ab, march ab1: new march tests for unlinked dynamic memory faults," in *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pp. 8 pp. –841, nov. 2005.

[23] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan, "Dynamic read destructive fault in embedded-srams: analysis and march test solution," in *Test Symposium, 2004. ETS 2004. Proceedings. Ninth IEEE European*, pp. 140 – 145, may 2004.