Testing Computation-in-Memory Architectures Based on Emerging Memories

by

Surya Nagarajan

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday November 28, 2019 at 13:30

Student number:	4743385	
Thesis number:	Q&CE-CE-MS-2019-21	
Project duration:	December 1, 2018 – November 28, 2019	
Thesis committee:	Prof. dr. ir. S. Hamdioui	TU Delft, supervisor
	Dr. ir. M. Taouil,	TU Delft
	Dr. ir. T. G. R. M. van Leuken,	TU Delft
	Ir. M. C. R. Fieback	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Many alternative computer architectures that use emerging devices are under investigation to address the challenges current architectures and technologies face. Computation-in-memory (CIM) architectures are one among these alternative that tries to solve these challenges by performing computations in the memory structure as opposed to transferring the data to a central processing unit.One class of these CIM architectures employs memristive devices. These are non-volatile devices that store data as a resistance, and are highly compatible with traditional CMOS process. Many research centers and companies are prototyping such architectures. Efficient and high-quality test solutions are required for these architectures, which is the subject of this thesis.

This thesis presents a methodology for testing any CIM architecture, focusing on their memory and computation configurations, and applies this methodology to an existing CIM architecture as an example. The configurations are tested in the mentioned order for maximum fault coverage, while minimizing test development complexity. The testing method is structural rather than functional, thereby maximizing and guaranteeing fault coverage. To create accurate tests, device-aware testing is employed to model these defective devices. As a case study, the methodology is applied to scouting logic, a bit-wise logic CIM architecture that performs operations on data stored in memristors. Defects in the memory array as well as in the peripheral circuitry were injected and simulated to obtain realistic faults. The resultant fault analysis shows that there exist faults that are unique to the computation configuration and are not observed in the memory configuration. This implies that testing a CIM architecture only as a memory will lead to test escapes. Hence, the proposed test solution tests both the memory and computation configuration, and detects all faults.

Acknowledgements

I would like to thank Prof. Said Hamdioui, my supervising professor for this thesis for his constant monitoring and support. I learnt a good deal about getting presenting topics with clarity and asking the right questions, which would automatically steer the direction of research in any case. Second, I would like to thank my daily supervisor Moritz Fieback who guided me through this thesis. I enjoyed the numerous discussion sessions, which made the learning process quite fruitful. I would also like to thank Mehdi Tahoori and Rajendra Bishnoi for their feedback during the implementation of this work.

Next, I would like to thank my friends who have been with me through thick and thin. They had made my two years in Netherlands memorable, even though they might not have been here in person. In particular, I would like to thank Prajish, Anand and Gautham for being my roommates and making me feel less homesick on a daily basis. I would also like to thank Pradeep, Abhairaj, Manasa and Siddharth for being there at the 9th floor when everything went down. A special thanks to San and Prithvi for their constant support and love. Even though they live far away, M.A.S.S. and PROMICE had provided me constant support and I am grateful for knowing them.

I will always be grateful for my family, for without their constant support and motivation I do not see myself in the position I am.

Last but not least, I would like to thank caffeine, adrenaline, Kanye West and heavy metal music for existing.

Thank you!

Surya Nagarajan
Delft, November 2019

Contents

List of Figures

List of Tables

List of Abbreviations

BE Bottom Electrode
BEOL Back-End-of-Line
CBRAM Conductive Bridge random access memory
CIM Computation-in-memory
CMOS complementary metal-oxide-semiconductor
DRAM Dynamic Random Access Memory
DUT Device under Test
ETD Easy to Detect
FEOL Front-End-of-Line
FP Fault Primitive
HRS High Resistance state
HTD Hard to Detect
ILP Instruction level parallelism
LRS Low Resistance state
MOSFET Metal-oxide-semiconductor field-effect transistor
OxRAM Oxide random access memory
PCM Phase chage memory
ReRAM Resistive Random Access Memory
SRAM Static Random Access Memory
STT-MRAM Spin-transfer-torque magnetic random access memory
TE Top Electrode
$\mathbf{V_{DD}}$ Operation Voltage
VLSI Very large scale integration

1 Introduction

This chapter introduces the thesis. First the motivation of the thesis is discussed. Then, the state of the art in CIM architecture testing is discussed. This is followed by the contributions of the thesis. Finally the organisation of this thesis is briefed.

1.1. Motivation

Modern computing systems are a collection of different sub-systems that make up the system as a whole. These sub-systems are arranged in different layers of abstraction [40]. The abstractions range from the device-level components, such as metal–oxide–semiconductor field-effect transistors (MOSFET) which form the basic logic-gates (such as the NAND and NOR logic gates) to operating systems that the users interact with [40]. MOSFET technology has paved the way for the development of complementary metal-oxide-semiconductor (CMOS) fabrication process. This enabled the development of integrated circuits which have more than a million transistors in them, all while having low static power consumption and high noise immunity [63]. In all layers of abstraction, there has been constant research and development to create the most efficient computing system in terms of power, area and performance [33]. However, with the recent development in the fields of medical imaging, meteorological sciences and DNA sequencing, there has risen a need for high performance computers that are able to solve large problems with minimum effort possible [24] [18].

While the aforementioned problems can be solved with existing computing technologies, there are several shortcomings of present day computers and their architectures. These serve as hindrances for improving the computing performance to keep up with the ever growing demand for computation and data analysis [33]. These hindrances are commonly known as "performance limiting walls" in different performance aspects. These walls exist for both the underlying CMOS technology and the computer architecture that make up the computing system [26] [4]. CMOS technology faces the following walls:

1. **Reliability Wall** - As traditional CMOS based devices are scaled down to smaller nanoscale dimensions, the resulting production variations, operational variations, and defects cause higher failure rates and tend to reduces the lifetime of computing systems. This can be visualised in the bathtub diagram shown in Figure 1.1 where there is an increase in failure rate as the sizes of the devices are scaled down [4].
2. **Leakage Wall** - To keep in check the power density of the devices, it was necessary to reduce

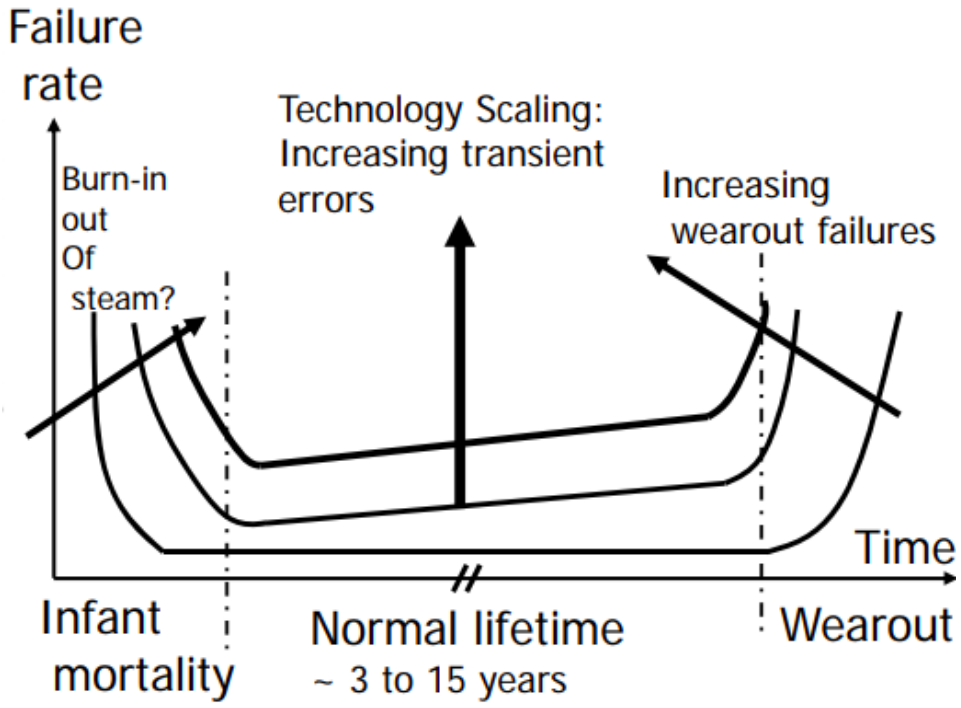


Figure 1.1: Reliability Wall [4]

the operating voltages of the devices. This enabled the integration of more and more devices in a small space. Unfortunately, reducing the operating voltage (V_{DD}) reduces the threshold voltage of the CMOS inverter as well. This has in-turn led to an increase in sub-threshold leakage in these devices [34]. This means that for smaller devices, the leakage power is larger than the active power in some cases. Figure 1.2 [1] shows the leakage power and the active power consumed as the technology size is reduced [34].

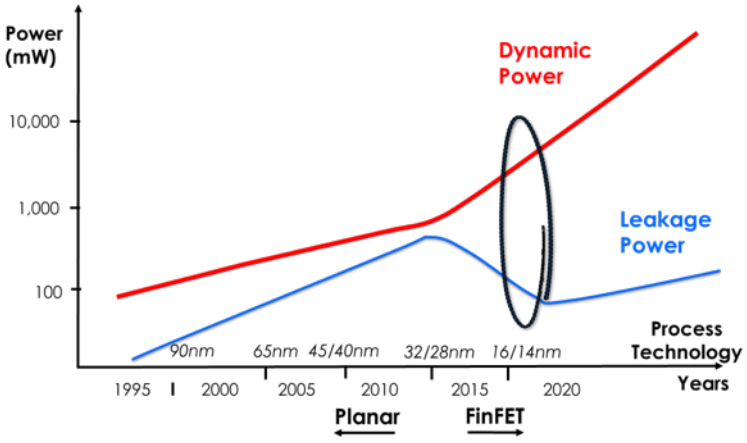


Figure 1.2: Leakage Wall [3]

3. **Cost Wall** - There is a clear increase in the production costs over the years for the CMOS devices as their geometric size is reduced to nanoscale [78], while there the cost per gate has been decreasing [2]. However, the decrease in cost per device more than compensated from 90nm technology that there was an overall decrease in the prices. There is however a considerable amount of increase in yield loss as a large number of transistors are integrated into a

single chip, which contributes to the increase in the overall cost of the device for smaller devices [55]. This has lead to the plateauing of the cost of production for smaller devices [36]. The cost trend with time is shown in the Figure 1.3 [36].

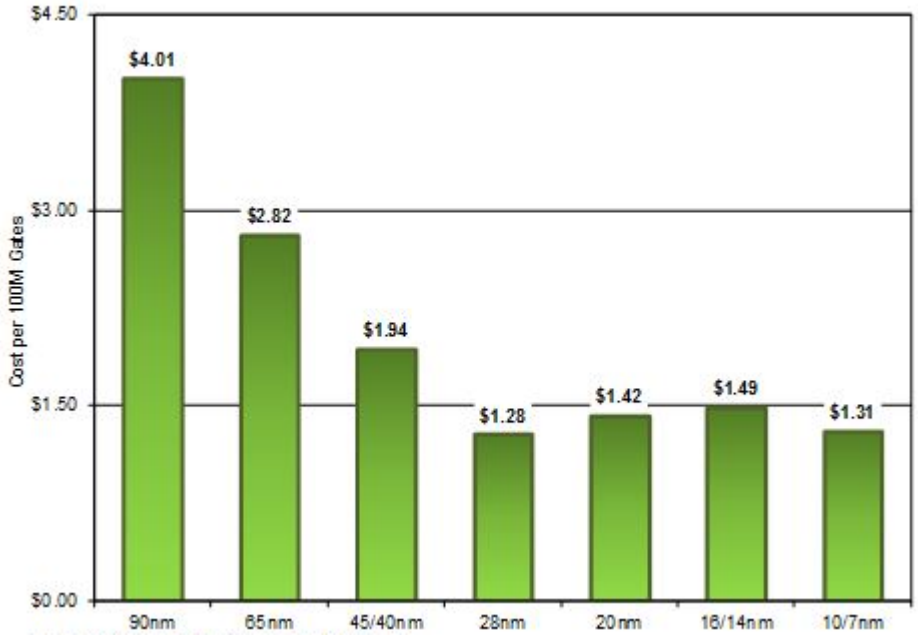


Figure 1.3: Cost Wall [36]

In addition to these limitations in the CMOS devices, there are also the following walls for the Von-Neumann architecture itself. These type of computer systems have distinct control unit, processing unit and memory that stores data and instructions. These kind of architectures are used predominantly in all computing systems nowadays [62][26].

1. **Memory Wall** - Some applications might require a large amount of data to be transferred between the memory and the processing unit in the Von-Neumann architecture. But the memory bandwidth has not increased as much as the processor throughput over the years, as shown in the Figure 1.4. Although cache memory mechanisms have been introduced to reduce the data access time, they are limited in size [33] [41]. This serves as a bottleneck for applications where there is a large amount of data transfer, which takes more time than computations. Hence overall throughput of the system has not increased [61].

2. **Power Wall** - While the performance of the processing unit has seen a significant increase over the years thanks to the increase in the frequency, the power consumption has also increased. This leads to overheating in the system [33]. The trend for the speed of the processor and the power consumed over the years is shown in Figure 1.5. The operation frequency of computing systems have reached a limit because of the available power.

3. **Instruction level parallelism (ILP) Wall** - While there is parallelism shown in the CPU level with the help of pipelining, and processor level with multi-threading and multi-core systems, the problems that need to be solved are not solvable with parallelism. This increases the power consumption of the computing system to solve the problems [33]. This stagnation can be seen in the Figure 1.5, where the instruction level parallelism has flat-lined.

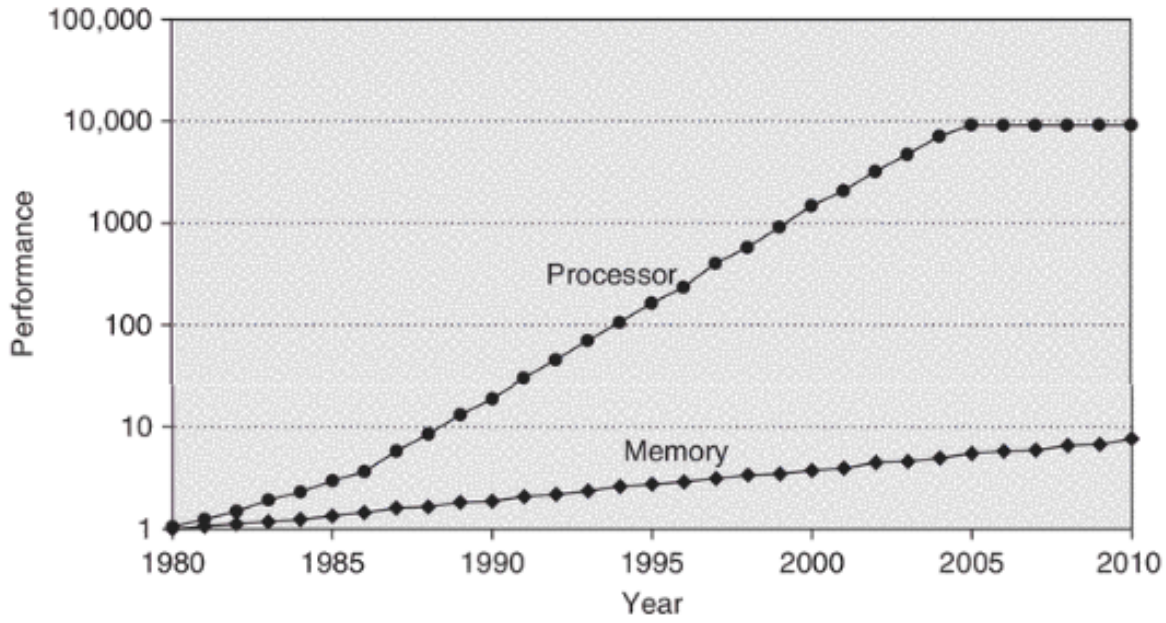


Figure 1.4: Memory Wall [33]

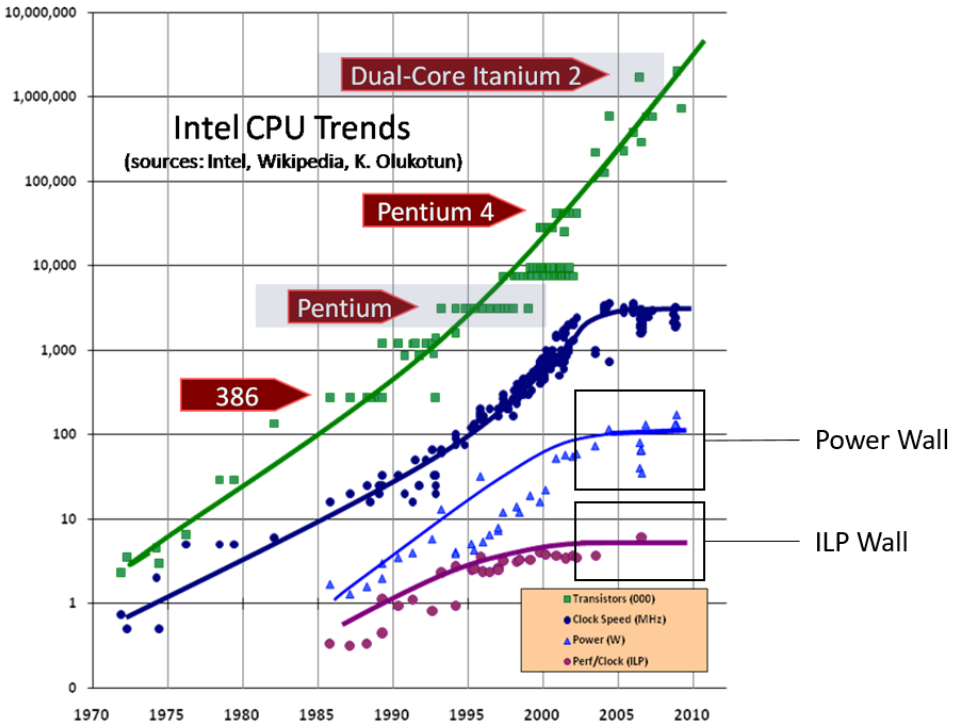


Figure 1.5: Leakage Wall [68]

The combination of these walls in modern computing systems has urged researchers to look towards novel methods of optimization, including modified architectures. Computation in memory (CIM) architectures have been developed as alternatives to traditional architectural solutions.

These CIM architectures work around the memory wall by performing logic operations in the memory, thus reducing memory transfers and thereby improving performance. With CIM architectures, massive parallelism is achieved because of the repeated structure possible with the memory cells that are present in these architectures. CIM architectures have been realised with DRAM [48] and SRAM [70] memory cells as well as employing emerging memory technology, as described next.

The walls in CMOS technology have also increased the interest in device level alternatives that have lower static leakage and lower cost. The realization of memristors devices on silicon has paved the way for development of new logic and architecture designs that have minimal static power dissipation, because of their non-volatile nature. Memristive devices are the physical implementations of memristors. Memristors were first conceptualized by Chua in 1971 as a missing two-terminal circuit element, along the lines of the resistor, inductor and capacitor [14]. The Memristor is characterized by the relation between the charge and the magnetic flux [14]. As the name suggests, one of the unique property of memristors is that they behave like a non-linear resistor with memory. The first practical memristor in modern times was developed by the research group of Stanley Williams at the Hewlett-Packard labs in 2008 [66]. It was realised as a metal/oxide/metal cross-point device with TiO_2 making up the oxide layer. Since then, there has been significant amount of research dedicated towards improvement of the devices. These include experimenting with different metal oxides, fine tuning physical properties for better yield and towards development of the circuits and architectures using these memristors that could solve real world problems. Many different kinds of memristive devices have been identified and produced, namely resistive RAM (ReRAM) [77], phase change memory (PCM) [76] and spin-transfer-torque magnetic RAM (STT-MRAM) [86].

These memristors, by virtue of their non-volatile nature, can be employed in creating alternative computer architectures which can be used to reduce, if not eliminate, the need for data movement between the memory and the computation unit. In our research, we concentrate on the CIM architectures based on memristive technologies because of the multiple advantages gained including low power consumption, non-volatility of the data in the memristor among others [26]. Since individual memristor devices are used for the computation of logic, massive amounts of parallelism can be achieved in the system, leading to improved efficiency.

1.2. Need for high quality test

The memristive devices themselves are not with their downside. Since research is still at a nascent stage with respect to the development of the memristor technology, many issues such as variations, faults etc., exist, that need to be tackled. The growth in popularity of memristor based CIM architectures has lead to huge developments in the area, such as improvement in reliability and faster operation timings because of selecting appropriate materials for producing memristive devices [83]. Memristive devices have a different production process as opposed to traditional CMOS devices, which makes them susceptible to defects that have not been studied yet. Studying, testing and detecting these defects and faults that arise in memristor-based system is essential in order to make sure that the end product can be considered consumer grade.

A characteristic feature of the CIM architectures based on memristive devices is their reliance of CMOS technology to control the operations of memristive devices. This includes operations such as driving the necessary voltages, writing logical values on the memristors, computing logic values based on the values in the memristors etc. There is also the added possibility of new defect that can occur in the memristor devices because of the interactions between the CMOS devices and memristors. These need to be accounted for during the testing of a memristor based circuit.

All these factors contribute towards potential failure of devices that are based on CIM architectures. This creates the need for high quality tests for these devices. These tests determine if the CIM

architecture based circuits are devoid of any known defects and if they are fit for consumer operation. In order to detect these defects, complete analysis of these architectures have to be made. Since these CIM architectures act as a memory *and* computing unit, tests have to be developed for these different modes of operations as well. Therefore, a structural test approach is needed for these devices. This forms the basis of this thesis, where we explore these architectures and device and create tests for the faults that might occur in them.

1.3. State of the art in CIM Testing

In the literature, there only exist limited works which revolves around the testing CIM architecture [15, 69]. In Tsai et al.s work, 8T cell SRAM based CIM structures are tested. However, it does not address the peripherals of the CIM architecture in the test sequences created, and it does not address tests for different CIM architectures based on other emerging memory devices [69]. While Emara et al.s work [15] tests for Memristor Ratioed Logic [44], it fails to expand on the same for other logic types. There are also works which test only the emerging memory technologies, based on which CIM architectures are developed. These works test ReRAM [31, 32] and STT-MRAM [56], thermally assisted switching MRAM (TAS-MRAM) [6] but only as standalone devices. The tests from these publications focus only on the defects and faults in a singular device and provide solutions for them. However, these test cannot, for example, take into account the interaction between two cells when performing a computing operation.

These publications have also been found to not take into account the physical behaviour of the defective devices while simulations, and use linear resistors as the defect model. This goes against the definition of these memristive device showing non-linear resistive characteristics. This leads to inaccurate fault models from the defect injections, which in turn lead to inaccurate tests [17]. A novel approach named the Device aware test approach models the physics of the defective device and is employed in this thesis to get accurate fault models and thereby better tests[16].

Through this thesis, the following goals are achieved: A methodology for CIM architectures testing is created that make use of emerging devices (RRAM, STT-MRAM, etc.) as well as traditional memory cells (DRAM, SRAM) in their core. The method of testing of these devices must make use of device aware testing to ensure the right modelling of the devices used. The method takes all operations, *both memory and computation*, into account, as well as the peripheral circuits when defining a test.

1.4. Contributions

The following contributions are made in this thesis:

1.4.1. Discussion of Test approach for CIM architectures

For testing of VLSI systems, there are two kinds of test approaches that are generally followed. These are *Functional* testing and *Structural* testing. This work systematically discusses the feasibility of applying these approaches to CIM architectures and concludes that structural testing is the more suitable method for testing CIM architectures.

1.4.2. Systematic Approach for Testing CIM Architectures

The functional operations in the CIM architectures are identified and the architecture itself is divided into two configurations based on the type of operation that is performed on the memristor devices:

- ***Memory configuration*** where one or more memory device is accessed for either reading the data stored in them or writing data into them.
- ***Computation configuration*** where the data in one or more memory devices are accessed and some logic or arithmetic operation is performed on them.

These configurations are studied and a systematic approach for testing all CIM architectures is proposed in this work.

1.4.3. Application of systemic approach to Scouting Logic

The test approach is verified on Scouting logic [80], a CIM architecture that is able to perform bitwise logic OR, AND and XOR operations on the data present in the memristor devices. The defects and the fault models in both memory and computation configuration in scouting logic are discussed. Tests are developed for faults observed in simulating the circuit after injecting defects.

1.4.4. Simulation Setup for Test Development for Scouting logic

In order to develop the tests for both configurations of scouting logic, the following steps were taken: Fault primitive modelling for both memory and computation configurations, setting up netlists in the Cadence Spectre language and injecting defects in them for simulations, automation of the simulation in a cluster using Mathworks MATLAB and bash scripting, extraction of the simulated data and identifying the tests to be performed.

1.4.5. Publications

Parts of this thesis have been used in the following publications, which have been included in the appendices:

1. A. Bosio, I. O'Connor, G. S. Rodrigues, F. K. Lima, E. I. Vatajelu, G. Di Natale, L. Anghel, *S. Nagarajan*, M. C. R. Fieback, S. Hamdioui "Rebooting Computing: The Challenges for Test and Reliability," 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Noordwijk, Netherlands, 2019, pp. 8138-8143.
2. S. Hamdioui, M. C. R. Fieback, *S. Nagarajan*, M. Taouil "Testing Computation-in-Memory Architectures Based on Emerging Memories", in 2019 *International Test Conference*, Washington D.C, U.S.A, 2019.
3. M. Fieback, *S. Nagarajan*, R. Bishnoi, M. Tahoori, M. Taouil, S. Hamdioui "Testing Scouting Logic-Based Computation-in-Memory Architectures", submitted for *European Test Symposium, Tallinn, Estonia, 2020.*

1.5. Organization

The rest of this work is organized into the following chapters.

Chapter 2 introduces and discusses memristive device based CIM architectures in detail. After this, the CIM architectures are classified based on the location of where computations reside. This is then followed by a detailed explanation of ReRAM cells.

Chapter 3 gives an introduction to testing electronics. Then, the different test methods employed in testing electronic devices are introduced. This is followed by detailed explanation of the

memory testing. Then it proposes the systematic approach that can be used to test CIM architectures.

Chapter 4 details the fault analysis of Scouting logic, describing the experimental setup and faults in each component.

Chapter 5 gives the tests that were developed for scouting logic, both in memory and computation configurations.

Chapter 6 concludes the work by summarizing each of the chapters, followed by discussions and suggestions for possible future work.

2

Memristor based CIM

This chapter introduces the memristor based CIM architectures and their classifications, and explains in detail the memristor cell. A general introduction to CIM architectures is presented in Section 2.1. Then, the classification of the CIM architectures is introduced in Section 2.2. Section 2.3 presents the memristor cell, which forms the building block of these CIM architectures.

2.1. CIM Introduction

As described earlier, traditional Von-Neumann architectures have to transfer data from the memory to the central processing unit (CPU). This is an overhead to the overall time taken for the computation of data. In addition, it also accounts for high energy consumption and naturally creates an upper limit to the amount of data that can be transferred. In order to reduce or at best eliminate this overhead, Computation-in-Memory (CIM) architectures were developed. A CIM architecture is one that actively tries to eliminate the cost of data transfer between the memory and computation unit and has the potential to perform computations with massive parallelization by processing the data in the memory. CIM architectures have been realised with DRAM and SRAM memories in the past [81] [48] [5] [70]. However with the commercial production of the memristor devices, architectures that make use of these devices have seen a rise in popularity [26].

How these CIM architectures can be part of the computing system is still under discussion [46]. The ideal case for the implementation would be to completely get rid of traditional computer architecture based sub-units such as an adder or multiplier and replace them with CIM architectures. But this cannot be achieved with the available technology because of the relatively low amount of development in the CIM architectures which can perform these complex tasks. One proposed way of integrating CIM architectures is to use them as accelerators with existing computing cores and memories as shown in Figure 2.1-a. The CIM accelerator accesses the data from the main memory and is controlled by the control units in the main CPU. Reduced memory access time between the CIM die and the memory would mean high amount of data traffic and hence higher rate of calculations in the CPU. The use case for these accelerators is easily illustrated when there is a need for repeated calculations or a loop in the program, that can be off-loaded to the accelerator. For example, a loop in a program that repeatedly reduces an equation based on simple calculations on a data stored in a fixed memory location. CIM architectures enable parallelism in the system, i.e. if there can be a part of the program that needs iterative computations, it can be processed in the CIM accelerator and the other parts of the program can be executed in the conventional CPU in the system, as shown in Figure 2.1-b. This can be achieved by modifying the instruction set to accommodate

macro-instructions in the system, which can be generated at the compiler level. This instruction can be sent from the CPU to the CIM core, where the instruction is executed. The instruction would point to the cell where an operation needs to take place and instruct which operation has to be performed on them. The final results are then returned to the main CPU to complete the program. The data produced in the intermediate steps of these iterations can be stored in the crossbar-array itself or in dedicated registers. While storing the intermediate results in the cross-bar can be resource efficient, they can also make the CIM accelerator to run at less than optimum speeds because of the overhead from the write operations used to store of data. We will now focus further on the internal architecture of the CIM accelerator.

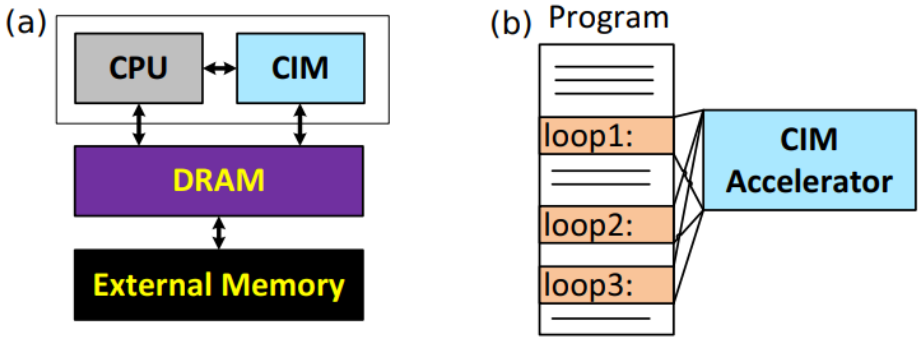


Figure 2.1: CIM architecture - overview [27]

CIM architectures using emerging memories tend to resemble an SRAM memory device in term of their architecture. In the core of the architecture, There is a very dense crossbar array, which is made of these memristor devices. In place of the SRAM cells in a regular SRAM architecture, memristive devices are present in CIM architectures. Owing to this similarity, we shall henceforth refer to the memristor devices in an array as being part of a memristor cells. The memristor cells are accessed with the help of peripheral circuits which include address decoders to identify the group of cells where operations are performed and the bit-line drivers which provide voltages to the cells to set the value of data in the memory and to enable computations in them. These structures would vary for each type of logic that is implemented in the CIM architecture. In addition to the drivers and decoders, there would also exist communication interfaces that would help with the commands to be sent to the CIM unit.

Due to the dual functionalities of the architecture,there would be changes in the circuits that build up the architecture in order to enable both memory operations and computation operations. The CIM architectures operate in two *configurations*:

- ***Memory configuration*** where one or more memory cells is accessed only for either reading the data that is stored in them or to write new values into these devices.
- ***Computation configuration*** where some logic or arithmetic operation is performed on the memory units separately or in combination with another memory cell. In these configurations, some architectures need changes in the driving circuits, the address decoders and the sensing circuits to facilitate the operations.

The Figure 2.2 shows these configurations and their operations. If we consider these memory and computation configurations as sets of operations, it can be seen that the memory configuration is a subset of the computation configuration. The results produced by computation operations are

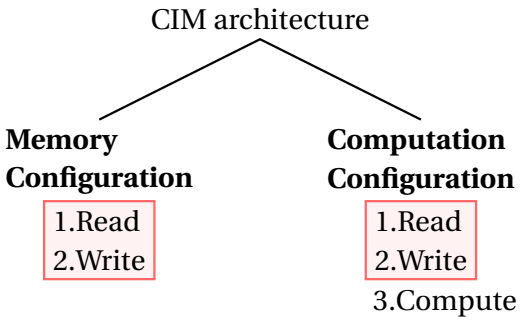


Figure 2.2: CIM configurations

either stored in the memory cells themselves or are available via peripherals as direct outputs in the form of electrical signals. This means that there needs to be a write operation in the former case and a read operation in the latter. Thus, it can be concluded that the computation configuration is a super-set of the memory configuration. This relation would play a role in the decision making for the test methodology explained in Chapter 3.

2.2. Classification of CIM

This section classifies the CIM architectures based on where the inputs for computation are obtained from and where data is stored post calculation. The CIM unit can store computation results in one of only two places (i) the crossbar array itself or (ii) the peripherals of the CIM crossbar array. From this the CIM architectures can be split in the following manner [46]:

- *CIM-Array (CIM-A)*: In CIM-A, the computation result is stored in the array after their production. The examples of such architectures include Snider [65], IMPLY [9], MAGIC [45] etc. These CIM-A architecture based cores may have significant changes in the memory array as opposed to regular memory structure in the crossbar. This is because some logic types such as MAGIC demand an array structure different from that of an SRAM.

- *CIM-Peripheral (CIM-P)*: in CIM-P, the computation result is produced within the peripheral based on electrical properties of one or more memory cells. For example the amount of current that passes through the device after an operation. These architectures thus have special circuits in their periphery that can perform logical operations. Example of these kinds of architectures include Pinatubo [50] and Scouting logic [80]. The memory cells still play a huge role in these architectures as they are involved in the generation of a base electrical value, which the peripherals use to perform the operation. For some logic types this would mean that a series of operations have to be performed in one or more of these memory cells in order to obtain a logic value.

While this distinction is made with the location of results, there exists a sub-division that can be made by studying where the inputs for these logical operations are stored. Logical and arithmetic operations generally operate on two input operands. This means that there has to be minimum one operand in the crossbar array. The second operand can be obtained from either the memory itself or in the form of a external voltage depending on the architecture. The former method has *resistive* inputs in the system and the latter input method has a *hybrid* set of inputs as it has both resistive and voltage inputs. This results in the classification of the CIM architecture mentioned above to have four sub-classes: CIM-Ar and CIM-Ah for architectures that store the results of the computation

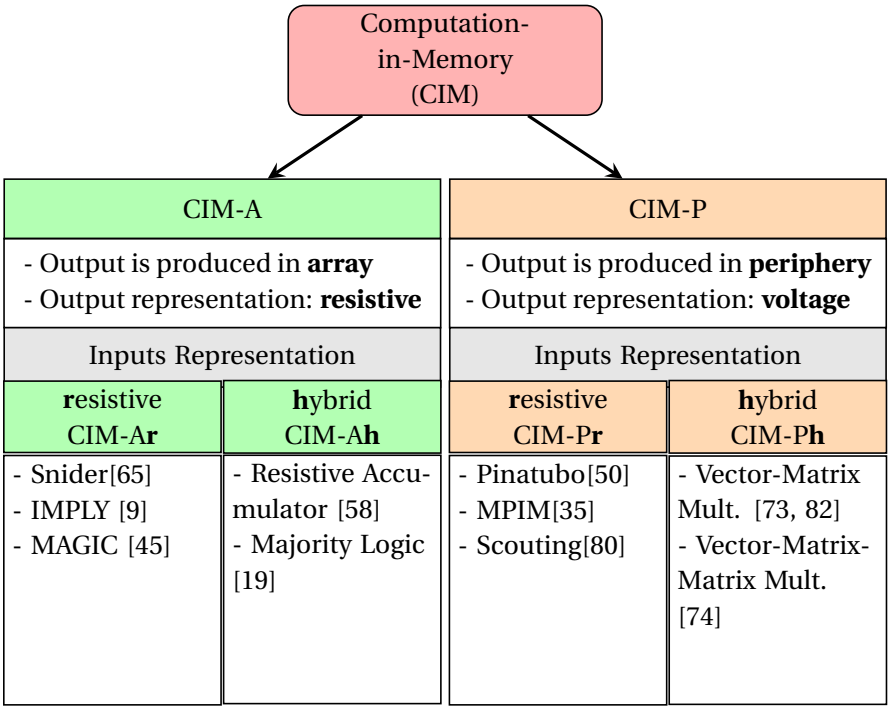


Figure 2.3: CIM Classification

in the array and have resistive and hybrid inputs respectively, and CIM-Pr and CIM-Ph where the results are obtained in the peripherals and have their inputs taking the resistive and hybrid forms respectively. The Figure 2.3 shows the various CIM logic architectures classified with the four sub-classes. This classification aids in the analysis of these architectures for test development, which will be discussed later in Section 3.

2.3. Memristor Cell

It is important to understand the operation of a memristive device before we can venture further into the test development. Here we discuss the three different types of memristive devices that have been fabricated and that can be used in the CIM architecture.

The three major non-volatile memoristor devices are STT-MRAM, PCRAM and ReRAM. All these memristor devices have common features: These are two terminal devices that act as non-volatile memories and they can be in one of many 'resistance states'. The states are determined by the resistance provided by these devices in the circuit. The resistance state boundary where there is minimal resistance to current in the circuit is the low resistance state (LRS) of the cell, also known as the 'On state'. Similarly, the state boundary where there is a large resistance is the high resistance state (HRS) and is known as the 'Off state'. LRS and HRS are dependant on the material of the device, process variations, noise factors and ambient conditions. The operation that switches the state of the cell from the HRS to the LRS is called as the "Set" operation, and the switching of the states in the opposite way is called as the "Reset" operation. The switching between these states is achieved with the help of some form of electrical stimulus, but the mechanism that is behind these operation varies between each of the different type of devices. We shall now look at the working principles behind each of these devices.

2.3.1. STT-MRAM

The spin-transfer-torque magnetic device is based on the magnetic torque switching that occurs in an atom because of electron spin under the influence of an electric field [8]. A STT-MRAM device consists of two dielectric materials, one with a magnetic moment of free polarity and other with a fixed polarity. These layers are separated by a thin tunneling insulation layer as shown in Figure 2.4. When a current of sufficient amplitude is applied to the device, a change in the angular momentum of the device occurs, that in turn changes the magnetisation of the free layer in the device. If the magnetisation of the free layer is in the same direction of the device, a high amount of current passes through the bulk, which is seen as a Low resistance state (LRS). If the direction is the opposite, then it is in the High resistance state (HRS).

STT-MRAM has high endurance, high scalability and high retention rate than other memristive devices [83]. However, STT-MRAM Devices have very low on resistance to off resistance ratio, which can make it difficult for state identification in some cases, where the on and off resistances are in the range $10-100\Omega$. Because of their large size, their package density is often lower than other forms of memristors [8]. The production of these devices have poor process compatibility with that of mainstream silicon CMOS devices, which creates a cost barrier due to the extra investment.

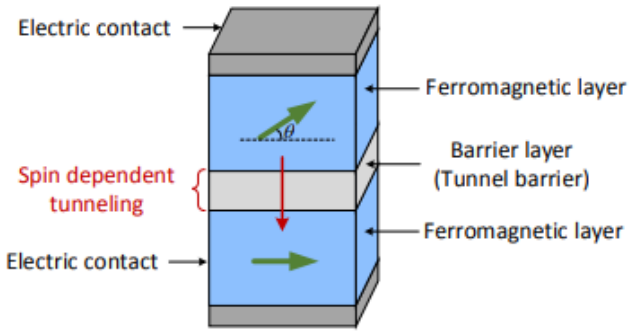


Figure 2.4: MTJ Cell structure [8]

2.3.2. PCRAM

Phase change devices is primarily made up of chalcogenide glass-based elements such as GeSbTe, Sb_2Ti_3 and AgInSbTe. An example of SbTe based phase change device is shown in the Figure 2.5 [76]. These materials have the property of changing their phase from crystalline, which represents low resistance state, to an amorphous form, which is the high resistance state. This phase change occurs with Joule heating, applied in the form of electric field or voltage. The transformation from LRS to HRS occurs when the material is heated above its melting point and then rapidly cooled down to room temperature. The "Set" operation is achieved by heating the material at a specific temperature between its critical point and melting point. The heating of the device is performed using a heating filament through which the current is passed [76].

PCRAM has a very large on resistance to off resistance ratio. This could mean that they could be used for multi-bit operations [83]. But the key challenges in PCRAM are large melt currents required to trigger the phase change, and the time it takes for the crystallization of the PCRAM material which can be more than 50 ns.

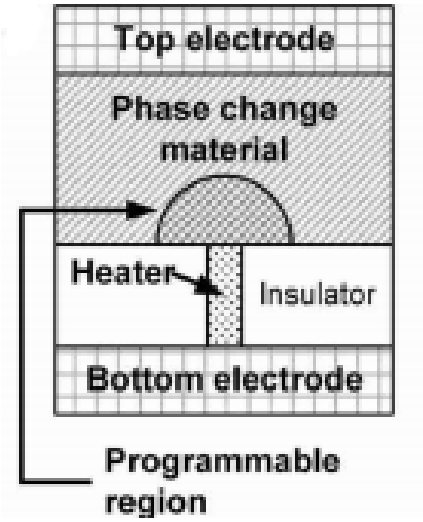


Figure 2.5: PCM Cell Structure [76]

2.3.3. ReRAM

The family of Resistive RAM consists of two different types of physical devices exhibiting different underlying physics. But, these devices operate in a similar fashion, have many common characteristics and a similar structure [12, 77, 83]. All ReRAM cells are made up of a metal top electrode and a metal bottom electrode which sandwich a switching medium where a *Conductive Filament* is formed. The two different types of ReRAM cells are explained as follows:

- **OxRAM**: The Oxide RAM consists of a top and bottom electrode with the bulk of the device made up of one of TiO_2 [75], ZnO [59], HfO_2[60], which act as the switching material. When there is certain amount of voltage applied across the device, a conductive filament made up of oxygen valencies is formed in the bulk that connects the top electrode and the bottom electrode.

- **CBRAM**: In the conductive bridge RAM, the conductive filament is made up of metal atoms which are formed by fast-diffusive Ag or Cu ions migrating into the solid-electrolyte [83]. These conductive filaments provide a path for current to flow, driving the device to a low resistance phase.

The Figure 2.6 shows the two types of ReRAM. The only difference between these cells in terms of the characteristics is that the ratio between the resistances at the on and off state which is higher in CBRAM devices. The endurance of OxRAM devices, however, are higher than that of CBRAM devices. For all comparison purposes with other types of memristor devices these two devices are grouped together as ReRAM.

The ReRAM devices have high compatibility with the CMOS process, low write time compared to other memristor devices, and low voltage requirement for read and write operations. While the ratio between on and off resistances are quite high, they also show intermediate resistance values with high precision, making them the perfect candidate for CIM architectures that have computations in the analog domain [47]. Because of these factors and the availability of well established models

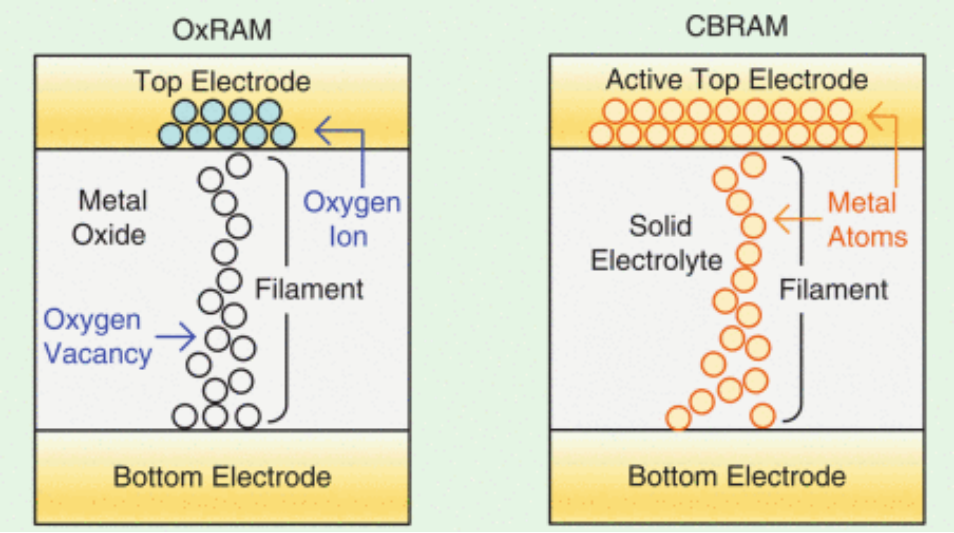


Figure 2.6: OxRAM and CBRAM [83]

for these ReRAMs we use these devices in the CIM architectures in our simulations. The operations and characteristics of the ReRAM devices are discussed below.

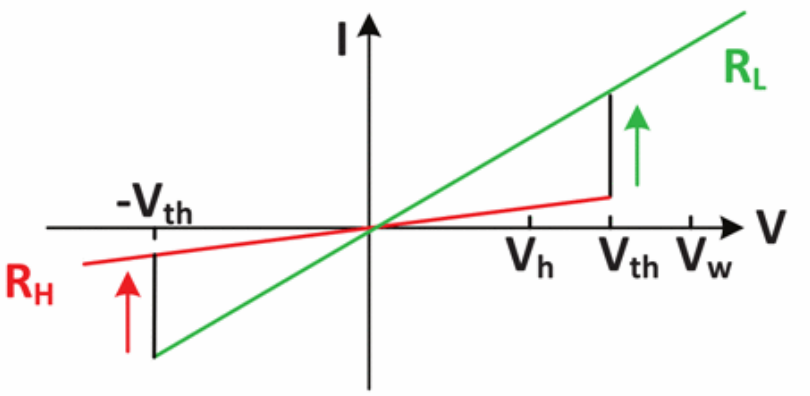


Figure 2.7: I-V Curve for ReRAM [79]

The I-V curve for ReRAM devices is shown in Figure 2.7. Here, it can be seen that when a positive voltage V_W is applied to the ReRAM device, the current increases slightly at first. At a threshold voltage (V_{th}), there is a sudden increase in the current that flows through the device. Here, there is enough energy in the switching material to break the bonds between the metal and oxygen ions, in the case of an OxRAM device. The free oxygen ions are attracted to the positively charged electrodes and start to form the conductive filament (CF). The CF is formed all the way from the positive electrode to the bottom electrode. This chain of ions facilitate the flow of the electrons through the device and hence the resistance of the device decreases, pushing it to the LRS. When the applied voltage is removed from the cell, the conductive filament stays intact, making the device *non-volatile*. The voltage at which the transformation takes place is called the set threshold voltage (V_{SET}). Similarly, when a negative voltage is applied to the device that is more negative than the reset threshold voltage (V_{RESET}), some ions in the conductive filament move back into the oxide. this reduces the size of the CF and breaks the chain of ions that connect the electrodes together, thus reverting back to the HRS. At the moment of fabrication of the device, there is no free oxide that can create the link through which the current can flow. Hence a process step where this CF is formed is required. This

step in the production of the device is called *CF Forming*. In this step, a voltage much larger V_{SET} is applied to the memristor device. This voltage, called the forming voltage (V_{form}) is dependant on the thickness of the switching material in the device. The forming of the cell happens along the grain boundaries that form in the poly-crystalline switching material [64].

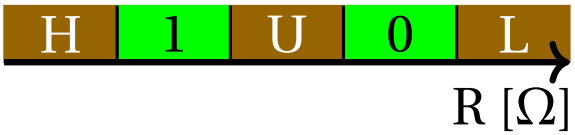


Figure 2.8: Resistance range in Memristors

The logical states in the ReRAM devices are shown in the Figure 2.8. The states are derived from the resistance of the device after a set or reset operation. As the resistance of the device is in a continuous range and varies with the write cycle, separate states for the logical values whose resistances do not lie within the exact limits of the HRS and LRS are defined. The state that corresponds to the HRS is the logical '0' and the state that corresponds to the LRS is the logical '1'. The state of the device when the resistance is less than that of the logical '1' is called as 'H'. Similarly, the state of the cell where the resistance is higher than that of the logical '0' is called as 'L'. The state that is between the LRS and HRS resistance range is the undefined state, denoted by 'U'. These states have been seen in defective ReRAMs, whose detection is an important part of the test development.

2.3.4. Memristor Array Architecture

The CIM architectures have the memristive devices in the crossbar. These devices can be arranged in different architectures. These are briefed as below:

- **1R configuration**: In the 1R configuration (Figure 2.9), each memristor cell has only the memristive device. It translates to the devices being connected end to end in the array.
- **1S1R**: Here (Figure 2.10)the cells consist of a selector that is connected to the bottom electrode. These selectors can include tunneling diodes,PT diodes etc.
- **1T1R**: In this configuration (Figure 2.11), the memristor device is connected on one end with a three terminal MOSFET device that can effectively control the current passing through the cell.

1T1R architecture is the preferred option because of the following reasons: the 1R configuration is susceptible to sneak path currents, which are caused in the array. Sneak paths are undesirable paths for the current which runs parallel to the intended path of the current in the array [87]. These sneak paths exist in the 1R configuration because of the lack of gating in the cells. There could be a path in the array that can have a lesser resistance that the intended access path in the array and can cause wrong addressing. This effect can be reduced with the usage of the 1S1R architecture for the array. This makes use of a rectifying diode or a bidirectional selector connected to the memristor device. But there are issues with maintaining stability with the cells, as there could be unstable oscillatory behaviour caused by the switching of the device or the selector. This is alleviated by the use of 1T1R architecture, which uses a transistor that is connected in series with the memristive device. This makes it a requirement to have three bit line addressing for the memory cells. Although this increases the addressing complexity and the area of the cell itself, there is a pay-off in having cells whose current can be controlled.

Figure 2.9: 1R memristor cell

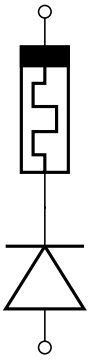


Figure 2.10: 1S1R memristor cell

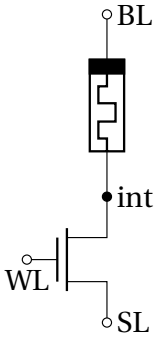


Figure 2.11: 1T1R memristor cell

The Figure 2.11 is a representation of a typical 1T1R cell. Here BL, WL and SL represent the bit line, word line and select line which enable the accessing and operations to be performed in the cell. When a positive voltage is applied to the WL, the NMOS gate opens, which allows current to pass from the BL through the device to the internal node (int in Figure 2.11) to the select line.

2.3.5. Production of ReRAM Devices

The production process of the ReRAM device in the 1T1R cell configuration is shown in Figure 2.12 and described here in detail. There are three steps that form the production process of the ReRAM device: Front-End-of-Line (FEOL), Back-End-of-Line (BEOL) and CF formation. The FEOL is the formation of the MOSFET device in the 1T1R cell structure, while the BEOL is the formation of the memristor itself. The final step is the forming of the CF, which is dependant on the oxide layer thickness at the end of the BEOL. We now look at each of these steps in detail.

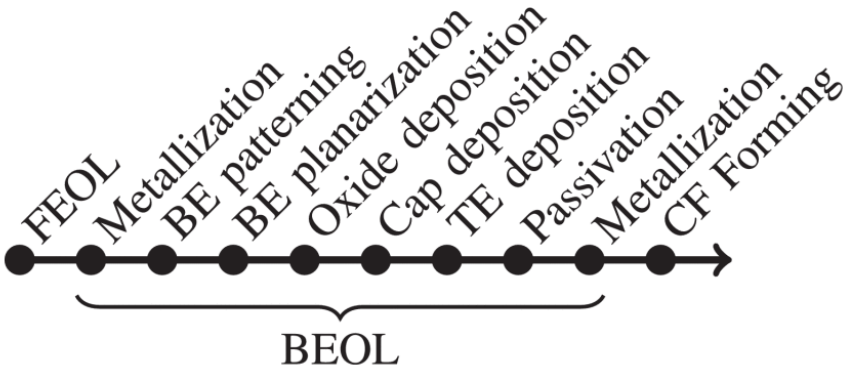


Figure 2.12: ReRAM Production Process

FEOL:

The standard production process that is taken up for the FEOL is called as "Gate first" process. the steps in the FEOL is given in the Table 2.1. These processes are preceded by the selection of the wafer to be used, Chemical-mechanical planarization and the cleaning of the wafer using plasma, dry-physical or super-critical fluid methods [39]. Details of these processes are out of scope and readers are encouraged to refer to [63]. After these processes, the ReRAM cell is fabricated in the BEOL processes which are explained in the next subsection.

Table 2.1: Front-End-of-Line Process [43]

S.No	Process steps
1	Shallow trench isolation (STI)
2	High-k gate deposition
3	Dual metal-gate deposition
4	Poly-silicon gate deposition
5	Poly-silicon/metal etch
6	Source/Drain formation
7	Salicide/contact etch stop deposition
8	First interlayer dielectric (ILD) film deposition
9	Polishing
10	Contact formation

BEOL:

After the FEOL process, the lower metal layers of the BEOL are manufactured. These are extended up to where the memristors are fabricated. First the BE has to be constructed. The BE is patterned and etched, followed by its planarization with chemical mechanical polishing. On top of the smoothed BE, the switching material is deposited uniformly, using atomic layer deposition to ensure uniformity [38]. Coating the oxygen layer is performed with care, as the larger number of grain boundaries that form between the poly-crystalline switching material, the wider the range of the resistance of the cell. Next the oxide is deposited with a capping layer made of TiN metal. This layer is inserted to deplete the O atoms in the switching material to act as an oxygen reservoir [77]. After this the TE is deposited and etched to remove excess depositions. With the ReRAM device built, it is isolated (passivation) and connected to the metal layers (metallization).

CF Forming

As mentioned, the grain boundaries between the poly crystalline material create the path for the CF to be formed. Thus the BEOL step plays an important role in this step of the production process, as the lesser grain boundaries in the switching layer, the higher the resistance in the device. The forming step can be seen as a dielectric soft breakdown [77]. After the CF forming process, the cell is in logical state '1' and can be used as a memory and logical unit. During the forming process, the forming current, I_{form} should be kept constant throughout the operation. In order to reduce the variability of the device after forming, different schemes are followed. These apply a sequence of pulses with varying pulse width and voltage [52]. The best way to form was found to employ trapezoidal waveforms that had their voltage cutoff when the expected switching behaviour is achieved [22].

These manufacturing processes are points where defects arise in the cell and in the peripherals. These defects cause the failure of CIM devices. The defects are discussed in Section 3.3.

3

CIM Architecture Testing

Chapter 2 gave an introduction to CIM architectures and memristor cells used in them. This chapter provides background on electronic testing. The first section gives an introduction to testing, followed by definitions of basic terms in electronics testing. Section 3.2 gives a brief about the different test methods available for electronics testing. The last section deals solely with testing of CIM based architectures.

3.1. Introduction

There are billions of transistors that are packed in a modern electronic design. This level of compactness can cause a lot of faulty devices if precision is not followed. They can sometimes fail even when care is taken to produce well fabricated devices. This is where the testing of the electronic devices comes into picture. The testing of these products after their production not only detects failed devices, after which diagnosis can identify the cause of the failure. It is of utmost importance to test components that would be used in critical applications such as military and automotive industry. Testing of these devices also plays a vital role in consumer goods as well, as they provides market advantages and overall customer satisfaction.

The general procedure followed in the testing of products is as follows: the manufactured goods are put under a test program, where devices are tested as pass or fail. The failed devices are diagnosed to find where the failure has occurred. When the failure is found, it would be used to fix either the design of the circuit or tweak the production method.

According to Micheal L. Bushnell, there are several roles taken up by testing process in electronic circuits[54]. these are listed as follows:

- **Detection**: The tests are used to detect the devices that do not work or are not up to the standards set by the company.
- **Diagnosis**: The devices that have failed are studied to identify the specific cause of the problem which resulted in the failure.
- **Device characterization**: It is imperative to understand the device operation and correct errors in the design and come up with a test procedure, which is performed during testing.
- **Failure mode analysis (FMA)**: Through the use of different test and instruments such as optical and electron microscopes, the cause of failure is detected and the process of production is rectified[54].

Before we look in detail the process of finding defective devices, a few definitions from [54] have to be introduced:

- **Device under test (DUT)**: The circuit system that needs to be verified to be fault free.
- **Defect**: A defect in an electronic system is the unintended difference between the implemented hardware and its intended design. These could include process defects, material defects, age defects and package defects.
- **Error**: A wrong output signal produced by a defective system is called an error. A defect causes an error.
- **Fault**: A representation of a defect at the abstracted function level is called a fault.

In order to create tests that would be able to detect the faults that are found in the devices, the internal circuits have to be studied beforehand and suitable tests for the devices have to be generated. Care has to be taken with these tests as the potential 'escape' of a fault can cause faulty devices to pass the test. These tests have to consider all possible defects, including parametric faults and if their functionality is as required by the end user. There are other concepts that are related to testing of components, but we concentrate on the testing methodologies for memories because CIM architectures are essentially memories which can do computations in them. For more information on testing electronic devices Micheal L. Bushnell's book [54] is recommended. In this thesis, the test developed is for the purpose of detection, to verify is the system developed matches the requirements set by the designer. The tests confirm the presence of a defect in the memory architecture.

3.2. Test Methods

This section deals with the different types of testing procedures that are taken up in order to verify the DUT for correctness.

3.2.1. Functional Testing

Functional testing makes use of functional tests that would check all possible inputs to the DUT to find if they produce the correct results. This means that for a circuit with n inputs a total of 2^n inputs are possible. This is a very costly operation, which would be impractical for high values of n. There is also the added disadvantage of not knowing where there actual defect is in the circuit, which makes the diagnosis process highly resource consuming. On top of all of this, there is no guarantee that these functional tests cover most of the faults that can be present in the DUT.

3.2.2. Structural testing

In Structural testing, the initial assumption is that the DUT has been manufactured according to specification. This means that unless there are defects that have come up during the manufacturing, the device would have no errors and would function properly. These tests are called structural because they rely on the specific structure of the circuits. With structural testing, it is possible to create algorithms that can test the devices for the specific faults are found to be in the system.

The Structural testing process can be streamlined into the following steps:

1. The netlist of the DUT is obtained and studied to identify potential defects. These defects are understood and adequately modelled.

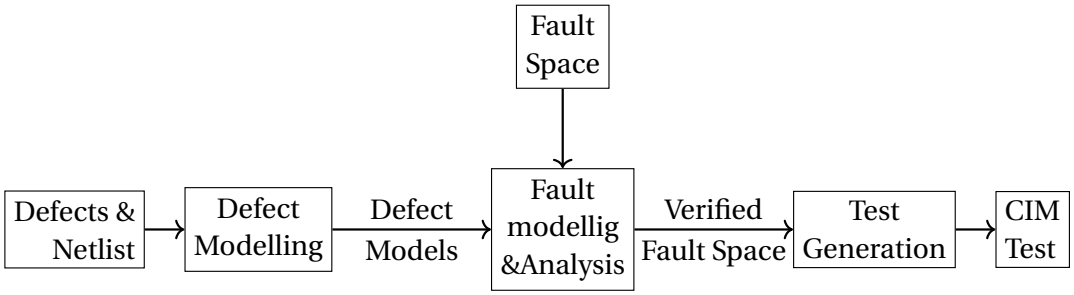


Figure 3.1: Structural testing approach

2. A set of possible faults that in the system based on individual components in the circuit is defined. This set is called the *Fault space*.
3. Defect models are then injected into to the netlist to observe the faulty behaviour caused by these defects.
4. All faulty behaviour observed is then combined to form fault models.
5. These fault models are verified against the fault space to identify missing faults, by injecting all possible defects in the system. This step is called as *fault analysis*.
6. Tests are generated for the realistic set of faults that occur in DUT.

The above process is shown in Figure 3.1. There might be cases where structural testing does not provide as much fault coverage as required by specification. In those cases, functional tests can be employed to improve the fault coverage[11]. In the next section, we deal with the special case of memory testing with memristive devices.

3.3. Memory Testing

As mentioned, the CIM architectures that use memristor cells of 1T1R architecture resemble an SRAM array architecture. It consists of peripheral that allow accessing of the cells to read and write data and also compute logic. Thus it is only natural that we discuss about the testing methodology for ReRAM devices and for the peripheral units that are made up of MOSFET technology devices. The Subsection 3.3.1 discusses about the failures that can occur in the memory unit, followed by Subsection 3.3.2 which discusses the faults that occur in the memory unit because of these defects. Then the tests for these fault types that are available are introduced.

3.3.1. Defects

Despite efforts to manufacture devices with high precision, there are always defects in the process. These can be in the form of parametric variations or physical irregularities that hinder the operation of the cell. Defects in the memory cell can occur in both the transistor component, memristor component and the metal interconnects between them. With the peripherals, it can occur in the transistors or the interconnects. In this section, the defects that occur in these locations are discussed.

Defects in MOSFET components (FEOL)

The study of defects that occur in the MOSFET transistors is a well established research field. The variation sources which cause defects were classified by Kuhn et al. as historical and emerging [42].

Historical sources are those which have been studied for a long time. These include patterning proximity effects, line roughness, polish variations and gate variations (in dielectric thickness, charge variation, traps etc.). Emerging sources are those which were seen to have lesser impact in the past, but have become of high significance lately. These include random dopant fluctuations, strain related variations etc. Long-term variation management is required to understand and keep these variation sources in check.

Defects in Memristor Device

Defects in the memristor device include those that occur in the FEOL, BEOL and CF forming. FEOL defects are similar to those discussed in Section 3.3.1. The Defects in each step discussed in the BEOL production phase described in Section 2.3.5 are discussed here:

- While the metal layer is extended to the layer where the memristor is to be constructed, standard defects that occur in CMOS production can be present. These were discussed in Section 3.3.1. These include wire opens or shorts, change in wire resistances due to line-width roughness.

- Next, in the construction of the BE, there can be variations with the area of the device due to line edge roughness. This causes lowering of the resistance of the device, which can lead to wrong logical states being stored. Smaller devices also have difficulties with the forming of CF [64].

- During the chemical-mechanical polishing of the device, there can be defects that are caused in the device due to the roughness of the BE. This can lead to variations in the device which results in faulty behaviour.

- In the switching material, the CF is formed along the grain boundaries of the poly-crystal. Since the distribution of the grain-boundaries varies with every device, the resistance range of the device varies as well.

- During the etching of the TE, there may be re-deposition of metal in the device, which could reduce resistance and forming voltage [57] [7].

CF Forming Defects

In the forming of the memristor device, two defects can occur [16]. First is the oxide breakdown, which can be compared to time dependant gate-oxide breakdown in MOSFET devices. This leads to the device to be permanently in the LRS state. If the defect is of a less severe form, then the device would be able to switch back to HRS, but the resistance range of the cell would have shifted below its nominal range. These devices are called as *overformed* devices[72]. The second type of defect that can occur is the failure of the CF forming step. This makes the device stuck in HRS.

Defects in peripherals

The ReRAM device is highly compatible with the MOSFET production process and operates in the same voltage region. Hence, for the peripheral devices that drive the ReRAM devices, MOSFET technology is employed. The defects in these devices have been studied extensively as mentioned in Section 3.3.1.

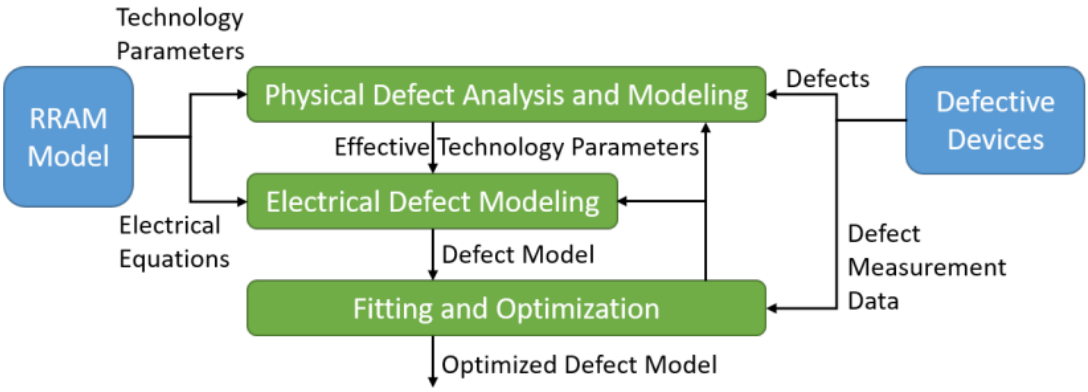


Figure 3.2: Device oriented modeling method

Defect modeling

The defects mentioned above are then modelled, as mentioned in Figure 3.1. Initially these defects were modelled as linear resistors that would help to realise opens and shorts in the memristor devices and in the peripheral circuits [37]. But since the memristor device is not linear, it is not advised to use linear resistor for modeling them. This resulted in the development of the *device-aware defect modeling* [16]. The Figure 3.2 explains the process of creating device-aware models with ReRAM as an example. First, the impact of the defect on the technology parameters are studied. for example, how a defect changes the length of the CF. Then these effects are fed to the electrical model to find their effect on the electrical parameters. The accuracy of the model is further tuned if real data is available. With this, an optimal electrical defect model is obtained.

3.3.2. Faults

From the defects that are modelled, the next step is to model the faults and analyse them. Fault modeling is based on two steps:(i)*Fault space* definition and (ii)fault space validation, using defect injection and circuit simulation. The fault space is a set of all possible faults that can occur in the device.This can be analytically generated given the knowledge of the operations that happen in the device. After the space is identified, the fault analysis takes place. Here sensitizing stimulus for each of these faults are developed and applied to an electrical model of the device to identify the response to the stimuli. Then a model that can accurately capture the defect in the cell is taken and the process is repeated. Based on the type of fault that is observed, the faults are classified as *Easy to detect* (ETD) or *Hard to detect* (HTD) faults. The classification helps in creating the tests for the system easier. This process can be classified into three steps, according to Fieback et al.et al., as follows [16]:

- Sensitize: Provide a set of inputs that will trigger the defective behaviour.
- Observe: The response of the defective behaviour in the system is observed.
- Classify: Here the faults are classified as ETD or HTD faults.

Fault Space

Since the peripheral circuit is specific to the CIM architecture, we do not go in detail about the same. Nonetheless, we introduce the fault space and faults for some common peripheral circuits. While

testing CIM architectures consisting of ReRAM cells, the fault space in the array is limited to single-cell faults. These faults can be sensitized by at most one operation on the cell or by performing more than one operation on the cell. The former is called as a *static* single-cell fault and latter is called as a *dynamic* single-cell fault. In case there exist a fault where there is more than one cell involved, then it is called a *coupling fault*. These faults are also considered in the fault space.

Before the simulation of the faults to find suitable tests, a method to describe the actions of these faults in the cell has to be defined. The most common way to describe a fault in a memory cell is by *Fault Primitives(FP* [28]. These fault primitives are compact notations that take the form ⟨S/F/R⟩. The components in the notation for ReRAM memories are explained below:

- S denotes the sensitizing sequence of the fault that is being tested. It could be a set of operations or the state of the cell which causes the fault. Mathematically it is represented as $S = x_0 O_1 x_1 \dots O_i x_i \dots O_n x_n$. Here x_i denotes the state of the cell, i.e., $x_i \in \{0,1\}$. O_i denotes the operation that takes place in the cell i.e, $O_i \in \{r, w\}$, where r and w indicate a read and write operation on the cell respectively. n is the number of operations that take place in the cell.
- F is the state of the cell after the operations in the sensitizing sequences denoted in S are performed. The values that can be in the cell were discussed in Section 2.3.3. Mathematically, this is represented as $F \in \{H, 1, U, 0, L\}$.
- R (Read output) denotes the output of a read operation, if the last operation in S is a read operation. The set of values that R takes is given by $R \in \{0, 1, ?, -\}$. Here '?' denotes a random read value in the sensing circuit. For example, this can happen when the sensing current is too close to the sense amplifiers' reference current. '−' denotes that the last operation in the sensitizing sequence S is not a read operation.

With the above, the fault space for the memory array can be defined. This is possible because the S, F and R elements in the notation give all the possible combinations of inputs and outputs that can occur in the memory cell. In case there are new operations that can be performed in the cell, they are added to the FPs, for example with computations. These added cases will be discussed later. Complex faults that involve multiple operations (i.e, *dynamic faults*) and faults that affect multiple cells (e.g., *coupling faults*) can be defined in a similar way by extending the appropriate part of the FP [54]. For example, a dynamic fault is given by the example ⟨0w0w0/0/−⟩, where the S component consists of consecutive operations, making it static. A coupling fault occurs between multiple cells, when a defect in one cell causes unwanted changes in the neighbouring cell [54].

The fault space for the peripherals is all the possible faults that might occur in them. This means that the fault space consists of the combination of all possible input operations and their effect on the system. For example, for an address decoder the fault space included all the possible faults that could occur in the addressing of the memories in the array. The faults in these peripheral units would be addressed in the section 3.3.2.

Fault Models

Based on the above fault primitives, the faults are defined for the memory cell. These are given in the Table 3.1, where all single-cell static FPs are listed with their names. The naming of the FPs follow the naming scheme given as:

$$FP = \{\text{read impact}\}\{\text{cell behaviour}\}\{\text{initial state}\}_{\{F\}} \tag{3.1}$$

In equation 3.1, *read impact* is conditional and is applicable only if a read operation is the sensitizing operation for the fault. This can be either an 'incorrect' (I) or 'random' (R) read impact. The cell behaviour gives which operation and the resulting fault effect in the cell. The operations are read (R) or write (W), and are extended when there is a need for additional operations. The fault effects are given as follows: (i) *Destructive*(D) where the cell has it's state changed when under an operation that is not supposed to change the state of the cell. (ii) *Transition* (T) where the cells do not undergo the transition that is required by the operation that is being performed on them. (iii) if both of the above can not describe the fault effect then the cell behaviour is left empty.

An example of these faults is given as follows:

$$RRDF0_1 = <0r0/1/?> \tag{3.2}$$

Equation 3.2 is a random read destructive fault that changes the state of the cell while giving a random read operation in the output.The case of dynamic faults is explained with an example:$\langle 1r1w0/L/-\rangle$ is described with as 2d-WTF1_L. The prefix *nd-* indicates that it is a dynamic fault, where *n* is the number of operations in *S*. The name of the fault is derived from the last operation performed. While this naming scheme works for all faults, they take an exception with state faults, where no operations are performed on the cells. They are described as:

$$FP = SP\{initial\}_{\{F\}} \tag{3.3}$$

For example, SF0_1 = $\langle 0/1/-\rangle$ is a state fault where the state of the cell flips from logic '0' to logic '1'.

Faults in Peripherals

In order to ensure complete coverage of tests, the faults in the peripherals have to be considered as well. These faults are dependant on the specific type of peripheral component used in the CIM architecture. These components inlcude voltage drivers, decoders for bitline, control registers and circuitry etc based on the type of CIM architecture used. However we concentrate mainly on the address decoders and sense amplifiers, two common peripherals in CIM architectures, as they are much more complex circuits that the others mentioned and have predefined fault model. The address decoder obtains the control signals and decodes it to select the particular memory cell and the sense amplifier deduces the value stored in the memory cell. We look at the faults that occur in them respectively as follows.

Address Decoder faults have been studied well [71]. The fault space includes all the faults that can occur while an addressing operation. The faults in address decoders can be classified as static or dynamic. *Static* faults are caused by completely broken interconnects in the lines that connect to the memristor cell or the line that connects the transistor to the memristor in the cell. These can also be caused by low ohmic bridges between the connections. These consists of four possible faults[71], as shown in Figure 3.3. These faults are given the following names:

- Fault 1 is *No-access*(AFna), where the address does not access the cell it is associated with.
- Fault 2 is *Multiple cells*(AFmc), where an address access more than one cell.
- Fault 3 is *Multiple addresses*(AFma), where a cell can be accessed by multiple addresses.
- Fault 4 is *Other cells*(AFoc), where an address accesses other cells in addition to the cell they are supposed to access.

Table 3.1: Complete single-cell static fault primitives.

#	S	F	R	FP notation	Name	#	S	F	R	FP notation	Name
1	0	1	-	⟨0/1/−⟩	$SF0_1$	27	0r0	1	0	⟨0r0/1/0⟩	$RDF0_1$
2	0	L	-	⟨0/L/−⟩	$SF0_L$	28	0r0	1	?	⟨0r0/1/?⟩	$RRDF0_1$
3	0	U	-	⟨0/U/−⟩	$SF0_U$	29	0r0	1	1	⟨0r0/1/1⟩	$IRDF0_1$
4	0	H	-	⟨0/H/−⟩	$SF0_H$	30	0r0	L	0	⟨0r0/L/0⟩	$RDF0_L$
5	1	0	-	⟨1/0/−⟩	$SF1_0$	31	0r0	L	?	⟨0r0/L/?⟩	$RRDF0_L$
6	1	L	-	⟨1/L/−⟩	$SF1_L$	32	0r0	L	1	⟨0r0/L/1⟩	$IRDF0_L$
7	1	U	-	⟨1/U/−⟩	$SF1_U$	33	0r0	U	0	⟨0r0/U/0⟩	$RDF0_U$
8	1	H	-	⟨1/H/−⟩	$SF1_H$	34	0r0	U	?	⟨0r0/U/?⟩	$RRDF0_U$
9	0w1	0	-	⟨0w1/0/−⟩	$WTF0_0$	35	0r0	U	1	⟨0r0/U/1⟩	$IRDF0_U$
10	0w1	L	-	⟨0w1/L/−⟩	$WTF0_L$	36	0r0	H	0	⟨0r0/H/0⟩	$RDF0_H$
11	0w1	U	-	⟨0w1/U/−⟩	$WTF0_U$	37	0r0	H	?	⟨0r0/H/?⟩	$RRDF0_H$
12	0w1	H	-	⟨0w1/H/−⟩	$WTF0_H$	38	0r0	H	1	⟨0r0/H/1⟩	$IRDF0_H$
13	1w0	1	-	⟨1w0/1/−⟩	$WTF1_1$	39	1r1	0	0	⟨1r1/0/0⟩	$IRDF1_0$
14	1w0	L	-	⟨1w0/L/−⟩	$WTF1_L$	40	1r1	0	?	⟨1r1/0/?⟩	$RRDF1_0$
15	1w0	U	-	⟨1w0/U/−⟩	$WTF1_U$	41	1r1	0	1	⟨1r1/0/1⟩	$RDF1_0$
16	1w0	H	-	⟨1w0/H/−⟩	$WTF1_H$	42	1r1	1	0	⟨1r1/1/0⟩	$IRF1_1$
17	0w0	1	-	⟨0w0/1/−⟩	$WDF0_1$	43	1r1	1	?	⟨1r1/1/?⟩	$RRF1_1$
18	0w0	L	-	⟨0w0/L/−⟩	$WDF0_L$	44	1r1	L	0	⟨1r1/L/0⟩	$IRDF1_L$
19	0w0	U	-	⟨0w0/U/−⟩	$WDF0_U$	45	1r1	L	?	⟨1r1/L/?⟩	$RRDF1_L$
20	0w0	H	-	⟨0w0/H/−⟩	$WDF0_H$	46	1r1	L	1	⟨1r1/L/1⟩	$RDF1_L$
21	1w1	0	-	⟨1w1/0/−⟩	$WDF1_0$	47	1r1	U	0	⟨1r1/U/0⟩	$IRDF1_U$
22	1w1	L	-	⟨1w1/L/−⟩	$WDF1_L$	48	1r1	U	?	⟨1r1/U/?⟩	$RRDF1_U$
23	1w1	U	-	⟨1w1/U/−⟩	$WDF1_U$	49	1r1	U	1	⟨1r1/U/1⟩	$RDF1_U$
24	1w1	H	-	⟨1w1/H/−⟩	$WDF1_H$	50	1r1	H	0	⟨1r1/H/0⟩	$IRDF1_H$
25	0r0	0	?	⟨0r0/0/?⟩	$RRF0_0$	51	1r1	H	?	⟨1r1/H/?⟩	$RRDF1_H$
26	0r0	0	1	⟨0r0/0/1⟩	$IRF0_0$	52	1r1	H	1	⟨1r1/H/1⟩	$RDF1_H$

Dynamic faults are caused by partial opens and shorts in the circuit, which translate to delays in the circuit. They can be either activation delays (ActD) where the activation of a line is delayed or deactivation delay (DeActD) where the deactivation of a line is delayed. These can also lead to one of the static faults based on the location and timing of the delay.

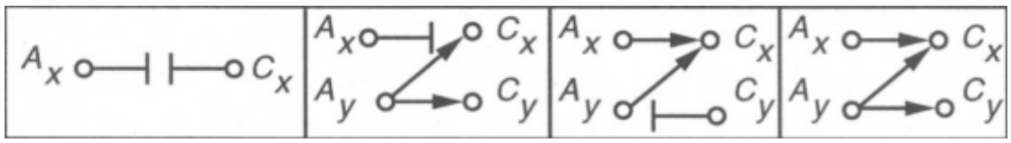


Figure 3.3: Address Decoder faults (A_X and A_Y are addresses and C_X and C_Y are cells that are addressed by them respectively.)

Sense amplifier faults have also been covered by researchers. These faults are again grouped as static and dynamic [71]. The *Static* faults are caused by opens and shorts in the circuit or low ohmic bridges. An example of these static faults is the SA stuck-at fault (SASF). Here, a sense amplifier always outputs the same result irrespective of the inputs fed to it. *Dynamic* faults are caused by partial opens and shorts in the circuit. They consist of two types:

1. *Unbalanced SA Fault*(USAF), where the sense amplifier tends to give an incorrect answer for equal inputs supplied to it because of the asymmetry in their internal circuits [84].

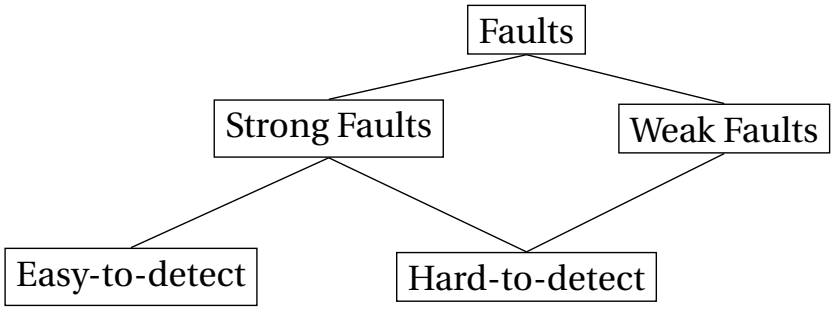


Figure 3.4: Fault Classification

2. *Slow SA Fault*(SSAF), where the signal timing of the sense amplifier is not synchronised with the clock of the system [20].

Fault Classification

Faults are classified as being either *strong* faults or *weak* faults. Strong faults are those which can be sensitized and detected using specific sensitizing sequences. These tend to result in functional faults in the system. On the other hand, weak faults do not result in functional faults, but rather in parametric faults such as change in voltage swing rate etc. These faults are not detected by sensitizing sequences and hence require special methods to have them detected. Based on the nature of faults and the effort that is needed to detect them, the faults are further classified as Easy-to-detect (ETD) and Hard-to-detect(HTD) faults, as shown in Figure 3.4. Following is the criteria for these faults to be identified as ETD or HTD:

- Detection of the ETD faults are guaranteed by the application of read/write or other common operations that are performed in the cell. for example, $\langle 1r1/0/0\rangle$ is a easy to detect fault.
- Detection of the HTD faults are not always guaranteed by these common operations, but there is a possibility that they can detect them. These need of additional tests or structures such as stress tests or DfTs to find them. For example, $\langle 1r1/U/?\rangle$ is a hard to detect fault.

Classification of these fault models into ETD and HTD eases test development processes. We now look at the different test approaches that available for each of these types of faults.

3.3.3. Testing

Tests for ETD Faults

For ETD Faults, the primary test is the March test[71]. These tests were defined by Suk and Reddy as finite operation sequences that detect functional faults in memories if only one type of fault is present [67]. The tests can be applied in either an increasing order of memory address (denoted by $\Uparrow$), a decreasing order (denoted by $\Downarrow$) or it may be irrelevant ($\Updownarrow$). The set of operations that are performed in a cell before it can be performed in the next cell is called a *march element* [54]. These are distinguished by semicolons separating them, with each march element enclosed in parenthesis. The entire march sequence is enclosed in braces. for example, the MATS+ march test [71] is written as

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

Here there are three march elements: M0 :⇳ (w0), M1 :⇑ (r0,w1), and M2 :⇓ (r1,w0). These elements are performed in the cell before proceeding to the next cell. The Table 3.2 gives the notations used in a match test. The march test is the preferred method for testing because of its O(n) complexity (where n is the number of memory bits), regularity and symmetry [54].

Table 3.2: March test notations

Notation	Description
r	Memory action: A read operation.
w	Memory action: A write operation.
r0	Memory action: Read '0' from the memory location.
r1	Memory action: Read '1' from the memory location.
w0	Memory action: Write '0' to the memory location.
w1	Memory action: Write '0' to the memory location.
↑	Write '1' to a cell containing '0' or the cell has a rising transition.
↓	Write '0' to a cell containing '1' or the cell has a falling transition.
↕	Complement the contents in the cell.
⇑	Increasing memory addressing order.
⇓	Decreasing memory addressing order.
⇳	Addressing order does not matter.
⇒	Write value x to a cell already containing x
∀	Denotes any memory write operation
< ... >	Denotes a particular fault, described by ...

Several extensive March tests have been described to test the memory elements which are made up of the memristor devices [13][51][53]. While these march tests cover faults for the memory operations, they cannot be used as such for CIM architectures as there will be faults that would not be covered by these tests.

Tests for HTD Faults

In order to test for HTD faults, the primary methods that are available are as follows:

- Some HTD faults can be identified by march tests, but these would require the addition of stresses[20]. These stresses can be algorithmic (changed initial states of the cell, fast addressing etc.) or environmental (voltage and speed variations) in nature.

- In case these tests are not enough, then *Design for testability* (DfT) is employed. Here a special circuit is added to the architecture in order to identify defective cells. For example Hamdioui et al. have created a scheme where they have exploited the access time duration and supply voltage level of the ReRAM cells in order to detect unique faults where the device is in the undefined 'U' state[25].

With these tests and methods, it is possible to get the desired amount of fault coverage with memory architectures that are based on ReRAM memories. But it is unknown if these methods can be used for ReRAM based CIM architectures, where the cells are also subjected to a new set of operations, based on the logic type used. Hence it is required to investigate the testing methodology for the CIM architectures. This is presented in Chapter 4.

3.4. CIM test methodology

A CIM core operates in one of the two configurations given below:

1. **Memory Configuration**, where the CIM architecture can only read and write data to the memory devices.
2. **Computation Configuration**, where the data stored in one or more memory device is used to perform some logic or arithmetic operation.

The hardware in the computing configuration of a CIM architecture can be seen as a modification of the hardware in its memory configuration. This leaves some gap in the defects covered if only the memory configuration is to be tested. This rules out common tests of regular memories to be applied to these CIM architectures. The computation configuration alone should not be tested as it can leave out faults in the components which arise only in the memory configuration, but do not affect the computation action. Hence it is required that both memory and computation configurations are tested for these CIM architectures.

As memory configuration is a subset of computing configuration as explained in Section 2.1, it has to be tested first. Testing computing configuration first can miss faults that can be detected easily in the memory configuration, which has a lower operational complexity. The tests performed are named *Memory configuration tests* and *Computation configuration tests*. The test development in these configurations is explained below:

3.4.1. Testing in Memory Configuration

Here, only the hardware used to perform memory operations are enabled. Testing in this configuration depends on the type of memory device used. If well established devices such as DRAM and SRAM are used, then test development is easier. It becomes difficult if emerging devices such as ReRAM, PCMRAM and STT-MRAM are used, as their operation model is new and have to be studied. The models used to describe these devices have not always been accurate as found by [17]. For these kinds of devices, device aware testing has to be employed to accurately model the defects in the memory device [17].

3.4.2. Testing in Computation Configuration

Here, testing strongly depends on the hardware that enables computation in the CIM device. This varies with the underlying logic design that is employed in each of these architectures. In order to test for the computation configuration, we need to first identify the hardware that forms the computation configuration. After these changes are studied, structural testing approach is used to find the faults in the device and thus develop tests in this configuration. The next section examines the computation configuration changes in Scouting logic based CIM architecture, a CIM-P architecture.

3.4.3. Scouting Logic

Scouting logic is a bit-wise logic that is able to perform OR, AND, and XOR operations. It performs them by modifying the read operation [80]. For a memristor device, the read operation is performed by applying a voltage V_r across its terminals and comparing the resulting current(I_{in}) to the reference current(I_{ref}) in the sense amplifier. If I_{in} is greater than I_{ref}, the output of the sense amplifier becomes V_{dd} (logic 1). if the opposite is true, then logic 0 is obtained. The value of the reference I_{ref} is fixed during the designing of the circuit, based on the resistance of the memristive device used and consequently the resulting I_{in} obtained by applying a voltage.

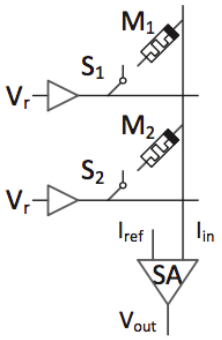


Figure 3.5: Memory array setup [80]

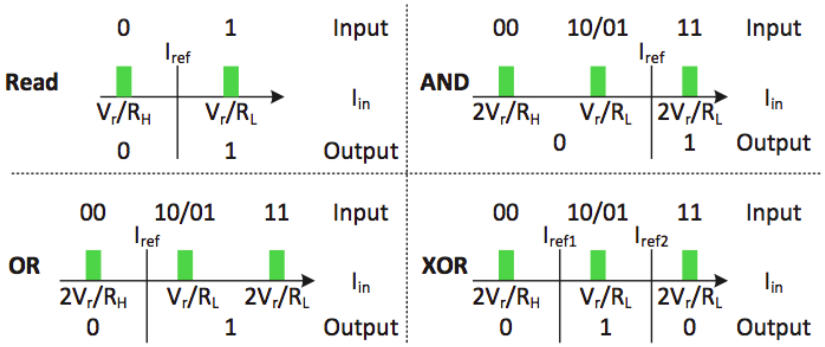


Figure 3.6: References of primitive operations [80]

Logic operations in scouting logic architecture are realized by extending this idea to reading two memristive cells (M1 and M2) in the same bit line as shown in Figure 3.5 [80]. These cell have a equivalent resistance of (M1||M2). This resistance determines the input current (I_{in}) to the sense amplifier when a voltage of V_{r} is passed through them. As the equivalent resistance is limited to one of three values possible ($R_{\text{L}}/2$, $R_{\text{H}}/2$ and $R_{\text{L}}||R_{\text{H}} \approx R_{\text{L}}$), the input current is also limited to three possible values. By adjusting the value of I_{ref}, different logic gates can be realised as shown in Figure 3.6 [80]. For logic AND operation, the reference current should be between $V_{\text{r}}/R_{\text{L}}$ and $2V_{\text{r}}/R_{\text{L}}$. Logic OR is realized by setting the reference current between $2V_{\text{r}}/R_{\text{H}}$ and $V_{\text{r}}/R_{\text{L}}$. Finally, logic XOR is realized by setting two reference currents, one between $2V_{\text{r}}/R_{\text{L}}$ and $V_{\text{r}}/R_{\text{L}}$, and the other between $V_{\text{r}}/R_{\text{L}}$ and $2V_{\text{r}}/R_{\text{L}}$.

Here the changes between the memory and computation configurations are in the memory array and in the peripheral circuits, namely the sense amplifier and the address decoder circuits. Two cells have to accessed at the same time in order to enable these logical operations. Multiple cell access is a unique configuration change from the memory configuration and can cause errors in the memory array. The sense amplifier is also modified to change its reference current when it receives an instruction. An address decoder that is similar to a multi-port memory address decoder is in place to access two cells simultaneously. This is different than a regular memory address decoder.

With these examples, we summarize the testing methodology for CIM architectures. First the CIM architecture in consideration has to be studied to distinguish between its memory configuration and computation configuration. Then, the memory configuration is tested using structural testing followed by analysis of the computation configuration. In structural testing, the defects in the configuration are identified and faults are modelled accordingly. Then the faults are simulated and their occurrence is validated. Then appropriate tests are then devised for these configurations. We apply this methodology to scouting logic in Chapter 4.

4

Defect and Fault modelling for Scouting Logic based CIM Architectures

This chapter deals with fault analysis of the two configurations in Scouting logic, a CIM-Pr architecture based logic. This serves as an example of applying the proposed methodology to a CIM architecture. Section 4.1 gives a detailed account of the experimental setup for the scouting logic. This is followed by examining the faults in the memory configuration in Section 4.3.1. This followed by examining the faults in the computation configuration in Section 4.3.2.

Scouting logic based CIM architecture is tested using structural testing method. The steps in structural testing was explained earlier and are repeated here again for clarity. The Figure 4.1 gives the block diagram for the flow of structural testing. This chapter deals with defect modeling and netlist, fault modeling, fault space and fault analysis for scouting logic based architecture. In the next chapter, the tests are developed for these faults found to improve the defect coverage.

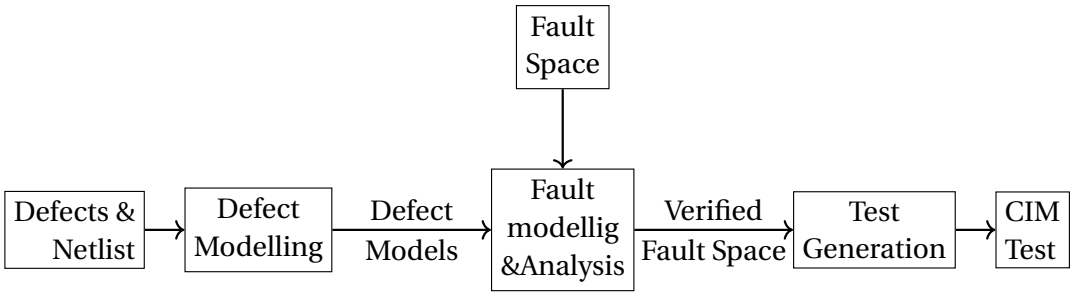


Figure 4.1: Structural testing approach

We start with explaining the netlist in Section 4.1. This is followed by the defect modeling in memory and computation configurations. Finally, fault analysis in memory and computation configurations are discussed.

4.1. Circuit Setup

The architecture for scouting logic resembles that of a regular ReRAM design, with modifications in the address decoder and the sense amplifier. This architecture is shown in Figure4.2 [80].

The memory array are arranged in words of consisting of three memory cells. These are accessed with the help of word line (WL), bit line (BL) and select line (SL), as shown in Figure 4.3. The words

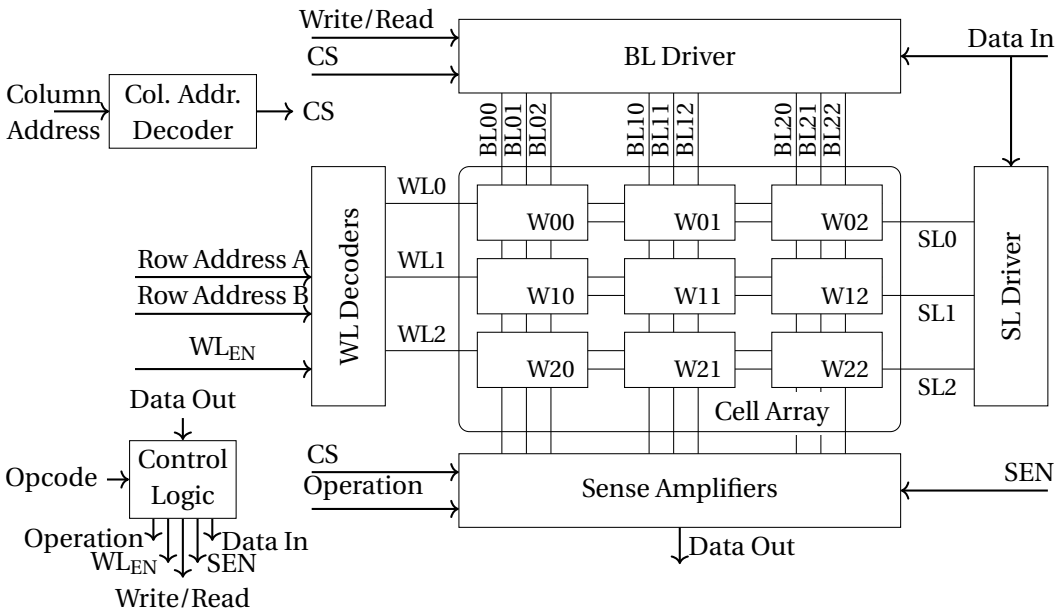


Figure 4.2: Simulation Architecture

are selected with the help of WL decoders, BL driver and SL driver. The array used in our simulations consist of 9 words, which are arranged in a 3x3 manner. This means that there are 27 individual cells that are present in the array, with 9 words in each row. The defects for the simulation are injected individually in the 26th cell of the array, when counted row-wise.

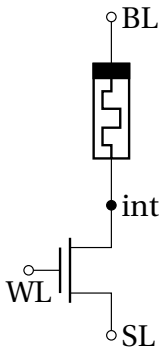


Figure 4.3: 1T1R

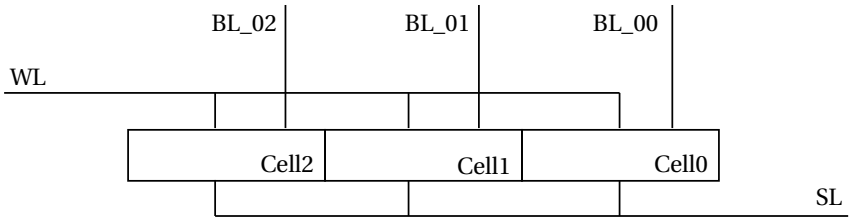


Figure 4.4: Word in array

The column decoder drives the column select (CS) line. This line is used in the BL driver to select the corresponding column, with the RESET signal and Datain signals, as shown in Figure 4.5. The Datain signals dictate the value to be written to the words. The waveforms for different levels of these signal and the corresponding output for the Bit line is shown in Truth table 4.1. In order to

enable the buffer to the bit line, the write enable line has to be activated with a delay. This is shown in the waveform in Figure 4.6.

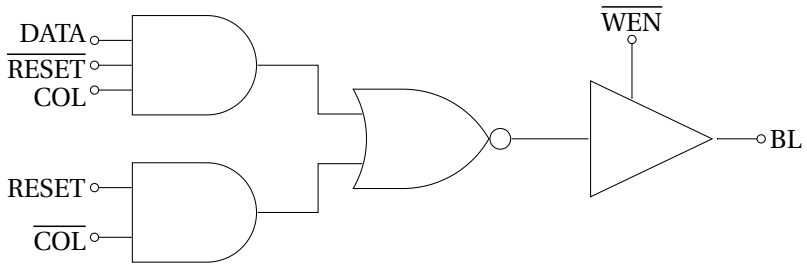


Figure 4.5: Bitline Decoder

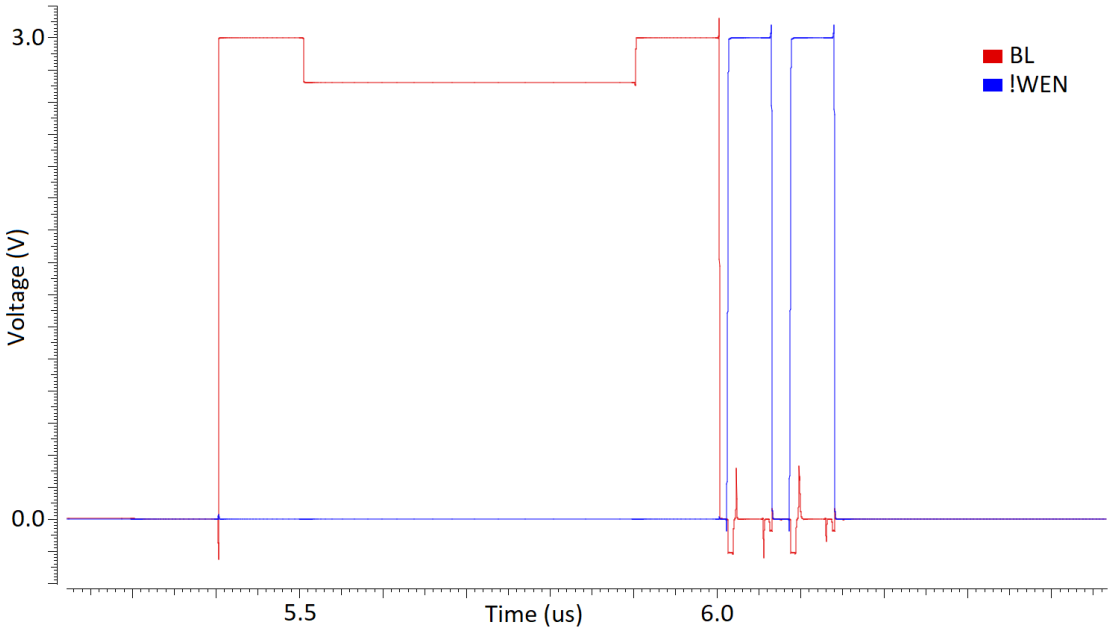


Figure 4.6: Bitline Decoder - Waveforms

The select line is activated by the WLInput line, in combination with the RESET signal. How the signal WLInput is generated is explained in the next paragraph. SL is driven to '0' when the cell is set and while reading, and to '1' when the cell has to be reset. This is shown in the Figure 4.8, where the input signals and the corresponding output signal are shown, while Figure 4.7 shows the circuit diagram for the SL driver.

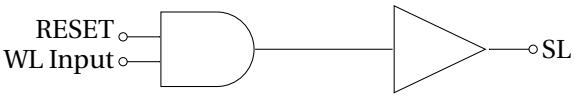


Figure 4.7: Select Line Driver

The peripherals and array components mentioned above do not differ for both memory and computation configurations. However, in order to perform computation, there are changes made in the address decoder and sense amplifier. The address decoders are shown in Figure 4.9. The circuit

DATA	RESET	COL	$\overline{WEN}$	BL
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Table 4.1: Truth Table - Bitline Decoder

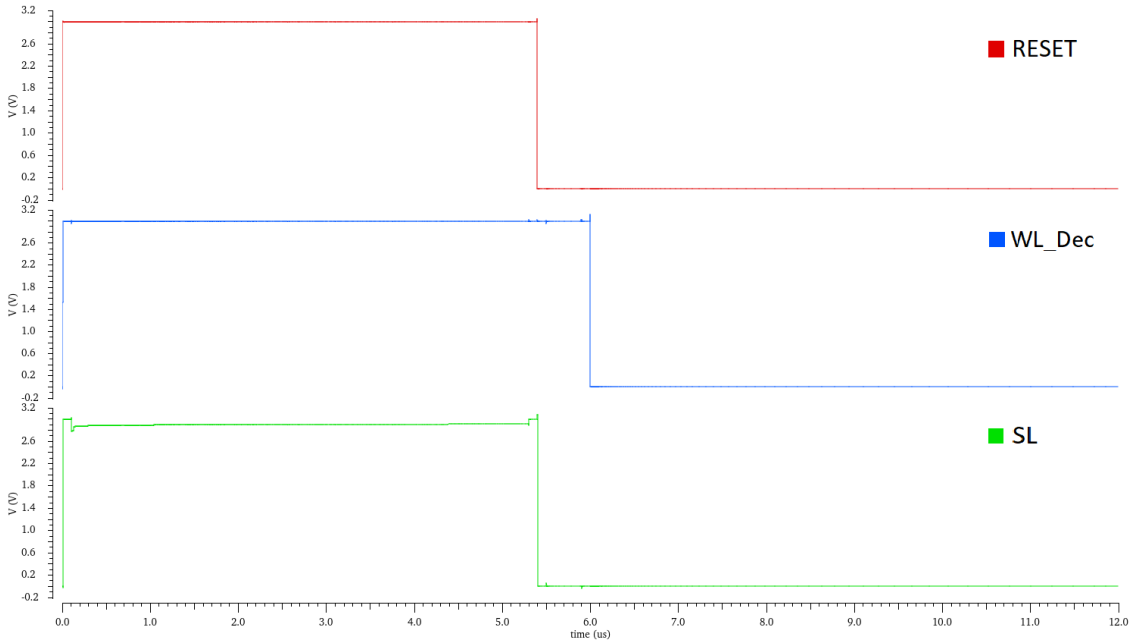


Figure 4.8: Select Line Driver - Waveforms

is based on the standard address decoder circuit, making use of NOR and NAND gates to select the appropriate output signal. It decodes the address it receives from the control circuit(shown in figure as A_1,$A_2 etc.$,) and selects the appropriate WL. The WL decoder shown in Figure 4.10 has two identical address decoders named Address decoder A and B respectively. These need to act in parallel for scouting logic operations such as AND and OR as they require two cells in the same BL to be open simultaneously. In order to have WL access for both these address decoders, an OR gate is introduced as shown in 4.10. This output, WL input is then provided as an input to the buffer which

is enabled by WL_{EN}.

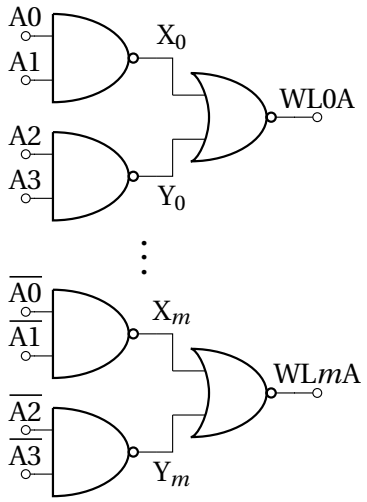


Figure 4.9: Address Decoder

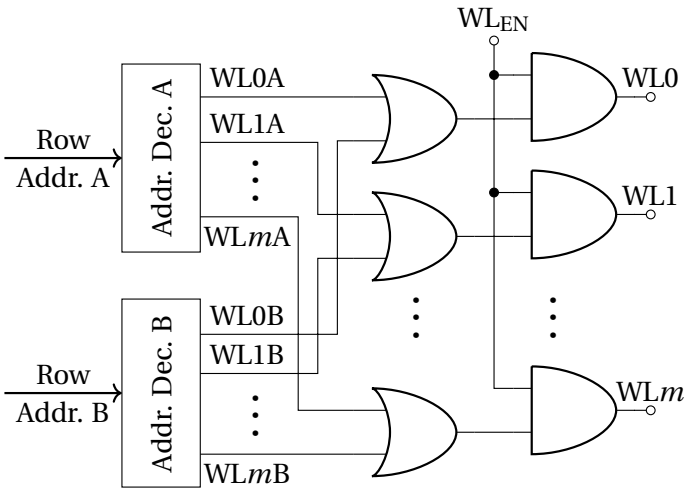


Figure 4.10: WL Decoder

The sense amplifier is adapted from Zhao et al., and is shown in Figure 4.11 [85]. The nodes A and B in the CMOS inverter couple are pre-charged by PMOS transistors P3 and P4 when there is no operation in the cell. Once the SA is enabled, the SEN and CS signals are triggered to turn off P3 and P4, and turn on NMOS transistors N3, N4, N5 and N6. This connects the sense amplifier to the array through the column and turns off the pre-charge connections. This allows nodes A and B to discharge through BL and BL_{ref}. The resistance of these nodes determine the rate of discharge in the cell. if $R_{BL} < R_{BLref}$, then BL will discharge faster. This is translated into node B being charged and A being discharged rapidly simultaneously. After some time, this result is captured as the operation outcome. Similarly, if $R_{BL} > R_{BLref}$, then node A discharges and node B charges rapidly, causing the opposite value to be saved. The Figure 4.14 gives the voltage level for the internal nodes A and B while a read operation takes place. It can be seen that the B discharges while A remains the same due to the positive feedback loop operation in the inverter pair. Logic AND and OR operations are enabled with a reference circuit, shown in Figure 4.12. This provides two resistances, R_{AND} and R_{OR}, that are selected with the help of the $Operation$ signal. These resistances have predetermined values that depend on the combined resistance of the cells in the array. This is illustrated in Figure 4.13. In the figure, it can be seen that the range of the logic values against the references. For example, '0' means a logic '0' in the cell and it is present in the high resistance end of the figure. '11' means the resistance is a combination of two cell each having logic '1'. When the references are set wrong with respect to the combined resistances of the cells under study, there could be wrong reading faults in the sense amplifier. For example, if the reference of R_{AND} is not set between '11' and '10' resistance ranges, then there could be an incorrect computation taking place. This reference can be used in the memory configuration as well, as the reference resistance for the read operation and the OR logic are the same. That is, $R_{read} = R_{OR}$.

4.2. Defect modeling

Defect that occur during the production phase of devices were discussed earlier in Chapter3. These defects have to be modeled electrically in order to be injected in the netlist and simulated to find the effect of their presence in the circuit. This is done in the defect modeling phase of structural testing. We take a look at the defect modeling in the memory configuration first, followed by the defect modeling in the computation configuration.

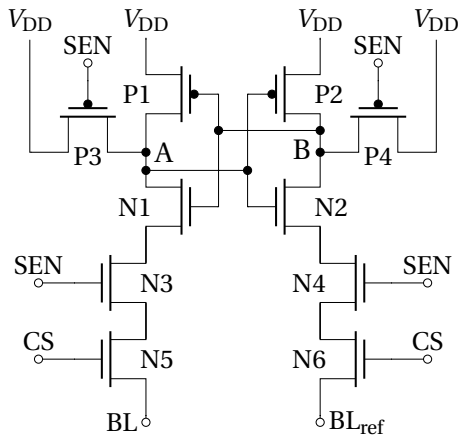


Figure 4.11: Sense Amplifier [85]

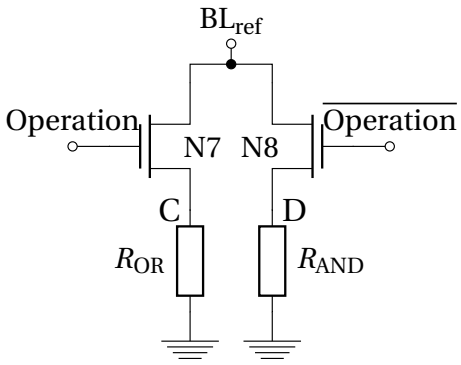


Figure 4.12: Operation reference

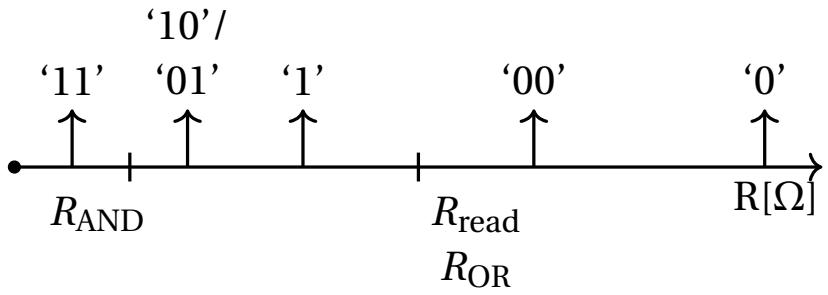


Figure 4.13: Scouting logic relative resistance and references

4.2.1. Defect modeling for memory configuration

The defects in Scouting logic can occur in the production of the ReRAM cell and the peripheral circuits. In the ReRAM cell, the defect can occur in any of the three phases of production: FEOL, BEOL and CF forming. The defects in FEOL discussed in Section 3.3.1 have always been modeled as linear resistors [37] [25]. The same holds true for defects that are formed during the build up of the metal layer from the FEOL to the BE of the memristor. All defects in the peripherals are modeled with linear resistors. The different defects are modeled by changing the resistance of the linear resistance injected as the defect. For example, a short is modeled with as a linear resistance with low resistance and a open is modeled witha linear resistance with high resistance. In order to model the forming defect that occurs in the CF formation phase in a ReRAM device, we make use of the device-aware defect modeling approach.

We use DAT to model the forming step in the ReRAM devices [17]. Here the forming current I_{form} is related to the length of the CF, l_{CF} and to the thickness of CF, ϕ_T. From [21], we obtain the median resistance of the device as a function of I_{form} by interpolation. This median resistance is used to obtain the standard deviation of the resistance, based on Equation (1) from [21]. These resistance values are used to model l_{CF} and ϕ_T. These parameters are modelled according to Equations (3) - (5) in [17].

The ReRAM device model used by Li et al. makes uses of all the above mentioned parameters as input [49]. It has two different equations for the SET and the RESET operation, which are based on the device parameters given in the Figure 4.15 adapted from [49].

The model is created in Verilog-A, and is fitted and calibrated to match the measurements of real

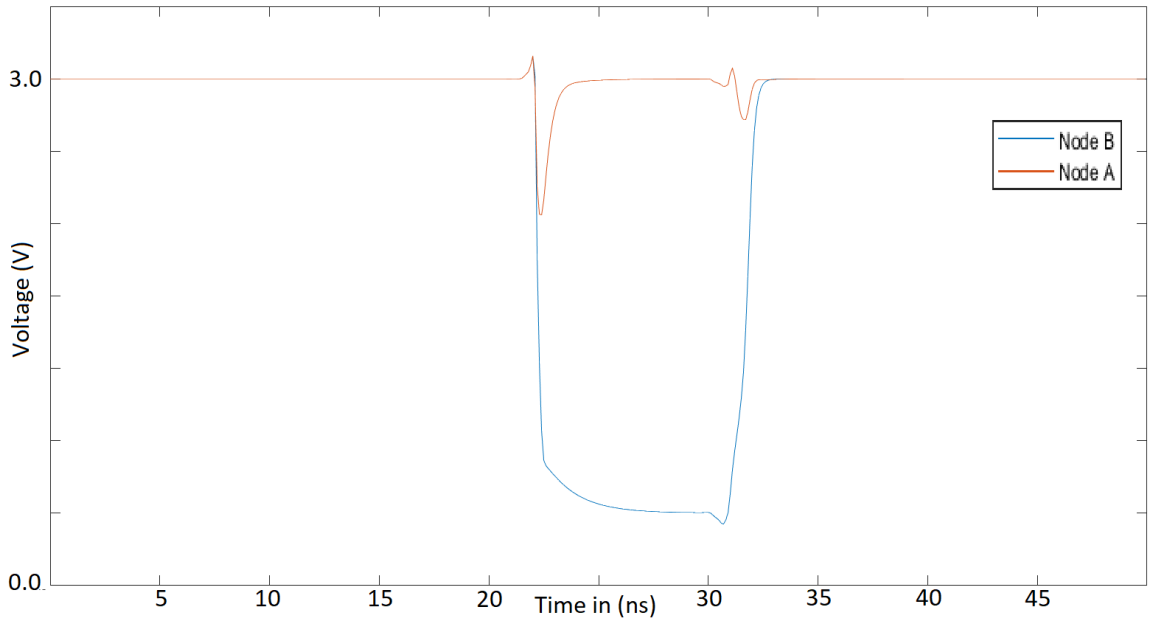


Figure 4.14: Sense Amplifier Internal Nodes

devices produced. In order to do that, the parameters l_{CF} and ϕ_T are varied and the corresponding mean resistance is found out. Then they are fitted against the real device measurements found in [21] and I_{form} is obtained from it. I_{form} is varied between 5 µA and 34.1 µA to obtain a wide range of resistances. With this a model that accurately describes the forming defect is obtained.

With the address decoder, a linear resistor is used as the defect model [30] [20]. Both strong and weak faults for the address decoder can be modeled using linear resistors by varying the resistance used [30]. In the sense amplifier, fault analysis is performed by assuming a linear resistor as the defect model. Similar to the address decoder, it can be proven that a linear resistance can cause both strong and weak faults.

4.2.2. Defect modeling in computation configuration

The defects are modeled for the computation configuration in the same way as the memory configuration. There will be no changes in the cause of the defects in these components hence a change in defect models is not required. Thus, we use linear resistors for the shorts, opens and bridges with the peripheral units that are modified for the computation configuration. The ReRAM is modeled as using DAT method to include the forming defect, and a linear resistance is used to model the defects that occur during the FEOL and BEOL phases of the production in ReRAM.

4.2.3. Defect Injection

In order to perform a complete fault analysis for the memory array, the defects need to be injected in the netlist. Forming defect was injected using the model described above and all other faults were analysed by injecting potential defects in the form of resistances in the memory cell. Table 4.2 gives the location of the different defects that were injected into the circuit. These defects were swept for a strength ranging from 1 Ω to 100 MΩ in a logarithmic scale in 10 steps. The defects in the array interconnects are modeled with the memory cell as series resistances in the connections. For the different resistances, ETD faults and HTD faults are expected to observed in the simulated scouting

Parameters And Values		Descriptions
I_0	10 μA/nm^2	hopping current density in the gap region
ρ	19.64 μΩ·m	resistivity of the CF
a	0.25 nm	distance between adjacent oxygen vacancy (V_O)
f	10^{13} Hz	vibration frequency of oxygen atom in V_O
E_a	1.2 eV	average active energy of V_O
E_h	1.0 eV	hopping barrier of oxygen ion (O^{2-})
E_i	1.2 eV	energy barrier between the electrode and oxide
α_a & α_h	0.75 nm	enhancement factor in lower E_a & E_h
γ	1.5	enhancement factor of external voltage
Z & e	1 & e	charge number & unit charge
Δw	0.5 nm	effective CF extending width
ΔE_r	0.8 eV	relaxation energy during recombination
R_{th}	5×10^5 K/W	effective thermal resistance
R_H	200 MΩ	parasitic resistance between electrodes
R_L	20 Ω	parasitic contact resistance of electrodes
C_P	20 fF	parasitic capacitance between electrodes
V_{TH0}	0.72 V	zero-bias threshold voltage of NMOS
C_G	1.42 fF	gate capacitance of NMOS
C_{GS}/C_{GD}	0.28 fF	overlap capacitance of NMOS
R_{wire}	12.78 Ω	wire resistance in the 1T-1R array
C_{wire}	0.046 fF	wire capacitance in the 1T-1R array

Figure 4.15: Model Parameters [49]

logic architecture, based on the location and strength of the defect.

S.No	Defect Resistance Location
1	BL and WL
2	BL and SL
3	BL and internal node
4	BL and VDD
5	BL and GND
6	Series with BL
7	WL and SL
8	WL and internal node
9	WL and VDD
10	WL and GND
11	Series with WL
12	SL and internal node
13	SL and VDD
14	SL and GND
15	Series with SL
16	Internal node and VDD
17	Internal node and GND

Table 4.2: Defect Location in the Memory cell

4.2.4. Experimental Setup

The above circuit was setup using HSPICE and the ReRAM memory device was modelled in Verilog-A. The defects are injected with varying inputs using a MATLAB script that generated a new netlist that specified the timings of the different input signal as required by the operation that takes place. The netlist was simulated using Cadence spectre circuit simulator in the command line. A batch process of these simulations were pushed to the server. In order to obtain only the required results

out of all the possible simulation results, a *.mdl* file was created. An .mdl file is based of the measurement descriptive language of Cadence, which enables the user to acquire measurements from the simulation at specified times. The results of such simulations were then parsed using MATLAB to obtain the results, which gave all the faults that were observed, as will be discussed in the following sections.

4.3. Fault Modeling and Analysis

This section deals with the modeling of the faults that occur in these devices. First the fault space for each configuration is defined and then fault verification is performed where we obtain all sensitized defects in the circuit. These faults would be used to develop the test for each configuration.

4.3.1. Fault Modeling and Analysis for Memory Configuration

The fault space for the different parts of the scouting logic architecture in memory configuration have already been explained in Section 3.3.2. We will now investigate the sensitized fault space for different defects injected in the array and also about the sensitized fault space in the peripherals.

Sensitized Fault space: Array

We now look at the faults that are sensitized in the memory configuration for the several defects mentioned in Table 4.2. These defects affect the operation in the memory structure, causing faults to be sensitized. These faults are represented as heat maps. We present an example heat map in Table 4.3 which is generated for Defect 1. The Black boxes indicate that there are no faults seen for the operations in the y-axis if a defect of strength given in the x-axis were to be injected in the circuit. White boxes indicate that the faults seen are easy to detect and the gray boxes indicate that the faults are hard to detect. The boxes themselves contain the state of the cells and the read output value of each of these operations. That is, the F/R component of the FPs introduced in Section 3.3.2 are indicated for every fault that was observed. In case of coupling faults, the unique fault combinations are mentioned in the tables for the sake of legibility. The faults sensitized are not only dependant on the cells that have been injected with defects, but also by the nature of the cells which form their neighbouring cells for the computation configuration. Hence they are simulated with different data combinations in the neighbouring cells. These give rise to four different array patterns which can provide different outputs. Tables 4.4 - 4.7 show the four different combinations for the memory configuration operations. In the first combination, the cells in the word that resides above the word with the defective cell is in a state opposite to the initial state of the defective cell. For example, it the defective cell is in the state '1', then all the cells in the word above the word containing the defective cell would be in state '0'. All the other cells in the array in this combination are in state '0'. In the second combination, the states of the cells in the word above the word are in the same state while the other cells in the array are in state '0'. The third and fourth combinations are an extension of combinations 1 and 2, but the state of the other cells in the array are '1'. With these combinations, the effects of the defect on the neighbouring cells can also be studied.

We shall now look at the different defects that were injected in the memory configuration, and analyse the faults that are obtained in these simulations.

Table 4.3: Heat Map Example

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 1 (Tables 4.4 - 4.7) is a resistive bridge between the wordline and the bitline. When the defect resistance is low, it acts as a bridge and it loses its voltage driving capability as the defect resistance increases. While performing a $0r0$ operation, the bitline reaches higher potential comparable to the SET voltage (V_{SET}) and thus sets the state of the cell to '1' causing a read disturb fault RDF0-(1). This effect vanishes at a high defective resistance (1.6 MΩ. In case of a $1w0$ operation, all the common word line is driven to GND by the bit line that is connected to via the defective resistance. This causes no operation to take place in the cells, and they remain in the '1' state, causing a write transition fault WTF0-(1). This effect can be seen in all the cells in the word where the same word line is used to activate the gate of the transistor. This operation can also cause incomplete transitions in the cell, which causes the cell to reach an undefined state 'U' because of the partial opening of the cells. For a $1r1$ operation, the bit line is at a higher voltage than the R_{BLref} arm of the sense amplifier at lower defective resistances and hence the R_{BLref} arm discharges faster. This causes an incorrect read fault (IRF-1(1)) in the sense amplifiers in the defective word.

Table 4.4: Heat Map Defect-1 Configuration 1

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/– U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.5: Heat Map Defect-1 Configuration 2

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.6: Heat Map Defect-1 Configuration 3

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.7: Heat Map Defect-1 Configuration 4

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 2(Tables 4.8 - 4.11) is a resistance that is injected between the bitline and the select line. This causes an incorrect read fault for $0r0$ as the bitline is short to the select line and hence the ReRAM cell is not registered at the sense amplifier for $R_{\text{Def}} < R_{\text{ON}}$. This defect also causes a coupling fault only in cells in the '0' state which are in the same word line as the defective cell to be set to state '1' even though no operation is performed on these cells during a $0w0$, $1w0$, $1w1$ and $0w1$ operations. A write transition fault is also seen for $1w0$ and $0w1$ operations as the bitline and select line are shorted. A transition fault to undefined region WTF-1(U) is seen in the defective cell for $R_{\text{Def}} = 3.59\,\text{k}\Omega$ as the voltage is not powerful enough to drive the transition $1w0$.

Table 4.8: Heat Map Defect-2 Configuration 1

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.9: Heat Map Defect-2 Configuration 2

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1;U/- 1;U/- 1/-	U/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.10: Heat Map Defect-2 Configuration 3

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1;U/- 1/-	U/-					
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.11: Heat Map Defect-2 Configuration 4

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1;U/- 1/-	U/-					
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 3 (Tables 4.12 - 4.15) is injected as a resistance between the bitline and the node between the ReRAM device and the transistor, referred to as the internal node. We observe an incorrect read fault with the operation $0r0$ for R_{Def} less than the resistance offered by a cell in logic '0'. This is because the current for measurement takes the shortest path, which is provided by the defective resistance. This defect also causes Write transition faults when the defective resistance R_{Def} is lesser that the resistance of the states. When R_{Def} is in the same order as the resistance, it causes a write transition fault to undefined region, as seen in operation $1w0$. The fault ceases to exist when the defect is of a higher resistance than the target state of the transition.

Table 4.12: Heat Map Defect-3 Configuration 1

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.13: Heat Map Defect-3 Configuration 2

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.14: Heat Map Defect-3 Configuration 3

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 4 (Tables 4.16 - 4.19) is a short between the bit line and VDD. The defective resistance adds up to be part of the read operations which makes the effective resistance to the sense amplifier to be $R_{\text{Def}} + R_{\text{cell}}$ when the defective resistance is in the same order of the cell in state '1'. This causes Incorrect read faults when $1r1$ is performed and an destructive read when $0r0$ is performed on the defective cell. Because of the presence of the voltage, $0w0$ causes write destructive fault WDF-0(1) and operation $1w0$ causes write transition fault WTF-0(1) as the defective cell would be set to logic '1' by the short.

Table 4.15: Heat Map Defect-3 Configuration 4

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.16: Heat Map Defect-4 Configuration 1

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.17: Heat Map Defect-4 Configuration 2

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.18: Heat Map Defect-4 Configuration 3

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 5 (Tables 4.20 - 4.23) is a short between the bit line and GND. A transition from '0' to '1' does not take place for smaller values of R_{Def} as the voltage that is at the bitline would be not high enough to set the ReRAM device to state '1'. Along the same reasoning, the operation $1w1$ would result in a write destructive fault as the cell would be in state '0' from the reset operation that precedes a set operation, as mentioned in Chapter 2. The $0r0$ would result in incorrect read fault as the R_{BLref} arm of the sense amplifier would be at a higher potential than the R_{BL} arm of the sense amplifier due to the direct short it has with GND. This causes the output of the sense amplifier to be '1' irrespective of the state of the cell where the read operation is performed. This would not affect the $1r1$ operation and hence a fault would not be captured.

Table 4.19: Heat Map Defect-4 Configuration 4

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.20: Heat Map Defect-5 Configuration 1

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0										
1w1	0/-	0/-	0/-	0/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.21: Heat Map Defect-5 Configuration 2

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0										
1w1	0/-	0/-	0/-	0/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.22: Heat Map Defect-5 Configuration 3

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0										
1w1	0/-	0/-	0/-	0/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 6 (Tables 4.24 - 4.27) is a resistance injected in series with the bit line. It is worth noting that this defect also doubles as a resistance injected in series with the internal node as it would make no difference. The resistance which sensitize faults in this defect is always in an order that is greater than or equal to the resistance of the ON state. For resistance ranges in the order of mega Ohms, the defect serves as an open, which causes the sense amplifier output to be always '1' as the R_{BLref} arm of the sense amplifier would discharge first. This is reflected in the incorrect read faults with $0r0$ operation. For resistance in the same range of the ReRAM resistances in logical state '0' and '1', the total resistance of the unit under consideration becomes $R_{\text{Def}}+R_{\text{cell}}$. This causes the sense amplifier

Table 4.23: Heat Map Defect-5 Configuration 4

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
1w0										
1w1	0/-	0/-	0/-	0/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

to give the output '0', which registers as an incorrect read fault for $1r1$ operation. During the write operations, the resistance is serves as a voltage drop which hinders the set operation. This causes write transition faults for operation $0w1$ for high defect resistances. During the reset operation ($1w0$) there is also a write transition fault, for the same reason.

Table 4.24: Heat Map Defect-6 Configuration 1

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.25: Heat Map Defect-6 Configuration 2

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.26: Heat Map Defect-6 Configuration 3

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.27: Heat Map Defect-6 Configuration 4

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
1w0					1/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 7 (Tables 4.28 - 4.31) is a defective resistance between word line and the select line. This causes the set operation $0w1$ and the reset operation $1w0$ do not occur as the transistor gate does not open properly. However, the $1w1$ operation shows a Write destruction fault as the operation is a combination of two transitions and the cell gets stuck in state '0'. The read operations change with the defect resistance, as with lower defect resistance, the discharge is much faster, which causes a fault in $0r0$ and with defect resistance that is higher, the gate partially opens, which causes an incorrect read fault in the operation $1r1$.

Table 4.28: Heat Map Defect-7 Configuration 1

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0		U;1/- U/-	1;1/- 1/-							
1w1		0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.29: Heat Map Defect-7 Configuration 2

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0										
1w1	0;0/- 0/-			0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.30: Heat Map Defect-7 Configuration 3

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0	1;1/- 1/-									
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.31: Heat Map Defect-7 Configuration 4

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0			1;1/- 1/-							
1w1		0;0/- 0/-								
1r1				1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 8 (Tables 4.32 - 4.35) is a bridge between the word line and the internal node. This causes the the bottom electrode of the ReRAM device to be at a higher voltage than usual. This causes a Write destructive fault to the logic state 'L' for the operation $0w0$ as the cell is pushed much further into the HRS because of the higher voltage. This is remains valid as long as the defective resistance is low. By the same operational logic, the operation $1w0$ has a write transition fault to state 'L'. Both read operations push the state of the device to 'L' as well, which gives a read destructive fault for operation $0r0$ and a incorrect read destructive fault for operation $1r1$

Table 4.32: Heat Map Defect-8 Configuration 1

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.33: Heat Map Defect-8 Configuration 2

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.34: Heat Map Defect-8 Configuration 3

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.35: Heat Map Defect-8 Configuration 4

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 9 (Tables 4.36 - 4.39) is a short between the word line and VDD. This causes the word line to be activate for the entire duration of the write operations. However, this does not cause any strong faults in the defective cell as this does not affect the operations in the cell. However, as the word line is activate, the neighbouring cells in the first two combinations would change their states from '0' to '1'.

Table 4.36: Heat Map Defect-9 Configuration 1

0										
1										
0w0	0;1/-	0;1/-	0;1/-							
0w1	1;1/-	1;1/-	1;1/-							
0r0										
1w0	0;1/-	0;1/-	0;1/-							
1w1	1;1/-	1;1/-	1;1/-							
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.37: Heat Map Defect-9 Configuration 2

0										
1										
0w0	0;1/-	0;1/-	0;1/-							
0w1	1;1/-	1;1/-	1;1/-							
0r0										
1w0	0;1/-	0;1/-	0;1/-							
1w1	1;1/-	1;1/-	1;1/-							
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.38: Heat Map Defect-9 Configuration 3

0										
1										
0w0										
0w1										
0r0										
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.39: Heat Map Defect-9 Configuration 4

0										
1										
0w0										
0w1										
0r0										
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 10 (Tables 4.40 - 4.43) is a short between the wordline and GND. This causes no operation to take place in the cells and thus no transitions are possible with this defect in the cell. This causes incorrect read faults with $0r0$ and $1r1$ operations. The IRF occurs in $0r0$ when the resistance of the defect is low enough that the cell is not open by the WL, whereas it occurs in $1r1$ for resistances where the cell is only partially open. When it is partially open, the resistance of the transistor is high enough to give incorrect read faults in the sense amplifier. This explains the presence of the defect for only one resistance value in the defect.

Table 4.40: Heat Map Defect-10 Configuration 1

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1				1;1/0 1/0						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.41: Heat Map Defect-10 Configuration 2

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1				1;1/0 1/0						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.42: Heat Map Defect-10 Configuration 3

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1						
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1				1;1/0 1/0						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.43: Heat Map Defect-10 Configuration 4

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1				1;1/0 1/0						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 11 (Tables 4.44 - 4.47) has a series resistance with the word line. This causes the cell to not open at high resistances, effectively behaving like an open. This causes incorrect read faults for the $0r0$ operation. This effect is not seen in operation $1r1$ as the sense amplifier gives an output of '1' when there is no input, which registers as no fault with the operation.

Table 4.44: Heat Map Defect-11 Configuration 1

0										
1										
0w0										
0w1										
0r0									0/1	0/1
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.45: Heat Map Defect-11 Configuration 2

0										
1										
0w0										
0w1										
0r0									0/1	0/1
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.46: Heat Map Defect-11 Configuration 3

0										
1										
0w0										
0w1										
0r0								0/1	0/1	0/1
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.47: Heat Map Defect-11 Configuration 4

0										
1										
0w0										
0w1										
0r0									0/1	0/1
1w0										
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 12 (Tables 4.48 - 4.51) connects the internal node and the select line with a resistor, effectively introducing a parallel resistor across the transistor. This causes the internal node to be at a higher voltage than usual during reset operations. This manifests as a Write destructive fault WDF-0(L) for $0w0$ operation and as a write transition fault WTF-0(L) for the operation $1w0$.

Table 4.48: Heat Map Defect-12 Configuration 1

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.49: Heat Map Defect-12 Configuration 2

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.50: Heat Map Defect-12 Configuration 3

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.51: Heat Map Defect-12 Configuration 4

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 13 (Tables 4.52 - 4.55) has a short between the select line and VDD. This causes write transition faults with the operation $0w1$ as the cell could not be reset by the circuit. This effect is seen in both the defective cell and other cells in the same word. By extension , this also causes a write destructive fault in $1w1$ operation as a set operation has both set and reset operations combined. During read operation, the sense amplifier is pushed to give an output '0' as a result of this defect. This gives an incorrect read fault with $1r1$ operation.

Table 4.52: Heat Map Defect-13 Configuration 1

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0										
1w0										
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.53: Heat Map Defect-13 Configuration 2

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0										
1w0										
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.54: Heat Map Defect-13 Configuration 3

0										
1										
0w0	0;U/-	0;U/-	0;U/-	0;U/-						
0w1	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-						
0r0										
1w0	0;U/-	0;U/-	0;U/-	0;U/-						
1w1	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.55: Heat Map Defect-13 Configuration 4

0										
1										
0w0	0;U/-	0;U/-	0;U/-	0;U/-						
0w1	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-						
0r0										
1w0	0;U/-	0;U/-	0;U/-	0;U/-						
1w1	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 14 (Tables 4.56 - 4.59) has a short between the select line and GND. A reset operation $1w0$ is affected by this defect and is seen as a write transition fault WTF-0(1) in case of lower defect resistance, and as a WTF-0(U) for a higher defect resistance. There are faults in the neighbouring cells in the same row as the defective cell, as the residual current that flows through the bit lines change the state of these cells to state '1'.

Table 4.56: Heat Map Defect-14 Configuration 1

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	1;1/-	1;1/-	1;1/-	1;1/-						
0r0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-						
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.57: Heat Map Defect-14 Configuration 2

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	1;1/-	1;1/-	1;1/-	1;1/-						
0r0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-						
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.58: Heat Map Defect-14 Configuration 3

0										
1										
0w0										
0w1										
0r0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-						
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.59: Heat Map Defect-14 Configuration 4

0										
1										
0w0										
0w1										
0r0										
1w0	1;1/- 1/-		1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1										
R_{Def}	1		7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 15 (Tables 4.60 - 4.63) has a series resistance with the select line. This causes faults where the write transitions do not occur for high defect resistance, as the resistance of the cell now equals the sum of resistance of the cell and the defect. This thus causes WTF-0(0) and WTF-1(1) in operations $0w1$ and $1w0$ respectively. The operation $1r1$ is also affected by this, as the resistance of the cell now causes lesser current to flow through the cell and hence desired output is not achieved.

Table 4.60: Heat Map Defect-15 Configuration 1

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.61: Heat Map Defect-15 Configuration 2

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.62: Heat Map Defect-15 Configuration 3

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.63: Heat Map Defect-15 Configuration 4

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 16 (Tables 4.64 - 4.67) shorts the internal node of the cell to VDD. This causes the state of the cell to flip to 'L' irrespective of the operation performed in them. This is seen as state faults (SF-0(L) and SF-1(L)) in the simulations. This also causes write transition faults (WTF-0(L) and WTF-1(L)) and write destruct faults (WDF-0(L) and WDF-1(L)). This defect also causes read destructive faults and incorrect read destructive faults in the operation.

Table 4.64: Heat Map Defect-16 Configuration 1

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.65: Heat Map Defect-16 Configuration 2

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
1w0	L/-	L/-	L/-	L/-	L/-	L;1/- L;1/- L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.66: Heat Map Defect-16 Configuration 3

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.67: Heat Map Defect-16 Configuration 4

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 17 (Tables 4.68 - 4.71) shorts the internal node of the cell to GND. This causes defective cells in array to have a write destructive fault WDF-0(1) while performing a $0w0$ operation, as the residual current that flows through the ReRAM device is strong enough to push the state of the device to logic '1'. For the $1w0$ the device is not reset to logic '0', hence it causes a write transition fault WTF-0(1). It is worth noting that both transition and destructive faults seen have a fault where the state of the defective cell ends up in the undefined state 'U' for $R_{\text{Def}} = 27.84\,\text{k}\Omega$.

Table 4.68: Heat Map Defect-17 Configuration 1

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.69: Heat Map Defect-17 Configuration 2

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.70: Heat Map Defect-17 Configuration 3

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.71: Heat Map Defect-17 Configuration 4

4 0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Forming Defect (Tables 4.72 - 4.75) can be split into three phases: Under forming, over forming and the accurate forming phases. In the under formed state, the cell is in a perpetually high resistant state. This is because the conducting filament has not been fully developed, and the cell behaves as if in the HRS region. This causes ETD faults in the $0w1$, $0r0$, $1w1$ and $1r1$ operations. The fault in operation $0r0$ is due to the limited points of fitting used int the forming model. This point of inflection from an extremely high resistance to a low resistance causes simulation inaccuracies. The states of these cells in the under formed cells is 'L'. As the forming current increases, the cell moves from being in the 'L' state to 'U' state. This then gets to the right phase, where there are no errors in the memory operations. When the cell becomes over formed it is pushed to the 'H' state. This is seen in the $1w0$ operation, where the state of the cell is in '1' and 'H'.

Sensitized Fault space: Peripherals

We shall now look at the sensitized fault space for the peripherals. All the faults in the fault space for the address decoders can be sensitized by defects. This has been studied and discussed in Section 3.3.2. Hence we do not repeat it here. The same applies for the sense amplifiers in the memory configuration. We shall look at some demonstrative faults in the address decoder and sense amplifier for the sake of clarity.

Address Decoder: An open defect in the WL translates an address delay (ADF) fault for resistance R_{def} = 40 kΩ while it causes a Address fault no access (AFna) for resistance R_{def} = 4 MΩ. This is illustrated in Figure 4.16, where the defect-free WL is also illustrated for comparison.

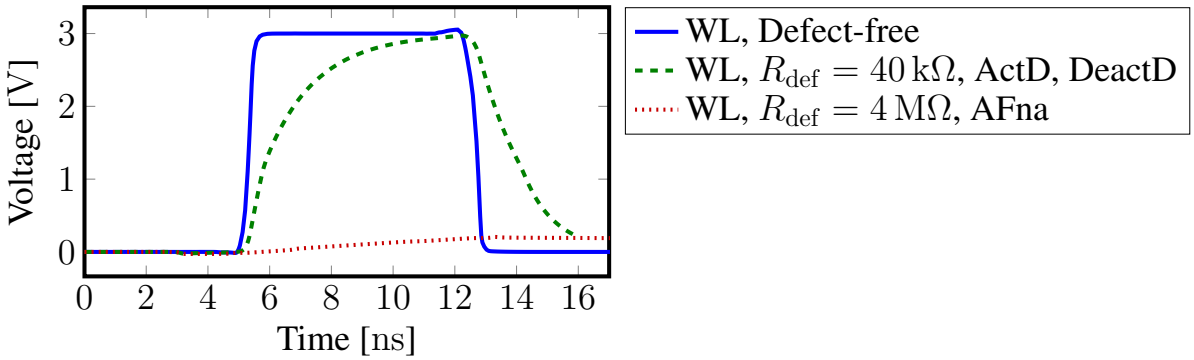


Figure 4.16: WL Decoder fault with different defect strengths

Sense Amplifier: Consider a resistive open defect R_{def} between N2 and N4 in Figure 4.11. This leads to slow discharge of node B as there is more resistance in the path. if $R_{def} < 18.8\,\text{k}\Omega$, the defect causes a slow operation in the sense amplifier, while a defect strength of $R_{def} \geq 18.8\,\text{k}\Omega$ would cause the sense amplifier to always switch to the wrong value when the CS and SEN signals are open. The

Table 4.72: Heat Map Forming defect Configuration 1

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																			U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	H/-	H/-	H/-	H/-	H/-		
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1													H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
I_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.73: Heat Map Forming defect Configuration 2

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																			U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	1/-	H/-	H/-	H/-		
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1													H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
I_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.74: Heat Map Forming defect Configuration 3

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																			U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	1/-	H/-	H/-	H/-		
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1													H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
I_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.75: Heat Map Forming defect Configuration 4

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																			U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	U/-	1/-	H/-	H/-	H/-		
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-													H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1													H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
I_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

former case gives rise to USAFs and SSAFs, while the latter leads to SASF. This is shown in the Figure 4.17, along with a defect free signal.

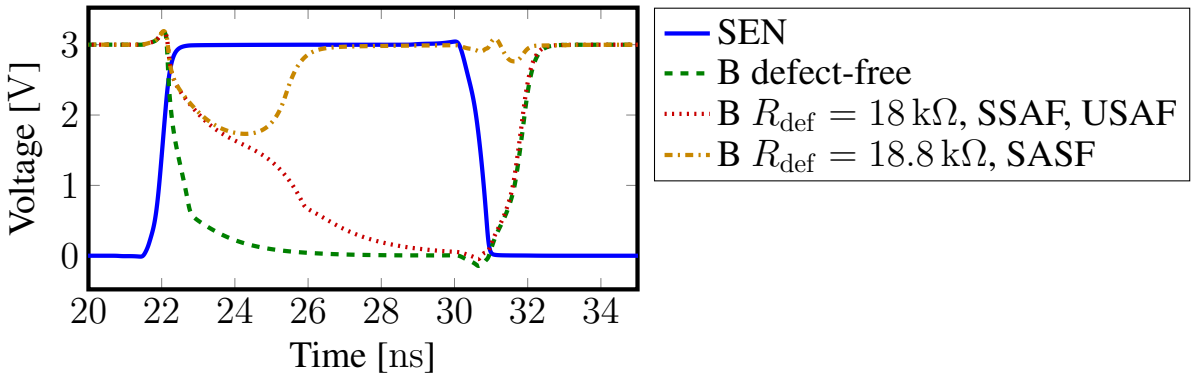


Figure 4.17: Sense Amplifier Faults

4.3.2. Fault modeling and Analysis for Computation configuration

This section deals with fault modeling in the computation configuration. We look at the fault modeling followed by fault analysis of the components in the scouting logic based CIM architecture for the computation configuration.

Fault modeling: Array

The fault space for the memory array in the computation configuration is strongly dependant on the crossbar design and architecture of the system. In the case of scouting logic based architecture, the memory array acts akin to a dual port memory as it allows for simultaneous access of two memory cells in the same column. This gives rise to new faults, which need to be defined. The FP notation introduced in Section 3.3.2 has to be expanded in order to accommodate these faults. We build on the notation developed for dual-port memory faults [29].

We denote the new FP as $\langle S_1 : S_2 \ / \ F_1 : F_2 \ / \ R \rangle_{OP}$. The elements in the FP are as follows:

1. $S_1 : S_2$ denote the sensitizing operations in the cells that are under consideration. The symbol ':' in between them signifies that the operations S_1 and S_2 are performed simultaneously.
2. F_1:F_2 denote the final state of the cells after the operations.
3. R is the read value from the sense amplifier.
4. OP is the operation that is performed in the cell. These include AND, OR and XOR for the scouting logic and varies with the logic architecture used.

For example, $\langle 0a0_1 : 1a1_2 \ / \ 1_1 : 1_2 \ / \ 0 \rangle_{AND}$ describes a cell (cell 1) that flips its state from '0' to '1' when an AND operation is performed on it. The result of the AND operation is '0' as expected. This fault can occur if cell 1 is overformed. In that case, the HRS and LRS if the cell is lowered. If such a cell is used for logic computations, it causes the equivalent resistance to move too close or even lower than R_{AND} in the sense amplifier. This causes a fault $\langle 0a0_1 : 1a1_1 \ / \ 1_1 : 1_2 \ / \ 0 \rangle_{AND}$. While naming of these faults is an open question, this thesis used the following method to name faults in the computation configuration: Faults caused by computation operations are prefixed with C(x) where C is the x denotes the type of computation operation performed on the cell, in place of the cell behavior of an FP as explained in Section 3.3.2. For example, C(a)DF-1/0 is a computation destructive fault with an AND operation with the states of the cell shown to change from '1' to '0', effectively causing a destruction.

Sensitized Fault space: Array

This subsection presents the faults that were sensitized in the computation configuration for the various defects injected. The heat maps are combined with the operations in the memory configuration for ease. The combinations are different for the computation configuration, where the opposite word is the first initiated in the word adjacent to the word containing the defective cell and then it is followed by the operation with the other word initiated with the same word. For example, the first combination is $0o0 : 1o1$ and the second combination is $0o0 : 0o0$. The third and fourth combinations have other cells in the array to be '1'.

Defect 1 (Tables 4.76 - 4.79) causes a C(a)DF-0(1)-24 fault while performing the operation $0a0$ and C(o)DF-0(1) fault while performing $0o0$ operation. These are attributed to the same cause as the $0r0$ operation in the memory configuration: A higher bitline voltage which can set the state of the cell to '1' while operations are performed in the defective cell. An Incorrect computation fault is seen with $1a1$ and $1o1$ operations on the defective cell and these are similar to the $1r1$ fault in the memory configuration: A voltage higher than the sense amplifier arm is seen at the bitline and hence there is an incorrect computation fault. However, it is interesting to note the different faults that arise in the case of $0o0$ for specific neighbouring cell configurations, as the voltage is not too high to cause of Destructive fault but is high enough to cause an incorrect computation fault. This is seen when the defective resistance $R_{\text{Def}} = 0.15\,\text{M}\Omega$ and when the cell which is the other operand of the logical operation is in a '1' state.

Table 4.76: Heat Map with computation Defect-1 Configuration 1

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0					
0o0	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;1/0 1;1/1	0;1/0 0;1/0	0;1/0 0;1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0					
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;1/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.77: Heat Map with computation Defect-1 Configuration 2

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0				
0o0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0				
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1			
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.78: Heat Map with computation Defect-1 Configuration 3

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0					
0o0	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;1/0 1;1/1	0;1/0 0;1/0	0;1/0 0;1/0			
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1;U/- 1/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0					
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;1/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.79: Heat Map with computation Defect-1 Configuration 4

0										
1										
0w0										
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0				
0o0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0				
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 2 (Tables 4.80 - 4.83) results in incorrect computation for the operation $0a0$ because of the short between the bitline and word line. The defect also causes incorrect computation fault with operations $1a1$ and $0o0$ when the other operand for the logic operation is in logic '0'.

Table 4.80: Heat Map with computation Defect-2 Configuration 1

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0					
0o0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.81: Heat Map with computation Defect-2 Configuration 2

0										
1										
0w0	0;1/-	0;1/-	0;1/-	0;1/-						
0w1	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-	0;1/- 0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1;U/- 1;U/- 1/-	U/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-						
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 3 (Tables 4.84 - 4.87) causes incorrect computation fault with operation $0a0$ for all possible logical operands. The defective resistance provides the current that is would give an output of '1' for an AND operation irrespective of the state of the neighbouring cells. This results in an IC(a)F-0(0) fault. The same principle applies to $1a1$ and $0o0$ operations when the other operand for computation in state '0'.

Table 4.82: Heat Map with computation Defect-2 Configuration 3

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0				
0o0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1;U/- 1/-	U/-					
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.83: Heat Map with computation Defect-2 Configuration 4

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;U/- 1;U/- 1/-	U/-					
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.84: Heat Map with computation Defect-3 Configuration 1

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0					
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.85: Heat Map with computation Defect-3 Configuration 2

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.86: Heat Map with computation Defect-3 Configuration 3

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0					
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.87: Heat Map with computation Defect-3 Configuration 4

0										
1										
0w0										
0w1	0/-	0/-	0/-	0/-						
0r0	0/1	0/1	0/1	0/1	0/1					
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1					
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 4 (Tables 4.88 - 4.91) causes computation destruction faults when the defective cell is in state '0' and a computation operation is performed on it in the presence of the short. When the defective cell is in state '1' already, there is an incorrect computation fault due to the presence of the voltage in the bitline. These defects, however, do not cause faults when the defective resistance R_{Def} is much higher in order than the resistance of state '0'.

Table 4.88: Heat Map with computation Defect-4 Configuration 1

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/1 1;1/0			
0o0	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;1/0 1;1/1	1;1/0 1;1/1	1;1/1			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/1 1;1/0			
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;1/0 1;1/0	1;1/0 1;1/0	1;1/1			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.89: Heat Map with computation Defect-4 Configuration 2

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
0o0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1			
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.90: Heat Map with computation Defect-4 Configuration 3

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/1 1;1/0			
0o0	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;0/0 1;1/0 1;1/1	1;1/0 1;1/1	1;1/0 1;1/1	1;1/1			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/0	1;1/1 1;1/0			
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;1/0 1;1/0	1;1/0 1;1/0	1;1/1			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.91: Heat Map with computation Defect-4 Configuration 4

0										
1										
0w0	1/-	1/-	1/-	1/-						
0w1										
0r0	1/0	1/0	1/0	1/0	1/0	1/0	1/0			
0a0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
0o0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
1w0	1/-	1/-	1/-	1/-	U/-					
1w1										
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1/0	1/0			
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 5 (Tables 4.92 - 4.95) pushes the read output of the sense amplifier to be '1' always. This was explained in the section describing the faults sensitized in the memory configuration. As logical computation in computation configuration is analogous to read operation in scouting logic, we extend the same logic to explain the faults present in the computation configuration. The operation $0a0$ gives a incorrect computation fault for all possible operand combinations as the output is always '1'. The operations $1a1$ and $0o0$ show incorrect computation faults when the state of the other operand for computation is logic '0'.

Table 4.92: Heat Map with computation Defect-5 Configuration 1

0									
1									
0w0									
0w1	0/-	0/-	0/-	0/-					
0r0	0/1	0/1	0/1	0/1	0/1				
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0				
0o0									
1w0									
1w1	0/-	0/-	0/-	0/-					
1r1									
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0				
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.93: Heat Map with computation Defect-5 Configuration 2

0									
1									
0w0									
0w1	0/-	0/-	0/-	0/-					
0r0	0/1	0/1	0/1	0/1	0/1				
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1				
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1				
1w0									
1w1	0/-	0/-	0/-	0/-					
1r1									
1a1									
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.94: Heat Map with computation Defect-5 Configuration 3

0									
1									
0w0									
0w1	0/-	0/-	0/-	0/-					
0r0	0/1	0/1	0/1	0/1	0/1				
0a0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0	0;1/1 0;1/0			
0o0									
1w0									
1w1	0/-	0/-	0/-	0/-					
1r1									
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0			
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.95: Heat Map with computation Defect-5 Configuration 4

0									
1									
0w0									
0w1	0/-	0/-	0/-	0/-					
0r0	0/1	0/1	0/1	0/1	0/1				
0a0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1				
0o0	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1	0;0/1 0;0/1				
1w0									
1w1	0/-	0/-	0/-	0/-					
1r1									
1a1									
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 6 (Tables 4.96 - 4.99) has faults only in the $1a1$ operation when the other operand is logic '1' and in the operation $1o1$ when the other operand is logic '0'. In the former, the both these cases, the added resistance in the bitline limits the voltage to the sense amplifier, which causes an incorrect read fault in the sense amplifier.

Table 4.96: Heat Map with computation Defect-6 Configuration 1

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
0a0										
0o0										
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
1a1										
1o1					1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.97: Heat Map with computation Defect-6 Configuration 2

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
0a0										
0o0										
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
1a1						1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.98: Heat Map with computation Defect-6 Configuration 3

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0							0/1	0/1	0/1	0/1
0a0										
0o0										
1w0					U/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0				
1a1										
1o1						1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.99: Heat Map with computation Defect-6 Configuration 4

0										
1										
0w0										
0w1						0/-	0/-	0/-	0/-	
0r0						0/1	0/1	0/1	0/1	
0a0										
0o0										
1w0					1/-	1/-	1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0			
1a1						1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 7 (Tables 4.100 - 4.103) has faulty 1a1 and 1o1 operations when the other operands in the other cells are in state '0'. This is because the cell is not open properly which causes the operations to fail.

Table 4.100: Heat Map with computation Defect-7 Configuration 1

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
0a0										
0o0										
1w0		U;1/- U/-	1;1/- 1/-							
1w1		0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
1a1										
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.101: Heat Map with computation Defect-7 Configuration 2

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
0a0										
0o0										
1w0										
1w1	0;0/- 0/-			0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.102: Heat Map with computation Defect-7 Configuration 3

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
0a0										
0o0										
1w0	1;1/- 1/-									
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1				1;1/0 1/0	1;1/0 1/0					
1a1										
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.103: Heat Map with computation Defect-7 Configuration 4

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
0a0										
0o0										
1w0			1;1/- 1/-							
1w1		0;0/- 0/-								
1r1				1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1					
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 8 (Tables 4.104 - 4.107) causes the state of the cell to move to HRS, which is denoted as logic state 'L'. This is also seen in the computation operations, as operations $0a0$ and $0o0$ cause computation destruction faults. These also cause coupling faults in the other operand, as the voltage that is obtained due to the bridge drives the state of this cell to '1'. With the operations $1a1$ and $1o1$, there is computation destruction fault to state 'L' seen as well.

Table 4.104: Heat Map with computation Defect-8 Configuration 1

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
0a0	L;1/0	L;1/0	L;1/0	L;1/0						
0o0	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	0;1/0 0;1/0					
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
1a1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	U;1/0 U;1/1					
1o1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	U;0/0 U;1/0 U;1/1	1;0/0 1;1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 9 (Tables 4.108 - 4.111) does not cause any fault in the computation configuration as well, as the bitline for the other columns are pulled down during computation configuration operations.

Table 4.105: Heat Map with computation Defect-8 Configuration 2

2 0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
0a0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0					
0o0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0					
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
1a1	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	U;1/0 U;0/0	1;1/0 1;0/1				
1o1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	U;1/0 U;0/1					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.106: Heat Map with computation Defect-8 Configuration 3

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
0a0	L;1/0	L;1/0	L;1/0	L;1/0						
0o0	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	0;1/0 0;1/0					
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
1a1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	U;1/0 U;1/1					
1o1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	U;0/0 U;1/0 U;1/1	1;0/0 1;1/0				
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.107: Heat Map with computation Defect-8 Configuration 4

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0	L/0	L/0	L/0	L/0						
0a0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0					
0o0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0					
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1;1/0 1/0	1/0				
1a1	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	U;1/0 U;0/0	1;1/0 1;0/1				
1o1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	U;1/0 U;0/1					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.108: Heat Map with computations Defect-9 Configuration 1

0										
1										
0w0	0;1/-	0;1/-	0;1/-							
0w1	1;1/-	1;1/-	1;1/-							
0r0										
0a0										
0o0										
1w0	0;1/-	0;1/-	0;1/-							
1w1	1;1/-	1;1/-	1;1/-							
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.109: Heat Map with computations Defect-9 Configuration 2

0										
1										
0w0	0;1/-	0;1/-	0;1/-							
0w1	1;1/-	1;1/-	1;1/-							
0r0										
0a0										
0o0										
1w0	0;1/-	0;1/-	0;1/-							
1w1	1;1/-	1;1/-	1;1/-							
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.110: Heat Map with computations Defect-9 Configuration 3

0										
1										
0w0										
0w1										
0r0										
0a0										
0o0										
1w0										
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.111: Heat Map with computations Defect-9 Configuration 4

0										
1										
0w0										
0w1										
0r0										
0a0										
0o0										
1w0										
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 10 (Tables 4.112 - 4.115) causes faults in computation for operations 1o1 and 1a1. For operation 1o1, the faults are seen when the other operand for computation is a logic '0'. The output for this logic operation (1 OR 0) should be '1'. However, as the gate is not open in the transistor due to the defect, the cell containing logic state '1' does not contribute towards the calculation of the logic and hence a fault is registered. Similarly, for 1a1 the defect causes faults when the other operand is in logic '1' and since the defective cell does not contribute towards logic calculation, an

incorrect computation is observed.

Table 4.112: Heat Map with computations Defect-10 Configuration 1

0									
1									
0w0									
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-					
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1						
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-				
1w1									
1r1				1;1/0 1/0					
1a1									
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.113: Heat Map with computations Defect-10 Configuration 2

0									
1									
0w0									
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-					
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1						
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-				
1w1									
1r1				1;1/0 1/0					
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1					
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.114: Heat Map with computations Defect-10 Configuration 3

0									
1									
0w0									
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-					
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1					
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-				
1w1									
1r1				1;1/0 1/0					
1a1									
1o1	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.115: Heat Map with computations Defect-10 Configuration 4

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0	0;0/1 0/1	0;0/1 0/1	0;0/1 0/1							
0a0										
0o0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1										
1r1				1;1/0 1/0						
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1						
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 11 (Tables 4.116 - 4.119) has a similar operation to defect 10 in the computation configuration. That is, the operation $1o1$ fail when the other operand is at logic state '0' and the operation $1a1$ fails when the other operand is at logic '1'. The reason is the same as for defect 10.

Table 4.116: Heat Map with computation Defect-11 Configuration 1

0										
1										
0w0										
0w1										
0r0									0/1	0/1
0a0										
0o0										
1w0										
1w1										
1r1										
1a1										
1o1								1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.117: Heat Map with computation Defect-11 Configuration 2

0										
1										
0w0										
0w1										
0r0									0/1	0/1
0a0										
0o0										
1w0										
1w1										
1r1										
1a1									1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.118: Heat Map with computation Defect-11 Configuration 3

0										
1										
0w0										
0w1										
0r0								0/1	0/1	0/1
0a0										
0o0										
1w0										
1w1										
1r1										
1a1										
1o1								1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.119: Heat Map with computation Defect-11 Configuration 4

0										
1										
0w0										
0w1										
0r0									0/1	0/1
0a0										
0o0										
1w0										
1w1										
1r1										
1a1									1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 12 (Tables 4.120 - 4.123) causes the R_{BL} arm of the sense amplifier to discharge faster due to the bypassing of the transistor by the defect. This causes the operation $1a1$ to fail when the other operand is in logic '0', as the output is a '1' because of the defect.

Table 4.120: Heat Map with computation Defect-12 Configuration 1

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
0a0										
0o0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0								
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.121: Heat Map with computation Defect-12 Configuration 2

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
0a0										
0o0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.122: Heat Map with computation Defect-12 Configuration 3

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
0a0										
0o0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0							
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.123: Heat Map with computation Defect-12 Configuration 4

0										
1										
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1										
0r0										
0a0										
0o0										
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 13 (Tables 4.124 - 4.127)as an extension of the read operation, the fault in the computation configuration of the defect causes the output of the computation to be at '0' if only the defective cell is open. Because of the defect however, the operation $0o0$ would be faulty and would result in an incorrect computation fault when the other operand is in logic state '1'. Similarly, incorrect computation fault occurs during operation $1a1$ while the other operand is in logic '1' and $1o1$ for both operand states in the complementary operand.

Table 4.124: Heat Map with computation Defect-13 Configuration 1

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0										
0a0										
0o0	0;0/0 0;1/0	0;0/0 0;1/0	0;0/0 0;1/0	0;0/0 0;1/0						
1w0										
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0	1;1/0	1;1/0	1;1/0						
1o1	1;1/0	1;1/0	1;1/0	1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.125: Heat Map with computation Defect-13 Configuration 2

0										
1										
0w0										
0w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
0r0										
0a0	0;1/0	0;1/0	0;1/0	0;1/0						
0o0	0;1/0	0;1/0	0;1/0	0;1/0						
1w0										
1w1	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-	0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1					
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.126: Heat Map with computation Defect-13 Configuration 3

0										
1										
0w0	0;U/-	0;U/-	0;U/-	0;U/-						
0w1	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-						
0r0										
0a0										
0o0	0;0/0 0;1/0	0;0/0 0;1/0	0;0/0 0;1/0	0;0/0 0;1/0						
1w0	0;U/-	0;U/-	0;U/-	0;U/-						
1w1	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0	1;1/0	1;1/0	1;1/0						
1o1	1;1/0	1;1/0	1;1/0	1;1/0	1;0/0 1;1/0					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.127: Heat Map with computation Defect-13 Configuration 4

0										
1										
0w0	0;U/-	0;U/-	0;U/-	0;U/-						
0w1	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-	0;0/- 0;U/- 0/-						
0r0										
0a0	0;1/0	0;1/0	0;1/0	0;1/0						
0o0	0;1/0	0;1/0	0;1/0	0;1/0						
1w0	0;U/-	0;U/-	0;U/-	0;U/-						
1w1	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-	0;U/- 0;0/- 0/-						
1r1	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0	1;1/0 1/0					
1a1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1					
1o1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1						
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

No faults were observed with defect 14 (Tables 4.128 - 4.131) in computation configuration, as the select line is designed to be naturally at GND during computation.

Table 4.128: Heat Map with computation Defect-14 Configuration 1

0									
1									
0w0	0;1/-	0;1/-	0;1/-	0;1/-					
0w1	1;1/-	1;1/-	1;1/-	1;1/-					
0r0									
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-					
1r1									
1a1									
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.129: Heat Map with computation Defect-14 Configuration 2

0									
1									
0w0	0;1/-	0;1/-	0;1/-	0;1/-					
0w1	1;1/-	1;1/-	1;1/-	1;1/-					
0r0									
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-					
1w1	1;1/-	1;1/-	1;1/-	1;1/-					
1r1									
1a1									
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.130: Heat Map with computation Defect-14 Configuration 3

0									
1									
0w0									
0w1									
0r0									
0a0									
0o0									
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-					
1w1									
1r1									
1a1									
1o1									
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 15 (Tables 4.132 - 4.135) behaves in the same way as defect 11 and defect 10 as the resultant faults for these defects are similar.

Table 4.131: Heat Map with computation Defect-14 Configuration 4

0										
1										
0w0										
0w1										
0r0										
0a0										
0o0										
1w0	1;1/- 1/-	1;1/- 1/-	1;1/- 1/-	U;U/- U/-						
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.132: Heat Map with computation Defect-15 Configuration 1

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
0a0										
0o0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
1a1										
1o1						1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.133: Heat Map with computation Defect-15 Configuration 2

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
0a0										
0o0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
1a1						1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 16 (Tables 4.136 - 4.139) causes incorrect computation faults and computation read destructive faults as all the cells are flipped to state 'L'.

Table 4.134: Heat Map with computation Defect-15 Configuration 3

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
0a0										
0o0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
1a1										
1o1						1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0	1;0/0 1;1/0
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.135: Heat Map with computation Defect-15 Configuration 4

0										
1										
0w0										
0w1							0/-	0/-	0/-	0/-
0r0										
0a0										
0o0										
1w0							1/-	1/-	1/-	1/-
1w1										
1r1					1/0	1/0	1/0	1/0	1/0	1/0
1a1						1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1	1;1/0 1;0/1
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.136: Heat Map with computation Defect-16 Configuration 1

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
0a0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	1;1/0				
0o0	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/1	1;1/1				
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
1a1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	1;1/0	1;1/1 1;1/0			
1o1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	1;1/1	1;1/1			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.137: Heat Map with computation Defect-16 Configuration 2

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
0a0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
0o0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/1 L;0/0	1;1/1 1;0/0	1;1/1 1;0/0			
1w0	L/-	L/-	L/-	L/-	L/-	L;1/- L;1/- L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
1a1	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	1;1/0 1;0/1				
1o1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;0/1					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.138: Heat Map with computation Defect-16 Configuration 3

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
0a0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	1;1/0				
0o0	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/1	1;1/1				
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
1a1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	L;1/0 L;1/1	1;1/0	1;1/1 1;1/0			
1o1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	L;0/0 L;1/0 L;1/1	1;1/1	1;1/1			
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.139: Heat Map with computation Defect-16 Configuration 4

0	L/-	L/-	L/-	L/-	L/-					
1	L/-	L/-	L/-	L/-	L/-					
0w0	L/-	L/-	L/-	L/-	L/-	L/-				
0w1	L/-	L/-	L/-	L/-	L/-					
0r0	L/0	L/0	L/0	L/0	L/0	1/0	1/1			
0a0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	1;1/0 1;0/0	1;1/1 1;0/0			
0o0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/1 L;0/0	1;1/1 1;0/0	1;1/1 1;0/0			
1w0	L/-	L/-	L/-	L/-	L/-	L/-				
1w1	L/-	L/-	L/-	L/-	L/-					
1r1	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	L;1/0 L/0	1/0				
1a1	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	L;1/0 L;0/0	1;1/0 1;0/1				
1o1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;1/0 L;0/1	L;0/1					
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Defect 17 (Tables 4.140 - 4.143) causes the same faults as defect 12 as logically, these defect behave the same for computation configurations. That is, the operation $1a1$ causes an incorrect computation fault when the other operand is '0'. This fault occurs for lower defect resistance ranges of $R_{\text{Def}} \leq 60\,\Omega$.

Table 4.140: Heat Map with computation Defect-17 Configuration 1

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
0a0										
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0							
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.141: Heat Map with computation Defect-17 Configuration 2

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
0a0										
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.142: Heat Map with computation Defect-17 Configuration 3

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
0a0										
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1	1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0							
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Table 4.143: Heat Map with computation Defect-17 Configuration 4

0										
1										
0w0	1/-	1/-	1/-	1/-	1/-	U/-				
0w1										
0r0										
0a0										
0o0										
1w0	1/-	1/-	1/-	1/-	1/-	U/-				
1w1										
1r1										
1a1										
1o1										
R_{Def}	1	7.7	6.01e+01	4.64e+02	3.593e+03	2.7825e+04	2.15e+05	1.66e+06	1.29e+07	1e+08

Forming Defect causes the defects that are similar to the memory configuration. The faults are easy to detect in the operations $1a1$ and $1o1$ as the resistance is too high which would always result in the sense amplifier giving the output '0'. The operation $1a1$ operation causes a fault with the computation with the other operand in state '0', which is due to the resistance of the cell being high enough that it causes a sense amplifier fault but is still registered as state '1'.

Table 4.144: Heat Map with computation Forming defect Configuration 1

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																				U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																							U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																				U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
0a0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0																				U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0
0o0	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1																				U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	H/-	H/-	H/-	H/-	H/-
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1																H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
1a1	L;1/1	L;1/1	0;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1									1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1
1o1	L;0/0 L;1/1	L;0/0 L;1/1	0;0/0 0;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1																H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1
l_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.145: Heat Map with computation Forming defect Configuration 2

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																				U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																						U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																				U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
0a0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0																				U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0
0o0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0																				U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	1/-	H/-	H/-	H/-	
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1																H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
1a1	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0 0;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0																H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0
1o1	L;0/1	L;0/1	0;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1																H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1
l_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.146: Heat Map with computation Forming defect Configuration 3

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																				U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																						U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																				U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
0a0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0	L;1/0																				U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0	U;1/0
0o0	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1	L;1/1																				U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	1/-	H/-	H/-	H/-		
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1																H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
1a1	L;1/1	L;1/1	0;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1									1;0/1 1;1/0	1;0/1 1;1/0	1;0/1 1;1/0					H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1
1o1	L;0/0 L;1/1	L;0/0 L;1/1	0;0/0 0;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1	U;1/1																H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1	H;1/1
l_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Table 4.147: Heat Map with computation Forming defect Configuration 4

0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																				U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-	L/-																						U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-
0w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
0r0	L/1	L/1	0/1	L/0	L/0	L/0	L/0	L/0																				U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0	U/0
0a0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0																				U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0
0o0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0	L;0/0																				U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0
1w0	L/-	L/-	L/-	L/-	L/-	L/-	L/-												U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	H/-	U/-	1/-	H/-	H/-	H/-
1w1	L/-	L/-	0/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-	U/-																H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-	H/-
1r1	L/1	L/1	0/0	U/0	U/0	U/0	U/1	U/1	U/1	U/1	U/1	U/1																H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/1	H/0
1a1	L;1/0 L;0/0	L;1/0 L;0/0	0;1/0 0;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0	U;0/0																H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0	H;0/0
1o1	L;0/1	L;0/1	0;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1	U;0/1																H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1	H;0/1
l_{form}	1e-6	2e-6	3e-6	4e-6	5e-6	6e-6	7e-6	8e-6	9e-6	10e-6	11e-6	12e-6	13e-6	14e-6	15e-6	16e-6	17e-6	18e-6	19e-6	20e-6	21e-6	22e-6	23e-6	24e-6	25e-6	26e-6	27e-6	28e-6	29e-6	30e-6	31e-6	32e-6	33e-6	34e-6	35e-6

Fault Modelling and Sensitized Fault space: Peripherals

Address Decoder Fault modeling for address decoder is similar to that in dual port memories[23]. This is due to the simultaneous access in the memory array. Faults seen the dual port memories apply here as well, as these circuits have the same structure. These faults are seen in the dual-port memories due to interaction between the two decoders are called *port interference faults*. Figure 4.18 gives the fault locations in the address decoders for two port memories[23].

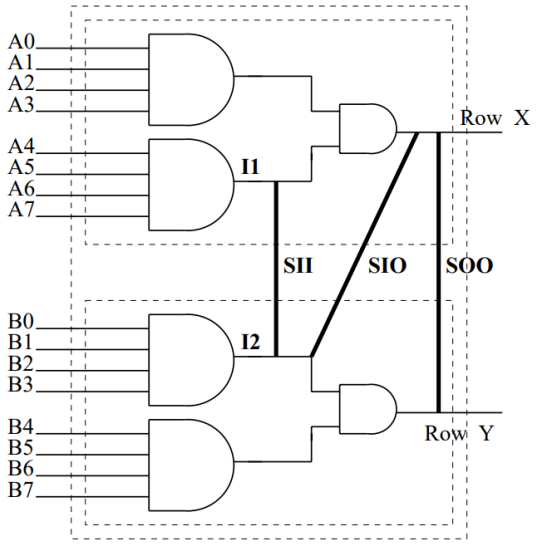


Figure 4.18: Fault location in address decoder for two port memory
Fault location in address decoder for two port memory [23]

I1 and I2 are internal wires and Row X and Row Y are representative of the lines that drive the lines in the memory array. There are three kinds of unique electrical behaviour that can occur in these circuits. These are short between output lines (SOO), short between an internal and output line (SIO) and Short between two internal lines (SII). Each of these cases there can be faults present or absent based on the logical values in these lines. The faults that arise due to these connections have been studied by Hamdioui and Van De Goor [23]. They are classified as faults that make use of only one address decoder for their sensitization and faults that make use of both of the decoders to be sensitized. The former are the faults in memory configuration and will be tested in the memory configuration. The latter need the accessing by two address decoders in order to be detected. These faults are described in Table 4.148.

Table 4.148: Unique defects in two port memory Address decoders

S.No	Short Type	Fault
1	SOO	Column X is selected by AD 1 and Column Y is not selected by AD 2, but a cell in column x and column y are selected by AD 1 and AD 2 respectively
2	SIO (Without inversion)	Column X is selected by AD 1 and Column Z is selected by AD 2, but a cell in column X is selected by AD 1 and cells in column Z and Y are selected by AD 2, where Z≠Y
3	SIO (With inversion)	Column X is not selected by AD 1 and Column Z is selected by AD 2, cells in column Z and Y are selected by AD 2, where Z≠Y

Sense Amplifier Unique faults in the sense amplifier are caused by switching between the references to perform logic operations. One unique fault has been identified in the scouting logic sense amplifier is the *Wrong reference fault* where the fault is caused by the wrong reference being selected in the sense amplifier. For example, an OR operation could take the place of an AND operation. This can also happen in memory configuration, but this is captured as an SASF. Apart from the unique faults, the faults that occur in the memory configurations are also applicable to the computation configuration.

The Unique faults in the sense amplifier can occur due to a number of reasons: incorrect gate signals, short between the two reference resistances and partial opening and closing of the gates in the sense amplifier arm (shown in Figure 4.12. These faults could also be detrimental to the memory operations, as they can cause read operations to fail as well in some cases. For example, if there is a bridge between the arms of the reference circuit, then the read operation would fail, as the reference is moved.

The fault analysis of the address decoder has been performed by Hamdioui and Van De Goor [23]. These address decoder faults have been well analysed and hence not repeated. Fault analysis of the sense amplifiers and the validation of their faults is an open question, as it requires the lay-out data of the sense amplifiers to reliably predict fault locations in the circuit and simulate the defects in the sense amplifier. This method is preferred over injecting resistive defects between all possible nodes, as it grows to have a large amount of simulations, which are impractical for our purposes.

5

Tests for Scouting Logic

This chapter briefs about the tests that are required for the faults identified in Chapter 5. The first section gives the tests for the components in their memory configurations and the second section gives the test for the computation configuration.

5.1. Tests in Memory Configuration

The final step in test development is the development of tests for the faults that were identified in Section 4.3.1 and 4.3.2. The tests should be performed for the memory configuration first, followed by the computation configuration. The tests in the memory configuration are presented for the three different components discussed in Section 4.3.1 are discussed below. These are based on the test development methodology discussed in Section 3.3.

5.1.1. Memory array

The memory array, if containing defects, can sensitize both ETD and HTD faults. Different testing strategies have to be employed for these faults, which are explained below:

ETD Faults

ETD faults are tested using their sensitizing sequence, which is given by the S component in their FP. The tests are verified against the detection sequence from the F and R component in the FP. It is worth nothing that a write sequence is always followed by a read sequence in order to detect a fault in the memory cell. For example, $\langle 0w1/0/- \rangle$ is tested using a March algorithm that contains a march element given by $\Updownarrow$ (r0,w1,r1). The r0 element is used to detect any faults in the cell prior to this march element.

HTD Faults

HTD faults in the memory array cells containing ReRAM devices are generally related to faults in which the state of the device is in one of the forbidden states (i.e., 'L', 'H' or 'U') [17]. The detection of these defects require DfTs in the system. Hamdioui et al. have presented two schemes programmable DfT schemes that can identify these defective cells [25]. Since the state of the ReRAM cell is determined by the *duration* and *strength* of the input flux, the two DfT schemes are developed based on short write time (varying duration factor) and low write voltage (varying strength factor) [25]. These DfT concepts are explained below:

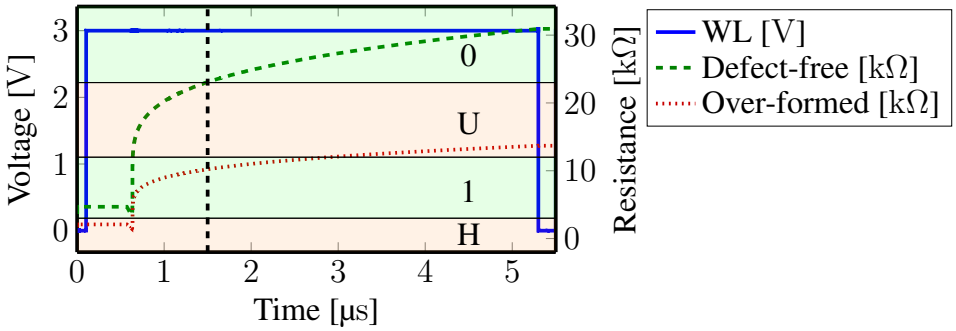


Figure 5.1: Defect-free and over-formed cell

- **Short Write Time-based DfT** makes use of the write access time. It is the time for which the memory device has to be accessed for a normal write operation. If the access time is not sufficient enough, then the state change from '0' to '1' will not take place in the device. However if the device was already in the Undefined state 'U', there will be a change in state even at a reduced access time. This is exploited with the Short write time scheme. Here a cell is subjected to w1 operation followed by a weak write operation ($\widehat{w}0$) where a shorter access time is given for the write operation. This is followed by a r1 operation. The cells that have defects would read a value '0' instead of '1', thus detecting the fault. if the cell is fault free, it would not change in state. This is repeated for a sequence containing ($\widehat{w}1$) in it.

- **Low Write Voltage-based DfT** makes use of the applied voltage value. This is the voltage drives the states of the cell. This is set at 3 V in our case. If the voltage is reduced the state of the cell will not change. But a cell that is already in the undefined state would have its state pushed to a defined state. This is exploited in this scheme. The cell is first written to 1 (w1) and then a weak write operation ($\widehat{w}0$) where a low write voltage is applied. If the cell suffers from a defect that puts it in the Undefined state, it would read a value of '0' instead of '1', and the fault will be detected. If the cell is fault free, it would not change in state. This is repeated for a sequence containing ($\widehat{w}1$)in it.

This thesis extends the scheme to also detect cells that suffer from the forming defect where this is over-formed. In these cells, the state of the cell would be in 'H' instead of '1', as shown in Figure 5.1. After a RESET operation is applied to the overformed and the Defect-free cell, they move to the '0' and 'U' state. This can then be captured by the DfT presented above to detect the over-formed cell that is currently in the undefined state. When a short write time based DfT is used and the write time is reduced to 1.5 µs as shown in Figure 5.1, an over-formed cell would remain in the undefined region and can be captured, whereas a defect free cell would flip to '0' state.

5.1.2. Address Decoder

The four static faults mentioned in 3.3.2 are ETD faults, while the two ADFs (ActD and DeActD) are HTD faults. ETD faults are detected using a March test which contains the following two march elements[71][20]: $\Uparrow (rx,\ldots,w\bar{x})$ and $\Downarrow (r\bar{x},\ldots,wx)$; here, $x \in \{0,1\}$ and $\bar{x}$ denotes the negation of x. These march elements guarantee the detection of the ETD faults [54].

The address decoder delay faults (ADFs) may be detected with the help of march tests, but this is not always guaranteed. ADFs are caused by opens in the wires in which the defect strength are in the intermediate strength. The detection probability is strongly depends on the delay that is seen in the signals [30]. The stronger the defect is, the larger the delay is in the circuit, thus making it easier to identify the fault. There are two fundamental requirements for the detection of ADFs:

1. *Sensitizing address transitions* (SATs): These can be caused by an address pair or an address triplet [30]. A *sensitizing addressing pair*(SAP) consists of a sequence of two addresses, say A^{f} and A_{g}, which cause an ADF in the circuit. When two SAPs are applied in sequence, they can be combined to form a triplet for more efficiency.

2. *sensitizing operation sequences* (SOSs): To each address of a SAP or a SAT at least one operation has to be applied, which gives rise to a sensitizing operation sequence(SOS). These have two operations for a SAP with two addresses and three operations for a SAT with three transitions.

The addresses of the SAPs/SATs are generated using *Addressing Methods*(AMs). Some of these methods used are discussed below:

1. *Binary AM* [30] where the addresses are accessed in an increasing manner. For example, number of address lines $N = 3$, a binary AM would consists of the address sequences $\Uparrow^{\mathrm{Bin}} = \{0,1,2,3,4,5,6,7\}$ and $\Downarrow^{\mathrm{Bin}} = \{7,6,5,4,3,2,1,0\}$.

2. *Address Complement AM* (AC AM) [30] where the addresses are accessed in such a way a transition from either $x \rightarrow \bar{x}$ or $\bar{x} \rightarrow x$ are generated. for example for $N = 3$, the sequence is given by $\Uparrow^{\mathrm{AC}} = \{000,111,001,110,010,101,011,100\}$ and $\Downarrow^{\mathrm{AC}} = \{100,011,101,010,110,001,111,000\}$. Here each address is followed by its one's complement.

3. *2^i AM* [30] where the addresses are accessed using a binary counting sequence, with increments or decrements of $j = 2^i$, where $0 \leq i \leq N-1$. The sequence "0, 1, 2, 3, 4, 5, 6, 7" is the $\Uparrow^0$ sequence for $j = 1$ and "7, 6, 5, 4, 3, 2, 1, 0" is for $\Downarrow^0$ sequence.

4. *H1 AM* [30] where the addressing is generated as minimal set of SATs where the hamming distance between the addresses is 1. For example, for the 3-bit code word "000", the SATs generated are: {000, 001, 000}, {000, 010, 000} and {000, 100, 000}. These SATs combine to form a *SuperSAT*: {000, 001, 000, 010, 000, 100, 000}.

These AMs are used to detect ActDs and DeactDs in the circuit based on the defects that are present in them.

On the other hand, the Sensitizing Operation Sequences should consist of two operations $Ox_f; Oy_g$, where one operation applied to A_{f} and the other to A_{g}. O denotes a read or write operation ($O \in \{r, w\}$. The expected or written data is denoted by x and y, such that $x, y \in \{1,0\}$. The operation in one address of the SAP must be performed on data that is the complement of the data in the other address. This is a necessity if the fault to be detected, as a failure in this operation indicates the presence of a delay fault [30].

Since the scouting logic architecture has two separate decoders in the computing configuration that can be used to access the memory cells, we have to repeat the testing process for individually.

5.1.3. Sense Amplifier

The static faults in the sense amplifier are ETD faults and hence can be easily detected using March tests consisting of the two March elements: $\Updownarrow(\ldots, \mathrm{r}x, \ldots)$ and $\Updownarrow(\ldots, \mathrm{r}\bar{x}, \ldots)$, with $x \in \{1,0\}$. These March elements can also be combined to one March element for efficiency [20].

HTD faults in the sense amplifier consists of their dynamic faults, namely USAF and SSAF. These can be detected by March tests but it does not provide guaranteed detection. This is because of the varying defect resistances that can cause these faults. In order to detect them better, we make use of special DfT. The sensitization and detection of these faults makes use of *back-to-back* operations to

the memory. Back-to-back operations employ two operations that take place in sequence without any delay. These fast operations are achieved by two methods. First is storing different data values in cells in the same bit line and reading them from the sense amplifier. If the sense amplifier is slow or unbalanced, a wrong output is read. The second method makes use of fast-row addressing, where the row adjacent to the row being addressed is also active, which tests for the sense amplifier at the speed of switching the addressing the cells. For example, consider the March element $\Updownarrow$ (rx,...,w$\bar{x}$);. This can be used as the input sequence for the fast-row addressing to detect SSAFs. The read and write operations are performed back-to-back and use data that are complementary to each other. Special DfT that can work with the March tests have been researched, which are better at detecting these faults [10].

5.1.4. Test Sequences for Memory Configuration

In order to generate the test that can cover the maximum number of defects with highest defect strength with minimum cost, a reduction of the sensitizing sequences to incorporate the most coverage of the defects is required. Since we are developing production tests, we do not need to produce tests that can capture all the faults, but rather produce tests that can capture all the defects that could be in the circuit. The procedure for finding the test sequence for ETD faults in the array is listed below.

- First, we make a heat maps with each of the defects, for varying operands and background data. These heat maps give the strength of the defect injected and the operations that are performed on the defective cell. An example of this heat map table is shown in Figure 4.4.
- Next, from the heat maps, we identify the operations that have the maximum coverage of defect based on the defect strength for which each fault is activated. We only consider the ETD faults for this example. This would give us the set of all operations that could be used to generate a test that could detect the defects in the circuit.
- After this, the sequences are reduced to the least amount of sequences by eliminating overlaps in the sequences that are employed in the test sequence. For example if two defects can be detected by $1w1$, then we make use of this sequence instead of two different ones for each defect.
- From the above, a set of sequences that would be obtained. These are converted into march sequences which have minimum operation cost.

The above procedure can be modified to include the HTD faults as well. This would give an increased defect coverage due to the inclusion of tests to detect undefined state 'U', and the extreme states in the ReRAM namely 'L' and 'H'. These faults would be tested for with the help of DfT as mentioned.

Test for ETD Faults

The Table 5.1 was constructed using the steps mentioned above. It gives the list of sensitizing sequences that have the maximum defect coverage for each defect and do not contain HTD faults. From this table, the test sequence is generated. The test sequence should consist of the operations $1r1$, $0r0$, $0w1$ and $1w0$. Based on these operations and the addressing requirements for the address decoder and sense amplifier faults, the following test sequence is generated:

$$M1 \Updownarrow (w1)\ ;\ M2 \Uparrow (r1, w0)\ ;\ M3 \Downarrow (r0, w1)\ ;\ M4 \Updownarrow (r1)$$

This sequence can test for ETD defects in both Sense amplifier and address decoder as well, as it covers the required test pattern for those peripherals. It should be noted that since there are two address decoders present in the circuit, there needs to be a repetition in place for testing both the address decoders. The test time is $6n$ where n is the time taken for an operation. The Table 5.2 gives the defect against the march elements that sensitize them and helps to identify the defects.

Table 5.1: Sensitized FP with Maximum Coverage - ETD

Defect Number	0	1	0w0	0w1	0r0	1w0	1w1	1r1
1					⟨0r0/1/0⟩			⟨1r1/0/1⟩
2					⟨0r0/0/1⟩			
3					⟨0r0/0/1⟩			
4					⟨0r0/1/0⟩			⟨1r1/0/1⟩
5					⟨0r0/0/1⟩			
6						⟨1w0/1/−⟩		
7				⟨0w1/0/−⟩				
8								⟨1r1/0/1⟩
9			⟨0w0;0/0;1/−⟩	⟨0w1;0/1;1/−⟩		⟨1w0;0/0;1/−⟩	⟨< 1w1;0/1;1/−⟩	
10				⟨0w1/0/−⟩		⟨1w0/1/−⟩		
11					⟨0r0/0/1⟩			
12								
13								⟨1r1/0/1⟩
14			⟨< 0w0;0/0;1/−⟩	⟨< 0w1;0/1;1/−⟩		⟨1w0/1/−⟩	⟨1w1;0/1;1/−⟩	
15								⟨1r1/1/0⟩
16					⟨0r0/1/0⟩			⟨1r1/L/0⟩
17			⟨0w0/1/−⟩			⟨1w0/1/−⟩		
Forming		⟨1/L/−⟩		⟨0w1/L/−⟩			⟨1w1/L/−⟩	⟨1r1/L/1⟩

Defect Number	**Sensitizing March Element**	**Detecting March Element**
Defect 1	M3, M4	M3, M4
Defect 2	M3	M3
Defect 3	M3	M3
Defect 4	M3, M4	M3, M4
Defect 5	M3	M3
Defect 6	M2	M3
Defect 7	M3	M4
Defect 8	M2, M4	M2, M4
Defect 9	M2, M3	M3, M4
Defect 10	M2,M3	M3, M4
Defect 11	M3	M3
Defect 12	-	-
Defect 13	M2, M4	M2, M4
Defect 14	M2, M3	M2, M4
Defect 15	M2, M4	M2, M4
Defect 16	M3	M3
Defect 17	M3	M3
Forming Defect	M3, M4	M4

Table 5.2: Defect Coverage in ETD test sequence - Memory configuration

Test for HTD Faults

The Table 5.3 extends the memory configuration test for HTD faults with using the same procedure. We take into consideration which HTD fault is activated by these operations, so as to modify the test sequence for the HTD faults. We make use of $\hat{w}0$ and $\hat{w}1$ operations to identify undefined states in the cells in the March test for memory configuration with HTD given below:

$$M1 \Updownarrow (w0, r0, w1, r1, w0)\ ;\ M2 \Uparrow (r0, w1, \hat{w}0)\ ;\ M3\ \Downarrow (r1, w0, \hat{w}1)\ ;\ M4\ \Updownarrow (r0)$$

The march element M1 in the test sequence shown above makes use of the back-to-back operations to detect HTD faults in the sense amplifiers. The HTD faults in the address decoder can be found by using the different addressing schemes explained in Section 5.1.2. The Table 5.4 gives the overview of coverage of the defects by the march elements. The time taken for the tests is $12n$ where n is the time taken for an operation.

Table 5.3: Sensitized FP with Maximum Coverage - HTD

Defect Number	0	1	0w0	0w1	0r0	1w0	1w1	1r1
1					⟨0r0/1/0⟩			⟨1r1/0/1⟩
2					⟨0r0/0/1⟩			
3						⟨1w0/U/−⟩		
4					⟨0r0/1/0⟩			⟨1r1/0/1⟩
5					⟨0r0/0/1⟩			
6						⟨1w0/U/−⟩		
7				⟨0w1/0/−⟩				
8								⟨1r1/0/1⟩
9			⟨< 0w0;0/0;1/−⟩	⟨< 0w1;0/1;1/−⟩		⟨< 1w0;0/0;1/−⟩	⟨< 1w1;0/1;1/−⟩	
10						⟨1w0/U/−⟩		
11					⟨0r0/0/1⟩			
12			⟨0w0/L/−⟩			⟨1w0/L/−⟩		
13								⟨1r1/0/1⟩
14			⟨< 0w0;0/0;1/−⟩	⟨< 0w1;0/1;1/−⟩		⟨1w0/1/−⟩	⟨1w1;0/1;1/−⟩	
15								⟨1r1/1/0⟩
16					⟨0r0/1/0⟩			⟨1r1/L/0⟩
17			⟨0w0/U/−⟩			⟨1w0/U/−⟩		
Forming		⟨1/U/−⟩		⟨0w1/U/−⟩		⟨1w0/U/−⟩	⟨1w1/U/−⟩	⟨1r1/U/1⟩

5.2. Tests in Computation configuration

Here, we discuss the tests that need to be performed in the computation configuration. The faults which would be tested for were discussed in Section 4.3.2. The approach in this configuration for test development is similar to one in the memory configuration.

5.2.1. Memory Array

The tests in computing configuration of Scouting logic depends on the faults that are seen in the same configuration. These faults are also classified as ETD and HTD faults and can be tested in the same way in the memory configuration. There can however be faults that are unique to this configuration which have to be tested. For example, a resistance in the BL between two memory cells increases the overall resistance of the BL. This can cause an change in the effective resistance that determines the logic operation in Scouting logic. The total resistance of the setup can increase

Defect Number	Sensitizing March Element	Detecting March Element
Defect 1	M1, M2, M3, M4	M1, M2, M3, M4
Defect 2	M1, M2, M3, M4	M1, M2, M3, M4
Defect 3	M3	M4
Defect 4	M1, M2, M3, M4	M1, M2, M3, M4
Defect 5	M1, M2, M3, M4	M1, M2, M3, M4
Defect 6	M3	M4
Defect 7	M1, M2	M1, M3
Defect 8	M1, M3	M1, M3
Defect 9	M1, M2, M3, M4	M1, M2, M3, M4
Defect 10	M3	M4
Defect 11	M1, M2, M3, M4	M1, M2, M3, M4
Defect 12	M3	M4
Defect 13	M1, M2, M3, M4	M1, M2, M3, M4
Defect 14	M1, M2, M3, M4	M1, M2, M3, M4
Defect 15	M1, M3	M1, M3
Defect 16	M1, M2, M3, M4	M1, M2, M3, M4
Defect 17	M3	M4
Forming Defect	M2, M3	M3, M4

Table 5.4: Defect Coverage in HTD test sequence - Memory configuration

so as to cause a fault where the OR operation results in a logic '0' even if there is a '1' in one of the operands. The DfTs can help in identifying faulty memristive devices, for example when the result of the computation operation gives a random output for every trial. This behaviour is seen when one of the cells is in the undefined 'U' state, and thus a short write time-based or low write voltage-based DfT can detect these faulty cells and thereby the defect that causes them.

5.2.2. Address Decoder

The Address decoder structure for the computation configuration for scouting logic resembles the address decoder circuit for dual-port memory systems as mentioned in Section 4.3.2 [23]. The tests that are developed for the those memories can be easily adapted and used for testing unique address decoder faults that were discussed. The test that is given by Hamdioui and Van De Goor, shown in Figure 5.2, covers all the unique faults for the two port memories [23]. It has a complexity of $\mathcal{O}(R^2)$, where R is the number of rows in the array.

5.2.3. Sense Amplifier

A sense amplifier in the computation configuration has the same faults as one in the memory configuration. Testing such faults in the computation configuration, however, requires special attention. This is because of the additional complexities in the rest of the circuit that add to the testing of the sense amplifier. For example, in order to test a stuck-at sense amplifier fault, we need to access two rows simultaneously. This is achieved by combining two March elements: $\Updownarrow_{c=0}^{C-1}$ (..., $\mathrm{r0}_i : \mathrm{r}x_j$, ...) and $\Updownarrow_{c=0}^{C-1}$ (..., $\mathrm{r1}_i : \mathrm{r1}_j$, ...), where $x \in \{0,1\}$ and (i, j) are addresses indicating any two cells/words in the same column. For example the fault SASF-1 will be detected by the operations $\mathrm{r0}_i : \mathrm{r0}_j$ as this will return 1 instead of 0 in the presence of a short to VDD. The sequences can be stopped when the first fault has been detected in the sense amplifier and does not have to be performed for all i and j. Slow sense amplifier faults can be identified with back-to-back operations men-

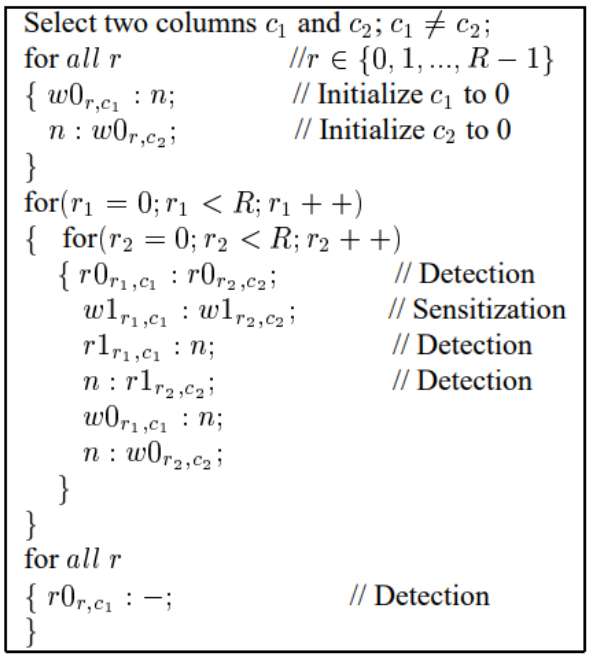


Figure 5.2: Test Unique faults in 2 Port Memories

tioned in Section 5.1.3, with the changes being made in the operation sequence. The sequence $\Uparrow_{c=0}^{C-1}$ (r1_i : r1_{i+1},...,r0_i : rx_{i+1}) has to be used in order to have several back-to-back AND operations in the same bit line. Here, the cells should be able to produce a logic value of '0' and '1' in the same bit line, so it is a necessity that at least one cell in the bit line has the logic value '0' stored in it. DfTs can also be developed for these configurations to obtain better results. Development of these DfTs is an open question.

Test for ETD Faults

We develop the test for computation configuration with ETD faults using the same method mentioned in Section 5.1.4. This test makes use of two rows at the same time, and hence special notations are added to the regular march test elements for completeness. We address the cells in the second row of computation using the symbol $\Updownarrow_{n+1}$. This symbol instructs to address all the second rows in the array and would be used in conjunction with read and write operations. In order to explicitly mention the logic operation, we make use of the notation in the extended fault primitive for computation configurations. For example, an operation $\Uparrow (o1_1 : o1_2)$ indicates an OR operation between two cells indicated by 1 and 2 respectively.

The Table 5.5 gives the list of all the faults that are sensitized which have the highest fault coverage amongst all the other defects. We also cover the memory configuration operations as they are an extension of the computation configuration. Table 5.6 gives the march element that sensitizes the fault and the march element that identifies the fault in tabular form. While constructing the test sequence, we replace the operation $r0$ with $o0_1 : o0_2$ and operation $r1$ with $a1_1 : a1_2$. This is possible by virtue of the scouting logic architecture and helps to speed up the test process in the system, as twice the amount of cells are verified in the same time. Based on this and the Table XXX, the following test sequence is generated:

$$M1\ \Updownarrow (w1)\ ;\ M2\ \Downarrow_2\ (a1_1 : a1_2)\ ;\ M3\ \Updownarrow_2\ (w0);\ M4\ \Downarrow_2 (a1_1 : a0_2)\ ; M5\ \Downarrow_2 (o1_1 : o0_2)$$

The March elements 2 and 4 act as read operations as well, as they perform read 0 on both cells

and read 1 and read 0 on the coupled cells. This improves the speed of operation. Along with this test, the tests mentioned for the address decoder faults for dual memories should also be performed to get the required result. The time taken for testing ETD faults with the given test sequence is $5n$ where n is the time taken per operation.

Table 5.5: Sensitized FP with Maximum Coverage for Computation Configuration - ETD

Defect Number	0	1	0w0	0w1	0r0	1w0	1w1	1r1	0a0	0o0	1a1	1o1
1											⟨1a1 : 1a1/1 : 1/0⟩	
2									⟨0a0 : 1a1/0 : 1/1⟩			
3									⟨0a0 : 1a1/0 : 1/1⟩			
4									⟨0a0 : 1a1/0 : 1/1⟩	⟨0o0 : 0o0/0 : 0/1⟩		
5									⟨0a0 : 1a1/0 : 1/1⟩			
6												⟨1o1 : 0o0/1 : 0/0⟩
7												⟨1o1 : 0o0/1 : 0/0⟩
8												⟨1o1 : 0o0/1 : 0/0⟩
9				⟨0w1; 0/1; 1/−⟩								
10											⟨1a1 : 1a1/1 : 1/0⟩	⟨1o1 : 0o0/1 : 0/0⟩
11												⟨1o1 : 0o0/1 : 0/0⟩
12											⟨1a1 : 0a0/1 : 0/1⟩	
13											⟨1a1 : 1a1/1 : 1/0⟩	⟨1o1 : 0o0/1 : 0/0⟩
14						⟨1w0/1/−⟩						
15								⟨1r1/1/0⟩				
16											⟨1a1 : 0a0/1 : 0/1⟩	
17			⟨0w0/1/−⟩			⟨1w0/1/−⟩						
Forming						⟨1w0/H/−⟩						⟨1o1 : 0o0/H : 0/0⟩

Test for HTD Faults

We finally develop the test for the HTD faults in the computation configuration. Since the tests for HTD faults rely on single cells being activated, we do not make use of the combined reads to find the HTD faults. However, we do use the those operations for finding ETD faults with the test sequence. The Table 5.7 gives the list of sensitized faults for HTD in the computing configuration. The following test sequence was generated as a result.

$$M1\ \Updownarrow(w1)\ ;\ M2\ \Downarrow_2\ (a1_1 : a1_2)\ ;\ M3\ \Downarrow_1\ (w0);\ M4\ \Downarrow_1 (a0_1 : a1_2);$$
$$M5\ \Downarrow_1 (w0); M6\ (\hat{w}1, 0o0_1 : 0o0_2)\ ; M7\ \Downarrow_2 (w1)\ ;\ M8\ \Downarrow_2\ (\hat{w}0, 0o0_1 : 1o1_2)$$

Here March sequence M4 contains the HTD test where the undefined state has to be tested for. This would requires special DfT. The Address decoder HTD can be tested with one of the methods introduced earlier and the sense amplifier test can be performed by using the sequence for the HTD fault in the memory configuration. The $\hat{w}1$ operation should test for both states 'L' and 'U' which have to be performed separately, as the timing for each of these short write operations would be different. The time taken for the tests is $10n$ where n is the time taken per operation. Table 5.8 gives the coverage of the defects injected, with the march element that sensitizes the fault and identifies the fault.

Defect Number	Sensitizing March Element	Detecting March Element
Defect 1	M2	M2
Defect 2	M4	M4
Defect 3	M4	M4
Defect 4	M4	M4
Defect 5	M4	M4
Defect 6	M5	M5
Defect 7	M5	M5
Defect 8	M5	M5
Defect 9	M1	M2
Defect 10	M2, M5	M2, M5
Defect 11	M5	M5
Defect 12	M4	M4
Defect 13	M2, M5	M2, M5
Defect 14	M3	M4
Defect 15	M1	M2
Defect 16	M4	M4
Defect 17	M1	M2
Forming Defect	M1, M5	M4, M5

Table 5.6: Defect Coverage in ETD test sequence - Computation Configuration

Table 5.7: Sensitized FP with Maximum Coverage for Computation Configuration - HTD

Defect Number	0	1	0w0	0w1	0r0	1w0	1w1	1r1	0a0	0o0	1a1	1o1
1											⟨1a1 : 1a1/1 : 1/0⟩	
2									⟨0a0 : 1a1/0 : 1/1⟩			
3						⟨1w0/U/−⟩						
4									⟨0a0 : 1a1/0 : 1/1⟩	⟨0o0 : 0o0/0 : 0/1⟩		
5									⟨0a0 : 1a1/0 : 1/1⟩			
6												⟨1o1 : 0o0/1 : 0/0⟩
7												⟨1o1 : 0o0/1 : 0/0⟩
8												⟨1o1 : 0o0/1 : 0/0⟩
9				⟨0w1;0/1;1/−⟩								
10						⟨1w0/U/−⟩						
11												⟨1o1 : 0o0/1 : 0/0⟩
12			⟨0w0/L/−⟩									
13											⟨1a1 : 1a1/1 : 1/0⟩	⟨1o1 : 0o0/1 : 0/0⟩
14						⟨1w0/1/−⟩						
15								⟨1r1/1/0⟩				
16						⟨1w0/L/−⟩						
17			⟨0w0/U/−⟩			⟨1w0/U/−⟩						
Forming						⟨1w0/U/−⟩						

Defect Number	Sensitizing March Element	Detecting March Element
Defect 1	M2	M2
Defect 2	M4	M4
Defect 3	M5	M6
Defect 4	M4, M6	M4, M6
Defect 5	M4	M4
Defect 6	M8	M8
Defect 7	M8	M8
Defect 8	M8	M8
Defect 9	M1	M2
Defect 10	M5	M6
Defect 11	M8	M8
Defect 12	M7	M8
Defect 13	M2, M8	M2, M8
Defect 14	M5	M6
Defect 15	M2	M2
Defect 16	M5	M6
Defect 17	M5, M7	M6, M8
Forming Defect	M5	M6

Table 5.8: Defect Coverage in HTD test sequence - Computation Configuration

6

Conclusions

This chapter concludes the thesis. First, the summaries of each of the previous chapters are presented. then we discuss the implications, shortcomings. Finally potential future research related to the thesis is presented.

6.1. Summary

This section summarizes every chapter in the thesis

Chapter 1 introduces the thesis, gives the motivation and sets the context for the rest of the thesis. Problems faced with the advancement of computing towards high speed computing systems are presented as 'walls' in the computer architectures and in the underlying CMOS devices. The walls in CMOS technology are the reliability wall, leakage wall and cost wall, while the walls in traditional Von-Neumann architecture are the memory wall, power wall and ILP wall. CIM architectures are introduced as an alternative computing paradigm that can potentially solve some of the problems with traditional computing systems. These CIM architectures make use of memory elements that can double as a computing unit, thus eliminating costly data transfer in some cases. Memristive devices can be employed in these CIM architectures because of their non-volatility, low power consumption among other properties. The need for testing these architectures is then elaborated, followed by the state of the art in testing of these CIM architectures. Finally the contributions followed by the structure of this thesis are described.

Chapter 2 explains CIM architectures, including their structure, classification, operation paradigms (memory and computation configurations) and usage methods. Then, CIM architectures are classified based on the location where the results of computation are stored as either being CIM-A (results stored in array) or CIM-P (results obtained in the peripheral). Further classification is made based on the input method as being resistive or hybrid (input as both resistance and voltage). The basic working concept of memristive devices are explained, along with their different types, namely STT-MRAM, PCRAM, and ReRAM. These devices are focused because of their interesting properties that make them a better device to be used in CIM architectures. ReRAM is further explained, with their electrical behaviour and production process discussed.

Chapter 3 briefs about electronic testing, gives definitions for terms in electronic testing. The test methods employed in testing the circuits, namely functional and structural testing, are introduced. Structural testing is the method that should be followed while testing CIM architectures as it reduces the effort in testing while promising similar test coverage. Memory testing is discussed

in detail as CIM structures are tested as memory units since it is part of their function. Here we look also at the defects that occur in ReRAM memories and peripherals of a CIM architecture, the fault space of these structures and the tests that help us capture the faults in them. Then, it gives the methodology for testing CIM architectures and explains the reasoning behind the methodology. Memory configuration and computation configuration of CIM architectures are introduced and the importance of testing them separately is also discussed. We establish that the memory configuration is a subset of the computation configuration, and it has to be tested first because of the potential faults escaping the tests. This should be followed by the computation configuration tests. Both configurations have to be tested by structural testing method.

Chapter 4 exhibits the methodology proposed in Chapter 3 by taking the example of scouting logic. The experimental set-up used for the test development is explained. Then faults in memory configuration for the different components are discussed. Then the faults in the computation configuration for the components are discussed. Both of these sections exhibited the fault modeling and fault analysis of the defects in different components for each component. Fault primitives introduced in Chapter 3 are extended to incorporate faults that occur during computation operations.

Chapter 5 gives the tests that should be performed on the CIM architectures, in both memory and computation configuration. The tests are decided based on the ease of testability of the faults. The easy-to-detect (ETD) faults are tested with the help of March tests while hard-to-detect (HTD) faults can be detected with March tests along with special tests, for example, with the help of DfTs, to ensure the detection of these faults. The tests for ETD and HTD faults in these configurations were presented.

6.2. Discussions

The following are the key points of discussion that are obtained from this thesis. CIM architectures have to be tested for ensuring quality products be delivered to users. The lack of a dedicated testing methodology for these architectures is a research field that should be looked into. This thesis serves as a primer for the research in that direction. While the methodology was tested for scouting logic, it can extended to other architectures as well.

- CIM architectures exist in two different configurations. These configurations arise from modification they have that enable transition from memory configuration to computer configuration and vice versa. It is of high importance to study these components which enable this change in the architecture. If CIM architectures are not considered to be in two different forms, it makes the testing approach very complex and may lead to erroneous tests to be generated.
- It is not enough to test the memory configuration alone, as there might be faults in the components that enable the computations. Similarly, the computing configuration alone cannot be tested, as there could be a fault with the read and write operations. Hence both the configurations have to be tested. This comes from the need for completeness of the test in terms of coverage of the defects that can occur in the CIM architecture, as seen in Section 3.4.
- The memory configuration has to be tested first as it is a subset of the computation configuration, and if the computation configuration is tested first then there might be a fault that is not identified in the computation configuration, that makes tests costly.
- Unique faults that could occur in the computation configuration due to the operations and the interaction of the circuit elements in this configuration have to be considered while generating the tests for the CIM architecture in the computation configuration. These faults can

affect the operation of not only the computation configuration, but also the memory configuration. Examples of these faults can be seen in Section 4.3.2.

- There can also be unique faults that occur in the CIM architecture because of its structure, for example parallel access of the memristive array cells in case of scouting logic 3.4.3. These faults have to be studied well. Faults in the peripherals in CIM architectures also exhibit faults that are characteristic of their usage and design, for example, the WRF in the sense amplifier (Section 4.3.2) and thus must be investigated.
- While known methods such as March tests can help detect the easy to detect faults in a device (Section 5.1.4), research has to be directed towards better test methods and DfTs that can detect HTD faults in the circuit.

6.3. Future Research

Based on the analysis in this thesis, the following recommendations are made for future research:

1. A combined test for the configurations can be created that is a combination of memory and computation operations that is a efficient of testing. For example, the march elements can combine both read, write and computation operations, thus increasing test efficiency.
2. The computation configuration operations can be used to speed up the fault detection in the memory configuration using their parallel access property in some CIM architectures like scouting logic. An AND operation between two cells that are in logic state '1' can find if the cells are stuck at '0'.
3. Improved models for the memristive devices will help in better detection of faults and test development.
4. DfT schemes that specifically target these CIM architectures should be developed to ensure better fault coverage.

A

Testing Scouting Logic-Based Computation-in-Memory Architectures

This appendix contains the paper submitted for presentation at European Test Symposium, Tallinn, Estonia 2020

Testing Scouting Logic-Based Computation-in-Memory Architectures

Moritz Fieback[1] Surya Nagarajan[1] Rajendra Bishnoi[1] Mehdi Tahoori[2] Mottaqiallah Taouil[1] Said Hamdioui[1]
[1]*Computer Engineering Laboratory*
Delft University of Technology
Delft, The Netherlands
{m.c.r.fieback, s.hamdioui}@tudelft.nl
[2]*Chair of Dependable Nano Computing*
Karlsruhe Institute of Technology
Karlsruhe, Germany

***Abstract*—Traditional CMOS-based von Neumann computing architectures are facing severe challenges to meet requirements of evolving applications such as ultra-low power edge computing. Therefore, new computer architectures are under investigation. One of these architectures is computation-in-memory (CIM) based on memristive devices. It performs computing operations in the memory itself, allowing for massive parallelism, and low energy consumption. One implementation of this architecture is based on Scouting logic; it requires the modification of a regular memory structure to allow the execution of logic operations within the memory. This paper discusses the fault models and test of such an architecture. It demonstrates that unique faults can occur during computing that cannot be detected by regular memory tests. Further, the paper demonstrates that memory faults are a subset of the computation faults, hence a test for the computation functionality will detect all memory faults. Finally, this paper shows that testing the computation functionality reduces the overall test time.**

***Index Terms*—test, computation-in-memory (CIM), in-memory computing, emerging memories, RRAM**

I. Introduction

Evolving applications such as big data require high performance computer architectures. Unfortunately, traditional CMOS-based von Neumann architectures have an increasing difficulty in providing this performance, due to their limited data throughput, limited instruction parallelism, and high leakage power consumption [1, 2]. To overcome these issues, new computer architectures are being investigated. One of them is computation-in-memory (CIM) based on memristive devices [3]. These architectures can either operate as a regular memory in the *memory configuration*, and as a computing device in the *computation configuration*. Because the computations take place in the memory, these architectures do not suffer from the memory bottleneck and allow for massive parallel execution of instructions. Further, the usage of memrisitve devices like resistive RAM (RRAM), spin-transfer-torque magnetic RAM (STT-MRAM), or phase-change memory (PCM), allows for dense memory structures that do not suffer from high leakage power consumption [2]. Scouting logic is an implementation of CIM that can perform binary logic operations in the memory itself [4]. To allow for the computation, changes are made to a regular memory structure, e.g., modifications to the sense amplifiers or address decoders. These changes introduce new faults and hence require new test solutions to detect them. Further, the usage of emerging memristive devices introduces new failure mechanisms that require different test methods to detect them compared to normal memory tests [5].

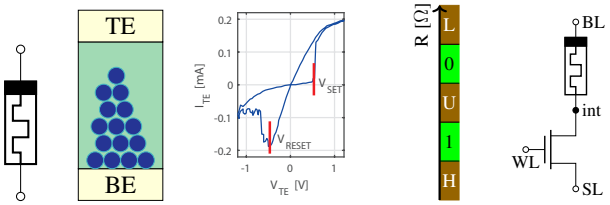
The testing aspects of CIM have not been well explored in the literature so far. Tsai *et al.* addressed testing of 8T-SRAM CIM [6]. The authors recognized that a CIM die needs to be tested in both its memory and computation configuration and presented test solutions for both of them. Emara *et al.* developed a test for memristor ratioed logic, focusing on transistor faults and memristive stuck-at faults [7]. However, it has been shown that stuck-at fault models do not represent all possible defects in a memristive device [8–10]. This is mainly caused by the fact that these models do not take into account the non-linear behavior of a defective device [8]. Therefore, a test based solely on these fault models will lead to test escapes. Hamdioui *et al.* presented the outlines for a structural test approach for memristive CIM devices, while taking into account the more complex behavior of a defective memristive device, but did not work out the details [11]. It does not present what faults are actually sensitized, neither does it present complete test solutions to detect these faults.

This paper discusses the test of Scouting logic-based CIM in detail, including defects, fault models, and test solutions. The contributions of this paper are:

- We present fault models and tests for Scouting logic-based CIM devices.
- We demonstrate that unique faults can occur in the computation configuration.
- We demonstrate that memory faults are a subset of compute faults. Hence, a test for the computation configuration also detects all faults in the memory configuration.
- We show that the test for the computation configuration faults is able to reduce the overall test time.

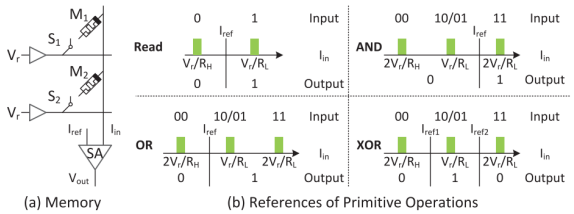
The computation configuration tests achieve 100% fault coverage and up to 18% test time reduction.

The remainder of this paper is structured as follows. Section II presents background information on Scouting logic-based CIM. Section III presents the test methodology. Section IV and Section V apply this methodology to develop a test for the memory and computation configuration respectively. Section VI discusses the results and draws a conclusion.



(a) Symbol (b) CF (c) I-V curve (d) Resistance (e) Bit-cell

Fig. 1: RRAM device



(a) Memory (b) References of Primitive Operations

Fig. 2: Scouting logic operations [4]

II. Background

This section presents the basics of RRAM devices, the CIM concept, and the implementation details of Scouting logic.

A. RRAM Device

RRAM devices are the storing elements in which data is stored in terms of resistance states. Details of RRAM devices such as symbol, conductive filament structure, I-V curve and bit-cell structure are shown in Fig. 1. Fig. 1b shows that the cell structure consists of a metallic oxide (green) that is sandwiched between a top electrode (TE) and a bottom electrode (BE) [12, 13]. When a sufficiently high positive voltage (higher than V_{set}) is applied, some of the bonds between the ions break and form a conductive filament (CF) of vacancies (blue circles) that can conduct a current. In the absence of the bias voltage, the CF remains intact, making this device non-volatile. When a negative voltage (lower than V_{reset}) is applied, some ions move back into the oxide region, thus reducing the size of the CF. The size of the CF decides the resistance state of the device. For instance, a larger and smaller CF represent the low resistive state (logical '1') and high resistive state (logical '0'), respectively. These resistance values can vary significantly due to defects or extreme process variations. Therefore, there are three more device states, i.e., 'L', 'U', and 'H' ranges corresponding respectively to extreme low logic state (resistance beyond the spec), undefined state, and extreme high logic state (resistance below the spec). A typical 1T-1R bit-cell is shown in Fig. 1e, where BL, WL, SL, and int indicate bit line, word line, select line, and internal node, respectively.

B. Scouting Logic-Based CIM

CIM architectures integrate logic operations into memory arrays in order to address the processor-memory data transfer bottleneck in computing systems. Similar to any standard memory, it comprises a memory array to store the data and periphery circuitry to access the memory and control the compute operations [3, 6]. The CIM core can operate in two different configurations [11]: the memory configuration (mem. config.) and the computation configuration (comp. config.). Since the comp. config. also uses read and write operations besides compute operations, it is a superset of the mem. config.

Scouting logic [4] is an implementation example of CIM that can execute bit-wise logic OR, AND, and XOR. Unlike a standard memory where a single bit-cell in a column is read at a time, Scouting logic activates two bit-cells in a column (as two inputs of the gate) simultaneously. Fig. 2 shows an example in which two bit-cells M1 and M2 are activated [4]. To perform a read operation, a suitable read voltage V_{r} is applied to the input terminals of both bit-cells, and the current (I_{in}) that is representing the equivalent parallel resistances of the two bit-cells is sensed using a sense amplifier. This current value is compared with a reference current (I_{ref}) to generate the final output value. By changing I_{ref} of the sense amplifier, different gates can be realized. Fig. 2 illustrates the OR, AND and XOR logic operations along with standard memory operations by adjusting the reference current values.

In this work, we have employed resistive RAM (RRAM) to act as the memristive device as this technology has many benefits such as high density, scalability, non-volatility, faster accesses, CMOS compatibility, etc., that makes it suitable for CIM operations. Implementation details can be found in [11].

III. CIM Test Methodology

Since the CIM core operates in mem. as well as comp. configs., it is required to test these two configurations before enabling them. Hence, we need tests for the mem. config. and comp. config. In the former case, only the memory functionality is tested. In the latter case, the hardware responsible for the computing functionality is tested. These tests are described in the following two sections. To *guarantee* a high defect coverage, the tests should be structural [11].

In general, any structural test development process for ICs has three steps [5]. The first step is to analyze defects thoroughly and develop defect models. These models are injected into the netlist of the design. Second, the faulty behaviors are observed after simulating that netlist with all possible defects and defect strengths to verify the fault space. The final step is the test generation step in which test patterns are generated that sensitize and detect the faults in the verified fault space. This test development approach is applied for both configurations.

RRAM crossbars comprise of memory arrays as well as periphery components such as address decoders (ADs), sense amplifiers (SAs), write circuitry, drivers and control logic. Some of these components require modifications to make the overall architecture suitable for Scouting logic. The components that are modified, need to be tested in the comp. config., while unmodified components only need to be tested in the mem. config. Most of the modifications need to be done for the periphery components. For instance, in order to access multiple rows, the decoder circuitry has to be doubled to allow selecting two independent row. Additionally, changes in the memory array, SA, and control logic designs are required.

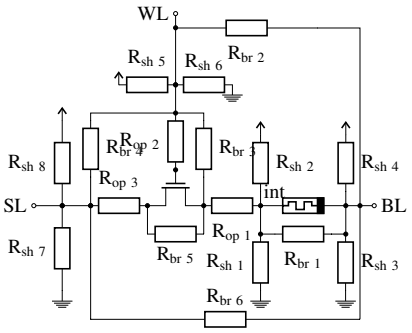


Fig. 3: Linear resistor defects

IV. Memory Configuration Test

Next, we will cover defect modeling, fault modeling, and test development for the CIM die in the memory configuration.

A. Defect Modeling

Defects may occur in three locations: the transistors, the interconnections, and the RRAM devices. Some typical transistor defects are: lithographic variations, polish variations, material impurity, pinholes, etc. [14]. These defects are typically modeled as linear resistor opens and bridges between the nodes of a transistor. Interconnection defects are: line edge roughness, irregular shapes, small particles, etc. These defects lead to increased resistance or bridges. Therefore these defects are also modeled as linear resistors between two nodes. Defects in the RRAM device are related to the oxide and electrode structures and have a strong effect on the conductive filament directly after forming the initial CF [8]. Due to the non-linear behavior of the RRAM device, these defects cannot be accurately modeled by linear resistors [8]. Therefore, these *forming defects* are modeled using the *device-aware* defect modeling approach that takes into account the actual physics of a defective device and appropriately models the behavior [5].

In this paper, we model transistor and interconnect defects with linear resistors between two nodes, as shown in Fig. 3. The strength of these defects (R_{def}) is varied from $1\,\Omega$ to $100\,\mathrm{M}\Omega$. The forming defects in the RRAM device are modeled using the device-aware defect model from [5] with forming current (I_{form}) between $1\,\mu\mathrm{A}$ and $35\,\mu\mathrm{A}$.

B. Fault Modeling

Fault modeling consists of defining the fault space and verifying it via defect model injection and circuit simulation. The model that is injected depends on the type of defect that is analyzed, as was described in the previous section.

1) Fault Space: We will define the fault space for the memory array, address decoders and sense amplifiers.

Array: Faults in a memory array are typically described using as a fault primitive (FP) in the $\langle \mathrm{S/F/R} \rangle$ notation [15]. Here, $\mathrm{S} \in \{0, 1, 0\mathrm{w}0, 0\mathrm{w}1, 1\mathrm{w}0, 1\mathrm{w}1, 0\mathrm{r}0, 1\mathrm{r}1\}$ denotes the sensitizing sequence, e.g., $0\mathrm{r}0$ denotes reading a '0' from a cell, and $1\mathrm{w}0$ denotes writing a '0' to a cell containing a '1', $\mathrm{F} \in \{\mathrm{L}, 0, \mathrm{U}, 1, \mathrm{H}\}$) denotes the state of the cell after the sensitizing operation is performed [5], and $\mathrm{R} \in \{0, 1, ?-\}$ denotes the read output ($?$ indicates a random read output, $-$ is used when S was a write operation). Using this notation, the fault space for *single cell* faults can be described as was done in [5]. However, it is possible that multiple cells are involved in a fault. To describe the fault space, the $\langle \mathrm{S/F/R} \rangle$ notation needs to be extended, e.g., a coupling fault can be described as $\langle \mathrm{S_a; S_v/F/R} \rangle$, where $\mathrm{S_a}$ denotes the aggressor's S and $\mathrm{S_v}$ the victim cell's [15].

Address Decoders: AD faults in a single decoder are well studied and can be grouped in static and delay faults. Static AD faults (AFs) will always lead to errors; they are [16]: no access, multiple cells, multiple addresses, and other cells. A defect may affect the timing of the AD and can sensitize an activation delay fault (ActD), or deactivation delay fault (DeactD) [17]. Due to these delays, other cells may be selected as well, or no selection occurs at all.

Sense Amplifiers: SA faults can be grouped in static and dynamic faults. Static faults are SA stuck-at faults (SASF) where the SA always switches to one value, irrespective of its inputs. Dynamic SA faults affect the switching behavior of the SA. These faults consist of the slow SA fault (SSAF) where the SA operation is too slow [18].

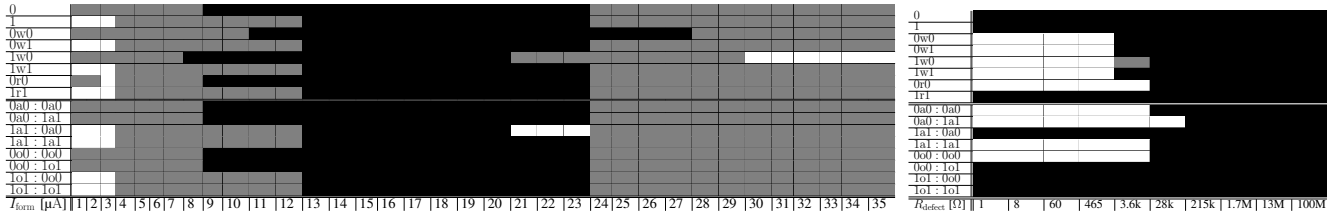
2) Validation of the Fault Space: Array: Fig. 4a graphically describes the faults that are sensitized by the forming defect for different strengths, while Fig. 4b does the same for defect $\mathrm{R_{br\,6}}$. The figures for the remaining defects of Fig. 3 are left out due to space considerations. The colors in these graphs relate to the class of the faults; a black box indicates that no FP was sensitized, while white and gray colors indicate that an FP was sensitized. The meaning of these colors is explained in the next section. It can be seen that a defect of certain strength can sensitize multiple faults. The combination of the defect and its strength with the FPs it sensitizes is called a fault class (FC). For example, take defect $\mathrm{R_{br\,6}}$ with a strength of $465\,\Omega$, then the fault class in the mem. config. contains the faults with $\mathrm{S} = 0\mathrm{w}0$, $0\mathrm{w}1$, $1\mathrm{w}0$, $1\mathrm{w}1$, and $0\mathrm{r}0$. In order to detect a defect with a certain strength, only one fault in the FC needs to be detected.

Address Decoders: Both static and dynamic AD faults have been shown to exist in Scouting logic-based CIM [11].

Sense Amplifiers: Both static and dynamic SA faults have been shown to exist in Scouting logic-based CIM [11].

C. Test Development

To develop a suitable test, we classify the faults based on the ease of their detection as easy-to-detect (ETD, white blocks) or hard-to-detect (HTD, gray blocks) [5]. ETD faults will always be sensitized by a given S and are guaranteed to be detected by regular memory operations, for example in a march algorithm, e.g., $\langle 0\mathrm{r}0/1/0 \rangle$. HTD faults are not guaranteed to be sensitized and detected with regular memory operations. For example, the FP $\langle 1\mathrm{r}1/\mathrm{U}/? \rangle$ is HTD, as it pushes the cell into the 'U' state and causes a random read output. Since the outcome of the read operation is not determined, a march algorithm cannot guarantee detecting this fault. Therefore, other test solutions are required to detect HTD faults, e.g., design-for-testability



(a) Sensitized FPs for forming defect

(b) Sensitized FPs for defect $R_{br\,6}$

Fig. 4: Fault maps for forming defect and defect $R_{br\,6}$. A white square indicates an ETD fault, a gray one an HTD fault, and a black one fault-free

(DFT) circuits. Note that this classification also applies to AD and SA faults, i.e., AFs and SASF are ETD, the others HTD.

The goal of a march algorithm is to maximize fault coverage while minimizing test time. The left side of Table I describes the FPs sensitized by the considered defects for the maximal defect range in the mem. config. The S in the $\langle \mathrm{S/F/R} \rangle$ notation denotes the sensitizing sequence in the top row of the table. Note that multiple faults can be sensitized by a single defect. For some defects, there is no single S that sensitizes faults for the complete defect strength range, e.g., the forming defect. To detect this defect, multiple faults need to be detected, which is indicated with an * in the table, e.g., to detect all forming faults, both $1\mathrm{w}0$ and $0\mathrm{w}1$ need to be applied as these sensitize the most faults fo this defect. Finally, FPs printed in red are HTD faults. The detection of these faults increase the fault coverage, e.g., detecting the gray squares (HTD faults) in Fig. 4a increases the fault coverage significantly, but detecting the gray square in Fig. 4b does not, because $0\mathrm{r}0$ already sensitizes this fault.

1) Test for ETD Faults: The test for the ETD array faults is a march algorithm that detects at least one of the ETD FPs that are sensitized per defect in Table I. That is, the algorithm detects at least one FP per FC. We select the FPs such that the total execution time of the march algorithm is minimized, while detecting all faults. This results in the following march algorithm for ETD faults in the mem. config.:

$$\begin{aligned}\text{March-ETD-Mem} = \text{M1} \Updownarrow (\mathrm{w0})\,; \text{M2} \Downarrow (\mathrm{r0}, \mathrm{w1})\,;\\ \text{M3} \Uparrow (\mathrm{r1}, \mathrm{w0})\,; \text{M4} \Updownarrow (\mathrm{r0})\,.\end{aligned}$$

Here, $\Uparrow$ indicates an increasing addressing order, $\Downarrow$ a decreasing order, and $\Updownarrow$ any addressing order. Every sequence of operations between the parentheses $(\dots)$ is called a march element with name My, with $y \in \mathbb{N}$. It is performed on an address before moving to the next. In this sequence, $\mathrm{w}x$ and $\mathrm{r}x$ denote respectively a read and write operation on the cell with value x, where $x \in \{0, 1\}$.

The fault coverage of this march algorithm is listed in Table II as a percentage of all FCs that are sensitized in the mem. config. for *every defect*. To illustrate, when analyzing the forming defect in Fig. 4a, there are 8 forming currents that do not sensitize a fault, hence there are 28 currents that do sensitize a fault and thus there are 28 FCs for this defect.Since, March-ETD-Mem detects all ETD faults, for the forming defect this means that the algorithm has a fault coverage of 9/28. The coverage for AD and SA faults is similarly defined.

To illustrate the detection capabilities, consider defect $R_{sh\,3}$ that sensitizes $\langle 0\mathrm{r}0/0/1 \rangle$ (see Table I). M2 sensitizes and detects this fault. Because this algorithm contains both $\mathrm{r}0$ and $\mathrm{r}1$ operations, all static SA faults are detected as well. The sequence also detects static AD faults in a single decoder with M2, M3, and M4 [16]. However, in Scouting logic, there are two row decoders and one column decoder. Hence, to detect faults in both row decoders, the sequence must be applied twice. This can be done efficiently by dividing the columns equally over both row decoders. For example, if the array has C columns, decoder A is used for all columns less than $C/2$ and decoder B for all remaining columns. As there is only one column decoder, the proposed test scheme will also detect all static faults there. The test time of this algorithm is $N + 2N + 2N + N = 6N$, where N is the execution time for one operation on the whole address space.

2) Test for HTD Faults: In order to detect the HTD faults in the array, we propose to use the DFT scheme in [19]. This scheme applies weak write ($\hat{\mathrm{w}}$) operations that decrease the operation duration or write voltage to detect cells that are in the 'U' state, but it can be extended to detect cells that are in 'L' and 'H' as well. The write operation is modified so that it pushes defective cells in a wrong state but defect-free cells remain unaffected. For example, the fault $\langle 0\mathrm{w}1/\mathrm{U}/- \rangle$ can be detected by performing a $\hat{\mathrm{w}}0$ operation that puts the defective cell into '0'. A defect-free cell does not have enough time to switch back to '0' and remains in '1'. When a $\mathrm{r}1$ is performed thereafter, the defective cell can be detected. Cells in 'L' and 'H' can be detected in a similar way, e.g., the fault $\langle 1\mathrm{w}1/\mathrm{H}/- \rangle$ can be detected by performing a $\hat{\mathrm{w}}0$ operation so that a defect-free cell switches to '0', while the defective cell remains in 'H' or '1'. By using weak write operations, the HTD array faults can be detected by the following algorithm:

$$\begin{aligned}\text{March-HTD-Mem} = \text{M1} \Updownarrow (\mathrm{w0})\,; \text{M2} \Downarrow (\mathrm{r0}, \mathrm{w1}, \hat{\mathrm{w}}0)\,;\\ \text{M3} \Uparrow (\mathrm{r1}, \mathrm{w0})\,; \text{M4} \Updownarrow (\hat{\mathrm{w}}1, \mathrm{r0})\,.\end{aligned}$$

The fault coverage of this algorithm is presented in Table II. This algorithm detects all ETD and almost all HTD faults in the array, SA and AD. It was shown in [17] that ActD and DeactD are sensitized when certain address transitions are made, e.g., by applying a $H1$ (Hamming distance between two addresses of 1) addressing order. Therefore, these HTD AD faults can be detected as well if the linear addressing order is replaced by an $H1$ order, e.g., $\Updownarrow$ is replaced by $\Updownarrow_{H1}$ [17]. SSAF can be sensitized by applying a stressing sequence to the SA that forces the output to switch quickly between '1'

TABLE I: Sensitized FPs that cover the maximal defect range. * indicates that a combination of faults needs to be detected, FPs printed in red are HTD faults.

	S Memory Configuration								S Computation Configuration							
Defect	0	1	0w0	0w1	1w0	1w1	0r0	1r1	0a0:0a0	0a0:1a1	1a1:0a0	1a1:1a1	0o0:0o0	0o0:1o1	1o1:0o0	1o1:1o1
Forming		⟨S/L/−⟩*, ⟨S/0/−⟩*, ⟨S/U/−⟩*		⟨S/L/−⟩*, ⟨S/0/−⟩*, ⟨S/U/−⟩*	⟨S/H/−⟩*, ⟨S/U/−⟩*	⟨S/L/−⟩*, ⟨S/U/−⟩*		⟨S/L/1⟩*, ⟨S/0/0⟩*, ⟨S/U/0⟩*, ⟨S/U/1⟩*			⟨S/U : 0/0⟩	⟨S/U : 1/1⟩, ⟨S/L : 1/0⟩			⟨S/U : 0/1⟩, ⟨S/L : 0/0⟩	⟨S/U : 1/1⟩
$R_{br\ 1}$					⟨S/1/−⟩, ⟨S/U/−⟩		⟨S/0/1⟩			⟨S/0 : 1/1⟩	⟨S/1 : 0/1⟩		⟨S/0 : 0/1⟩			
$R_{br\ 2}$							⟨S/1/0⟩	⟨S/1/0⟩						⟨S/1 : 1/0⟩		⟨S/1 : 1/0⟩
$R_{br\ 3}$								⟨S/L/0⟩, ⟨S/1/0⟩				⟨S/1 : 1/0⟩			⟨S/1 : 0/0⟩	
$R_{br\ 4}$				⟨S/0/−⟩*			⟨S/0/1⟩	⟨S/1/0⟩*				⟨S/1 : 1/0⟩			⟨S/1 : 0/0⟩	
$R_{br\ 5}$			⟨S/L/−⟩		⟨S/L/−⟩											
$R_{br\ 6}$							⟨S/0/1⟩			⟨S/0 : 1/1⟩						
$R_{op\ 1}$					⟨S/1/−⟩										⟨S/1 : 0/0⟩	
$R_{op\ 2}$							⟨S/0/1⟩								⟨S/1 : 0/0⟩	
$R_{op\ 3}$								⟨S/1/0⟩								
$R_{sh\ 1}$			⟨S/1/−⟩, ⟨S/U/−⟩		⟨S/1/−⟩, ⟨S/U/−⟩											
$R_{sh\ 2}$							⟨S/1/1⟩*	⟨S/L/0⟩*	⟨S/1 : 1/1⟩		⟨S/1 : 1/1⟩		⟨S/1 : 1/1⟩		⟨S/1 : 1/1⟩	
$R_{sh\ 3}$							⟨S/0/1⟩			⟨S/1 : 1/1⟩	⟨S/1 : 0/1⟩		⟨S/0 : 0/1⟩			
$R_{sh\ 4}$							⟨S/1/0⟩	⟨S/1/0⟩	⟨S/1 : 1/1⟩	⟨S/1 : 1/1⟩	⟨S/1 : 1/1⟩	⟨S/1 : 1/0⟩	⟨S/1 : 1/1⟩		⟨S/1 : 1/1⟩	
$R_{sh\ 5}$			⟨S; 0/1/−⟩	⟨S; 0/1/−⟩	⟨S; 0/1/−⟩	⟨S; 0/1/−⟩										
$R_{sh\ 6}$				⟨S/0/−⟩	⟨S/1/−⟩, ⟨S/U/−⟩		⟨S/0/1⟩					⟨S/1 : 1/0⟩			⟨S/1 : 0/0⟩	
$R_{sh\ 7}$			⟨S; 0/1/−⟩	⟨S; 0/1/−⟩	⟨S; 0/1/−⟩	⟨S; 0/1/−⟩										
$R_{sh\ 8}$								⟨S/1/0⟩				⟨S/1 : 1/0⟩			⟨S/1 : 0/0⟩	

TABLE II: Fault coverage for the proposed tests

Test	Memory Configuration ETD	Memory Configuration HTD	Computation Configuration ETD	Computation Configuration HTD
Array	77.3%	100%	81.8%	100%
Sense Amplifier	50.0%	100%	50.0%	100%
Address Decoder	33.3%	100%	50.0%	100%

and '0', which defective SAs will fail to do. The algorithm can be extended with such a stressing element, e.g., by adding M5 $\Updownarrow$ $(\mathrm{r0}, \mathrm{w1}, \mathrm{r1})$ to the sequence. The test time is then $11N$.

V. Computation Configuration Test

Next, we will cover defect modeling, fault modeling, and test development for the CIM die in the comp. config.

A. Fault Modeling

In this section, we define and verify the fault space in the comp. config. for the array, ADs, and SAs.

1) Fault Space:

Array: In Scouting logic, a compute operation can be seen as a special read operation in which two cells are read at once by a single SA. This dual access causes new faults in the memory array and hence requires a test solution. To define the fault space for these computation faults, the $\langle \mathrm{S/F/R} \rangle$ notation scheme needs to be extended to include the compute operations as: $\langle \mathrm{S}_1 : \mathrm{S}_2 \ / \ \mathrm{F}_1 : \mathrm{F}_2 \ / \ \mathrm{R} \rangle$. In this notation, S_1 and S_2 describe the sensitizing sequence applied to cell 1 and 2 respectively, : indicates that these sensitizing operations are applied in parallel, F_1 and F_2 indicate the state of the cells after the sensitizing operation, and R indicates the read output. In addition, the operations in S are also extended with the compute operations OR (o) and AND (a). For example, consider $\langle 0\mathrm{a}0_1 : 1\mathrm{a}1_2 \ / \ 0_1 : \mathrm{H}_2 \ / \ 1 \rangle$. Here, an AND operation is performed on two cells. After the operation is performed, the value of cell 2 is flipped from '1' to 'H' and the SA gave a '1' as output. This extended FP notation can be used to define the fault space for all faults in the memory array, both in the mem. config. and comp. config. Therefore, it can be concluded that the fault space of the array in the comp. config. is a superset of the fault space of the array in the mem. config.

Address Decoders: In the comp. config., the two decoders operate at the same time to select the operands for the compute operation. The parallel operation of the two decoders in the presence of a defect between them, can lead to port interference faults (AFpi) [20]. That is, one decoder enables a wrong word line only if the other decoder points to certain address. These faults do not occur when the ADs are operated independently. However, the AD faults that occur in the mem. config. can also occur in the comp. config. Hence, we can conclude that the AD fault space in the comp. config. is a superset of the AD fault space in the mem. config.

Sense Amplifiers: The SAs in the comp. config. suffer from the same faults as in the mem. config., except that these faults can occur for every reference current individually. That is, the AND reference current may suffer from SASF or SSAF as well as the OR/read reference current. Therefore it can be concluded that the SA fault space in the comp. config. is a superset of the SA fault space in the mem. config.

2) Sensitized Fault Space:

Array: Fig. 4a and 4b also present faults that are sensitized in the comp. config. We can conclude the following:

- *March optimization:* Many defects that sensitize faults in the mem. config. also sensitize faults in the comp. config., e.g., defect $R_{br\ 6}$ sensitizes both $\langle 0\mathrm{r}0/0/1 \rangle$ and $\langle 1\mathrm{a}1_1 : 1\mathrm{a}1_2 \ / \ 0_1 : 1_2 \ / \ 0 \rangle$ for the strength from $1\,\Omega$ to $3.6\,\mathrm{k}\Omega$, increasing the amount of faults in the FC. The march algorithm can use either one of them and thus has the potential to become more time efficient.
- *Fault coverage:* Defect $R_{br\ 6}$ in the mem. config. sensitizes FPs up to a strength of $3.6\,\mathrm{k}\Omega$, while in the comp. config. FPs are also sensitized for strengths up to $28\,\mathrm{k}\Omega$. Therefore, including this fault leads to higher fault coverage.

Address Decoders: Both static and both dynamic AD faults have been shown to exist in Scouting logic-based CIM [11].

Sense Amplifiers: Both static and dynamic SA faults have been shown to exist in Scouting logic-based CIM for both references [11].

From the above paragraphs it follows that the fault space of the comp. config. is always a superset of the fault space of the mem. config. Therefore, a test that detects all faults in the comp. config.also detects all memory faults. Hence, test time can be reduced by only testing the comp. config. Further, the

second point above proves that the computation fault space covers a larger defect space than the mem. config. Hence, a high-quality CIM test should focus on testing the comp. config.

B. Test Development

This section presents a test for the ETD and HTD faults.

1) Test for ETD Faults: The right side of Table I lists the compute faults that were sensitized per defect. As was discussed above, march algorithms can be made more efficient by adding compute operations. Besides the wider selection of faults in an FC that can be included, the compute operations can also be used to speed up parts of the algorithm. For example, the execution time of a march element $\Updownarrow (\mathrm{r0})$ can be halved by performing an OR operation per two rows and verifying the output to be '0'. That is, an OR operation is performed first on row 0 and 1, second on row 2 and 3, etc., We denote this addressing as $\Updownarrow_2 (...)$.

The following algorithm uses compute operations to sensitize faults and applies the proposed addressing scheme.

$$\begin{aligned}\text{March-ETD-Comp} = \text{M1} \Downarrow (\mathrm{w0})\,;\text{M2} \Downarrow_2 (\mathrm{o0})\,;\\ \text{M3} \Uparrow (\mathrm{r0w1r1o1})\,;\text{M4} \Updownarrow_2 (\mathrm{o1a1})\,;\\ \text{M5} \Updownarrow (\mathrm{w0a0})\,;\text{M6} \Updownarrow_2 (\mathrm{o0})\,.\end{aligned}$$

The fault coverage of this algorithm is denoted in Table II. The algorithm detects all comp. config. ETD array faults. Further, it detects all SSAFs for the all references. With $\Updownarrow_2$ addressing, the test time is $9N$ instead of $11N$, a speedup of 18%.

Static AD faults in a single decoder are sensitized and detected by elements M1, M2, and M3. However, no AFpis are detected by this algorithm. In [20] it was shown that the detection of all AFpis requires that a sensitizing and detection operation is applied to all combinations of addresses. The algorithm in [20] has a test time of $2R + 8R^2$, where R indicates the amount of rows. For Scouting logic, this can be slightly reduced by applying an OR operation instead of two subsequent r0 operations, resulting in the following algorithm with execution time $\frac{3}{2}R + 7R^2$:

$$\begin{aligned}\text{March-AFpi} = \text{M1} \Updownarrow_{c1\in R} (\mathrm{w1})\,;\text{M2} \Updownarrow_{c1\in R} (\Updownarrow_{c2\in R} (\mathrm{r1}_{c1},\\ \mathrm{r1}_{c2}, \mathrm{w0}_{c1}, \mathrm{w0}_{c2}, \mathrm{o0}_{c1} : \mathrm{o0}_{c2}, \mathrm{w1}_{c1}, \mathrm{w1}_{c2}))\,;\\ \text{M3} \Updownarrow_{c1,c2\in R} (\mathrm{a1}_{c1} : \mathrm{a1}_{c2})\,.\end{aligned}$$

Here, $c1$ and $c2$ indicate a selected cell within the same column, and $c1 \neq c2$. March-ETD-Comp can be modified to detect AFpis, e.g., by performing the AFpi algorithm after M4. The total test execution time then becomes: $9N + \frac{1}{2}R + 7R^2$ (M1 in March-AFpi is not required, because the memory is already initialized to '1' by M3 in March-ETD-Comp).

2) Test for HTD Faults: From Table I, it follows that the compute operations do not sensitize additional HTD faults that cover more FCs, except for the forming defect. To detect the HTD faults in the comp. config., we again apply the weak write operations DFT [19] resulting in the following algorithm:

$$\begin{aligned}\text{March-HTD-Comp} = \text{M1} \Downarrow (\mathrm{w0})\,;\text{M2} \Downarrow_2 (\mathrm{o0})\,;\\ \text{M3} \Uparrow (\mathrm{r0w1r1o1})\,;\text{M4} \Updownarrow_2 (\mathrm{o1a1})\,;\\ \text{M5} \Updownarrow (\mathrm{w0\hat{w}1r0a0})\,;\text{M6} \Updownarrow_2 (\mathrm{o0})\,.\end{aligned}$$

From Table II it follows that this algorithm detects all array faults. Although it detects all SSAFs, an additional stressing sequence is required to detect the SASFs for both reference currents. A sequence that can do this stressing and can be added at the end of the algorithm is: M7 $\Updownarrow (\mathrm{o0}, \mathrm{w1}, \mathrm{o1})\,;$ M8 $\Updownarrow (\mathrm{a1}, \mathrm{w0}, \mathrm{a0})$. The static AD faults can be detected in the same way as in march-ETD-Comp. To detect the delay faults in the ADs, $H1$ addressing should be used as well. With all improvements to increase the fault coverage, the total test time becomes $17N + \frac{1}{2}R + 7R^2$.

VI. Conclusion

In this paper, we defined the fault space and presented tests for the two operating configurations of a Scouting logic-based CIM architecture. We proved that the fault space of the comp. config. is a superset of the fault space of the mem. config. by showing that there exist faults that can only occur in the comp. config. and thus not be detected by regular memory tests, leading to test escapes. We showed that our HTD test for the comp. config. is able to 100% of the faults. Further, we have also demonstrated how compute operations can be used to speed up the execution of a march algorithm. For example, we have demonstrated how the time it takes to verify whether all cells are '0' can be halved by applying OR operations, resulting in a test time reduction up to 18%.

References

[1] D. A. Patterson, "Future of Computer Architecture," in *BEARS*, 2006.

[2] S. Hamdioui *et al.*, "Memristor for Computing: Myth or Reality?" In *DATE*, 2017, pp. 722–731.

[3] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015.

[4] L. Xie *et al.*, "Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing," in *ISVLSI*, 2017.

[5] M. Fieback *et al.*, "Device-Aware Test: A New test Approach Towards DPPB Level," in *ITC*, 2019.

[6] T.-L. Tsai *et al.*, "Testing of In-Memory-Computing 8T SRAMs," in *DFT*, 2019.

[7] A. S. Emara *et al.*, "Testing of memristor ratioed logic (MRL) XOR gate," in *ICM*, 2016.

[8] M. Fieback *et al.*, "Testing Resistive Memories: Where are We and What is Missing?" In *ITC*, 2018.

[9] C. Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE TC*, 2015.

[10] S. Kannan *et al.*, "Sneak-Path Testing of Crossbar-Based Nonvolatile Random Access Memories," *IEEE TN*, 2013.

[11] S. Hamdioui *et al.*, "Testing Computation-in-Memory Architectures Based on Emerging Memories," in *ITC*, 2019.

[12] H.-S. P. Wong *et al.*, "Metal-Oxide RRAM," *Proc. IEEE*, 2012.

[13] S. Yu *et al.*, "Emerging Memory Technologies: Recent Trends and Prospects," *IEEE SSCM*, 2016.

[14] K. J. Kuhn *et al.*, "Process Technology Variation," *IEEE TED*, 2011.

[15] A. Van de Goor *et al.*, "Functional memory faults: a formal notation and a taxonomy," in *VTS*, 2000, pp. 281–289.

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

[17] S. Hamdioui *et al.*, "Opens and Delay Faults in CMOS RAM Address Decoders," *IEEE TC*, 2006.

[18] A. van de Goor *et al.*, "Detecting faults in the peripheral circuits and an evaluation of SRAM tests," in *ITC*, 2004, pp. 114–123.

[19] S. Hamdioui *et al.*, "Testing Open Defects in Memristor-Based Memories," 2015.

[20] S. Hamdioui *et al.*, "Address decoder faults and their tests for two-port memories," in *MTDT*, 1998.

B

Testing Computation-in-Memory Architectures Based on Emerging Memories

This appendix contains the paper published at the International Test Conference - 2019: S. Hamdioui, M. C. R. Fieback, *S. Nagarajan*, M. Taouil "Testing Computation-in-Memory Architectures Based on Emerging Memories", in 2019 *International Test Conference*, Washington D.C, U.S.A, 2019

Testing Computation-in-Memory Architectures Based on Emerging Memories

Said Hamdioui Moritz Fieback Surya Nagarajan Mottaqiallah Taouil
Computer Engineering Laboratory
Delft University of Technology
Mekelweg 4, 2628CD, Delft, The Netherlands
Email: S.Hamdioui@tudelft.nl

***Abstract*—Today's computing architectures and device technologies are becoming incapable of meeting the increasingly stringent demands on energy and performance posed by evolving applications. Therefore, alternative novel post-CMOS computing architectures are being explored. Some of these are Computation-in-Memory (CIM) architectures based on memristive devices; they integrate the processing units and the storage in the same physical location (i.e., the memory based on memristive devices). Due to their advanced manufacturing processes, use of new materials, and dual functionality, testing such chips requires specific schemes and therefore special attention. This paper describes the need for testing CIM architectures, proposes a systematic test approach, and shows the strong dependency of the test solutions on the nature of the architecture. All of these will be demonstrated using a design that is designed for computation-in-memory bit-wise logical operations.**

I. Introduction

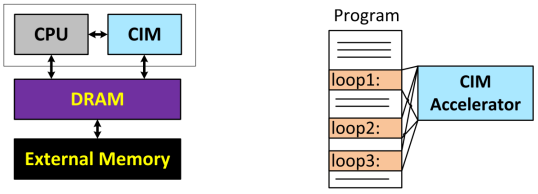
In the past decades, the world has seen a phenomenal increase in computing performance, resulting in smaller, faster, and more energy efficient computers. However, today's computer architectures as well as the CMOS technology used to manufacture them are facing major challenges such as memory wall, power wall, leakage wall, and cost wall [1, 2]; these make them economically not attractive for many evolving applications which are extremely demanding, e.g., in terms of MOPs/Watt. Therefore, continuing with delivering sustainable benefits in the foreseeable future requires the exploration of alternative (unconventional) computing architectures that leverage novel post-CMOS device technologies such as memristive devices (e.g., resistive RAM (RRAM), phase change memory (PCM), spin-transfer-torque magnetic RAM (STT-MRAM)). One of these is a Computation-in-Memory (CIM) architecture based on memristive devices [3, 4]; it is based on integrating the processing units and the memory in the *same physical location*. As a consequence, it signicantly reduces the memory accesses and data movements while supporting massive parallelism, potentially resulting in orders of magnitude improvement in terms of energy and computing efciency [5, 6]. Many companies (e.g., IBM, ARM), research institutes (e.g., IMEC), and universities are investigating and demonstrating such an architecture [5]. There are still many issues that have to be solved in order to get this computer technology mature enough; examples are: endurance of the memristive devices, variability, complexity of the control units within the CIM core, etc [7, 8]. In addition, and like all other ICs, these CIM dies need to be tested for manufacturing defects, in order to guarantee sufcient outgoing product quality to the customer. The manufacturing process of memristive-based CIM cores involves additional steps and makes use of new materials [9], which may lead to new failure mechanisms. In addition, a CIM die acts both as a memory as well as a computing unit; and hence, it has to be tested for both functionalities.

To the best of our knowledge, this is the first paper to discuss the test needs for memristive device-based CIM architectures. Nevertheless, there is some published work on emerging memories (relying on memristive devices) upon which CIM architectures are based. Most of this work is based on modeling defects as linear resistors, injecting them in the memory netlist in order to perform circuit simulation and derive fault models, and thereafter test and design-for-testability (DfT) solutions [10–16]. However, recent work has demonstrated that using resistors to model defects in, for example, RRAM and STT-MRAM is not accurate enough due to the non-linearity of these devices [17, 18]. Inaccurate defect modeling may lead to non-realistic fault models, and hence, low-quality tests; this has enabled the development of Device-Aware-Test approach [19].

This paper addresses the test aspects of CIM architectures based on emerging memristive devices. It briefly discusses the feasibility of functional and structural testing. In addition, it provides a systematic and structural approach for testing, and it highlights the need for testing the CIM die for its two different functional configurations, once as a memory and once as a computing unit. The paper also shows the dependency of the test solutions on the nature of the CIM architecture itself by demonstrating this test approach for a CIM design that performs bit-wise logic operations.

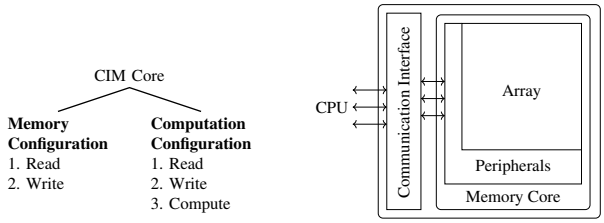
The remainder of this paper is structured as follows. Section II presents background information on CIM, including a classification of different CIM architectures. Section III presents the proposed structural test approach for CIM architectures. Sections IV, V and VI apply this approach for a case study based on bit-wise logical CIM implementation. Section VII presents a discussion and conclusion.

978-1-7281-4823-6/19/$31.00



(a) CIM core (b) CIM program

Fig. 1: CIM accelerator [5]



(a) CIM configurations (b) CIM block diagram

Fig. 2: CIM configurations and block diagram

II. Computation-in-Memory

In this section, we briefly present the concept of CIM architectures and classify them. Then, an implementation example for each class is given; one of them will be used as case study in this paper. However, in order to better understand these implementations, the working principles of an RRAM (used as a memristive device) will be be introduced first.

A. CIM Concept and Classification

The CIM architecture is based on integrating the processing units and the storage in the same physical memory location. A realistic implementation that many researchers are prototyping is shown in Fig. 1a [5, 20, 21]; the CIM core may consist of very dense memristive crossbar array and CMOS peripheral circuitry. The CIM die takes over the memory-intensive computation parts from the processor, thus significantly speeding up the execution and reducing the energy consumption by eliminating large amounts of data transfers. Fig. 1b illustrates a program that could be executed efciently on a CIM architecture; multiple loops can be executed within the CIM core while the other parts of the program can be executed on the conventional core. Each time a loop is invoked, the CPU sends a macro-instruction to the CIM core which decodes and executes it locally, and returns the final results.

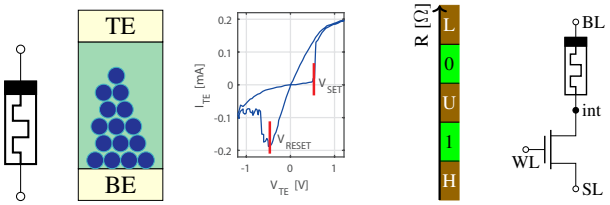
As already mentioned, computing in the CIM core takes actually place within the memory. Hence, the CIM core can operate in two different configurations: *memory* and *computation* configuration. Fig. 2a shows these configurations, as well as the operations that each configuration requires. Since the computation configuration also uses read and write operations, it is a superset of the memory configuration. Fig. 2b shows a block diagram of a CIM die. In addition to the memory core, it consists also of a communication interface. It is worth noting that computations in the CIM core take place *within* the memory core. Because a memory core consists of a *memory array* and *peripheral circuits*, and depending on *where* the result of the computation is produced, CIM architectures can be divided into two classes [22]:

- *CIM-Array (CIM-A):* In CIM-A, the computation result is produced within the array. Examples of such architectures are PLiM [23], ReVAMP [24], CIM device [25], etc. The CIM-A core typically requires a significant redesign of the memory array to support computing, as conventional memory cell layouts are typically optimized for storage functionality only.
- *CIM-Periphery (CIM-P):* In CIM-P, the computation result is produced within the peripheral circuitry. Examples of such architectures are PRIME [26], Pinatubo [27], CIM-Accelerator [28], etc. This architecture focuses on special circuits in the peripherals to realize, for example, bit-wise logic operations [27, 29], matrix-vector multiplication [6, 30], etc. Even though the computational results are produced in the peripheral circuits, the memory array could be a significant component in the computations. For example, to perform bit wise logic operations, multiple rows in the array need to be simultaneously activated.

As CIM performs operations within the memory core, at least part of the operands should be stored in the memory array. In other words, the operator being executed within the memory needs to have *all operands* stored in the array (as *resistive*) or *only* part of the operands is stored in the array and the other part is received via the memory port(s) (hence their logic values are *hybrid*, i.e., resistive and voltage). This results in four sub-classes: CIM-Ar, CIM-Ah, CIM-Pr and CIM-Ph; the additional letters 'r' and 'h' denote the nature of the inputs (operands), namely resistive and hybrid, respectively. An example of CIMP-Ah and CIM-Pr will be discussed in Subsection C and D.

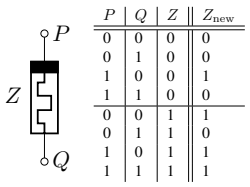
B. RRAM Device Technology

RRAM devices are one of the most popular memristive devices; they are non-volatile, two-terminal, non-linear devices that can switch their resistance [7, 31, 32]. The symbol to denote an RRAM device is shown in Fig. 3a, while the structure of the device is shown in Fig. 3b; the structure consists of a metallic oxide (green) that is stacked between two electrodes (yellow, top (TE) and bottom electrode (BE)) [7, 32]. When a voltage higher than the set threshold ($V_{\mathrm{TE}}>V_{\mathrm{SET}}$) is applied, some of the bonds between the metal and oxygen ions break. The oxygen ions are attracted to the positively charged electrode, leaving behind a chain of vacancies (blue circles). This chain, called the *conductive filament* (CF), can conduct a current. Even if the bias voltage is removed, the CF will remain intact, making this device *non-volatile*. On the contrary, when a negative voltage lower than the reset threshold ($V_{\mathrm{TE}}<V_{\mathrm{RESET}}$) is applied, some of the ions move back into the oxide, thus reducing the size of the CF. The shape of the CF determines the resistance of the device; larger CFs have a lower resistance. Fig. 3c shows the RRAM switching behavior with a current-voltage graph.



(a) Symbol (b) CF (c) I-V curve (d) Resistance (e) 1T-1R

Fig. 3: RRAM device



P	Q	Z	Z_{new}
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	1

Fig. 4: Majority logic

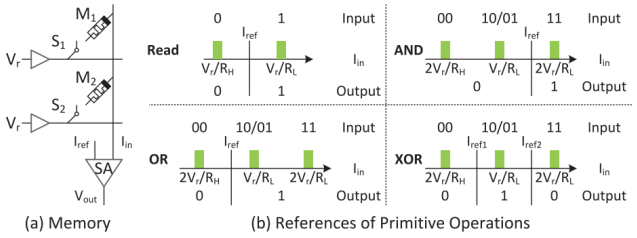
For memory applications, we distinguish two resistive states: the low resistive state (LRS, SET state, or logical '1'), and high resistive state (HRS, RESET state, or logical '0'). As this resistance is continuous and slightly varies per write cycle [7, 32], ranges that correspond to these two states are defined. Fig. 3d shows these specs for the two logic ranges ('0' and '1'), as well ranges outside the defined specs that an RRAM device can enter due to defects or extreme process variations [7]; these are 'L', 'U', and 'H' ranges corresponding respectively to extreme low logic state (resistance beyond the spec), undefined state, and extreme high logic state (resistance below the spec); the states 'L', 'U', and 'H' have been seen in defective RRAMs [19]. Fig. 3e presents a typical 1T-1R cell; here, BL, WL, and SL indicate bit line, word line, and select line respectively, while int is the internal node of the cell.

C. CIM-Ah: Majority Logic

The majority logic gate [3] shown in Fig. 4 is an implementation example of the CIM-Ah class using a memristive device Z. It has three inputs: P and Q supplied as voltages from the *peripherals*, and Z stored in the *array*; The output Z_{new} is produced after a majority operation is performed. The output is '1' if the majority of the inputs P, $\overline{Q}$, and Z are '1', as described in the truth table. Here $\overline{Q}$ denotes the negation of Q. The state of Z can only change to another state for a limited amount of input combinations P and Q, as shown in the truth table of the figure. This function can be used to develop other logic functions, like imply or inversion.

D. CIM-Pr: Scouting Logic

Scouting logic [29] pictured in Fig. 5a is an implementation example of CIM-Pr executing bit-wise logic OR, AND, and XOR. The operands are initially programmed in the *memory cells* M1 and M2. The operation is performed by selecting the two cells simultaneously (by applying a *read voltage* V_{r}) and comparing the resulting current (I_{in}) to a reference current (I_{ref}) using a dedicated sense amplifier (SA); the reference to be used depends on the operation to be performed as shown in Fig. 5b.



(a) Memory (b) References of Primitive Operations

Fig. 5: Scouting logic operations [29]

III. Test Methodology for CIM

This section presents a testing methodology for CIM cores. However, the difference between functional versus structural testing for such cores is first discussed.

A. Functional and Structural Testing

Tests for electronics can be classified into two categories: *functional* and *structural* tests [33]. Functional tests aim at checking the *proper operation* of the device-under-test, while structural tests aim at checking if the device is *manufactured correctly*. The question is which of these two approaches could be used for CIM die testing.

Functional tests apply a range of input stimuli to a device and observe if the corresponding output responses are correct, as defined by the device operation. To illustrate this for a CIM core, assume a functional test is used to test the CIM core being able to perform bit-wise logic operations of two operands. If we assume that the memory within the CIM core has r-bit row addresses, then there are $2^r \cdot (2^r - 1)$ possible combinations of selecting two operands. Even if extremely high test frequencies of 10 GHz are used, testing for all these combinations would take more than 58 years per chip for an address size of r=32. Besides being extremely time consuming, detection of all faults is still not guaranteed. For instance, the above case does not consider different values for the operands.

Structural tests, in contrary to functional tests, verify if a device is *manufactured correctly*; i.e., the device is free of manufacturing defects such as broken connections. It assumes that if the device is manufactured correctly, the device should functionally work properly. These tests rely on *fault models* that describe the faulty behavior of the device in the presence of a defect. This makes it feasible to define how these faults are sensitized and measure whether a test detects them or not. Therefore, *accurate* fault models are the key enabler for high quality structural test solutions. Note that structural tests do not require all input combinations to be tested, but merely those that *sensitize* the targeted faults models. Therefore, structural tests are both faster and achieve a higher and measurable fault coverage [33]. As a result, structural testing is more widely adopted. Thus, a high-quality CIM test should be a *structural* test, and will require accurate fault models reflecting the real defect behavior of CIM dies. Nevertheless, functional tests could be used to increase the fault coverage of faults that cannot be detected with structural testing, as is recognized in the community [34].

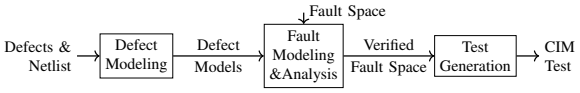


Fig. 6: Structural test approach

B. CIM Test Approach

As already mentioned, a CIM core operates in two configurations: memory configuration and computation configuration. Note that at least part of the CIM hardware used in the computation configuration is not used in the memory configuration. Hence, CIM cores cannot be tested as regular memories. CIM cores have to be tested for both configurations. As the computation configuration makes use of the memory, the latter has to be tested first. Testing CIM cores has to be performed as follows:

1) **Memory Configuration Test:** In this case, the memory functionality is tested; i.e., only the hardware that is required to perform memory operations is enabled and tested. Obviously, common memory test solutions applicable to the type of memory can be used (e.g., RRAM, STT-MRAM, PCM). Note that testing CIM in this configuration is an independent step, and does not test all hardware involved in the computation configuration.
2) **Computation Configuration Test:** In this case, the hardware responsible for all the computing functionalities is tested. This hardware strongly depends on the CIM architecture and the computing features it enables. For example, testing a CIM die with (analog) vector matrix multiplication features could be different than testing for logic bit-wise operations.

Test development for any IC follows three known steps illustrated in Fig. 6. First, the *defects* must be understood and adequately modeled. The resulting *defect models* are injected into the electrical netlist of the design. Second, this netlist is simulated and the faulty behaviors are observed and compiled into *fault models*. Ideally, before the fault analysis, the complete *fault space* should be defined (when applicable). During the *fault analysis*, the fault space is verified by injecting every defect in the netlist, which results in a set of realistic faults for that specific design or layout. Third, test solutions for the realistic faults are *generated*. Applying the above test development approach to CIM would mean applying it two times; once for each CIM configuration (i.e., memory and computation).

Test development for CIM as memory: the memory core of CIM can be any kind of memory such as conventional ones (SRAM, DRAM) as well as emerging ones (RRAM, PCM, STT-MRAM). Although testing of SRAM and DRAM is very mature, testing of emerging memories is still under investigation. They may need radically new approaches in defect modeling; a defective non-linear device (e.g., an RRAM device) cannot be accurately modeled with a linear resistor in series or in parallel with a perfect device [18, 19].

Test development for CIM as computing unit: As already mentioned, testing CIM in this configuration is strongly dependent on the design of the architecture. Defining what to test for implies the identification of the modified or new blocks integrated with the memory core to realize the computing functionality. To illustrate this, we will briefly analyze two examples of CIM architectures: CIM-Ah and CIM-Pr as discussed in Section II.

CIM-Ah Majority Logic: realizing such functionality within e.g. RRAM crossbar will need the modification of the following memory components: a) Memory array, b) BL and SL drivers, and c) Control logic. CIM-A architectures require always a redesign of the memory cells, as the conventional memory cell dimensions and their embedding in the bit and word line structure do not allow them to be used for logic. A conventional memory cell is namely heavily optimized in terms of processing stack and layout. Therefore, any modifications of the array require a new cell design and characterization process for the new control voltages, currents, etc. In addition, modifications in the periphery are needed to support the changes in the cell. In case of CIM-Ah Majority Logic, the write drivers and the control circuitry have to be redesigned to support the required functionality; e.g., the control logic needs to assure that the output of the sense amplifier can be fed back into the array via the drivers for operations on data from multiple cells. Therefore, testing CIM-Ah Majority Logic requires the guarantee of testing the memory array, BL and SL drivers, and the control logic. Note that the memory array is tested both in the memory configuration as well as in computation configuration; an access to the memory during computation could lead to an erroneous bit flip of the cell.

CIM-Pr Scouting Logic: As Fig. 5a shows, realizing such functionality, for example within RRAM crossbars, will need the modification of the following memory components: a) Memory array, b) Word line decoders, c) Sense amplifiers, and d) Control logic. Even though the computational results are produced in the peripheral circuits, the memory array for CIM-P is a substantial component in the computation. As the peripheral circuits are modified, the currents and voltages applied to the memory array are typically different than in the conventional memory. Obviously, the majority of the changes take place in the peripheral circuits and minimal to medium changes are required in the memory array. CIM-Pr Scouting Logic activates two or more (but not many) rows of a memory array simultaneously (similar to multi-port memories) during computations. Hence, in addition to a customized sense amplifier to perform the logic operation, this architecture also requires modifications in the address decoder to activate several rows at the same time. Note, however, that modifications in the cell array could be minimal as the total read current is still small. Therefore, testing CIM-Pr Scouting Logic requires to test the memory array, sense amplifiers, the decoders, and the control logic. Note also here that the memory array is tested both in the Memory Configuration as well as in Computation configuration; e.g., simultaneous access of the memory array during computing may lead to a fault in a cell.

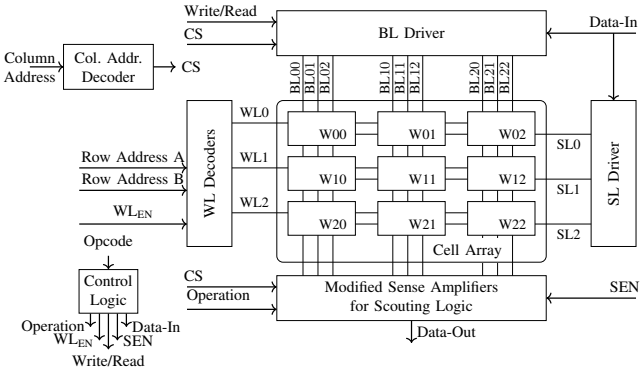
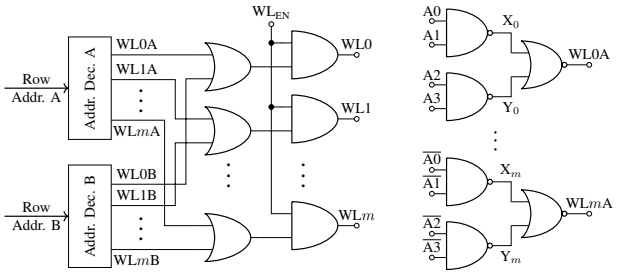


Fig. 7: Simulation Architecture



(a) WL Decoder (b) Address Decoder

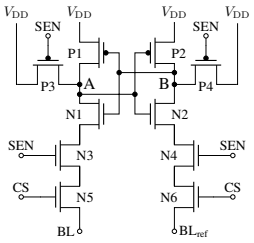
Fig. 8: WL circuitry

IV. CIM-Pr Architecture

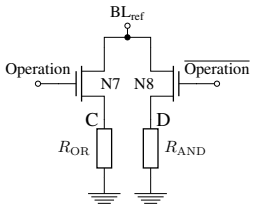
This section describes the implementation of CIM-Pr architecture based on Scouting Logic [29], which will be used to demonstrate the proposed test approach in the next two sections. This architecture is shown in Fig. 7; it is based on a regular RRAM design. Note that the majority of the building blocks (subcircuits) remain unmodified; these consist of the column address decoder, the BL driver, and the SL driver. The column address decoder decodes the column address and drives the corresponding column select (*CS*) line. The BL driver drives the BL corresponding to the *CS* line with the data in *Data-In*. To prevent the decoder from disturbing read operations, its output is fed through a tri-state buffer that is controlled by *Write/Read*. The SL driver controls the SLs based on *Data-In*; the SL is '0' when setting (w1) and reading, and '1' when resetting (w0) the cells.

However, in order to perform Scouting logic (i.e., bit-wise logic operations on two operands), some sub-circuits needed to be redesigned; these consist of: a) the WL decoder (that should be able to select two wordlines simultaneously), b) the SAs (that need to support appropriate logic functions; see Fig. 5), c) the control circuitry (to provide appropriate control signals based on the *Opcode*), and d) memory array. The latter is typically optimized for storage and would undergo some minimal modifications to allow for specific drive voltages and read currents. As the modified subcircuits will need special attention during testing, they will be briefly explained next; we focus on WL decoders and SAs.

- **WL Decoders:** The WL decoder, shown in Fig. 8a, decodes two *Row Addresses* A and B and drives the corresponding WLs (i.e., address m drives WLm) for the selection of the appropriate two words for a logic operation. Each address can be used to select any of the WLs. Besides logic gates, the WL decoder consists of two identical address decoders. Fig. 8b illustrates such a 4-bit decoder; for each input combination (e.g., $A_3A_2A_1A_0$=1111) one WL is selected. Each two selected WLs per input combination (e.g., $A_3A_2A_1A_0$=$B_3B_2B_1B_0$=1111) are ORed, and the resulting signal is ANDed with the *WL$_{EN}$* signal to control the timing.
- **Sense Amplifier:** One possible modified SA design for Scouting logic is shown in Fig. 9a, which is based on [35]. The two nodes A and B are precharged when no operation takes place, i.e., *SEN*='0'. Once the SA is enabled via the *SEN* and *CS* signals, the two nodes will be discharged via BL and BL$_{ref}$. The time it takes to discharge the nodes depends on the connected resistances to these nodes. For example, if R_{BL}<R_{BLref}, BL will discharge faster. After some time, the cross-coupled inverters begin to charge node B, allowing for even faster discharging of node A and the capturing of the operation outcome; we use node B in our design. To enable OR and AND bit-wise logic operations, the SA needs to have two corresponding reference currents, I_{OR} and I_{AND} (see Fig. 5). These are implemented using two different resistors, R_{OR} and R_{AND}, as shown in Fig. 9b. The *Operation* signal is used to select a reference; $\overline{Operation}$ is its logic complement. Fig. 10 shows the relative resistance of these references with respect to the equivalent resistance of the two cells being selected for the operation. In the memory configuration, the equivalent resistance is equal to the resistance of the cell being read, while in the computation configuration, it is equal to the parallel resistance of two cells being accessed. Note that R_{read}=R_{OR}.



(a) Sense Amplifier [35]



(b) Operation reference

Fig. 9: Sensing Circuitry

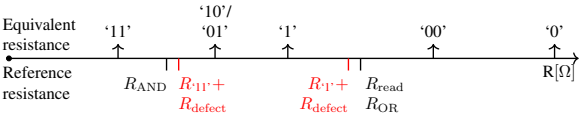


Fig. 10: Scouting logic relative resistance and references

V. CIM-Pr Memory Configuration Test

This section illustrates the test approach for the CIM-Pr core in the memory configuration; it includes defect modeling, fault analysis, and test generation; see Fig. 6.

A. Defect Modeling

The manufacturing process of a CIM core consists of three production phases: the front-end-of-line (FEOL), the back-

end-of-line (BEOL), and CF forming. To accurately estimate the impact of manufacturing defects on the circuit behavior, these defects need to be understood and modeled such that they can be used during circuit simulation for fault analysis. Two classes of defect models exist; they are discussed next.

Linear resistor as defect model: During the FEOL phase, transistors are fabricated on the wafer. Here standard transistor defects may occur that are related to line edge roughness, random dopant fluctuations, gate material granularity, etc. [36]. These defects may result, e.g., in reduced driving capabilities. They can have an impact on the peripheral circuitry as well as on the memory array, e.g., the SA becomes biased towards one logical value. After the FEOL phase, the lower metal layers are deposited in the BEOL phase. Lithographic issues or misalignment may cause defects here, resulting in shorts or opens in the wiring [37]. These defects again affect both the peripherals and the memory array. For example, the address decoder may wrongfully access multiple cells at the same time. These defects have been always modeled as linear resistors [11, 13] that act as a short or an open between two nodes.

Device-Aware defect models: The RRAM device is fabricated between two metal layers. Defects that may occur can be related to the electrode [38] and the oxide structure [39], which do affect the memory array. After this step, the remaining metal layers are deposited. To create a CF in the RRAM device, a forming step is required; this step strongly depends on the forming current (I_{form}) and may cause defects like over-forming or non-forming [17]. Although it can be convincing for modeling opens and shorts in interconnects, using linear resistors for defect modeling has never been validated for any device. It has recently been demonstrated that this assumption is inaccurate for emerging technologies such as (RRAM) [17] and (STT-MRAM) [40]; the results showed that the traditional approach may even lead to wrong fault models. Hence, it is incapable of delivering high-quality test solutions. This has resulted in the development of *device-aware defect modeling* approach [17, 19, 40]; it aims at accurately modeling physical defects, by incorporating the way the defect impacts the technology parameters (e.g., length, width) and thereafter the electrical parameters (e.g., the critical switching current) of the device [40]. This results in an electrical model of the defective device (e.g. RRAM device). This model can be then used to replace a defect free model at the circuit level to investigate its impact on the memory behavior. Note that in case of Device-Aware defect modeling, each defect may result in a different electrical model of the device.

B. Fault Modeling

Fault modeling is ideally based on two steps: 1) fault space definition, and 2) fault space validation using defect injection and circuit simulation. The fault space identifies *all possible* faults that can take place; i.e., any deviation from the correct functional behavior of a memory. This can be done analytically as the space of the potential memory operations is defined. However, the space is huge and constraints should be made in order to limit the space to a reasonable sized one. Once the space is identified, the fault analysis can take place; stimuli sensitizing each of the faults should be developed and applied to an appropriate memory simulation model while the defective device is replaced with its model. This should be repeated for all possible defects. Next, we will illustrate the above, first for the memory array and thereafter for the key peripheral circuits (i.e., address decoder and sense amplifier).

Fault Modeling for memory array

Fault Space: Memory array faults can be described by *Fault Primitive (FPs)* [41]. A fault is noted in the $\langle \mathrm{S/F/R} \rangle$ notation. In this notation, S denotes the sensitizing sequence for the fault, i.e., $\mathrm{S} = x_0 O_1 x_1 \ldots O_i x_i \ldots O_n x_n$. Here, x_i denotes the cell state, i.e., $x_i \in \{0, 1\}$, O_i denotes the operation that takes place, i.e., $O_i \in \{\mathrm{r}, \mathrm{w}\}$, where r and w indicate a read and write operation, respectively, and n is the number of operations. F denotes the value that is stored in the cell after S is performed, i.e., $\mathrm{F} \in \{\mathrm{H}, 1, \mathrm{U}, 0, \mathrm{L}\}$, where 'U' denotes the undefined state [41], 'H' the extreme logical 1 state, and 'L' the extreme logical 0 state, as demonstrated by measurements performed on defective RRAM and STT-MRAM devices [12, 18]. Finally, R (read output) describes the output of a read operation if the last operation in S is a read operation. $\mathrm{R} \in \{0, 1, ?, -\}$, where ? denotes a random read value (e.g., the sensing current is very close to sense amplifier reference current), and '$-$' denotes that R is not applicable, i.e., when the last operation in S is a write operation.

Given the above S, F, and R, the fault space for the memory array can be defined, like it was done in [19] for *static* single-cell faults. More complex faults such as those involving more than one operation (i.e., *dynamic* faults) or those involving multiple cells (e.g., *coupling* faults) can be defined in a similar manner by extending the FP notation [41].

Fault Analysis: some work on RRAM fault analysis is presented in [12, 13, 42] where the defects were modeled as a linear resistor (LR), and other work in [17, 19] where the authors used Device-Aware (DA) defect modeling. We only illustrate the results for the forming defect as presented in [19]. Table I lists the results of the static single-cell fault analysis; the FPs sensitized when assuming LR (both as a series and parallel resistor) and DA models for the forming defect are shown. The results are obtained by simulating different sizes of the defect. The table clearly highlights the difference between the two approaches. The unique DA faults (7 out of 8 of the realistic faults) cannot be sensitized with LR approach. Moreover, the LR model approach triggers 8 unique faults which are not realistic for forming defects, hence leading to a waste of test time. Note that only 1 common fault is observed by both approaches.

A complete fault analysis should consider each potential defect in the memory array, model it using the DA approach, and thereafter perform defect injection and circuit simulation.

Fault Modeling for some peripheral circuits

Address Decoder: Address decoder faults (AFs) in semiconductor memories are well studied. These faults can be

TABLE I: Validated faults using LR and DA models.

Range	FPs	DA	LR series	LR parallel
5 µA	⟨1w0/L/−⟩	Yes	No	No
[5; 13] µA	⟨1/U/−⟩, ⟨1w1/U/−⟩, ⟨1r1/U/1⟩	Yes	No	No
(13; 34] µA	⟨0/L/−⟩, ⟨0r0/L/0⟩, ⟨0w1/L/−⟩	Yes	No	No
(13; 34] µA; [4k; 40k] Ω	⟨0w0/L/−⟩	Yes	Yes	No
[12k; 16k) Ω	⟨0w1/U/−⟩	No	Yes	No
[16k; ∞) Ω	⟨0w1/0/−⟩	No	Yes	No
[1.6k; 5k] Ω	⟨1w0/U/−⟩	No	Yes	No
(5k; ∞) Ω	⟨1w0/1/−⟩	No	Yes	No
[8k; ∞) Ω	⟨1r1/1/0⟩	No	Yes	No
[0; 12k] Ω	⟨0w1/0/−⟩, ⟨0r0/0/1⟩	No	No	Yes
[0; 3k] Ω	⟨1w0/1/−⟩	No	No	Yes
(3k; 20k] Ω	⟨1w0/U/−⟩	No	No	Yes
[0; 6] Ω	⟨1w1/H/−⟩	No	No	Yes
[0; 1] Ω	⟨1r1/H/1⟩	No	No	Yes

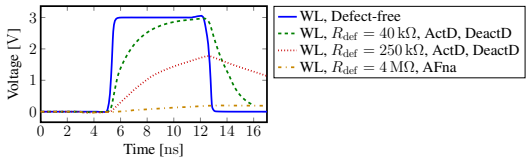


Fig. 11: WL decoder faults

static or dynamic. Static AFs are mainly caused by completely broken interconnects (e.g., wordline) or low ohmic bridges between connections and consist of four possible faults [43]: 1) *No-access* (AFna): an address does not access its cell, 2) *Multiple cells* (AFmc): an address uniquely accesses multiple cells, 3) *Multiple addresses* (AFma): a cell is uniquely accessed by multiple addresses, and 4) *Other cells* (AFoc): an address additionally accesses other cells. On the other hand, dynamic or delay address decoder faults (ADFs) are caused by partial opens and shorts; they consist of two possible faults [44]: 1) Activation delay (ActD): the activation, e.g., of a wordline, is delayed, and 2) Deactivation delay (DeActD): the deactivation, e.g., of a wordline, is delayed. These faults may lead to erroneously addressing of multiple cells at the same time, or to shortening the cell access time which may cause a e.g., a write operation to fail.

Fault analysis for address decoders has been studied also very well by assuming a linear resistor as defect model [44, 45]. For example, Fig. 11 illustrates how an open defect in a WL can cause AFna or ADFs, depending on the defect size.

Sense Amplifier: Sense amplifier faults in semiconductor memories have been well studied [45, 46]. They can be divided into static and dynamic faults. Static faults are assumed to be caused by complete opens, low ohmic shorts to V_{DD} or GND, or low ohmic bridges [43]; they consist of the traditional *SA Stuck-at fault* (SASF), an SASF means that the SA always outputs the same value, independent of its inputs. Dynamic faults are caused by partial opens and shorts and consist of two faults: 1) *Unbalanced SA fault* (USAF) [46]: the SA has a continuous tendency to switch to a certain value under equal input conditions, rather than being balanced, and 2) *Slow SA fault* (SSAF) [45]: the SA is too slow to switch, which may result in incorrect read values.

Fault analysis for SAs has been performed by assuming that any defect can be modeled as a linear resistor. Fig. 12 illustrates the faults that may occur when performing a r0 operation in an SA in the presence of an open defect (R_{def}) between transistors N2 and N4 of Fig. 9a. This defect leads to a slower discharge of node B, as the path to GND now has

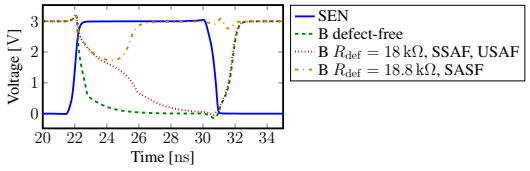


Fig. 12: SA faults

a higher resistance. It can be seen that when $R_{\text{def}} < 18.8\,\text{k}\Omega$, the defect causes an unbalance in the SA and thus is slowing down the sensing operation causing USAFs and SSAFs. When $R_{\text{def}} \geq 18.8\,\text{k}\Omega$, the SA will always switch to the wrong value, thus leading to an SASF.

C. Test Development

The output of the fault modeling (i.e., a set of fault models) is crucial for the development of efficient and high-quality test solutions. Faults can be classified in two categories [19]: strong and weak faults. Strong faults cause functional errors in the memory operation, and can be sensitized (and may be detected) by a known sensitizing sequence. On the contrary, weak faults do not result in any functional error; instead, weak faults are parametric faults, e.g., reduced bit line swing. These faults also need to be detected, as they may pose a reliability risk, e.g., increased in-field failure rate. Moreover, depending on the effort needed to detect them, faults can be divided into easy-to-detect (ETD) and hard-to-detect (HTD) faults. The detection of ETD faults can be *guaranteed* by applying write and read operations, e.g., by using a March test [43]. However, March tests *cannot* guarantee the detection of HTD faults, although they may detect them. Guaranteeing their detection may require additional effort; e.g., the use of a special Design-for-testabilty (DfT) circuitry. An example of an ETD fault is $\langle 1\text{r}1/0/0\rangle$, and an example of an HTD fault is $\langle 1\text{r}1/\text{U}/?\rangle$;

In order to develop appropriate test solutions for the CIM core in its memory configuration, first the obtained faults from fault modeling should be analyzed and classified into ETD and HTD faults and thereafter test solutions should be developed. In the rest of this section, we will illustrate the above for the previously discussed faults for the three components.

Memory Array: Let us consider the results shown in Table I for the forming defect when using Device-Aware fault modeling. The defect can sensitize in total 8 FPs, which can be grouped into 4 fault classes, where a fault class is a set of FPs sensitized by the same single defect with a certain range/size. Inspecting the table reveals that only $\langle 0\text{w}1/\text{L}/-\rangle$ is an ETD fault, while the rest is HTD faults. Detecting $\langle 0\text{w}1/\text{L}/-\rangle$ can be easily done by a March element $\Updownarrow (\text{w}0, \text{w}1, \text{r}1)$.

HTD faults in the memory array are typically related to the cell being in a forbidden state (i.e., 'H', 'U', or 'L') [19]. As already mentioned, March tests may detect some of these faults; repeating tests targeting HTD faults with different memory backgrounds and different address sequence [44, 45] will increase the detection probability. For example, the FP $\langle 1/\text{U}/-\rangle$ may be detected with a March element $\Updownarrow (\text{w}1, \text{r}1)$. However, detection is not guaranteed. Therefore, using DfT is a common practice to further increase the chance of detecting

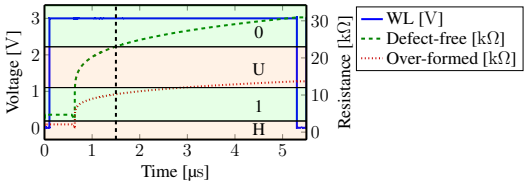


Fig. 13: Defect-free and over-formed cell

HTD faults. For example, the scheme in [13] uses shortened write times and reduced write voltages to detect cells that are in the 'U' state. This scheme can be modified to also detect cells suffering from an over-formed defect; i.e., cells whose state is 'H' instead of '1' after forming, as illustrated in Fig. 13. The figure shows a RESET operation on two cells that are initially in '1' and 'H' (i.e., a 1w0 operation) for the defect-free and over-formed cell, respectively. From the figure it follows that the defect-free cell switches quicker to the correct resistance range than the over-formed cell. The DfT is now used to shorten the write operation time to 1.5 µs (indicated by the black dashed line in the figure). The defect-free cell will have switched to '0', while the over-formed cell is still in '1'. A subsequent read operation on both cells will reveal this, and thus the defective cell can be detected.

Address Decoder: The four static AFs (AFna, AFmc, AFma, AFoc) belong to the ETD faults, while the two ADFs (ActD, DeActD) belong to the HTD faults. It has been shown that the detection of static AFs can be guaranteed by a March test that contains the following two March elements [43, 45]: $\Uparrow (\mathrm{r}x, \ldots, \mathrm{w}\bar{x})$ and $\Downarrow (\mathrm{r}\bar{x}, \ldots, \mathrm{w}x)$; here, $x \in \{0, 1\}$ and $\bar{x}$ denotes the negation of x. The ADFs, however, *may* be detected by March tests; the detection probability strongly depends on the delay [44]. Their detection requires: 1) Sensitizing Address Transitions, and 2) Sensitizing Operation Sequences. Sensitizing address transition(s) can be caused by an address pair or an address triplet. For example, a Sensitizing Address Pair consists of a sequence of two addresses A_f and A_g which have to be applied in sequence because ADFs are sensitized by address transitions. These transitions are generated using an Addressing Method such as Address Complement, The H1 Addressing Method (H1 stands for hamming distance is 1), etc. [44]. On the other hand, the Sensitizing Operation Sequence should be generated and applied to each of the generated address pairs (A_f, A_g); this sequence consists of two operations ($Ox_\mathrm{f}; Oy_\mathrm{g}$), one operation applied to A_f and the other to A_g. O denotes a read or write operation ($O \in \{\mathrm{r}, \mathrm{w}\}$) with expected or written data $x, y \in \{0, 1\}$. The operation on A_g has to be performed with the *complement* of the data value applied to A_f in order to detect e.g., ActD; because of the fault, Ox_f may fail. It is worth noting that each of the decoders should be tested individually; hence the test for address decoders need to be repeated twice for our CIM core in memory configuration.

Sense Amplifier: the static SASF belongs to the ETD faults and its detection can be easily guaranteed by any March test consisting of the two March elements (or a single March element combining both of them): $\Updownarrow (..., \mathrm{r}x, ...)$ and $\Updownarrow (..., \mathrm{r}\bar{x}, ...)$, with $x \in \{0, 1\}$ [45]. Detecting dynamic faults (USAF, SSAF), which belong to the HTD faults may be done with March tests, although special DfT can do a better job. The sensitization and detection of SSAF requires the application of *back-to-back* operations to the memory using 1) *different* data values (0 and 1) and 2) *fast-row* addressing (i.e., each address increment or decrement causes an adjacent physical row to be accessed) [45]; back-to-back operations indicate that the two operations take place after each other without any delay. For example, a test consisting of the following March element (using fast-row addressing) may detect SSAFs: $\Updownarrow (\mathrm{r}x, ..., \mathrm{w}\bar{x})$; the read and write are back-to-back and use different data. E.g., the operation w0 brings the bit lines in the worst case state for the following r1 operation, applied to the next cell in the same column. Special DfT which can be used to complement March tests, can work better in detecting such faults. For example, the DfT proposed in [47] to detect HTD faults in SRAMs can be used here; it is based on monitoring the bit line swing at the input of the SA.

VI. CIM-Pr Computation Configuration Test

This section presents the test approach for the CIM-Pr core in the computation configuration. We follow again the approach that was presented in Section III. Note that the CIM-Pr under consideration is based on scouting logic.

A. Defect Modeling

Obviously, the same defect models apply for this configuration as those discussed in the previous section; they are Linear resistor and Device-Aware defect modeling. Linear resistors are suitable for interconnect defects and have been also shown to do a good job for transistor defects, while Device-Aware defect modeling is suitable for the RRAM defects.

B. Fault Modeling

Next fault modeling will be applied first to the memory array, then to the address decoders and sense amplifiers.

Memory Array: defining the fault space of memory array in the computation configuration is still an open question and can be strongly array design and architecture dependent [48, 49]. The memory array in the computing configuration acts as a special case of dual port memory; it allows for simultaneous access of two cells/locations in the same column. Hence, this may give rise to new faults. For example, accessing two cells simultaneously may unintentionally flip the state of one of them. Defining the fault space will need also the extension of the FP notation $\langle \mathrm{S/F/R} \rangle$. We can build on the notation developed for dual-port memory faults [50]; we denote a FP due to the simultaneously access as $\langle \mathrm{S_1 : S_2 \ / \ F_1 : F_2 \ / \ R} \rangle_{\mathrm{OP}}$, where $\mathrm{S_1}$ and $\mathrm{S_1}$ specify the sensitizing operations, ':' denotes the fact that $\mathrm{S_1}$ and $\mathrm{S_1}$ are applied *simultaneously*, $\mathrm{F_1}$ and $\mathrm{F_2}$ describe the value of the accessed cells after the sensitizing operations, R gives the read value, and OP specifies the operation performed (e.g., AND, OR). For example, $\langle 1\mathrm{r}1_1 : 1\mathrm{r}1_2 \ / \ 1_1 : 1_2 \ / \ 0 \rangle_{\mathrm{AND}}$ describes an AND operation on two cells containing '1' that results in

a wrong output '0'. To illustrate that such a fault is realistic, consider an open defect (R_{defect}) in the bit line that increases its resistance slightly. When the AND operation takes place, the equivalent resistance (see Fig. 10) $R_{\text{eq}} = R_{\text{`11'}} + R_{\text{defect}}$ can become higher than $R_{\text{AND}}$ and thus results in a wrong read output. In the memory configuration, however, no fault occurs as $R_{\text{`1'}} + R_{\text{defect}} < R_{\text{read}}$. Hence, this fault only occurs in the computation configuration. Defining the complete fault space and validating it, is still an open question.

Address Decoder: The Scouting logic computation configuration requires both address decoders to act simultaneously to select the appropriate word lines. This configuration may give rise to unique address decoder faults, and is quite similar to dual-port memories [51]; also here two addresses should be selected simultaneously. Hence, the same fault space and fault models can apply. Such faults are called *port interference faults* and are due to potential interference/bridges between the two decoders (between wires of the two different decoders). They differ from single AFs in the sense that they only occur when two decoders are accessed simultaneously, and not when operating sequentially. E.g., one of the decoder erroneously select an additional world line when the inputs of both decoders have defined value. Consider Fig. 8a and assume the two addresses $\mathrm{A_1A_2A_1A_0}$='1111' and $\mathrm{B_3B_2B_1B_0}$='1110' are selected in a 4-bit WL decoders; these will drive WL0A and WL1B simultaneously. If now a low ohmic bridge defect exist between the node $\mathrm{Y_1}$ of the decoder circuit driving WLB1 and the node $\mathrm{X_2}$ of the decoder circuit driving WLA2 (see Fig. 8a), then the simultaneous selection of WL0A and WL1B will result in erroneous selection of WL2A, i.e., WL0, WL1, and WL2 will be activated.

Sense Amplifier: The modified SA in the computation configuration may suffer from similar faults as the SA in the memory configuration. These faults (consisting of SASF, USAF and SSAF) can take place in each of the computing configurations of the SA including OR, AND, and XOR; note that the modified SA uses different reference currents to perform the different logic operations. The validation of such faults using fault analysis is still an open question.

C. Test Development

Tests for the computation configuration focus on: 1) testing the hardware that was not used during memory configuration test and, 2) on testing of unique faults that may be sensitized due to simultaneous access of the memory array (due to the selection of the operands of the logic operation). The test development approach in the computation configuration is similar to that of the memory configuration. Next we will illustrate the approach for (some of) the faults discussed in previous subsection.

Memory Array: Defining the complete fault space and validating it is still an open question. Nevertheless, we will illustrate how to develop an appropriate test for such faults. Let's consider the fault $\langle 1\mathrm{r}1_1 : 1\mathrm{r}1_2 \ / \ 1_1 : 1_2 \ / \ 0 \rangle_{\text{AND}}$ discussed in the previous subsection. This is an ETD fault as it produces a wrong output 0 instead of 1. If we assume that this fault only takes place when two accessed cells/operands (in the same column) are physically adjacent, then such a fault can be detected by a March test containing e.g., the following two March elements: $\Uparrow_{c=0}^{C-1} (\Uparrow_{r=0}^{R-2} (..., \mathrm{r}1_{r,c} : \mathrm{r}1_{r+1,c}, ...))$. Note that a nested addressing is used; R and C denote the number of rows and columns of the array, respectively. For each column c, cells at row r and $r+1$ are simultaneously accessed by an r1 operation. Note that before such operations are performed, the cells have to be initialized with an appropriate data-background [45] (i.e., the pattern of 1's and 0's as seen in the memory array). For example, a solid 1 background (1111... /1111... /1111...) satisfies this requirement.

Address Decoder: Tests developed for dual-port memory address decoder faults (i.e., port interference faults) [51], can be easily adapted and used for testing the unique address decoder faults in computation configuration. Such tests have a time complexity (in the worst case) of $\mathcal{O}\left(R^2\right)$ where R is the number of array rows.

Sense Amplifier: An SA in each of the computation configuration (e.g., AND, OR) can suffer from the same faults as an SA in the memory configuration; these faults consist of SASF, USAF and SSAF. However, testing such faults will require special attention. For example, to detect the ETD fault SASF in the AND mode, a March test should contain the two March elements (or a single March element combining both of them): $\Updownarrow_{c=0}^{C-1} (..., \mathrm{r}0_i : \mathrm{r}x_j, ...)$ and $\Updownarrow_{c=0}^{C-1} (..., \mathrm{r}1_i : \mathrm{r}1_j, ...)$, where $x \in \{0, 1\}$ and (i, j) two addresses indicating any two cells/operands in the same column. The fault SASF1 will be detected by the parallel operations $\mathrm{r}0_i : \mathrm{r}x_j$ as this will return 1 instead of 0, while SASF0 will be detected by the parallel operations $\mathrm{r}1_i : \mathrm{r}1_j$ as this will return 0 instead of 1. Note that actually performing each of the two parallel operations once is enough for the detection of SSAF, and there is no need to repeat them for different address combinations (i, j). Next, we show how we can detect the SSAF in the AND configuration. As already mentioned in Section V, this fault is HTD and may be detected by a March test when applying *back-to-back* operations resulting in *different* data values (0 and 1) and using *fast-row* addressing. For example, a test consisting of the following March element (using fast-row addressing) may detect SSAFs of the SA in the AND configuration: $\Uparrow_{c=0}^{C-1} (\Uparrow_{r=0}^{R-2} (\mathrm{r}1_{r,c} : \mathrm{r}1_{r+1,c}, \mathrm{w}0_{r,c}, \mathrm{r}0_{r,c} : \mathrm{r}1_{r+1,c}))$; the two parallel operations are back-to-back and result in different data output. For example, the operation $\mathrm{r}0_{r,c} : \mathrm{r}x_{r+1,c}$ results in 0 bringing the SA in the worst case state for the following $\mathrm{r}1_{r,c} : \mathrm{r}1_{r+1,c}$ operation that has to result in 1, applied to the next cells in the same column. $\mathrm{w}0_{r,c}$ is just a write operation. Here also special DfT can be developed to complement March tests, and even do a better job in detecting such faults.

VII. Discussion and Conclusion

This work highlighted the structural testing of CIM dies. Although our case study was based on Scouting logic, the approach is applicable to any CIM design.

Testing CIM dies solely as a memory is not enough, as each computation configuration needs to be tested as well, where

the focus is on testing 1) the partial and completely non-tested hardware during the memory test phase (this hardware consists of the modified or newly added components to the memory), 2) the unique faults that could take place due to simultaneous memory access (e.g., when executing a logic operation).

Although a lot of memory fault models and test solutions can be reused for CIM in the computation configuration, many new solutions are needed. These are strongly CIM architecture dependent. For example, the test solutions for CIM based on Scouting logic will differ from analog vector matrix multiplication with ADCs. Clearly there are still many open questions to be worked out such as:

- Fault modeling: defining the fault spaces for the different CIM architectures in the computation configuration and validating them using realistic design.
- Test solutions and optimization: developing appropriate test solutions (test algorithms, DfT, BIST solutions, etc) for the different architectures; optimizing the test approach by exploring the combination of the test solutions for the memory configuration and the computation configuration, especially for production test.

ACKNOWLEDGMENT

This research on CIM architecture is supported by EC Horizon 2020 Research and Innovation Program through MNEMOSENE project under Grant 780215.

REFERENCES

[1] D. A. Patterson, "Future of Computer Architecture," in *BEARS*, 2006.
[2] S. Hamdioui *et al.*, "Memristor for Computing: Myth or Reality?" In *DATE*, 2017.
[3] E. Linn *et al.*, "Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, 2012.
[4] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *DATE*, 2015.
[5] S. Hamdioui *et al.*, "Applications of Computation-In-Memory Architectures based on Memristive Devices," in *DATE*, 2019.
[6] D. Fujiki *et al.*, "In-Memory Data Parallel Processor," in *ASPLOS*, vol. 53, 2018.
[7] H.-S. P. Wong *et al.*, "Metal-Oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, 2012.
[8] H. A. Du Nguyen *et al.*, "On the Implementation of Computation-in-Memory Parallel Adder," *IEEE TVLSIS*, vol. 25, no. 8, 2017.
[9] Y. Chen *et al.*, "Recent Technology Advances of Emerging Memories," *IEEE Des. Test*, vol. 34, no. 3, 2017.
[10] S. Hamdioui *et al.*, "Test and Reliability of Emerging Non-volatile Memories," in *ATS*, 2017.
[11] S. Kannan *et al.*, "Sneak-Path Testing of Crossbar-Based Nonvolatile Random Access Memories," *IEEE TN*, vol. 12, no. 3, 2013.
[12] C. Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE TC*, vol. 64, no. 1, 2015.
[13] S. Hamdioui *et al.*, "Testing Open Defects in Memristor-Based Memories," *IEEE TC*, vol. 64, no. 1, 2015.
[14] Chin-Lung Su *et al.*, "MRAM defect analysis and fault modeling," in *ITC*, 2004.
[15] J. Azevedo *et al.*, "A Complete Resistive-Open Defect Analysis for Thermally Assisted Switching MRAMs," *IEEE TVLSIS*, vol. 22, no. 11, 2014.
[16] X. Pan *et al.*, "Modeling and test for parasitic resistance and capacitance defects in PCM," in *NVMTS*, 2012.
[17] M. Fieback *et al.*, "Testing Resistive Memories: Where are We and What is Missing?" In *ITC*, 2018.
[18] L. Wu *et al.*, "Pinhole Defect Characterization and Fault Modeling for STT-MRAM Testing," in *ETS*, 2019.
[19] M. Fieback *et al.*, "Device-Aware Test: A New test Approach Towards DPPB Level," in *ITC*, 2019.
[20] A. Sebastian *et al.*, "Temporal correlation detection using computational phase-change memory," *Nat. Comm.*, vol. 8, no. 1, 2017.
[21] B. Chen *et al.*, "Efficient in-memory computing architecture based on crossbar arrays," in *IEDM*, 2015.
[22] M. A. Lebdeh *et al.*, "Memristive Device Based Circuits for Computation-in-Memory Architectures," in *ISCAS*, 2019.
[23] P.-E. Gaillardon *et al.*, "The Programmable Logic-in-Memory (PLiM) computer," in *DATE*, 2016.
[24] D. Bhattacharjee *et al.*, "Revamp: Reram based vliw architecture for in-memory computing," in *DATE*, 2017.
[25] S. Hamdioui *et al.*, "Computing device for big data applications using memristors," in *US Patent 9,824,753*, 2017.
[26] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Comp. Arch. News*, vol. 44, 2016.
[27] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016.
[28] H. A. D. Nguyen *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *VLSI-SoC*, 2017.
[29] L. Xie *et al.*, "Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing," in *ISVLSI*, 2017.
[30] M. Le Gallo *et al.*, "Mixed-precision in-memory computing," *Nat. Elec.*, vol. 1, no. 4, 2018.
[31] L. Chua, "Memristor-The missing circuit element," *IEEE TCT*, vol. 18, no. 5, 1971.
[32] S. Yu *et al.*, "Emerging Memory Technologies: Recent Trends and Prospects," *IEEE SSC*, vol. 8, no. 2, 2016.
[33] M. L. Bushnell *et al.*, *Essentials of Electronic Testing for Digital Memory & Mixed-Signal VLSI Circuits*. Springer Science+Business Media, 2000.
[34] H. H. Chen, "Beyond structural test, the rising need for system-level test," in *VLSI-DAT*, 2018.
[35] W. Zhao *et al.*, "Synchronous Non-Volatile Logic Gate Design Based on Resistive Switching Memories," *IEEE TCSI*, vol. 61, no. 2, 2014.
[36] K. J. Kuhn *et al.*, "Process Technology Variation," *IEEE TED*, vol. 58, no. 8, 2011.
[37] E. I. Vatajelu *et al.*, "Challenges and Solutions in Emerging Memory Testing," *IEEE TETC*, 2017.
[38] H. Y. Lee *et al.*, "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," in *IEDM*, 2010.
[39] M. Lanza *et al.*, "Grain boundaries as preferential sites for resistive switching in the HfO2 resistive random access memory structures," *Appl. Phys. Lett.*, vol. 100, no. 12, 2012.
[40] L. Wu *et al.*, "Electrical Modeling of STT-MRAM Defects," in *ITC*, 2018.
[41] S. Hamdioui *et al.*, "An experimental analysis of spot defects in SRAMs: realistic fault models and tests," in *ATS*, 2000.
[42] Y.-X. Chen *et al.*, "Fault modeling and testing of 1T1R memristor memories," in *VTS*, 2015.
[43] A. J. van de Goor, *Testing Semiconductor Memories - Theory and Practice*. John Wiley & Sons, 1991.
[44] S. Hamdioui *et al.*, "Opens and Delay Faults in CMOS RAM Address Decoders," *IEEE TC*, vol. 55, no. 12, 2006.
[45] A. van de Goor *et al.*, "Detecting faults in the peripheral circuits and an evaluation of SRAM tests," in *ITC*, 2004.
[46] K. Zarrineh *et al.*, "Defect analysis and realistic fault model extensions for static random access memories," in *IWMTDT*, 2000.
[47] G. C. Medeiros *et al.*, "DFT Scheme for Hard-to-Detect Faults in FinFET SRAMs," in *ETS*, 2019.
[48] D. Niu *et al.*, "Low power memristor-based ReRAM design with error correcting code," in *ASP-DAC*, 2012.
[49] B. Zhao *et al.*, "Common-source-line array: An area efficient memory architecture for bipolar nonvolatile devices," *ACM TODAES*, vol. 18, no. 4, 2013.
[50] S. Hamdioui *et al.*, "Efficient tests for realistic faults in dual-port srams," *IEEE TC*, vol. 51, no. 5, 2002.
[51] S. Hamdioui *et al.*, "Address decoder faults and their tests for two-port memories," in *IWMTDT*, 1998.

C

Rebooting Computing: The Challenges for Test and Reliability

This appendix contains the paper published at International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems: A. Bosio et al., "Rebooting Computing: The Challenges for Test and Reliability," 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Noordwijk, Netherlands, 2019, pp. 8138-8143.

Rebooting Computing: The Challenges for Test and Reliability

A. Bosio[1], I. O'Connor[1], G. S. Rodrigues[2], F. K. Lima[2], E. I. Vatajelu[3], G. Di Natale[3], L. Anghel[3], S. Nagarajan[4], M. C. R. Fieback[4], S. Hamdioui[4]

[1]*INL - École Centrale de Lyon, France* – Email: alberto.bosio@ec-lyon.fr

[2]*Instituto de Informatica, PGMicro - Universidade Federal do Rio Grande do Sul, Brazil* – Email: gsrodrigues@inf.ufrgs.br

[3]*Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, France* – Email: {firstname.lastname}@univ-grenoble-alpes.fr

[4]*Computer Engineering Lab, Delft University of Technology, The Netherlands* – Email: S.Hamdioui@tudelft.nl

***Abstract*—Today's computer architectures and semiconductor technologies are facing major challenges making them incapable to deliver the required features (such as computer efficiency) for emerging applications. Alternative architectures are being under investigation in order to continue deliver sustainable benefits for the foreseeable future society at affordable cost. These architectures are not only changing the traditional computing paradigm (e.g., in terms of programming models, compilers, circuit design), but also setting up new challenges and directions on the way these architectures should be tested to guarantee the required quality and reliability levels. This paper highlights the major open questions regarding test and reliability of three emerging computing paradigms being approximate computing, computation-in-memory and neuromorphic computing.**

***Index Terms*—Alternative computing architectures, emerging technology, fault model, test, reliability**

I. Introduction

Energy and computer efficiency is undoubtedly one of the major driving forces of current computer industry, which is relevant not only for supercomputers, but also for small portable personal electronics and sensors. However, today's computing architectures (mainly based on the CMOS technology) are facing major challenges making them unable to meet the requirements. Such challenges are: power wall, memory wall and Instruction Level Parallelism wall [1], [2]. For example, the memory wall is due to the increasing gap between processor and memory speeds, which limits the data transfer time and leads to significant energy consumption during the data transfer varying from 70% up to 90% of the overall energy spent by the computing system [3]. Moreover, even the dominating CMOS technology (which made manufacturing of computers feasible) is suffering, especially nodes below 20 nm. At this level the physical characteristics of such devices are leading to high static power consumption, reduced reliability; not to mention increased cost [4]. All of these have led to saturated computer performance and the slowdown of the traditional device scaling, making today's computing systems unable to deliver the required computing and energy efficiency. For example, artificial intelligence is ready to provide solutions in many domains; however, the resource and power demands of the underlying algorithms and implementations are way too high for the target applications. For instance, the amazing performance of AlphaGo [5] required 4 to 6 weeks of training executed on 2000 CPUs and 250 GPUs for a total of about 600kW of power consumption (while the human brain of a go player requires about 20W). Due to these limitations, many alternative architectures and technologies (being able to deliver the required demands at affordable cost) are under investigation; examples are approximate computing [6]–[8], computation-in-memory [9]–[11], and neuromorphic computing [12]–[14]. These will not only change the way we used to design and program our computers, but also the way we used to test them to provide the required quality and reliability. Providing high-quality testing is a very critical step in the commercialization of any electronic product responsible for screening out all the defective chips before they are sold.

Testing and design-for-test for emerging computing paradigms such as the three mentioned above is still in an infancy stage, and almost no work is published in this field. Understating the related challenges and setting up directions toward the development of efficient solutions is of great importance in order to provide appropriate solutions. This paper addresses the test and reliability related challenges for three emerging computing paradigms being approximate computing, computation-in-memory, and neuromorphic computing. It presents the actual state of the art and aims also at providing some preliminary results and setting up some research directions.

The paper is structured as follows. Section II covers the design of low-cost fault tolerant mechanisms exploiting the Approximate Computing paradigm. Section III presents the Computation-in-Memory paradigm and its test and reliability challenges and sets up some directions. Section IV focuses on a comprehensive fault model dictionary for HW-based Spiking Neural Networks with online learning (during learning and inference) and methodologies test for such faults. Finally Section V concludes the paper.

II. Exploiting Approximate Computing for implementing Low-cost Fault Tolerant Mechanisms

Approximate computing has been proposed to achieve energy efficient computation at the cost of accuracy reduction [15]. Hardware designs can profit from approximation to generate circuits with smaller area, thus reducing energy consumption and delay. Software projects use approximation mainly to reduce memory footprint and execution time. Approximation also impacts the system fault tolerance due to its nature [16]. Approximate computing algorithms already handle small inaccuracies generated by the approximation. Thus, very small data corruption errors might not even be noticed by the system as a whole. Some approximation strategies are also inherently fault tolerant. Such is the case of successive approximation: an approximation method that consists of loop executions generating an ever-improving output. This approximation method can also work as a fault tolerance mechanism by itself, given that an error affecting one iteration of the loop can be corrected on the following ones [17]. A designer can use loop perforation to balance execution time and accuracy on successive approximation algorithms, which also

*This work has been partially founded by CNRS PICS07968 project.

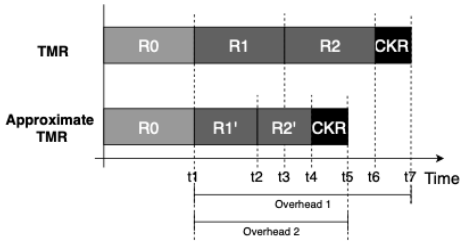


Fig. 1. Approximate TMR diagram.

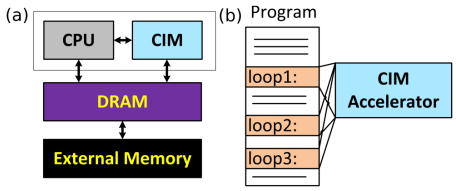


Fig. 2. (a) CIM as an accelerator (b) Example of a program

impacts the fault tolerance of the system [17]. Another very common approximation method is data size reduction [8], which consists of representing data with less bits than usual. This method has little-to-none impact on software execution time but can highly reduce memory footprint.

Numerical and mathematical properties can also be used to provide valid functional approximation. Taylor series, for example, are used in mathematics to represent a function as a sum of previously calculated terms. The more terms are used, the more accurate the approximation. This type of method can be applied both to software and hardware designs, with different costs [18]. On hardware, the price to pay for more accuracy is either more hardware area or a higher delay: a designer might choose an implementation with pipelines to make it faster (and bigger) or a smaller, loop-execution circuit with a higher delay. On software, the price to pay for this type of approximation is always the execution time. Even on parallel systems, where multiple terms could be executed concurrently, this execution would take processing resources that could otherwise be used to improve the system's performance. Naturally, this approximation method also has a high impact on the system fault tolerance: using bigger hardware increases the probability of a fault, due to a higher number of critical bits. Algorithms with higher execution times are also known to have a higher susceptibility to errors [19], given that they are exposed to more faults per second (in a real use-case scenario of the system execution in a hazardous environment).

Approximate computing can also be used to reduce the costs of traditional fault tolerance methods. Triple modular redundancy (TMR) is one of the most studied fault tolerance and error masking methods in the literature [19]. In its more traditional form, it consists of triplicating a circuit or software code and implementing a checker to verify the consistency of the three execution outputs. If one of the outputs is different from the other two, it shall contain an error that can be masked by the method by accepting the output from the other redundancies as the correct one. Triplicating a whole portion of the system, however, has a high cost (at least 300% area overhead, or execution time for non-parallel software). Approximate computing can be used to provide approximate low-cost redundancies, thus reducing the fault tolerance method costs.

Approximate TMR (ATMR) consists of implementing a TMR with approximate redundancies. It can be applied to both hardware and software projects. Nevertheless, ATMR has to deal with the accuracy loss inherent to approximation. On a traditional TMR approach, the three output values can be compared and checked for errors by a simple bitwise operation. However, an ATMR method needs to handle a possible accuracy difference between the three redundancies. One way of dealing with approximation on ATMR is defining design spaces and assuring that, even in the absence of faults, at least two results will always have the same output [20]. This technique assures that a possible difference caused by the approximation will not turn into an error in the absence of faults. Another way of dealing with the approximation issue on the ATMR checker is with difference thresholds. In this case, the ATMR checker shall only consider an error if the difference between the redundancies outputs is higher than a given threshold. This threshold is defined by the system inaccuracy acceptance. Fig. 1 depicts an example of ATMR compared to the TMR. It can be noticed that ATMR will execute tasks R1' and R2' that are approximate version of the task R0. In this way the overall execution time (t5) will be lower than the TMR execution time (t7).

Some safety-critical systems, in special real-time systems, might not need error masking. Real-time systems deal with *data freshness* requirements, which define time intervals on which data is considered to be updated and valid. A navigation system, for example, might present an error in the data that comes from a radar scan, but because new data coming from a new scan will be generated soon the erroneous data will be overwritten (or even become useless) shortly. In those cases, error masking might be not only unnecessary but also impracticable due to the short data freshness time interval. It is, however, important for the user to know if the current data is to be trusted or not. In an avionics system, for instance, a pilot must know if the date he sees in a panel is trustworthy or not, and take safety measures if needed. Approximation can be used to provide cheap redundancy to mathematically predict if a certain data is inside a possible window of value, and warn the user in the case where the data is absurd [21].

III. Computation-in-Memory: Test and Reliability

Computation-in-Memory (CIM) is one of the alternative computing architectures being explored in the light of emerging new memristive device technologies [3], [22], [23]. CIM aims at eliminating the communication bottleneck while supporting massive parallelism. Although, the ideal would be to fully integrate the processing units and the memory in the same physical location, it is not clear if this is technology-wise feasible. One potential realistic implementation is to use the CIM die as an on-chip accelerator as shown in Figure 2(a) [24]. The CIM die may consist of: (a) a very dense crossbar memory array where memristive devices are fabricated at each junction of the crossbar, and (b) a peripheral circuitry (realized using CMOS technology) that is responsible for the communication and control with the crossbar. The philosophy behind the CIM accelerator is to get the intense memory access part of an application (e.g., due to bad data locality, or big data sizes) to be executed within the CIM die rather than by the CPU; this leads to significant energy saving and performance improvement. Figure 2(b) illustrates a program that could be executed efficiently on this architecture; multiple loops can be executed on the CIM die, while the other parts of the program can be executed on the conventional core. Each time a loop is invoked, the CPU sends a "macro-instruction" (complex instruction) to the CIM die which decodes and executes it locally, before returning the results.

As the name indicates, CIM takes place within the memory core (CIM die). As the CIM die consists of a memory array and the peripheral circuits, and depending where the result of the computation is produced, CIM can be divided into two classes [25]:

- CIM-Array (CIM-A): the computing result is produced within the memory array. Hence, the output should be stored in a memristive device in the array in form of a resistance state.
- CIM-Periphery (CIM-P): the computing result is produced within the peripheral circuitry. Given the fact that memory

periphery is based on CMOS technology, the nature of the produced output is voltage.

It is worth noting that even though the computational results are produced in the array/peripheral circuits, the peripheral circuit/memory array could be a substantial component in the computations. For example, when multiple rows are activated simultaneously in the array, different logic and arithmetic operations can be realized in the periphery [11], [23], [26]. Hence, both CIM-A and CIM-P impact the design of the memory, although the impact of CIM-A could be more severe.

A. Test Challenges

CIM accelerators cannot be tested in the same way as traditional memory structures. This stems from the fact that they operate in two different configurations: memory and computation.

- In the *Memory* configuration, the CIM accelerator behaves like a memory. Hence, testing the *storage* functionality is needed.
- In the *Computation* configuration, the CIM accelerator is able to perform operations on the stored data. Hence, testing of the *computing* functionality is needed.

The CIM accelerator switches between these configurations by modifying the way in which some components (e.g. the sense amplifiers, decoders [26]) perform their function. To maximize fault coverage, it must be ensured that a test targets both configurations. This division of configurations directly leads to increased complexity in the development of test solutions. Note that in theory both functional and structural testing could be used; however, due to its efficiency and measurable coverage, structural testing is more suitable. Next, test challenges for the memory configuration and the computation configuration are discussed.

Testing CIM as memory: CIM accelerator typically consists of a crossbar memristive devices where each device could be e.g., a RRAM, STT-MRAM or a PCM memory device. Although some test and design-for-testability (DfT) schemes for such memories have been developed [27]–[30], there are still many open questions. The most important one arises from the lack of good defect models for the memristive devices. Traditionally, fault modeling is based on (linear) resistor injection and (SPICE) circuit simulation. However, due to the non-linear nature of the memristive device, it becomes questionable if the traditional approach could be sufficient. Recent work on RRAM and STT-MRAM [31], [32] has revealed the need of a new fault modeling approach in order to appropriately and accurately model the fault behavior of memristive devices. In addition, it has demonstrated that the traditional approach may lead to erroneous fault models; hence low quality solutions. Appropriate defect modeling needs to incorporate the impact of a defect on the technological parameters as well as on the electrical parameters of the memristive device in order to derive the way one particular defect manifest itself at the electrical/functional level. Clearly this will result in new fault models which will require new test solutions and Design-for-test (DFT) solutions. Depending on the nature of the fault model and their detection conditions, different test schemes may be needed. For example, the detection of a fault resulting into a non-deterministic or random read value cannot be guaranteed with a March test and a specific DFT will be needed. Furthermore, it is worth to note that the most popular defects and their occurrence probability (or importance) is not clear yet; obviously there is a lack of industry data in the public domain which make it for researchers harder to make the right trade-offs.

Testing CIM in the computing configuration: Testing CIM for memory functionality does not necessarily cover the computing functionality. For example, the peripheral circuit of the CIM die may performs logic or arithmetic operations in the computing configuration, while it acts just as a write or a read path in the memory configuration. To illustrate the additional complexity computing brings to

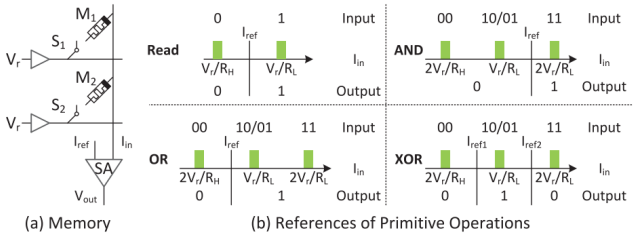


Fig. 3. Scouting logic

the testing of CIM die, let's consider Scouting logic as an example, shown in Figure 3 [26]. Figure 3(a) presents a simplified design of a crossbar memory with 2 bits (M1 and M2), two wordlines selectors (presented by S1 and S2), and a common sense amplifier (SA) used to read the data. Reading a memory cell means selecting the appropriate wordline and sensing the current through the SA. By slightly modifying the SA design, Scouting logic enables the execution of bit-wise OR, AND, and XOR logic functions; this is done based on reading e.g. two rows simultaneously and activating the required reference current for the SA in order to distinguish the right outputs as shown in Figure 3 (b) for AND, OR and XOR. As the example reveals, realizing the computation configuration requires the design changes of at least in the address decoder and sense amplifier; the address decoder (AD) should be able to select multiple rows (for bits to be operated on) and the SA should be able to be set in the right configuration to perform the selected logic operations by choosing the right reference current to be compared to the read current. Hence, for a CIM die with Scouring logic, additional tests should be performed to detect potential defects in the ADs and the SAs.

Testing CIM address decoder (AD): One can assume that fault models and tests used for ADs in traditional multi-port memories can be applicable here as well [33]. However, more accurate investigations to explore the impact of defects in AD on the computing functionally and how they can be detected are still open questions.

Testing for SAs: Also here one can assume that the fault models used to for SAs in traditional memories can be applied [34]; the faults could be static (e.g., stuck-at-fault) or dynamic (e.g., a partial open causing the SA to be slow). For tests, special algorithms should be developed; these should be able to cover the faults and guarantee that the configuration of the SA for different reference currents to realize different logic operation is fault free.

The above example clearly shows that the development of fault models and test solutions of CIM in its computing configuration is quite complex and design dependent; hence it requires special attention. For instance, if the periphery circuit is performing a vector matrix multiplication, then the fault models and the test solutions required may be different from those required by CIM with Scouting logic. Testing for CIM in its computing configuration means identifying the peripheral components with more than one configuration, develop appropriate fault models, and thereafter test solutions.

B. Reliability Challenges

Emerging memory technologies introduce new reliability challenges in the devices, that in turn affect the system reliability. These reliability issues pose a limitation on the scalability of the circuits, as they can generate read and write errors or have unwanted device interactions. To achieve high-quality CIM, it is necessary to understand what these new challenges are and what causes them. We list the most important ones: endurance, variability, and retention.

1) Endurance: The endurance of a storage element is defined as the number of switching cycles a device can perform until it

breaks down and becomes unable to switch. Emerging memory technologies have already shown better endurance than flash memories. However, their endurance is still rather low in comparison with SRAM and DRAM (10^{15} cycles for SRAM vs. $10^{6\sim12}$ cycles for emerging memories) [35]. Because CIM circuits access the storage elements frequently, the device endurance needs to be increased in order to have a highly reliable circuit [36].

2) Variability: The stochastic nature of the filament growth and dissolution in an RRAM device causes cycle-to-cycle variability [37]. That is, when a filament grows, its shape will differ with respect to other cycles, and hence have a different resistance. The shape of the filament depends on many factors. An important one of them is the current that flows through the device when the filament is formed [38]. If the variability of a device is too large, soft faults may occur. For example, a storage element may store an unexpected logical value. This in turn causes operational faults in the computation configuration. Therefore, variability needs to be controlled. This can be done by optimizing the device structure [39], or by applying write verification schemes [40].

3) Retention: After a certain amount of time, the storage element can fail to retain its data, e.g. when the RRAM filament has dissolved, or the polarization of an Spin Transfer Torque (STT) device has flipped. The time it takes for the failure to occur depends on the operating or storing conditions of the device. Temperature [36] and the applied voltages [41] have the most impact among them. Higher temperatures and higher voltages lead to a decrease of retention time. The retention capabilities can be improved by optimizing the production process [42], but care should be taken to prevent the loss of data.

IV. Neuromorphic Computing Paradigms and Test/Reliability Issues

In the post Von Neumann architectures context, neuromorphic computing paradigm has a huge potential when it makes use of emerging NV technologies (STT-MRAM, memristors), however, reliable and testable HW designs enabling the neuromorphic computing are still missing. The Spiking Neural Networks (SNN) are widely studied nowadays due to the high level of realism they bring to neural simulation, their energy efficiency and their ability for on-line learning. The related bio-inspired learning rule is known as STDP (Spike Based Dependent Plasticity) and is applied on each synapse independently of the global state of the network. In return, the synapse must be doted of computation capabilities. A hardware implementation of an SNN requires architectural co-localization of the processing and memory (non-Von Neumann architecture). The circuits solutions used to implement silicon neurons are application dependent, but the vast majority are built with a temporal integration block, a spike generation block, a refractory period mechanism, and a spike adaptation block [12]. Synapses are required to exhibit plasticity (i.e., modulation in their efficacy) and to support online learning algorithms, that manifest in changes in their strengths. Emerging memory devices can be used as synaptic elements thanks to their tunable conductivity, compatibility with advanced CMOS fabrication process, low power consumption, non-volatility and scalability. The synaptic conductance modulation can be emulated using: (i) the analog approach (cumulative decrease and increase of resistance), where multiple resistance states emulate long-term potentiation and depression; or (ii) the binary approach, uses two distinct resistance states per device associated with a probabilistic programming scheme [13]. The strong restrictions on the size of embedded Spiking Neural Network architectures (limited silicon area and interconnectivity ability) require minimization of the network redundancy which in turn reduces its the intrinsic fault tolerance. We postulate that there is an acute need to evaluate the reliability and perform manufacturing test of the neuromorphic

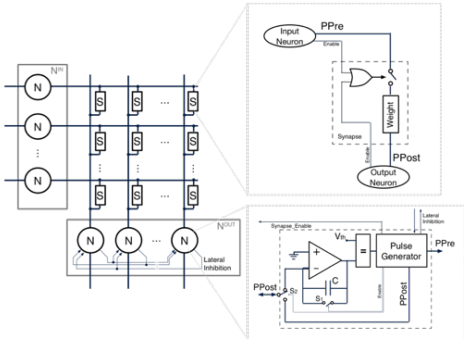


Fig. 4. Schematic representation of the SNN architecture under study, with detailed view of the integrate and fire neuron and the artificial synapse.

hardware architectures to guarantee their correct operation and robustness. Our preliminary analysis supports this research hypothesis by showing that fabrication- and environmental-induced parameter variations affect the neuron/synaptic behavior, which in turn affects the robustness of the SNN [14], [43]. Reliability analysis, post fabrication test, design-for-test and design-for-reliability are commonly used when dealing with traditional computing architectures, however, they are not common practice when dealing with neuromorphic structures. In this context, there are several research works focusing on the fault tolerance (and how it can be improved) of artificial neural networks (ANNs) [44], on boosting fault tolerance of hardware implemented neural accelerators [45], and even on the effect of fabrication-induced variability of memristive devices on the behavior of deep networks [46] and SNNs [47]. These papers show that faulty neurons have stronger impact on the neural network's behavior than faulty synapses. In addition, it is shown that the on-line learning algorithm used in SNNs is efficiently mitigating the effect of synapse variability on the network robustness. However, to the best of our knowledge, the effect of continuous learning (i.e., updating of synaptic weights) on the network lifespan due to limited synapse endurance has not yet been studied.

Performing post fabrication test on a hardware implemented spiking neural network based on emerging memory devices is not a trivial task. It involves testing the integrity and functionality of the neurons and of the synaptic arrays. In addition, the emerging technologies are facing various fundamental research and scientific challenges that are mostly related to manufacturing yield and reliability. They are built with novel materials and subjected to novel operation modes. These all result in novel fault models translating in new dependability issues and a shift in the test paradigm. The defect rates, fault modeling and test solutions for emerging-memory based RAM arrays have been (and still are) extensively studied [48]. Nevertheless, there is no fault modeling, or post-fabrication test solution provided dedicated to alternate operation modes of the memory arrays (such as analog data storage in the case of memristors, or stochastic programming in the case of spintronic devices).

In this context, our work focuses on a fully-connected SNN, that learns using the Spike Timing Dependent Plasticity (STDP) method with lateral inhibition, with integrate-and-fire neuron and resistive synapses. The considered architecture is illustrated in Fig. 4 and described in detail in [49]. In order to achieve the ambitious goal of designing robust and efficient hardware implemented SNNs, one has to jointly-consider the characteristics of the SNN itself (connectivity, neuronal activation function, learning rule and synaptic update), the characteristic of the devices used to implement it (CMOS ON/OFF current and threshold voltage, conductivity modulation and current-compliance of the synaptic devices, etc.) and the environment in which the circuit will be deployed.

In this section we present an overview of fault models pertinent to an SNN with on-line unsupervised learning and the estimated severity of fault injection with respect to the recognition error of the affected neuromorphic architecture. We have defined fault models to enable fault injection campaigns and to allow us to identify scenarios of faulty operations, happening before and after the STDP learning. So far, we have considered only permanent faults caused by manufacturing defects and aging-related phenomena. Due to the fact that there are a large number of SNN circuit implementations, and the number keeps growing, we have defined fault models which do not take into consideration the micro-architecture of the functional units, i.e. neuron and synapse, only their behavior. In particular, we have defined how the inputs and outputs of the functional interface of the neurons and synapses can be affected by the faults, while considering the hardware root causes that can lead to those faults. These faults are similar to, for instance, the stuck-at, where the fault is defined at the interface of a logic gate, without the knowledge of the actual transistor-level implementation of the gate, but still being representative of the majority of physical defects that may appear at the transistor level. In this way we have defined the following fault models: DSF (dead synapse fault), DPF (degraded plasticity fault), SSA0, SSA1 (Synaptic stuck-at-0, Synaptic stuck-at-1), DNF (dead neuron fault), ISLIF, OSLIF (input/output stuck lateral inhibition fault), IDSF and ODSF (input/output delayed spike fault), IDSAF and ODSAF (input/output delayed synapse activation fault), IDLIF and ODLIF (input/output delayed lateral inhibition fault). A complete description of the defined fault models is presented in [50].

Starting from the behavioral model of the SNN under study, we have evaluated the functional accuracy of the SNN during inference and learning under different scenarios of fault injection, in our attempts to answer questions such as: which one is more detrimental to the functionality of a Neural Network (NN): defective neuron or defective synapse? How many of these critical components have to fail such that the entire network fails? In which state does a certain defect matter the most: learning or inference?

We have implemented a spiking neural network with learning strategy based on spike-timing dependent plasticity. The network is designed to solve the MNIST database [51], i.e., to be trained to recognize hand written digits. This data base has 60000 examples for the network training and 10000 examples for testing the network. Each example consists in the image of a hand-written digit. The hand-written digit is a 28x28 pixels image in grey-scale (256 tones of grey from white to black). The information carried by each image is transmitted to network in the form of spikes. The spike encoding is performed by frequency encoding of each pixel's tone of grey. With this encoding, the black pixels carry no information, while the white pixels carry the maximum amount of information, i.e., maximum frequency (255 spikes per time unit). Each image is presented to the network for 10 time units. In order to respond to the requirements of this data base, the network is designed with 784 input neurons, one for every image pixel. The input neurons are connected in a one-to-all fashion (as illustrated in Fig. 4) to the output neurons.

The results of the fault injection campaign are summarized in Fig. 5. It is important to note that different faults have different effects if they happen during the learning or during the inference stages of a network operation. Indeed the synaptic faults (DSF, DPF and SSAx) have a stronger influence during the inference stage of the SNN than during the learning stage. This is due to the fact that the network manages to learn around the faulty synapses due to the on-line learning algorithm (STDP). If the fault occurs during the inference stage, we observe a fast degradation of the recognition rate, due to the fact that the network is found in the situation of recognizing degraded patterns. The location of occurrence of synaptic faults is also very important as stated in the most-right column of the table in Fig. 4. Indeed, if a fault (DSF, DPF or SSA0) occurs on a minimum

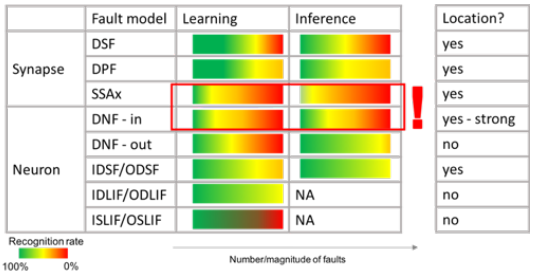


Fig. 5. Summary of SNN accuracy under fault injection.

weight - depressed synapse no effect will be observed on the network behavior. However, if a fault such as DSF, DPF or SSA0 occurs on a maximum weight - excited synapse a strong effect will be observed on the network behavior. The situation is the exact opposite for the synapses affected by SSA1.

During neuron-related fault injection campaigns (DNFin, DNFout, IDSF/ODSF, IDLIF/ODLIF and ISLIF/OSLIF) we observe the opposite effect, i.e., a stronger influence during the learning stage of the SNN than during the inference stage. This is due to the fact that at this stage, the computation element is affected, which means that during learning mode the injected fault leads to wrong behavior learning, while a fault injected during the inference leads to less recognition accuracy. Faults affecting the input neurons are the most critical, since these neurons encode the information. The effect of DNFin is strongly dependent on the location of the faulty neuron. Faults affecting the output neurons are less catastrophic due to the intrinsic redundancy of the SNN networks with STDP, where a pattern is learned by multiple output neurons. Stuck-at fault occurring at lateral inhibition stage is the most critical, since even a single fault can cause full system failure. Indeed, if a OSLIF fault occur on one neuron, it will prevent all other neurons from firing, hence only a single pattern will be learned by the network containing features from multiple patterns, making the network unusable.

This analysis represents a preliminary study of the fault tolerance of SNNs. Further evaluations are necessary to be able to evaluate, with high confidence the reliability of a SNN. Multiple fault injection scenarios need to be further performed to have a full picture of the network accuracy: different locations, different fault magnitudes should be studied as well as plausible clustering scenarios and combinations between synaptic and neural faults. In addition, the network should be evaluated under different application scenarios (or databases with same dimensionalities) to evaluate the fault effects also independently of the application.

V. Conclusion

In this paper we presented the test and reliability challenges for three emerging computing paradigms being approximate computing, computation-in-memory, and neuromorphic computing. Despite the existence of some works, test and reliability for both emerging computing architectures and technologies still needs to be systematically addressed such as defect modelling, fault modelling, test generation and test application.

References

[1] B. Hoefflinger, "Chips 2020," *The Frontiers Collection*, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23096-7

[2] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[3] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1718–1725.

[4] S. Hamdioui *et al.*, "Memristor for Computing: Myth or Reality?" in *Proc. Conf. Des. Autom. Test Eur.* European Design and Automation Association, 2017, pp. 722–731.

[5] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[6] Q. Xu *et al.*, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[7] L. Anghel *et al.*, "Test and reliability in approximate computing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, Aug 2018. [Online]. Available: https://doi.org/10.1007/s10836-018-5734-9

[8] S. Rehman *et al.*, *Heterogeneous Approximate Multipliers: Architectures and Design Methodologies*. Springer International Publishing, 2019, pp. 45–66.

[9] J. Yu *et al.*, "Memristive devices for computation-in-memory," in *Design, Automation and Test in Europe DATE*, 2018.

[10] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, p. 873, 2010.

[11] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.

[12] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, p. 73, 2011. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2011.00073

[13] M. Suri *et al.*, "Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 4.4.1–4.4.4.

[14] E. I. Vatajelu *et al.*, "Reliability analysis of mtj-based functional module for neuromorphic computing," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2017, pp. 126–131.

[15] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[16] G. S. Rodrigues *et al.*, "Evaluating the behavior of successive approximation algorithms under soft errors," in *2017 18th IEEE Latin American Test Symposium (LATS)*, March 2017, pp. 1–6.

[17] G. S. Rodrigues *et al.*, "Exploring the inherent fault tolerance of successive approximation algorithms under laser fault injection," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, March 2018, pp. 1–6.

[18] G. S. Rodrigues *et al.*, "Analyzing the use of taylor series approximation in hardware and embedded software for good cost-accuracy tradeoffs," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros *et al.*, Eds. Cham: Springer International Publishing, 2018, pp. 647–658.

[19] ——, "Performances vs reliability: how to exploit approximate computing for safety-critical applications," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, July 2018, pp. 291–294.

[20] I. A. Gomes *et al.*, "Exploring the use of approximate tmr to mask transient faults in logic with low area overhead," *Microelectronics Reliability*, vol. 55, no. 9, pp. 2072 – 2076, 2015, proceedings of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271415300676

[21] G. S. Rodrigues *et al.*, "Arft: An approximative redundant technique for fault tolerance," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov 2018, pp. 1–6.

[22] E. Linn *et al.*, "Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, 2012.

[23] D. Fujiki *et al.*, "In-Memory Data Parallel Processor," in *Proc. Twenty-Third Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS '18*, vol. 53, no. 2. New York, New York, USA: ACM Press, 2018, pp. 1–14.

[24] S. Hamdioui *et al.*, "Applications of Computation-In-Memory Architectures based on Memristive Devices," in *2019 Des. Autom. Test Eur. Conf. Exhib.* IEEE, mar 2019, pp. 486–491.

[25] M. A. Lebdeh *et al.*, "Memristive Device Based Circuits for Computation-in-Memory Architectures," in *2019 IEEE Int. Symp. Circuits Syst.* IEEE, may 2019, pp. 1–5.

[26] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017, pp. 176–181.

[27] N. Z. Haron *et al.*, "DfT schemes for resistive open defects in RRAMs," in *DATE 2012*. IEEE, mar 2012, pp. 799–804.

[28] C. Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, jan 2015.

[29] I. Yoon *et al.*, "Test challenges in embedded stt-mram arrays," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*, March 2017, pp. 35–38.

[30] X. Pan *et al.*, "Modeling and test for parasitic resistance and capacitance defects in pcm," in *2012 12th Annual Non-Volatile Memory Technology Symposium Proceedings*, Oct 2013, pp. 73–76.

[31] M. Fieback *et al.*, "Testing resistive memories: Where are we and what is missing?" in *2018 IEEE International Test Conference (ITC)*, Oct 2018, pp. 1–9.

[32] L. Wu *et al.*, "Electrical modeling of stt-mram defects," in *2018 IEEE International Test Conference (ITC)*, Oct 2018, pp. 1–10.

[33] S. Hamdioui *et al.*, "Testing Address Decoder Faults in Two-Port Memories: Fault Models, Tests, Consequences of Port Restrictions, and Test Strategy," *Journal of Electronic Testing*, vol. 16, no. 5, pp. 487–498, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1008320716847

[34] A. van de Goor *et al.*, "Detecting faults in the peripheral circuits and an evaluation of SRAM tests," in *International Conference on Test (ITC)*, 2004, pp. 114–123.

[35] S. Yu *et al.*, "Emerging Memory Technologies: Recent Trends and Prospects," *IEEE Solid-State Circuits Mag.*, vol. 8, no. 2, pp. 43–56, 2016.

[36] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semicond. Sci. Technol.*, vol. 31, no. 6, p. 063002, jun 2016.

[37] D. Garbin *et al.*, "Resistive memory variability: A simplified trap-assisted tunneling model," *Solid. State. Electron.*, vol. 115, pp. 126–132, jan 2016.

[38] A. Fantini *et al.*, "Intrinsic switching variability in HfO2 RRAM," in *IMW 2013*. IEEE, may 2013, pp. 30–33.

[39] Y. Fang *et al.*, "Improvement of HfOx -Based RRAM Device Variation by Inserting ALD TiN Buffer Layer," *IEEE Electron Device Lett.*, vol. 39, no. 6, pp. 819–822, jun 2018.

[40] Y. S. Chen *et al.*, "Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity," in *IEDM 2009*. IEEE, dec 2009, pp. 1–4.

[41] C. Wang *et al.*, "Conduction mechanisms, dynamics and stability in ReRAMs," *Microelectron. Eng.*, vol. 187-188, pp. 121–133, feb 2018.

[42] Y. Y. Chen *et al.*, "Improvement of data retention in HfO2/Hf 1T1R RRAM cell under low operating current," in *IEDM 2013*. IEEE, dec 2013, pp. 10.1.1–10.1.4.

[43] E. I. Vatajelu *et al.*, "Fully-connected single-layer stt-mtj-based spiking neural network under process variability," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2017, pp. 21–26.

[44] E. B. Tchernev *et al.*, "Investigating the fault tolerance of neural networks," *Neural Computation*, vol. 17, no. 7, pp. 1646–1664, 2005. [Online]. Available: https://doi.org/10.1162/0899766053723096

[45] S. Kim *et al.*, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1–6.

[46] L. Xia *et al.*, "Fault-tolerant training enabled by on-line fault detection for rram-based neural computing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[47] D. Querlioz *et al.*, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, May 2013.

[48] E. I. Vatajelu *et al.*, "Challenges and solutions in emerging memory testing," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[49] L. Anghel *et al.*, "Neuromorphic computing - from robust hardware architectures to testing strategies," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2018, pp. 176–179.

[50] E. I. Vatajelu *et al.*, "Special session: Reliability of hardware-implemented spiking neural networks (snn)," in *EEE VLSI Test Symposium (VTS)*, 2019.

[51] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

Bibliography

[1] Efficient power management in the 90-nanometer foundry reference flow. URL http://www.chipdesignmag.com/print.php?articleId=76?issueId=6.

[2] Technology and cost trends at advanced nodes. URL https://www.icknowledge.com/news/Technology%20and%20Cost%20Trends%20a%20Advanced%20Node%20-%20Revised.pdf.

[3] An introduction to reducing dynamic power. URL https://semiengineering.com/an-introduction-to-reducing-dynamic-power/.

[4] Shifting bathtub curve for reliability. URL https://ocw.tudelft.nl/wp-content/uploads/Module_14_Reliability.pdf.

[5] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy. X-sram: Enabling in-memory boolean computations in cmos static random access memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12):4219–4232, Dec 2018. doi: 10.1109/TCSI.2018.2848999.

[6] Joao Azevedo, Arnaud Virazel, Alberto Bosio, Luigi Dilillo, Patrick Girard, Aida Todri-Sanial, Jeremy Alvarez-Herault, and Ken Mackay. A Complete Resistive-Open Defect Analysis for Thermally Assisted Switching MRAMs. *IEEE TVLSIS*, 22(11):2326–2335, November 2014. ISSN 1063-8210. doi: 10.1109/TVLSI.2013.2294080. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6701223.

[7] K Beckmann, J Holt, W Olin-Ammentorp, Z Alamgir, J Van Nostrand, and NC Cady. The effect of reactive ion etch (rie) process conditions on reram device performance. *Semiconductor Science and Technology*, 32(9):095013, 2017.

[8] Rajendra Kumar Bishnoi. *Reliable Low-Power High Performance Spintronic Memories*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2017.

[9] Julien Borghetti, Gregory S Snider, Philip J Kuekes, J Joshua Yang, Duncan R Stewart, and R Stanley Williams. 'memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464(7290):873, 2010.

[10] G Cardoso Medeiros, M Taouil, MCR Fieback, LM Bolzani Poehls, and S Hamdioui. Dft scheme for hard-to-detect faults in finfet srams. 2019.

[11] H. H. Chen. Beyond structural test, the rising need for system-level test. In *VLSI-DAT*, pages 1–4, April 2018. doi: 10.1109/VLSI-DAT.2018.8373238.

[12] Y. Chen. Reram: History, status, and future. *IEEE Transactions on Electron Devices*, pages 1–14, 2020. ISSN 1557-9646. doi: 10.1109/TED.2019.2961505.

[13] Y. Chen and J. Li. Fault modeling and testing of 1t1r memristor memories. In *2015 IEEE 33rd VLSI Test Symposium (VTS)*, pages 1–6, April 2015. doi: 10.1109/VTS.2015.7116247.

[14] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, Sep. 1971. doi: 10.1109/TCT.1971.1083337.

[15] A. S. Emara, A. H. Madian, H. H. Amer, S. H. Amer, and M. B. Abdelhalim. Testing of memristor ratioed logic (mrl) xor gate. In *2016 28th International Conference on Microelectronics (ICM)*, pages 181–184, Dec 2016. doi: 10.1109/ICM.2016.7847939.

[16] M. Fieback, M. Taouil, and S. Hamdioui. Testing resistive memories: Where are we and what is missing? In *2018 IEEE International Test Conference (ITC)*, pages 1–9, Oct 2018. doi: 10.1109/TEST.2018.8624895.

[17] M.C.R. Fieback and Said Hamdioui. Device-aware testing: A new test approach towards dppb. 11 2019.

[18] Samuel H Fuller and Lynette I Millett. *The Future of Computing Performance: Game Over or Next Level?* National Academy Press, 2011.

[19] Pierre Emmanuel Gaillardon, Luca Amarú, Anne Siemon, Eike Linn, Rainer Waser, Anupam Chattopadhyay, and Giovanni De Micheli. The programmable logic-in-memory (plim) computer. In *Design, Automation & Test in Europe (DATE) Conference*, pages 427–432, 2016.

[20] Ad.J. Goor, Said Hamdioui, and R. Wadsworth. Detecting faults in the peripheral circuits and an evaluation of sram tests. pages 114– 123, 11 2004. ISBN 0-7803-8580-2. doi: 10.1109/TEST.2004.1386943.

[21] Alessandro Grossi, E Nowak, Cristian Zambelli, C Pellissier, S Bernasconi, G Cibrario, K El Hajjam, R Crochemore, JF Nodin, Piero Olivo, et al. Fundamental variability limits of filament-based rram. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 4–7. IEEE, 2016.

[22] Alessandro Grossi, Cristian Zambelli, Piero Olivo, Enrique Miranda, Valeriy Stikanov, Christian Walczyk, and Christian Wenger. Electrical characterization and modeling of pulse-based forming techniques in rram arrays. *Solid-State Electronics*, 115:17 – 25, 2016. ISSN 0038-1101. doi: https://doi.org/10.1016/j.sse.2015.10.003. URL `http://www.sciencedirect.com/science/article/pii/S0038110115002828`.

[23] S Hamdioui and AJ Van De Goor. Address decoder faults and their tests for two-port memories. In *Proceedings. International Workshop on Memory Technology, Design and Testing (Cat. No. 98TB100236)*, pages 97–103. IEEE, 1998.

[24] S. Hamdioui, M. Taouil, H. A. Du Nguyen, A. Haron, L. Xie, and K. Bertels. Memristor: the enabler of computation-in-memory architecture for big-data. In *2015 International Conference on Memristive Systems (MEMRISYS)*, pages 1–3, Nov 2015. doi: 10.1109/MEMRISYS.2015.7378391.

[25] S. Hamdioui, M. Taouil, and N. Z. Haron. Testing open defects in memristor-based memories. *IEEE Transactions on Computers*, 64(1):247–259, Jan 2015. doi: 10.1109/TC.2013.206.

[26] S. Hamdioui, S. Kvatinsky, G. Cauwenberghs, L. Xie, N. Wald, S. Joshi, H. M. Elsayed, H. Corporaal, and K. Bertels. Memristor for computing: Myth or reality? In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 722–731, March 2017. doi: 10.23919/DATE.2017.7927083.

[27] S. Hamdioui, H. A. Du Nguyen, M. Taouil, A. Sebastian, M. L. Gallo, S. Pande, S. Schaafsma, F. Catthoor, S. Das, F. G. Redondo, G. Karunaratne, A. Rahimi, and L. Benini. Applications of computation-in-memory architectures based on memristive devices. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 486–491, March 2019. doi: 10.23919/DATE.2019.8715020.

[28] Said Hamdioui and Ad J Van De Goor. An experimental analysis of spot defects in srams: realistic fault models and tests. In *Proceedings of the Ninth Asian Test Symposium*, pages 131–138. IEEE, 2000.

[29] Said Hamdioui and Ad J Van de Goor. Efficient tests for realistic faults in dual-port srams. *IEEE Transactions on Computers*, 51(5):460–473, 2002.

[30] Said Hamdioui, Zaid Al-Ars, and Ad J Van de Goor. Opens and delay faults in cmos ram address decoders. *IEEE Transactions on Computers*, 55(12):1630–1639, 2006.

[31] N. Z. Haron and S. Hamdioui. On defect oriented testing for hybrid cmos/memristor memory. In *2011 Asian Test Symposium*, pages 353–358, Nov 2011. doi: 10.1109/ATS.2011.66.

[32] N. Z. Haron and S. Hamdioui. DfT schemes for resistive open defects in RRAMs. In *DATE*, pages 799–804, March 2012. ISBN 978-1-4577-2145-8. doi: 10.1109/DATE.2012.6176603. URL `http://ieeexplore.ieee.org/document/6176603/`.

[33] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[34] Bernd Hoefflinger. *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.

[35] Mohsen Imani, Yeseong Kim, and Tajana Rosing. Mpim: Multi-purpose in-memory processing using configurable resistive memory. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 757–763, 2017.

[36] Handel Jones. Whitepaper: Semiconductor industry from 2015 to 2025. *International Business Strategies*, 2015.

[37] Sachhidh Kannan, Jeyavijayan Rajendran, Ramesh Karri, and Ozgur Sinanoglu. Sneak-path testing of crossbar-based nonvolatile random access memories. *IEEE Trans. Nanotechnol.*, 12 (3):413–426, May 2013. ISSN 1536-125X. doi: 10.1109/TNANO.2013.2253329. URL `http://dx.doi.org/10.1109/TNANO.2013.2253329`.

[38] G. S. Kar, A. Fantini, Y. Chen, V. Paraschiv, B. Govoreanu, H. Hody, N. Jossart, H. Tielens, S. Brus, O. Richard, T. Vandeweyer, D. J. Wouters, L. Altimime, and M. Jurczak. Process-improved rram cell performance and reliability and paving the way for manufacturability and scalability for high density memory application. In *2012 Symposium on VLSI Technology (VLSIT)*, pages 157–158, June 2012. doi: 10.1109/VLSIT.2012.6242509.

[39] Werner Kern. 1 - overview and evolution of silicon wafer cleaning technology. In Karen A. Reinhardt and Werner Kern, editors, *Handbook of Silicon Wafer Cleaning Technology (Second Edition)*, pages 3 – 92. William Andrew Publishing, Norwich, NY, second edition edition, 2008. ISBN 978-0-8155-1554-8. doi: https://doi.org/10.1016/B978-081551554-8.50004-5. URL `http://www.sciencedirect.com/science/article/pii/B9780815515548500045`.

[40] Charles R Kime and M Morris Mano. *Logic and computer design fundamentals*. Prentice Hall, 2003.

[41] Aris-Kyriakos Koliopoulos, Paraskevas Yiapanis, Firat Tekiner, Goran Nenadic, and John Keane. Towards automatic memory tuning for in-memory big data analytics in clusters. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 353–356. IEEE, 2016.

[42] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai. Process technology variation. *IEEE Transactions on Electron Devices*, 58(8): 2197–2208, Aug 2011. doi: 10.1109/TED.2011.2121913.

[43] Kelin Kuhn, Chris Kenyon, Avner Kornfeld, Mark Liu, Atul Maheshwari, Wei-kai Shih, Sam Sivakumar, Greg Taylor, Peter VanDerVoorn, and Keith Zawadzki. Managing process variation in intel's 45nm cmos technology. *Intel Technology Journal*, 12(2), 2008.

[44] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman. Mrl — memristor ratioed logic. In *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, pages 1–6, Aug 2012. doi: 10.1109/CNNA.2012.6331426.

[45] Shahar Kvatinsky, Dmitry Belousov, Slavik Liman, Guy Satat, Nimrod Wald, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. Magic - memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, 2014.

[46] Muath Abu Lebdeh, Uljana Reinsalud, Hoang Anh Du Nguyen, Stephan Wong, and Said Hamdioui. Memristive Device Based Circuits for Computation-in-Memory Architectures. In *ISCAS*, pages 1–5, May 2019. ISBN 978-1-7281-0397-6. doi: 10.1109/ISCAS.2019.8702542. URL `https://ieeexplore.ieee.org/document/8702542/`.

[47] H. Y. Lee, P. S. Chen, T. Y. Wu, Y. S. Chen, C. C. Wang, P. J. Tzeng, C. H. Lin, F. Chen, C. H. Lien, and M. . Tsai. Low power and high speed bipolar switching with a thin reactive ti buffer layer in robust hfo2 based rram. In *2008 IEEE International Electron Devices Meeting*, pages 1–4, Dec 2008. doi: 10.1109/IEDM.2008.4796677.

[48] Won Jun Lee, Chang Hyun Kim, Yoonah Paik, Jongsun Park, Il Park, and Seon Wook Kim. Design of processing-"inside"-memory optimized for dram behaviors. *IEEE Access*, 7:82633–82648, 2019.

[49] Haitong Li, Peng Huang, Bin Gao, Bing Chen, Xiaoyan Liu, and Jinfeng Kang. A spice model of resistive random access memory for large-scale memory array simulation. *IEEE Electron Device Letters*, 35(2):211–213, 2013.

[50] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[51] P. Liu, Z. You, J. Kuang, Z. Hu, H. Duan, and W. Wang. Efficient march test algorithm for 1t1r cross-bar with complete fault coverage. *Electronics Letters*, 52(18):1520–1522, 2016. doi: 10.1049/el.2016.1693.

[52] P. Lorenzi, R. Rao, and F. Irrera. Forming kinetics in HfO_2 -based rram cells. *IEEE Transactions on Electron Devices*, 60(1):438–443, Jan 2013. doi: 10.1109/TED.2012.2227324.

[53] Y. Luo, X. Cui, M. Luo, and Q. Lin. A high fault coverage march test for 1t1r memristor array. In *2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 1–2, Oct 2017. doi: 10.1109/EDSSC.2017.8126415.

[54] Vishwani D. Agarwal Micheal L. Bushnell. *Essentials of Electronic Testing*. Kluwer Academic Publishers, 2000.

[55] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[56] S. M. Nair, R. Bishnoi, M. B. Tahoori, H. Grigoryan, and G. Tshagharyan. Variation-aware fault modeling and test generation for stt-mram. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 80–83, July 2019. doi: 10.1109/IOLTS.2019.8854376.

[57] Janice H. Nickel, John Paul Strachan, Matthew D. Pickett, C. Tom Schamp, J. Joshua Yang, John A. Graham, and R. Stanley Williams. Memristor structures for high scalability: Non-linear and symmetric devices utilizing fabrication friendly materials and processes. *Microelectronic Engineering*, 103:66 – 69, 2013. ISSN 0167-9317. doi: https://doi.org/10.1016/j.mee.2012.09.007. URL http://www.sciencedirect.com/science/article/pii/S0167931712005114.

[58] Stanford R Ovshinsky and Boil Pashmakov. Innovation providing new multiple functions in phase-change materials to achieve cognitive computing. *MRS Online Proceedings Library Archive*, 803, 2003.

[59] Debashis Panda and Tseung-Yuen Tseng. Perovskite oxides as resistive switching memories: A review. *Ferroelectrics*, 471, 11 2014. doi: 10.1080/00150193.2014.922389.

[60] Debashis Panda, Chun-Yang Huang, and Tseung-Yuen Tseng. Resistive switching characteristics of nickel silicide layer embedded hfo. *Applied Physics Letters*, 100, 03 2012. doi: 10.1063/1.3694045.

[61] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent ram. *IEEE micro*, 17 (2):34–44, 1997.

[62] David A. Patterson. Future of Computer Architecture. In *BEARS*, 2006.

[63] Jan M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-178609-1.

[64] Nagarajan Raghavan. Performance and reliability trade-offs for high-k rram. *Microelectronics Reliability*, 54(9):2253 – 2257, 2014. ISSN 0026-2714. doi: https://doi.org/10.1016/j.microrel.2014.07.135. URL http://www.sciencedirect.com/science/article/pii/S0026271414003370. SI: ESREF 2014.

[65] G Snider. Computing with hysteretic resistor crossbars. *Applied Physics A*, 80(6):1165–1172, 2005.

[66] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008. ISSN 1476-4687. doi: 10.1038/nature06932. URL https://doi.org/10.1038/nature06932.

[67] Suk and Reddy. A march test for functional faults in semiconductor random access memories. *IEEE Transactions on Computers*, C-30(12):982–985, Dec 1981. doi: 10.1109/TC.1981.1675739.

[68] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's journal*, 30(3):202–210, 2005.

[69] T. Tsai, J. Li, C. Hsu, and C. Sun. Testing of in-memory-computing 8t srams. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4, Oct 2019. doi: 10.1109/DFT.2019.8875487.

[70] Tsai-Ling Tsai, Jin-Fu Li, Chun-Lung Hsu, and Chi-Tien Sun. Testing of in-memory-computing 8t srams. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4. IEEE, 2019.

[71] Ad J Van de Goor and AJ Van De Goor. *Testing semiconductor memories: theory and practice*, volume 225. J. Wiley & Sons, 1991.

[72] Elena Ioana Vatajelu, Peyman Pouyan, and Said Hamdioui. State of the art and challenges for test and reliability of emerging nonvolatile resistive memories. *International Journal of Circuit Theory and Applications*, 46(1):4–28, 2 2018. ISSN 1097-007X. doi: 10.1002/cta.2418. URL `https://doi.org/10.1002/cta.2418`.

[73] Alvaro Velasquez and Sumit Kumar Jha. Parallel boolean matrix multiplication in linear time using rectifying memristors. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1874–1877, 2016.

[74] Alvaro Velasquez and S. K. Jha. Computation of boolean matrix chain products in 3d reram. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[75] E Vianello, O Thomas, G Molas, O Turkyilmaz, N Jovanović, D Garbin, G Palma, M Alayan, C Nguyen, J Coignus, et al. Resistive memories for ultra-low-power embedded computing design. In *2014 IEEE International Electron Devices Meeting*, pages 6–3. IEEE, 2014.

[76] H.-S. Philip Wong, Simone Raoux, Sangbum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98:2201–2227, 2010.

[77] H.-S. Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T. Chen, and Ming-Jinn Tsai. Metal-Oxide RRAM. *Proc. IEEE*, 100 (6):1951–1970, June 2012. ISSN 0018-9219. doi: 10.1109/JPROC.2012.2190369. URL `http://ieeexplore.ieee.org/document/6193402/`.

[78] HSP Wong, C Ahn, J Cao, HY Chen, SW Fong, Z Jiang, C Neumann, S Qin, J Sohn, Y Wu, et al. Stanford memory trends. *tech. report*, 2016.

[79] L. Xie, H. A. D. Nguyen, M. Taouil, S. Hamdioui, and K. Bertels. Boolean logic gate exploration for memristor crossbar. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, April 2016. doi: 10.1109/DTIS.2016.7483889.

[80] Lei Xie, HA Du Nguyen, Jintao Yu, Ali Kaichouhi, Mottaqiallah Taouil, Mohammad AlFailakawi, and Said Hamdioui. Scouting logic: A novel memristor-based logic design for resistive computing. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 176–181, 2017.

[81] T. Yoo, H. Kim, Q. Chen, T. T. Kim, and B. Kim. A logic compatible 4t dual embedded dram array for in-memory computation of deep neural networks. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, July 2019. doi: 10.1109/ISLPED.2019.8824826.

[82] Jintao Yu, Hoang Anh Du Nguyen, Lei Xie, Mottaqiallah Taouil, and Said Hamdioui. Memristive devices for computation-in-memory. In *Design, Automation & Test in Europe (DATE) Conference*, pages 1646–1651, 2018.

[83] S. Yu and P. Chen. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine*, 8(2):43–56, Spring 2016. doi: 10.1109/MSSC.2016.2546199.

[84] K. Zarrineh, A. P. Deo, and R. D. Adams. Defect analysis and realistic fault model extensions for static random access memories. In *Records of the IEEE International Workshop on Memory Technology, Design and Testing*, pages 119–124, Aug 2000. doi: 10.1109/MTDT.2000.868625.

[85] Weisheng Zhao, Mathieu Moreau, Erya Deng, Yue Zhang, Jean-Michel Portal, Jacques-Olivier Klein, Marc Bocquet, Hassen Aziza, Damien Deleruyelle, Christophe Muller, et al. Synchronous non-volatile logic gate design based on resistive switching memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(2):443–454, 2013.

[86] Jian-Gang Zhu. Magnetoresistive random access memory: The path to competitiveness and scalability. *Proceedings of the IEEE*, 96(11):1786–1798, 2008.

[87] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. Memristor-based memory: The sneak paths problem and solutions. *Microelectronics Journal*, 44(2):176 – 183, 2013. ISSN 0026-2692. doi: https://doi.org/10.1016/j.mejo.2012.10.001. URL `http://www.sciencedirect.com/science/article/pii/S0026269212002108`.