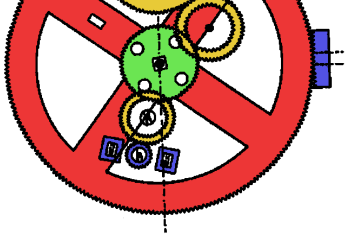

2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

MSc THESIS

Hierarchical Memory Diagnosis Approach

Vishwas Raj Jain

CE-MS-2011-15

Abstract

Semiconductor memories are an inherent part of many modern electronic systems. Due to the fast development of memory process technology and the escalating computing speeds, the on-chip share of memories is rapidly increasing. Additionally, the quality and reliability requirements are becoming more severe especially for critical applications such as automotive and aerospace. Zero defect per million level is now a reality. To satisfy the quality constraints, it is vital to investigate the failure mechanisms. Also, it is required to address the problem of continually decreasing memory yield. Low yield is one of the major threats of the miniaturized electronic systems. Desire for high yield along with the stress to decrease the time to market, has heightened the importance of memory fault diagnosis. The traditional ways of fault diagnosis are not adequate for covering the entire memory fault scope. They suffer from various drawbacks like high complexity, high cost, platform dependence and limited scope. There is a need to introduce changes to the fundamental principles of memory testing and diagnosis approaches.

This thesis presents a novel memory fault diagnosis approach which accurately identifies the faulty memory block and determines the fault type. The proposed approach is platform independent, based on a hierarchical methodology and incorporates several innovative ideas and algorithms. It builds upon the concepts of Test Primitives, Test Classes and Design for Diagnosis. The strength of Hierarchical Memory Diagnosis approach lies in the fact that, unlike conventional approaches, there are no specific implementation requirements other than running a test and determining the pass/fail status of the applied diagnostic test. The scope of the target faults includes all static and dynamic faults occurring in all parts of the memory system. The new approach contributes to the acceleration of characterization of possible defect mechanisms responsible for yield loss in the emerging technologies.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Hierarchical Memory Diagnosis Approach

Deals with defects in all parts of a memory system

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Vishwas Raj Jain
born in Bikaner, India

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Hierarchical Memory Diagnosis Approach

by Vishwas Raj Jain

Abstract

Semiconductor memories are an inherent part of many modern electronic systems. Due to the fast development of memory process technology and the escalating computing speeds, the on-chip share of memories is rapidly increasing. Additionally, the quality and reliability requirements are becoming more severe especially for critical applications such as automotive and aerospace. Zero defect per million level is now a reality. To satisfy the quality constraints, it is vital to investigate the failure mechanisms. Also, it is required to address the problem of continually decreasing memory yield. Low yield is one of the major threats of the miniaturized electronic systems. Desire for high yield along with the stress to decrease the time to market, has heightened the importance of memory fault diagnosis. The traditional ways of fault diagnosis are not adequate for covering the entire memory fault scope. They suffer from various drawbacks like high complexity, high cost, platform dependence and limited scope. There is a need to introduce changes to the fundamental principles of memory testing and diagnosis approaches.

This thesis presents a novel memory fault diagnosis approach which accurately identifies the faulty memory block and determines the fault type. The proposed approach is platform independent, based on a hierarchical methodology and incorporates several innovative ideas and algorithms. It builds upon the concepts of Test Primitives, Test Classes and Design for Diagnosis. The strength of Hierarchical Memory Diagnosis approach lies in the fact that, unlike conventional approaches, there are no specific implementation requirements other than running a test and determining the pass/fail status of the applied diagnostic test. The scope of the target faults includes all static and dynamic faults occurring in all parts of the memory system. The new approach contributes to the acceleration of characterization of possible defect mechanisms responsible for yield loss in the emerging technologies.

Laboratory : Computer Engineering
Codenumber : CE-MS-2011-15

Committee Members :

Advisor:	Dr. ir. Said Hamdioui, CE, TU Delft
Chairperson:	Dr. ir. Koen Bertels, CE, TU Delft
Member:	Prof. Dr. ir. Ad J. van de Goor, CE, TU Delft
Member:	Dr. ir. Zaid Al-Ars, CE, TU Delft
Member:	Dr. ir. Jaap Hoekstra, Electronics, TU Delft

To my parents and to my teachers

Contents

List of Figures

List of Tables

Acknowledgements

It is a pleasure to thank the many people, who made this thesis possible. First and foremost, I wish to thank my supervisor Dr. ir. Said Hamdioui whose guidance, feedback and continued support helped me throughout the project. I would further like to thank Sandra Irobi for her valuable comments on the thesis work and methodology. I wish to give my personal gratitude to Venkataraman Krishnaswami for the time and effort he dedicated to prolonged discussions at every step of the project. The thesis writing benefited greatly from the inputs and suggestions contributed by Seyab Khan and Mottaqiallah Taouil. I would like to mention Halil Kukner for his help and support during the onset of the project. My gratitude also goes to my flatmates and all my friends for their moral support and for making my stay at Delft cheerful and fun. I would like to thank the Computer Engineering department for the technical facilities and the Netherlands government for the financial support without which this thesis could not have been realized.

Many thanks go to my parents for their love, unlimited support and patience along all these years. I would like to mention my brother and sister for their belief in me and never ending support during this long stay away from home. Lastly, I offer my regards to everyone associated with research work in the field of memory testing and diagnosis.

Vishwas Raj Jain
Delft, The Netherlands
September 8, 2011

1 Introduction

Semiconductor memories are gaining importance in every aspect of electronics, starting from simple calculators and extending to super computers. With increasing computing power and decreasing technology dimensions, more and more memory can be accommodated in the same silicon area. Such an exponential increase in density comes at the cost of higher sensitivity to manufacturing defects and process variations, which in turn leads to an increasing number of faults in the memory and thus, decreasing yield. Effective memory diagnosis and failure analysis methodologies are therefore essential to improve and speed up the yield learning. Ensuring fast yield ramp up in turn helps achieving rapid revolution and short time-to-market for new products.

This chapter presents an introduction to memory fault testing, fault diagnosis and motivates the thesis. It also presents the main contributions and outcomes of the thesis. This chapter is organized as follows. Section 1.1 describes the classification and trend for semiconductor memories. Section 1.2 discusses the importance of memory testing and diagnosis. Section 1.3 presents the challenges of memory diagnosis and the specific contributions of this thesis. Section 1.4 outlines the contents of this thesis.

1.1 Semiconductor Memories

In a layman's language, memory is a simple device used to store and retrieve information. It contains inputs for writing data into the memory and outputs for reading data from the memory. There are control signals and address lines to facilitate the read/write operations. Semiconductor memories have come a far way ahead of their contemporaries (for example, magnetic memories, drum memories etc.). The reason is the optimum combination of price, performance and area, which puts them at the heart of today's electronics industry. Semiconductor memories are built mainly from transistors and depending on the characteristics, they can be classified into different categories. Figure 1.1 shows a broad level classification of semiconductor memories. They can be divided among Read Only Memories (ROMs) and Random Access Memories (RAMs).

ROMs are pre-programmed devices which produce the same output data at all times. Data stored in ROM cannot be modified, or can be modified only slowly or with difficulty. ROMs can be further classified into Programmable read-only memory (PROM), Erasable programmable read-only memory (EPROM), Electrically erasable programmable read-only memory (EEPROM) and Flash memories. PROM can be written to or programmed via a special device called a PROM programmer. This device uses high voltages to permanently create internal links within the chip. Consequently, a PROM can only be programmed once. EPROM is a type of ROM that can be erased by exposure to strong ultraviolet light, then rewritten with a process that again needs higher than usual voltage applied. Writing EPROMs is a difficult and slow process. EEPROM is based

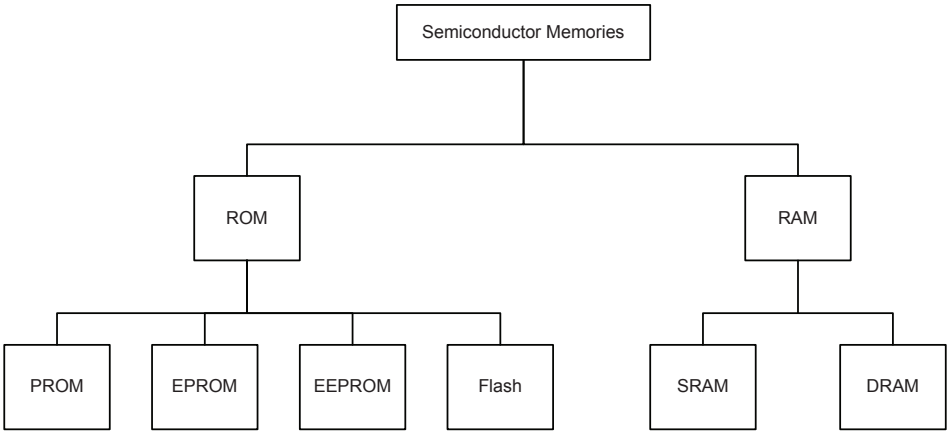

Figure 1.1: Classification of semiconductor memories

on a similar semiconductor structure to EPROM, but it can be electrically erased, then rewritten electrically. Writing an EEPROM is a slow process and takes up to milliseconds per bit. Flash memory is a modern type of EEPROM. Flash memory can be erased and rewritten faster than ordinary EEPROM.

RAMs are memory devices that can be accessed to read or write data without any predetermined order. The two main forms of RAM are static RAM (SRAM) and dynamic RAM (DRAM). SRAM stores a bit of data using cross coupled inverters. SRAMs are expensive to produce, but are generally faster and require less power than DRAM. DRAM stores a bit of data using a transistor and capacitor pair. DRAMs are required to be refreshed periodically and because of this refresh requirement, it is a dynamic memory as opposed to SRAM (static memory). This thesis considers SRAM as the memory device for the presented work.

Nowadays, embedded memories represent the great majority of embedded electronics in Systems on Chip (SoC). It is very common to find SoCs with hundreds of memories representing more than 50% of the overall chips area. According to the ITRS, todays SoCs are moving from logic-dominant to memory-dominant chips in order to deal with application requirements of today and the future. Figure 1.2 shows how the dominant-logic is changing to memory, approaching 94% of the chip area in 2014 [5]. This clearly establishes the importance of semiconductor memory components in our day-to-day life.

Such exponential increase in the on-chip share of memory is due to the ongoing developments in the fabrication process. Silicon area per memory cell is decreasing exponentially. Progressive technology scaling, as tracked by the International Technology Roadmap for Semiconductors (ITRS) and encapsulated by Moores law [38], has driven the phenomenal success of the semiconductor industry. Silicon technology has now entered the nano-era and the 10nm transistors are expected to be in production by 2018. Consequently, embedded memory test and diagnosis challenges will significantly impact the overall testability of SoC. Solving such challenges for memories will substantially contribute to the resolution of electronic system test problems in the future; hence, supporting the continuation of the semiconductor technology revolution and the

manufacturability of future highly complex systems (giga-scale) and highly integrated technologies (nano-scale).

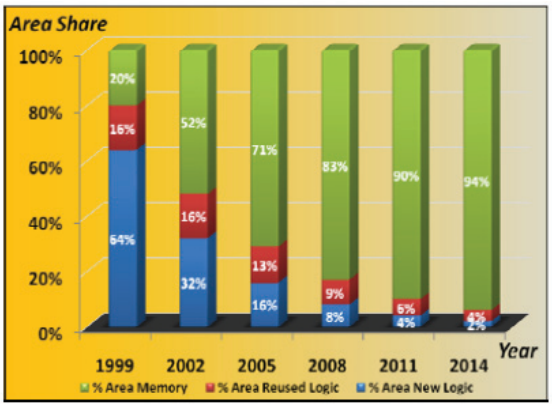

Figure 1.2: Share of embedded memories in systems on chip [5]

1.2 Importance of Memory Testing and Diagnosis

Semiconductor memories are an inherent part of many modern System-on-Chip (SoC) designs. Due to an increase in computing capacity of SoCs, a proportional increase in the amount of data to be processed can be observed, thus creating larger memory requirements for electronic systems. This, along with the shrinking technology dimensions, has motivated the design and development of Ultra-Large Scale Integrated (ULSI) circuits.

The manufacturing of such large circuits is a complicated and time-consuming process and defects in them are inevitable. Defects can be present due to impurities or dislocations in original silicon. These defects manifest in forms of fluctuations, extra/missing transistors, spot defects etc. With decreasing technological dimensions, defects are escalating as devices are more vulnerable to process, voltage and technology (PVT) variations. It is widely recognized that variability in device characteristics and its impact on the overall quality and reliability of the system represents major challenges to scaling and integration for present and future nanotechnology generations [6, 9]. Sensitivity to faults is increasing and faults are becoming more and more complex. Therefore, it is becoming very important to carry out appropriate testing of a memory device in order to guarantee the required product quality.

Testing is a very critical step in the whole design and manufacturing chain; not only because it has to screen out all the defective chips before they are sold, but also because it is the last chance to deliver the required quality and reliability to the end customer. It constitutes a major part of the manufacturing costs of todays products, especially in critical applications such as the automotive, health care, security and aerospace sectors [49]. Developing new fundamentals, which will enable the resolution of the electronic system test challenges is a must in order to sustain the technology growth and the manufacturability of future technology. At the same time, it is required to achieve the quality and reliability standards in an economically viable fashion as any increase in cost

of testing leads to a proportional increase in device cost.

From both economic and technological point of view, it is very important to carry out testing in any memory manufacturing process. All these factors make memory fault testing a topic of vital importance. Memory fault testing is a fast evolving field and a lot of work is going on in this area. The research work in memory testing can be broadly divided into the following challenges:

- Fault models: Defining new fault models to characterize the possible memory defects.
- Design of test algorithms: Developing specific test algorithms for specific memory structures while maintaining low cost, low test complexity and high fault coverage.
- Fault diagnosis: Diagnose faults in an extensible and efficient way in order to realize a fast yield ramp up.

Fault Models

For testing a memory system, the logical behavior of the device under test is compared against the behavior of a good memory. This is done by modeling physical faults as logical faults. Modeling the physical faults as logical fault makes testing process more general and independent of technology and manufacturing process. With the decreasing technology dimensions and the emerging of new failure mechanisms, it is required to establish new fault models. Precise fault modeling is essential for efficient test design.

Design of test algorithms

A test can be defined as a manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect. Test application is to be performed on every manufactured device and are responsible for quality of devices. The cost of testing memories increases rapidly with every new generation of memory chips [15]. Thus, efficient test design is very important to maintain high fault coverage while keeping test cost and time within economically acceptable limits.

Fault Diagnosis

Diagnosis consists in locating the physical faults in the structural model of the device under test and is considered very important for ensuring fast yield ramp up. This thesis targets memory fault diagnosis.

Manufacturing yield is a critical economic parameter in the semiconductor industry. When starting a new process, the yield is generally very low. A rapid yield ramp-up (as well as a high yield maintain in volume production) is a mandatory. Figure 1.3 shows the three phases of the yield learning [52]. In the early phase, yield bring up is mainly based on fab-owned monitor structures. Test chips/ modules are designed to cover most known topological problems, and an extensive test data analysis (local to the fab) is performed. In the intermediate phase, the first products (which may have unexpected low yield) are then introduced. Different topologies are then generated. The root cause

of yield loss in this phase may be topology specific. In the mature phase, the yield is stable and PCMs (Process Module Structures) are used for monitoring. Product test data is then used for yield learning

As the memory cores usually represent a significant portion of the chip area, the memory yield will have a dramatic impact on the overall SoC yield. Today, embedded memories are increasingly identified as having potential for introducing new yield loss mechanisms at a rate, magnitude, and complexity large enough to demand major changes in fault diagnosis schemes [39, 19]. Figure 1.4 shows how the yield decreases dramatically with increasing memory size [20].

With the increasing on-chip share of memory, it is increasing becoming important to take extra measures to ensure high yield and fast yield ramp up. It is not enough to run memory tests designed only to detect memory faults, but it is of utmost importance to to understand the nature of these unknown faults. It is vital to use diagnosis methods to identify the origin of fault and location of defect.

A full diagnosis process requires on-chip and off-chip analysis using several algorithmic sequences. It includes a set of diagnostic test algorithms and a method/tool to analyze the collected diagnostic data. This data is to be further processed to generate a detailed fault report for failure analysis. Memory diagnosis process helps to reduce the ever increasing test cost of newly developed chips. Firstly, it helps in eliminating the faults by allowing required corrections in the process/design. Secondly, using information from the diagnosis, the tests for memory chips can be limited to detect the set of faults occurring in that particular memory architecture, instead of testing for the whole possible memory fault space. All of these significantly contribute to fast yield ramp up and overall cost reduction.

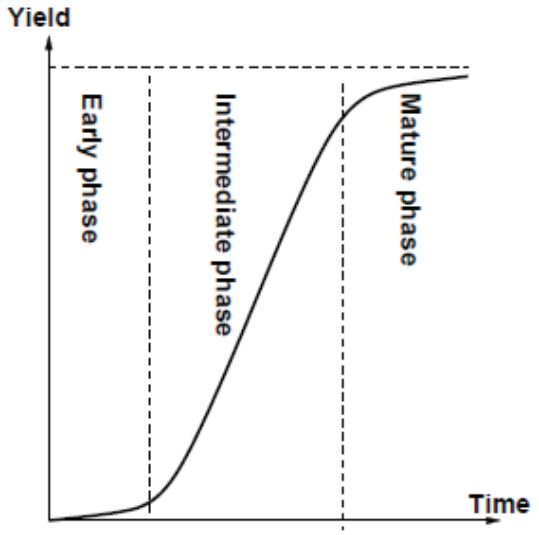

Figure 1.3: Yield vs time curve: Yield learning phases [52]

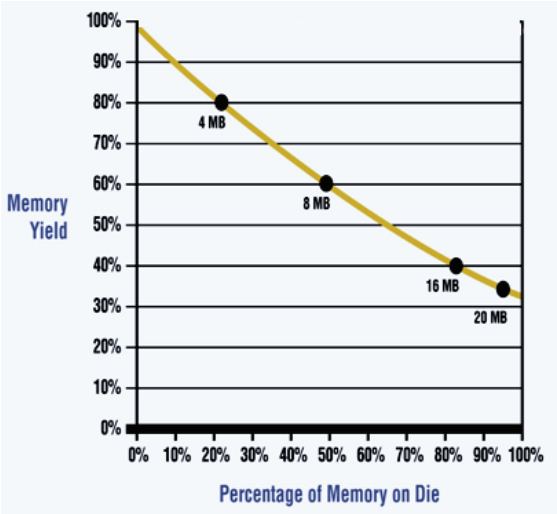

Figure 1.4: Effect of memory size on yield [20]

1.3 Contributions of the Project

The traditional fault diagnosis approaches are not adequate for covering the entire memory fault scope. They suffer from various drawbacks like high complexity, high cost, platform dependence and limited scope. Most of the diagnosis approaches consider only the memory cell array for the diagnosis purposes, while ignoring the possibility of defects in other memory blocks like the address decoder and the peripheral circuitry [12, 28, 45, 58]. So far, no serious attempt has been made to target all the static and dynamic faults in all the memory blocks. Also, nearly all the traditional approaches assume that there is information available on the pass/fail status of each read operation of the applied diagnostic test while this is not practical in the case of plenty of test platforms [2]. Clearly, there is a need to bring in changes to the fundamental principles of memory testing and diagnosis approaches. The challenge is to come up with new diagnosis solutions which are able to identify the fault and provide accurate location of the faulty component while ensuring that all the static and dynamic faults occurring in memory are covered.

This thesis targets the development of a new hierarchical diagnosis approach which is able to diagnose unconventional faults in all parts of the memory system including address decoders and peripheral circuits. The approach is both extendable and platform independent. Traditional methods deal only with conventional faults within only the memory cell array part of the memory system. Following is a summary of the thesis contributions:

- A new diagnosis approach to realize memory fault diagnosis in an hierarchical fashion: Hierarchical Memory Diagnosis.
- The approach targets static and dynamic faults in all memory blocks: the memory cell array, the address decoder and the peripheral circuitry.
- The approach uses the idea of Test classes and Test Primitives as foundation; this

makes the approach platform independent.

- The efficiency and superiority of the approach is demonstrated using defect injection and SPICE simulations.

1.4 Organization of Thesis

This thesis is organized in 8 chapters. Chapter 1 presented a brief introduction to memory fault testing and established the importance of memory fault diagnosis. A brief outline of remaining chapters is provided below.

Chapter 2 explains the basic architecture of SRAM in an hierarchical manner. Starting from the behavioral memory model, at each hierarchical level, memory model is presented and explained with the help of block diagrams. Design of essential memory components like the memory cell array, the address decoder and the peripheral circuitry is explained. Finally, a brief introduction to memory process technology is provided.

Chapter 3 starts with defining the reduced functional model. Further, the concepts of Fault Primitives and Functional Fault Models are described. A classification of the fault space is presented and the behavior of targeted faults is described in detail.

Chapter 4 initiates the subject of memory fault detection and diagnosis. It provides an overview of available memory fault diagnosis algorithms. Shortcomings of the prevalent approaches are discussed and the need for a new diagnosis solution is established.

Chapter 5 introduces the proposed Hierarchical Memory Diagnosis (HMD) approach. The concept of Test Primitives is explained and a new concept of Test Classes is introduced. A number of selected/developed diagnostic test algorithms are presented to diagnose static and dynamic faults in the memory. A complete theoretical proof of the methodology is provided.

Chapter 6 validates the theory developed in the previous chapters. Defects are injected in the memory model and SPICE simulations are done. Simulation results are presented for analysis of static and dynamic faults in order to prove the efficiency and superiority of the proposed diagnosis approach.

Chapter 7 concludes the thesis by presenting the major contributions. Additionally, recommendations are given for further improvements and extension of the project.

2 Memory Architecture

This chapter describes the SRAM architecture in an hierarchical fashion. It starts with a description of models for presenting any electrical system and specifically addresses a memory system. Following a top-down approach, SRAM is discussed at different levels of abstraction starting with the behavioral description and ending with the actual physical implementation.

This chapter is organized as follows. Section 2.1 defines different models used to present the memory architecture. Section 2.2 describes memory as a black-box with only the input and output ports visible to the external world. Section 2.3 further details the black-box model and present memory as a system of several independent subsystems. Section 2.4 discusses the implementation and working of these subsystems. Finally a brief introduction to SRAM cell layout is provided in Section 2.5.

2.1 Memory Models

This section introduces the concept of modeling. A model presents physical phenomena and processes in a logical and objective way while maintaining a level of abstraction. Figure 2.1 presents different levels of abstraction for a memory model [54]. These models are developed to simplify the process of understanding a system by explicitly presenting only required information at that particular level, while hiding irrelevant details. As we move up in the hierarchy, the abstraction level increases and the model defines the way the system is expected to behave while not going in details of physical implementation. The higher the model is in hierarchy, the farther it is from physical representation and closer to the way the system behaves. A fault at a higher level cannot be necessarily mapped to a particular physical failure. Similarly, the low level information like the physical implementation of gates/transistors is not visible at higher levels. The modeling levels presented in Figure 2.1 are further explained below.

Behavioral Model

This is the highest level of abstraction in the hierarchy and defines the functional behavior of the system. It defines a system as a box consisting of several input and output pins. The only available description is the relation between input and output signals with no details about the internal memory structure. As the implementation details of the system are not visible with only the functional behavior specified, such a model is often referred to as black-box model. Further details about the behavioral model are presented in Section 2.2.

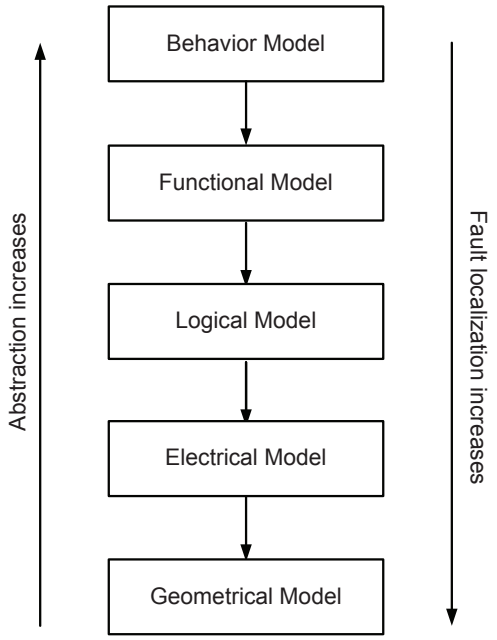

Figure 2.1: Memory models and abstraction levels

Functional Model

Functional model is similar to the behavioral model in describing the function specifications but it also provides some details about the internal structure of a system. This model divides the system into several interacting subsystems and defines specific function of each of the subsystems. In a broader sense, functional model can be defined as a collection of behavioral models of related subsystems of a main system. Behavioral model can be considered a special case of functional model, where only one function is presented, the system itself. Each of the subsystems acts as a functional block with an independent behavioral model of its own. Functional model is further described in detail in Section 2.3.

Logical Model

Logical model represents the system at the gate level. It is useful to describe the system's functionality in terms of boolean relations and logic equations. But in the case of memory, it is not a common practice to use logical models as memory is represented using transistors instead of gates. Therefore, logical model will not be considered anywhere further.

Electrical Model

An electrical model is an electrical equivalent circuit that represents the behavior of a system and also contains details about the internal structure at electrical level. This model describes the electrical components constituting the system. Since this research is concerned with memory testing and diagnosis, it is important to understand the detailed electrical structure of a memory. Electrical model is discussed in more detail in Section 2.4.

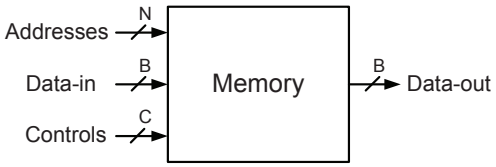

Figure 2.2: Black-box model of SRAM

Geometrical Model

Geometrical model is the lowest level in hierarchy and assumes complete knowledge of the system layout. Thus, this model is also referred to as the layout model. It includes detailed description of physical implementation and considers factors like device geometries, line widths, distances between different components, etc. This model is of importance for manufacturing processes. It is not studied much in literature due to confidentiality reasons and so, only a brief introduction to fabrication process for memory is provided in Section 2.5.

2.2 Memory as a Black-box

The most general model of a system is the black-box model which describes just the input and output pins. This section describes such a black-box model for memory. It is worth noting that this model makes no assumption about the internal structure of memory and contains the least information required to describe the functioning of memory. As shown in Figure 2.2, the input pins provide memory device with control signals, address and input data values. The output produced by memory is provided to external world through data-out pins. Input values consist of C control signals, N address lines and B input data values where B is the word width of memory. It is possible to combine data-in and data-out lines to reduce the total number of pins. To carry out a basic read/write operation, following signals are required:

- Control signals to indicate a read/write operation and to activate the start of an operation.
- Address indicating where to read the data from or where the data is to be written.
- Data input values in case of write operation.

Further details can be added to the black-box model presented in Figure 2.2 to produce a slightly more detailed version. Figure 2.3 divides the black-box model in memory core and input/output ports to produce the so-called two-dimensional memory model [17, 46]. The memory cell array consists of a large number of memory cells. Every cell is capable of storing only one bit of data. Memory ports act as an interface between memory core and the external world.

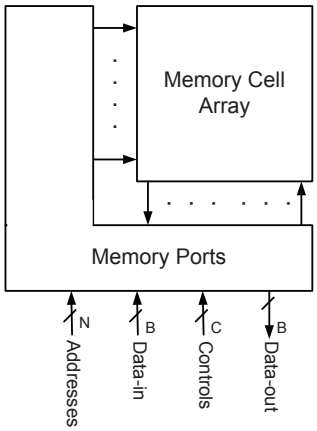

Figure 2.3: Two-dimensional black-box model of SRAM

2.3 Functional Memory Model

As discussed above, a functional model consists of a collection of behavioral models of related subsystems of a main system. Figure 2.4 shows a block diagram of SRAM functional model. As shown in the figure, SRAM can be divided in 4 major subsystems: (1) Memory cell array, (2) Address Decoders, (3) Read/Write Circuitry, (4) Control circuits. A brief description of these subsystems follows here and further details are presented in Section 2.4.

The memory cell array is the heart of the SRAM. It consists of n cells which are organized in an array structure. The capacity of the memory is RxC where R is the number of rows and C is the number of columns. It is important to note the difference in external and internal organization of the memory cell array. A memory chip of 1 Kbit can be logically seen as 1K addresses with a word size of 1 bit while physically it might be organized as an matrix of 100 rows x 10 columns. The word width of the memory also plays a role in deciding the physical organization of the memory cell array. While the number of rows can be any integer, the number of columns should be an integer multiple of the word width.

Address decoder consists of the row decoder and the column decoder. As the memory cell array is organized as a matrix, the memory cell address is divided into row address bits and column address bits. The high order bits select appropriate row and the low order bits select appropriate column. Both of them combined point to a unique cell of the memory cell array. It is possible to select more than one column at a time when the word width $B > 1$. In that case, B cells are accessed together at a given time.

Read/write circuitry can be further divided in sense amplifiers, write drivers and precharge circuits. During a read operation, the content of the selected memory cells is read and amplified by sense amplifiers, loaded into the data register and further presented on the data-out pins. During a write operation, the data present on data-in pins is loaded into data registers and written into the selected memory cells using write drivers. As mentioned before, it is possible to combine data-in and data-out lines to

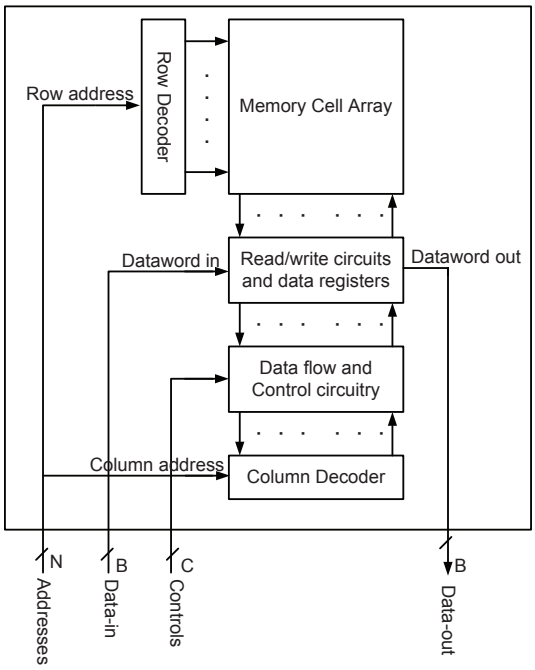

Figure 2.4: Functional model of SRAM

form bidirectional data lines.

Control circuits are responsible for the co-ordination of the operations by means of control signals like read enable, write enable, sense amplifier enable, etc.

2.4 Electrical Memory Model

Electrical model of the SRAM is closely related to this work and thus is described in detail as compared to other models. The subsystems presented in Section 2.3 will be taken one by one and closely examined to learn the internal structure of the memory and how things actually happen in the memory. This will be done for the memory cell array, the address decoder and the read/write circuitry. This is important to understand the faults that can take place and to devise methods capable of detecting them.

2.4.1 Memory Cells

Memory cell for the SRAM is a bistable circuit, which can be driven into one of the two stable states (‘1’ referred to as true or ‘0’ referred to as false). After removing the driving stimulus, the circuit retains its state. Depending on the application, the design of a memory cell is affected by various factors. Thus, different designs are available for memory cells. Figure 2.5 shows the general configuration and three other possible configurations for a memory cell [17].

Memory cell consists of two load elements, two storage elements and two pass transistors. This can be seen in the general configuration of the memory cell presented in

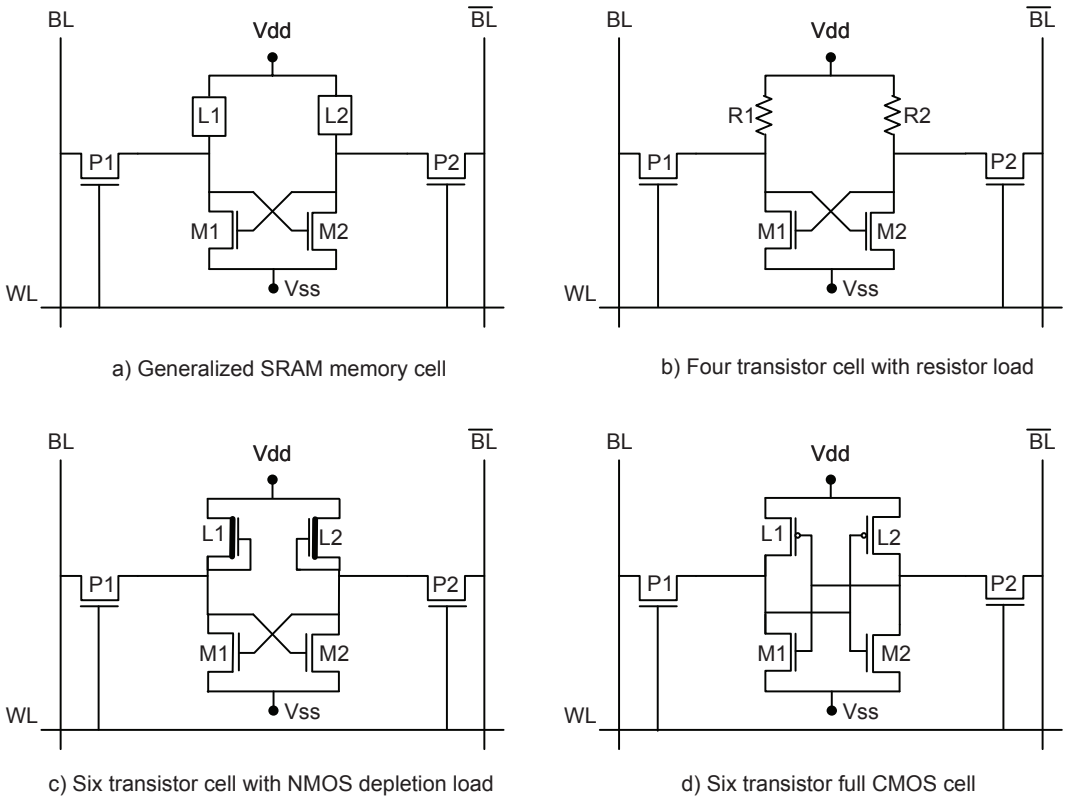

Figure 2.5: Different configurations for SRAM cells [17]

Figure 2.5(a). The pass transistors (P1 and P2) are used to isolate the cell from external circuitry when the cell is not being accessed. The load elements (L1 and L2) along with the storage elements (M1 and M2) form two cross-coupled inverters, thus forming a latch to store a single data bit. Complementary bit-lines are connected to the source/drain of pass transistors and are used to read/write the memory cell. The word line is connected to the gate of the pass transistors and is used to activate access to a particular memory cell. Thus, the memory cell can be accessed via the *Word Line (WL)* and *Bit-Lines (BL and $\overline{BL}$)*. During a write operation, BL and $\overline{BL}$ are driven to complementary data values. When the WL is turned high, the cell is forced to the state presented on BL and $\overline{BL}$ as these lines are driven with more force than the force with which the cell retains its information. To read the data from a memory cell, first the bit-lines BL and $\overline{BL}$ are precharged to high level (Vdd or '1'). When the WL is turned high, depending on the value stored, the memory cell starts pulling down one of the bit-lines. The voltage difference between the complementary bit-lines is sensed and amplified by the read circuit and appropriate value is loaded in the data register. It is to be understood that read process for the SRAM cell is a non-destructive process. The cell retains its data after the read operation.

Depending on the application, the load elements of the generalized configuration can be replaced by polysilicon resistors, depletion mode NMOS transistors, or PMOS transistors. Figure 2.5(b) shows SRAM cell with polysilicon resistors being used as load

element. This design choice reduces the silicon area when compared to other configurations. The disadvantage of this approach is increase in dissipated power due to small but continuously flowing current through resistors. Another choice is shown in Figure 2.5(c) which uses depletion mode NMOS transistors as load element. Compared to previous design with polysilicon resistors, this design occupies a higher silicon area but with decreased power consumption. It is also possible to use enhancement mode NMOS transistors but depletion mode transistors are preferred due to better switching performance, higher impedance and relatively less sensitivity towards power supply variations [17, 48]. The most commonly used and prevalent design choice uses PMOS transistors as load elements (Figure 2.5)(d). This design choice enables zero static power consumption (except the leakage current) as either the NMOS or the PMOS transistor is always *off*. Dynamic power is consumed during the switching activity. The disadvantage of this choice comes during fabrication as due to the presence of both the NMOS and the PMOS transistors, the process steps are more complex and costly than other design configurations.

2.4.2 Address Decoder

Address decoder is required to access a particular cell(s) out of a number of cells present in the memory cell array. As discussed before, the memory cell array is arranged as a two-dimensional matrix. This is useful for the address decoder circuitry. If the memory cell array is arranged as a one-dimensional array with only row or column decoder, the size of the decoder and the length of the word line and the bit-lines would be prohibitive. The size of the decoder and area required for the bit-lines is proportional to $\sqrt{n}$ for two-dimensional addressing as opposed to n for one dimensional addressing, where n is the number of bits in memory chip [54]. Thus, two-dimensional addressing scheme is used with a row decoder for selecting word lines and a column decoder for selecting bit-lines.

2.4.2.1 Row Decoders

The row decoder is required to select only one row of the memory cell array by activating a particular word line. Depending upon the implementation, decoders can be divided in the category of static decoders and dynamic or clocked decoders. The inputs of a decoder are formed by address bits A_0 to A_{n-1} or their complements and the output is the word line. When a word line is selected by the row decoder, all the cells on that row have their respective pass transistors in *on* state and the cells get connected to the bit-lines. Depending on the control signals, read/write operation is performed. For selecting a particular address, for example address 39 which is 100111 in binary, the input to gates A_0 to A_5 should be $A_0A_1A_2\overline{A}_3\overline{A}_4A_5$.

Figure 2.6 shows two implementations (CMOS decoder [17] and PMOS-load decoder [50]) for the static row decoder. Both of these have their advantages and disadvantages. The address lines are connected to both the NMOS and the PMOS transistors in CMOS decoder while they are connected only to the NMOS transistors in PMOS-load decoder. This causes the signal load capacitance to be half in case of PMOS-load decoder when compared to load capacitance of CMOS decoder. Also, the area of PMOS-load decoder is less than that of CMOS decoder. But this comes at a price of comparatively more

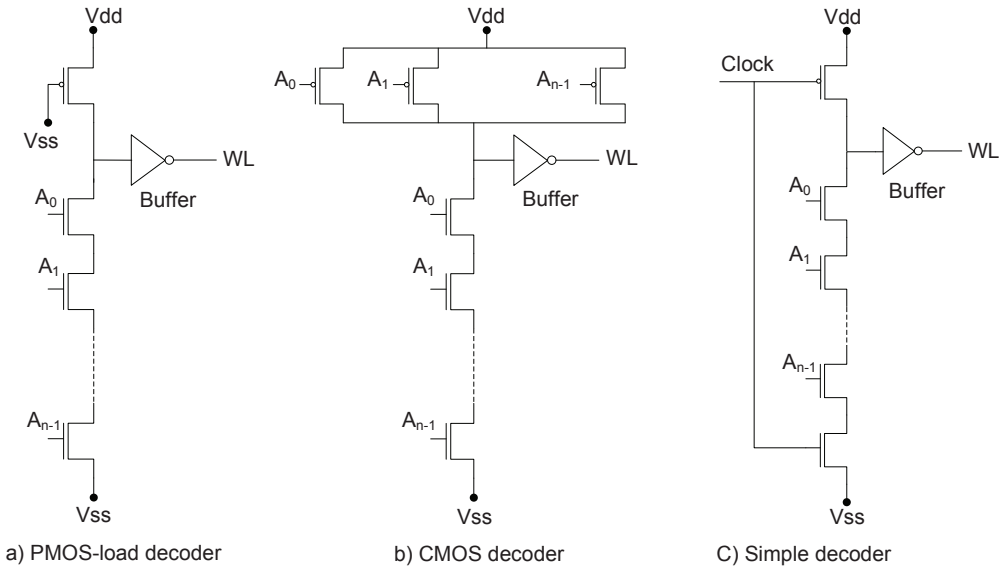

Figure 2.6: Row decoders [17, 50]

static power consumption. The CMOS decoder has an advantage of drawing no static current (other than leakage current).

Figure 2.6 shows an implementation (Simple decoder) for the dynamic row decoder [17]. The advantage of the dynamic decoder is that they combine compact layout with zero static power consumption. Only power consumed is due to the switching activity during the period of address transition. Dynamic decoders are smaller and faster, as no long series of gates is used. The additional cost involved is of the clocking circuit.

2.4.2.2 Column Decoders

The column decoder is responsible for selecting appropriate bit-line pair(s) to read data from or to write data on a memory cell for a successful read/write operation. Depending on the application, different types of column decoder can be used. Figure 2.7(a) shows an example of logarithmic tree decoder [17]. Such a decoder is used for single-ended memory i.e., the memory system which uses only one bit-line for read/write operations. It is a simple but slow decoder as the output has to be routed through $\log_2 n$ levels. Also, in total there are 2*$\log_2 n$ addressing signals. A faster implementation of column decoder is shown in 2.7(b) [37]. It is based on the PMOS-load decoder of Figure 2.6. The output of the decoder goes to an inverter, where the output signal is amplified, after which it enables the selected transistors. This design has the advantage of being compact.

2.4.3 Read/Write Circuitry

The write circuitry is responsible for loading the data input value on the memory cell. Once the appropriate bit-line pair(s) has been selected, write circuitry forces the complementary data values on the selected bit-lines. Figure 2.8 shows two very simple

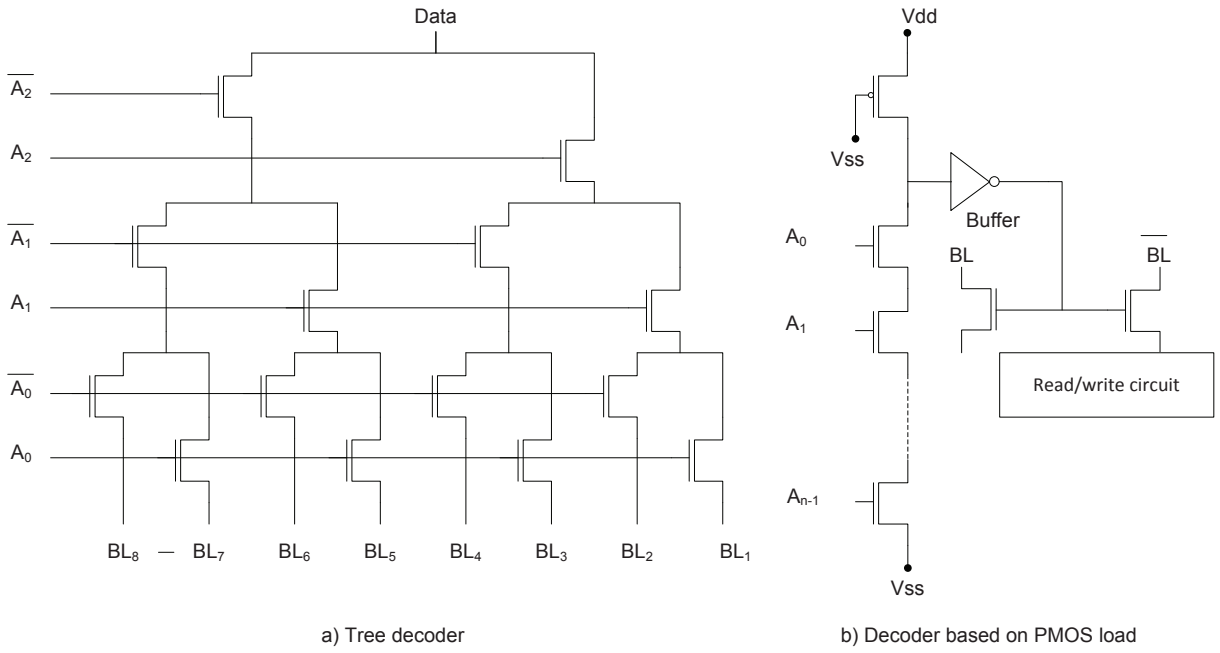

Figure 2.7: Column decoders [17, 37]

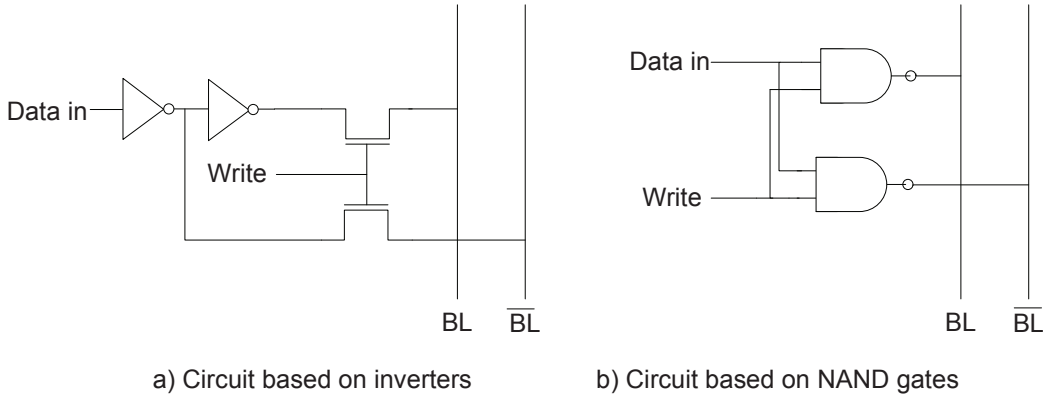

Figure 2.8: Write circuitry [17]

implementations of write circuitry. When *write* control input is high, data to be written is passed from data-in pin to the complementary bit-lines. True bit-line BL contains data-in while the complementary bit-line $\overline{BL}$ contains the complementary of data-in. The data-in value then gets written on the selected memory cell.

The read circuitry is much complex when compared to the write circuitry. The purpose of the read circuitry is to sense the voltage difference between true and complementary bit-line, amplify the difference and load the resulting logic value in the data register. Thus, the read circuitry is often referred to as sense amplifiers. Depending on the implementation of the memory cells, different read circuitry designs are used in practice. It can be single-ended or differential design [17]. Due to fast switching capability to

sense and amplify small voltage differences, differential sense amplifiers are preferred for high performance SRAMs [31]. This capability of differential sense amplifiers is due to their cross-coupled inner structure. Sense amplifiers can also be differentiated based on signal transporting technique: (a) Voltage based sense amplifier, or (b) Current based sense amplifier. Current based sense amplifiers operate faster than the voltage based sense amplifiers [17].

Figure 2.9 shows two different implementations of voltage based sense amplifier: (a) a single-ended PMOS differential sense amplifier [17], and (b) a double-ended PMOS cross-coupled amplifier [17]. The sense amplifiers are activated by making the column switch control signal high. In circuit 2.9(a), when *BL* is high (i.e., the data on the *BL* is '1'), the transistor M1 is *on* and M2 is *off*. The voltage at the gate of transistor Q2 becomes '0' and the transistor is turned *on*. Hence, the Out signal becomes high. In circuit 2.9(b), output voltage transitions are accelerated due to the cross coupled structure.

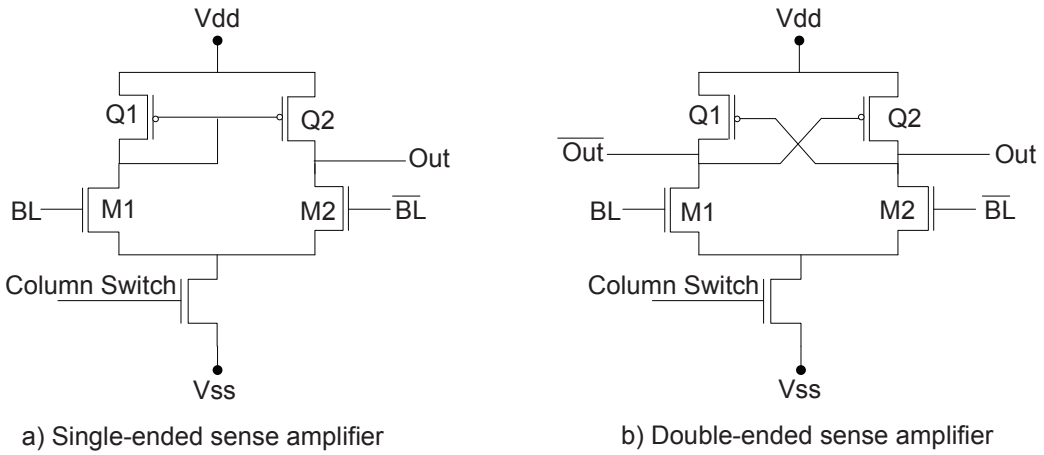

Figure 2.9: Voltage mode sense amplifiers [17]

Two different implementations of current based sense amplifier are available: a double-ended current mirror amplifier, and a hybrid current sense amplifier [11, 17]. The former one is prevalent in use owing to its fast sensing speed, large voltage gain and good output voltage stability [16].

2.5 Memory Process Technology

The most detailed information about memory can be provided using the layout information. It deals with the physical implementation of the memory and thus, is the lowest level in the hierarchy of describing memory architecture. A large number of steps are involved in putting a design on silicon, i.e., going from the electrical model to the layout model. A number of constraints like area, power and cost influence the layout design and thus, make the process exhaustive and labor-intensive. Layout level details for memory are rarely published in the literature due to the confidentiality and sensitivity of this information for semiconductor companies. After the layout is complete and simulations

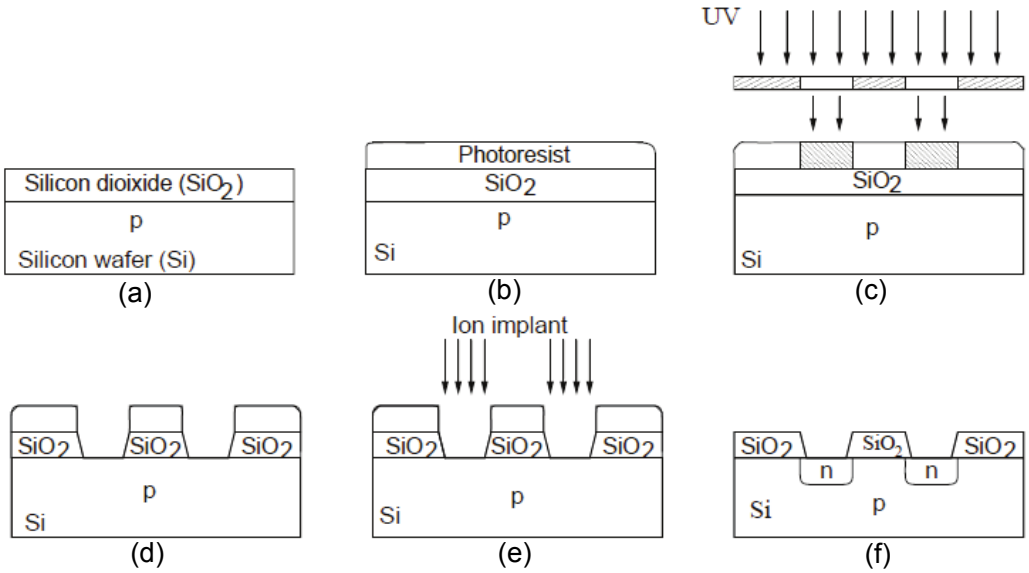

Figure 2.10: Basic steps in forming a MOS transistor [17]

are performed, the process to physically manufacture the memory takes place. A brief description of the MOS technology based memory fabrication process follows.

The basic raw material used for manufacturing memory is silicon. It is because of the special electrical properties of silicon which allow to alter the resistivity of silicon as a function of the impurity atoms introduced in silicon crystal. Silicon is not available in its purest form in nature. So the first and foremost step is growth of a single crystal of pure silicon. Once grown, the silicon crystal is sliced to form circular disks called *wafers*. These wafers are usually 20 to 30 cm in diameter and are less than 1 mm thick. All the further process steps take place on the surface of wafer. The memory circuit is constructed on the wafer surface through a number of processes executed in succession. The most important steps for a MOS process are described further.

The process starts with a pure silicon disk or wafer. Depending upon the requirement, the wafer can be doped with p-type or n-type impurity atoms to create positively or negatively doped semiconductor. A p-type semiconductor has holes as the majority carriers while an n-type semiconductor has electrons as its majority carriers. Figure 2.10 illustrates the MOS process on an p-type silicon wafer. The first step, i.e., step (a) is the thermal oxidization of the silicon wafer. It creates a layer of silicon dioxide on the wafer's surface. This oxide layer is used as gate insulator in MOSFET. Then in step (b) oxide is covered with a layer of photoresist, which is sensitive to UV light. In step (c) the actual circuitry is created through use of photomasks and photolithography. On exposing photoresist to UV light, it becomes soluble in certain solutions. A pattern is created on photoresist by exposing certain areas to UV light with help of a patterned mask. The mask will eventually define the source, drain and channel of the MOSFETs. The area exposed to UV light is now soluble and is removed. Then in step (d) the exposed oxide layer is removed using for e.g., chemical etch. Step (e) shows ion implantation process where dopant atoms are introduced in now exposed silicon. This can also be done by

diffusion process. Diffusion is performed in high temperature gas environment while ion implantation is carried out using an ion beam accelerator. The implantation causes the silicon to act as N-type or P-type doped. Finally in step (f) the remaining photoresist is removed. This process can be repeated several times to produce required pattern and configurations on wafer. The last step is to form metal contacts to form interconnect lines and to access the MOS structures from outside.

3 Memory Fault Space

This chapter introduces the reduced functional model of the memory consisting of three main subsystems: the memory cell array, the address decoder and the peripheral circuitry. Without any impact on available information, reduced functional model is simpler and effective with the perspective of memory testing and diagnosis. Further, the memory fault space is defined and the targeted faults in this work are discussed.

This chapter is organized as follows. Section 3.1 defines the reduced functional fault model. Section 3.2 introduces the concept of fault primitives and fault models. Section 3.3 categorizes fault primitives into different fault classes and shows the scope of targeted faults in this work. Section 3.4 and Section 3.5 discusses the targeted fault models in more detail.

3.1 Reduced Memory Functional Model

For memory testing purposes, it is considered sufficient to test the major functional blocks of the memory: the memory cell array, the row decoder, the column decoder, sense amplifiers, write drivers and data registers. In order to simplify the presentation, the functional model presented in Figure 2.4 can be modified, without suffering any loss in the available information. These changes produce the reduced functional fault model as shown in Figure 3.1 [54]. The row decoder and the column decoder, both concerned with addressing the memory cell(s), can be combined in one single block called the address decoder. Read/write circuits, data registers and all other circuitry concerned with the transport of data to and from the memory cell array can be put together as one block called the peripheral circuitry. This reduced functional model reduces the complexity of the testing and analysis process, makes it faster and to the point.

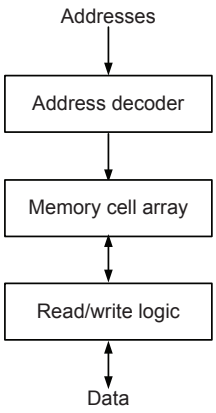

Figure 3.1: Reduced functional model

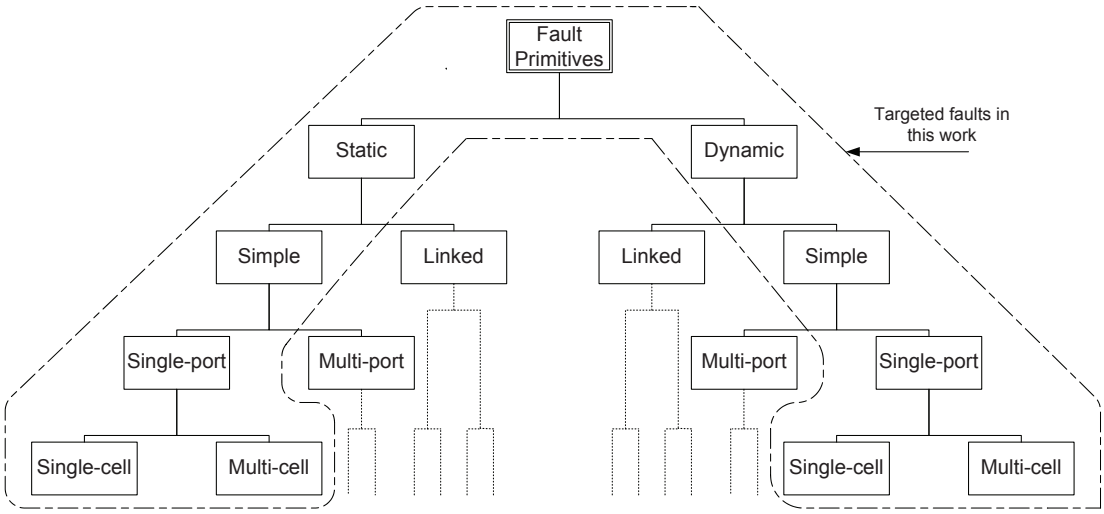

Figure 3.2: Classification of fault primitives

3.2 Fault Primitive: Concept

A functional fault can be detected by applying a number of operation sequences on the memory under test and then observing the output behavior. Functional faults can be precisely defined as deviation from the expected behavior under the influence of performed operations(s). Thus, a function fault model can be defined as [17]:

- A list of performed operations that cause a deviation from specified/expected behavior. As these operations sensitize the fault, they are also called *Sensitizing Operation Sequence* (SOS).
- A list of faulty behavior i.e., corresponding deviations observed under performed SOS.

Recently, a new functional parameter has also been added to the functional fault model. The logical output level of a read operation is relevant to modeling read operation related faults [17]. The combination of these fault parameters, the *Fault Primitive* (FP), is thus annotated as $< S/F/R >$ [55]. Here, S represents the SOS which sensitizes the fault, F represents the value or the behavior in the faulty cell and R represents the logical output of the read operation. FPs can be grouped on the basis of similar or complementary SOS to form *Functional Fault Models* (FFM) [55].

3.3 Fault Primitive: Classification

Figure 3.2 shows four different and independent categories of FPs [17]. They can be classified based on:

- the way the FPs manifest themselves, into simple and linked faults.

- the number of sequential operations required in the SOS, into static and dynamic faults.
- the number of simultaneous operations required in the SOS, into single-port and multi-port faults.
- the number of different cells the FPs do involve, into single-cell and multi-cell faults.

Simple and Linked Faults

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

Simple faults are the faults which have no affect on the behavior of other faults and so, no fault masking can take place. While linked faults are the ones which can influence other faults. The behavior of one fault can affect the behavior of another fault and thus fault masking can happen [17]. Linked faults consist of two or more simple faults.

Static and Dynamic Faults

Depending on the number of sequential operations required in the SOS, faults can be classified into static and dynamic faults.

Let #O be defined as the number of operations performed sequentially. Static faults are the faults that can be sensitized using at most one operation i.e., #O$\leq$1 while dynamic faults are the faults sensitized by performing at least two operations sequentially i.e., #O$\geq$1. With the decreasing dimensions of the process technology, more attention is being paid to two-operation dynamic faults.

Single-port and Multi-port Faults

Depending on the number of simultaneous operations required in the SOS, faults can be classified into single-port and multi-port faults.

Let #P be defined as the number of ports required simultaneously to apply a SOS. Single-port faults (1PFs) require at most one port to be sensitized i.e., #P$\leq$1 while multi-port faults (pPFs) can be sensitized using at least 2 ports i.e., #P$\geq$1. Multi-port faults require two or more operations to be applied simultaneously through different ports. It should be clear that single-port faults can be present in both the single-port and the multi-port memory.

Single-cell and Multi-cell Faults

Depending on the number of different cells the FPs do involve, faults can be classified into single-cell and multi-cell faults.

Let #C be defined as the number of cells accessed during a SOS. Single-cell faults take place in the same cell where the SOS is applied while multi-cell faults involve more than one cell while in the case of multi-cell faults, fault may appear in a different cell

than the one on which the sensitizing sequence is applied. Depending on #C, multi-cell faults can be further categorized in two-cell coupling FPs, three-cell coupling FPs etc.

In this dissertation, we will focus on simple single-port faults, both static and two-operation dynamic faults, including single-cell and two-cell faults, see Figure 3.2. From here on, the term *fault* or *FP* will refer to simple single-port FP. Details about targeted FPs will be discussed in Section 3.4 and Section 3.5.

3.4 Static Faults

Static faults are faults that are timing independent and can be sensitized by performing at most one operation. Referring back to the reduced memory functional model of Figure 3.1, memory static faults can be classified into three classes: (a) memory cell array faults, (b) address decoder faults and (c) peripheral circuitry faults. Each of these classes are discussed next.

3.4.1 Static Memory Cell Array Faults (sMCAFs)

Faults in the memory cell array can be divided among single-cell faults and multi-cell coupling faults. Single-cell FPs cover faults occurring in a single cell while multi-cell FPs involve more than one cell at a time. In addition to single-cell FPs, we will only consider two-cell FPs (coupling faults) as they are demonstrated to be an important class in SRAM faults [17].

Figure 3.3 shows single-cell and two-cell faults in the memory cell array. As can be seen from the figure, the aggressor and the victim cells are same for the single-cell fault; i.e., the fault appears in the same cell on which the sensitizing operation is applied. For the two-cell fault, depending upon the cell on which the sensitizing operation is applied, there can be three cases:

- $1PF2_s$: Rater than any operation, the state of the aggressor cell sensitizes the fault in the victim cell. No operation is required to sensitize the fault. In notation $1PF2_s$, subscript s denotes state.
- $1PF2_a$: A particular operation performed on the aggressor cell sensitizes the fault in the victim cell. In notation $1PF2_a$, subscript a denotes aggressor.
- $1PF2_v$: A particular operation performed on the victim cell sensitizes the fault in the victim cell, given that the aggressor cell is in a particular state. In notation $1PF2_v$, subscript v denotes victim.

Single-cell Faults

Before proceeding to the listing and description of single-cell FPs, it is a good idea to define the FP notation in proper detail. $< S/F/R >$ denotes a FP for a single memory cell. S represents the sensitizing value or operation: $S \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$. Here, 0 (1) denotes the state 0 (1) of the cell, 0w0 (0w1) denotes a non-transition

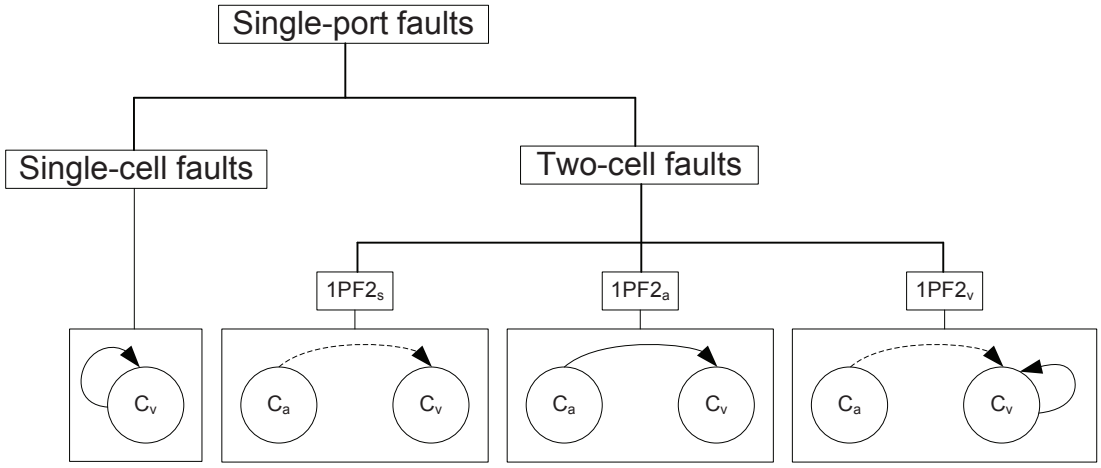

Figure 3.3: Static one-cell and two-cell faults in memory cells

write 0 (1) operation, 0w1 (1w0) denotes a transition write 0→1 (1→0) operation and r0 (r1) denotes a read 0 (1) operation. F is the value/behavior of the faulty cell: $F \in \{0, 1, \uparrow, \downarrow\}$ where '↑' ('↓') denotes an up (down) transition in faulty cell. R is the logical value received on memory output when the sensitizing operation is a read operation: $R \in \{0, 1, -\}$. R contains a value '0' or '1' when the fault is sensitized by a read operation while $R = -$ signifies that the fault is sensitized by a write operation and no output value is applicable. For example, in the FP $< 1w1/0/- >$, which is the write destructive fault, $S = 1w1$ means a $w1$ operation is applied to a cell initialized to '1'. The fault effect $F = 0$ indicates that after performing $w1$ operation, faulty cell state changes to '0'. $R = -$ denotes that no output is expected of S.

Given that $S \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$, $F \in \{0, 1, \uparrow, \downarrow\}$ and $R \in \{0, 1, -\}$; three cases can be distinguished (Table 3.1):

- $S \in \{0, 1\}$
  if $S = 0$, $F = 1$ and $R = -$; this results in FP1 of Table 3.1.
  if $S = 1$, $F = 0$ and $R = -$; this results in FP2.

- $S \in \{0w0, 1w1, 0w1, 1w0\}$
  if $S = 0w0$, $F = \uparrow$ and $R = -$; this results in FP5.
  if $S = 1w1$, $F = \downarrow$ and $R = -$; this results in FP6.
  if $S = 0w1$, $F = 0$ and $R = -$; this results in FP3.
  if $S = 1w0$, $F = 1$ and $R = -$; this results in FP4.

- $S \in \{r0, r1\}$
  if $S = r0$, $F = \uparrow$ and $R = 1$; this results in FP7.
  if $S = r0$, $F = \uparrow$ and $R = 0$; this results in FP9.
  if $S = r0$, $F = 0$ and $R = 1$; this results in FP11.
  if $S = r1$, $F = \downarrow$ and $R = 0$; this results in FP8.
  if $S = r1$, $F = \downarrow$ and $R = 1$; this results in FP10.
  if $S = r1$, $F = 1$ and $R = 1$; this results in FP12.

Table 3.1: Single-cell static FPs

#	S	F	R	$< S/F/R >$	FFM	#	S	F	R	$< S/F/R >$	FFM
1	0	1	-	$< 0/1/- >$	SF	7	$0r0$	$\uparrow$	1	$< 0r0/\uparrow/1 >$	RDF
2	1	0	-	$< 1/0/- >$	SF	8	$1r1$	$\downarrow$	0	$< 1r1/\downarrow/0 >$	RDF
3	$0w1$	0	-	$< 0w1/0/- >$	TF	9	$0r0$	$\uparrow$	0	$< 0r0/\uparrow/0 >$	DRDF
4	$1w0$	1	-	$< 1w0/1/- >$	TF	10	$1r1$	$\downarrow$	1	$< 1r1/\downarrow/1 >$	DRDF
5	$0w0$	$\uparrow$	-	$< 0w0/\uparrow/- >$	WDF	11	$0r0$	0	1	$< 0r0/0/1 >$	IRF
6	$1w1$	$\downarrow$	-	$< 1w1/\downarrow/- >$	WDF	12	$1r1$	1	0	$< 1r1/1/0 >$	IRF

Table 3.2: Single-cell static FFMs

#	FFM	Fault Primitives	#	FFM	Fault Primitives
1	SF	$< 0/1/- >, < 1/0/- >$	4	RDF	$< 0r0/\uparrow/1 >, < 1r1/\downarrow/0 >$
2	TF	$< 0w1/0/- >, < 1w0/1/- >$	5	DRDF	$< 0r0/\uparrow/0 >, < 1r1/\downarrow/1 >$
3	WDF	$< 0w0/\uparrow/- >, < 1w1/\downarrow/- >$	6	IRF	$< 0r0/0/1 >, < 1r1/1/0 >$

As can be seen from Table 3.1, write operations are capable of sensitizing 4 FPs and read operations are capable of sensitizing 6 FPs. In total there are 12 single-cell FPs. The above mentioned FPs can be combined and categorized into FFMs. Table 3.2 summarizes the FFMs together with their respective FPs.

1. **State Fault (SF)**: The logic value stored in a cell flips before accessing the cell. It is a special case as no operation is required to sensitize the fault and the fault depends on the initial stored value in the cell. SF consists of 2 FPs: $< 0/1/- >$ and $< 1/0/- >$.

2. **Transition Fault (TF)**: A transition write operation to the cell fails i.e., the cell is unable to undergo a transition ($0 \rightarrow 1$ or $1 \rightarrow 0$). TF consists of two FPs: $< 0w1/0/- >$ and $< 1w0/1/- >$.

3. **Write Destructive Fault (WDF)**: A non-transition write operation ($0w0$ or $1w1$) reverses the logic value stored in the cell. WDF consists of 2 FPs: $< 0w0/\uparrow/- >$ and $< 1w1/\downarrow/- >$.

4. **Read Destructive Fault (RDF)**: A read operation causes a transition in the cell and returns the new incorrect value on output. RDF consists of 2 FPs: $< 0r0/\uparrow/1 >$ and $< 1r1/\downarrow/0 >$.

5. **Deceptive Read Destructive Fault (DRDF)**: A read operation causes a transition in the cell but returns the old correct value on output. DRDF consists of 2 FPs: $< 0r0/\uparrow/0 >$ and $< 1r1/\downarrow/1 >$.

6. **Incorrect Read Fault (IRF)**: A read operation returns an incorrect value on output while the cell contains the correct logic value. IRF consists of 2 FPs: $< 0r0/0/1 >$ and $< 1r1/1/0 >$.

Two-cell Faults

Two-cell faults involve more than one cell at a time. For two-cell coupling faults, the FPs are described as $< S_a; S_v/F/R >_{a,v}$ where S_a (S_v) is the sensitizing operation applied to aggressor (victim) cell: $S_a, S_v \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$. Here a (v) denotes the address of the aggressor (victim) cell for a coupling fault. It is to be noted that, if S_a is an operation, than S_v can be only a state (0,1). If S_a is a state, S_v can be either a state or an operation. The meaning of F and R has already been defined in section 3.4.1.

Given that $S_a, S_v \in \{0, 1, 0w0, 1w1, 0w1, 1w0, r0, r1\}$, $F \in \{0, 1, \uparrow, \downarrow\}$ and $R \in \{0, 1, -\}$; two cases can be distinguished while maintaining restriction of #O$\leq$1 :

- $S_a \in \{0, 1\}$
  Here, notation $< S_a; S_v/F/R >$ can be divided in two subparts: $< 0; S_v/F/R >_{a,v}$ (12 FPs) and $< 1; S_v/F/R >_{a,v}$ (12 FPs). Based on the state/sensitizing operation of S_v, we can further classify these 24 FPs in two subclasses:
  - $S_v \in \{0, 1\}$: This the special case when the fault is sensitized without performing any operation. The state of the aggressor cell sensitizes a fault in the victim cell i.e., a $1PF2_s$ fault. This results in 4 FPs namely FP1 to FP4 of Table 3.3.
  - $S_v \in \{0w0, 1w1, 0w1, 1w0, r0, r1\}$: A particular operation performed on the victim cell sensitizes a fault in the victim cell, given that the aggressor cell is in a particular state i.e., a $1PF2_v$ fault. This results in 20 FPs namely FP5 to FP24 of Table 3.3.
- $S_a \in \{0w0, 1w1, 0w1, 1w0, r0, r1\}$
  A particular operation performed on the aggressor cell sensitizes the fault in the victim cell i.e., a $1PF2_a$ fault. This results in 12 FPs. Based on the state of S_v, we can further classify these 12 FPs in two subclasses:
  - $< S_a; 0/ \uparrow /- >$: As S_a can be any one of the six SOSs, this notation represents 6 FPs namely FP25 to FP30 of Table 3.3.
  - $< S_a; 1/ \downarrow /- >$: As S_a can be any one of the six SOSs, this notation represents 6 FPs namely FP30 to FP36 of Table 3.3.

As can be seen from Table 3.3, in total there are 36 two-cell coupling FPs. The above mentioned FPs can be combined and categorized into FFMs. Table 3.4 summarizes the classification of FFMs together with their respective FPs. Please note that the FFMs are arranged as $1PF2_s$, $1PF2_v$ and $1PF2_a$ FFMs.

1PF2$_s$ FFMs

1. **State Coupling Fault (CFst)**: A particular state in the aggressor cell sensitizes the fault by flipping the value in the victim cell. It is a special case as no operation is required to sensitize the fault and the fault depends on the initial stored value in the cell. CFst consists of 4 FPs: $< 0; 0/1/- >$, $< 0; 1/0/- >$,$< 1; 0/1/- >$ and $< 1; 1/0/- >$.

Table 3.3: Two-cell static FPs

#	S_a	S_v	F	R	$< S_a; S_v/F/R >$	FFM	#	S_a	S_v	F	R	$< S_a; S_v/F/R >$	FFM
1	0	0	1	-	$< 0; 0/1/- >$	CFst	19	1	$r0$	$\uparrow$	0	$< 1; r0/\uparrow/0 >$	CFdrd
2	0	1	0	-	$< 0; 1/0/- >$	CFst	20	1	$r1$	$\downarrow$	1	$< 1; r1/\downarrow/1 >$	CFdrd
3	1	0	1	-	$< 1; 0/1/- >$	CFst	21	0	$r0$	0	1	$< 0; r0/0/1 >$	CFir
4	1	1	0	-	$< 1; 1/0/- >$	CFst	22	0	$r1$	1	0	$< 0; r1/1/0 >$	CFir
5	0	$0w1$	0	-	$< 0; 0w1/0/- >$	CFtr	23	1	$r0$	0	1	$< 1; r0/0/1 >$	CFir
6	0	$1w0$	1	-	$< 0; 1w0/1/- >$	CFtr	24	1	$r1$	1	0	$< 1; r1/1/0 >$	CFir
7	1	$0w1$	0	-	$< 1; 0w1/0/- >$	CFtr	25	$r0$	0	$\uparrow$	-	$< r0; 0/\uparrow/- >$	CFds_{rx}
8	1	$1w0$	1	-	$< 1; 1w0/1/- >$	CFtr	26	$r1$	0	$\uparrow$	-	$< r1; 0/\uparrow/- >$	CFds_{rx}
9	0	$0w0$	$\uparrow$	-	$< 0; 0w0/\uparrow/- >$	CFwd	27	$0w1$	0	$\uparrow$	-	$< 0w1; 0/\uparrow/- >$	$\text{CFds}_{xw\overline{x}}$
10	0	$1w1$	$\downarrow$	-	$< 0; 1w1/\downarrow/- >$	CFwd	28	$1w0$	0	$\uparrow$	-	$< 1w0; 0/\uparrow/- >$	$\text{CFds}_{xw\overline{x}}$
11	1	$0w0$	$\uparrow$	-	$< 1; 0w0/\uparrow/- >$	CFwd	29	$0w0$	0	$\uparrow$	-	$< 0w0; 0/\uparrow/- >$	CFds_{xwx}
12	1	$1w1$	$\downarrow$	-	$< 1; 1w1/\downarrow/- >$	CFwd	30	$1w1$	0	$\uparrow$	-	$< 1w1; 0/\uparrow/- >$	CFds_{xwx}
13	0	$r0$	$\uparrow$	1	$< 0; r0/\uparrow/1 >$	CFrd	31	$r0$	1	$\downarrow$	-	$< r0; 1/\downarrow/- >$	CFds_{rx}
14	0	$r1$	$\downarrow$	0	$< 0; r1/\downarrow/0 >$	CFrd	32	$r1$	1	$\downarrow$	-	$< r1; 1/\downarrow/- >$	CFds_{rx}
15	1	$r0$	$\uparrow$	1	$< 1; r0/\uparrow/1 >$	CFrd	33	$0w1$	1	$\downarrow$	-	$< 0w1; 1/\downarrow/- >$	$\text{CFds}_{xw\overline{x}}$
16	1	$r1$	$\downarrow$	0	$< 1; r1/\downarrow/0 >$	CFrd	34	$1w0$	1	$\downarrow$	-	$< 1w0; 1/\downarrow/- >$	$\text{CFds}_{xw\overline{x}}$
17	0	$r0$	$\uparrow$	0	$< 0; r0/\uparrow/0 >$	CFdrd	35	$0w0$	1	$\downarrow$	-	$< 0w0; 1/\downarrow/- >$	CFds_{xwx}
18	0	$r1$	$\downarrow$	1	$< 0; r1/\downarrow/1 >$	CFdrd	36	$1w1$	1	$\downarrow$	-	$< 1w1; 1/\downarrow/- >$	CFds_{xwx}

Table 3.4: Two-cell static FFMs

#	FFM	Fault Primitives
1	CFst	$< 0; 0/1/- >$, $< 0; 1/0/- >$,$< 1; 0/1/- >$, $< 1; 1/0/- >$
2	CFtr	$< 0; 0w1/0/- >$, $< 0; 1w0/1/- >$, $< 1; 0w1/0/- >$, $< 1; 1w0/1/- >$
3	CFwd	$< 0; 0w0/\uparrow/- >$, $< 0; 1w1/\downarrow/- >$, $< 1; 0w0/\uparrow/- >$, $< 1; 1w1/\downarrow/- >$
4	CFrd	$< 0; r0/\uparrow/1 >$, $< 0; r1/\downarrow/0 >$, $< 1; r0/\uparrow/1 >$, $< 1; r1/\downarrow/0 >$
5	CFdrd	$< 0; r0/\uparrow/0 >$, $< 0; r1/\downarrow/1 >$, $< 1; r0/\uparrow/0 >$, $< 1; r1/\downarrow/1 >$
6	CFir	$< 0; r0/0/1 >$, $< 0; r1/1/0 >$, $< 1; r0/0/1 >$, $< 1; r1/1/0 >$
7	CFds_{rx}	$< r0; 0/\uparrow/- >$, $< r0; 1/\downarrow/- >$, $< r1; 0/\uparrow/- >$, $< r1; 1/\downarrow/- >$
8	$\text{CFds}_{xw\overline{x}}$	$< 0w1; 0/\uparrow/- >$, $< 0w1; 1/\downarrow/- >$, $< 1w0; 0/\uparrow/- >$, $< 1w0; 1/\downarrow/- >$
9	CFds_{xwx}	$< 0w0; 0/\uparrow/- >$, $< 0w0; 1/\downarrow/- >$, $< 1w1; 0/\uparrow/- >$, $< 1w1; 1/\downarrow/- >$

1PF2$_v$ FFMs

1. **Transition Coupling Fault (CFtr)**: A transition write operation to the victim cell fails, given that the aggressor cell is in a particular state i.e., depending on the state of the aggressor cell, the victim cell is unable to undergo a transition ($0 \rightarrow 1$ or $1 \rightarrow 0$). CFtr consists of 4 FPs: $< 0; 0w1/0/- >$, $< 0; 1w0/1/- >$, $< 1; 0w1/0/- >$ and $< 1; 1w0/1/- >$.

2. **Write Destructive Coupling Fault (CFwd)**: A non-transition write operation (0w0 or 1w1) to the victim cell reverses the logic value stored in the victim cell, given that the aggressor cell is in a particular state. CFwd consists of 4 FPs:

$< 0; 0w0/\uparrow/- >$, $< 0; 1w1/\downarrow/- >$, $< 1; 0w0/\uparrow/- >$ and $< 1; 1w1/\downarrow/- >$.

3. **Read Destructive Coupling Fault (CFrd)**: A read operation in the victim cell causes a transition in the victim cell and returns the new incorrect value on output, given that the aggressor cell is in a particular state. CFrd consists of 4 FPs: $< 0; r0/\uparrow/1 >$, $< 0; r1/\downarrow/0 >$, $< 1; r0/\uparrow/1 >$ and $< 1; r1/\downarrow/0 >$.

4. **Deceptive Read Destructive Coupling Fault (CFdrd)**: A read operation to the victim cell causes a transition in the victim cell but returns the old correct value on output, given that the aggressor cell is in a particular state. CFdrd consists of 4 FPs: $< 0; r0/\uparrow/0 >$, $< 0; r1/\downarrow/1 >$, $< 1; r0/\uparrow/0 >$ and $< 1; r1/\downarrow/1 >$.

5. **Incorrect Read Coupling Fault (CFir)**: A read operation to the victim cell returns an incorrect value on output while the cell contains the correct logic, given that the aggressor cell is in a particular state. CFir consists of 4 FPs: $< 0; r0/0/1 >$, $< 0; r1/1/0 >$, $< 1; r0/0/1 >$ and $< 1; r1/1/0 >$.

1PF2$_a$ FFMs

1. **Disturb Coupling Fault (CFds)**: A particular operation in the aggressor cell sensitizes the fault by flipping the value in the victim cell. Depending on the operation performed on the aggressor cell (read, transition write, non-transition write), CFds can be divided in three sub-parts.
   - CFds$_{rx}$: $< r0; 0/\uparrow/- >$, $< r0; 1/\downarrow/- >$, $< r1; 0/\uparrow/- >$ and $< r1; 1/\downarrow/- >$.
   - CFds$_{xw\overline{x}}$: $< 0w1; 0/\uparrow/- >$, $< 0w1; 1/\downarrow/- >$, $< 1w0; 0/\uparrow/- >$ and $< 1w0; 1/\downarrow/- >$.
   - CFds$_{xwx}$: $< 0w0; 0/\uparrow/- >$, $< 0w0; 1/\downarrow/- >$, $< 1w1; 0/\uparrow/- >$ and $< 1w1; 1/\downarrow/- >$.

3.4.2 Static Address Decoder Faults (sADFs)

Static address decoder faults refer to those faults in the address decoder, which can be sensitized using at most one operation. The faults in the address decoder are assumed to demonstrate similar behavior during the read and the write operation [31]. Only the faults relevant to bit-oriented memory (one bit of information at one address) are discussed here. Figure 3.4 shows four kinds of functional faults that may be present in the address decoder [16].

- **Fault 1**: No memory cell is addressed with a particular address.
- **Fault 2**: A particular memory cell is not accessed by any address.
- **Fault 3**: A particular address accesses more than one memory cell.
- **Fault 4**: A particular memory cell is accessed by more than one address.

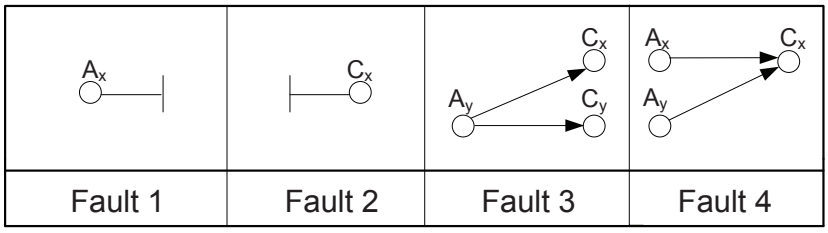

Figure 3.4: Static address decoder faults

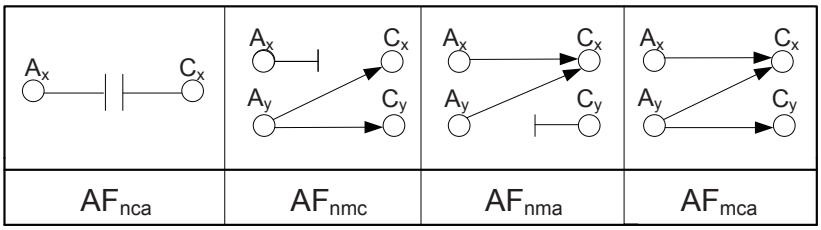

Figure 3.5: Combination of static address decoder faults

As the number of the memory cells and addresses is same, it is obvious that only one of these fault cannot exist in memory and these faults take place in pairs. For example, when Fault 1 occurs Fault 2 or Fault 3 should also take place. With Fault 2, Fault 1 or Fault 4 will also occur. Figure 3.5 shows four possible fault combinations. These faults can be defined as [16]:

- Fault AF_{nca}: It is a combination of Fault 1 and Fault 2, and is called a no cell and no address fault.
- Fault AF_{nmc}: It is a combination of Fault 1 and Fault 3, and is called a no cell and multiple cell fault.
- Fault AF_{nma}: It is a combination of Fault 2 and Fault 4, and is called a no address and multiple address fault.
- Fault AF_{mca}: It is a combination of Fault 3 and Fault 4, and is called a multiple cell and multiple address fault.

3.4.3 Static Peripheral Circuitry Faults (sPFs)

Static peripheral circuitry faults refer to those faults in the peripheral circuitry, which can be sensitized using at most one operation. These faults are related to defects in the rest of the memory (write drivers, sense amplifiers, data registers, precharge circuits). Short and open defects in the peripheral circuitry lead to and are mapped to faults in the memory cell array. Fault models considered for peripheral circuitry are similar to those considered for logic circuits such as stuck-at-faults and bridging faults [10].

3.5 Dynamic Faults

Dynamic faults are timing and speed related faults. Referring back to Figure 3.1, dynamic faults can be also classified into dynamic memory cell array faults, dynamic address decoder faults and dynamic peripheral circuitry faults.

For memory cell array, the dynamic faults are different from static faults in that, they require more than one operation to be applied sequentially in order to sensitize a dynamic fault. Unlike static faults where #O≤1, for dynamic faults #O≥1. Depending upon the number of operations, dynamic faults can be further classified as two-operation dynamic faults (#O = 2), three-operation dynamic faults (#O = 3), etc. It has been stated in literature that the possibility to trigger a dynamic fault falls down with increase in number of operations [3], thus investigation of dynamic fault space is restricted to two-operation dynamic fault space.

For address decoder, dynamic faults mainly means delay faults in the selection of appropriate word line and/or column select lines; while for peripheral circuitry, dynamic faults are caused by slow circuitry (for e.g., write driver) due to defects like partial opens.

In the rest of this section, each of the dynamic fault classes will be discussed.

3.5.1 Dynamic Memory Cell Array Faults (dMCAFs)

Dynamic faults in the memory cell array can be divided into single-cell faults and multi-cell faults. This work restricts the considered dynamic faults to only two-operation single-cell and two-cells dynamic faults; this is because they are the most addressed dynamic faults in the literature.

Single-cell Faults

Single-cell dynamic faults are sensitized by applying more than one operation sequentially to the same cell. As mentioned earlier, a single-cell fault primitive is denoted as $< S/F/R >$. S is the sensitizing operation sequence. Since, we are considering two operations, S can be of the form xO_1yO_2z where $x, y, z \in \{0, 1\}$ and O_1A,O_2 can be read/write operations. Combining all permutations, 18 different sensitizing sequences are possible [24].

- 8 S have the form of write after write operation: $xwywz$. For example, $1w1w0$ denotes a w1 operation to a memory cell whose initial state is '1', immediately followed by a $w0$ operation.
- 2 S have the form of read after read operation: $xrxrx$. For example, $1r1r1$ denotes a r1 operation to a memory cell whose initial state is '1', immediately followed by a $r1$ operation.
- 4 S have the form of write after read operation: $xrxwz$. For example, $1r1w0$ denotes a r1 operation to a memory cell whose initial state is '1', immediately followed by a $w0$ operation.

Table 3.5: Single-cell dynamic FFMs and their FPs [24]

#	FFM	Fault Primitives
1	dRDF	$< 0r0r0/1/1 >$, $< 1r1r1/0/0 >$, $< 0w0r0/1/1 >$, $< 1w1r1/0/0 >$, $< 0w1r1/0/0 >$, $< 1w0r0/1/1 >$
2	dDRDF	$< 0r0r0/1/0 >$, $< 1r1r1/0/1 >$, $< 0w0r0/1/0 >$, $< 1w1r1/0/1 >$, $< 0w1r1/0/1 >$, $< 1w0r0/1/0 >$
3	dIRF	$< 0r0r0/0/1 >$, $< 1r1r1/1/0 >$, $< 0w0r0/0/1 >$, $< 1w1r1/1/0 >$, $< 0w1r1/1/0 >$, $< 1w0r0/0/1 >$
4	dTF	$< 0w0w1/0/- >$, $< 1w1w0/1/- >$, $< 0w1w0/1/- >$, $< 1w0w1/0/- >$, $< 0r0w1/0/- >$, $< 1r1w0/1/- >$
5	dWDF	$< 0w0w0/1/- >$, $< 1w1w1/0/- >$, $< 0w1w1/0/- >$, $< 1w0w0/1/- >$, $< 0r0w0/1/- >$, $< 1r1w1/0/- >$

- 4 S have the form of read after write operation: $xwyry$. For example, $1w1r1$ denotes w1 operation to a memory cell whose initial state is '1', immediately followed by a $r1$ operation.

In the FP notation, F denotes the behavior/state of the faulty cell: $F \in \{0, 1\}$. R denotes the logical value at the output of the memory: $R \in \{0, 1, -\}$. $R = -$ signifies that the last operation was a write operation and output data is not available. Based on the values of S, F and R, 30 dynamic single-cell FPs are determined. All these FPs can be compiled in 5 FFMs as shown in Table 3.5 [24]; they are explained next.

1. **Dynamic Read Destructive Fault (dRDF)**: A read operation on a cell, performed immediately after a read/write operation on the same cell, changes the data in the cell and returns the new incorrect value on output. dRDF consists of six FPs: $< 0r0r0/1/1 >$, $< 1r1r1/0/0 >$, $< 0w0r0/1/1 >$, $< 1w1r1/0/0 >$, $< 0w1r1/0/0 >$ and $< 1w0r0/1/1 >$.

2. **Dynamic Deceptive Read Destructive Fault (dDRDF)**: A read operation on a cell, performed immediately after a read/write operation on the same cell, changes the data in the cell but returns the old correct value on output. dRDF consists of six FPs: $< 0r0r0/1/0 >$, $< 1r1r1/0/1 >$, $< 0w0r0/1/0 >$, $< 1w1r1/0/1 >$, $< 0w1r1/0/1 >$ and $< 1w0r0/1/0 >$.

3. **Dynamic Incorrect Read Fault (dIRF)**: A read operation on a cell, performed immediately after a read/write operation on the same cell, returns an incorrect value on output while the cell contains the correct logic value. dIRF consists of six FPS: $< 0r0r0/0/1 >$, $< 1r1r1/1/0 >$, $< 0w0r0/0/1 >$, $< 1w1r1/1/0 >$, $< 0w1r1/1/0 >$ and $< 1w0r0/0/1 >$.

4. **Dynamic Transition Fault (dTF)**: A transition write operation on a cell, performed immediately after a read/write operation on the same cell, fails. dTF consists of six FPs: $< 0w0w1/0/- >$, $< 1w1w0/1/- >$, $< 0w1w0/1/- >$, $< 1w0w1/0/- >$, $< 0r0w1/0/- >$ and $< 1r1w0/1/- >$.

5. **Dynamic Write Destructive Fault (dWDF)**: A non-transition write operation ($0w0$ or $1w1$) on a cell, performed immediately after a read/write operation on the same cell, reverses the logic value stored in the cell. dWDF consists of six FPs: $< 0w0w0/1/- >$, $< 1w1w1/0/- >$, $< 0w1w1/0/- >$, $< 1w0w0/1/- >$, $< 0r0w0/1/- >$ and $< 1r1w1/0/- >$.

Two-cell Faults

Two-cell dynamic faults are sensitized by applying more than one operation sequentially to two cells: the aggressor cell and the victim cell. As in the case of single-cell dynamic operations, here also we restrict our fault space to two-operation dynamic faults. Depending on the order in which sensitizing operations(state) is applied to the aggressor cell and the victim cell, four different sensitizing sequences can be distinguished:

1. S_{aa}: Both sensitizing operations are sequentially applied to the aggressor cell. The fault is sensitized in the victim cell.

2. S_{vv}: Both sensitizing operations are sequentially applied to the victim cell. The fault is sensitized if the aggressor cell is in a particular state.

3. S_{av}: First sensitizing operation is applied to the aggressor cell followed immediately by a second operation to the victim cell. The sensitization of the fault requires access to two different cells sequentially.

4. S_{va}: First sensitizing operation is applied to the victim cell followed immediately by a second operation to the aggressor cell. The sensitization of the fault requires access to two different cells sequentially.

Since, we are considering two operations, S can be of the form xO_1yO_2z where $x, y, z \in \{0, 1\}$ and O_1,O_2 can be read/write operations. Combining all permutations and 4 possible orders results into 18*4 = 72 different sensitizing sequences.

Faults caused by S_{aa}

S_{aa} requires both sensitizing operations to be applied sequentially to the aggressor cell. Given that the victim cell is in a certain state, the logical value stored in the victim cell flips as a result of the fault effect. Since both operations are applied to the aggressor cell, FP $< S_{aa}/F/R >$ is denoted as $< S_a; S_v/F/R >$. $S_a = yO_1zO_2t$ and $S_v = x$ where $x, y, z, t \in \{0, 1\}$ and, O_1 and O_2 are read/write operations. S_a is the SOS applied to the aggressor cell and S_v is the state of the victim cell. F represents the flip of the victim cell and is denoted as $\overline{x}$. $R = -$ as no operation is performed on the victim cell. Thus, faults sensitized by S_{aa} can be denoted as $< yO_1zO_2t; x/\overline{x}/- >$.

Table 3.6: Two-cell dynamic FPs and FFMs caused by S_{aa} [24]

#	FFM	Fault Primitives
1	dCFds$_{ww}$	$<0w0w0;x/\overline{x}/->$, $<1w1w1;x/\overline{x}/->$, $<0w0w1;x/\overline{x}/->$, $<1w1w0;x/\overline{x}/->$, $<0w1w0;x/\overline{x}/->$, $<1w0w1;x/\overline{x}/->$, $<0w1w1;x/\overline{x}/->$, $<1w0w0;x/\overline{x}/->$
2	dCFds$_{wr}$	$<0w0r0;x/\overline{x}/->$, $<1w1r1;x/\overline{x}/->$, $<0w0r1;x/\overline{x}/->$, $<1w0r0;x/\overline{x}/->$
3	dCFds$_{rw}$	$<0r0w0;x/\overline{x}/->$, $<1r1w1;x/\overline{x}/->$, $<0r0w1;x/\overline{x}/->$, $<1r1w0;x/\overline{x}/->$
4	dCFds$_{rr}$	$<0r0r0;x/\overline{x}/->$, $<1r1r1;x/\overline{x}/->$

The operation sequence yO_1zO_2t gives 18 possible sensitizing sequences. Victim cell state x can have two values {0,1}. So, in total 18*2 = 36 FPs are possible for S_{aa}. Table 3.6 shows all FPs and FFM for S_{aa}.

- **Dynamic Disturb Coupling Fault (dCFds)**: Two operations applied sequentially on the aggressor cell sensitize the fault by flipping the value in the victim cell, given that the victim cell is in a certain state. Depending on the operations performed on the aggressor cell dCFds can be divided in four sub-parts.
  - dCFds$_{ww}$: Operations applied in a write after write sequence. dCFds$_{ww}$ consists of 16 FPs: $<0w0w0;x/\overline{x}/->$, $<1w1w1;x/\overline{x}/->$, $<0w0w1;x/\overline{x}/->$, $<1w1w0;x/\overline{x}/->$, $<0w1w0;x/\overline{x}/->$, $<1w0w1;x/\overline{x}/->$, $<0w1w1;x/\overline{x}/->$ and $<1w0w0;x/\overline{x}/->$.
  - dCFds$_{wr}$: Operations applied in a read after write sequence. dCFds$_{wr}$ consists of 8 FPs: $<0w0r0;x/\overline{x}/->$, $<1w1r1;x/\overline{x}/->$, $<0w0r1;x/\overline{x}/->$ and $<1w0r0;x/\overline{x}/->$.
  - dCFds$_{rw}$: Operations applied in a write after read sequence. dCFds$_{rw}$ consists of 8 FPs: $<0r0w0;x/\overline{x}/->$, $<1r1w1;x/\overline{x}/->$, $<0r0w1;x/\overline{x}/->$ and $<1r1w0;x/\overline{x}/->$.
  - dCFds$_{rr}$: Operations applied in a read after read sequence. dCFds$_{rr}$ consists of 4 FPs: $<0r0r0;x/\overline{x}/->$ and $<1r1r1;x/\overline{x}/->$.

Faults caused by S_{vv}

S_{vv} requires both sensitizing operations to be applied sequentially on the victim cell. Given that the aggressor cell is in a certain state, the sensitizing operations sensitizes a fault in the victim cell. Since both operations are applied to the victim cell, FP $<S_{vv}/F/R>$ is denoted as $<S_a;S_v/F/R>$. $S_a = x$ and $S_v = yO_1zO_2t$ where $x, y, z, t \in \{0, 1\}$ and, O_1 and O_2 are read/write operations. S_a is the state of the aggressor cell and S_v is the SOS applied to the victim cell. F represents the faulty behavior of the victim cell. R gives the read results if the second operation applied to victim cell (i.e.,

Table 3.7: Two-cell dynamic FPs and FFMs caused by S_{vv} [24]

#	FFM	Fault Primitives
1	dCFrd	$< x; 0r0r0/1/1 >$, $< x; 1r1r1/0/0 >$, $< x; 0w0r0/1/1 >$, $< x; 1w1r1/0/0 >$, $< x; 0w1r1/0/0 >$, $< x; 1w0r0/1/1 >$
2	dCFdrd	$< x; 0r0r0/1/0 >$, $< x; 1r1r1/0/1 >$, $< x; 0w0r0/1/0 >$, $< x; 1w1r1/0/1 >$, $< x; 0w1r1/0/1 >$, $< x; 1w0r0/1/0 >$
3	dCFir	$< x; 0r0r0/0/1 >$, $< x; 1r1r1/1/0 >$, $< x; 0w0r0/0/1 >$, $< x; 1w1r1/1/0 >$, $< x; 0w1r1/1/0 >$, $< x; 1w0r0/0/1 >$
4	dCFtr	$< x; 0w0w1/0/- >$, $< x; 1w1w0/1/- >$, $< x; 0w1w0/1/- >$, $< x; 1w0w1/0/- >$, $< x; 0r0w1/0/- >$, $< x; 1r1w0/1/- >$
5	dCFwd	$< x; 0w0w0/1/- >$, $< x; 1w1w1/0/- >$, $< x; 0w1w1/0/- >$, $< x; 1w0w0/1/- >$, $< x; 0r0w0/1/- >$, $< x; 1r1w1/0/- >$

O_2) is a read operation, otherwise $R = -$. Thus, faults sensitized by S_{vv} can be denoted as $< x; yO_1zO_2t/F/R >$. From the discussion on single-cell dynamic faults, we know that $< x; S_v/F/R >$ with $x \in \{0, 1\}$ is a superset of $< S_v/F/R >$. Thus, the notation $< x; yO_1zO_2t/F/R >$ represents 60 FPs: 30 Fps denoted as $< 0; yO_1zO_2t/F/R >$ and 30 FPs denoted as$< 1; yO_1zO_2t/F/R >$. Table 3.6 summarizes all FPs and FFMs for S_{vv}.

1. **Dynamic Read Destructive coupling Fault (dCFrd)**: A read operation on the victim cell, performed immediately after a read/write operation on the same cell, causes a transition in the victim cell and returns the new incorrect value on output, given that the aggressor cell is in a particular state. dCFrd has the forms of $< x; ywzrz/\overline{z}/\overline{z} >$ and $< x; yryry/\overline{y}/\overline{y} >$ and consists of 12 FPs: $< x; 0r0r0/1/1 >$, $< x; 1r1r1/0/0 >$, $< x; 0w0r0/1/1 >$, $< x; 1w1r1/0/0 >$, $< x; 0w1r1/0/0 >$ and $< x; 1w0r0/1/1 >$.

2. **Dynamic Deceptive Read Destructive coupling Fault (dCFdrd)**: A read operation on the victim cell, performed immediately after a read/write operation on the same cell, causes a transition in the victim cell but returns the old correct value on output, given that the aggressor cell is in a particular state. dCFdrd has the forms of $< x; ywzrz/\overline{z}/z >$ and $< x; yryry/\overline{y}/y >$ and consists of 12 FPs: $< x; 0r0r0/1/0 >$, $< x; 1r1r1/0/1 >$, $< x; 0w0r0/1/0 >$, $< x; 1w1r1/0/1 >$, $< x; 0w1r1/0/1 >$ and $< x; 1w0r0/1/0 >$.

3. **Dynamic Incorrect Read coupling Fault (dCFir)**: A read operation on the victim cell, performed immediately after a read/write operation on the same cell, returns an incorrect value on output while the cell contains the correct logic,

given that the aggressor cell is in a particular state. dCFir has the forms of $< x; ywzrz/z/\overline{z} >$ and $< x; yryry/y/\overline{y} >$ and consists of 12 FPs: $< x; 0r0r0/0/1 >$, $< x; 1r1r1/1/0 >$, $< x; 0w0r0/0/1 >$, $< x; 1w1r1/1/0 >$, $< x; 0w1r1/1/0 >$ and $< x; 1w0r0/0/1 >$.

4. **Dynamic Transition coupling Fault (dCFtr)**: A transition write operation ($0w1$ or $1w0$) on the victim cell, performed immediately after a read/write operation on the same cell, results in failing second operation (O_2), given that the aggressor cell is in a particular state. dCFtr has the forms of $< x; ywzw\overline{z}/z/- >$ and $< x; yrywy\overline{y}/y/- >$ and consists of 12 FPs: $< x; 0w0w1/0/- >$, $< x; 1w1w0/1/- >$, $< x; 0w1w0/1/- >$, $< x; 1w0w1/0/- >$, $< x; 0r0w1/0/- >$ and $< x; 1r1w0/1/- >$.

5. **Dynamic Write Destructive coupling Fault (dCFwd)**: A non-transition write operation ($0w0$ or $1w1$) on the victim cell, performed immediately after a read/write operation on the same cell, reverses the logic value stored in victim cell, given that the aggressor cell is in a particular state. dCFwd has the forms of $< x; ywzwz/\overline{z}/- >$ and $< x; yrywy/\overline{y}/- >$ and consists of 12 FPs: $< x; 0w0w0/1/- >$, $< x; 1w1w1/0/- >$, $< x; 0w1w1/0/- >$, $< x; 1w0w0/1/- >$, $< x; 0r0w0/1/- >$ and $< x; 1r1w1/0/- >$.

Faults caused by S_{av}

S_{av} requires first of the two sensitizing operation to be applied to the aggressor cell followed immediately by the second operation on the victim cell. FP $< S_{av}/F/R >$ is denoted as $< S_a; S_v/F/R >$. $S_a = xO_1y$ and $S_v = zO_2t$ where $x, y, z, t \in \{0, 1\}$ and, O_1 and O_2 are read/write operations: $xO_1y, zO_2t \in \{0w0, 0w1, 1w0, 1w1, 0r0, 1r1\}$. $S_a = xO_1y$ denotes the sensitizing operation applied to the aggressor cell and $S_v = zO_2t$ is the sensitizing sequence applied to the victim cell. F represents the faulty behavior of the victim cell. R gives the read results if the applied operation on the victim cell (i.e., O_2) is a read operation, otherwise $R = -$. Thus, faults sensitized by S_{av} can be denoted as $< xO_1y; zO_2t/F/R >$ and represents 60 FPs as shown:

- if $S_v = zwt$ then $F = \overline{t}$ and $R = -$; $z, t \in \{0, 1\}$, $S_v \in \{0w0, 0w1, 1w0, 1w1\}$. This case can be denoted as $< S_a; zwt; \overline{t} >$ and results in 6*4 = 24 FPs as S_a can be any one of the six read/write operation sequences.
- if $S_v = yry$; $y \in \{0, 1\}$, $S_v \in \{0r0, 1r1\}$
  - if $F = y$ then $R = \overline{y}$. This case can be denoted as $< S_a; yry/y/\overline{y} >$ and results in 12 FPs.
  - if $F = \overline{y}$ then $R \in \{y, \overline{y}\}$. This case can be denoted as $< S_a; yry/\overline{y}/y >$ (12 FPs) and $< S_a; yry/\overline{y}/\overline{y} >$ (12 FPs).

Table 3.8 summarizes all FPs and FFM for S_{av}. In the table, xOy denotes any read/write operation: $xOy \in \{0w0, 0w1, 1w0, 1w1, 0r0, 1r1\}$. The FPs are compiled into 5 FFMs; Note that the names used for FFMs are the same as that used for faults based in S_{vv} sensitizing operations.

Table 3.8: Two-cell dynamic FPs and FFMs caused by S_{av} [24]

#	FFM	Fault Primitives
1	dCFrd	$< xOy; 0r0/1/1 >, < xOy; 1r1/0/0 >$
2	dCFdrd	$< xOy; 0r0/1/0 >, < xOy; 1r1/0/1 >$
3	dCFir	$< xOy; 0r0/0/1 >, < xOy; 1r1/1/0 >$
4	dCFtr	$< xOy; 0w1/0/- >, < xOy; 1w0/1/- >$
5	dCFwd	$< xOy; 0w0/1/- >, < xOy; 1w1/0/- >$

1. **Dynamic Read Destructive coupling Fault (dCFrd)**: A read operation on the victim cell, performed immediately after a read/write operation on the aggressor cell, causes a transition in the victim cell and returns the new incorrect value on output. dCFrd has the form of $< S_a; zrz/\overline{z}/\overline{z} >$ and consists of 12 FPs: $< xOy; 0r0/1/1 >$ and $< xOy; 1r1/0/0 >$.

2. **Dynamic Deceptive Read Destructive coupling Fault (dCFdrd)**: A read operation on the victim cell, performed immediately after a read/write operation on the aggressor cell, causes a transition in the victim cell but returns the old correct value on output. dCFdrd has the form of $< S_a; zrz/\overline{z}/z >$ and consists of 12 FPs: $< xOy; 0r0/1/0 >$ and $< xOy; 1r1/0/1 >$.

3. **Dynamic Incorrect Read coupling Fault (dCFir)**: A read operation on the victim cell, performed immediately after a read/write operation on the aggressor cell, returns an incorrect value on output while the cell contains the correct logic. dCFdrd has the form of $< S_a; zrz/z/\overline{z} >$ and consists of 12 FPs: $< xOy; 0r0/0/1 >$ and $< xOy; 1r1/1/0 >$.

4. **Dynamic Transition coupling Fault (dCFtr)**: A transition write operation ($0w1$ or $1w0$) on the victim cell, performed immediately after a read/write operation on the aggressor cell, results in a failing write operation on the victim cell. dCFtr has the form of $< S_a; zw\overline{z}/z/- >$ and consists of 12 FPs: $< xOy; 0w1/0/- >$ and $< xOy; 1w0/1/- >$.

5. **Dynamic Write Destructive coupling Fault (dCFwd)**: A non-transition write operation ($0w0$ or $1w1$) on the victim cell, performed immediately after a read/write operation on the aggressor cell, reverses the logic value stored in the victim cell. dCFwd has the form of $< S_a; zwz/\overline{z}/- >$ and consists of 12 FPs: $< xOy; 0w0/1/- >$ and $< xOy; 1w1/0/- >$.

Faults caused by S_{va}

S_{va} requires first of the two sensitizing operation to be applied to the victim cell followed immediately by the second operation on the aggressor cell. FP $< S_{va}/F/R >$ is denoted as $< S_v/F/R; S_a >$. $S_v = zO_1t$ and $S_a = xO_2y$ where $x, y, z, t \in \{0, 1\}$ and, O_1 and O_2 are read/write operations: $zO_1t, xO_2y \in \{0w0, 0w1, 1w0, 1w1, 0r0, 1r1\}$. $S_v = zO_1t$ denotes the sensitizing operation applied to the victim cell and $S_a = xO_2y$ is

Table 3.9: Two-cell dynamic FPs and FFMs caused by S_{va} [24]

#	FFM	Fault Primitives
1	dCFrd	$< 0r0/1/1; xOy >$, $< 1r1/0/0; xOy; >$
2	dCFdrd	$< 0r0/1/0; xOy >$, $< 1r1/0/1; xOy >$
3	dCFir	$< 0r0/0/1; xOy >$, $< 1r1/1/0; xOy >$
4	dCFtr	$< 0w1/0/-; xOy >$, $< 1w0/1/-; xOy >$
5	dCFwd	$< 0w0/1/-; xOy >$, $< 1w1/0/-; xOy >$

the sensitizing operation sequence applied to the aggressor cell. The number of faults and FFMs for S_{va} is same as for S_{av}. The only difference is the order in which the aggressor cell and the victim cell are accessed. Faults sensitized by S_{va} can be denoted as $< zO_1t/F/R; xO_1y; >$ and represents 60 FPs as shown:

- if $S_v = zwt$ then $F = \overline{t}$ and $R = -$; $z, t \in \{0, 1\}$, $S_v \in \{0w0, 0w1, 1w0, 1w1\}$. This case can be denoted as $< zwt; \overline{t}; S_a >$ and results in 24 FPs.
- if $S_v = zrz$; $z \in \{0, 1\}$, $S_v \in \{0r0, 1r1\}$
  - if $F = z$ then $R = \overline{z}$. This case can be denoted as $< zrz/z/\overline{z}; S_a >$ and results in 12 FPs.
  - if $F = \overline{z}$ then $R \in \{z, \overline{y}\}$. This case can be denoted as $< zrz/\overline{z}/z; S_a >$ (12 FPs) and $< zrz/\overline{z}/\overline{z}; S_a >$ (12 FPs).

Table 3.9 summarizes all FPs and FFM for S_{va}. In the table, xOy denotes any read/write operation: $xOy \in \{0w0, 0w1, 1w0, 1w1, 0r0, 1r1\}$

1. **Dynamic Read Destructive coupling Fault (dCFrd)**: A read operation on the aggressor cell, performed immediately after a read/write operation on the victim cell, causes a transition in the victim cell and returns the new incorrect value on output. dCFrd has the form of $< zrz/\overline{z}/\overline{z}; S_a >$ which consists of 12 FPs: $< 0r0/1/1; xOy >$ and $< 1r1/0/0; xOy >$.
2. **Dynamic Deceptive Read Destructive coupling Fault (dCFdrd)**: A read operation on the aggressor cell, performed immediately after a read/write operation on the victim cell, causes a transition in the victim cell but returns the old correct value on output. dCFdrd has the form of $< zrz/\overline{z}/z; S_a >$ and consists of 12 FPs: $< 0r0/1/0; xOy >$ and $< 1r1/0/1; xOy >$.
3. **Dynamic Incorrect Read coupling Fault (dCFir)**: A read operation on the aggressor cell, performed immediately after a read/write operation on the victim cell, returns an incorrect value on output while the cell contains the correct logic. dCFdrd has the form of $< zrz/z/\overline{z}; S_a >$ and consists of 12 FPs: $< 0r0/0/1; xOy >$ and $< 1r1/1/0; xOy; >$.
4. **Dynamic Transition coupling Fault (dCFtr)**: A read operation on the aggressor cell, performed immediately after a read/write operation on the victim cell, results in failing write operation on the victim cell. dCFtr has the

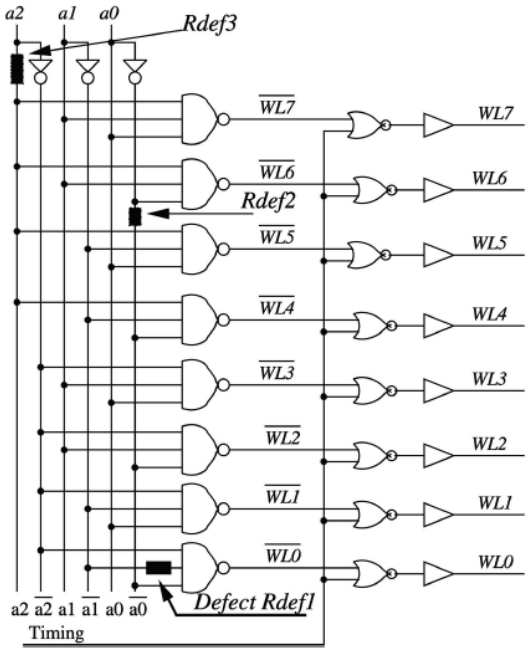

Figure 3.6: Example of an inter-gate open in a CMOS address decoder [22]

form of $< zw\overline{z}/z/-; S_a >$ and consists of 12 FPs: $< 0w1/0/-; xOy >$ and $< 1w0/1/-; xOy >$.

5. **Dynamic Write Destructive coupling Fault (dCFwd)**:A read operation on the aggressor cell, performed immediately after a read/write operation on the victim cell, reverses the logic value stored in the victim cell. dCFwd has the form of $< zwz/\overline{z}/-; S_a >$ and consists of 12 FPs: $< 0w0/1/-; xOy >$ and $< 1w1/0/-; xOy >$.

3.5.2 Dynamic Address Decoder Faults (dADFs)

Dynamic faults considered in this work are the address decoder delay faults (ADDFs). It is an important class of memory faults. Klaus [29] claims that the defect-per-million level decreases by 670 due to tests for ADDFs. This class of faults is getting more and more important for high speed memories. Resistive open defects are the major cause of ADDFs. Depending on the physical presence of the defect in the address decoder, resistive opens can be differentiated between:

- Inter-gate opens: Figure 3.6 shows three examples of inter-gate open defects (*Rdef1, Rdef2, Rdef3*). *Rdef1* is located in the line connecting $\overline{a1}$ to the NAND gate decoding WL0. Due to the long global wiring in address decoders, the probability of inter-gate defects is at least one order of magnitude larger than the probability of intra-gate defects [29].

- Intra-gate opens: Figure 3.7 shows an example of an intra-gate open defect(*Rdef4*) in the source of the pull up transistor for input $\overline{a1}$.

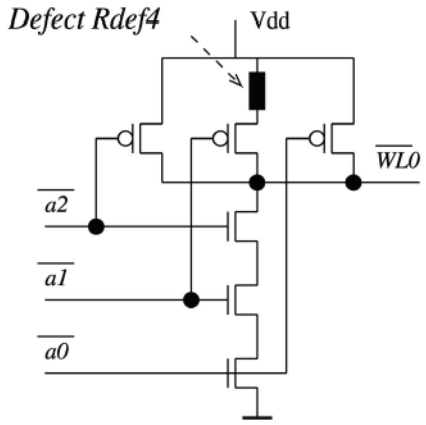

Figure 3.7: Example of an intra-gate open [22]

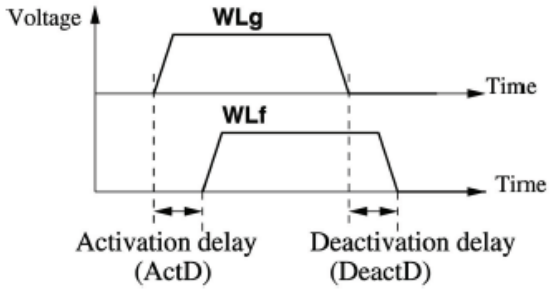

Figure 3.8: Activation and deactivation delays [18]

These opens may cause the activation or the deactivation of a word line (or a column select line) to be delayed. Figure 3.8 shows an example of memory access with a good word line *WLg*, immediately followed by memory access with a potentially faulty word line *WLf*. In presence of an address decoder delay fault, the activation or deactivation of *WLf* will be delayed causing an Activation Delay (ActD) or a deactivation delay (DeactD) fault [18]. Figure 3.9 shows a simulation example of the impact of a resistive open on the word line, while Figure 3.10 shows an example of the impact of a resistive open on the column select line. Two faults models for the ADDFs are defined [22]:

- Activation Delay (ActD): This is a delay-related fault that effects the rising edge of word line or column select signal due to resistive defects. ActD can be observed due to both inter-gate opens and intra-gate opens.
- Deactivation Delay (DeactD): This is a delay-related fault that effects the falling edge of word line or column select signal due to resistive defects. DeactD can be observed due to both inter-gate opens and intra-gate opens.

It is worth noting that ADDFs occur for intermediate values of resistive defects. For very high resistive value, the open will behave as a open connection and the resulting fault will be a static address decoder fault.

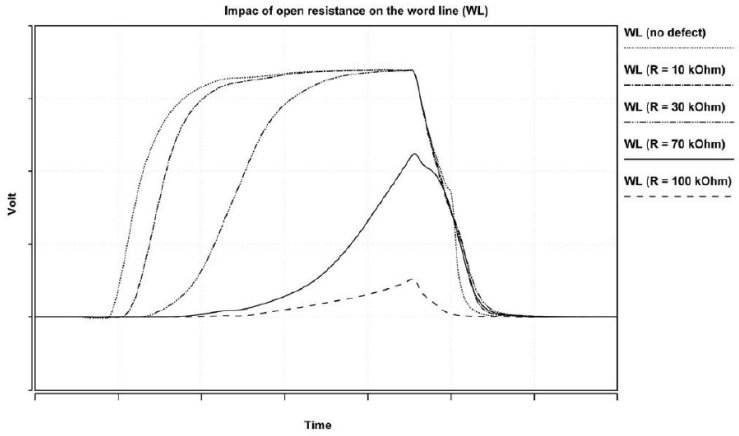

Figure 3.9: Impact of open defect on word line timing [18]

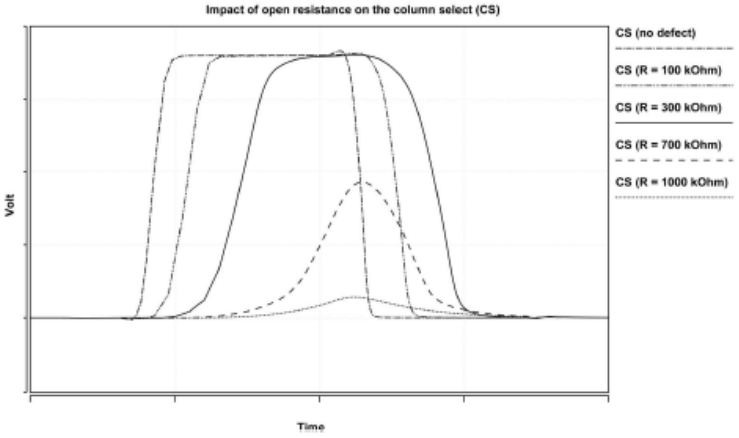

Figure 3.10: Impact of open defect on column select timing [18]

3.5.3 Dynamic Peripheral Circuitry Faults (dPFs)

Dynamic peripheral circuitry faults are speed related faults in the peripheral circuitry (for e.g., write drivers, sense amplifiers and precharge circuits) or faults due to excessive leakage (due to leaky pass transistors) [56]. Below is the description of the faulty behavior of the peripheral circuitry and operation sequences required to sensitize these faults.

1. **Slow Write Driver Fault (SWDF)**: Presence of a defect (open, short, bridge) in the write driver (for example, a partial open via) can lead to a slow write driver. As a result of the defect, the voltage difference between the complementary bit lines (BL and $\overline{BL}$) is decreased. This can cause the write operation to fail. SWDF can be sensitized by sequentially applying complementary write operations on two different memory cells which use the same write driver. For example, applying a $w0$ operation on a memory cell immediately followed by a $w1$ operation on another memory cell, but in the same column as that of the previous memory cell, will sensitize the slow write driver fault. The second write operation (i.e., $w1$) will fail.

2. **Slow Sense Amplifier Fault (SSAF)**: Presence of a defect in the sense amplifier can lead to a slow or asymmetric sense amplifier. An asymmetric sense amplifier will have some voltage offset between the two bit lines. As a result of the defect, the read operation on the memory cell will produce incorrect result. SSAF can be sensitized by sequentially applying complementary operations on two different memory cells which share the same sense amplifier. The second of the two operations should be a read operation. For example, applying a $w0$ operation on a memory cell immediately followed by a $r1$ operation on another memory cell, but in the same column as that of the previous memory cell, will sensitize the slow sense amplifier fault. The read operation will produce an incorrect result on output.

3. **Slow Precharge Circuit Fault (SPRF)**: Presence of a defect in the precharge circuit can lead to a slow or biased precharge circuit. A biased precharge circuit will not precharge both bit lines to same voltage level. As a result of the defect, the read operation on the memory cell will produce incorrect result. Sensitizing conditions for SPRF are same as for SSAF faults.

4. **Bit Line Imbalance Fault (BLIF)**: As the transistor sizes are decreasing, the proportion of the leakage current with respect to switching current is increasing. With the shrinking process technology sizes, transistors increasingly draw more current in the off-state. The effect of leakage current on pass transistors might impact the read operations applied to the memory cell. In order to correctly detect the logical value stored in a cell, during the read operation the voltage difference developed between the complementary bit lines should be higher than a certain threshold value. But the leakage current can act against the developed voltage difference and lead to an incorrect read operation. For example, assume a column of memory cells where all cells store logic value '1' except one which has the logic value '0'. When a read operation is performed on the cell storing '0', the voltage difference developed between bit lines can decrease under the required threshold level and even neutralized by the leaking pass transistors of the cells storing logic value '1'. Ideally, while reading logic value '0', voltage of $\overline{BL}$ should be higher than that of BL but due to the leakage current from other cells, the voltage of BL can be higher, resulting in an incorrect read data output. BLIF can be sensitized by performing a read $\overline{x}$ operation on a memory cell when all other cells sharing the same bit lines contain logic value x.

State of the Art in SRAM Diagnosis

4

This chapter presents the existing memory diagnosis approaches along with their strengths and drawbacks. It also highlights the need of a new memory diagnosis scheme and lists the requirements for such a scheme.

This chapter is organized as follows. Section 4.1 differentiates between memory fault detection and diagnosis. Section 4.2 classifies the existing diagnosis approaches into three different classes, namely probability-based diagnosis, signature-based diagnosis and design for diagnosis. These approaches are discussed in Section 4.3 to 4.5 respectively. Finally, Section 4.6 summarizes the main shortcomings of the existing approaches and justifies the need for a new approach.

4.1 Detection versus Diagnosis

Semiconductor memories are an integral part of modern VLSI and ULSI circuits. For SoC designs, memories are the most space-consuming components; and therefore they dominate the chip yield. According to the Semiconductor Industry Association (SIA), the on-chip share of memory is expected to increase to 94% by 2014 [5].

Just like in any other technology, faults are inherent in the new/established memory fabrication procedure. In today's competitive world, it is necessary to satisfy very high quality constraints ranging from 50 DPM for computers to less than 10 DPM for critical applications. The importance of memory fault testing cannot be therefore overlooked.

There are two major aspects to testing: fault detection and fault diagnosis. Fault detection stands for monitoring a system and identifying whether a fault has occurred while fault diagnosis pinpoints the type of fault, its location and tries to identify the reasons for fault manifestation. With developing technology, the exponential increase in the density of memory components has made memory testing and fault analysis very important. Along with satisfying quality and reliability issues, it is desirable to have a fast yield learning curve. Thus, it is not sufficient to just screen faulty memories after production but efficient mechanisms are required to precisely point out the physical presence of a fault in order to take corrective measures and get the yield as high as possible within minimum time. Manufacturing yield is an extremely critical economical parameter in the semiconductor industry. A fast yield ramp-up is a must!

The final outcome of the diagnosis is that the root cause of failures is located and understood, and therefore either the design, manufacturing process or test program will be tuned in order to realize a better yield, quality and reliability. Another advantage of the diagnosis process is that the manufacturer accumulates the knowledge on importance of different faults and their occurrence probabilities. The original test programs can be modified and/or optimized to target the most important faults and thus greatly improve efficiency and reduce the test cost [32]. Customer returns i.e., the products which passed

the original test but failed in the field, can be analyzed using the diagnosis process and can provide insight to help identify or define the faults the product contains. This is of use for finding and locating the design and/or process errors and inconsistency. Thus, memory diagnosis is very important for improving the fault testing procedure and improve the yield.

A well-established memory diagnosis process helps to reduce the ever-increasing test cost of newer ICs. Firstly, it helps to eliminate the faults by allowing the required corrections to be made in the process/design. Secondly, by using the information derived from the diagnosis, the tests for memory chips can be limited to detecting the set of faults occurring in that particular memory architecture, instead of testing the whole possible memory fault space. A full diagnosis process requires on-chip and off-chip analysis using several algorithmic sequences. It includes a set of diagnostic test algorithms and a method/tool to analyze the collected diagnostic data. This data is further processed to generate a detailed fault report for failure analysis.

Traditional test algorithms aim at just detecting faults and do not have implicit support for fault diagnosis. Memory fault diagnosis has not been given as much importance as the testing of memory chips. But with the growing importance of fault diagnosis, this subject is now being studied by a number of researchers. In the recent past, more attention has been paid to this problem and several diagnostic tests have been introduced, which are capable not only of detecting but also of identifying the fault.

4.2 Classification of Existing Diagnosis Schemes

Despite the huge growth potential of the field of memory diagnosis, little work has been published on to this important aspect by the electronics industry. New faults models and test generation techniques are being developed. Due to the company confidential nature of this field, not much information about the existing memory diagnosis procedures in industry is openly available, which makes it extremely difficult to take advantage of experience drawn from existing approaches.

The techniques being used for memory diagnosis are still evolving. Depending on the underlying technique and principles of operation, the majority of the published memory diagnosis techniques (mainly by academia) can be classified into three categories (see Figure 4.1):

- **Probability-based diagnosis**: These are non-deterministic methods that were introduced during the early stages of memory diagnosis research area [12, 35, 54]. These methods are not based on any mathematical/analytical thinking but they define the occurrence probability of a fault using statistical analysis.

- **Signature-based diagnosis**: This is the most common approach to memory diagnosis. Diagnostic tests provide unique signatures for each targeted fault which are then used to identify the fault type [28, 45, 58]. These techniques make use of march tests for the diagnosis.

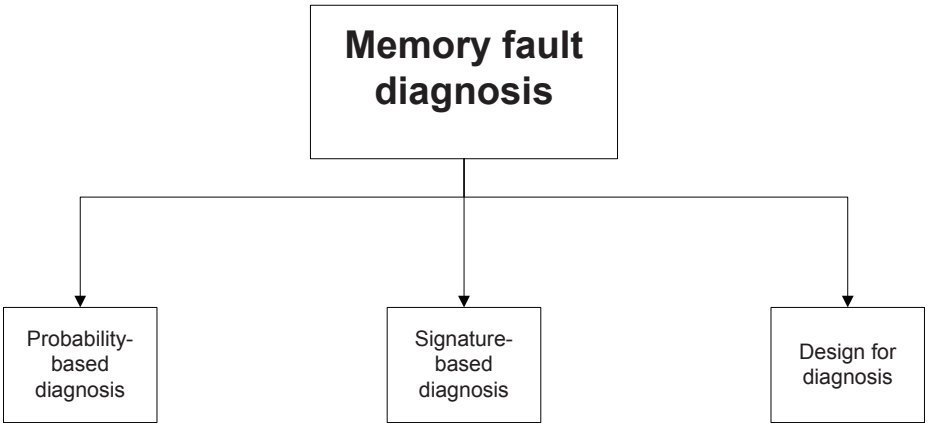

Figure 4.1: Classification of different diagnosis approaches

- **Design for Diagnosis**: These techniques advocate the use of additional hardware for the purpose of memory diagnosis [43, 44]. Applied diagnostic tests along with the inserted special hardware help in diagnosing the fault and in locating faulty blocks of the memory.

In the rest of this section each of the above classes will be discussed, and their drawbacks will be highlighted.

4.2.1 Probability-based diagnosis

Probability-based fault analysis methods for memory diagnosis were introduced in the nineties during the onset of the field of memory fault diagnosis [12, 54]. These methods did not find a wide application and were soon replaced by other diagnosis techniques. The basic principle behind these methods is to implement a large number of random experiments. Fault sites are narrowed down by appropriately overlapping the faulty areas and faults are distinguished by comparing the pass/fail data with statistically generated fault probabilities.

David and Fuentes demonstrated a diagnosis approach that uses pseudo-random experiments to distinguish between different faults [12]. The fundamental principle behind the approach is to have tests target different faults. The strategy used by the approach involves several steps. First of all, random testing experiments are performed on different areas of the memory cell array i.e., a subset of the memory words. Each experimental result gives some information to distinguish the targeted faults. Each experiment consists of initializing an area, testing an area, and then obtaining probabilistic conclusions. If the operator is not absolutely sure of the result, further experiments can be performed to determine whether the result is good or not. A similar methodology is introduced by Savir et. al where five pseudo-random tests for coupling faults are described [51].

Probability-based methods showed that interesting deductions can be obtained from random testing experiments. The diagnosis can guarantee with a degree of certainty confidence that a certain fault type exists in the device being tested. Some methods have been also been shown to determine the fault location but only for limited number

of faults [12]. The proposed random testing methodology depends on tests of a higher order, such as $O(N^2)$, which leads to high test application times. These methods are not deterministic in nature, have a low fault coverage and are rather time-consuming in terms of the test time. The fault type and fault location can be identified only for a small set of faults. These methods were soon replaced by other diagnosis techniques.

4.2.2 Signature-based diagnosis methods

Signature-based diagnosis methods apply diagnostic march tests and use the results to identify the fault. A march test is a finite sequence of march elements; and a march element is a finite sequence of operations or primitives applied to every memory cell before proceeding to next cell in a given address direction [53]. March tests have become dominant because their test times are in linear proportion with the size of the memory, and their fault coverage can be proved mathematically [13]. Diagnostic march tests are march tests that provide unique signatures for targeted faults. The number of bits in diagnostic signature is equal to the number of read operations in the march test. When a diagnostic test is applied on the memory, all the read operations that fail are recorded along with the information on failing memory cells. A failed read operation is denoted as a '1' in the signature bit of that particular read operation for that particular memory cell. A '0' in signature denotes that the cell is fault-free or that the fault cannot be detected through the applied test. Depending upon the test, unique or non-unique signatures are generated for different memory faults and a fault dictionary is created. All the faults with unique signatures can be distinguished.

The underlying principle of the signature-based diagnosis can be demonstrated by means of a simple example using the well-known March C- test: $\{ \Updownarrow (w0); \Uparrow (r0, w1); \Uparrow (r1, w0); \Downarrow (r0, w1); \Downarrow (r1, w0); \Updownarrow (r0) \}$ [34]. Signatures can be generated for targeted fault models and their corresponding syndromes. For example, signatures concerning state faults and transition faults for March C- are given in Table 4.1; see Table 3.1 and Section 3.4.1 for the definition of the fault models. Here, Read i = 0 (1) signifies that the i^{th} read operation of the test algorithm has returned a correct (faulty) value. For example, a state-0 fault (SF_0) is detected by all r1 operations. Consequently, the march syndrome for SF_0 is (01010). More fault signatures can be derived for all the fault models covered by March C-. By comparing these signatures with the obtained signatures during the test, one can identify the fault type causing the memory to fail.

Table 4.1: Fault signatures for March C- algorithm

Fault Model	Read 0	Read 1	Read 2	Read 3	Read 4
SF_0	0	1	0	1	0
SF_1	1	0	1	0	1
TF_0	0	1	0	1	0
TF_1	0	0	1	0	1

Yarmolik et al. presented a very basic diagnostic test to distinguish Coupling Faults (CFs) occurring in the memory cell array [58]. It is one of the first attempts to develop

a signature-based diagnostic test. It modifies the already available march tests (March E and March C). Each of the targeted faults is detected in a separate read operation of the test. A similar method based on combinations of fault decomposition and output tracing of the memory outputs is proposed by Niggemeyer and Rudnickin [45]. The idea is to keep track of the pass/fail information of every read operation in the diagnostic test, thereby generating a signature for each fault. But these tests are designed to diagnose a limited number of memory faults. Hierarchical procedure for fault diagnosis targeting memory cell array faults is proposed in [15]. First, a march algorithm is used for fault detection and classification; then more march algorithms are introduced to divide the fault groups with the same syndrome into smaller groups. On similar foundations, a two-phase diagnostic procedure to distinguish different CFs in SRAMs is proposed by Vardanian et al. [57, 8, 33]. Lately, a novel diagnosis approach based on the concept of Test Primitives is proposed [2]. The strength of this method lies in the fact that it is extensible and implementation independent. It requires only the pass/fail status of the applied tests instead of keeping track of all failed read operations.

Recently, dynamic faults in the memory are also being considered and diagnosis approaches are being extended to cover dynamic faults as well. Harutunyan et al. proposed a march-based fault location and a full diagnosis algorithm which uses signatures for diagnosis of dynamic faults in bit-oriented SRAMs [28]. A march algorithm is defined for the detection of all unlinked dynamic faults and for their partial diagnosis. Further march algorithms are used to locate the aggressor bits of the CFs. More march algorithms are proposed for the full diagnosis of dynamic faults. The target faults include all two-operation single-cell dynamic faults, as well as a subclass of two-operation two-cell dynamic faults where both of the sensitizing operations are applied either to the aggressor cell or to the victim cell.

The drawback of signature-based approaches is that such signatures are hardwired to a predefined diagnostic test. Consequently, if a memory is affected by a fault that is not considered in the fault dictionary, the diagnosis phase fails to provide any result or may even provide a wrong response. Any modification to the set of targeted faults needs a new diagnostic test with a new set of fault signatures. In order to increase the signature fields and therefore improve the diagnosis capability, the tests considered need to be extended by adding extra read operations. Such increased complexity of march tests can be excessive if used for industrial purposes. Another drawback of the signature approach is that it assumes knowledge of the pass/fail status of every read operation for a diagnostic march test. This is not generally possible on every memory test platform [2]. The scope of the existing approaches is also limited as most of the existing signature-based solutions target only the diagnosis of static faults. Moreover, these solutions are generally unable to distinguish between all faults (or all fault models) and so, make it impossible to determine which memory component is defective. It should be noted that a fault can occur in any part of the memory system (address decoders, core-cells, sense amplifiers, write drivers). For example, a signature-based diagnosis approach would indicate that the memory is affected by a transition fault but without providing any information on the faulty block of the memory where the malfunction is actually caused. This fault might be caused by a defect in the memory cell or a defect in the write driver. Such information is useful for the yield ramp up as well as for guiding the repair schemes.

Identifying which block of the memory (the memory cell array, the address decoder, the peripheral circuitry) is defective leads to considerable time-saving during the ramp up phase. An irredundant march test for the diagnosis of all distinguishable simple static faults in the memory cell array is proposed to distinguish static address decoder faults from static single cell and two-cell memory cell array faults [4]. But it only covers static faults in the address decoder and the memory cell array. As it is a fault signature-based approach, it cannot be justified as an efficient approach.

Another signature-based approach which locates faulty memory block and is useful for both static and dynamic faults is proposed by A. Ney et al. [41]. It consists of the extension of the fault syndrome and it operates without modifying the test algorithm. This extension is made by using information taken from the march test execution and from the memory structure itself. Although this work improves the diagnosis capability without escalating test complexity, it has the same limitations as all the signature-based approaches proposed so far.

As an alternative to the traditional signature-based approaches, an effect-cause paradigm-based diagnosis approach is presented in [42]. It consists of creating a database containing the history of operations (read and write) performed on those memory cells where read operations returned faulty logic values during the test phase. This information is used to generate the set of possible fault primitives representing the suspected fault models and to further determine the root cause of the observed faulty behavior. Such a history-based diagnosis approach does not rely on an established fault dictionary. This method is able to perform the diagnosis of both static and dynamic faults and provides better resolution as compared to other signature-based diagnosis approaches. From our viewpoint, history-based diagnosis is so far the best way available for locating the faulty block of memory. The drawback of this approach lies in the fact that the diagnosis tool is to be modified with respect to the faults being targeted. Adding a new fault to the list means that the whole procedure/tool has to be updated. In addition, the approach does not always succeed to provide the exact fault type and the exact faulty component of the memory system. Moreover, just like the signature-based approach, the history-based diagnosis approach assumes that the pass/fail information of all read operations is available.

4.2.3 Design for Diagnosis

This is an altogether different class of methods which insert extra hardware into the memory system for the diagnosis purposes. This extra hardware is termed Design for Diagnosis (DfD) hardware. This technique is based on the philosophy of not just identifying the fault but also determining the faulty block of the memory [43, 44]. It has been shown that a fault model may be related to more than one defect in the various blocks of the memory. Thus, it is not sufficient to just identify the type of fault but the faulty block also has to be identified. The novelty of this technique is that unlike other diagnostic techniques, it does not depend on the pass/fail status of applied diagnostic tests. Instead, the DfD hardware is monitored regularly and an alarm is raised if any anomaly is found. Diagnostic tests are applied for the purpose of imposing maximum stress on the targeted memory block, which sensitize the fault and, in turn, detect it by

the DfD hardware.

A. Ney et al. made an attempt in this direction and proposed a low cost DfD solution for identifying faulty write drivers [43]. It consists of verifying logic and analog conditions that guarantee the fault-free behavior of the write driver. The logic condition is verified by comparing the bit-line logic levels with the data to be written. The analog condition consists of verifying whether the write driver delivers enough current to the bit-lines. The hardware implementation of such a principle is composed of two parts; the analog structure and the data processing providing the diagnosis result. It is achieved by using several additional transistors and gates. The proposed solution allows a fast diagnosis using only three consecutive write operations to fully diagnose the write driver and induces a low area overhead (about 0.5% for a 512x512 SRAM) [43].

The proposed DfD in [43] targets only faults in write drivers. In fact, a fault can occur in any part of the memory system. Adding hardware solutions to target all static and dynamic faults in each component of the memory system would surely lead to a huge area overhead and to a surge in power consumption. Moreover, it does not provide any insight into the nature of the fault in the faulty block.

4.3 The Need for a New Approach

As silicon technology has entered the nano-era, diagnosis challenges have become more complex. Today, embedded memories are increasingly identified as having potential for introducing new yield loss mechanisms at a rate, magnitude, and complexity large enough to demand major changes in fault diagnosis schemes [39, 19]. In particular, emerging unconventional faults (such as timing related or complex faults) that originate in the highest density areas of semiconductor designs require new methods to diagnose such faults affecting not only large groups of memory cells but also other parts of the memory system as well such as address decoders. Traditional methods deal only with conventional faults within only the memory cell array part of the memory system.

The published work on memory diagnosis schemes suffer from the following shortcomings:

- They are all hardwired and are designed to diagnose a limited number of conventional memory faults related to only memory cell array; any modification of a set of targeted faults needs a new diagnosis test along with the new set of fault signatures.
- They do not target unconventional faults that emerge in the nano-era (for e.g., eak faults, timing related faults). In addition, they do not deal with faults in all parts of the memory system (for e.g., address decoders, sense amplifiers, etc).
- The existing approaches are unable to determine the faulty memory block and this leads to prolonged diagnosis periods and NTFs.
- Most of techniques are off-line reasoning procedures gearing toward an efficient fault diagnosis. In order to diagnose the memory against unconventional faults, it must be tested at-speed, hence on-line.

- The prevalent signature-based approaches assumes knowledge of the pass/fail status of every read operation for a diagnostic march test. This is not generally possible on every memory test platform.

It can be concluded from the above-mentioned approaches and their drawbacks that existing approaches are not adequate for covering the entire memory fault scope. Clearly, there is a need to bring in changes to the fundamental principles of memory diagnosis approaches. In the absence of new diagnosis methods being able to accurately deal with upcoming failure mechanisms in future technologies, it will be impossible to manufacture such technologies. A new approach is required, which can contribute to the resolution of failure analysis in the highly complex systems of future. With the downscaling of the technology, the demand of the hour is for an efficient approach which builds on the drawbacks of the traditional approaches. The challenge is to come up with an ideal diagnosis solution that:

- Targets faults in all parts of a memory system.
- Can accurately locate the faulty block and determine the fault type.
- Has high fault coverage, i.e., covers all static and dynamic faults.
- Is platform independent i.e., diagnosis scheme does not require any specific implementation techniques or technologies, other than running a set of algorithms..
- Is extensible i.e., new diagnosis capabilities for any new faults can be added easily without the need to modify anything in the method.

5 Hierarchical Memory Diagnosis

This chapter introduces a new memory diagnosis approach referred to as Hierarchical Memory Diagnosis. The underlying concepts of the approach will be explained and the advantages will be discussed. The approach will be applied to the diagnosis of both static and dynamic faults in different parts of a memory system (the memory cell array, the address decoder and the peripheral circuitry).

This chapter is organized as follows. Section 5.1 gives an introduction to Hierarchical Memory Diagnosis approach. Section 5.2 a gives a brief description of notation for test algorithms and stresses which will be used through out this thesis. Section 5.3 explains the concept and application of test primitives and test classes. Section 5.4 discusses the diagnosis procedure for static faults while Section 5.5 discusses the diagnosis procedure for dynamic faults in all parts of the memory system. The outcomes and conclusions are summarized in Section 5.6.

5.1 Introduction to Hierarchical Memory Diagnosis

In the previous chapters, the drawbacks of the traditional memory diagnosis approaches were established. There is certainly a need for a new approach which targets faults, not only in the memory cell array, but in all memory blocks and with a higher resolution. We propose a new approach to perform memory diagnosis; it is referred to as Hierarchical Memory Diagnosis (HMD).

The concept behind the HMD approach is to diagnose memory faults and memory blocks in an hierarchical order. Figure 5.1 gives an insight into intended application of the proposed approach. Given a faulty memory, first of all it is subjected to the diagnosis for static faults. If a static fault is present, it is mapped to an already defined functional fault model. Otherwise, if no fault is found, the faulty memory is subjected to the diagnosis for dynamic faults. If a dynamic fault is present, it is mapped to a fault model; otherwise the faulty memory is categorized as No Trouble Found.

HMD aims to narrow down the possible faulty component from the beginning and diagnostic tests can be applied accordingly. This helps in putting the effort into fault diagnosis in a well-directed way. This, in turn, cuts down the time required for the diagnosis and ensures fast yield ramp up. Figure 5.2 shows a high level overview of the proposed methodology. First of all, given a faulty memory, HMD finds out which memory block is faulty: the memory cell array, the address decoder or the peripheral circuitry. After identifying the faulty block, at the next level down the hierarchy, diagnostic tests designed for that particular faulty block are applied. Depending on the resolution of applied tests, the fault type and the location can be determined. Let us assume that we have a faulty memory with a fault in the address decoder. The first diagnosis level determines that the address decoder block is faulty. Then, at the next level down the

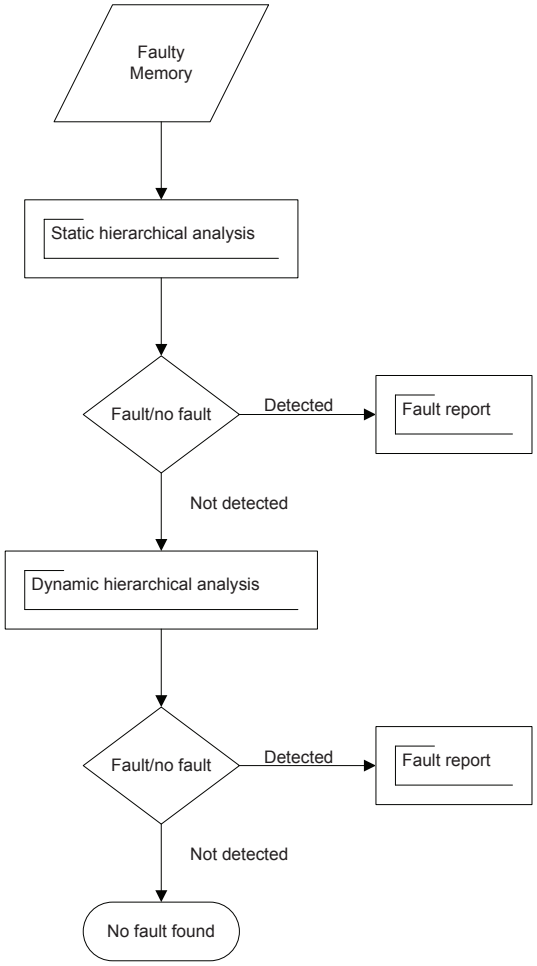

Figure 5.1: Procedure of Hierarchical Memory Diagnosis

hierarchy, it can be determined whether the fault is a row decoder fault or a column decoder fault. Further down the hierarchy, the fault type can be determined.

The most common drawback attached to the traditional approaches is that they assume the knowledge of the pass/fail status for each read operation in the applied diagnostic test while this is not practical for plenty of test platforms. This is because normally a test machine will stop running when it encounters the first read operation failure. To eliminate this drawback, HMD is built on the concept of Test Primitive (TP) [2]. A TP is a march test designed with the specific purpose of detecting a particular Fault Primitive (FP). Developing on the key concept of TP, a modified version of TP referred to as Test Class (TC) is introduced. A Test Class is a diagnostic test designed with the specific purpose of detecting a particular set of faults or a single Fault Class (FC). The implementation details of the HMD approach, based on TP and TC; will be discussed later in this chapter.

The major strengths of the HMD approach are:

- Targets static and dynamic faults in all the memory blocks, namely, the memory

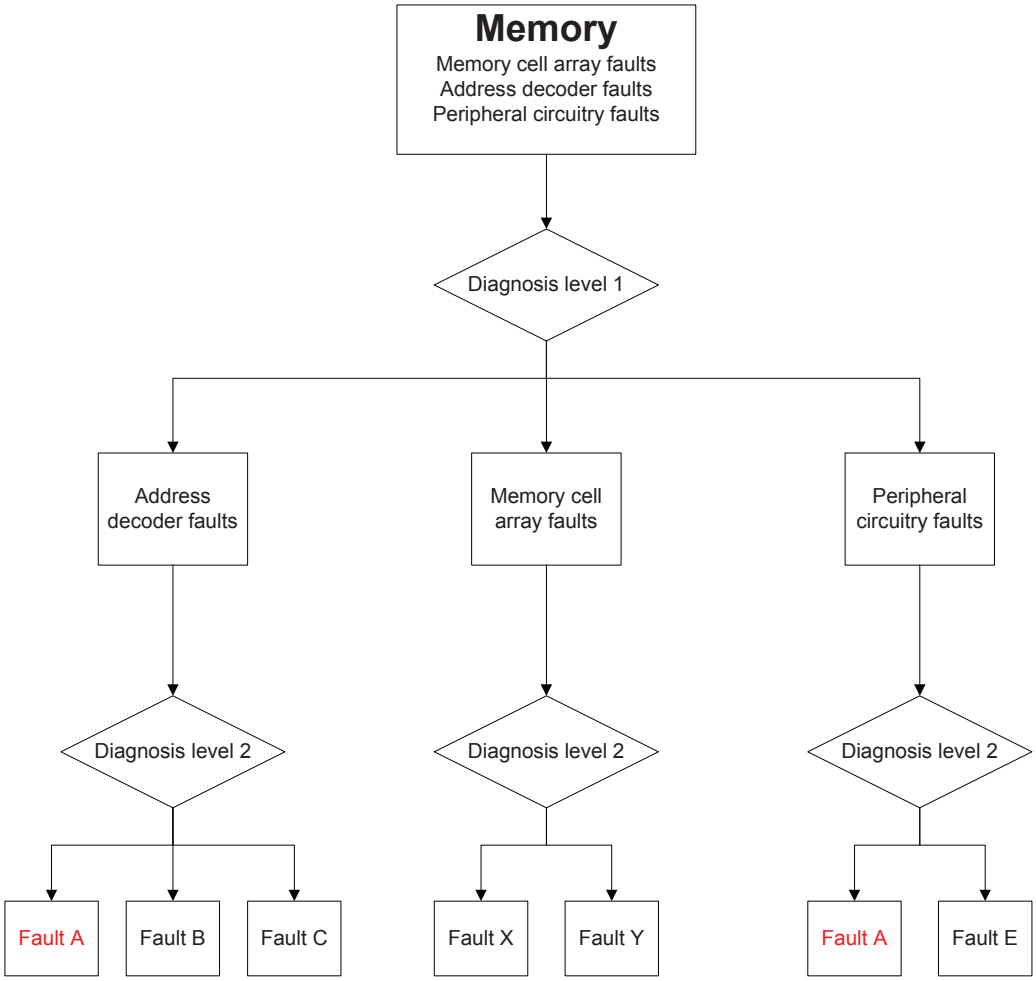

Figure 5.2: High level overview of Hierarchical Memory Diagnosis approach

cell array, the address decoder and the peripheral circuitry.

- No specific implementation requirements other than applying a test and determining the pass/fail status of the diagnostic test. Sometimes requires information on the address of first fail of the applied test.
- Accurate information on faulty block.
- Hierarchical methodology significantly reduces diagnosis effort. For example, there is no need to apply diagnostic tests for coupling faults in the memory cell array when a fault exists in the address decoder. The faulty memory block is identified at the first step of the diagnosis.
- New diagnosis capabilities for new FPs can be added without the need to modify existing tests or diagnostic signatures.
- Improved diagnostic ratio without escalating test complexity.

- Assists in test program optimization.

5.2 Notation of test algorithms and stresses

A test consists of a Base Test 'BT', applied using a particular Stress Combination 'SC'. A BT forms the main test algorithm, whereas an SC consists of a combination of values for the different stresses; for e.g., VDD = 1.8V, Temp = 70C, etc. Most BTs have a march test format. March test algorithms are the most common algorithms used for testing memories [54]. A *march test* consists of a finite sequence of march elements. A *March Element (ME)* is a finite sequence of operations applied to every cell in the memory before proceeding to the next cell. A complete march test is delimited by the '{...}' bracket pair; while a march element is delimited by the '(...)' bracket pair. The march elements are separated by semicolons, and the operations within a march element are separated by commas. The way one proceeds to the next cell is determined by the *Address Orders (AOs)* which can be an increasing address order (for e.g., increasing AO from the cell 0 to the cell $n-1$), denoted by $\Uparrow$ symbol, or a decreasing AO, denoted by $\Downarrow$ symbol, and which is the exact inverse of the $\Uparrow$ AO. When the AO is irrelevant, the symbol $\Updownarrow$ (i.e., $\Uparrow$ or $\Downarrow$) will be used. An example of a march algorithm is MATS+ [40], defined as:

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

MATS+ consists of three MEs, which are separated by the ';' symbol. The ME '$\Uparrow (r0, w1)$' specifies the $\Uparrow$ AO, while to each address a read operation with expected logic value '0' will be applied, after which a logic '1' will be written.

A BT can be applied with different stresses. The stresses can be divided into two types i.e., *algorithmic* and *non algorithmic* stresses [56].

- A *non-algorithmic* stresses is also referred as an environmental stress, as it specifies the environmental values, such as the supply voltage, the temperature, the timing (the clock frequency), etc.; they are effective during the application of the test [56].
- An *algorithmic* stress specifies the way the test is performed, and therefore it influences the sequence and/or the type of the memory operations performed. The most known algorithm stresses are the *address direction*, *counting method* and *data-background* [56] [22].

Next, the three different algorithmic stresses will be discussed.

Address direction is the addressing extension of the one-dimensional 'AO' to the two dimensional space of the memory cell array [56]. A real memory consists of a number or rows and columns (and thus also of a number of diagonals). The address directions specifies the direction (i.e., rows, columns, or diagonals) in which the address sequence has to be performed. The commonly used address directions in the industry are:

- **Fast row**: In fast row addressing, each address increment or decrement operation causes an adjacent physical row to be accessed.
- **Fast column**: In fast column addressing, each address increment or decrement operation causes an adjacent physical column to be accessed.
- **Fast diagonal**: In fast diagonal addressing, each address increment or decrement operation causes an adjacent physical diagonal to be accessed. Fast diagonal addressing is used less frequently in industry [56].

Counting method determines the address sequence. It has been shown that the counting method is important for detecting AD delay faults. Some of the counting methods used are linear, Address Complement, 2^i, H1 addressing etc [22].

- **Linear counting method**: This is the most common counting method used. It is denoted by the superscript 'L of the AO (for e.g., $^{L}\Uparrow$), where 'L' specifies the address sequence 0,1,2,3, etc. Because it is the default counting method, the superscript 'L is usually deleted.
- **Address Complement counting method**: It specifies an address sequence: 000, **111**, 001, **110**, 010, **101**, 011, and **100**; each bold address is the 1s complement of the preceding address.
- **2^i counting method**: This counting method is typically used by the MOVI algorithm It repeats the PMOVI algorithm 'N' times (N = is the number of memory address bits) with an address increment/decrement value of 2^i; with $0 \leq i \leq N-1$.
- **H1 counting method**: It specifies an address sequence: 000, 001, 000, 010, 000, 100, 000; each address has a hamming distance of '1' with respect to the preceding address.

Data Background 'DB' is the pattern of zeros and ones as seen in an array of memory cells [56]. The most common types of DBs are:

- **Solid (sDB)**: (0000.../0000...) or (1111.../1111...)
- **Checkerboard (cDB)**: (0101.../1010.../0101.../1010...)
- **Column Stripe (cDB)**: (0101.../0101.../0101.../0101..)
- **Row Stripe (rDB)**: (0000.../1111.../0000.../1111..)

5.3 Concept of Test Class and Test Primitive

For the HMD approach, we make use of Test Classes (TCs) and test Primitives (TPs). A TC is defined as a march test designed with the specific purpose of detecting a particular Fault Class (FC) while maintaining the minimum number of operations required to detect faults from the the targeted fault class. For diagnosis purpose, We require only the pass/fail status of applied TCs.

TCs are used at higher diagnosis levels to distinguish between FCs i.e., identifying which block of the memory is faulty. For example, if we target the distinction of memory cell array faults and address decoder faults, a single TC may detect all memory cell array faults and no address decoder faults while another TC may detect all address decoder faults and no memory cell array faults. FCs can then be classified using the TC signatures by comparing the pass/fail information of TCs to an already developed TC diagnostic dictionary. It is worth noting that often a TC developed for a particular FC will also detect faults in other FCs. This happens when the detection conditions of faults from different FCs are same and faults get detected by the same operation sequence. For example, a test developed to detect memory cell array faults will also detect some peripheral circuitry faults. This makes it difficult to distinguish FCs. Often, to insulate the detection of several faults, additional read/write operations are added to diagnostic tests. This leads to an increase in test length but it improves the diagnosis capability and resolution.

The use of TPs becomes more significant at lower levels of hierarchy where it is required to identify the type of the fault. A TP is a march test designed with the specific purpose of detecting a particular Fault Primitive (FP) while maintaining the minimum number of operations required to detect the targeted fault. The condition of targeting a single FP is useful for the diagnosis purpose as it helps avoiding the detection of any fault other than the targeted one. It is also important for optimizing the test length and thus, reducing test cost. Faults can be classified using the TP signatures by comparing the pass/fail information of TPs to an already developed TP diagnostic dictionary. TPs have the same notation as of a march test.

HMD constructs over the basic idea of TCs and TPs due to the following advantages they offer over other fault detection and diagnosis methodologies:

- **Extensibility**: Support for new FPs can be added without modifying the existing diagnosis scheme or diagnostic dictionary.

- **Platform Independence**: TCs and TPs have the same notation as of march tests and can be applied in the same regular way as any fault detecting test is applied. No extra memory is required to store specific read operations failing or the test signatures as the diagnostic process requires only the pass/fail information of applied tests. Thus, HMD can be used on any memory test platform irrespective of its capabilities.

- **Unknown Fault Identification**: There can be several combinations of failing TCs and TPs that do not attribute to any described fault. Thus, making it possible to define a fault and further develop/modify tests to detect the fault. This is a useful contribution to theoretical understanding of new faults.

In the rest of this section, it will be shown how the concepts of TC and TP are used to generate appropriate diagnosis dictionaries in order to identify different fault classes and fault primitives, respectively.

5.3.1 Test Class Dictionary Generation

For the classification of the targeted fault classes, it is required to develop a TC diagnostic dictionary. The procedure for it is described as follows:

- Step 1: Develop TCs for targeted FCs.
- Step 2: Create FC x TC dictionary based on pass/fail information.

Develop TCs for targeted FCs

A TC can be developed by combining the detection conditions of all the targeted faults. It is required to generate TCs with march elements containing the minimum number of operations to sensitize and detect the targeted FPs. This is needed to prevent the TC from sensitizing any other non-targeted FPs. For example, the TC for detecting all single-cell static faults in the memory cell array can be:

- **TCx**: $\{ \Uparrow (w0); \Uparrow (w1); \Uparrow (w1); \Uparrow (r1); \Uparrow (r1); \Uparrow (w0); \Uparrow (w0); \Uparrow (r0); \Uparrow (r0) \}$ or
- **TCy**: $\{ \Uparrow (w0); \Uparrow (w1, w1, r1, r1); \Uparrow (w0, w0, r0, r0) \}$

However, TCy is not preferred since it may also detect some dynamic faults like dynamic write destructive fault $< 1; 1w0w0/1/- >$ as well as some static coupling faults. Maintaining minimum test length is easier for single-cell faults, but for coupling faults, meeting minimum test length criteria is not so direct and needs systematic analysis. Thus, for this work, priority has been given to establish a proper diagnostic process and if required, neglecting the minimum test length requirement.

Next, an example will be given to illustrate the use of TC dictionary. Obviously, developing TCs requires in depth understanding of memory fault models and test generation. Nevertheless, the example below is intended to only show how the dictionary can be used in order to identify the fault class. More details about the TC development is given in Section 5.4.2. Let us consider the following TCs:

- **TC1**: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (r1, r1, w0, w0); \Uparrow (r0, w1) \}$
- **TC2**: $\{ \Downarrow (w0); \Downarrow (r0, r0, w1, w1); \Downarrow (r1, r1, w0, w0); \Downarrow (r0, w1) \}$

The above TCs are adapted from March MSS [27]. TC1 has only up addressing, and is able to detect all static single-cell faults. It can only detect 50% of the two-cell coupling faults; the two-cell faults that will be detected here have victim cell with higher address than aggressor cell. On the other hand, TC2 uses only down addressing, and is able to detect all static single-cell faults. It can only detect 50% of the two-cell coupling faults. Note that only two-cell faults where the address of the victim cell is lower than the aggressor cell will be detected.

Table 5.1: FC x TC dictionary

FC	TC1	TC2
No fault	0	0
Static single-cell faults	1	1
Static two-cell coupling faults	0	1
	1	0

Create FC x TC dictionary based on pass/fail information

The next step is to incorporate signatures of all FCs with respect to all TCs into a table. Through unique signatures, different sets of FPs can be identified; i.e., distinguishing static single-cell faults from static two-cell coupling faults. Although it looks similar to the traditional signature-based diagnosis method, it is not the case. Traditional signature-based methods require the outcome of each read operation in a march test which is not practical. But for the TP based approach, we just need to know whether one TP fails or passes and move on to another TP, which is easily implementable for automatic test machines. We will produce FCs x TCs table following two steps:

- Compare all the developed TCs and if two or more TCs are same, include only one of them. Thus, only unique TCs are included.
- Fill in the pass/fail information of TCs for the FPs, and then compare the generated signatures. If two targeted sets of FPs have the same signature, then add new TC(s) to distinguish them.

The process has to be repeated till all the fault classes have unique signatures or it is not possible to distinguish further. Table 5.1 gives the FCs x TCs dictionary. The table symbolizes fault detected as '1' and fault not detected as '0'. It is clear that based on the pass/fail information of each TC we can identify the appropriate fault class. If none of the TCs fail, then no single-cell nor two-cell coupling fault occurs. If only one TC fails, then a two-cell coupling fault is present in the memory. Finally, if both TCs fail, then a single-cell fault occurs. It is to be noted, that signatures for single-cell faults and two-cell coupling faults are unique and thus, fault classes can be distinguished.

It is worth noting that we only care about the pass/fail information of the TC, and that we assume the presence of a single fault at a time in the memory system.

5.3.2 Test Primitive Diagnostic Dictionary Generation

Once the fault class is identified, one can use the TP concept to further precisely pinpoint the fault. In a similar way as done for TC dictionary generation, the diagnosis process for FPs requires two steps:

- Step 1: Develop TPs for targeted FPs.
- Step 2: Create FP x TP dictionary based on pass/fail information.

Develop TPs for Targeted FPs

A TP can be developed for a FP by using the detection conditions of the target fault. Maintaining minimum length is important to avoid detection of any other non-targeted FPs. For example, the TP for FP $\mathrm{WDF}_0 = < 0w0/1/- >$ (see Table 3.1) can be:

- **TPx**: $\{\Updownarrow (w0); \Updownarrow (w0); \Updownarrow (r0)\}$ or
- **TPy**: $\{\Updownarrow (w0); \Updownarrow (w0, r0)\}$

The latter is not preferred since it can also detect the dynamic read destructive fault $< 0w0r0/1/1 >$.

Next, an example will be given to illustrate the use of TP dictionary. Developing TPs requires in depth understanding of memory fault models and test generation. Nevertheless, the example below is intended to only show how the dictionary can be used in order to identify the fault type. More details about the TC development is given in Section 5.4.2. Let us consider four TPs which can be used to distinguish static single-cell faults: TF_1, WDF_0, IRF_0, and SF_0 occurring in the memory cell array; (see Table 3.1).

- **TP1**: $\{\Updownarrow (w1); \Updownarrow (w0); \Updownarrow (r0)\}$ for $\mathrm{TF}_1 = < 1w0/1/- >$
- **TP2**: $\{\Updownarrow (w0); \Updownarrow (w0); \Updownarrow (r0)\}$ for $\mathrm{WDF}_0 = < 0w0/1/- >$
- **TP3**: $\{\Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1)\}$ for $\mathrm{IRF}_0 = < 0r0/1/- >$
- **TP4**: $\{\Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1)\}$ for $\mathrm{SF}_0 = < 0/1/- >$

The above TCs are adapted from [32]. Each TP targets a particular FP by including the detection conditions only for the targeted FP. This is essential to generate unique signatures for distinguishing the faults.

Create FP x TP dictionary based on pass/fail information

The next step is to incorporate signatures of all FPs with respect to all TPs into a table. Through unique signatures, different set of FPs can be identified. Next, We will produce FPs x TPs table following two steps:

- Compare all the developed TPs and if two or more TPs are same, include only one of them. Thus, only unique TPs are included. It is to be noted that TP3 for IRF_0 and TP4 for SF_0 are same so TP4 has been eliminated.
- Fill in the pass/fail information of TPs for the FPs, and then compare the generated signatures. If two targeted sets of FPs have the same signature, then add new TP(s) to distinguish them.

The process has to be repeated till all FPs have unique signatures or it is not possible to distinguish them further. Table 5.2 gives the FPs x TPs dictionary. It is clear that based on the pass/fail information of each TP we can identify the appropriate fault type. If none of the TPs fail, then none of the targeted FPs occur. If only TP1 fails,

Table 5.2: FP x TP dictionary

FP	TP1	TP2	TP3
No Fault	0	0	0
TF_1	1	0	0
WDF_0	0	1	0
IRF_0	1	1	1
SF_0	1	1	1
Not Applicable	0	0	1
	0	1	1
	1	0	1
	1	1	0

then the fault TF_1 is present in the memory. If only TP2 fails, then the fault WDF_0 is present in the memory. If all the TPs fail, then either IRF_0 or SF_0. These faults are externally indistinguishable and thus cannot be distinguished even by including more TPs. Signatures for other faults are unique and thus, those faults can be distinguished. The other signatures (for e.g., 110,101, etc.) do not attribute to any of the targeted FPs and can be used to define new faults.

5.4 Static Hierarchical Analysis

This section describes the HMD approach to diagnose static faults in the memory. We assume the presence of only a single defect at a time causing a static fault in the memory. The targeted faults are memory cell array faults, address decoder faults and peripheral circuitry faults. Detailed description of these faults can be found in Section 3.4. In this section, the term "memory cell array faults" refers to static faults in the memory cell array, "peripheral circuitry faults" refers to static faults in the peripheral circuitry and "address decoder faults" refers to static faults in the address decoder. The diagnosis procedure consists of two levels and follows a hierarchical order. To simplify the explanation of diagnostic tests and methodology, it is assumed that every column of the memory cell array has its own set of peripheral circuitry (sense amplifier, write drivers, precharge circuits).

In the rest of this section, first an overall overview of the different diagnosis levels will be presented. Then, the first level of diagnosis distinguishing the different fault classes (related to different memory blocks) will be addressed; this includes the differentiation of memory cell array, address decoder and peripheral circuitry faults. Third, the second level of hierarchy will be explained in order to be able to identify the exact fault type.

It is worth noting that for diagnosis of static faults, the target fault space is explained in Section 3.4.

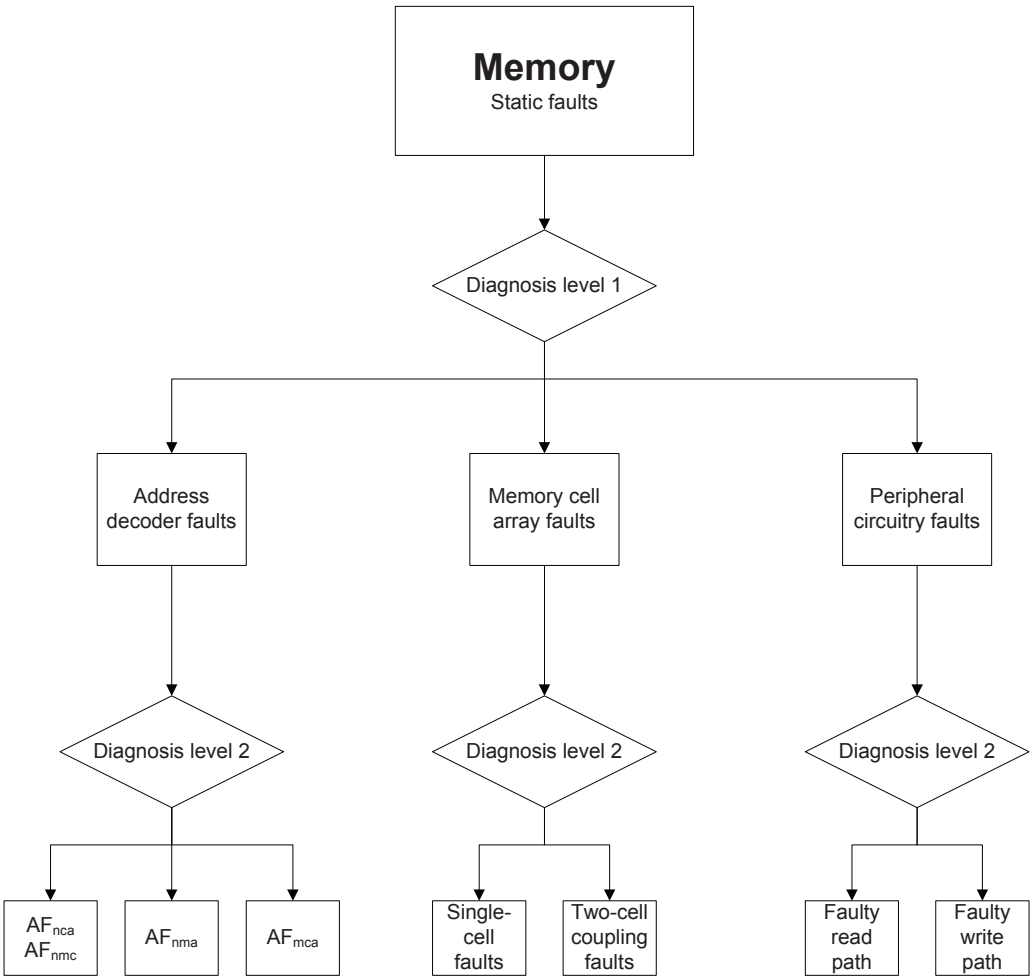

Figure 5.3: Diagnostic levels for static faults in SRAM

5.4.1 Diagnostic Levels

One of the benefits of the HMD approach lies in applying first only the minimum required effort by determining the faulty block at the first step itself. In this way, later diagnostic procedures will have to be applied only to identify faults in faulty block instead of applying diagnostic tests for all the memory components. Figure 5.3 provides a high level overview of the different diagnostic levels considered for static hierarchical analysis. We consider the reduced memory functional model of the SRAM consisting of 3 major subsystems: (1) Memory Cell Array, (2) Peripheral Circuitry, (3) Address Decoder. A detailed description of these subsystems can be found in Section 3.1. During the diagnosis procedure, first of all the faulty memory block is determined, which informs whether the memory cell array, the address decoder or the peripheral circuitry is defective. Further down the hierarchy, a new set of diagnostic tests is applied to diagnose the fault in the faulty memory block. Level 2 of HMD provides details like fault type and location.

5.4.2 Level 1: Diagnosing the Fault Class

The aim of level 1 is to determine which block of the memory is faulty, i.e., either the address decoder, the peripheral circuitry or the memory cell array is defective. It is to be achieved by means of TCs. As it is difficult to maintain the minimum test length for such a large scope of faults, establishing a proof of concept is prioritized. Still, significant effort has been put to minimize the number of operations and to use already established minimum length tests wherever possible.

As explained in Section 5.3, diagnosing the fault class is a two step process:

- Step 1: Develop TCs for targeted FCs.
- Step 2: Create FC x TC dictionary.

Next, these two steps are analyzed in detail.

Step 1: Test Classes

The basis for developing TCs that detect only a certain set of faults is to include the detection condition for only the targeted faults. At level 1, it is required to distinguish between the peripheral circuitry, the memory cell array and the address decoder fault classes. The problem arises when faults from two different fault classes have the same detection conditions. For example, all peripheral circuitry faults are detected by any test detecting all static faults in the memory cell array [21]. Also any test detecting *all* memory cell array faults may also detect address decoder faults. Similar is the case for address decoder faults and peripheral circuitry faults. The peripheral circuitry faults may lead to stuck-at faults in the associated memory cell(s) and thus, demonstrate similar behavior as stuck-at address decoder faults (AF_{nca} and AF_{nmc}).

As the detection conditions are same for the fault classes, distinguishing them using algorithms/tests depending only on the pass/fail information of read operations is not possible. Thus, there arises a need to introduce a new parameter. Tests provide many parameters that can be explored; for e.g., data background, address orders, address directions, counting methods, etc [54]. A possible way is to distinguish the faults by observing the $first\ failing\ address$ of the test. First failing address is the memory address (row address bits and column address bits) where first read operation failure is observed. It is important for the diagnosis process, as otherwise any test targeting one fault class also detects faults in another fault class, thus making it impossible to distinguish these fault classes using just the pass/fail information. We need to observe the address of only the first fail of a read operation. This can be easily implemented for the automatic test machine without any specific requirements. The information of first failing address, used along with different *addressing directions* can provide unique signatures for fault classes with same detection conditions.

Many memory test algorithms have been proposed [28, 45, 58]. Some of them target all static faults including memory cell array, address decoder and peripheral circuitry; Example are March SS [26] and its optimized version March MSS [27]. The description of March MSS is:

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

Using the test as it is will not help in diagnosis. In order to make it suitable, we make use of two important parameters already mentioned: (a) first failing address, and (b) address order. Therefore, we will split the test in two parts: (1) a part with an increasing address order, and (2) a part with a decreasing address order, and add a march element for each part; the latter is necessary to guarantee the detection of 50% two-cell coupling faults for each part. This results into:

- **March MSSm-up**: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (r1, r1, w0, w0); \Uparrow (r0, w1) \}$
- **March MSSm-down**: $\{ \Downarrow (w0); \Downarrow (r0, r0, w1, w1); \Downarrow (r1, r1, w0, w0); \Downarrow (r0, w1) \}$

Next, we will show that when we apply March MSSm-up and March MSSm-down, all static faults will be detected. This will be done for memory cell array, address decoder and peripheral circuitry faults.

Memory Cell Array Faults

It is easy to verify that *both* March MSSm-up and March MSSm-down detects all single-cell faults; see Table 3.1. Moreover, both tests will produce the *same address* in case of a failure. For two-cell faults, depending on the address of the aggressor cell (say a) and the address of the victim cell (say v), either March MSSm-up or March MSSm-down will detect the fault but *not* both of them. Table 5.3 gives the fault coverage of these two tests regarding two-cell coupling faults. In the table, a distinction is made between faults where $a > v$ and $a < v$. For example, $< 0; 0/1/- >_{av}$ means a coupling state fault where $a < v$, while $< 0; 0/1/- >_{va}$ means the same fault for $a > v$. Table 5.3 clearly shows that each test detects 50% of the two cell coupling faults. The results presented have also been verified using RASTA memory fault simulator [7].

Address Decoder Faults

It can be easily verified that the application of March MSSm-up and March MSSm-down will detect all address decoder faults as they satisfy the detection conditions for such faults, which are reported in [22, 54]. Nevertheless, we will explain how these faults will be detected.

Table 5.4 shows the fault coverage of March MSSm-up and March MSSm-down with respect to address decoder faults of Figure 5.4. The first column in the table gives the fault, and the second column the address (see also Figure 5.4). For example in the presence of AF_{nca}, irrespective of the behavior of the fault (for e.g., stuck-at-0 or stuck-at-1), both tests will fail at address A_x. In case of AF_{nmc}, the tests will fail at A_x but not at A_y, irrespective of the property of the fault. For fault AF_{nma}, depending upon the property (i.e., whether $A_x > A_y$ or $A_x < A_y$), March MSSm-up and March MSSm-down will fail at A_x or A_y. For example, when $A_x > A_y$, March MSSm-up fails for A_x while March MSSm-down fails for A_y. But both the tests fail at atleast one address. Similar to AF_{nmc}, in case of AF_{mca}, depending upon the property (i.e., whether $A_x > A_y$ or $A_x < A_y$), March MSSm-up and March MSSm-down will fail at A_x or A_y.

Table 5.3: Fault coverage of March MSSm-up and March MSSm-down for two-cell coupling faults

#	FP	Fault	MSSm-up	MSSm-down	#	FP	Fault	MSSm-up	MSSm-down
1	$<0;0/1/->_{av}$	CFst	0	1	19	$<1;r0/\uparrow/0>_{av}$	CFdrd	1	0
2	$<0;1/0/->_{av}$	CFst	1	0	20	$<1;r1/\downarrow/1>_{av}$	CFdrd	0	1
3	$<1;0/1/->_{av}$	CFst	1	0	21	$<0;r0/0/1>_{av}$	CFir	0	1
4	$<1;1/0/->_{av}$	CFst	0	1	22	$<0;r1/1/0>_{av}$	CFir	1	0
5	$<0;0w1/0/->_{av}$	CFtr	0	1	23	$<1;r0/0/1>_{av}$	CFir	1	0
6	$<0;1w0/1/->_{av}$	CFtr	1	0	24	$<1;r1/1/0>_{av}$	CFir	0	1
7	$<1;0w1/0/->_{av}$	CFtr	1	0	25	$<r0;0/\uparrow/->_{av}$	$CFds_{rx}$	1	0
8	$<1;1w0/1/->_{av}$	CFtr	0	1	26	$<r1;0/\uparrow/->_{av}$	$CFds_{rx}$	0	1
9	$<0;0w0/\uparrow/->_{av}$	CFwd	0	1	27	$<0w1;0/\uparrow/->_{av}$	$CFds_{xw\overline{x}}$	1	0
10	$<0;1w1/\downarrow/->_{av}$	CFwd	1	0	28	$<1w0;0/\uparrow/->_{av}$	$CFds_{xw\overline{x}}$	0	1
11	$<1;0w0/\uparrow/->_{av}$	CFwd	1	0	29	$<0w0;0/\uparrow/->_{av}$	$CFds_{xwx}$	0	1
12	$<1;1w1/\downarrow/->_{av}$	CFwd	0	1	30	$<1w1;0/\uparrow/->_{av}$	$CFds_{xwx}$	1	0
13	$<0;r0/\uparrow/1>_{av}$	CFrd	0	1	31	$<r0;1/\downarrow/->_{av}$	$CFds_{rx}$	0	1
14	$<0;r1/\downarrow/0>_{av}$	CFrd	1	0	32	$<r1;1/\downarrow/->_{av}$	$CFds_{rx}$	1	0
15	$<1;r0/\uparrow/1>_{av}$	CFrd	1	0	33	$<0w1;1/\downarrow/->_{av}$	$CFds_{xw\overline{x}}$	0	1
16	$<1;r1/\downarrow/0>_{av}$	CFrd	0	1	34	$<1w0;1/\downarrow/->_{av}$	$CFds_{xw\overline{x}}$	1	0
17	$<0;r0/\uparrow/0>_{av}$	CFdrd	0	1	35	$<0w0;1/\downarrow/->_{av}$	$CFds_{xwx}$	1	0
18	$<0;r1/\downarrow/1>_{av}$	CFdrd	1	0	36	$<1w1;1/\downarrow/->_{av}$	$CFds_{xwx}$	0	1
1	$<0;0/1/->_{va}$	CFst	1	0	19	$<1;r0/\uparrow/0>_{va}$	CFdrd	0	1
2	$<0;1/0/->_{va}$	CFst	0	1	20	$<1;r1/\downarrow/1>_{va}$	CFdrd	1	0
3	$<1;0/1/->_{va}$	CFst	0	1	21	$<0;r0/0/1>_{va}$	CFir	1	0
4	$<1;1/0/->_{va}$	CFst	1	0	22	$<0;r1/1/0>_{va}$	CFir	0	1
5	$<0;0w1/0/->_{va}$	CFtr	1	0	23	$<1;r0/0/1>_{va}$	CFir	0	1
6	$<0;1w0/1/->_{va}$	CFtr	0	1	24	$<1;r1/1/0>_{va}$	CFir	1	0
7	$<1;0w1/0/->_{va}$	CFtr	0	1	25	$<r0;0/\uparrow/->_{va}$	$CFds_{rx}$	0	1
8	$<1;1w0/1/->_{va}$	CFtr	1	0	26	$<r1;0/\uparrow/->_{va}$	$CFds_{rx}$	1	0
9	$<0;0w0/\uparrow/->_{va}$	CFwd	1	0	27	$<0w1;0/\uparrow/->_{va}$	$CFds_{xw\overline{x}}$	0	1
10	$<0;1w1/\downarrow/->_{va}$	CFwd	0	1	28	$<1w0;0/\uparrow/->_{va}$	$CFds_{xw\overline{x}}$	1	0
11	$<1;0w0/\uparrow/->_{va}$	CFwd	0	1	29	$<0w0;0/\uparrow/->_{va}$	$CFds_{xwx}$	1	0
12	$<1;1w1/\downarrow/->_{va}$	CFwd	1	0	30	$<1w1;0/\uparrow/->_{va}$	$CFds_{xwx}$	0	1
13	$<0;r0/\uparrow/1>_{va}$	CFrd	1	0	31	$<r0;1/\downarrow/->_{va}$	$CFds_{rx}$	1	0
14	$<0;r1/\downarrow/0>_{va}$	CFrd	0	1	32	$<r1;1/\downarrow/->_{va}$	$CFds_{rx}$	0	1
15	$<1;r0/\uparrow/1>_{va}$	CFrd	0	1	33	$<0w1;1/\downarrow/->_{va}$	$CFds_{xw\overline{x}}$	1	0
16	$<1;r1/\downarrow/0>_{va}$	CFrd	1	0	34	$<1w0;1/\downarrow/->_{va}$	$CFds_{xw\overline{x}}$	0	1
17	$<0;r0/\uparrow/0>_{va}$	CFdrd	1	0	35	$<0w0;1/\downarrow/->_{va}$	$CFds_{xwx}$	0	1
18	$<0;r1/\downarrow/1>_{va}$	CFdrd	0	1	36	$<1w1;1/\downarrow/->_{va}$	$CFds_{xwx}$	1	0

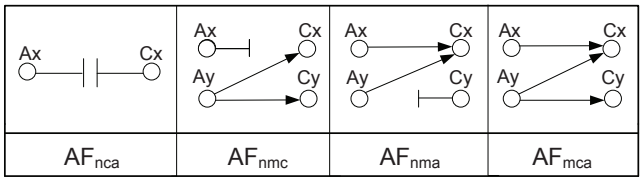

Figure 5.4: Static address decoder faults

Thus, both March MSSm-up and March MSSm-down will fail in the presence of address decoder faults. Also, since any fault in address decoder will effect the entire row/column, March MSSm-up and March MSSm-down will fail will produce *different* first failing address.

Table 5.4: Fault coverage of March MSSm-up and March MSSm-down for address decoder faults

#	Static ADF	Property	Address	MSSm-up	MSSm-down
1	AF_{nca}	Stuck-at-0 behavior	A_x	1	1
		Stuck-at-1 behavior	A_x	1	1
2	AF_{nmc}	Stuck-at-0 behavior	A_x	1	1
			A_y	0	0
		Stuck-at-1 behavior	A_x	1	1
			A_y	0	0
3	AF_{nma}	$A_x > A_y$	A_x	1	0
			A_y	0	1
		$A_x < A_y$	A_x	0	1
			A_y	1	0
4	AF_{mca}	$A_x > A_y$	A_x	1	0
		Address A_y reads logical AND/OR of values in C_x and C_y	A_y	0	1
		$A_x < A_y$	A_x	0	1
		Address A_y reads logical AND/OR of values in C_x and C_y	A_y	1	0

Peripheral Circuitry Faults

Peripheral circuitry faults will be detected by both TCs. Since a fault in peripheral circuitry (write driver, sense amplifier, etc.) will affect all operations through that circuit, the fault will be detected on accessing any cell in the column associated with that circuit. Hence, March MSSm-up and March MSSm-down will fail at *different* addresses.

Step 2: Diagnostic Dictionary

So far in this section, we have developed TCs for distinguishing the fault classes into memory cell array faults, address decoder faults and peripheral circuitry faults. The next step is to incorporate signatures of all FCs with respect to all TPs into a diagnostic dictionary. Table 5.5 presents the diagnostic signatures which use aforementioned TCs and DfD to distinguish the faulty memory block. The failing address considered is the memory address where first read operation failure is observed.

It can be seen from the table that the fault signature for address decoder FC and peripheral circuitry FC is same and thus, these two FCs cannot be distinguished using the developed TCs. Due to the overlapping detection conditions of both fault classes, they cannot be distinguished using different addressing sequences (fast-row, fast-column or diagonal addressing) or introducing new TCs. A possible way to distinguish these faults is by introducing some extra hardware (transistors) for the diagnosis purpose. Introduction of hardware for the purpose of the diagnosis is referred to as Design for Diagnosis (DfD).

For the diagnosis purpose, other than normal mode of operation of a memory, a diagnosis mode is introduced. The concept can be understood from Figure 5.5. In diagnosis mode, every column of the memory is addressed by the addressing sequence of the adjacent column i.e., column 1 of the memory is operated on when address bits

Table 5.5: Diagnostic dictionary

#	Faulty Block	MSSm-up in (failing address)	MSSm-down in (failing address)	Failing address
1	Address decoder	1 (A_α)	1 (A_β)	$\alpha \neq \beta$
2	Peripheral circuitry	1 (A_α)	1 (A_β)	$\alpha \neq \ \beta \neq \delta$
3	Memory cell array	1 (A_α)	1 (A_α)	$\alpha \neq \beta$
		1 (A_α)	0	$\alpha \neq \beta$
		0	1	

for column 2 are supplied and column 2 of the memory is operated on when address bits for column 1 are supplied. This can be realized using one multiplexer per memory cell array column and a control pin. It is to be noted that when the peripheral circuitry is faulty, and a diagnostic test is applied, the fault will be detected at different memory addresses in diagnosis mode and in normal mode. For example, let us assume fault in the peripheral circuitry for column 1 of the memory cell array. March MSSm can detect all the static faults in the peripheral circuitry and all the static faults in the address decoder as well. On application of March MSSm in both the normal and diagnosis mode, due to the faulty peripheral circuitry, March MSSm applied in normal mode will fail for a memory address in column 1 while March MSSm applied in diagnosis mode will fail for a memory address in column 2. But if the fault exists in the address decoder, the fault will be detected at the same memory addresses in diagnosis mode and in normal mode. Thus, address decoder faults and peripheral circuitry faults can be distinguished by applying March MSSm in normal mode and diagnosis mode.

Table 5.6 presents the diagnostic signatures which use aforementioned TCs and DfD to uniquely distinguish the faulty memory block. The failing address considered is the memory address where first read operation failure is observed. The total complexity for all the diagnostic tests at level 1 is 11*3*n = 33n.

5.4.3 Level 2: Diagnosing the Fault Type

Once the faulty block of the memory has been determined, it is required to identify the fault type. For this purpose, diagnostic tests specific to the faulty memory block are applied to the memory. As explained in Section 5.3, diagnosing the fault type is a two step process:

- Step 1: Develop TPs for targeted FCs.
- Step 2: Create FP x TP dictionary.

Next, these two steps are analyzed in detail with respect to different faulty blocks.

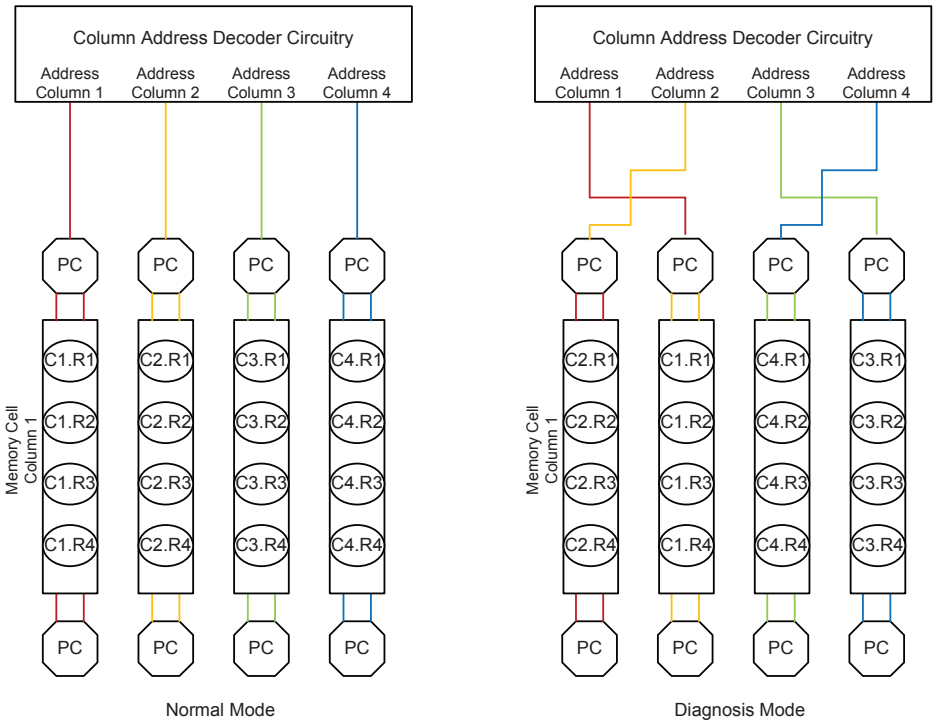

Figure 5.5: Design for diagnosis

Table 5.6: Diagnostic dictionary

#	Faulty Block	MSSm-up in normal mode (failing address)	MSSm-down in normal mode (failing address)	MSSm-up in diagnosis mode (failing address)	Failing address
1	Address decoder	1 (A_α)	1 (A_β)	1 (A_α)	$\alpha \neq \beta$
2	Peripheral circuitry	1 (A_α)	1 (A_β)	1 (A_γ)	$\alpha \neq \ \beta \neq \delta$
3	Memory cell array	1 (A_α)	1 (A_α)	1 (A_β)	$\alpha \neq \beta$
		1 (A_α)	0	1 (A_β)	$\alpha \neq \beta$
		0	1	0	

Faulty Memory Cell Array

Once the faulty block has been identified as the memory cell array, the fault can be classified as a single-cell or a two-cell coupling fault.

Table 5.7: Diagnostic dictionary for memory cell array faults

#	Fault	MSSm-up	MSSm-down
1	Single-cell fault	1	1
2	Two-cell coupling fault	1	0
		0	1

Step 1: Test Primitives

This can be achieved without applying any new tests. The pass/fail information of TCs applied at level 1 is sufficient for this classification. As stated before, the single-cell faults in the memory cell array are detected by both March MSSm-up and March MSSm-down. But the two-cell coupling faults in the memory cell array will be detected by either one of the two TCs. This is because March MSSm-up and March MSSm-down are the same diagnostic test applied in different address directions and depending on the address of aggressor and victim cell, a two-cell coupling fault will fail in one of these two TCs (see Table 5.3).

Step 2: Diagnostic Dictionary

As mentioned above, no new tests are applied to classify the memory cell array fault as single-cell or two-cell coupling fault. Table 5.7 uses the pass/fail status of TCs applied at level 1 and provides the diagnostic dictionary for classification of memory cell array faults.

Faulty Address Decoder

It is already known from level 1 that the fault lies in the address decoder. It is of interest to find out which of the four address decoder faults (AF_{nca}, AF_{nmc}, AF_{nma}, AF_{mca}) is present.

Step 1: Test Primitives

From the description of address decoder faults, it can be observed that fault AF_{nca} and AF_{nmc} cannot be distinguished from each other. This is because both the faults display the stuck-at-0 or stuck-at-1 behavior and have same detection conditions. To classify the distinguishable address decoder faults, we need to introduce new diagnostic tests. We propose four TPs for classifying all distinguishable address decoder faults:

- **March AFr0up**: $\{ \Updownarrow (w0); \Uparrow (r0, w1) \}$
- **March AFr0down**: $\{ \Updownarrow (w0); \Downarrow (r0, w1) \}$
- **March AFr1up**: $\{ \Updownarrow (w1); \Uparrow (r1, w0) \}$
- **March AFr1down**: $\{ \Updownarrow (w1); \Downarrow (r1, w0) \}$

Table 5.8: Fault coverage of March AFr0up, March AFr0down, March AFr1up and March AFr1down for address decoder faults

#	Static ADF	Property	Address	AFr0up	AFr0down	AFr1up	AFr1down
1	AF_{nca}	Stuck-at-0 behavior	A_x	0	0	1	1
		Stuck-at-1 behavior	A_x	1	1	0	0
2	AF_{nmc}	Stuck-at-0 behavior	A_x	0	0	1	1
			A_y	0	0	0	0
		Stuck-at-1 behavior	A_x	1	1	0	0
			A_y	0	0	0	0
3	AF_{nma}	$A_x > A_y$	A_x	1	0	1	0
			A_y	0	1	0	1
		$A_x < A_y$	A_x	0	1	0	1
			A_y	1	0	1	0
4	AF_{mca}	$A_x > A_y$ Address A_y reads logical AND of values in C_x and C_y	A_x	1	0	1	0
			A_y	0	0	0	1
		$A_x > A_y$ Address A_y reads logical OR of values in C_x and C_y	A_x	1	0	1	0
			A_y	0	1	0	0
		$A_x < A_y$ Address A_y reads logical AND of values in C_x and C_y	A_x	0	1	0	1
			A_y	0	0	1	0
		$A_x < A_y$ Address A_y reads logical OR of values in C_x and C_y	A_x	0	1	0	1
			A_y	1	0	0	0

Address decoder faults AF_{nca} and AF_{nmc} demonstrate either a stuck-at-0 behavior or stuck-at-1 behavior and will be detected by either a r0 operation or a r1 operation. Hence, these faults will be detected by both March AFr0up and March AFr0down or both March AFr1up and AFr1down. Address decoder fault AF_{nma} will be detected by all four TPs. AF_{mca} takes place when a single address accesses more than one cell. Detection of fault AF_{mca} depends on whether the read value obtained on accessing faulty address is the result of ANDing the contents of both cells or ORing them. It depends on the technology used to manufacture memory chips. Due to this behavior, depending on the technology, AF_{mca} will not be detected by one of the four TPs.

Table 5.8 shows the fault coverage of the above mentioned TPs for address decoder faults. The table uses the same notation as of Figure 5.4 to represent addresses of the memory cells.

Step 2: Diagnostic Dictionary

Previous sections provide details about the generation of adequate TPs and have demonstrated the capability of TPs to further classify address decoder faults in the memory. The next step is to create the diagnostic dictionary. Table 5.9 presents the TP signatures which use the above mentioned TPs to classify address decoder faults. The table uses the same notation as of Figure 5.4 to represent addresses of the memory cells. The total

Table 5.9: Diagnostic dictionary for address decoder faults

#	Static ADF	AFr0up	AFr0down	AFr1up	AFr1down
1	AF_{nca}	0	0	1	1
		1	1	0	0
2	AF_{nmc}	0	0	1	1
		1	1	0	0
3	AF_{nma}	1	1	1	1
4	AF_{mca}	1	0	1	1
		1	1	1	0
		0	1	1	1
		1	1	0	1

complexity for all the diagnostic tests is 12n.

The fault can also be classified as a row decoder or column decoder fault. It can be done using information about the pattern of failing addresses. If the failing addresses for different TPs fall on the beginning and ending addresses of a row, it can be classified as a row decoder fault and if on the ends of a column, as a column decoder fault.

Faulty Peripheral circuitry

Once the faulty block has been identified as the peripheral circuitry, the fault can be classified as a read path fault or a write path fault.

Step 1: Test Primitives

The classification of a peripheral circuitry fault in read path fault or write path fault cannot be made using only the pass/fail information of read operations as it cannot be identified whether the read operation sensitized the fault or the memory cell contained a wrong value due to a faulty write operation. To classify peripheral circuitry faults, DfD hardware is used.

In order to accommodate more diagnosis capabilities, extra diagnosis hardware is required. We need to introduce a new diagnosis mode here. The new diagnosis mode can be understood from Figure 5.6. As we need to find whether the read path is faulty or write path is faulty, in diagnosis mode, we alter the write path for every column of the memory cell array. In diagnosis mode, every column of the memory cell array is connected to the write circuitry of the adjacent column while the read circuitry remains unchanged i.e., the memory cells of column 1 of the memory cell array are connected to the write circuitry of column 2 and read circuitry of column 1 while the memory cells of column 2 are connected to the write circuitry of column 1 and read circuitry of column 2. This can be realized using two multiplexers per column of the memory cell array. It is to be noted that when write path circuitry is faulty and a diagnostic test is applied, the fault will be detected in different memory cells/addresses in diagnosis mode and in normal mode. But if the read path is faulty, and a diagnostic test is applied, the fault will be detected in the same memory cells/addresses in diagnosis mode and in normal mode.

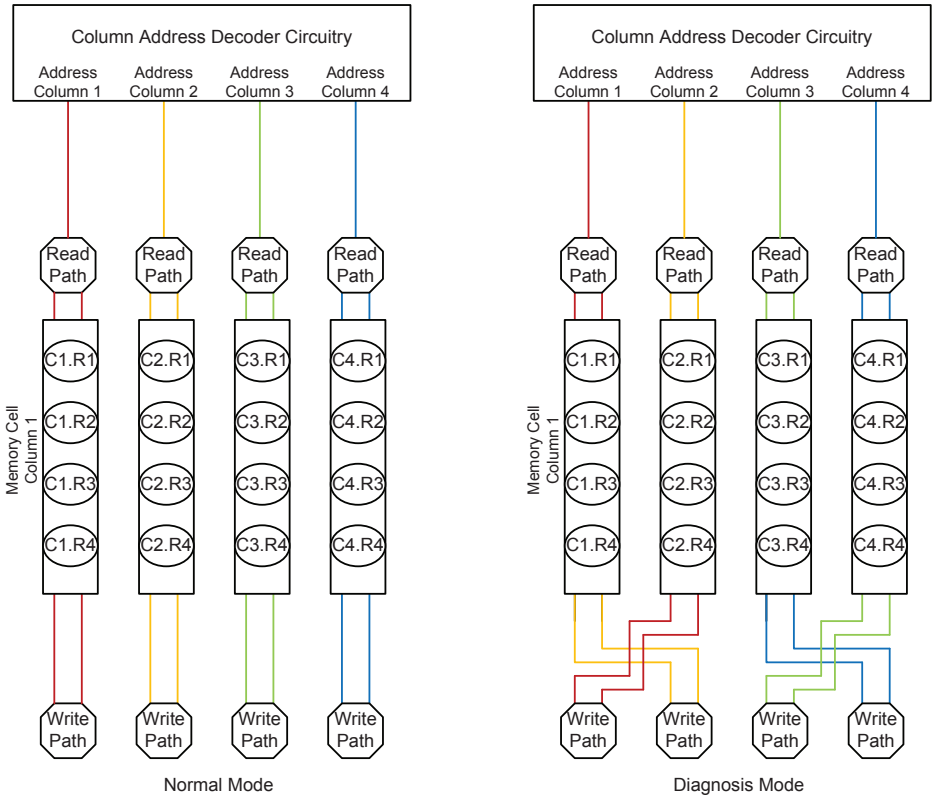

Figure 5.6: Design for diagnosis for faulty peripheral circuitry

For example, let us assume a faulty sense amplifier in column 1 of the memory cell array. We also assume that there exists a march test called March XYZ that can detect all the static faults in the peripheral circuitry. The automatic test machine applies March XYZ in normal mode and in diagnosis mode. March XYZ applied in diagnosis mode will fail for a cell in column 1 and March XYZ applied in normal mode will also fail for a cell in column 1. The reason is that diagnosis mode altered only the write path for column 1 while maintaining the same (and thus, faulty) read path. For classifying peripheral circuitry faults, a single test is to be applied in normal mode and in diagnosis mode. It is important to develop the test while keeping in attention the diagnosis hardware. This is because, in diagnosis mode, for a given column address, different memory cell array columns are written and read. For this purpose, only one read/write operation is kept in one march element. The TP used for diagnosing peripheral circuitry faults is well known Memory Scan test [1]:

Memory Scan test: $\{ \Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1); \Updownarrow (r1) \}$

Observing the test, it can be easily verified that Memory Scan test can detect all the stuck-at-faults faults in the peripheral circuitry.

Table 5.10: Diagnostic dictionary for peripheral circuitry faults

#	Faulty Block	Memory SCAN in normal mode (failing address)	Memory SCAN in diagnosis mode (failing address)	Failing address
1	Read Path circuitry	1 (A_α)	1 (A_α)	
2	Write Path circuitry	1 (A_α)	1 (A_β)	$\alpha \neq \beta$

Step 2: Diagnostic Dictionary

Previous sections provide details about the generation of TP and have demonstrated the capability of TP and DfD to further classify peripheral circuitry faults in the memory. The next step is to create the diagnostic dictionary. Table 5.10 presents the diagnostic dictionary for the faulty peripheral circuitry. Memory Scan test is applied in normal mode and diagnosis mode; hence the total complexity for all the diagnostic tests is 8n.

In this section, detailed analysis was presented for the diagnosis of static faults in the memory. The diagnosis approach has been based on the concepts of TCs, TPs and DfD. The outcome of the diagnosis contains details about the faulty block of the memory and the fault type. The failing addresses give some insight into possible fault location. The work can be extended to go further down the hierarchy to precisely pin-point the defect location. Also, with this proof of concept, more TCs and TPs can be added to include support for newer fault models.

The limitation here is that while targeting static faults, several dynamic faults can also get sensitized and detected. This might lead to wrong classification of a dynamic fault as a static fault. Care has been taken to detect as less dynamic faults as possible. For this purpose, all TCs and TPs are to be applied in fast-row direction to nullify the effect of read equivalent stress [14]. Also, the DfD hardware will lead to some extra cost and delay in the system but this is a considerably negligible cost compared to the benefits of the diagnosis. Moreover, these multiplexers can be removed from design when the characterization testing procedure is over and the memory is approved for mass production.

5.5 Dynamic Hierarchical Analysis

This section describes the HMD approach to diagnose dynamic faults in the memory. We assume the presence of only a single defect at a time causing a dynamic fault in the memory. The targeted faults include memory cell array faults, address decoder faults and peripheral circuitry faults. Detailed description of these faults can be found in Section 3.5. In this section, the terms “memory cell array faults” refer to dynamic faults in the memory cell array, “peripheral circuitry faults” refer to dynamic faults in the peripheral circuitry and “address decoder faults” refer to dynamic faults in the address decoder.

The diagnosis process consists of two levels and follows a hierarchical order.

In the rest of this section, first an overall overview of the different diagnosis levels will be presented. Then, the first level of diagnosis distinguishing the different fault classes (related to different memory blocks) will be addressed; this includes the differentiation of memory cell array, address decoder and peripheral circuitry faults. Third, the second level of hierarchy will be explained in order to be able to identify the exact fault type.

It is worth noting that for diagnosis of dynamic faults, the target fault space is explained in Section 3.5.

5.5.1 Diagnostic Levels

The benefits of following an hierarchical order for the diagnosis are already mentioned before. One of the advantages is applying only required effort by determining the faulty block at the first step itself. Figure 5.7 provides a high level overview of the different diagnostic levels considered for dynamic hierarchical analysis. We consider the reduced memory functional model of the SRAM consisting of 3 major subsystems: (1) Memory cell array, (2) Peripheral Circuitry, (3) Address Decoder. A detailed description of these subsystems can be found in Section 3.1. At the first level of the HMD approach, the faulty memory block is identified. As we move down in hierarchy, diagnostic tests are applied for detecting faults only in the identified block instead of applying diagnostic tests for all the memory components. Level 2 of HMD gives details like type and location of fault.

5.5.2 Level 1: Diagnosing the Fault Class

Similar to level 1 for static faults, the aim of level 1 for dynamic faults is to determine which block of the memory is faulty i.e., either the address decoder, the peripheral circuitry or the memory cell array is defective. As it is difficult to maintain the minimum test length for such a scope of faults, establishing a proof of concept is prioritized. Still, significant effort has been put to minimize the number of operations and to use already established minimum length tests wherever possible.

As explained in Section 5.3, diagnosing the fault class is a two step process:

- Step 1: Develop TCs for targeted FCs.
- Step 2: Create FC x TC dictionary.

Next, these two steps are analyzed in detail.

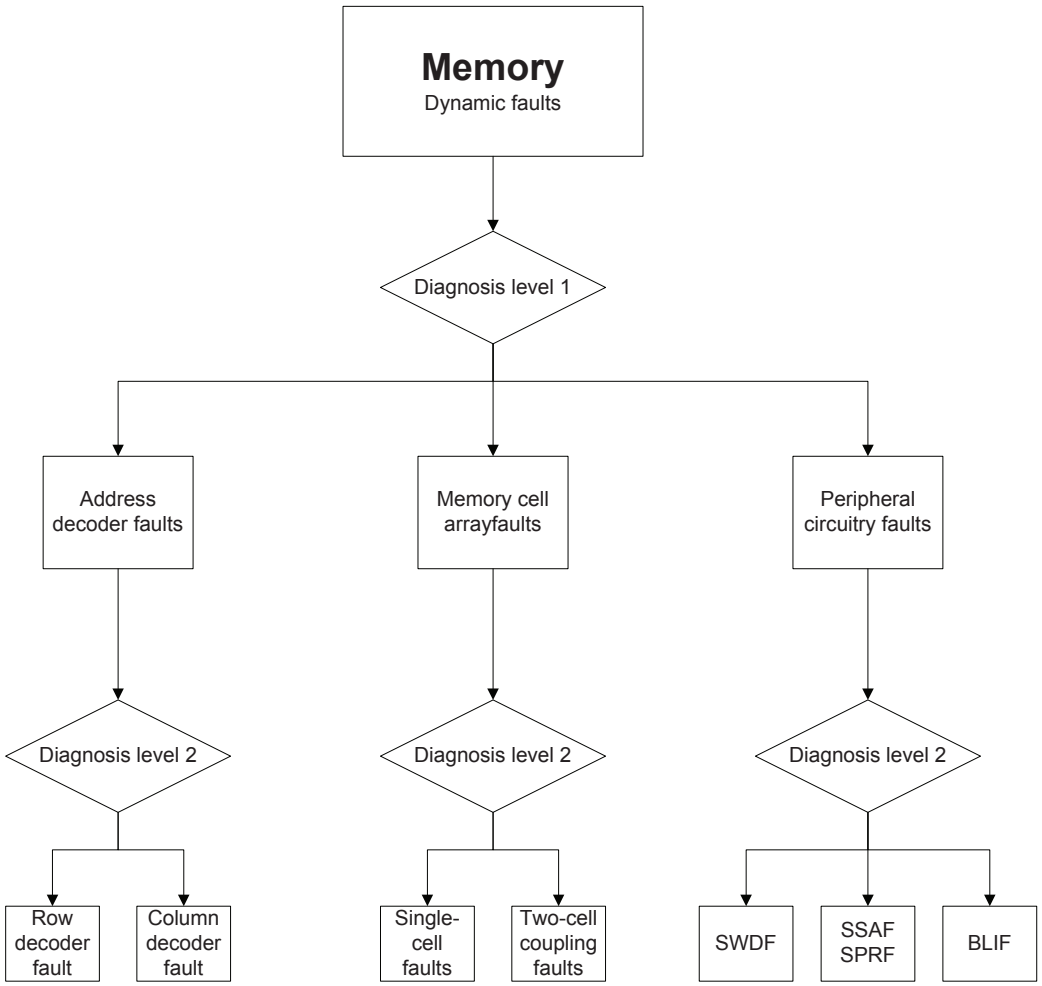

Figure 5.7: Diagnostic levels for dynamic faults in SRAM

Step 1: Test Classes

Minimum length test to detect all the dynamic single-cell and two-cell coupling faults in the memory cell array is March MD2 [15]. The description of March MD2 is:

March MD2:
$\{ \Updownarrow (w0); \Uparrow (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1);$
$\Uparrow (r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0);$
$\Downarrow (r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1);$
$\Downarrow (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); \Updownarrow (r0) \}$

It is to be established that the targeted faults in the memory cell array are single-cell faults and two-cell coupling faults where both sensitizing operations are sequentially applied either on the aggressor cell or on the victim cell. In order to maximize the test stress, March MD2 is to be applied in fast-column direction and is termed as March $MD2_y$. Applying a test in fast-column direction is useful for detecting dynamic faults

as it increases the test stress due to the read equivalent stress [14]. March MD2$_y$ might also detect some peripheral circuitry faults due to presence of operation sequences like $(..w0, w1)$, $(..w1, w0)$, $(r1, ..w0)$ and $(r0, ..w1)$. Consecutive complementary write operations can sensitize slow write driver fault while a write operation followed by a complementary read operation can sensitize slow sense amplifier and slow precharge circuit faults. As March MD2$_y$ makes address transitions in both row and column decoder, it will also detect some address decoder delay faults. Thus, March MD2$_y$ detects all the dynamic faults in the memory cell array, some dynamic faults in the peripheral circuitry and some address decoder delay faults.

Several march tests are introduced in the literature for detecting the activation and deactivation delay faults in the address decoder. March RAWAW-H1 is a march test based on RAWAW (read after write after write) address sequence and H1 addressing [22]. The description of March RAWAW-H1 is:

March RAWAW-H1$_x$: $\{ \Updownarrow_x (w0); \Updownarrow_x^{H1} (w0_g, w1_f, r0_g); \Updownarrow_x (w1); \Updownarrow_x^{H1} (w1_g, w0_f, r1_g) \}$

When applied in fast-row address direction, it can detect all the activation and deactivation delay faults in the row decoder. This test also detects all peripheral circuitry faults. March RAWAW-H1 has been modified to ensure maximum stress for detection of bit-line coupling faults and is termed as March RAWAW-H1m. This same test can be applied in fast-column direction to detect all the activation and deactivation delay faults in the column decoder. When applied in fast-column direction, March RAWAW-H1m can also detect some memory cell array faults due to the read equivalent stress.

For level 1 TCs, March MD2 is applied in fast-column direction (March MD2$_y$), March RAWAW-H1m is applied in fast-row direction (March RAWAW-H1m$_x$) and fast-column direction (March RAWAW-H1m$_y$). March MD2$_y$ is required for detecting memory cell array faults. March RAWAW-H1m$_x$ and March RAWAW-H1m$_y$ are required for detecting address decoder delay faults and peripheral circuitry faults. In total, three TCs are proposed:

- **March MD2$_y$**:
  $\{ \Updownarrow_y (w0); \Uparrow_y (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1);$
  $\Uparrow_y (r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0);$
  $\Downarrow_y (r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1);$
  $\Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$
- **March RAWAW-H1m$_x$**: $\{ \Updownarrow_x (w0); \Updownarrow_x^{H1} (w0_g, w1_f, r0_g, r1_f, w0_f); \Updownarrow_x (w1); \Updownarrow_x^{H1} (w1_g, w0_f, r1_g, r0_f, w1_f) \}$
- **March RAWAW-H1m$_y$**: $\{ \Updownarrow_y (w0); \Updownarrow_y^{H1} (w0_g, w1_f, r0_g, r1_f, w0_f); \Updownarrow_y (w1); \Updownarrow_y^{H1} (w1_g, w0_f, r1_g, r0_f, w1_f) \}$

Ideally, memory cell array faults should not be detected by March RAWAW-H1m$_x$ or March RAWAW-H1m$_y$. This is because in these tests no two consecutive operations are performed on the same memory cell. But it does not hold true; as due to the read

equivalent stress March RAWAW-H1m$_y$ will detect some memory cell array faults. Also, as stated before, March MD2$_y$ detects some address decoder delay faults. For some specific faults, this causes same diagnostic signature for the faulty address decoder and the faulty memory cell array. Generalizing, any test for detecting column decoder delay faults is to be applied in fast-column direction and due to read equivalent stress, it will detect some memory cell array faults. Also, any test used for detecting memory cell array faults will include some address decoder transitions, leading to detection of some address decoder delay faults. Thus, it is not possible to generate unique diagnostic signatures to distinguish address decoder delay faults and memory cell array faults by using just the pass/fail status of TCs.

Similar problem comes while distinguishing peripheral circuitry faults and address decoder delay faults. Any test for detecting peripheral circuitry faults is to be applied in fast-row direction and thus, will also detect some address decoder delay faults. Also, any test applied for detecting row decoder delay faults is applied in fast-row direction and will detect some peripheral circuitry faults. Again, it is not possible to generate unique diagnostic signatures to distinguish address decoder delay faults and peripheral circuitry faults by using just the pass/fail status of TCs. All these conclusions have been verified by simulations on HSpice. Thus, an additional parameter is required to distinguish peripheral circuitry faults from address decoder delay faults and to distinguish memory cell array faults from address decoder delay faults.

This can be accomplished by avoiding unnecessary address transitions in March RAWAW-H1m$_x$ and March RAWAW-H1m$_y$; i.e., by applying March RAWAW-H1m$_x$ individually to every column of the memory cell array and March RAWAW-H1m$_y$ to every row of the memory cell array. Diagnostic signatures will include logical AND and logical OR of the pass/fail status of march tests applied on each row and each column. As only one fault is present in the memory, if a memory cell array fault gets detected by March RAWAW-H1m$_y$, it is detected only by the TC applied to a particular row and logical ANDING of the pass/fail status of March RAWAW-H1m$_y$ for each row gives a '0' in diagnostic signature for memory cell array faults. Similarly, peripheral circuitry faults are detected by March RAWAW-H1m$_x$ applied to a particular column and logical ANDING of the pass/fail status of March RAWAW-H1m$_x$ for each column gives a '0' in diagnostic signature for peripheral circuitry faults. But an address decoder fault will be detected by diagnostic tests applied to all the rows (columns) and logical ANDING of the pass/fail status of March RAWAW-H1y$_x$ for each row (March RAWAW-H1m$_x$ for each column) gives a '1' in diagnostic signature. Logical ORing of the pass/fail status of March RAWAW-H1m$_x$ for each column and getting a '1' denotes presence of a peripheral circuitry faults or an address decoder delay fault. In this way, unique diagnostic signatures can be generated for memory cell array faults, address decoder faults and peripheral circuitry faults.

Step 2: Diagnostic Dictionary

In the previous section, TCs have been developed/presented and have demonstrated the capability to successfully locate the faulty block in the memory. The next step is to create the diagnostic dictionary. Table 5.11 presents the diagnostic signatures which

Table 5.11: Diagnostic dictionary

#	Faulty Block	March MD2$_y$	March RAWAW -H1m$_x$(AND)	March RAWAW -H1m$_x$(OR)	March RAWAW -H1m$_y$(AND)
1	Address decoder	- -	1 0	1 0	0 1
2	Peripheral circuitry	-	0	1	0
3	Memory cell array	1	0	0	0

use the above mentioned TCs and uniquely distinguish the faulty memory block. The value '-' in the diagnostic dictionary symbolizes the irrelevance of that bit in diagnostic signature. Depending on the fault, '-' can be either '0' or '1'. The total complexity for all the diagnostic tests at level 1 is 70n for March MD2$_y$ + 2*(2n + 5nN) for March RAWAW-H1m$_x$ and March RAWAW-H1m$_y$. Here, n is the number of memory cells and N is number of address bits.

5.5.3 Level 2: Diagnosing the Fault Type

Once the faulty block of the memory has been determined, it is required to identify the fault type. For this purpose, diagnostic tests specific to the faulty memory block are applied to the memory. As explained in Section 5.3, diagnosing the fault type is a two step process:

- Step 1: Develop TPs for targeted FCs.
- Step 2: Create FP x TP dictionary.

Next, these two steps are analyzed in detail with respect to different faulty blocks.

Faulty Memory Cell Array

Once the faulty block has been identified as the memory cell array, the fault can be classified as a single-cell or a two-cell coupling fault. To further classify memory cell array faults, we need to introduce new diagnostic tests. As we need to classify fault subclasses, TCs are used instead of TCs. For level 2, March MD2 is divided in two TCs (March MD2$_1$ and March MD2$_2$).

Step 1: Test Classes

We propose two TCs for classifying all memory cell array faults:

- **March MD2**$_1$: $\{ \Updownarrow_y (w0); \Uparrow_y (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1); \Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$

- **March MD2$_2$**: $\{ \Updownarrow_y (w0); \Downarrow_y (r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1); \Uparrow_y (r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$

March MD2$_1$ is able to detect all static single-cell faults but it can only detect 50% of the two-cell coupling faults. Similarly, March MD2$_2$ is able to detect all static single-cell faults but can only detect 50% of the two-cell coupling faults. Thus, a single-cell fault is detected by both March MD2$_1$ and March MD2$_2$ but a two-cell coupling fault is detected either by March MD2$_1$ or March MD2$_2$. March MD2$_1$ and March MD2$_2$, combined can detect all the targeted memory cell array faults. All these conclusions have been verified using the RASTA memory fault simulator [7].

Step 2: Diagnostic Dictionary

Previous sections provide details about the generation of adequate TCs and have demonstrated the capability of TCs to further classify memory cell array faults in the memory. The next step is to create the diagnostic dictionary. Table 5.12 presents the diagnostic signatures which use the above mentioned TCs to classify memory cell array faults. The total complexity for all the diagnostic tests is 72n.

Table 5.12: Diagnostic dictionary for memory cell array faults

#	Fault	March MD2$_1$	March MD2$_2$
1	Single-cell fault	1	1
2	Two-cell coupling fault	1	0
		0	1

Faulty Address Decoder

It is already known from level 1 that the fault lies in the address decoder. It is of interest to find whether the fault is a row decoder delay fault or a column decoder delay fault.

Step 1: Test Primitives

This can be achieved without applying any new tests. The pass/fail information of TCs applied at level 1 is sufficient for this classification. As March RAWAW-H1m$_x$ is applied individually for every column, there are no column address transitions and a fault in column decoder cannot be detected. Similarly, March RAWAW-H1m$_y$ is applied individually for every row and there are no row address transitions. Hence, a fault in row decoder cannot be detected by March RAWAW-H1m$_y$. Logical ANDING of the pass/fail status of March RAWAW-H1m$_x$ for each column gives a '0' in diagnostic signature for column decoder faults and Logical ANDING of the pass/fail status of March RAWAW-H1m$_y$ for each row gives a '0' in diagnostic signature for row decoder faults.

Table 5.13: Diagnostic dictionary for address decoder faults

#	Faulty Block	March RAWAW-H1m$_x$ (AND)	March RAWAW-H1m$_y$ (AND)
1	Row decoder	1	0
2	Column decoder	0	1

Step 2: Diagnostic Dictionary

As mentioned above, no new tests are applied to further classify the address decoder faults. Table 5.13 uses the pass/fail status of TCs applied at level 1 and provides the diagnostic dictionary for classification of address decoder faults.

Faulty Peripheral Circuitry

Once the faulty block has been identified as the peripheral circuitry, the fault can be further classified as slow write driver fault, bit-line imbalance fault, slow sense amplifier fault or slow precharge circuit fault. To make this classification, we need to introduce new diagnostic tests.

Step 1: Test Primitives

We propose two TPs for further classifying peripheral circuitry faults. The first test March WDmm is derived from March WDm [56]. The description of March WDm is:

> **March WDm**: { $\Updownarrow_x$ (wD), $\Updownarrow_x$ (rD, wD), $\Updownarrow_x$ $(w\bar{D})$, $\Updownarrow_x$ $(r\bar{D}, w\bar{D})$
> Here D represents checkerboard or row stripe data background.

March WDm can detect slow write driver faults in the peripheral circuitry. It is modified to include detection conditions for slow sense amplifier fault and slow precharge circuit fault. The modified test is termed March WDmm. The second test is March BLI, which is an already established march test for detecting slow write driver fault and bit-line imbalance fault [56].

- **March WDmm**: { $\Updownarrow_x$ (wD), $\Updownarrow_x$ (rD, wD), $\Updownarrow_x$ $(w\bar{D})$, $\Updownarrow_x$ $(r\bar{D}, w\bar{D})$
  Here D represents checkerboard or row stripe data background.
- **March BLI**: { $\Updownarrow_x$ (wD), $\Updownarrow_x$ $(w\bar{D}, r\bar{D}, wD)$, $\Updownarrow_x$ $(w\bar{D})$, $\Updownarrow_x$ $(wD, rD, w\bar{D})$
  Here D represents solid or column stripe data background.

Step 2: Diagnostic Dictionary

Previous sections provide details about the generation of adequate tests and have demonstrated the capability of TPs to classify distinguish peripheral circuitry faults in the memory. The next step is to create the diagnostic dictionary. Table 5.14 presents the diagnostic signatures which use the above mentioned tests to classify peripheral circuitry faults. The total complexity for all the diagnostic tests is 14n.

Table 5.14: Diagnostic dictionary for peripheral circuitry faults

#	Fault	March WDmm	March BLI
1	Slow write driver fault	1	1
2	Bit-line imbalance fault	0	1
3	Slow sense amplifier fault Slow precharge circuit fault	1	0

It can be observed from the diagnostic dictionary that the fault signatures for slow sense amplifier fault and slow precharge circuit fault are same. It is not possible to distinguish slow sense amplifier fault and slow precharge circuit fault as both of these faults have same detection conditions. Any test applied to detect slow sense amplifier fault will also detect slow precharge circuit fault and vice versa. However, these faults can be distinguished if internal structure of the memory is known. We assume that every column of the memory cell array has its own set of peripheral circuitry. In that case, these two faults cannot be distinguished by using just the outcome of read operations. But if the memory structure is different; for e.g., four columns of the memory cell array share a sense amplifier while every column has its own precharge circuit, faults can be distinguished. March WDmm applied in fast-column direction will detect slow sense amplifier fault as complementary operations with RAW sequence are applied on the same sense amplifier but will not detect slow precharge circuit fault as no consecutive operation are applied on same precharge circuit. Thus, by using information about the memory structure, slow sense amplifier fault and slow precharge circuit fault can be distinguished.

In this section, detailed analysis was presented for the diagnosis of dynamic faults in the memory. The diagnosis approach has been based on the concepts of TCs and TPs. The outcome of the diagnosis contains details about the faulty block of the memory and what type of fault is present. The work can be extended to go further down the hierarchy to precisely pin-point the defect location. Also, with this proof of concept, more TCs and TPs can be added to include support for newer fault models.

5.6 Advantages and cost of HMD

In this chapter, a theoretical proof has been given for the proposed HMD approach. HMD has been demonstrated to diagnose static and dynamic faults in all the blocks of the memory system. The outcome of the diagnosis includes the details about the faulty block of the memory at level 1 and further down the hierarchy, more details like fault location and fault type are made available. In this section, we will summarize the advantages of the proposed approach and the cost involved.

5.6.1 Advantages

The advantages of the proposed HMD approach are:

- Targets static and dynamic faults in all the memory blocks, namely, the memory cell array, the address decoder and the peripheral circuitry.
- No specific implementation requirements other than applying a test and determining the pass/fail status of the diagnostic test. Sometimes requires information on the address of first fail of the applied test.
- Accurate information on faulty block.
- Hierarchical methodology significantly reduces diagnosis effort. For example, there is no need to apply diagnostic tests for coupling faults in the memory cell array when a fault exists in the address decoder. The faulty memory block is identified at the first step of the diagnosis.

5.6.2 Cost

HMD has been based upon the concept of TCs, TPs and DfD. The diagnosis process is able to locate the fault but every such process involves cost (for e.g., test complexity, post-test analysis, platform dependence etc.). As HMD uses only pass/fail status of the diagnostic tests, involved costs are minimal. In addition to cost of applying diagnostic tests to the faulty memory, DfD hardware is the only other requirement.

DfD uses extra hardware for introducing new operational modes of the memory. These modes are referred to as diagnosis mode and have been explained in Section 5.4.2 and Section 5.4.3. These sections provide the conceptual description of the DfD hardware. For the sake of simplicity, an assumption has also been made that each column of the memory cell array has its own set of peripheral circuits such as sense amplifies, write drivers, pre-charge circuits, etc. Here, we will talk about the implementation and cost of using DfD for real case memories where peripheral circuits (sense amplifies, write drivers) are shared by several memory cell array columns. We will do this for the diagnosis mode described in Section 5.4.3.

First of all, we will present the memory model without the DfD hardware. Figure 5.8 shows the memory model where peripheral circuits are shared by two columns of the memory cell array. It can be seen that memory cell array column 1 and column 2 share the same set of peripheral circuitry (PC1) and are accessed by address line 1 and address line 2 respectively. Similarly, memory cell array column 3 and column 4 share the same set of peripheral circuitry (PC2) and are accessed by address line 3 and address line 4 respectively.

As explained in Section 5.4.3, a new operational mode (diagnosis mode) is required to distinguish the address decoder and peripheral circuitry fault classes. This is done by accessing same peripheral circuits (PC1 and PC2) by different address lines. This can be achieved by inserting extra multiplexers in the memory. Figure 5.9 shows the implementation of DfD for the memory model presented in Figure 5.8. For the diagnosis purpose, additional multiplexers have been inserted in the memory system. Depending on the

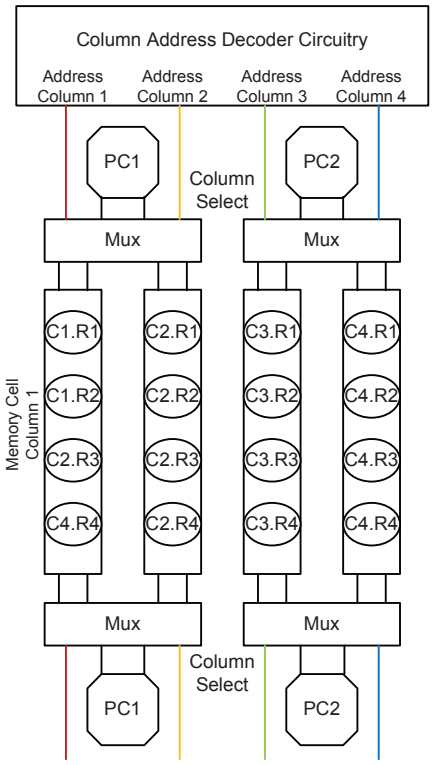

Figure 5.8: Memory Model

select line of the multiplexers, memory operates in normal mode (diagnosis (OFF)) or diagnosis mode (diagnosis (ON)). Fig 5.9(a) shows the connections between the functional blocks of the memory system in normal mode. It can be seen that the peripheral circuitry PC1 is connected to address line 1 and address line 2 while in diagnosis mode (see Fig 5.9(b)), the peripheral circuitry PC1 is connected to address line 3 and address line 4.

For this work, we have used extra multiplexers for diagnosis purpose. The implementation of DfD is explained above. The aim of DfD is to distinguish address decoder and peripheral circuitry fault class and this is done by accessing same peripheral circuits by different address lines. However, the above presented implementation is not the only way to achieve this. Depending on the internal structure of the memory, it is possible to implement the same operational mode by inserting DfD hardware elsewhere. For example, it is possible to extend the peripheral circuitry multiplexers and include the diagnosis pin as select line. The select lines will thus include the different address lines and diagnosis pin, and depending on the state of diagnosis pin, a particular address line will access the memory cell array.

The addition of the multiplexers adds extra overhead in terms of silicon area. However, this overhead is minimal as a multiplexer consists of only 4 transistors and is smaller than a single memory cell (6 transistors) in terms of silicon area. For the explained implementation, the requirement is of only one multiplexer per address line and thus, the area overhead is negligible. The DfD hardware also leads to some extra delay in the mem-

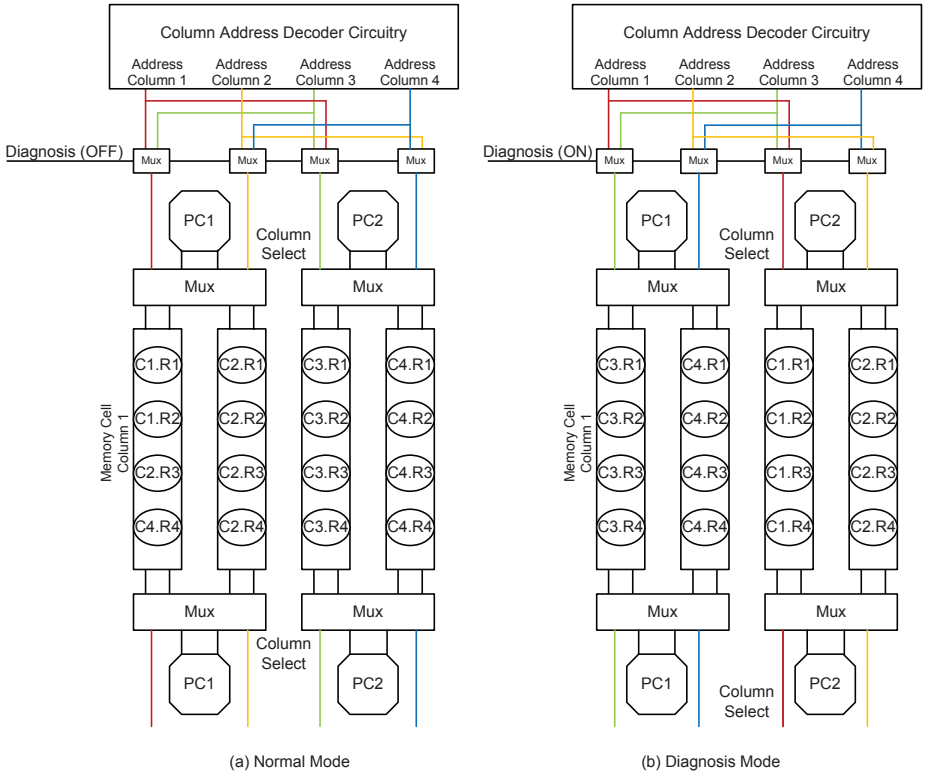

Figure 5.9: Memory model with design for diagnosis

ory system but experimental results show that it does not affect the normal memory operation. The involved cost of DfD is negligible compared to the benefits of the diagnosis. Moreover, these multiplexers can be removed from design when the characterization testing procedure is over and the memory is approved for mass production.

6 Case Studies

This chapter validates the Hierarchical Memory Diagnosis (HMD) approach by means of simulations. Defects are injected in various blocks of the SRAM simulation model and the HMD approach is applied for the diagnosis. Simulation results are presented for both static faults and dynamic faults. The injected fault is diagnosed and fault type is determined, thus establishing the usefulness of HMD.

This chapter is organized as follows. Section 6.1 explains the simulation model and the simulation approach. Section 6.2 presents the simulation results for static analysis and Section 6.3 presents the simulation results for dynamic analysis. Section 6.4 summarizes the conclusions.

6.1 SRAM Simulation Model and Simulation Approach

Theoretical proof for the HMD approach has been established in chapter 5. This chapter validates theoretical proof using simulations. In this section, we will first start with the explanation of the simulation model, list the simulation parameters and explain the additional Design for Diagnosis (DfD) hardware. Then, the simulation approach, and memory addressing will be explained using a memory block diagram.

6.1.1 Simulation Model

For the simulations, an appropriate memory simulation model has been used; it consists of a 4x4 cell array, address decoder and each column has its own set of peripheral circuits such as sense amplifies, write drivers, pre-charge circuits, etc. Figure 6.1 presents the gate and transistor level description of the memory model used for simulation. It is to be noted that the dotted lines of same color means that there is a physical connection between the two points. The entire physical connections have been avoided due to lack of space.

As mentioned in Section 3.1, all the memory components of a memory system can be categorized in three major functional blocks, namely, the memory cell array, the address decoder and, the peripheral circuitry. The three functional blocks are highlighted in the figure and they consist of following components:

- Memory cell array: A 4x4 cell array, with cells numbered from 1 to 16 along horizontal direction; see Figure 2.5 for cell description.
- Address decoder
  - Row decoder
  - Column decoder

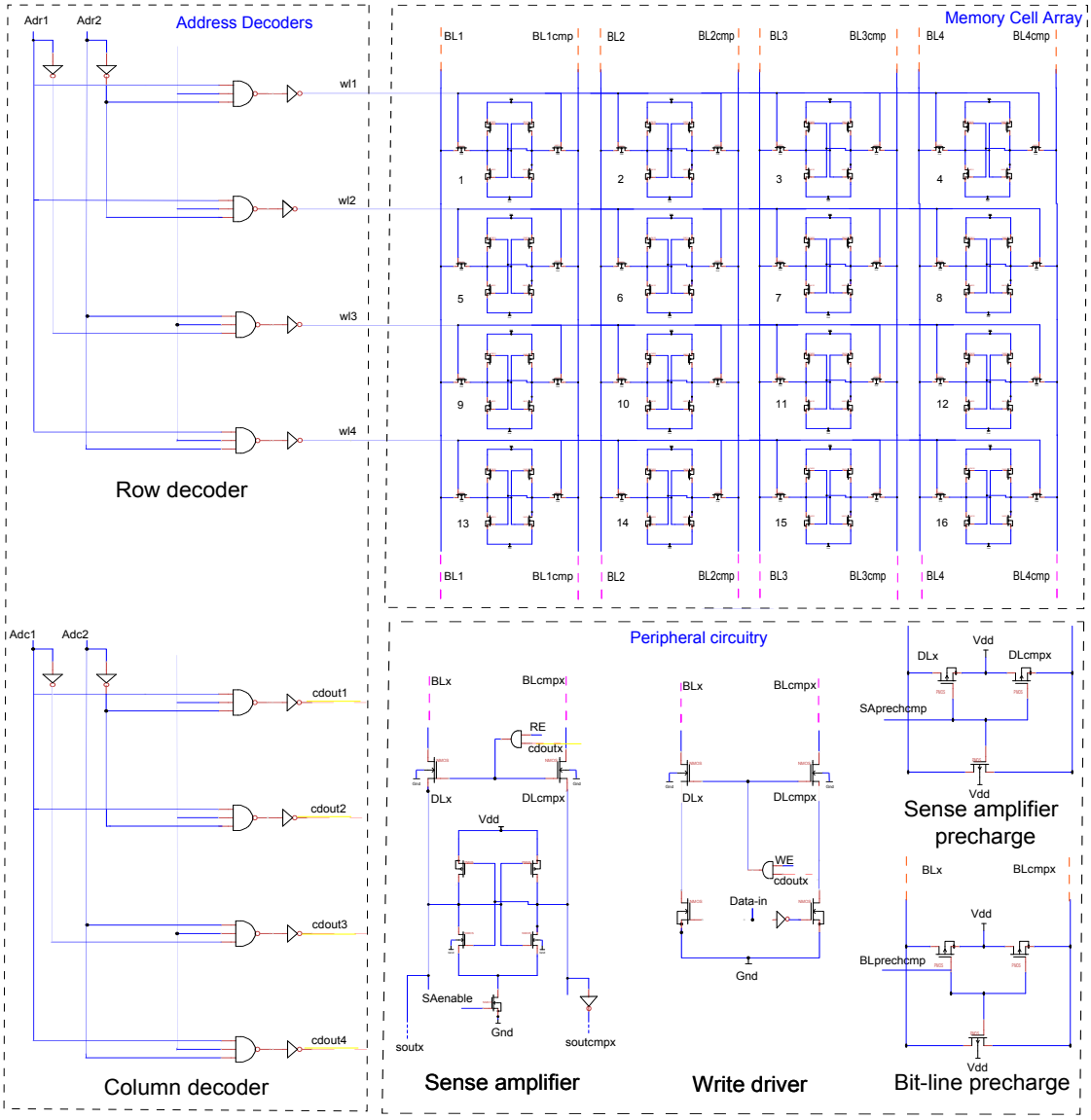

Figure 6.1: Electrical level schematic of SRAM

- Peripheral circuitry
  - Precharge circuits for bit lines and sense amplifiers.
  - Differential mode voltage sense amplifiers
  - Write drivers

In the figure, only one set of peripheral circuitry has been shown but as it has been mentioned above every column in the memory cell array has its own set of peripheral circuitry. In the labels for peripheral circuitry, x $\in$ {1, 2, 3, 4 } signify that the peripheral circuitry is connected to different bit-lines. For the ease of presenting the simulation

results, depending on which memory cell column is read, the outputs of four sense amplifiers (soutx and soutcmpx) will be multiplexed to give outputs sout and soutcmp; thus in the simulation waveforms the results of the read operations performed on memory will be presented using one common signal sout and soutcmp (see Figure 6.7).

The timing generation and control circuitry has not been shown, as for simulations signals were defined manually using Piece Wise Linear (PWL) function in HSPICE. The voltage levels for different signals are defined for different instances of time and the simulations are performed through transient analysis. The inputs to the memory model consists of signals:

- Bit-line precharge enable (BLprechcmp in figure)
- Data input (Data-in)
- Write enable signal (WE)
- Read enable (RE)
- Sense amplifier precharge enable (SAprechcmp)
- Sense amplifier enable (SAenable)
- Row decoder enable (rdenable)
- Column decoder enable (cdenable)
- Row and column address lines (Adr1, Adr2, Adc1 and, Adc2)

The behavior of simple components like resistors, capacitors, etc. can be easily simulated by HSPICE. However, to simulate the behavior of a transistor, a number of parameters must be taken into account. The MOS model cards provides the specification for the parameters, which describe the working of a transistor. The transistor parameters used for design and implementation of the simulation model are as described for 65nm BSIM4 (level 54) Berkeley MOS models. Several other parameters describing the simulation circuit and simulation conditions are listed below:

- The supply voltage (vdd) is 1.2v. Voltage levels between 0v to vdd/2 (0.6v) are refereed as logic 0, while voltage levels between vdd/2 to vdd are refereed as logic 1.
- The bit lines, word lines and data lines have a capacitance to ground of 0.1pF.
- The coupling capacitances between bit lines and word lines are neglected.
- All simulations are performed at 300K temperature conditions.

As it has been explained in Section 5.4.2 and Section 5.4.3, for diagnosis purposes, additional hardware referred to as DfD is required. For the simulations, the above described memory model is modified to add DfD hardware. The additions consists of extra multiplexers added at two places, the column decoder circuitry and the write

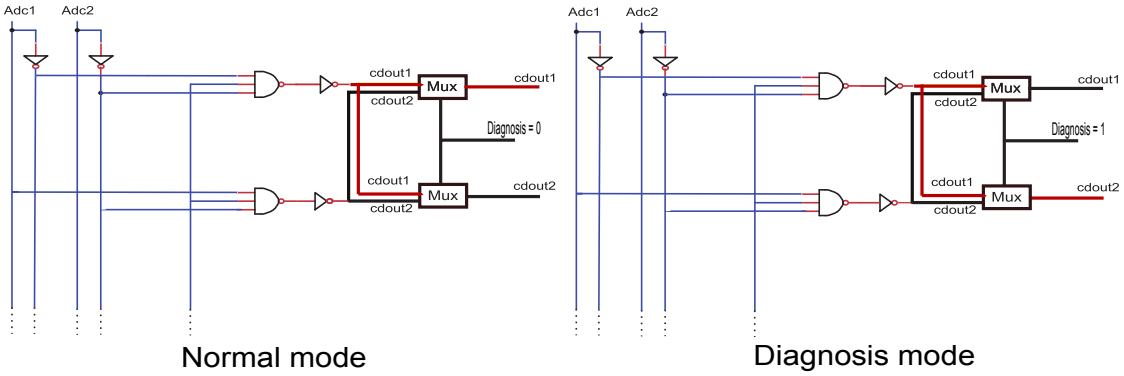

Figure 6.2: Diagnosis hardware for column decoder

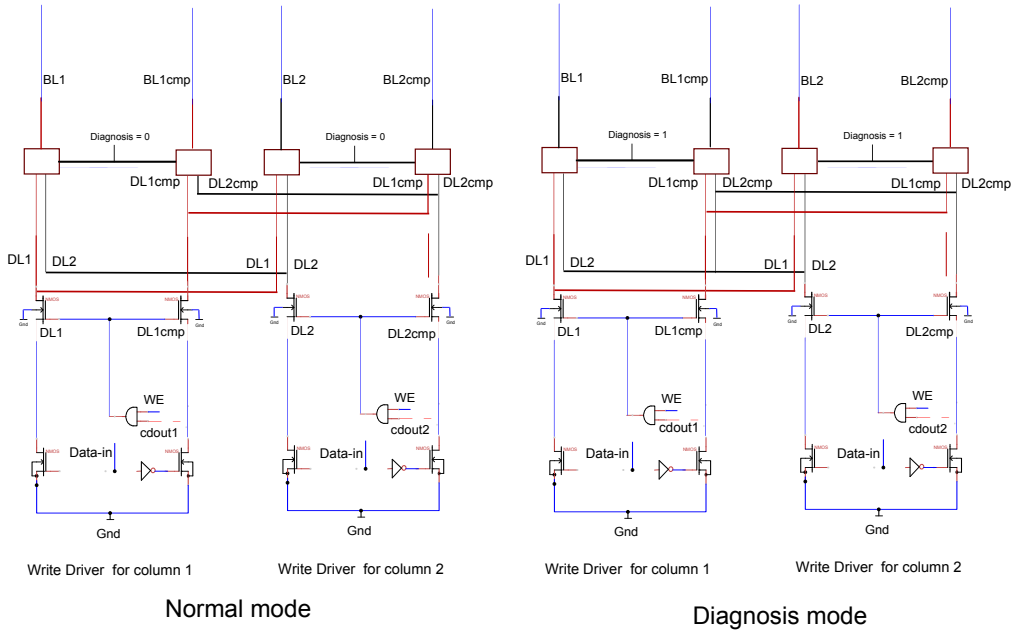

Figure 6.3: Diagnosis hardware for write path

circuitry. Figure 6.2 shows the multiplexers for DfD explained in Section 5.4.2. In diagnosis mode, every column of the memory is addressed by the addressing sequence of the adjacent column i.e., column 1 of the memory is operated on when address bits for column 2 are supplied and column 2 of the memory is operated on when address bits for column 1 are supplied. As it can be observed, in normal mode cdout1 is “ON” when address “00” is applied but in diagnosis mode cdout2 is “ON” when address “00” is applied.

Figure 6.3 shows the multiplexers for DfD explained in Section 5.4.3. In diagnosis mode, every column of the memory cell array is connected to the write circuitry of the adjacent column i.e., the bit lines cells of column 1 of the memory cell array are connected to the write circuitry of column 2 while the bit lines of column 2 are connected to the write circuitry of column 1.

6.1.2 Simulation Approach

For the validation of the proposed approach, faults are introduced in the simulation model and then diagnosed using HMD approach. Defect injection is used induce faults at the hardware level. It is done by introducing additional components (defects) between any two nodes in the circuit, producing voltage or current changes in the circuit. Restive opens and bridges are injected in various blocks of the memory following the philosophy of only one defect at a time. The resulting fault is then diagnosed using the HMD approach; the procedure for diagnosis has been shown in Figure 5.1. The simulation approach can be briefly summarized as:

- Inject defect in a single block
- Apply static hierarchical analysis
- If no fault is detected, apply dynamic hierarchical analysis

As, it is already known which fault will be caused by the injected defect, simulation results have been presented only for dynamic analysis and not for static analysis when a dynamic fault is induced in the simulation model.

For analyzing simulation results and generating signatures, identification for memory addresses is required. Figure 6.4 denotes the addressing of the memory. Here, C stands for column and R stands for row. A diagnostic test applied in fast-row addressing direction accesses the memory addresses in order:

Up addressing ($\Uparrow$): C1.R1 $\rightarrow$ C1.R2 $\rightarrow$ C1.R3 $\rightarrow$ C1.R4 $\rightarrow$ C2.R1 $\rightarrow$ C2.R2 $\rightarrow$ C2.R3 $\rightarrow$ C2.R4 $\rightarrow$ C3.R1 $\rightarrow$ C3.R2 $\rightarrow$ C3.R3 $\rightarrow$ C3.R4 $\rightarrow$ C4.R1 $\rightarrow$ C4.R2 $\rightarrow$ C4.R3 $\rightarrow$ C4.R4.

Down addressing ($\Downarrow$): C4.R4 $\rightarrow$ C4.R3 $\rightarrow$ C4.R2 $\rightarrow$ C4.R1 $\rightarrow$ C3.R4 $\rightarrow$ C3.R3 $\rightarrow$ C3.R2 $\rightarrow$ C3.R1 $\rightarrow$ C2.R4 $\rightarrow$ C2.R3 $\rightarrow$ C2.R2 $\rightarrow$ C2.R1 $\rightarrow$ C1.R4 $\rightarrow$ C1.R3 $\rightarrow$ C1.R2 $\rightarrow$ C1.R1.

For a test applied in fast-column addressing direction with up addressing; the order in which the cells are accessed is: C1.R1, C2.R1, C3.R1...C3.R4, C4.R4. Similarly, for a test applied in fast-column addressing direction with down addressing; the order in which the cells are accessed is: C4.R4, C3.R4, C2.R4...C2.R1, C1.R1.

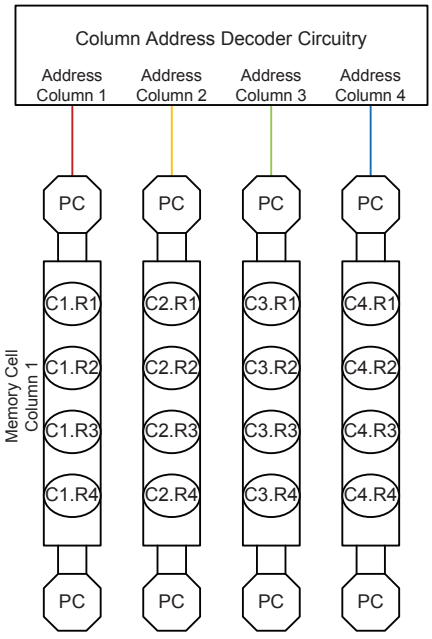

Figure 6.4: Memory addressing

As the cell addressing is defined in terms of column and row address lines, in diagnosis mode defined for level 1 (Section 5.4.2), the mapping between the memory addresses

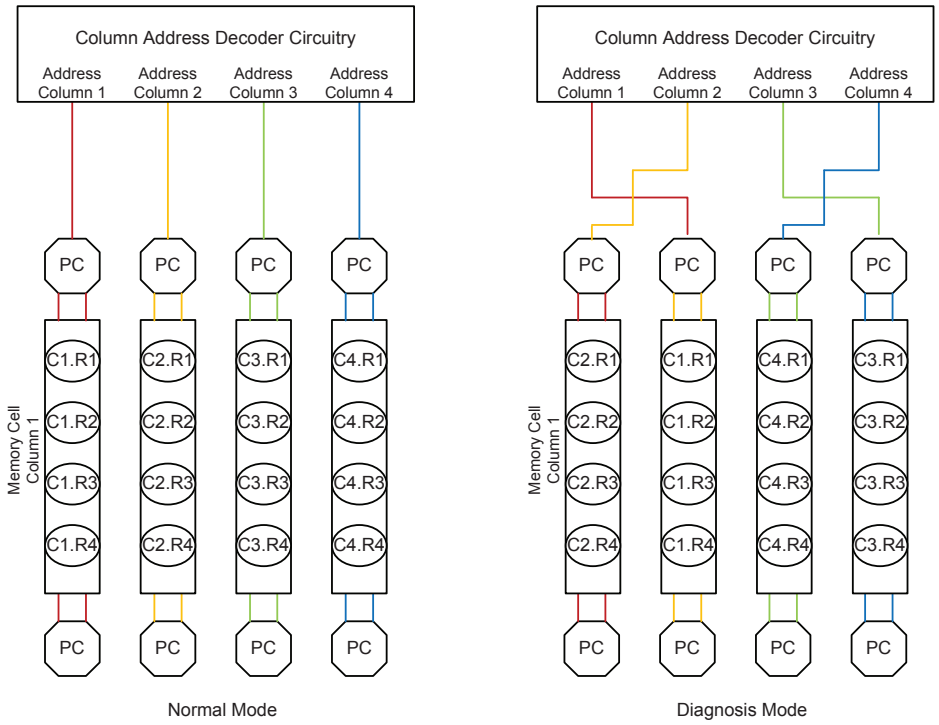

Figure 6.5: Memory addresses vs. memory cells location

and memory cells location will change. Figure 6.5 shows the mapping of the memory addresses and cells location in the normal mode and the diagnosis mode.

6.2 Static Hierarchical Analysis

This section presents the simulation results for diagnosing static faults occurring in the memory. A single defect causing a static fault is injected in the memory model at a time. Three experiments have been performed:

- Inject a defect in the memory cell array
- Inject a defect in the address decoder
- Inject a defect in the peripheral circuitry

In each of the experiment, the HMD procedure is performed and the faulty memory block as well as the fault type has been identified. In the rest of this section, the three experiments will be discussed and the results will be reported.

6.2.1 Diagnosing Faults in Memory Cell Array

A resistive defect has been injected in the memory cell at address C3.R1; see Figure 6.4. The defect is injected between the gate of pull down transistor at true node and the

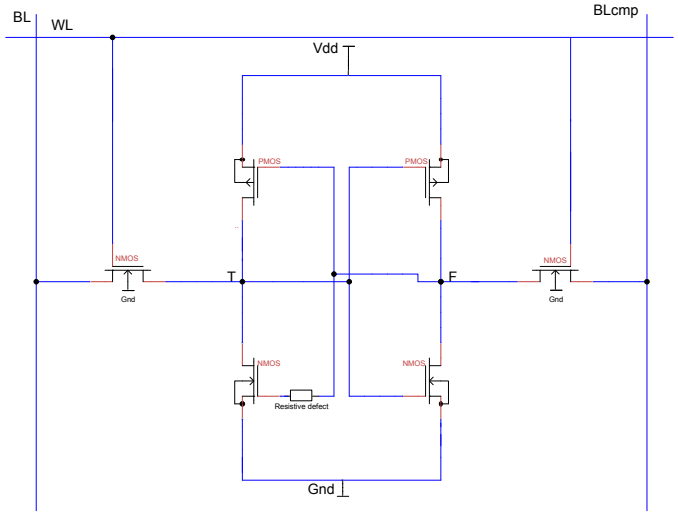

Figure 6.6: Resistive defect causing static fault in the memory cell array

false node as shown in Figure 6.6. The resistance of the defect is taken to be very high to make sure that it causes a static fault. In fact, such a defect can cause state fault $< 1/0/- >$; see Table 3.1.

As mentioned in Section 5.1, the diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and its location will be pinpointed. The two levels are discussed next.

Diagnosis Level 1

The TCs developed in Section 5.4.2 will be applied. They consist of:

- **March MSSm-up**: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (r1, r1, w0, w0); \Uparrow (r0, w1) \}$
- **March MSSm-down**: $\{ \Downarrow (w0); \Downarrow (r0, r0, w1, w1); \Downarrow (r1, r1, w0, w0); \Downarrow (r0, w1) \}$

The procedure consists of the following:

- Apply March MSS-up in normal mode
- Apply March MSS-down in normal mode
- Apply March MSS-up in diagnosis mode

All the tests will be applied in fast-row direction to nullify the effect of read equivalent stress [14], and thus, to avoid detection of dynamic faults.

The simulation results of the application of the above tests are presented in Figure 6.7, Figure 6.8, and Figure 6.9.

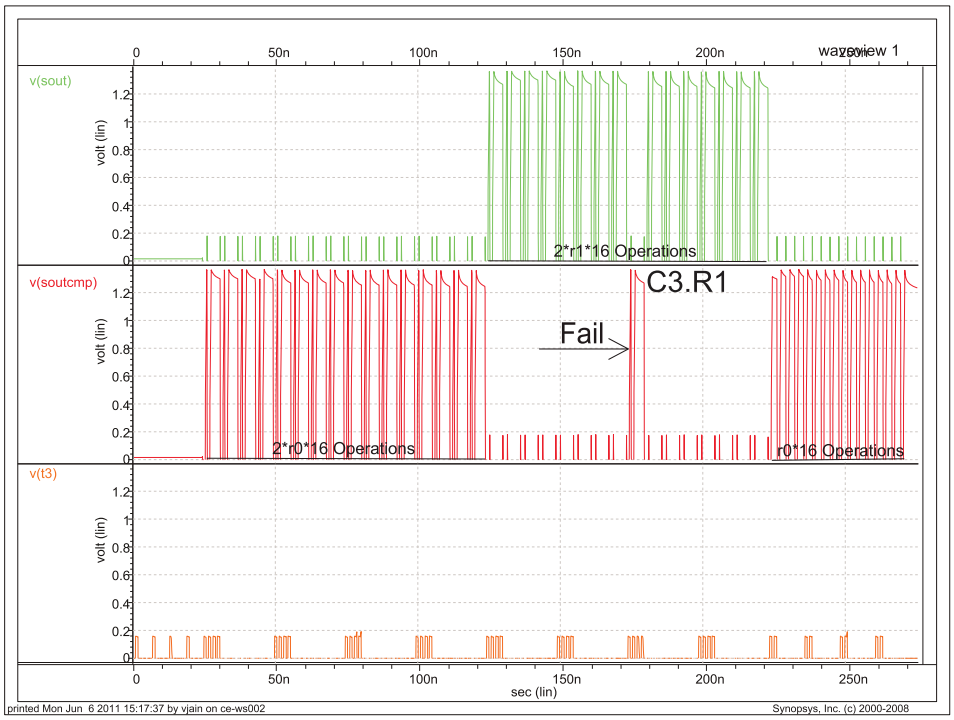

Figure 6.7: HSpice simulation: March MSSm-up in normal mode

March MSSm-up in normal mode: The simulation result of this test is shown in Figure 6.7. The top and middle graph in the figure give the outputs of the sense amplifier $V(sout)$ and its complement $\overline{V(sout)}$ (denoted as $V(soutcmp)$), respectively, while the bottom graph gives the true node voltage of the faulty cell (denoted as $V(t3)$).

Each operation of the test is applied to all 16 cells considered in the simulation model; see Figure 6.4. As we are using up addressing with fast-row address direction, the order in which the cells are accessed is: C1.R1, C1.R2, C1.R3...C4.R3, C4.R4. First, all the 16 cells are initialized to 0 by the first march element $\Uparrow (w0)$. Second, each cell is read two times and thereafter written two times with 1 by the second march element $\Uparrow (r0, r0, w1, w1)$. The top and the middle graph in Figure 6.7 shows the $V(sout)$ and $V(soutcmp)$ signal of the cell read operations. As it can be seen, all the read operations pass correctly; i.e., $V(sout) = 0$ and $V(soutcmp) = 1$. Third, each cell is read two times and then written with 1 by the third march element $\Uparrow (r1, r1, w0, w0)$. As the figure shows, the r1 operation applied to the cell labeled as C3.R1 fails. Finally, every cell is read and written to 0 by the fourth march element $\Uparrow (r0, w1)$; these read operations pass correctly. It is worth noting that the true node of the faulty cell shown in the bottom graph remains always 0; hence it fails to undergo an up transition. In conclusion, March MSSm-up fails only once at C3.R1.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (\mathbf{r1}, r1, w0, w0); \Uparrow (r0, w1) \}$

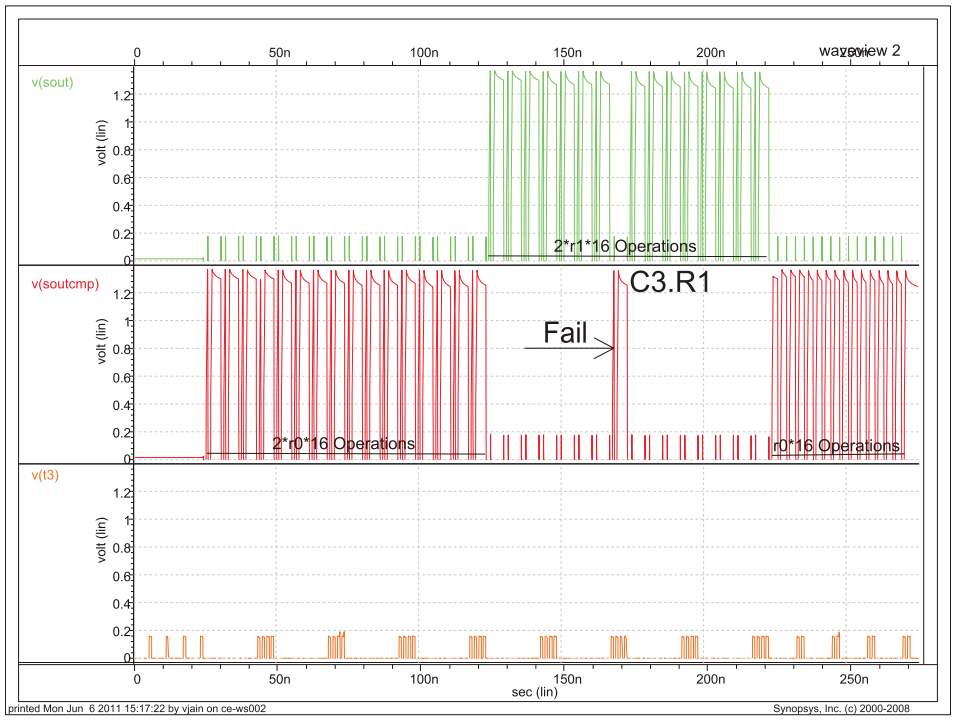

Figure 6.8: HSpice simulation: March MSSm-down in normal mode

March MSSm-down in normal mode: The simulation result of this test is shown in Figure 6.8.

In March MSSm-down, as we are using down addressing with fast-row address direction, the order in which the cells are accessed is: C4.R4, C4.R3, C4.R2...C1.R2, C1.R1. First, all the 16 cells are initialized to 0 by the first march element $\Downarrow (w0)$. Second, each cell is read two times and thereafter written two times with 1 by the second march element $\Downarrow (r0, r0, w1, w1)$. As it can be seen from the figure, all the read operations pass correctly. Third, each cell is read two times and then written with 1 by the third march element $\Downarrow (r1, r1, w0, w0)$. As the figure shows, the r1 operation applied to the cell labeled as C3.R1 fails. Finally, every cell is read and written to 0 by the fourth march element $\Downarrow (r0, w1)$; these read operations pass correctly. In conclusion, March MSSm-down fails only once at C3.R1

The fault detecting read operation is marked in bold.
March MSSm-down: $\{ \Downarrow (w0); \Downarrow (r0, r0, w1, w1); \Downarrow (\mathbf{r1}, r1, w0, w0); \Downarrow (r0, w1) \}$

March MSSm-up in diagnosis mode: The simulation result of this test is shown in Figure 6.9.

It is worth noting that in *diagnosis* mode, every column of the memory is addressed by the addressing sequence of the adjacent column i.e., column 1 of the memory is operated on when address bits for column 2 are supplied and column 2 of the memory is operated on when address bits for column 1 are supplied. First, all the 16 cells are initialized to 0 by the first march element $\Uparrow (w0)$. Second, each cell is read two times

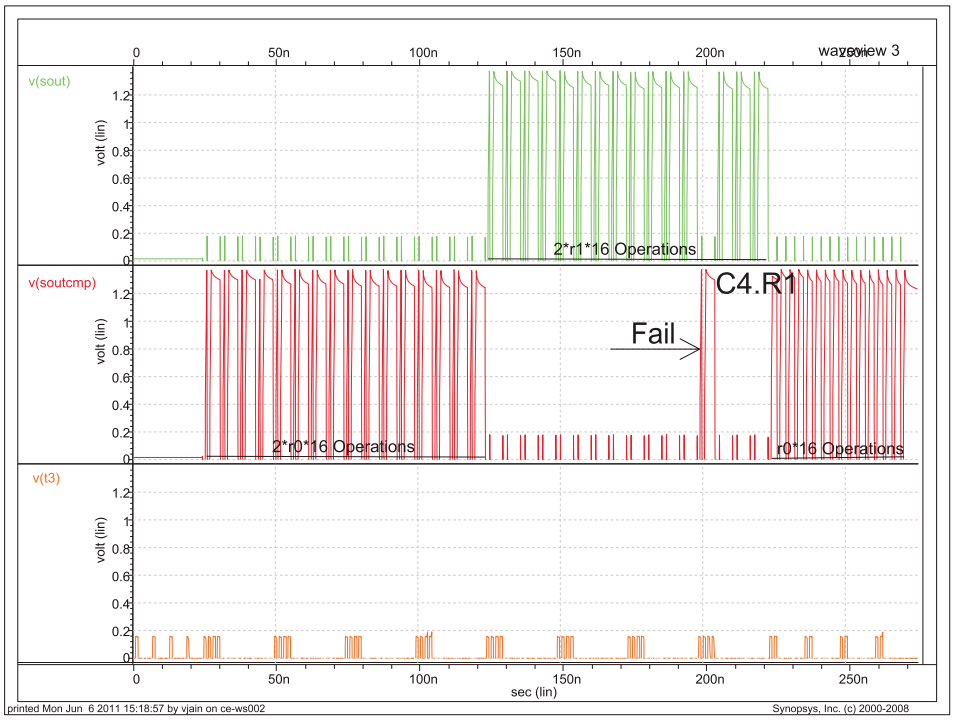

Figure 6.9: HSpice simulation: March MSSm-up in diagnosis mode

and thereafter written two times with 1 by the second march element $\Uparrow (r0, r0, w1, w1)$. Third, each cell is read two times and then written with 1 by the third march element $\Uparrow (r1, r1, w0, w0)$. As the figure shows, the r1 operation applied to the cell labeled as C4.R1 fails. Finally, every cell is read and written to 0 by the fourth march element $\Uparrow (r0, w1)$; these read operations pass correctly. In conclusion, March MSSm-up fails only once at C4.R1.

It is to be noted that the first failing address is different from when the same test (March MSSm-up) is applied in normal mode. This is because, in diagnosis mode the memory cell column containing the faulty cell is accessed by a different address line than in normal mode; hence the faulty cell is operated upon when a different address is accessed than that in normal mode.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (\mathbf{r1}, r1, w0, w0); \Uparrow (r0, w1) \}$

It is to be observed from provided simulation waveforms that logic value at true node of the faulty memory cell is always 0 i.e., the cell shows fault behavior of state-0 fault. All the applied tests fail generating a diagnostic signature "1(C3.R1) 1(C3.R1) 1(C4.R1)". In the signature, the first entry 1(C3.R1) signifies that the first test has failed and the first fail for this test is observed at address location C3.R1.

Table 6.1 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.6). On comparing the obtained signature with the diagnostic dictio-

nary, the fault can be classified as a fault in the memory cell array. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

Table 6.1: Level 1: Memory cell array identified as the faulty block

Faulty Block	Signature	MSSm-up in normal mode (failing address)	MSSm-down in normal mode (failing address)	MSSm-up in diagnosis mode (failing address)	Failing address
Memory cell array	Dictionary signature	1 (A_α)	1 (A_α)	1 (A_β)	$\alpha \neq \beta$
	Observed signature	1 (C3.R1)	1 (C3.R1)	1 (C4.R1)	

Diagnosis Level 2

At level 1, the faulty block has been identified as the memory cell array. At level 2, the fault type will be identified. As explained in Section 5.4.3, when the memory cell array is faulty, we do not need to apply any new diagnostic tests at level 2. Further classification can be made on the basis of pass/fail status of tests applied at level 1. This will be done using the results of:

- **March MSSm-up in normal mode**
- **March MSSm-down in normal mode**

The simulation results of the above tests are presented in Figure 6.7 and Figure 6.8. The results have already been explained above during the discussion of diagnosis level 1. Both the tests fail generating a diagnostic signature "1 1". In the signature, the first entry 1 signifies that the first test has failed.

Table 6.2 shows comparison of the obtained signature with level 2 diagnostic dictionary (see Table 5.7). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a single-cell fault. Also, the fault location is known to be at C3.R1 as the first failing address is same (C3.R1) for both the tests. The injected defect causes a state-0 fault in a single cell located at C3.R1; thus the correct fault type is indicated by the diagnosis process.

6.2.2 Diagnosing Faults in Address Decoder

A resistive defect has been injected in the row decoder between word line 1 (WL1) (corresponding to row 1 of the memory cell array) and WL2 (corresponding to row 2 of the memory cell array); see Figure 6.10. This defect causes the address decoder fault AF_{nma} between row 1 and row 2 of the memory; see Figure 3.5. As shown in the figure, the defect causes memory cells of row 1 (shown as C_x) to be accessed when address sequence for either row 1 and row 2 is applied while the memory cells of row 2 (shown as C_y) are inaccessible.

Table 6.2: Level 2: Fault in the memory cell array identified as single-cell fault

Fault	Signature	MSSm-up in normal mode	MSSm-down in normal mode
Single-cell fault	Dictionary signature	1	1
	Observed signature	1	1

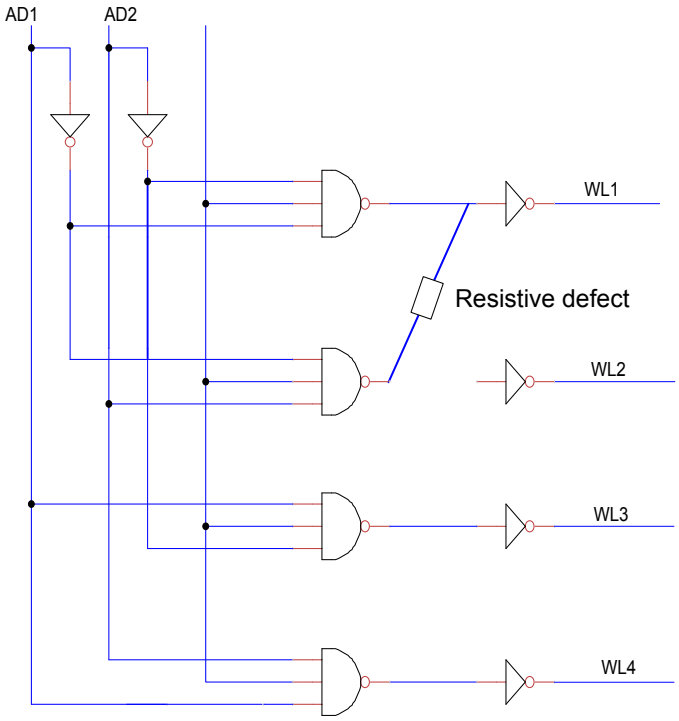

Figure 6.10: Resistive defect causing static fault in the address decoder

The diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and its location will be pinpointed. The two levels are discussed next.

Diagnosis Level 1

The TCs developed in Section 5.4.2 will be applied. The procedure consists of the following:

- Apply March MSS-up in normal mode
- Apply March MSS-down in normal mode
- Apply March MSS-up in diagnosis mode

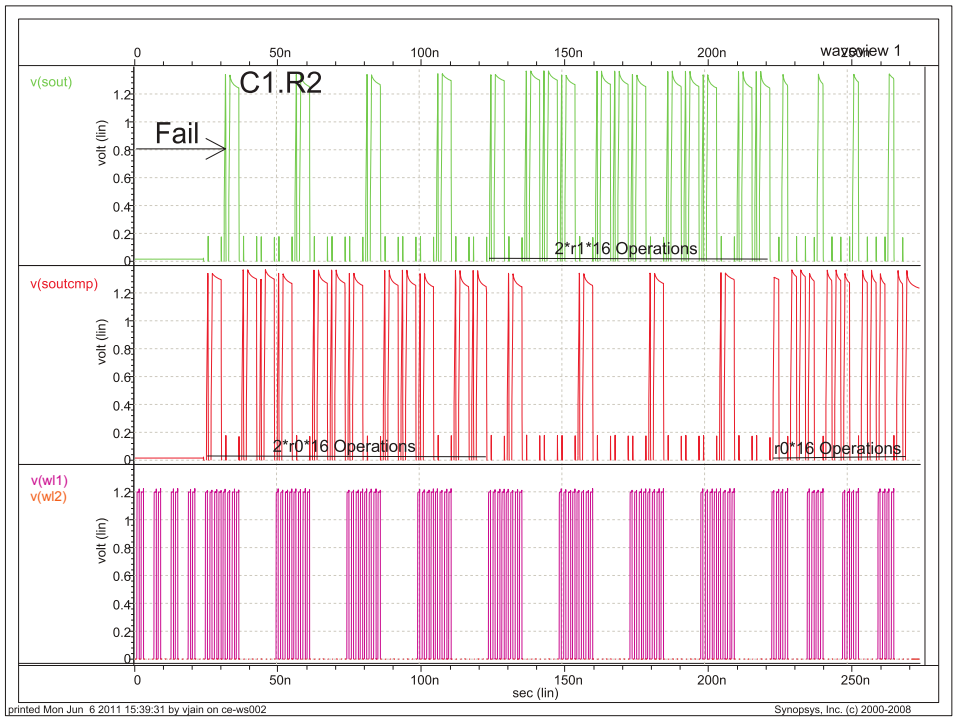

Figure 6.11: HSpice simulation: March MSSm-up in normal mode

The simulation results of the application of the above tests are presented in Figure 6.11, Figure 6.12, and Figure 6.13.

March MSSm-up in normal mode: The simulation result of this test is shown in Figure 6.11. Here, the bottom graph gives the voltage of the faulty word lines (denoted as $V(wl1)$ and $V(wl2)$).

As the figure shows, the r0 operation applied to the cell labeled as C1.R2 fails. Other cells accessed by address lines for row 2 (C2.R2, C3.R2 and, C4.R2) also fail for the r0 operation. It is worth noting that voltage of WL2 shown in the bottom graph remains always 0; hence the memory cells of row 2 are never accessed. It can also be seen that WL1 is activated consecutively two times for every operation in the march test; thus demonstrating the fault behavior for fault address decoder fault AF_{nma}. In conclusion, March MSSm-up has its first failing address at C1.R2.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (\mathbf{r0}, r0, w1, w1); \Uparrow (r1, r1, w0, w0); \Uparrow (r0, w1) \}$

March MSSm-down in normal mode: The simulation result of this test is shown in Figure 6.12.

As the figure shows, the r0 operation applied to the cell labeled as C4.R1 fails. Other cells accessed by address lines for row 1 (C3.R1, C2.R1 and, C1.R1) also fail for the r0 operation. In conclusion, March MSSm-down has its first failing address at C4.R1.

The fault detecting read operation is marked in bold.
March MSSm-down: $\{ \Downarrow (w0); \Downarrow (\mathbf{r0}, r0, w1, w1); \Downarrow (r1, r1, w0, w0); \Downarrow (r0, w1) \}$

March MSSm-up in diagnosis mode: The simulation result of this test is shown in Figure 6.13.

As the figure shows, the r0 operation applied to the cell labeled as C1.R2 fails. Other cells accessed by address lines for row 2 (C2.R2, C3.R2 and, C4.R2) also fail for the r0 operation. In conclusion, March MSSm-up has its first failing address at C1.R2.

It is to be noted that the first failing address for March MSS-up in diagnosis mode is same as that of March MSSm-up in normal mode. This is because, the diagnosis mode alters the column decoder to access different peripheral circuitries and does not effect the row decoder. As the complete row is affected by the injected defect, a defect in row decoder produces the same first failing address on application of same test.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (\mathbf{r0}, r0, w1, w1); \Uparrow (r1, r1, w0, w0); \Uparrow (r0, w1) \}$

It is to be observed from provided simulation waveforms that WL2 is never active and WL1 is active in its place. This is because of the injected defect, which demonstrates the fault behavior of address decoder fault AF_{nma}. All the applied tests fail generating a diagnostic signature "1(C1.R2) 1(C4.R1) 1(C1.R2)".

Table 6.3 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.6). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a fault in the address decoder. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

Table 6.3: Level 1: Address decoder identified as the faulty block

Faulty Block	Signature	MSSm-up in normal mode (failing address)	MSSm-down in normal mode (failing address)	MSSm-up in diagnosis mode (failing address)	Failing address
Address decoder	Dictionary Signature	1 (A_α)	1 (A_β)	1 (A_α)	$\alpha \neq \beta$
	Observed Signature	1 (C1.R2)	1 (C4.R1)	1 (C1.R2)	

Diagnosis Level 2

At level 1, the faulty block has been identified as the address decoder. At level 2, the fault type will be identified. When the address decoder is faulty, we need to apply new diagnostic tests at level 2. The TPs developed in Section 5.4.3 will be applied. They consist of:

- **March AFr0up**: $\{ \Updownarrow (w0); \Uparrow (r0, w1) \}$

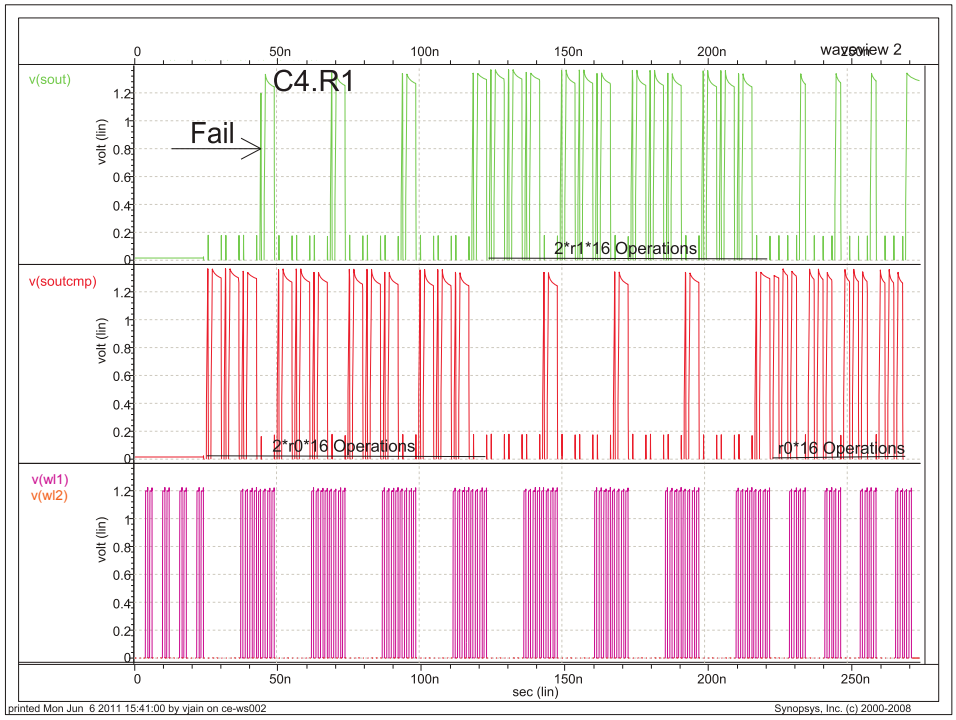

Figure 6.12: HSpice simulation: March MSSm-down in normal mode

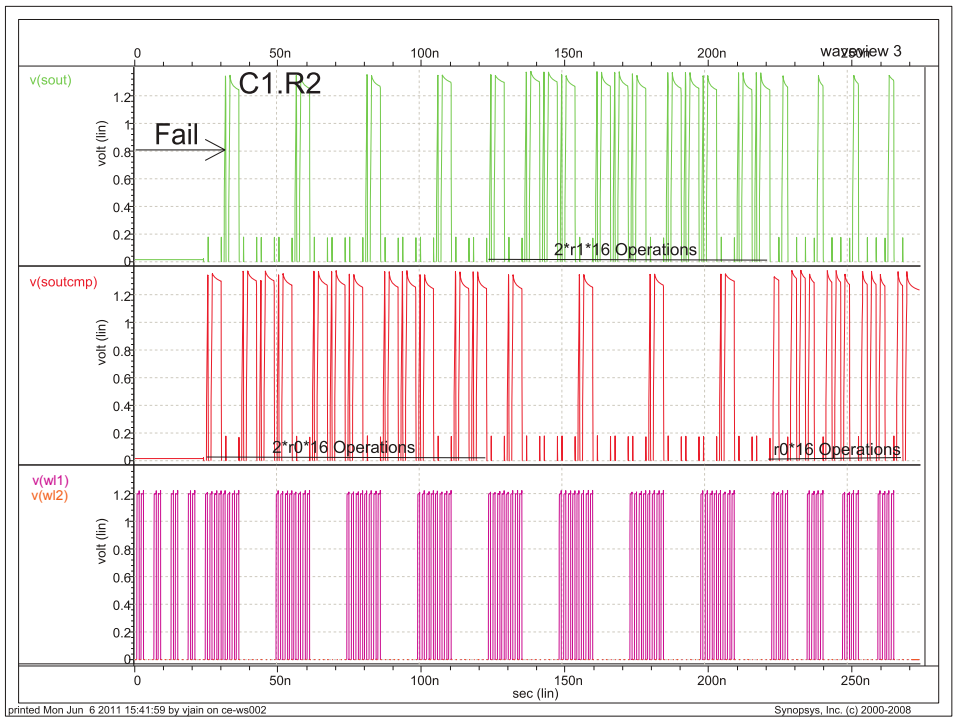

Figure 6.13: HSpice simulation: March MSSm-up in diagnosis mode

- **March AFr0down**: $\{ \Updownarrow (w0); \Downarrow (r0, w1) \}$
- **March AFr1up**: $\{ \Updownarrow (w1); \Uparrow (r1, w0) \}$
- **March AFr1down**: $\{ \Updownarrow (w1); \Downarrow (r1, w0) \}$

The procedure consists of the following:

- Apply March AFr0up
- Apply March AFr0down
- Apply March AFr1up
- Apply March AFr1down

March AFr0up: The simulation result of this test is shown in Figure 6.14. Here, the bottom graph gives the voltage of the faulty word lines (denoted as $V(wl1)$ and $V(wl2)$).

This test is applied using up addressing with fast-row address direction. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow (w0)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using up addressing here. Second, each cell is read and thereafter written with 1 by the second march element $\Uparrow (r0, w1)$. As the figure shows, the r0 operation applied to the cell labeled as C1.R2 fails. Other cells accessed by address lines for row 2 (C2.R2, C3.R2 and, C4.R2) also fail for the r0 operation. In conclusion, March AFr0up has its first failing address at C1.R2.

The fault detecting read operation is marked in bold.
March AFr0up: $\{ \Updownarrow (w0); \Uparrow (\mathbf{r0}, w1) \}$

March AFr0down: The simulation result of this test is shown in Figure 6.15.

This test is applied using down addressing with fast-row address direction. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow (w0)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using down addressing here. Second, each cell is read and thereafter written with 1 by the second march element $\Downarrow (r0, w1)$. As the figure shows, the r0 operation applied to the cell labeled as C4.R1 fails. Other cells accessed by address lines for row 1 (C3.R1, C2.R1 and, C1.R1) also fail for the r0 operation. In conclusion, March AFr0down has its first failing address at C4.R1.

The fault detecting read operation is marked in bold.
March AFr0down: $\{ \Updownarrow (w0); \Downarrow (\mathbf{r0}, w1) \}$

March AFr1up: The simulation result of this test is shown in Figure 6.16.

This test is applied using down addressing with fast-row address direction. First, all the 16 cells are initialized to 1 by the first march element $\Updownarrow (w1)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing;

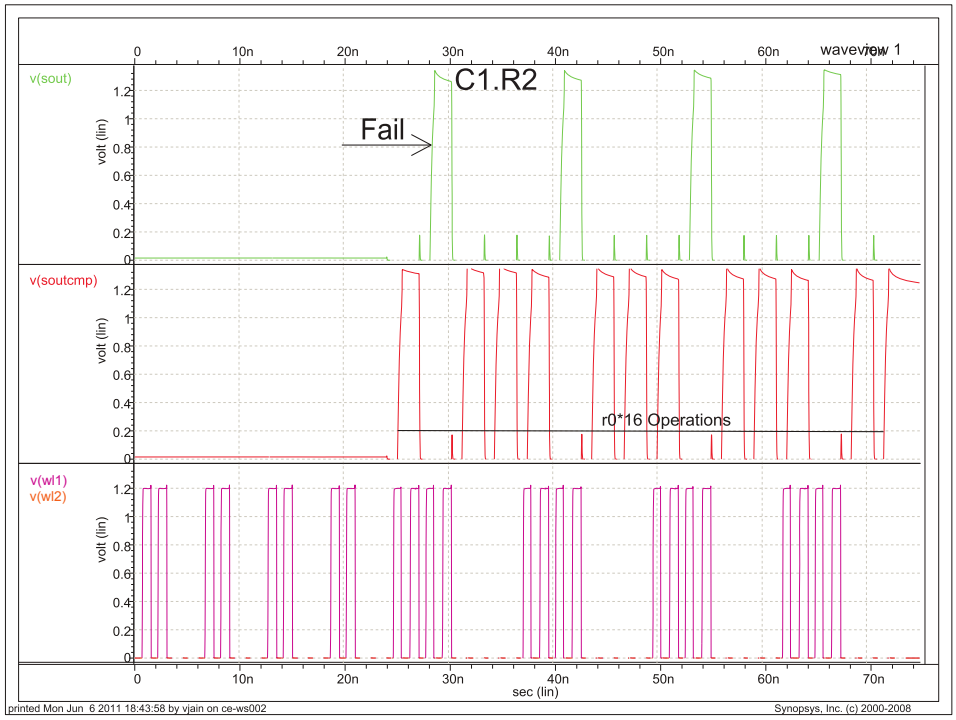

Figure 6.14: HSpice simulation: March AFr0up

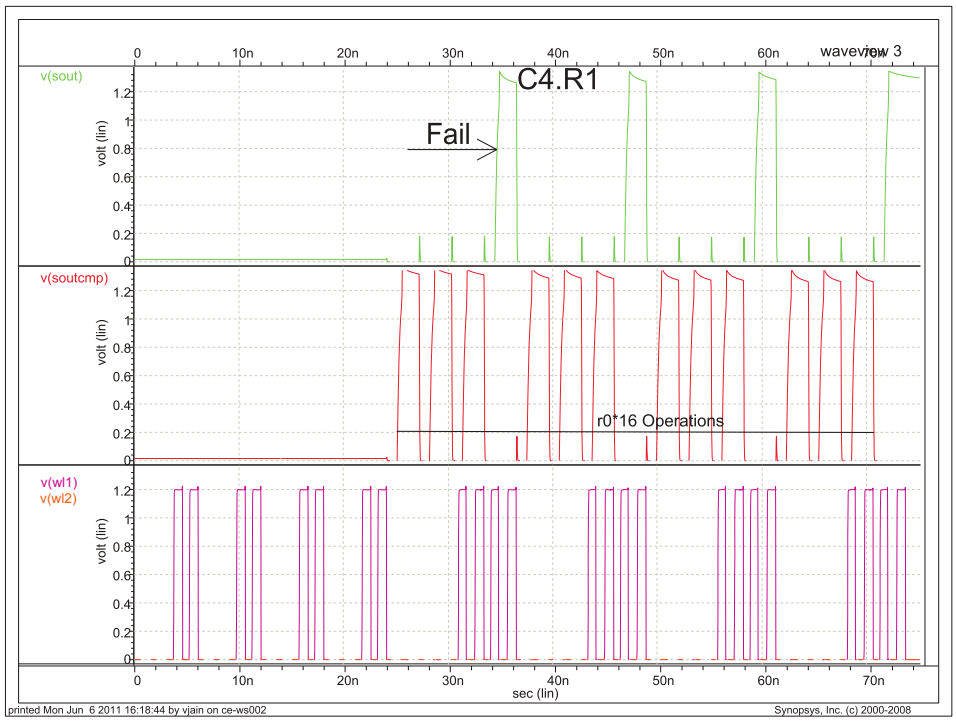

Figure 6.15: HSpice simulation: March AFr0down

for simulation we are using up addressing here. Second, each cell is read and thereafter written with 0 by the second march element $\Uparrow (r1, w0)$. As the figure shows, the r1 operation applied to the cell labeled as C1.R2 fails. Other cells accessed by address lines for row 2 (C2.R2, C3.R2 and, C4.R2) also fail for the r1 operation. In conclusion, March AFr1up has its first failing address at C1.R2.

The fault detecting read operation is marked in bold.
March AFr1up: $\{ \Updownarrow (w1); \Uparrow (\mathbf{r1}, w0) \}$

March AFr1down: The simulation result of this test is shown in Figure 6.17.

This test is applied using down addressing with fast-row address direction. First, all the 16 cells are initialized to 1 by the first march element $\Updownarrow (w1)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using down addressing here. Second, each cell is read and thereafter written with 0 by the second march element $\Downarrow (r1, w0)$. As the figure shows, the r1 operation applied to the cell labeled as C4.R1 fails. Other cells accessed by address lines for row 1 (C3.R1, C2.R1 and, C1.R1) also fail for the r1 operation. In conclusion, March AFr1down has its first failing address at C4.R1.

The fault detecting read operation is marked in bold.
March AFr1down: $\{ \Updownarrow (w1); \Downarrow (\mathbf{r1}, w0) \}$

It is to be observed from provided simulation waveforms that WL2 is never active and WL1 is active in its place. This is because of the injected defect, which demonstrates the fault behavior of address decoder fault AF_{nma}. All the applied tests fail generating a diagnostic signature "1 1 1 1". Here, we do not require the information of failing addresses for identifying the fault type. But if the failing addresses are known, the fault can be pinpointed to particular row(s) or column(s) of the address decoder.

Table 6.4 shows comparison of the obtained signature with level 2 diagnostic dictionary (see Table 5.9). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as address decoder fault AF_{nma}. Also, as the applied tests observe first failing addresses (C1.R2, C4.R1, C1.R2, and C4.R1) at the opposite ends of the memory cell rows, the fault can be classified as a row decoder fault in row 1 and 2 of the address decoder. The injected defect causes address decoder fault AF_{nma} in the row decoder; thus the correct fault type is indicated by the diagnosis process.

Table 6.4: Level 2: Fault in the address decoder identified as AF_{nma}

Fault	Signature	AFr0up	AFr0down	AFr1up	AFr1down
AF_{nma}	Dictionary Signature	1	1	1	1
	Observed Signature	1	1	1	1

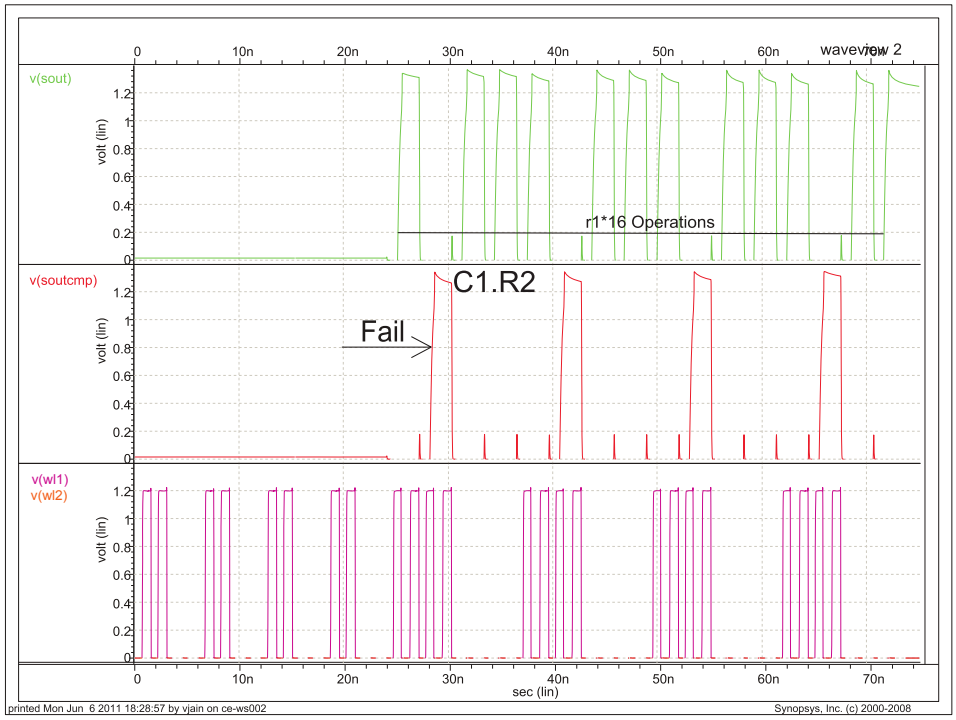

Figure 6.16: HSpice simulation: March AFr1up

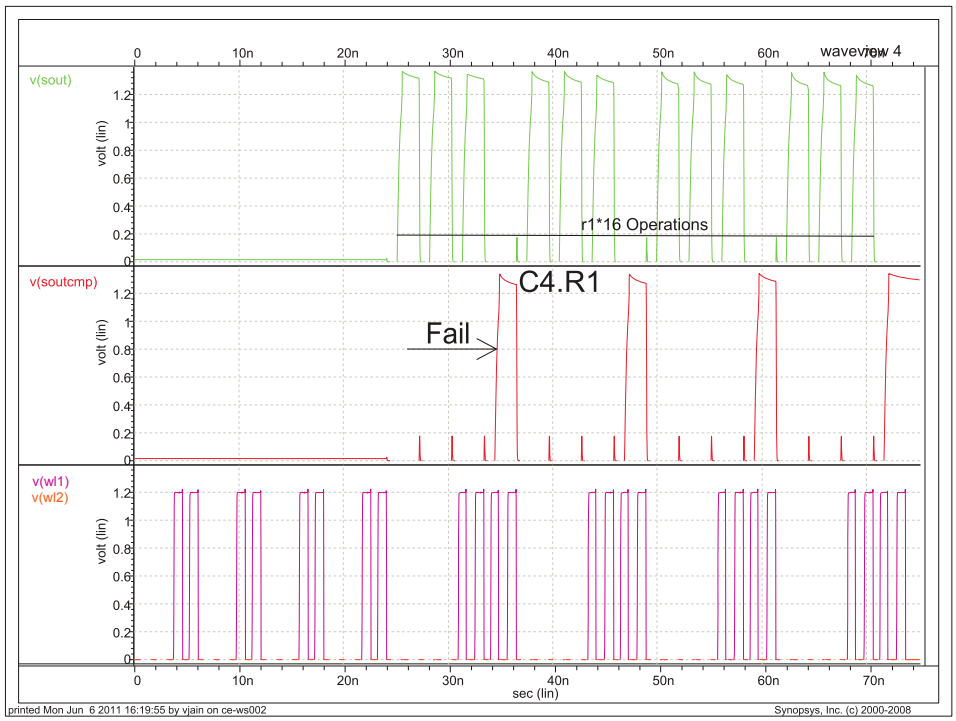

Figure 6.17: HSpice simulation: March AFr1down

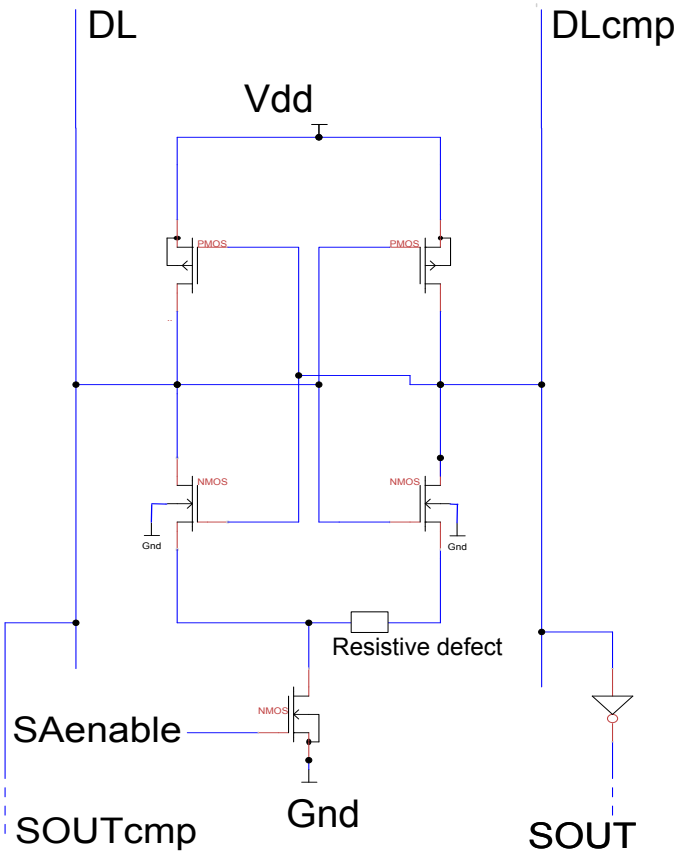

Figure 6.18: Resistive defect causing static fault in the peripheral circuitry

6.2.3 Diagnosing Faults in Peripheral Circuitry

A resistive defect has been injected in the sense amplifier for column 2 of the memory cell array; see Figure 6.18. This defect causes an incorrect read fault in memory cells connected to the faulty sense amplifier; see Table 3.1.

The diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and its location will be pinpointed. The two levels are discussed next.

Diagnosis Level 1

The TCs developed in Section 5.4.2 will be applied. The procedure consists of the following:

- Apply March MSS-up in normal mode
- Apply March MSS-down in normal mode
- Apply March MSS-up in diagnosis mode

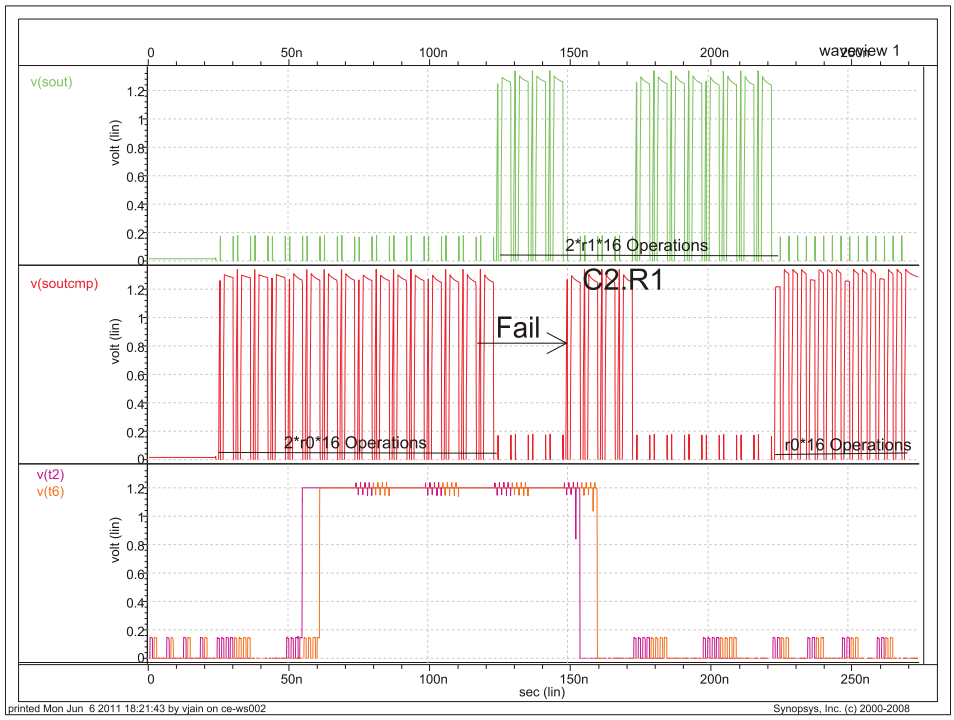

Figure 6.19: HSpice simulation: March MSSm-up in normal mode

The simulation results of the application of the above tests are presented in Figure 6.19, Figure 6.20, and Figure 6.21.

March MSSm-up in normal mode: The simulation result of this test is shown in Figure 6.19. Here, the bottom graph gives the true node voltage for two of the memory cells connected to faulty sense amplifier (denoted as $V(t2)$ and $V(t6)$).

As the figure shows, the r1 operation applied to the cell labeled as C2.R1 fails. Other cells connected to faulty sense amplifier (C2.R2, C2.R3 and, C2.R4) also fail for the r1 operation. It is worth noting that 0 is read from memory cells connected to faulty sense amplifier even when they contain 1. In conclusion, March MSSm-up has its first failing address at C2.R1.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (\mathbf{r1}, r1, w0, w0); \Uparrow (r0, w1) \}$

March MSSm-down in normal mode: The simulation result of this test is shown in Figure 6.20.

As the figure shows, the r1 operation applied to the cell labeled as C2.R4 fails. Other cells connected to faulty sense amplifier (C2.R3, C2.R2 and, C2.R1) also fail for the r1 operation. In conclusion, March MSSm-down has its first failing address at C2.R4.

The fault detecting read operation is marked in bold.
March MSSm-down: $\{ \Downarrow (w0); \Downarrow (r0, r0, w1, w1); \Downarrow (\mathbf{r1}, r1, w0, w0); \Downarrow (r0, w1) \}$

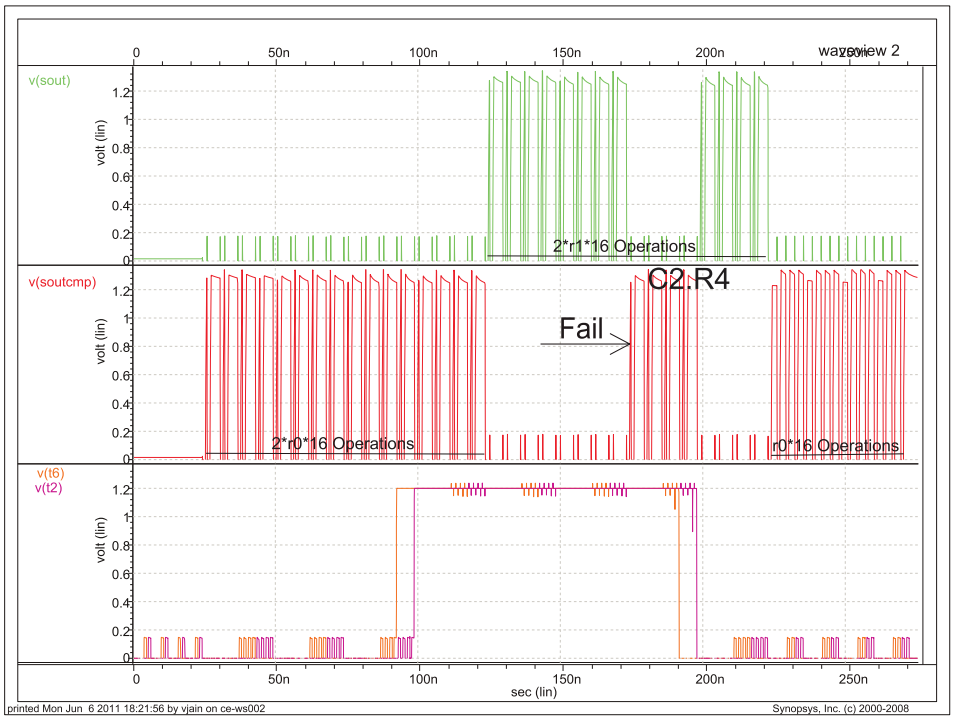

Figure 6.20: HSpice simulation: March MSSm-down in normal mode

March MSSm-up in diagnosis mode: The simulation result of this test is shown in Figure 6.21.

As the figure shows, the r1 operation applied to the cell labeled as C1.R1 fails. Other cells connected to faulty sense amplifier (C1.R2, C1.R3 and, C1.R4) also fail for the r1 operation. In conclusion, March MSSm-up has its first failing address at C1.R1.

It is to be noted that the location of first fail for March MSS-up in diagnosis mode is different from that of March MSSm-up in normal mode. This is because, in diagnosis mode the faulty sense amplifier is accessed by a different address line than in normal mode; hence the fail is observed at different addresses than those in normal mode.

The fault detecting read operation is marked in bold.
March MSSm-up: $\{ \Uparrow (w0); \Uparrow (r0, r0, w1, w1); \Uparrow (\mathbf{r1}, r1, w0, w0); \Uparrow (r0, w1) \}$

It is to be observed from provided simulation waveforms that due to the injected defect, even when the memory cells connected to faulty sense amplifier contain 1, 0 is read by the sense amplifier. All the applied tests fail generating a diagnostic signature "1(C2.R1) 1(C2.R4) 1(C1.R1)".

Table 6.5 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.6). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a fault in the peripheral circuitry. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

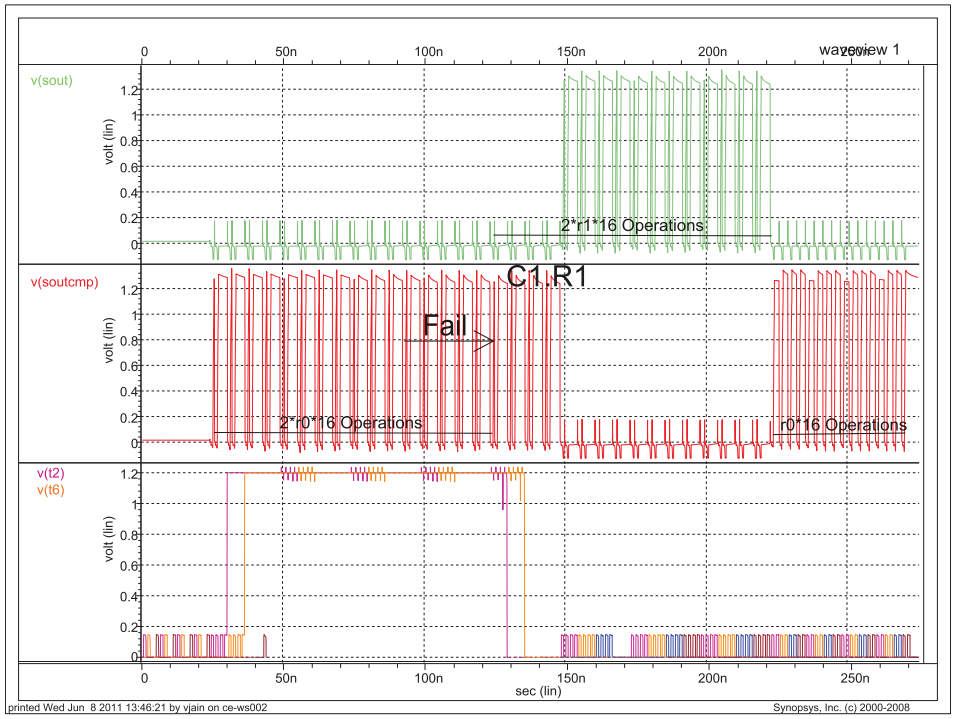

Figure 6.21: HSpice simulation: March MSSm-up in diagnosis mode

Table 6.5: Level 1: Peripheral circuitry identified as the faulty block

Faulty Block	Signature	MSSm-up in normal mode (failing address)	MSSm-down in normal mode (failing address)	MSSm-up in diagnosis mode (failing address)	Failing address
Peripheral circuitry	Dictionary signature	1 (A_α)	1 (A_β)	1 (A_γ)	$\alpha \neq \beta \neq \gamma$
	Observed signature	1 (C2.R1)	1 (C2.R4)	1 (C1.R1)	

Diagnosis Level 2

At level 1, the faulty block has been identified as the peripheral circuitry. At level 2, the fault type will be identified. As explained in Section 5.4.3, when the peripheral circuitry is faulty, we need to use DfD hardware and apply new diagnostic tests to identify the type of fault. It is worth mentioning again, that for the diagnosis of peripheral circuitry at level 2, diagnosis mode alters the write path for every column of the memory i.e., in diagnosis mode every column of the memory cell array connects to the write path circuitry of the adjacent column but the read path remains the same. The TP developed in Section 5.4.3 will be applied. Only one TP is required:

- **Memory Scan test**: $\{ \Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1); \Updownarrow (r1) \}$

The procedure consists of the following:

- Apply Memory Scan test in normal mode
- Apply Memory Scan test in diagnosis mode

Memory Scan test in normal mode: The simulation result of this test is shown in Figure 6.22. Here, the bottom graph gives the true node voltage for two of the memory cells connected to faulty sense amplifier (denoted as $V(t2)$ and $V(t6)$).

It is to be noted, that all the march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing with fast-row address direction. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow (w0)$. Second, each cell is read by the second march element $\Updownarrow (r0)$. As can be seen in the figure, all the read operations pass correctly. Third, each cell is written 1 by the third march element $\Updownarrow (w1)$. Then each cell is read by the fourth march element $\Updownarrow (r1)$. As the figure shows, the r1 operation applied to the cell labeled as C2.R1 fails. Other cells connected to faulty sense amplifier (C2.R2, C2.R3 and, C2.R4) also fail for the r1 operation. In conclusion, Memory Scan test has its first failing address at C2.R1.

The fault detecting read operation is marked in bold.
Memory Scan test: $\{ \Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1); \Updownarrow (\mathbf{r1}) \}$

Memory Scan test in diagnosis mode: The simulation result of this test is shown in Figure 6.23.

It is to be noted, that all the march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing with fast-row address direction. The diagnosis mode alters the write path for every memory cell array column and it has no affect on the way cells are addressed. Thus, the addressing remains the same as in normal mode. As the figure shows, the r1 operation applied to the cell labeled as C1.R1 fails. Other cells connected to faulty sense amplifier (C1.R2, C1.R3 and, C1.R4) also fail for the r1 operation. In conclusion, Memory Scan test has its first failing address at C2.R1.

It is to be noted that the failing addresses are same as in the test applied in normal mode. This is because, in diagnosis mode the write path for every memory cell column is changed but the read path remains the same; as the fault lies in the sense amplifiers i.e., the read path, Memory Scan test fails for the same address locations.

The fault detecting read operation is marked in bold.
Memory Scan test: $\{ \Updownarrow (w0); \Updownarrow (r0); \Updownarrow (w1); \Updownarrow (\mathbf{r1}) \}$

It is to be observed from provided simulation waveforms that due to the injected defect, even when the memory cells connected to faulty sense amplifier contain 1, 0 is read by the sense amplifier. All the applied tests fail generating a diagnostic signature "1 1".

Table 6.6 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.10). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a read path fault. The injected defect causes a faulty sense amplifier; thus the correct fault type is indicated by the diagnosis process.

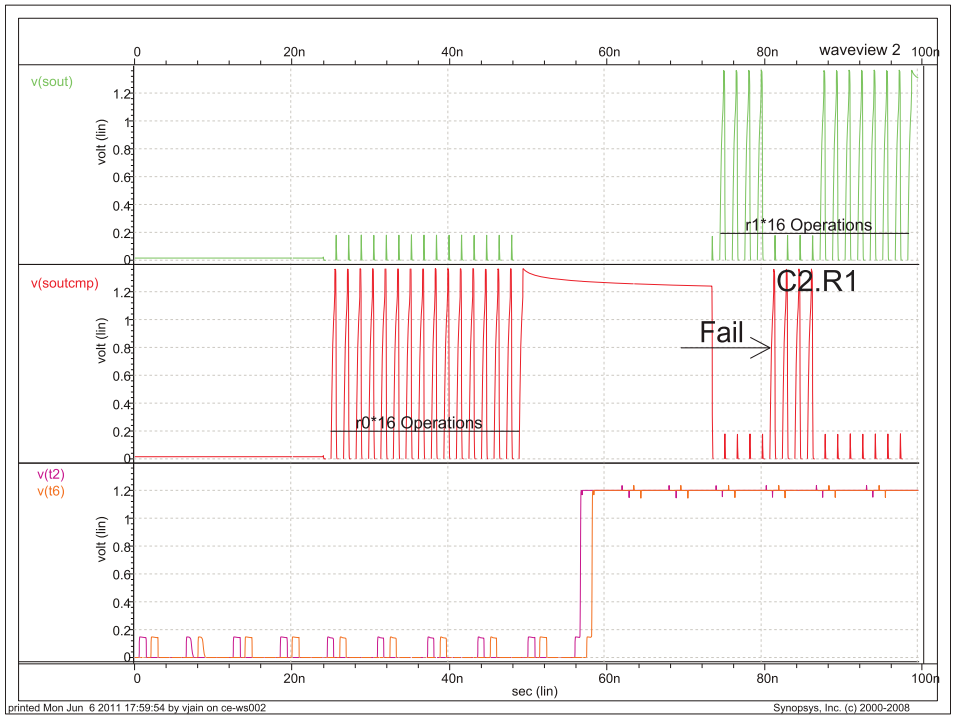

Figure 6.22: HSpice simulation: Memory SCAN test in normal mode

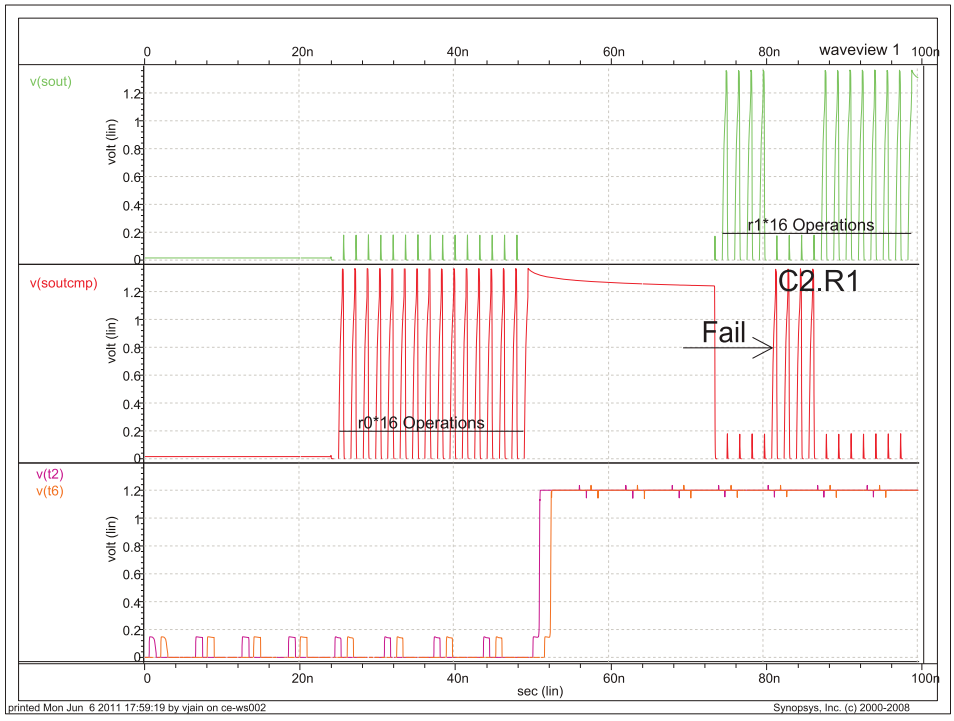

Figure 6.23: HSpice simulation: Memory SCAN test in diagnosis mode

Table 6.6: Level 2: Fault in the peripheral circuitry identified as read path fault

Faulty Block	Signature	Memory SCAN test in normal mode (failing address)	Memory SCAN test in diagnosis mode (failing address)
Read path	Dictionary signature	1 (A_α)	1 (A_α)
	Observed signature	1 (C2.R1)	1 (C2.R1)

6.3 Dynamic Hierarchical Analysis

This section presents the simulation results for diagnosing dynamic faults occurring in the memory. A single defect causing a dynamic fault is injected in the memory model at a time. Three experiments have been performed:

- Inject a defect in the memory cell array
- Inject a defect in the address decoder
- Inject a defect in the peripheral circuitry

In each of the experiment, the HMD procedure is performed and the defective memory block as well as the fault have been identified. In the rest of this section, the three experiments will be discussed and the results will be reported.

6.3.1 Diagnosing Faults in Memory Cell Array

A resistive defect has been injected in the memory cell at address C4.R1. The defect is injected between the between the vdd and power supply of the cell as shown in Figure 6.24. The resistance of the defect is taken to be in medium range to make sure that it causes a dynamic fault. Such a defect causes a dynamic deceptive read destructive fault ¡1r1r1/0/1¿; see Table 3.5.

As mentioned in Section 5.1, the diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and its location will be pinpointed. The two levels are discussed next.

Diagnosis Level 1

The TCs developed in Section 5.5.2 will be applied. They consist of:

- **March MD2$_y$**:
  $\{ \Updownarrow_y (w0); \Uparrow_y (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1);$
  $\Uparrow_y (r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0);$
  $\Downarrow_y (r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1);$
  $\Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$

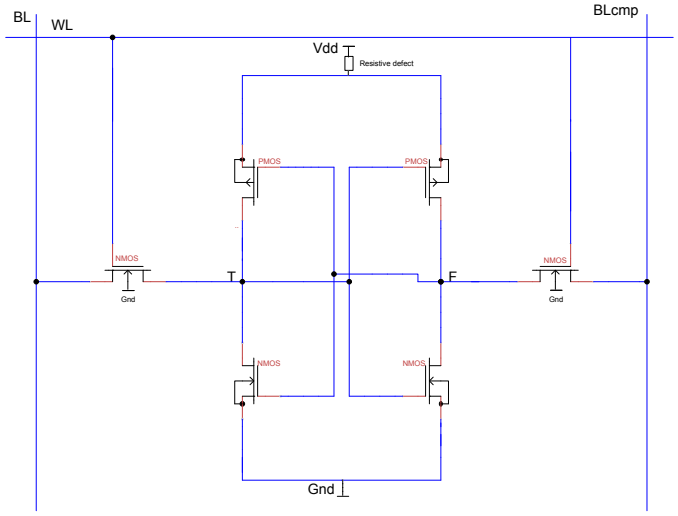

Figure 6.24: Resistive defect causing dynamic fault in the memory cell array

- **March RAWAW-H1m$_x$**: $\{ \Updownarrow_x (w0); \Updownarrow_x^{H1} (w0_g, w1_f, r0_g, r1_f, w0_f); \Updownarrow_x (w1); \Updownarrow_x^{H1} (w1_g, w0_f, r1_g, r0_f, w1_f) \}$
- **March RAWAW-H1m$_y$**: $\{ \Updownarrow_y (w0); \Updownarrow_y^{H1} (w0_g, w1_f, r0_g, r1_f, w0_f); \Updownarrow_y (w1); \Updownarrow_y^{H1} (w1_g, w0_f, r1_g, r0_f, w1_f) \}$

The procedure consists of the following:

- Apply March MD2$_y$
- Apply March RAWAW-H1m$_x$ separately on each column of the memory cell array
- Apply March RAWAW-H1m$_y$ separately on each row of the memory cell array

The simulation results of the application of the above tests are presented in Figure 6.25, Figure 6.26, and Figure 6.27.

March MD2$_y$: The simulation result of this test is shown in Figure 6.25. The top and middle graph in the figure give the outputs of the sense amplifier $V(sout)$ and its complement $\overline{V(sout)}$ (denoted as $V(soutcmp)$), respectively, while the bottom graph gives the true node voltage of the faulty cell (denoted as $V(t4)$).

The test is applied to the memory in fast-column direction. Each operation of the test is applied to all 16 cells considered in the simulation model. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow_y (w0)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using up addressing here. As we are using up addressing with fast-column address direction, the order in which the cells are accessed is: C1.R1, C2.R1, C3.R1...C3.R4, C4.R4. Second, each cell is read and thereafter more read/write operations are performed as given in the second march element

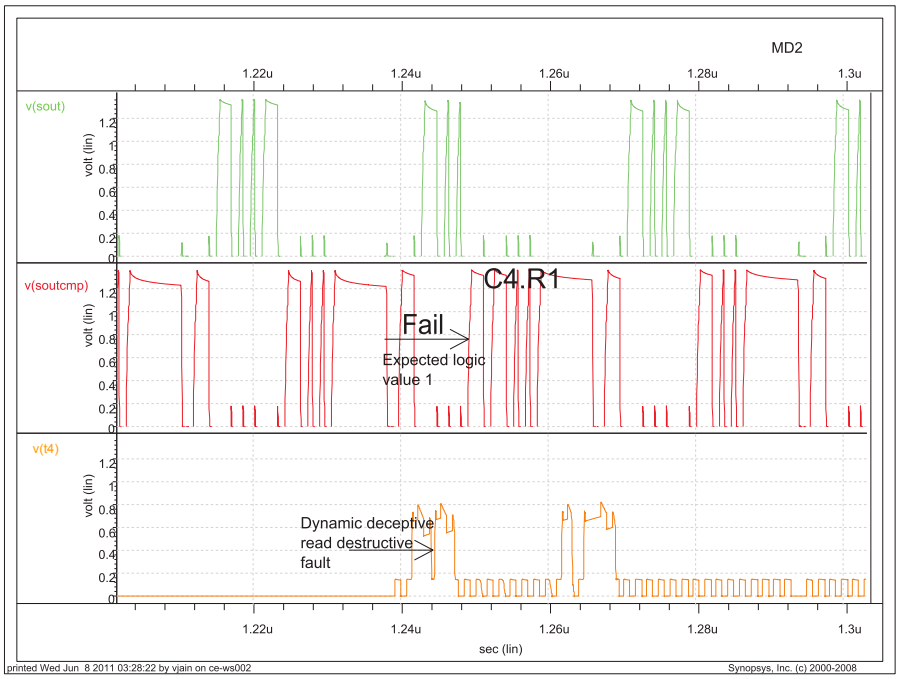

Figure 6.25: HSpice simulation: March MD2$_y$

$\Uparrow_y$ $(r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1)$ and the third march element $\Uparrow_y$ $(r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0))$. All the read operations pass correctly. Then the fourth march element $\Downarrow_y$ $(r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1)$ is applied in down addressing; the order in which the cells are accessed is: C4.R4, C3.R4, C2.R4...C2.R1, C1.R1. Figure 6.25 shows the simulation result when fourth march element is applied. As it can be seen, the r1 operation applied to the cell labeled as C4.R1 fails. Further read operations are then performed by fifth march element $\Downarrow_y$ $(r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0)$ and finally every cell is read by the sixth march element $\Updownarrow_y$ $(r0)$. It is worth noting that, the true node of the faulty cell shown in the bottom graph is flipped on application of consecutive read operations. In conclusion, March MD2$_y$ fails only once at C4.R1.

The fault detecting read operation of the fourth march element is marked in bold.
March element 4: $\Downarrow_y$ $(r0, w1, r1, w1, r1, r1, \mathbf{r1}, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1)$

March RAWAW-H1m$_x$: The simulation result of this test is shown in Figure 6.26. As explained in 5.5.2, to avoid column decoder transitions March RAWAW-H1m$_x$ is applied separately to every column of the memory i.e., first it is applied to column 1 and the pass/fail status of the test is stored, then the test is applied to column 2 and so on. Diagnostic signature will include logical AND and logical OR of the pass/fail status of March RAWAW-H1m$_x$ applied on each column. The first four graphs in the figure give the outputs of the sense amplifier ($V(sout)$ and $V(soutcmp)$), for the four memory cell columns (column 1, column 2, column 3 and, column 4 respectively), while

the bottom graph gives the true node voltage of the faulty cell (denoted as $V(t4)$).

For the march RAWAW-H1m$_x$, special addressing mode called H1 addressing is used. For accessing a column with four rows using H1 addressing method, the row address lines should include the following transitions: 00 → 01 → 00, 00 → 10 → 00, 11 → 01 → 11, and 11 → 10 → 11. For the used memory model, row address lines 00 signify row 1, 10 signify row 2, 01 signify row 3 and, 11 signify row 4. Thus, the transitions should be: R1 → R3 → R1, R1 → R2 → R1, R4 → R3 → R4, and R4 → R2 → R4. H1 addressing used Sensitizing Operation Pairs (SOP) and Sensitizing Operation Triplets (SOT) for accessing the memory. SOPs and SOTs are explained in detail in [22]. Read write operations are performed using addresses triplets "Ag, Af, Ag; where the first and the third addresses are identical; for e.g., "C1.R1 C1.R3 C1.R1". Next, the simulation result for application of March RAWAW-H1m$_x$ to column 1 will be explained.

Each operation of the test is applied to all 4 cells of column 1 (C1.R1, C1.R2, C1.R3, C1.R4). It is to be noted, that all march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing here. First, all the 4 cells are initialized to 0 by the first march element $\Updownarrow_x$ $(w0)$. Then the second march element is applied in H1 addressing and so the read/write operations will be applied using the above explained row address transitions. For the second march element $\Updownarrow_x^{H1}$ $(w0_g, w1_f, r0_g, r1_f, w0_f)$, initially g will assume the location corresponding to address lines 00 i.e., cell C1.R1 and f will assume the location corresponding to address lines 01, i.e., cell C1.R3. So, C1.R1 is written with 0, then C1.R3 is written to 1; then both the cells are read and then C1.R1 is written with 0. This gives us address transitions R1 → R3 → R1. For the next address transition, g will assume the location corresponding to address lines 00 i.e., cell C1.R1 and f will assume the location corresponding to address lines 10, i.e., cell C1.R2 and the process is repeated until all the required row address transitions are performed. The top graph in Figure 6.26 shows the results of cell read operations for the first column of the memory. All the read operations pass correctly; i.e., $V(sout) = 0$ and $V(soutcmp) = 1$ for r0 operations and $V(sout) = 1$ and $V(soutcmp) = 0$ for r1 operations. After the completion of second march element, all the 4 cells are written with 1 by the third march element $\Updownarrow_x$ $(w1)$. Then the fourth march element $\Updownarrow_x^{H1}$ $(w1_g, w0_f, r1_g, r0_f, w1_f)$ is applied in H1 addressing and follows the same addressing order as for second march element. As can be seen in the figure, all the read operations pass correctly.

In a similar way, March RAWAW-H1m$_x$ is applied for column 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.26. As can be seen, all the read operations pass correctly and no failure is detected.

March RAWAW-H1m$_y$: The simulation result of this test is shown in Figure 6.27. As explained in 5.5.2, to avoid row decoder transitions March RAWAW-H1m$_y$ is applied separately to every row of the memory i.e., first it is applied to row 1 and the pass/fail status of the test is stored, then the test is applied to row 2 and so on. The first four graphs in the figure give the outputs of the sense amplifier ($V(sout)$ and $V(soutcmp)$), for the four memory cell rows (row 1, row 2, row 3 and, row 4 respec-

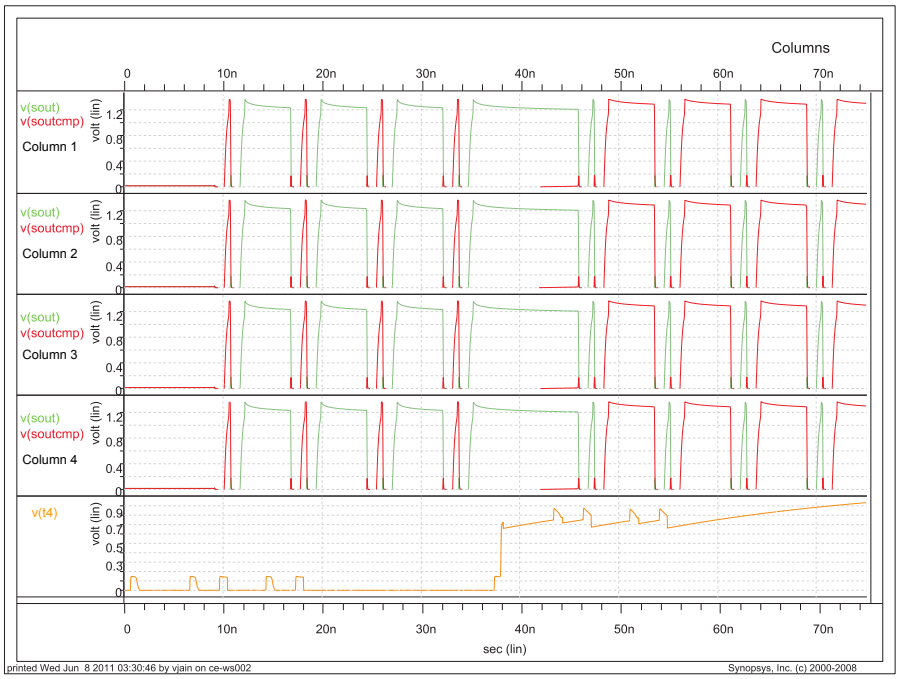

Figure 6.26: HSpice simulation: March RAWAW-H1m$_x$ applied for each row of the memory cell array

tively), while the bottom graph gives the true node voltage of the faulty cell (denoted as $V(t4)$).

For the march RAWAW-H1m$_y$, special addressing mode called H1 addressing is used. For accessing a row with four columns using H1 addressing method, the column address lines should include the following transitions: 00 $\rightarrow$ 01 $\rightarrow$ 00, 00 $\rightarrow$ 10 $\rightarrow$ 00, 11 $\rightarrow$ 01 $\rightarrow$ 11, and 11 $\rightarrow$ 10 $\rightarrow$ 11. For the used memory model, column address lines 00 signify column 1, 10 signify column 2, 01 signify column 3 and, 11 signify column 4. Thus, the transitions should be: C1 $\rightarrow$ C3 $\rightarrow$ C1, C1 $\rightarrow$ C2 $\rightarrow$ C1, C4 $\rightarrow$ C3 $\rightarrow$ C4, and C4 $\rightarrow$ C2 $\rightarrow$ C4. Next, the simulation result for application of March RAWAW-H1m$_y$ to row 1 will be explained.

Each operation of the test is applied to all 4 cells of row 1 (C1.R1, C2.R1, C3.R1, C4.R1). It is to be noted, that all march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing here. First, all the 4 cells are initialized to 0 by the first march element $\Updownarrow_y$ $(w0)$. Then the second march element is applied in H1 addressing and so the read/write operations will be applied using the above explained column address transitions. For the second march element $\Updownarrow_y^{H1}$ $(w0_g, w1_f, r0_g, r1_f, w0_f)$, initially g will assume the location corresponding to address lines 00 i.e., cell C1.R1 and f will assume the location corresponding to address lines 01, i.e., cell C3.R1. So, C1.R1 is written with 0, then C3.R1 is written to 1; then both the cells are read and then C1.R1 is written with 0. This gives us address transitions C1 $\rightarrow$ C3 $\rightarrow$ C1. For the next address transition, g will assume the location corresponding to address lines 00 i.e., cell C1.R1 and f will assume the location

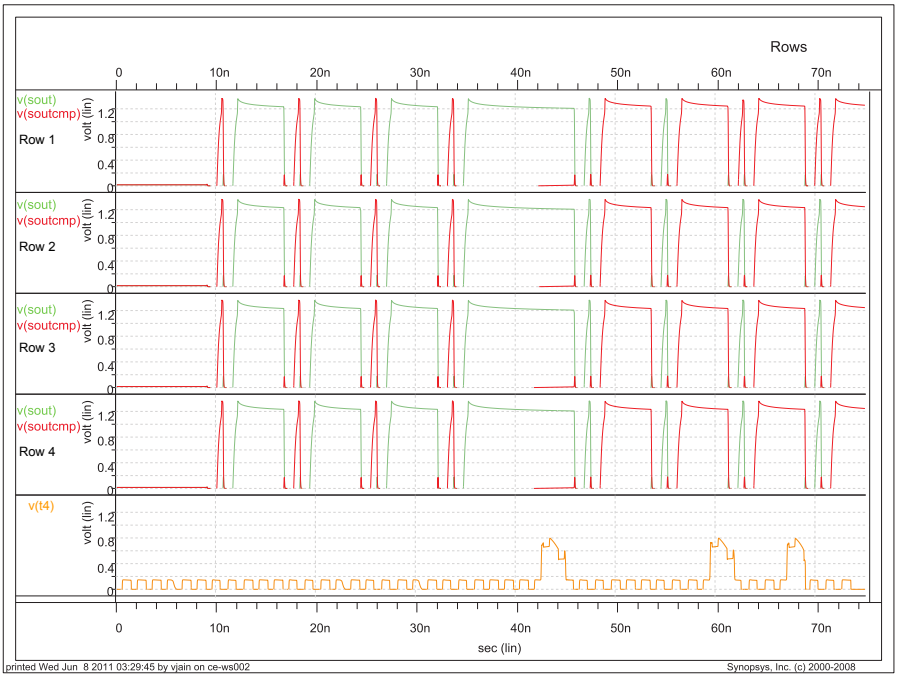

Figure 6.27: HSpice simulation: March RAWAW-H1m$_y$ applied for each row of the memory cell array

corresponding to address lines 10, i.e., cell C2.R1 and the process is repeated until all the required column address transitions are performed. The top graph in Figure 6.27 shows the results of cell read operations for the first row of the memory. All the read operations pass correctly; i.e., $V(sout) = 0$ and $V(soutcmp) = 1$ for r0 operations and $V(sout) = 1$ and $V(soutcmp) = 0$ for r1 operations. After the completion of second march element, all the 4 cells are written with 1 by the third march element $\Updownarrow_y$ $(w1)$. Then the fourth march element $\Updownarrow_y^{H1}$ $(w1_g, w0_f, r1_g, r0_f, w1_f)$ is applied in H1 addressing and follows the same addressing order as for second march element. As can be seen in the figure, all the read operations pass correctly.

In a similar way, March RAWAW-H1m$_y$ is applied for row 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.27. As can be seen, all the read operations pass correctly and no failure is detected.

It is to be observed from provided simulation waveforms that due to the injected defect, despite of several write 1 operations, the true node of the faulty cell shown in the bottom graph either does not contain a strong 1 value or is pulled up very slowly. Of all the applied tests, only March MD2$_y$ fails generating a diagnostic signature "1 0 0".

Table 6.7 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.11). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a memory cell array fault. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

Table 6.7: Level 1: Memory cell array identified as the faulty block

Faulty Block	Signature	March $MD2_y$ (AND)	March RAWAW -H1m$_x$(AND)	March RAWAW -H1m$_x$(OR)	March RAWAW -H1m$_y$(AND)
Memory cell array	Dictionary signature	1	0	0	0
	Observed signature	1	0	0	0

Diagnosis Level 2

At level 1, the faulty block has been identified as the memory cell array. At level 2, the fault type will be identified. This will be done using tests developed in Section 5.5.3. They consist of:

- **March MD2**$_1$: $\{ \Updownarrow_y (w0); \Uparrow_y (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1); \Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$
- **March MD2**$_2$: $\{ \Updownarrow_y (w0); \Downarrow_y (r0, w1, r1, w1, r1, r1, r1, w0, r0, w0, r0, r0, r0, w0, w1, w0, w1); \Uparrow_y (r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0); \Updownarrow_y (r0) \}$

The procedure consists of the following:

- Apply March MD2$_1$
- Apply March MD2$_2$

The simulation results of the application of the above tests are presented in Figure 6.28 and Figure 6.29.

March MD2$_1$: The simulation result of this test is shown in Figure 6.28. Here, the bottom graph gives the true node voltage of the faulty cell (denoted as $V(t4)$).

The test is applied to the memory in fast-column direction. Each operation of the test is applied to all 16 cells considered in the simulation model. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow_y (w0)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using up addressing here. Second, each cell is read and thereafter more read/write operations are performed as given in the second march element $\Uparrow_y (r0, w1, w1, r1, w1, w1, r1, w0, w0, r0, w0, w0, r0, w0, w1, w0, w1)$. Then the third march element $\Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, r1, w1, w0, w1, w0)$ is applied in down addressing. Figure 6.28 shows the simulation result when third march element is applied. As it can be seen, the r1 operation applied to the cell labeled as C4.R1 fails. Finally, every cell is read by the fourth march element $\Updownarrow_y (r0)$.

The fault detecting read operation of the fourth march element is marked in bold.
March element 3: $\Downarrow_y (r1, w0, r0, w0, r0, r0, r0, w1, r1, w1, r1, r1, \mathbf{r1}, w1, w0, w1, w0)$

Figure 6.28: HSpice simulation: March $MD2_1$

March $MD2_2$: The simulation result of this test is shown in Figure 6.29.

The test is applied to the memory in fast-column direction. Each operation of the test is applied to all 16 cells considered in the simulation model. First, all the 16 cells are initialized to 0 by the first march element $\Updownarrow_y$ $(w0)$. It is to be noted, that the first march element can be applied using either up addressing or down addressing; for simulation we are using up addressing here. Then the second march element $\Downarrow_y$ $(r0, w1, r1, w1, r1, w1, r1, w0, r0, w0, r0, w0, r0, w0, w1, w0, w1)$ is applied in down addressing. Figure 6.29 shows the simulation result when second march element is applied. As it can be seen, the r1 operation applied to the cell labeled as C4.R1 fails. Then the third march element $\Uparrow_y$ $(r1, w0, w0, r0, w0, w0, r0, w1, w1, r1, w1, w1, r1, w1, w0, w1, w0)$ is applied in up addressing and finally every cell is read by the fourth march element $\Updownarrow_y$ $(r0)$.

The fault detecting read operation of the fourth march element is marked in bold.
March element 2: $\Downarrow_y$ $(r0, w1, r1, w1, r1, w1, \mathbf{r1}, w0, r0, w0, r0, w0, r0, w0, w1, w0, w1)$

All the applied tests fail generating a diagnostic signature "1 1".

Table 6.8 shows comparison of the obtained signature with level 2 diagnostic dictionary (see Table 5.12). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as single-cell fault in the memory cell array. Also, the fault location is known to be at C4.R1 as the first failing address is same (C4.R1) for both the tests. The injected defect causes a dynamic deceptive read destructive fault

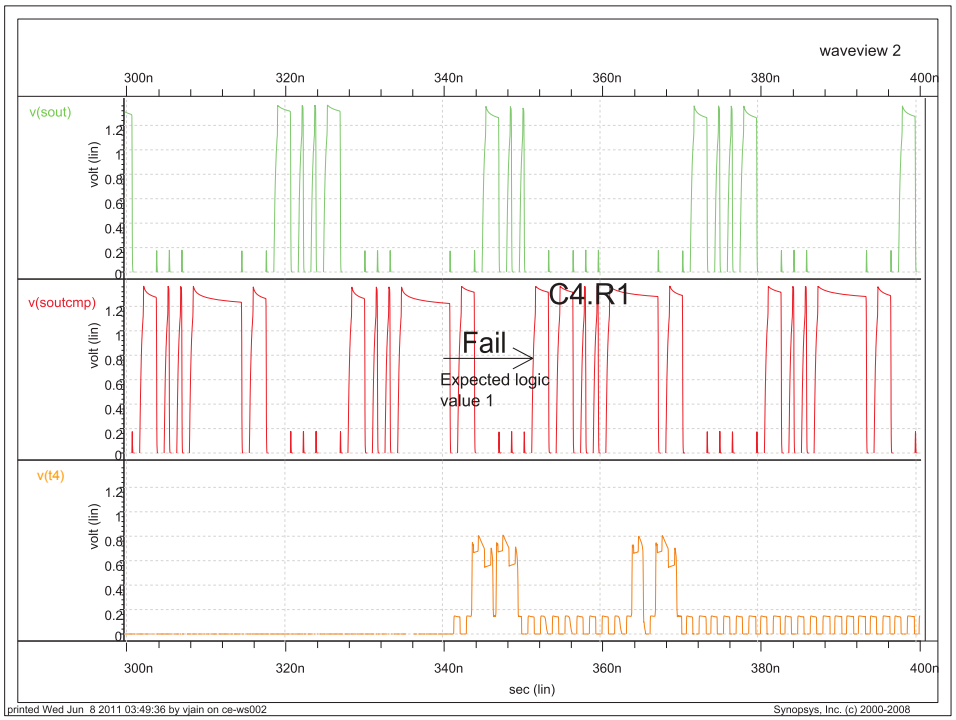

Figure 6.29: HSpice simulation: March $MD2_2$

in a single cell located at C4.R1; thus the correct fault type is indicated by the diagnosis process.

Table 6.8: Level 2: Fault in the memory cell array identified as single-cell fault

Faulty Block	Signature	March $MD2_1$	March $MD2_2$
Memory cell array	Dictionary signature	1	1
	Observed signature	1	1

6.3.2 Diagnosing Faults in Address Decoder

An inter-gate resistive defect has been injected in the row decoder that causes activation delay fault in WL1 (corresponding to row 1). The defect is injected at the input of the NAND gate controlling WL1; as shown in Figure 6.30. The resistance of the defect is taken to be in medium range to make sure that it causes a delay fault.

The diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and its location will be pinpointed. The two levels are discussed next.

The procedure consists of the following:

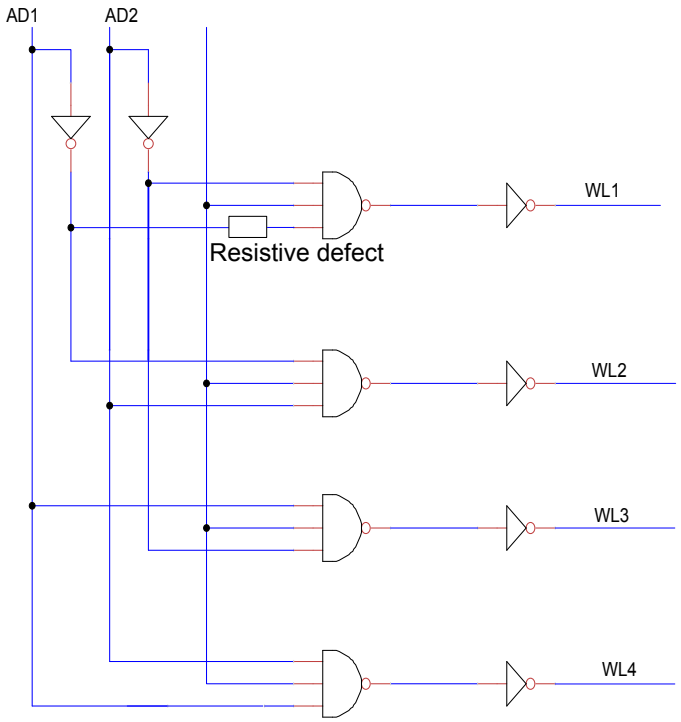

Figure 6.30: Resistive defect causing dynamic fault in the address decoder

- Apply March MD2$_y$
- Apply March RAWAW-H1m$_x$ separately on each column of the memory cell array
- Apply March RAWAW-H1m$_y$ separately on each row of the memory cell array

The simulation results of the application of the above tests are presented in Figure 6.31 and Figure 6.32.

March MD2$_y$: As can be seen from Table 5.11, the pass/fail status of March MD2$_y$ is irrelevant for diagnosing a fault in the address decoder. Hence, the simulation result for application of March MD2$_y$ is not presented here.

March RAWAW-H1m$_x$: The simulation result of this test is shown in Figure 6.31. Here, the bottom graph gives the voltage of the faulty word line (denoted as $V(wl1)$).

March RAWAW-H1m$_x$ is applied for each column of the memory. The address transitions for H1 addressing have been explained above. Considering the application of test for column 1 of the memory, the top graph in Figure 6.31 shows the $V(sout)$ and $V(soutcmp)$ signal of the cell read operations for the first column of the memory. As can be seen from the graph, r1 operation in fourth march element $\Updownarrow_x^{H1}$ $(w1_g, w0_f, r1_g, r0_f, w1_f)$ fails when applied to the cell labeled as C1.R1

In a similar way, March RAWAW-H1m$_x$ is applied for column 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.31.

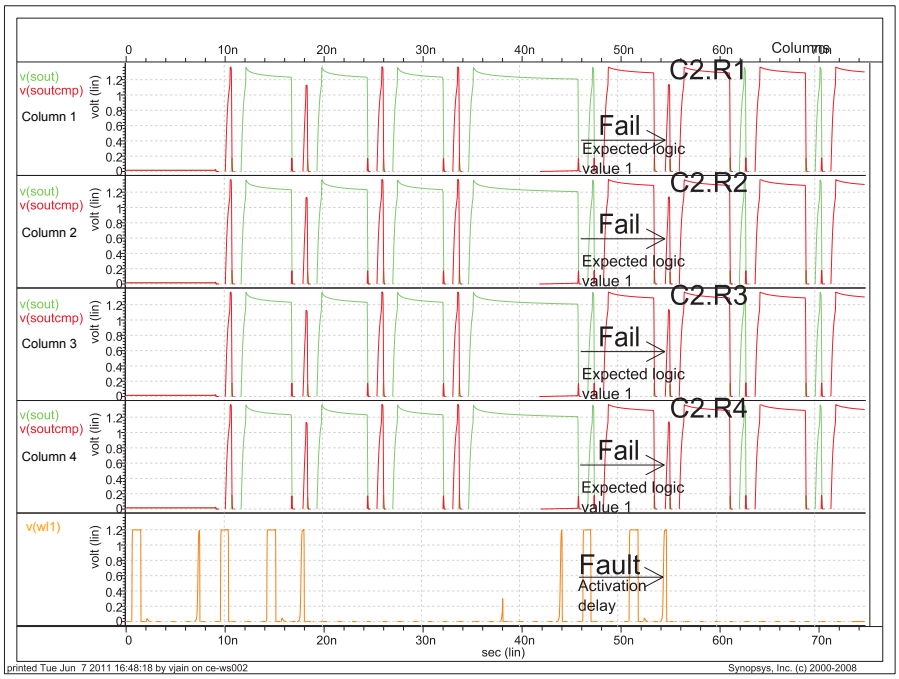

Figure 6.31: HSpice simulation: March RAWAW-H1m$_x$ applied for each row of the memory cell array

Fail can be observed at cells labeled as C2.R1, C3.R1 and C4.R1. It is to be noted from the bottom graph, that activation delay fault is present in WL1 due to the injected defect. In conclusion, March RAWAW-H1m$_x$ fails for all the columns of the memory cell array.

The fault detecting read operation of the fourth march element is marked in bold.
March RAWAW-H1m$_x$: $\Updownarrow_x^{H1}\ (w1_g, w0_f, \mathbf{r1_g}, r0_f, w1_f)$

March RAWAW-H1m$_y$: The simulation result of this test is shown in Figure 6.32.

March RAWAW-H1m$_y$ is applied for each row of the memory. The address transitions for H1 addressing have been explained above. Considering the application of test for row 1 of the memory, the top graph in Figure 6.32 shows the $V(sout)$ and $V(soutcmp)$ signal of the cell read operations for the first row of the memory. As can be seen in the figure, all the read operations pass correctly.

In a similar way, March RAWAW-H1m$_y$ is applied for row 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.32. As can be seen, all the read operations pass correctly and no failure is detected.

It is to be observed from provided simulation waveforms that activation delay fault is present in WL1 due to the injected defect. The pass/fail status of March MD2$_y$ is irrelevant here, and March RAWAW-H1m$_x$ fails for all the columns.

Table 6.9 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.11). On comparing the obtained signature with the diagnostic

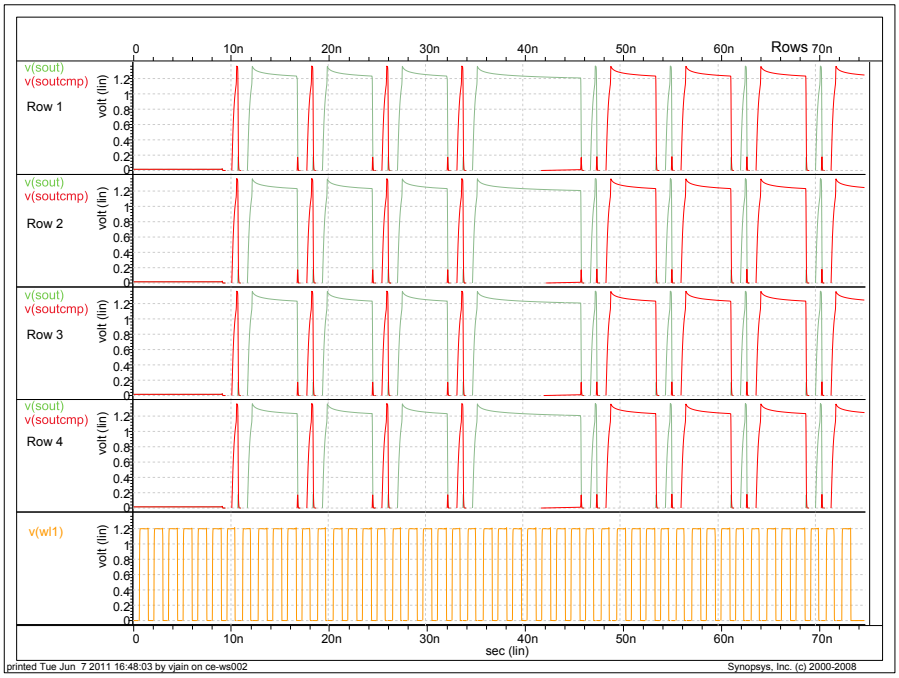

Figure 6.32: HSpice simulation: March RAWAW-H1m$_y$ applied for each row of the memory cell array

Table 6.9: Level 1: Address decoder identified as the faulty block

Faulty Block	Signature	March MD2$_y$ (AND)	March RAWAW -H1m$_x$(AND)	March RAWAW -H1m$_x$(OR)	March RAWAW -H1m$_y$(AND)
Address decoder	Dictionary signature	- -	1 0	1 0	0 1
	Observed signature	1	1	1	0

dictionary, the fault can be classified as a address decoder fault. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

Diagnosis Level 2

At level 1, the faulty block has been identified as the address decoder. At level 2, the fault type will be identified. As explained in Section 5.5.3, when the address decoder is faulty, we do not need to apply any new diagnostic tests at level 2. Further classification can be made on the basis of pass/fail status of tests applied at level 1. This will be done using the results of:

- **March RAWAW-H1m$_x$**
- **March RAWAW-H1m$_y$**

Table 6.10: Level 2: Fault in the address decoder identified as the row decoder fault

Fault	Signature	March RAWAW-H1m$_x$ (AND)	March RAWAW-H1m$_y$ (AND)
Row decoder	Dictionary signature	1	0
	Observed signature	1	0

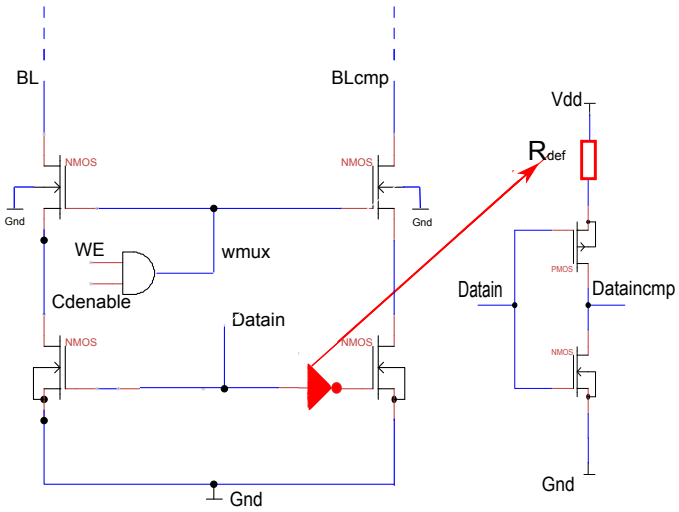

Figure 6.33: Resistive defect causing dynamic fault in the peripheral circuitry

The simulation results of the above tests are presented in Figure 6.31 and Figure 6.32. The results have already been explained above during the discussion of diagnosis level 1.

Table 6.10 shows comparison of the obtained signature with level 2 diagnostic dictionary (see Table 5.13). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as row decoder fault. The injected defect causes an activation delay fault in the row decoder; thus the correct fault type is indicated by the diagnosis process.

6.3.3 Diagnosing Faults in Peripheral Circuitry

A resistive defect has been injected the write driver for column 2 causing a slow write driver fault. The defect is injected at the input of the inverter used for complementing data-in value; as shown in Figure 6.33. The resistance of the defect is taken to be in medium range to make sure that it slows down the peripheral circuitry.

The diagnosis procedure consists of two levels of hierarchy. In the first level, the faulty memory block will be identified, and in the second level, the fault primitive and

its location will be pinpointed. The two levels are discussed next.

The procedure consists of the following:

- Apply March MD2$_y$
- Apply March RAWAW-H1m$_x$ separately on each column of the memory cell array
- Apply March RAWAW-H1m$_y$ separately on each row of the memory cell array

The simulation results of the application of the above tests are presented in Figure 6.34 and Figure 6.35.

March MD2$_y$: As can be seen from Table 5.11, the pass/fail status of March MD2$_y$ is irrelevant for diagnosing a fault in the peripheral circuitry. Hence, the simulation result for application of March MD2$_y$ is not presented here.

March RAWAW-H1m$_x$: The simulation result of this test is shown in Figure 6.34. Here, the bottom graph gives the true node voltage of tow of the memory cells connected to faulty write driver (denoted as $V(t2)$ and $V(t6)$).

March RAWAW-H1m$_x$ is applied for each column of the memory. The address transitions for H1 addressing have been explained above. Considering the application of test for column 1 of the memory, the top graph in Figure 6.34 shows the $V(sout)$ and $V(soutcmp)$ signal of the cell read operations for the first column of the memory. As can be seen in the figure, all the read operations pass correctly.

In a similar way, March RAWAW-H1m$_x$ is applied for column 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.34. As can be seen, fail can be observed at all the cells of column 2. All the read operations for column 3 and 4 pass correctly. It is to be noted from the bottom graph, that the second march element fails to write 0 in the cells connected to the faulty write driver i.e., the memory cells of column 2. In conclusion, March RAWAW-H1m$_x$ fails only for column 2.

The fault detecting read operation of the fourth march element is marked in bold.
March RAWAW-H1m$_x$: $\Updownarrow_x^{H1}$ $(w1_g, w0_f, r1_g, \mathbf{r0_f}, w1_f)$

March RAWAW-H1m$_y$: The simulation result of this test is shown in Figure 6.35.

March RAWAW-H1m$_y$ is applied for each row of the memory. The address transitions for H1 addressing have been explained above. Considering the application of test for row 1 of the memory, the top graph in Figure 6.35 shows the $V(sout)$ and $V(soutcmp)$ signal of the cell read operations for the first row of the memory. As can be seen in the figure, all the read operations pass correctly.

In a similar way, March RAWAW-H1m$_y$ is applied for row 2, 3 and 4 of the memory. The simulation results can be seen in second, third and fourth graph of Figure 6.35. As can be seen, all the read operations pass correctly and no failure is detected.

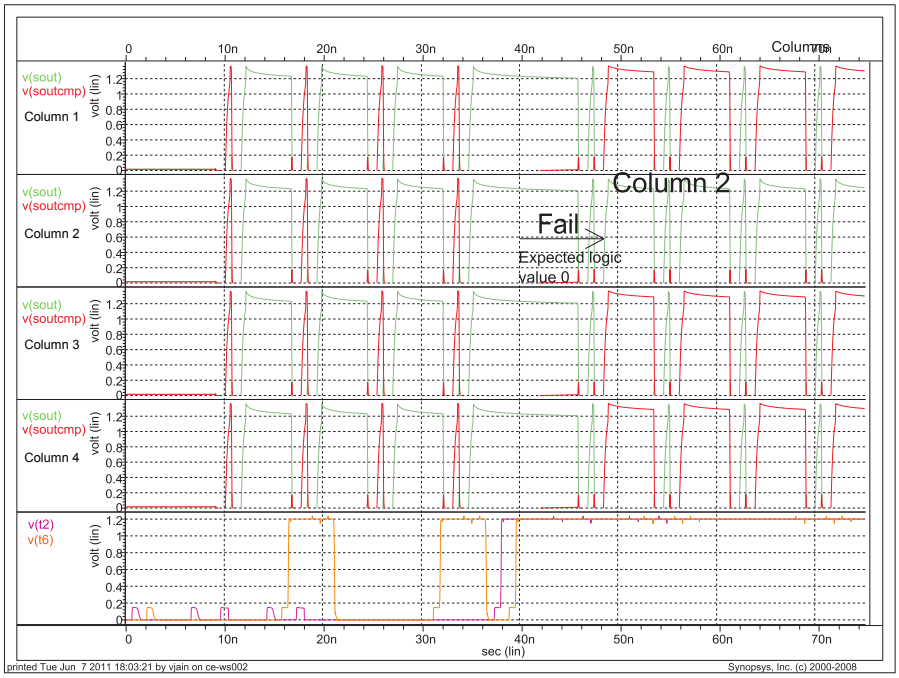

Figure 6.34: HSpice simulation: March RAWAW-H1m$_x$ applied for each row of the memory cell array

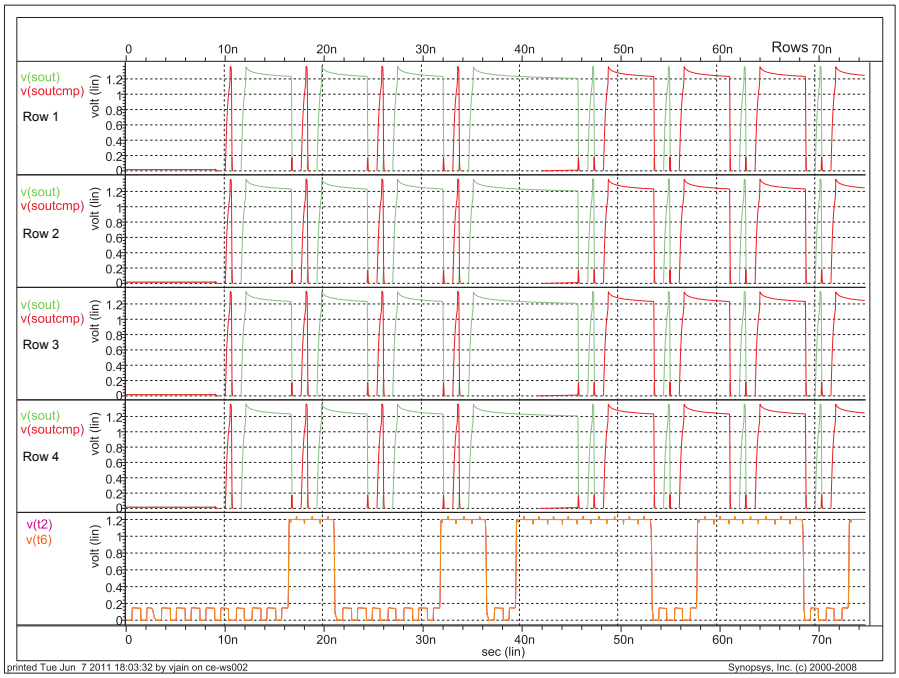

Figure 6.35: HSpice simulation: March RAWAW-H1m$_y$ applied for each row of the memory cell array

It is to be observed from provided simulation waveforms that slow write driver fault occurs in column 2 of the memory. When the diagnostic test is applied in fast-row direction, the faulty write driver (for column 2) is able to write 0 only in the first cell and all other write 0 operations fail. The pass/fail status of March MD2$_y$ is irrelevant here, and March RAWAW-H1m$_x$ fails for column 2.

Table 6.11 shows comparison of the obtained signature with level 1 diagnostic dictionary (see Table 5.11). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as a peripheral circuitry fault. This was indeed the memory block where the defect was injected; thus the correct faulty block is indicated by the diagnosis process.

Table 6.11: Level 1: Peripheral circuitry cell array identified as the faulty block

Faulty Block	Signature	March MD2$_y$ (AND)	March RAWAW -H1m$_x$(AND)	March RAWAW -H1m$_x$(OR)	March RAWAW -H1m$_y$(AND)
Memory cell array	Dictionary signature	-	0	1	0
	Observed signature	1	0	1	0

Diagnosis Level 2

At level 1, the faulty block has been identified as the peripheral circuitry. At level 2, the fault type will be identified. When the peripheral circuitry, we need to apply new diagnostic tests at level 2. The TPs developed in Section 5.5.3 will be applied. They consist of:

- **March WDmm**: { $\Updownarrow_x$ (wD), $\Updownarrow_x$ (rD, wD), $\Updownarrow_x$ $(w\bar{D})$, $\Updownarrow_x$ $(r\bar{D}, w\bar{D})$
  Here D represents checkerboard or row stripe data background (See Section 5.2).
- **March BLI**: { $\Updownarrow_x$ (wD), $\Updownarrow_x$ $(w\bar{D}, r\bar{D}, wD)$, $\Updownarrow_x$ $(w\bar{D})$, $\Updownarrow_x$ $(wD, rD, w\bar{D})$
  Here D represents solid or column stripe data background (See Section 5.2).

The procedure consists of the following:

- Apply March WDmm
- Apply March BLI

The simulation results of the application of the above tests are presented in Figure 6.36 and Figure 6.37.

March WDmm: The simulation result of this test is shown in Figure 6.36. Here, the bottom graph gives the true node voltage of tow of the memory cells connected to faulty write driver (denoted as $V(t2)$ and $V(t6)$).

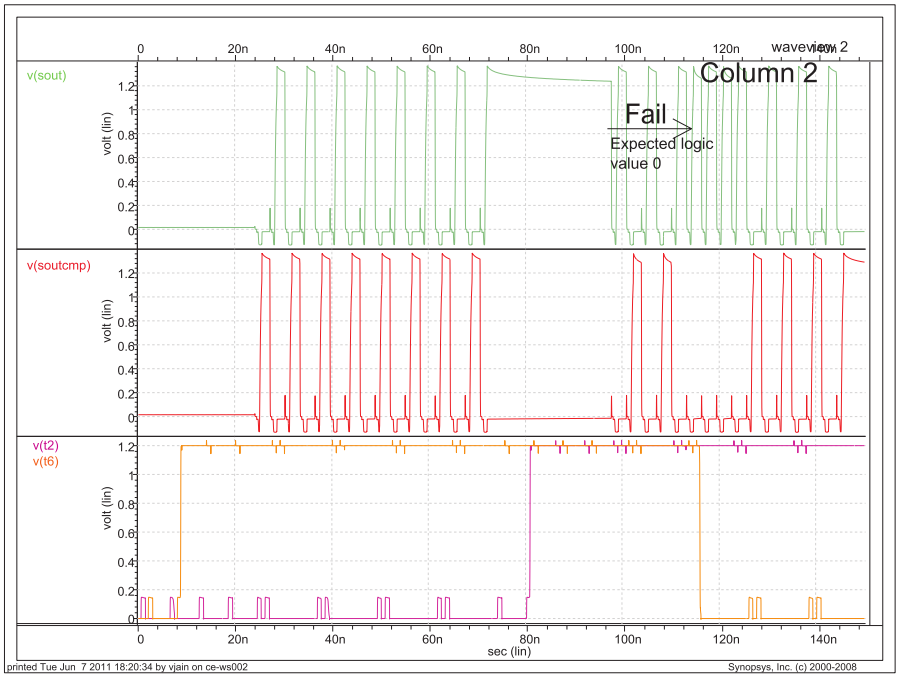

Figure 6.36: HSpice simulation: March WDmm

It is to be noted, that all the march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing with fast-row address direction. First, all the 16 cells are written to row stripe data background by the first march element $\Updownarrow_x\ (wD)$. Second, each cell is read and again written (to the initial data background) by the second march element $\Updownarrow_x\ (rD, wD)$. As can be seen in the figure, all the read operations pass correctly. Third, each cell is written (to complementary of initial data background) by the third march element $\Updownarrow_x\ (w\bar{D})$. Then each cell is read and written (to complementary of initial data background) by the fourth march element $\Updownarrow_x\ (wD, rD, w\bar{D})$. As the figure shows, the r$\bar{D}$ operation applied to the cells of column 2 fails. In conclusion, March WDmm fails for all memory cells of column 2.

The fault detecting read operation is marked in bold.
March WDmm: { $\Updownarrow_x\ (wD)$, $\Updownarrow_x\ (rD, wD)$, $\Updownarrow_x\ (w\bar{D})$, $\Updownarrow_x\ (\mathbf{r\bar{D}}, w\bar{D})$

March BLI: The simulation result of this test is shown in Figure 6.37.

It is to be noted, that all the march elements can be applied using either up addressing or down addressing; for simulation we are using up addressing with fast-row address direction. First, all the 16 cells are written to solid data background (all cells initialized to 0) by the first march element $\Updownarrow_x\ (wD)$. Second, each cell is written (to complementary of initial data background), then read and again written (to initial data background) by the second march element $\Updownarrow_x\ (w\bar{D}, r\bar{D}, wD)$. As can be seen in the figure, all the read operations pass correctly. Third, each cell is written (to complementary of initial data background) by the third march element $\Updownarrow_x\ (w\bar{D})$. Then, each cell is written (to initial data background), then read and again written (to comple-

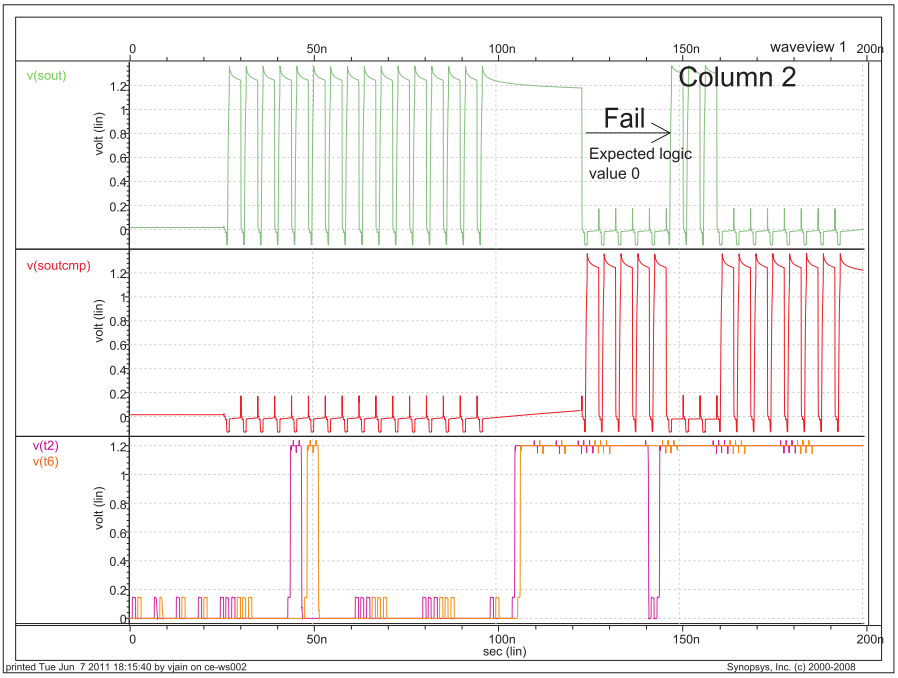

Figure 6.37: HSpice simulation: March BLI

mentary of initial data background) by the forth march element $\Updownarrow_x (wD, rD, w\bar{D})$. As the figure shows, the rD operation applied to the cells of column 2 fails. In conclusion, March BLI fails for all the memory cells of column 2.

The fault detecting read operation is marked in bold.
March BLI: { $\Updownarrow_x (wD)$, $\Updownarrow_x (w\bar{D}, r\bar{D}, wD)$, $\Updownarrow_x (w\bar{D})$, $\Updownarrow_x (wD, \mathbf{rD}, w\bar{D})$

It is to be observed from provided simulation waveforms that slow write driver fault occurs in column 2 of the memory. As can be seen, only the first write 0 operation of the faulty write driver succeeds and rest all write 0 operations fail. All the applied tests fail generating a diagnostic signature “1 1”.

Table 6.12 shows comparison of the obtained signature with level 2 diagnostic dictionary (Table 5.14). On comparing the obtained signature with the diagnostic dictionary, the fault can be classified as write driver fault. The injected defect causes a slow write driver fault; thus the correct fault type is indicated by the diagnosis process.

Table 6.12: Level 2: Fault in the peripheral circuitry identified as write path fault

Faulty Block	Signature	March WDmm	March BLI
Memory cell array	Dictionary signature	1	1
	Observed signature	1	1

6.4 Summary

This chapter provided simulation results for the validation of theory presented in Chapter 5. Defects were injected in various memory blocks of the SRAM memory model and the HMD approach was applied for the diagnosis. Simulation results were presented for analysis of static faults and dynamic faults in the memory. The injected fault was diagnosed and correct fault type/class was determined, thus establishing the usefulness of HMD.

7 Conclusions and Future Work

This chapter concludes the thesis and gives some recommendations for future research work. This chapter is organized as follows. Section 8.1 gives conclusions of the work presented in previous chapters. Section 8.2 gives an insight into future research directions.

7.1 Conclusions

This thesis started with a discussion about the importance of semiconductor memory testing and diagnosis. An overview of the memory architecture is provided with focus on explaining the functional and electrical properties of the single-port memory. Thereafter, the reduced functional model of the SRAM is discussed; it consists of 3 major subsystems: (1) Memory Cell Array, (2) Peripheral Circuitry, and (3) Address Decoder is described. The functional behavior of the memory has been explained using block diagrams while the electrical model is described in terms of transistors.

In this thesis, single-port static and single-port two-operation dynamic faults in the memory have been considered. The concept of Fault Primitives and Functional Fault Models is presented. Furthermore, the established fault models used for SRAM have been thoroughly described. Memory faults have been divided into memory cell array faults, address decoder faults and peripheral circuitry faults.

An overview of the available memory fault diagnosis algorithms is provided and shortcomings of the prevalent approaches is discussed. Thereafter, the need for a new diagnosis solution is established and a new approach called Hierarchical Memory Diagnosis (HMD) is proposed.

The concept behind the HMD approach is to diagnose memory faults in a hierarchical order. HMD narrows down the faulty area from the beginning and diagnostic tests are applied accordingly. This helps in putting the effort into fault diagnosis in a well-directed way. The proposed approach is able to locate the fault in any part of the memory system. Diagnosing the faults in all parts of the memory system will speed up the yield learning curve. The strength of the HMD lies in the fact that, unlike traditional approaches, it has no specific implementation requirements other than running a test and determining the pass/fail status of the test. The only other information required is the first failing address for the applied diagnostic test. The approach is based on the concept of Test Classes and Test Primitives, combined with Design for Diagnosis (DfD). A number of selected/developed diagnostic test algorithms are demonstrated to diagnose static and dynamic faults occurring in all parts of the memory system. Detailed discussion about the diagnosis of static and dynamic faults in the memory is presented. The outcome of the diagnosis includes details about the faulty block of the memory and further down the hierarchy, more details like fault location and fault type are made available.

A complete theoretical explanation of the approach is provided for static and dynamic faults in the memory. The approach has been validated using SPICE simulations. Defects are inserted into the memory model and simulation results are presented for analysis of static and dynamic faults.

The main contributions of the work can be summarized as follows:

- A new diagnosis approach is proposed to realize memory fault diagnosis in an hierarchical fashion: Hierarchical Memory Diagnosis.
- The approach targets static and dynamic faults in all memory blocks: the memory cell array, the address decoder and the peripheral circuitry.
- The approach can successfully locate the faulty block and identify the fault type.
- The approach uses the idea of Test Classes and Test Primitives as foundation; this makes the approach platform independent.
- The efficiency and superiority of the approach is demonstrated using defect injection and SPICE simulations.

7.2 Future Work

In this thesis, the focus has been on introducing a new approach for diagnosing static and dynamic faults in the memory. A number of optimizations can be performed and new ideas can be explored. Some recommendations for future work are given below:

- **Exploring different stress combination**: First failing address and different addressing orders have been used for diagnosis in this work. It is suggested to explore other parameters (for e.g., data backgrounds, counting methods, etc.) as well.
- **Indus trail application of proposed approach**: Industrial application and analysis of the HMD approach can give further insight into efficiency and the required changes/possible optimizations.
- **Extending the diagnosis scope**: With the proposed approach, more TCs and TPs can be added to include support for new emerging fault mechanisms. The work can also be extended to go further down the hierarchy to precisely pin-point the defect location.
- **Optimization of diagnostic tests**: For the diagnosis purpose, minimum length tests have been used wherever possible but further effort can be put into optimizing the complexity of presented diagnostic tests, especially for dynamic hierarchical analysis.
- **Optimization of DfD hardware**: Diagnosis hardware can careful designed to reduce the area overhead and nullify the effect of DfD on the delay of the memory system. This can be done efficiently if internal structure of the memory is known.

- **Diagnosis of weak faults**: The proposed approach can be extended to perform the diagnosis of weak faults. A weak fault can be defined as small disturbance that does not produce an error but additive combinations of two or more weak faults can produce an error. The concept of weak faults is new and no specific tests are available for the detection and diagnosis of weak faults. Future research can be conducted in the area of weak faults by developing new tests for fault detection, and combining them with the proposed approach for diagnosis purposes.

Bibliography

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

[2] Z. Al-Ars and S. Hamdioui, *Fault Diagnosis using Test Primitives in Random Access Memories*, Asian Test Symposium, 2009, pp. 403–408.

[3] Z. Al-Ars and A. J. van de Goor, *Approximating infnite dynamic behavior for DRAM cell defects*, Proceedings of 20th IEEE VLSI Test Symposium, 2002, pp. 401–406.

[4] S. M. Al-Harbi, F. Noor, and F. M. Al-Turjman, *March DSS: A new diagnostic march test for all memory simple static faults*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **26** (2007), no. 9, 1713–1720.

[5] A. Allan, D. Edenfeld, Jr. W. H. Joyner, A. B. Kahng, M. Rodgers, and Y. Zorian, *2001 technology roadmap for semiconductors*, Computer **35** (2002), 42–53.

[6] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H. J. Wunderlich, *Massive statistical process variations: A grand challenge for testing nanoelectronic circuits*, International Conference on Dependable Systems and Networks Workshops, 2010, pp. 95–100.

[7] A. Benso, S. D. Carlo, G. D. Natale, , and P. Prinetto, *Specification and design of a new memory fault simulator*, Proceedings of the 11th Asian Test Symposium, 2002, pp. 92–97.

[8] T. J. Bergfeld, D. Niggemeyer, and E. M. Rudnick, *Diagnostic testing of embedded memories using BIST*, Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2000, pp. 305–309.

[9] A. Bhavnagarwala, S. Borkar, T. Sakurai, and S. Narendra, *The semiconductor industry in 2025*, IEEE international Solid-State Circuits Conference Digest of Technical Papers, 2010, pp. 534–535.

[10] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing*, Kluwer Academic Publishers, 2000.

[11] P. Y. Chee, P.C. Liu, and L. Siek, *High-speed Hybrid Current-Mode Sense-Amplifier for CMOS SRAM's*, Electronics Letters **28** (1992), no. 9, 871–873.

[12] R. David and A. Fuentes, *Fault Diagnosis of RAMs from Random Testing Experiments*, IEEE Transactions on Computers **39** (1990), no. 2, 220–229.

[13] R. Dekker, F. Beenker, and L. Thijssen, *A realistic fault model and test algorithms for static random access memories*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **9** (1990), no. 6, 567–572.

[14] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, S. Borri, and M. Hage-Hassan, *Efficient March Test Procedure for Dynamic Read Destructive Fault Detection in SRAM Memories*, Journal of Electronic Testing **21** (2005), 551–561.

[15] G.Harutunyan, V.A.Vardanian, and Y.Zorian, *Minimal March Tests for Dynamic Faults in Random Access Memories*, Eleventh IEEE European Test Symposium, 2006, pp. 43–48.

[16] S. Hamdioui, *Fault Models and Tests for Multi-port Memories*, Master's thesis, Delft University of Technology, 1997.

[17] ———, *Testing Multi-Port Memories : Theory and Practice*, Ph.D. thesis, Delft University of Technology, 2001.

[18] ———, *Efficient tests and DFT for RAM address decoder delay faults*, 3rd International Design and Test Workshop, 2008, pp. 225–230.

[19] ———, *Testing Embedded Memories in the Nano-Era: Will the Existing Approaches Survive?*, Proceedings of the 2009 Asian Test Symposium, 2009, p. 339.

[20] ———, *Trends in Testing Embedded Memories in the nano-Era*, Presentation, 2011.

[21] S. Hamdioui and Z. Al-Ars, *Scan More with Memory Scan Test*, 4th International Conference on Design and Technology of Integrated Systems in Nanoscal Era, 2009, pp. 204–209.

[22] S. Hamdioui, Z. Al-Ars, and A. J. Van De Goor, *Opens and Delay Faults in CMOS RAM Address Decoders*, IEEE Transactions on Computers **55** (2006), 1630–1639.

[23] S. Hamdioui, Z. Al-Ars, J. Jimenez, and J. Calero, *PPM Reduction on Embedded Memories in System on Chip*, IEEE proceedings of European Test Symposium, 2007, pp. 85–90.

[24] S. Hamdioui, G. N. Gaydadjiev, and A. J. van de Goor, *A Fault Primitive Based Analysis of Dynamic Memory Faults*, 2003.

[25] S. Hamdioui, V. Krishnaswami, S. Irobi, and Z. Al-Ars, *A New Test Paradigm for Semiconductor Memories in the Nano-Era*, Submitted to Asian test Symposium, 2011.

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

[27] G. Harutunyan, V. A. Vardanian, and Y. Zorian, *Minimal March Tests for Unlinked Static Faults in Random Access Memories*, Proceedings of the 23rd IEEE Symposium on VLSI Test, 2005, pp. 53–59.

[28] ———, *An Efficient March-Based Three-Phase Fault Location and Full Diagnosis Algorithm for Realistic Two-Operation Dynamic Faults in Random Access Memories*, 26th IEEE VLSI Test Symposium, 2008, pp. 95–100.

[29] M. Klaus and A. J. van de Goor, *Tests for resistive and capacitive defects in address decoders*, Proceedings of 10th Asian Test Symposium, 2001, pp. 31–36.

[30] V. Krishnaswami, *A New Test Paradigm for Semiconductor Memories in the Nano-Era*, Master's thesis, Delft University of Technology, 2011.

[31] H. Kukner, *Generic and Orthogonal March Element based Memory BIST Engine*, Master's thesis, Delft University of Technology, 2010.

[32] C. Li, *Testing of Deep-Submicron Embedded Memories in FPGAs*, Master's thesis, Delft University of Technology, 2008.

[33] J. Li, K. Cheng, C. Huang, and C. Wu, *March-based RAM diagnosis algorithms for stuck-at and coupling faults*, Proceedings of International Test Conference, 2001, pp. 758–767.

[34] M. Marinescu, *Simple and Efficient Algorithms for Functional RAM Testing*, IEEE Test Conference, 1982, pp. 236–239.

[35] W. H. McAnney, P. H. Bardell, and V. P. Gupta, *Random testing for stuck-at storage cells in an embedded memory*, Proceedings of the 1984 international test conference on The three faces of test: design, characterization, production, 1984, pp. 157–166.

[36] O. Mende, *Halbleiter Bauelemente in der Automobil Elektronik*, Presentation, 2008.

[37] O. Minato, T. Masuhara, T. Sasaki, Y. Saka, T. Hayashida, K. Nagasawa, K. Nishimura, and T. Yasui, *A Hi-CMOSII 8K8 bit static RAM*, IEEE Journal of Solid-State Circuits **16** (1982), no. 5, 793–798.

[38] G. E. Moore, *Cramming more components onto integrated circuits*, Readings in computer architecture, Morgan Kaufmann Publishers Inc., 2000, pp. 56–59.

[39] N. Mukherjee, A. Pogiel, J. Rajski, and J. Tyszer, *High Volume Diagnosis in Memory BIST Based on Compressed Failure Data*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **29** (2010), no. 3, 441–453.

[40] R. Nair, *Comments on "An Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories"*, IEEE Transactions onComputers **C-28** (1979), no. 3, 258–261.

[41] A. Ney, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, and A. Virazel, *A signature based approach for diagnosis of dynamic faults in SRAMs*, 3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era, 2008, pp. 1–6.

[42] A. Ney, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Bastian, *A History-Based Diagnosis Technique for Static and Dynamic Faults in SRAMs*, IEEE International Test Conference, 2008, pp. 1–10.

[43] A. Ney, P. Girard, S. Pravossoudovitch, A. Virazel, M. Bastian, and V. Gouin, *A Design-for-Diagnosis Technique for SRAM Write Drivers*, Design, Automation and Test in Europe, 2008, pp. 1480–1485.

[44] _____, *An SRAM Design-for-Diagnosis Solution Based on Write Driver Voltage Sensing*, Proceedings of the 26th IEEE VLSI Test Symposium, 2008, pp. 89–94.

[45] D. Niggemeyer and E. Rudnick, *Automatic Generation of Diagnostic March Tests*, Proceedings on 19th IEEE VLSI Test Symposium, 2001, pp. 299–304.

[46] A. Offerman, *Automatic Memory Test Verification and Generation*, Master's thesis, Delft University of Technology, 1995.

[47] T. Powell, A. Kumar, J. Rayhawk, and N. Mukherjee, *Chasing subtle embedded RAM defects for nanometer technologies*, International Test Conference, 2005, pp. 842–850.

[48] B. Prince, *Semiconductor Memories: A Handbook Of Design Manufacture And Application*, Johan Wiley and Sons Ltd. England, 1991.

[49] W. C. Rhines, *Keynote speech at IEEE Workshop on Design and Test*, 2007.

[50] K. Sasaki, S. Hanamura, K. Ueda, T. Oono, O. Minato, Y. Sakai, S. Meguro, M. Tsunematsu, T. Masuhara, M. Kubotera, and H. Toyoshima, *A 15-ns 1-Mbit CMOS SRAM*, IEEE Journal of Solid-State Circuits, 1988, pp. 1067–1072.

[51] J. Savir, W. H. McAnney, and S.R. Vecchio, *Testing for coupled cells in random-access memories*, IEEE Transactions on Computers, 2011, pp. 1177–1180.

[52] J. Segal and R. Segers, *Test as a key enabler for faster yield ramp-up*, Proceedings of 20th IEEE VLSI Test Symposium, 2002, p. 177.

[53] A. J. van de Goor, *Using march tests to test SRAMs*, Design and Test of Computers, IEEE **10** (1993), no. 1, 8–14.

[54] _____, *Testing Semiconductor Memories : Theory and Practice*, ComTex Publishing, 1998.

[55] A. J. van de Goor and Z. Al-Ars, *Functional Fault Models: A Formal Notation and Taxonomy*, Proceedings of 18th IEEE VLSI Test Symposium, 2000, pp. 281–289.

[56] A. J. van de Goor, S. Hamdioui, and R. Wadsworth, *Detecting faults in the peripheral circuits and an evaluation of SRAM tests*, Proceedings of International Test Conference, 2004, pp. 114–123.

[57] V. Vardanian and Y. Zorian, *A march-based fault location algorithm for static random access memories*, Proceedings of the Eighth IEEE International On-Line Testing Workshop, 2002, pp. 256–261.

[58] V. Yarmolik, Y. Klimets, A. J. van de Goor, and S. Demidenko, *RAM Diagnostic Tests*, Records of the 1996 IEEE International Workshop on Memory Technology, Design and Testing, 1996, pp. 100–102.