I, ______ Robert Rohan Martin ____________________________, hereby submit this work as part of the requirements for the degree of:

Master Of Science

in:

Computer Engineering

It is entitled:

“Multi-Level Cell Flash Memory Fault Testing And Diagnosis”

This work and its defense approved by:

Chair:  Dr. Wen-Ben Jone
        Dr. Harold Carter
        Dr. Karen Tomko
Multi-Level Cell Flash Memory Fault Testing and Diagnosis

A Thesis submitted to the
Division of Research and Advanced Studies
of
The University of Cincinnati
in partial fulfillment of the requirements for the degree of

Master of Science
in the Department of Electrical and Computer Engineering and Computer Science of The College of Engineering
June 9th, 2005

by

Robert Rohan Martin
B.Tech (EEE), Jawaharlal Nehru Technological University, Hyderabad, India, 2003

Thesis Advisor and Committee Chair: Dr. Wen-Ben Jone
To my loving parents - Peter and Lata Martin
Abstract

Single bit per cell flash memories have been widely used and many efficient testing and diagnosis methodologies have been proposed. On the other hand, their multi-level cell counterparts are relatively not well-known, even though they have many advantages such as low area, high-density, low power and short access times. To the best of our knowledge, no research papers have been published for MLC flash memory testing. One reason is that conventional march algorithms cannot be directly applied. Secondly, the faults affecting MLC are not formulated efficiently, and no theoretical analysis of such faults is available. This thesis is an attempt to bridge this gap by providing a simple solution to test and diagnose a MLC flash memory array. The fault model proposed takes into account many physical defects which cause the state of the memory cell to change. The diagonal flash test (FTX) and flash diagnosis (FDX) march algorithms proposed in this thesis are a first-of-its-kind for MLC flash. Their 100% fault coverage for the fault model we propose in this thesis, low complexity and test time make them an attractive methodology for testing and diagnosing faults for multi-level flash memories.
Acknowledgements

I wish to express my sincere thanks to my advisor, Dr. Wen-Ben Jone, for all the guidance and constructive criticism that he provided throughout my work. He was always ready to provide his guidance and help in solving problems, even during weekends. This work would never have taken this form without his encouragement and expertise. Thank you Dr. Jone, for all that you have done for my research.

I would like to thank the members of my thesis committee, Dr. Harold Carter and Dr. Karen Tomko, for spending their invaluable time reviewing this work.

Thanks to all the brilliant people I had the privilege of working with, at the Flash Products Group, Intel Corporation, Folsom, CA whose guidance and mentoring, helped me chose to work on an intriguing technology such as MLC flash. This thesis would not have been possible without their help, guidance and the experience I received in their midst.

I want to thank God without whom nothing is possible. Through HIM my parents made me what I am today. I would be nothing without them. I dedicate this thesis to my parents.
Contents

1 Introduction .......................................................................................... 1
   1.1 Introduction to Flash Memories ..................................................... 1
   1.2 SBC vs. MLC Flash Memories ....................................................... 2
   1.3 Contemporary SBC Testing Methodologies ................................. 3
   1.4 MLC Testing Methodologies ....................................................... 3
   1.5 Our Approach ............................................................................... 4
   1.6 Thesis Organization .................................................................... 5

2 The Flash Single Bit Per Cell and Multi-level Cell Overview ............. 8
   2.1 Conventional Semi-conductor Memories ....................................... 8
   2.2 The Flash Memory Cell ............................................................... 11
   2.3 Flash Operating Conditions: Single Bit / Cell Storage .................. 13
      2.3.1 The Flash Memory Program Operation ................................ 13
      2.3.2 The Flash Memory Erase Operation ..................................... 15
      2.3.3 The Flash Memory Read Operation ..................................... 17
   2.4 Flash Sensing Mechanism ........................................................... 19
      2.4.1 Read Path: Sense Mechanisms ............................................ 19
      2.4.2 Sensing Analysis ............................................................... 20
      2.4.3 Threshold Voltage Distribution ......................................... 23
   2.5 Multi-Level Flash Memories ....................................................... 24
      2.5.1 MLC Charge Placement ...................................................... 25
      2.5.2 MLC Accurate Charge Sensing ........................................... 28
      2.5.3 Method 1: Multiple Sensing ................................................ 29
      2.5.4 Method 2: Parallel Sensing ................................................ 30
      2.5.5 MLC Accurate Charge Retention ........................................ 32
   2.6 Multi-Level Cell Threshold Voltage Distribution .......................... 33

3 The Multi-level Cell Architecture ....................................................... 37
   3.1 The Flash Array Layout - Full Chip Level .................................... 38
   3.2 Background: Traditional Flash Memory Architectures ................ 39
   3.3 A Simple Flash Structure ........................................................... 40
   3.4 The Flash Memory Block ........................................................... 41
      3.4.1 The Array Core Division ..................................................... 41
      3.4.2 The Flash Array Block Internal Architecture ........................ 42
   3.5 The Flash Memory Addressing Scheme ........................................ 45
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>The MLC Fault Diagnosis</td>
<td>119</td>
</tr>
<tr>
<td>6.1</td>
<td>March Algorithms Modification for Diagnosis</td>
<td>120</td>
</tr>
<tr>
<td>6.2</td>
<td>Fault Diagnosis Pattern Generation</td>
<td>120</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Fault Behavior and Diagnosis</td>
<td>121</td>
</tr>
<tr>
<td>6.3</td>
<td>The MLC Diagonal Diagnosis Algorithm FDX</td>
<td>121</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Introduction</td>
<td>121</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Motivation</td>
<td>123</td>
</tr>
<tr>
<td>6.3.3</td>
<td>The Diagnostic Algorithm</td>
<td>124</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Implementation of Diagonal FDX Algorithm</td>
<td>125</td>
</tr>
<tr>
<td>6.4</td>
<td>Test Pattern Merging</td>
<td>132</td>
</tr>
<tr>
<td>6.5</td>
<td>Test Complexity and Test Length Calculation</td>
<td>134</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>139</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Conventional MOS Memory Classification</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Flash Memory Properties</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Flash Memory Cross-section and Symbol</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Programming a Flash Cell</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Erasing a Flash Cell</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>The Single Bit per Cell Flash Read Window</td>
<td>16</td>
</tr>
<tr>
<td>2.7</td>
<td>Reading a Flash Cell</td>
<td>18</td>
</tr>
<tr>
<td>2.8</td>
<td>SBC Read Voltage</td>
<td>19</td>
</tr>
<tr>
<td>2.9</td>
<td>The Flash Memory Sensing Mechanism</td>
<td>20</td>
</tr>
<tr>
<td>2.10</td>
<td>The Flash Memory Array Sensing Mechanism for an array</td>
<td>21</td>
</tr>
<tr>
<td>2.11</td>
<td>SBC Threshold Voltage Distribution</td>
<td>23</td>
</tr>
<tr>
<td>2.12</td>
<td>Multi-Level Cell Charge Placement</td>
<td>26</td>
</tr>
<tr>
<td>2.13</td>
<td>Multi-Level Cell Charge Sensing</td>
<td>28</td>
</tr>
<tr>
<td>2.14</td>
<td>Multiple Sensing</td>
<td>30</td>
</tr>
<tr>
<td>2.15</td>
<td>Parallel Sensing</td>
<td>32</td>
</tr>
<tr>
<td>2.16</td>
<td>Multi-Level Threshold Voltage Distribution</td>
<td>34</td>
</tr>
<tr>
<td>2.17</td>
<td>Multi-Level Read Window</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>The Flash Array Location at the Full Chip level</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Multi-Level Cell Array Architecture Classification</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>The Flash Core Architecture</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>The Flash Array Core Division / Hierarchy</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td>The Flash Block Internal Architecture</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>The Flash Array Addressing Scheme Implementation</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>State Diagrams for Stuck-at Faults</td>
<td>57</td>
</tr>
<tr>
<td>4.2</td>
<td>Array with Bridging Faults</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>Broken Connection Fault Possibilities</td>
<td>60</td>
</tr>
<tr>
<td>4.4</td>
<td>Broken Word Line Connection Case 1</td>
<td>61</td>
</tr>
<tr>
<td>4.5</td>
<td>Broken Word Line Connection Case 2</td>
<td>61</td>
</tr>
<tr>
<td>4.6</td>
<td>Broken Bit Line Connection Case 1</td>
<td>62</td>
</tr>
<tr>
<td>4.7</td>
<td>Broken Bit Line Connection Case 2</td>
<td>63</td>
</tr>
<tr>
<td>4.8</td>
<td>The Cross Section of a Flash Cell</td>
<td>68</td>
</tr>
<tr>
<td>4.9</td>
<td>The flash cell Sensing Mechanism with Increased Series Source Resistance</td>
<td>69</td>
</tr>
<tr>
<td>4.10</td>
<td>Faults due to Leakage along a column</td>
<td>76</td>
</tr>
<tr>
<td>4.11</td>
<td>Faults due to Leakage along a row</td>
<td>78</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Flash Operating Conditions: SBC Storage</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Summary of Flash Read Operation</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Relationship between Drain current, State and Stored Bits</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>Parallel Sensing / Decoding of Bits</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Window Budget Voltages / State Detection</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Flash Operating Conditions: SBC Storage</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>DC - Erase Faults</td>
<td>65</td>
</tr>
<tr>
<td>4.3</td>
<td>DC - Program Faults</td>
<td>65</td>
</tr>
<tr>
<td>4.4</td>
<td>Program Disturb: Drain Stress</td>
<td>66</td>
</tr>
<tr>
<td>4.5</td>
<td>Series Resistance Faults: Summary</td>
<td>70</td>
</tr>
<tr>
<td>4.6</td>
<td>Faults due to Traps in the Oxide: Summary</td>
<td>72</td>
</tr>
<tr>
<td>4.7</td>
<td>Contamination Faults: Summary</td>
<td>74</td>
</tr>
<tr>
<td>4.8</td>
<td>Leakage Faults along a Column: Summary</td>
<td>77</td>
</tr>
<tr>
<td>4.9</td>
<td>Leakage Faults Along a Row: Summary</td>
<td>77</td>
</tr>
<tr>
<td>4.10</td>
<td>Leakage Faults due to Gate-Drain Coupling</td>
<td>79</td>
</tr>
<tr>
<td>4.11</td>
<td>Stress Induced Leakage Current Faults: Summary</td>
<td>80</td>
</tr>
<tr>
<td>5.1</td>
<td>Table Summarizing Flash Memory Fault Classification</td>
<td>84</td>
</tr>
<tr>
<td>5.2</td>
<td>Gate-Drain Capacitive Coupling faults (Low Vt): Summary</td>
<td>103</td>
</tr>
<tr>
<td>5.3</td>
<td>Test Pattern Summary</td>
<td>106</td>
</tr>
<tr>
<td>5.4</td>
<td>MLC Disturb Fault Summary</td>
<td>108</td>
</tr>
<tr>
<td>5.5</td>
<td>Diagonal FT - Step 1</td>
<td>108</td>
</tr>
<tr>
<td>5.6</td>
<td>Diagonal FT - Step 2</td>
<td>109</td>
</tr>
<tr>
<td>5.7</td>
<td>Diagonal FT - Step 3</td>
<td>109</td>
</tr>
<tr>
<td>5.8</td>
<td>Diagonal FT - Step 4</td>
<td>110</td>
</tr>
<tr>
<td>5.9</td>
<td>Diagonal FT - Step 5</td>
<td>110</td>
</tr>
<tr>
<td>5.10</td>
<td>Diagonal FT - Step 6</td>
<td>110</td>
</tr>
<tr>
<td>5.11</td>
<td>Diagonal FT - Step 7</td>
<td>111</td>
</tr>
<tr>
<td>5.12</td>
<td>Diagonal FT - Step 8</td>
<td>111</td>
</tr>
<tr>
<td>5.13</td>
<td>Diagonal FT - Step 9</td>
<td>111</td>
</tr>
<tr>
<td>5.14</td>
<td>Diagonal FT - Step 10</td>
<td>112</td>
</tr>
<tr>
<td>5.15</td>
<td>Diagonal FT - Step 11</td>
<td>112</td>
</tr>
<tr>
<td>5.16</td>
<td>Diagonal FT - Step 12</td>
<td>113</td>
</tr>
<tr>
<td>5.17</td>
<td>Diagonal FT - Step 13</td>
<td>113</td>
</tr>
<tr>
<td>5.18</td>
<td>Diagonal March Algorithm</td>
<td>113</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction to Flash Memories

The world of semi-conductor memories has seen the birth of new technologies as well as
process of many semi-conductor technologies becoming obsolete. The memory industry classifies
memories on the basis of data retention into volatile (e.g., RAM) and non-volatile memories (e.g.,
ROM). The most popular memory devices in use today are metal-oxide semiconductor memories.
RAMs may be further divided into SRAM and DRAM technologies which are huge markets by
themselves. These memories are power-dependent, since the data stored in SRAM and DRAM gets
lost once they are switched off. Due to the high speed, they are used in high memory requirement
applications. For mass storage, read-only memories (ROMs) play an important part. Even though
the broad term "ROM" would be a misnomer for erasable non-volatile memories, this generic classi-
fication is sufficient for the following discussion. ROMs can further be classified as ROM, EPROM,
EEPROM and FLASH memories. An ideal memory subsystem must optimize density, preserve
data in a nonvolatile condition, must be easy to program and reprogram, facilitate fast reading,
and must be cost-effective for the target application. Some of the memory technologies mentioned
above meet one or more of these requirements very well, but offsetting limitations can prevent
the product from becoming an ideal solution. Each memory would be suitable for different appli-
cations, corresponding to different market "niches" for memories. Today, flash memory occupies the most significant place in the mass storage and mobile device industry. The properties of flash that make it one of the most attractive semi-conductor memories in use today are: non-volatility, updateability, low cost, low power, high reliability and high speed.

1.2 SBC vs. MLC Flash Memories

Various semiconductor technologies have different requirements to store charge. For example, a D-latch needs 16 CMOS transistors to store a single bit of charge. Similarly, a SRAM Cell needs 6 transistors. Flash memories on the other hand just require a single transistor to store a bit of charge. Flash memories can be further classified (based on the number of bits which can be stored in each cell), into single bit per cell (SBC) and multi-level cell (MLC). Single bit per cell technology has been in use for many decades, and find their most widespread applications in the mobile device industry such as digital cameras (NAND) and cell phones (NOR). As the technology scales down, every effort is made to increase the memory density within the same area constraints. Hence, this effort spawned the era of the multi-level flash cell which involves storing more than one bit of data. Usually two bits of data are stored in a single cell for Intel Corporation [1, 2, 3] and NEC Corporation, [4]) SBC flash. This has revolutionized the flash industry as any flash application can now have double the density in the same array-area. Efforts by various organizations to store more than two bits of data in one flash cell have been made. For example, Sony Corporation, Japan [5] has reported an 8-level multi-level flash cell which is a symbolic product of storing 3 bits of charge per transistor.
1.3 Contemporary SBC Testing Methodologies

Traditional SBC memories are affected mostly by program disturbs, and most of the testing methodologies employed by march and non-march tests focus mainly to detect these faults. The most efficient fault models and testing algorithms in [6, 7, 8] all propose march-like algorithms which have been suitably extended from algorithms used to test SRAMs and DRAMs. The fault simulator proposed in [9] is one of the most efficient pseudo-march algorithms proposed for SBC in terms of fault coverage and test application time. Similar diagnosing algorithms have also been proposed but the problem of fault distinguishing is not solved. Traditional algorithms such as EF [7], Flash-March [6] and March-FT [8] employed read and write operations to flash cells to successfully detect the faults. The sensing scheme proposed in [3] is an effective means to detect the multi-level bits by supplying reference cells and comparing the array cell currents with reference currents. All these contributions, though few, has made SBC testing both popular and economical for any organization to include on-chip testing within the design [10, 11, 12]. Finally, the diagonal algorithm was proposed to diagnose and distinguish all of the SBC disturbance faults [13]. All the above test methods are limited as they only have provision to test two states namely logic "0" and logic "1". Moreover they cannot be used to test MLC flash memories, since the faults that occur in MLC are different, and the testing requirements for various levels are not met by these algorithms.

1.4 MLC Testing Methodologies

Being a relatively novel concept, the testing methodologies for MLC are very few and all testing schemes are limited to actual analog probing in post silicon validation laboratories. There is no literature which states any new test methodology. Moreover, only the faults which affect SBC are popular, and no research has been done to analyze the probable effects of device failures within
a flash cell to MLC arrays. In SBC testing, many march algorithms have been proposed to test the memory under test. However, when we move to multi-level flash memories, the straight-forward march algorithms cannot be applied directly as the basic functioning of the memory is different. Since the invention of multi-level flash memories, there have been absolutely no attempts to test and to diagnose an MLC array using march algorithms. This research work is an attempt to fill in the gap. This thesis attempts to provide a simple solution to the complex problem of multi-level fault testing and diagnosis.

1.5 Our Approach

Our testing strategy is a two-step process. Initially, testing for each fault is done individually. This results in the most optimal test patterns which will test each fault fully and in the most timely manner. Once all the individual faults are exercised, with the patterns that detect them, we employ the technique of pattern minimization. This results in a golden or universal test set that is able to detect all of the traditional and non-traditional faults indicated in the MLC fault model [14, 15, 16, 17, 18, 19, 20, 21]. The universal test set has the lowest complexity and test time. However, we will employ a slightly different approach to fault detection than this process. We then employ the diagonal algorithm which was first proposed for SBC in [13]. However, the fault model proposed is different and moreover, this is not efficient for our MLC flash array and needs to be modified to detect all faults for all states. This diagonal algorithm implements portions of the universal test set at strategic points in the algorithm to detect all of the MLC faults. The diagonal algorithm which is employed has a slightly higher and longer test time when compared to the universal test set. This is compromised for a very good reason. If the diagonal algorithm is applied to test the array, the same diagonal algorithm through suitable modifications will be re-used
to diagnose every single fault in the MLC fault model. In this case, our fault diagnosis strategy governs our testing strategy as the testing and diagnosis algorithm application times are very comparable. Thus, we elucidate the advantages of using the diagonal algorithm for MLC fault testing (Chapter 5) and diagnosis (Chapter 6).

The diagonal algorithm will involve programming cells along the diagonals of the array which induces faults in the non-diagonal elements. By reading the non-diagonal cells, all faults which occur will be detected and diagnosed. Then, the opposite process is carried out, i.e., the non-diagonal elements are programmed and the faults are induced on the diagonal cells. By reading the diagonal cells appropriately, all faults can be detected and diagnosed. Thus, by these steps the entire array will be traversed thus giving us a 100% coverage. This process is done for all states, and thus every fault in the model will be tested at all voltage levels. This thesis takes a four-level MLC array which can store two bits per transistor as a test platform. However, for higher multi-level arrays, the algorithms proposed only need to be modified for the extra states involved. Thus, this strategy is like the one-size-fits-all concept but with a few modifications.

1.6 Thesis Organization

Chapter 2 deals with the basic concepts of flash memory technology. Traditional single bit per cell flash memories are described briefly. The concept of multi-level flash is summarized, where more than one bit of data can be represented in one transistor. The chapter concludes with a discussion of this MLC technology and why its testing is so important in today’s designs.

Chapter 3 begins with listing the conventional flash architectures. The beauty of MLC tech-
nology is that any suitable SBC architecture can be implemented to realize a MLC product. Since the MLC architecture strongly governs: the faults that occur and the faults that can be detected, an appropriate MLC architecture needs to be selected. The application of the march test and diagnosis algorithms is governed partially by the architecture, and therefore it is very important. This chapter ends with the addressing scheme that will be used to access the flash array, and pass test vectors to program the individual flash cells to various levels.

Chapter 4 lists the various faults that occur in flash memory architectures. These faults are common to both SBC as well as MLC flash memories. There are faults, however, that affect MLC flash architectures to a greater extent than SBC, due to the sensitivity of $V_t$ levels within each cell. These faults will be explained in detail, and their impact on MLC arrays will be emphasized. Initially, the traditional SBC fault model is discussed and then various non-traditional faults are included to formulate the MLC fault model.

Chapter 5 begins with a brief review of the inability of conventional march algorithms to test flash memories. After acknowledging the behavior of each fault in the fault model, the test patterns that detect these faults will be determined. The main aim of any test algorithm, is to detect the maximum number of faults with a minimal number of test patterns or algorithm steps. This reduces the on-tester time, thus keeping the cost of fault detection to a minimum. This is the driving force behind merging the test patterns of various faults into one effective diagonal test algorithm. The last section of this chapter deals with calculating the run-time and complexity of the testing algorithm in terms of read, program and erase operations. This chapter only deals with the patterns for flash testing. No emphasis is made to the fault types.
Chapter 6 introduces a new MLC diagonal algorithm which will diagnose all the traditional and non-traditional faults discussed in the fault model (Chapter 4). Initially, all of the program disturbs are distinguished very effectively by the diagonal algorithm. Then, patterns to diagnose all of the remaining faults will be added to this base diagonal algorithm to determine the modified Diagonal-FDX algorithm. Again, the main aim of this diagnosis algorithm, is to diagnose the maximum number of faults with a minimal number of patterns or algorithm steps. The last section of the chapter deals with calculating the run-time and complexity of the diagnosing algorithm in terms of read, program and erase operations.

Chapter 7 concludes this research work and summarizes the effectiveness of the diagonal method to detect all of the MLC flash memory faults.
Chapter 2

The Flash Single Bit Per Cell and Multi-level Cell Overview

This chapter deals with the basic concepts of flash memory technology. A brief introduction to the importance of non-volatile and re-programmable memories is done, and an in-depth coverage of the flash cell physics is covered. Traditional single bit per cell flash memories will also described. The limits of flash technology have been pushed further with the concept of multi-level flash where more than one bit of data can be represented in one transistor. This chapter concludes with a discussion of this MLC technology and why its testing is so important in today’s designs.

2.1 Conventional Semi-conductor Memories

In general, metal-oxide semiconductor memories can be broadly classified on the basis of data retention into two categories namely volatile and non-volatile memories. Examples of volatile and non-volatile memories are RAMs and ROMs, respectively. RAMs may be further divided into SRAM and DRAM technologies which are huge markets by themselves (Fig. 2.1). However, these memories will retain their data only if power is turned on. They lose all the charge stored in them as soon as they are switched off. Due to their high speed, they are usually used in high memory requirement applications but not good for mass storage. This is where non-volatile memories
such as read only memories come into play. Even though the broad term "ROM" would be a
misnomer for erasable non-volatile memories, this generic classification is sufficient for the following
discussion. ROMs can further be classified as ROM, EPROM, EEPROM and FLASH memories
Fig. 2.1). An ideal memory subsystem must optimize density, preserve data in a nonvolatile
condition, must be easy to program and reprogram, facilitate fast reading and must be cost-effective
for the target application. Some of the memory technologies mentioned above meet one or more
of these requirements very well, but offsetting limitations can prevent the product from becoming
an ideal solution. Today, flash memory occupies the most significant place in the mass storage and
mobile device industry. This section is limited to a discussion of the advantages of flash memories
and a few of its properties have been highlighted.

Each of the above memories has its own advantages and disadvantages. This can be clearly
summarized in the following diagram. Clearly, the importance and versatility of flash memories is
the key reason for its widespread application in the industry today. Hence, from the venn diagram
(Fig. 2.2), the properties of flash can be summarized as:
- Non-volatile
- Updateable
- Low cost
- Low power
- High reliability
- High speed

Figure 2.2: Flash Memory Properties.

Area and cost-constraints also govern the success of any semi-conductor memory. To make a comparison of various conventional technologies:

1. A standard CMOS D-Latch will require sixteen CMOS transistors to store one bit of information.

2. A SRAM cell needs six transistors to store one bit of information.
3. A flash cell requires only one transistor to store one bit of information. This is traditional single bit per cell technology. This alone allows a very high density memory to be included into the target application.

4. A new concept in flash is the multi-level cell technology. MLC flash technology facilitates the storage of two or more bits per transistor. SBC and MLC products have the same architecture, and so for a given area requirement, the memory density can be doubled or tripled. This is discussed in section 6.

Clearly, from the above discussion, the advantage of flash technology outweighs its semiconductor memory counterparts, and is the forerunner in the memory industry today.

### 2.2 The Flash Memory Cell

A flash memory is a non-volatile (NVM) device, whose *unit cells* are fabricated using CMOS technology. They can be programmed and erased electrically. The most popular flash cell which has widespread application is the *Industry Standard Flash Memory Cell* (ISFC) which is a NOR, common ground, double polysilicon device which is usually programmed using CHE (channel hot electron) injection and erased by the Fowler-Nordheim (FN) tunneling at the source junction (Source Erase). Almost 85% of today’s flash devices are based on NOR structures.

Now, the internal device structure of the ISFC is discussed [22]. It is composed of an n-channel MOSFET transistor with an additional gate which is floating as shown in Fig. 2.3. The floating gate means there is no physical connection to any of the device terminals (G, D or S). It is made of a dielectric material and is separated from the channel of the MOSFET by a thin oxide layer (called the gate oxide, approx 10nm). The control gate is made of a triple layer dielectric (oxide-nitride-oxide) called *interpoly dielectric*. The thickness of this dielectric is equivalent to
20nm of silicon dioxide layer. The source junction is smoother and deeper than the drain junction to achieve higher source-substrate breakdown voltages. Also, since the cell undergoes FN source erase, the source must be able to withstand large program-erase cycling voltages, and is therefore made deeper than the drain region within the MOSFET.

![Flash Cell Cross-section and Symbol.](image)

The gate oxide is very thin. If a high voltage is applied at the source, when the control gate is grounded, then a high electric field exists in the oxide enabling tunneling effects from the FG to the source. This operating condition is very close to the breakdown region of the source-substrate junction. Hence, the source diffusion is processed differently from the drain one as shown in Fig. 2.3, to counter the effects described above. The thickness of the oxide-nitride-oxide (ONO) interpoly dielectric strongly influences the program/erase speed and the magnitude of the read current for the ISFC. The reliability of the flash cell and charge retention capacity are dependent on the quality of both the interpoly and the tunnel oxide. The only difference in the fabrication of a flash cell from a standard CMOS transistor is the intermediate steps of introducing the floating gate and interpoly dielectric between the control gate and the substrate. Hence, the capacitance introduced due to the floating gate would be the only variant in the analog behavior of the flash cell. Thus,
all standard CMOS equations can be applied to a flash cell with an extra term in certain cases to account for this.

2.3 Flash Operating Conditions: Single Bit / Cell Storage

This section describes the three operations of the ISFC namely Read, Program, Erase. The electrical mechanisms which lead to facilitating these operations will also be discussed. In order to read an industry standard cell, the information stored in the cell must be decoded. The read, program and erase bias configurations are shown in Table 4.1. The typical values for biasing the flash cell are:

\[
V_{cc} = 5v; \ V_{pp} = 12v; \ V_{neg} = -8v; \ V_{dd} = 7v; \ V_{read} = 1v
\]

Table 2.1: Flash Operating Conditions: SBC Storage

<table>
<thead>
<tr>
<th>No</th>
<th>Process</th>
<th>Source</th>
<th>Control Gate</th>
<th>Drain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Read</td>
<td>Gnd</td>
<td>Vcc</td>
<td>Vread</td>
</tr>
<tr>
<td>2</td>
<td>Program</td>
<td>Gnd</td>
<td>Vpp</td>
<td>Vdd</td>
</tr>
<tr>
<td>3</td>
<td>Erase</td>
<td>Vcc</td>
<td>Vneg</td>
<td>Float</td>
</tr>
</tbody>
</table>

By definition, the minimum amount of gate voltage needed to turn on a transistor is called threshold voltage. In flash cells, the Vt determines the cell state, i.e., programmed or erased.

2.3.1 The Flash Memory Program Operation

Having described the cross section of a basic NOR based flash cell, the programming of a flash cell may now be introduced. The process of getting electrons onto the floating gate is called programming [22]. Usually, program consists of two steps; program and verify (to ensure the cell is programmed to the desired level). The flash cell bias conditions have been summarized in Table 4.1 and is depicted in Fig 2.4 [22]. The control gate is given a relatively high voltage of about 12 volts (Vpp). This causes a high electric field between the control gate and the grounded substrate.
When the biasing drain voltage (Vdd) of about 5-7 volts (source grounded) is supplied, there is a current flow in the channel of the cell. The electrons are now hot due to this high electric field, and get attracted to the positive control gate. This happens when the electrons tunnel through the oxide between the floating gate and the substrate, and this is how the oxide gets its name. This process of injecting the electrons onto the floating gate is called programming, and the mechanism responsible is channel hot electron injection (or CHE).

![Figure 2.4: Programming a Flash Cell.](image)

Typical programming times are of the order of 5 to 20 micro seconds. After the mechanism of programming, the flash cell under consideration is said to hold logic "0". The mechanism of how this logic value is determined will be explained in Section 4.3. By programming a cell, its threshold voltage or Vt increases. This is due to the excess number of electrons in the floating gate which cause a field which is opposes the gate voltage. Hence, compared to a virgin cell, the Vt of a programmed cell will be higher.
2.3.2 The Flash Memory Erase Operation

The mechanism of Erase is just the opposite of programming. Erasing a flash cell involves removing the electrons from the floating gate of the cell [22]. This would mean biasing the cell such that the electrons would tunnel through the tunnel oxide and into the source. The flash cell bias conditions have been summarized in Table 4.1, and the erase condition is depicted in Fig 2.5 [22]. The source is given a positive voltage of Vcc and the gate is given a high negative voltage. The drain throughout the erase process is kept floating. Due to the high negative field produced at the control gate, the electrons in the floating gate are strongly repelled. The positive voltage in the source further strengthens this field and this scenario causes what is called Fowler Nordheim tunneling, where there is a sloping energy barrier through which the electrons tunnel through the tunnel oxide into the source. The cell is now said to store logic "1".

![Diagram of Flash Cell Erase](image)

Figure 2.5: Erasing a Flash Cell.

The mechanism of how this logic value will determined, is explained in Section 4.3. Flash memories derive their name from this operation as an entire block of memory is erased at one time,
i.e., they are erased in a flash. Typical erase times are of the order of 200ms per block (for 2MBit array [13]) and up to a second for the entire array. The erase time depends on the size of the entire block. The removal of electrons causes the threshold voltage or the Vt of the erased cell to get lowered. This is due to the lack/dearth of electrons in the floating gate, and a very small voltage is enough to turn on the channel. Hence, compared to a virgin cell, the Vt of an erased cell will be lower.

From Sections 4.1 and 4.2, the Vt diagram of a programmed and an erased cell can be depicted as shown in the Fig 2.6. Clearly, as the gate voltage is varied, depending on the number of electrons trapped in the floating gate, the device turns on at a different voltage.

- Vte is the Vt of the cell when it is erased.
- Vtp is the Vt of the cell when it is programmed.
- Vt is the window budget between the erased and programmed Vt’s of the cell.

![Figure 2.6: The Single Bit per Cell Flash Read Window.](image-url)
2.3.3 The Flash Memory Read Operation

After the flash cell has been programmed to a particular logic level, the next step is to read out its content. In order to carry out a read, the cell must be biased in the following manner. Since the flash cell must operate as a normal transistor, the biasing conditions are very similar to traditional CMOS operations. The gate voltage is maintained at Vcc or 5V, the drain bias voltage is low at around 1V, and the source is grounded (Fig. 2.7). Under these conditions, the drain current that flows through the cell will be compared with a reference current (Iref) and passed through a sense amplifier. Depending on the magnitude of the cell current (Id), the logic value contained in the cell is determined.

Reading a Programmed Cell

A cell which is programmed will have electrons in the floating gate and this increases the Vt (the voltage required to activate the channel) of the cell (Fig. 2.8) [22]. This results in a low current flowing through the channel (Id). When this current is compared with the reference current (Iref), it results in a logic ”0”. This will be graphically explained in Section 5.

Reading an Erased Cell

Conversely, if the cell is erased, the channel will be devoid of electrons and as a result the Vt of the cell is low. Hence, since it turns on at a low Vt or Vgs, there will be a higher current flow for a given read voltage (Fig. 2.8). This current as in the previous case is compared with a reference current through a sense amplifier. Since this current will be much higher than the reference current provided by the reference cell, the logic value recorded in the cell would be logic ”1”. This will be graphically explained in Section 5.
Since the read operation must be such that it can distinguish very clearly between the erase and programmed cell currents, the read voltage (i.e. the gate voltage (Table 4.1)) must be in between the threshold voltages of erased and programmed cells [22]. Fig. 2.8 indicates this process and Table 2.2 summarizes the logic value corresponding to the Vt value of the flash cell.

Table 2.2: Summary of Flash Read Operation

<table>
<thead>
<tr>
<th>No.</th>
<th>Vt</th>
<th>Id</th>
<th>State</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>High</td>
<td>Low</td>
<td>Programmed</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>2</td>
<td>Low</td>
<td>High</td>
<td>Erased</td>
<td>&quot;1&quot;</td>
</tr>
</tbody>
</table>
2.4 Flash Sensing Mechanism

2.4.1 Read Path: Sense Mechanisms

This section deals with the principle architectures employed in a non-volatile flash memory to read the information stored in the cells. The functional unit block which accomplishes the reading of a cell content is called the sense amplifier, and is usually divided into two components: the current-voltage converter and the comparator (Fig. 2.9) [14]. The communication speed of the converter from various works in literature is reported to be between 10 to 30 ns, depending on the current sunk by the cell and also the type of technology (SBC vs MLC). During this operation, the cell source is tied to GND and the gate is driven by the row decoder to a voltage which is usually close to Vcc or slightly higher (5 to 7v). The cell’s drain voltage must be low enough to avoid stress but high enough to allow proper current flow to guarantee quick reading.

Fig. 2.9 depicts the sensing mechanism of one cell in the array. In order to get a clear picture of what
happens in a practical array, a 3 x 3 flash array of 9 cells is considered as shown in Fig. 2.10. The bit lines are connected together (labelled Y) and compared against the reference voltage. Another way to compare the selected to be read would be to multiplex the bit lines (Fig. 2.10). The bit line which gets activated would be applied to the MUX select lines to select the appropriate MUX output. The output of the MUX would be compared against the reference value to compute the contents of the flash cell.

2.4.2 Sensing Analysis

Consider the Following cases:

Case 1: Sensing Analysis When Cell is Programmed. From Fig. 2.9, let the cell to be read be programmed. Now, the Vt value of the cell has increased, so the current flowing through it for
a given read voltage will be small. Due to this small current, the voltage drop across $R_m$ is also small. Since there is a constant overhead voltage $V_{cc}$, the net voltage at Node B will be high. This value is greater than the corresponding value of the voltage at Node A (obtained similarly). When these two voltages $V_A$ and $V_B$ are passed through the differential amplifier, the output is Zero (since the positive input is less than the negative one).

**Case 2: Sensing Analysis When Cell is Erased.** From Fig. 2.9, let the cell to be read be erased. Now, the $Vt$ value of the cell has decreased, so the current flowing through it for a given read voltage will be large. Due to this large current, the voltage drop across $R_m$ is also large. Since
In recapitulation - Case 1: Cell is programmed

\[ Vt(ArrayCell) > Vt(ReferenceCell) \]
\[ \Rightarrow ICell < IRef \]
\[ \Rightarrow V_B > V_A \]
\[ \Rightarrow ComparatorOutput = Logic"0" \]

there is a constant overhead voltage Vcc, the net voltage at Node B will be small. This value is smaller than the corresponding value of the voltage at Node A (obtained similarly). When these two voltages VA and VB are passed through the differential amplifier, the output is One (since the positive input is greater than the negative one).

In recapitulation - Case 2: Cell is erased

\[ Vt(ArrayCell) < Vt(ReferenceCell) \]
\[ \Rightarrow ICell > IRef \]
\[ \Rightarrow V_A > V_B \]
\[ \Rightarrow ComparatorOutput = Logic"1" \]

The output of the comparator circuit is then sent to the data pin as output, and the data value at the addressed location is sent out. There are 16 such sense amps per strip (this will be discussed in Chapter 3 which deals with flash architecture), indicating that 16 bits of data are read out at a time. Hence, the bandwidth or the bus width of the output data bus must be 16 bits. From the above sensing scheme, it is clear that a single flash cell can store either a logic "0" or a logic "1". Note that traditional flash memories are also called single bit per cell flash memories or SBCFMs.
2.4.3 Threshold Voltage Distribution

Let us assume for simplicity that 50% of the cells in the array core are programmed, and 50% of the cells are erased. If every cell in the array core has been sensed in the fashion just described, then we can draw a plot between the number of cells and the threshold voltages of all the cells. This is called the Vt distribution graph (Fig. 2.11). The number of cells in the array is on the Y axis and the range of the threshold voltages Vt of all the cells is represented on the X axis. Let all the cells which are erased be represented by region 1, and all the cells which are programmed be represented by region 2. The distribution depicted arises from the fact that every cell in the array that is programmed need not have a constant Vt, even though it may be still considered programmed and the same applies to all cells which are erased.

![Figure 2.11: SBC Threshold Voltage Distribution.](image)

Fig. 2.11 shows that during a read, a reference voltage is fixed between threshold voltages of the least erased and the least programmed cell. Having this window, any voltage below the reference voltage and outside this window is assumed erased, and conversely any voltage above this reference and outside the window is assumed programmed. This diagram is very important and
will be referenced at various points in this thesis.

The window described is the voltage gap between the erased (region 1) and programmed states (region 2). If the Vt of the least erased cell is $V_{teM}$ (we use a upper case "$M$" as this is the max Vt of the erased cell distribution) and the Vt of the least programmed cell is $V_{tpm}$ (we use lower case "$m$" as this is the min Vt of the programmed cell distribution), then we have a Vt window budget of $V_{tpm} - V_{teM}$. This also implies that a read voltage must be anywhere between this window in order to properly sense the data present in the cell. (This window is also termed Read Window Budget). Thus, having described the sensing circuitry and the method used to determine the data value stored in the flash cell, this section concludes the discussion of single bit per cell flash technology. With the invention of the concept of being able to store more than one bit of information in one flash cell, the multi-level and multi-bit flash memory technology has its inception. Multi-level flash cells are of particular interest and are covered in the following section.

2.5 Multi-Level Flash Memories

The flash cell operation is governed by electron charge storage on the floating gate. Hence, the amount of charge stored in the floating gate modulates the flash cell’s transistor characteristics. Multi-level cell technology implements the same flash cells as the ones used in traditional SBC memories. The only difference is that the programming voltages of a cell can be varied to different Vt values. Different Vt levels are symbolic of the charge stored within the floating gate. The larger the number of levels that the flash cell can be programmed to, the more effective number of bits each cell will be able to store. The Vt window discussed for SBC will be split up into many levels depending on the number of bits that each flash cell is required to store. Hence, if there are $2^n$
levels which can be stored in a cell, then the cell is said to store $n$ bits. Conversely, if a target application requires a flash cell to store $p$ bits, then the cell must be able to vary its $V_t$ between $2^p$ levels (or there must be $2^p$ programming voltage levels provided in the chip to represent the max number of states corresponding to $p$ bits).

MLC requires three basic tasks to achieve these intermediate states [22]:

- **Accurate Charge Control / Charge Placement**: Accurate charge placement involves placing the right amount of charge on the floating gate such that multiple charge levels or multiple bits can be stored within each cell. This is called *placement*.

- **Accurate Charge Measurement**: Accurate measurement is to determine what the charge level is within a particular cell. This is called *sensing*.

- **Accurate Charge Storage**: Accurate charge storage ensures the charge level or data remains intact over time. This is called *data retention*.

### 2.5.1 MLC Charge Placement

This subsection has been taken from Intel’s Corporate Website [22]:

Accurate charge placement involves placing the right amount of charge on the floating gate such that multiple charge levels or multiple bits can be stored within each cell. This is called placement. MLC is basically a means of varying the charge in the floating gate by proper modulation. Detecting intermediate charge levels or states can be extracted from each cell of the array. These states do not represent only logic 0 and logic 1 as in the case of single bit per cell (SBC), but rather represent MLC voltage states with four distinct logic values of 11, 10, 01 and 00. In MLC products, flash cells are programmed from the erased state of 11 to either of the programmed
levels of 10, 01 or 00. However, during programming if too many electrons get injected into the floating gate, the state of the cell may change. The excess charge may cause an intermediate level to breach the inter-state gap, and cause the $V_t$ to shift to the next level. For example, the desired programmed level of 10 could be erroneously be shifted to the 01 state due to excessive over-programming. Therefore, a method of controlling precisely how much charge is transferred to the floating gate is required. The programming threshold vs time curve is shown in Fig. 2.12 [22].

Fig. 2.12: Multi-Level Cell Charge Placement.

Fig. 2.12 depicts the flash cell threshold voltage shift due to varying bias conditions [22]. Two regions of operation are shown in the figure. In the linear region, the programming of a flash cell is faster when compared to programming in the saturation region. In the linear region the gate voltage has little influence on the rate of programming, while in the saturation region, the gate voltage has a predominant dependency in the saturated $V_t$. From the figure, programming of a flash cell slows as charge is added onto the floating gate. The high negative charge that
develops in the floating gate would cause a repulsive force against the electrons trying to tunnel onto the floating gate. This slows the process of programming and this is clearly noticed in Fig. 2.12.

Given the characteristic curve, the charge placement process must be reliable, i.e., no overshoot, controllable and fast. Programming in the linear region is quick but controllability would be a major concern. In this region, the programming Vt is exponentially dependent on the time and the electron energy distribution (drain bias, channel length, doping profiles etc.) [22]. Hence, small variations will produce large changes in the threshold voltage, and therefore the probability of an overshoot is high and reliability is an issue. In the saturation region, the cell Vt depends on the applied control gate voltage. Control in this region is more achievable [22]. A suitable placement algorithm will ensure that the cell is programmed to the right level. However, due to intrinsic defects or other faults that occur in the chip due to various reasons (discussed later), this charge level may be changed unintentionally causing the flash cell to store a faulty value. This is the basis of the fault model which will be elaborately explained in Chapter 4 - The MLC Fault Model. Multi-level cells have the exact cell structure and architecture as SBC which makes the fabrication of an MLC product very similar. The only difference is the number of levels that need to be incorporated. Since the voltage levels are usually generated on-chip for single power supply devices, the charge pumps will need to generate them. This leads to additional circuitry in the charge pump and high voltage circuitry. The sensing circuitry will also need to be modified to facilitate easy sensing of all the multi levels which must be efficient enough to sense all the bits.
2.5.2 MLC Accurate Charge Sensing

Accurate measurement is to determine what the charge level is within a particular cell. This is called *sensing*. The MLC flash memory under test is a two-bit per cell or a four level multi-level flash architecture. Therefore, the sensing circuitry must be in a position to determine all four states. The four different charge states are "11", "10", "01" & "00" or 2 bits in one cell. These four states are illustrated in Fig. 2.13 [22].

![Figure 2.13: Multi-Level Cell Charge Sensing.](image)

From Fig. 2.13, there are four cell drain current levels which relate to the corresponding Vt of the cell under consideration. State "11" is said to be the erased state. If the cell current is greater than reference current 1, then the state is recognized as an erase state [22]. States 10, 01 and 00 are the programmed states with the amount of programming charge increasing progressively (respectively).

Table 2.3 summarizes the relationship between states and the sensing currents. It can be found that state "11" is erased. If this is programmed to a higher Vt, the next programmed state achieved is "10". If further programmed the Vt increases to realize a state corresponding to
Table 2.3: Relationship between Drain current, State and Stored Bits

<table>
<thead>
<tr>
<th>No.</th>
<th>Sensing Current</th>
<th>State</th>
<th>Bits Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( I_{cell} &gt; I_{ref1} )</td>
<td>Erased</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>( I_{ref2} &lt; I_{cell} &lt; I_{ref1} )</td>
<td>Programmed</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>( I_{ref3} &lt; I_{cell} &lt; I_{ref2} )</td>
<td>Programmed</td>
<td>01</td>
</tr>
<tr>
<td>4</td>
<td>( I_{cell} &lt; I_{ref3} )</td>
<td>Programmed</td>
<td>00</td>
</tr>
</tbody>
</table>

"01". State "00" is the most programmed state. These states are detected by \( I_{ref1} \), \( I_{ref2} \) and \( I_{ref3} \). If the drain current of the cell under consideration is between \( I_{ref1} \) and \( I_{ref2} \), then the state is determined to be a programmed state and the value of the date stored within the flash cell is 10. If the drain current is between \( I_{ref2} \) and \( I_{ref3} \), then the state is determined to be a programmed state and the data within the flash cell is 01. Finally, if the drain current of the cell under consideration is greater than \( I_{ref3} \), the state is the most programmed and the data value decoded is 00. There can be two ways in which the above bits can be decoded, and they are presented in the following sub-sections.

2.5.3 Method 1: Multiple Sensing

The first method is to employ the same sense amplifier twice to determine each bit at time. In other words, the first sense would set the reference cell current to \( I_{ref2} \). Depending on whether the drain current of the cell under consideration is greater or less than this value, the MSB will be determined. For example if the drain current \( (I_{cell}) \) is larger than \( I_{ref2} \), then the MSB = 1. (MSB = 0 if \( I_{cell} < I_{ref2} \)). Once the MSB is determined, the on-board controller will determine what reference current to set the reference cell to. Since the MSB was detected as 1, the second sense causes the reference current to be set to \( I_{ref1} \). Now, again, \( I_{cell} \) and \( I_{ref} \) are compared. If \( I_{cell} > I_{ref1} \), then we have LSB = 1 (If \( I_{cell} < I_{ref1} \), LSB = 0). States "01" and "00" can be detected by first setting reference current to \( I_{ref2} \) and then \( I_{ref3} \), respectively. Thus, using two
senses, four $V_t$ levels can be decoded into 2 bits, and the correct data value of the cell can thus be calculated (Fig. 2.14). The above algorithm is very similar to the binary search algorithm and is summarized below.

\begin{verbatim}
Initially Set $I_{ref} = I_{ref2}$.
State is now $R_2$.
If $I_{cell} > I_{ref2}$; $\Rightarrow MSB = 1$.
State is now $R_1$.
If $I_{cell} > I_{ref1}$ $\Rightarrow LSB = 1$; $\Rightarrow Data in Cell = 11$.
If $I_{cell} < I_{ref1}$ $\Rightarrow LSB = 0$; $\Rightarrow Data in Cell = 10$.

If $I_{cell} < I_{ref2}$; $\Rightarrow MSB = 0$.
State is now $R_3$.
If $I_{cell} > I_{ref3}$ $\Rightarrow LSB = 1$; $\Rightarrow Data in Cell = 10$.
If $I_{cell} < I_{ref3}$ $\Rightarrow LSB = 0$; $\Rightarrow Data in Cell = 00$.
\end{verbatim}

Figure 2.14: Multiple Sensing.

2.5.4 Method 2: Parallel Sensing

This section has been taken from [22].
High speed random access and precise charge sensing are accomplished through a parallel charge sensing scheme (Fig. 2.15) [22]. There must be a physical and direct connection with each cell to each of the reference cells in order to determine the level of each array cell. The data read operation senses which of the four levels the memory cell falls within based on the $V_t$ of the reference cells. This is achieved by simultaneous sensing with three sense amplifiers where each SA compares the flash cell current being sensed with the current of the reference cell it is connected to. The memory cells and the reference cells are biased such that each conducts a current ($I_{cell}$ and $I_{ref}$) proportional to their threshold voltage ($V_{tcell}$ and $V_{tref}$ respectively).

<table>
<thead>
<tr>
<th>No.</th>
<th>Cell $V_t$</th>
<th>$V_t &lt; V_{tR1}$</th>
<th>$V_{tR1} &lt; V_t &lt; V_{tR2}$</th>
<th>$V_{tR2} &lt; V_t &lt; V_{tR3}$</th>
<th>$V_t &gt; V_{tR3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V_t &lt; V_{tR1}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$V_{tR1} &lt; V_t &lt; V_{tR2}$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$V_{tR2} &lt; V_t &lt; V_{tR3}$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$V_t &gt; V_{tR3}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

During a read operation, $V_{read}$ is placed on the control gates of the memory and ref cells, the sources of all cells are grounded, and the drain voltages are set through the bias circuit that utilizes a precision voltage reference circuit. The current of the cell being sensed is compared to the currents of the three reference cells. The memory cell and the reference cell currents are converted into a voltage through an active load transistor. These resulting voltages are compared at the sense amplifiers. A sense amplifier is associated with each of the reference cells. Each sense amplifier has an input from the flash cell being sensed. If the current of the cell being sensed $I_{cell} > I_{ref}$ or $V_t < V_{tref}$, the sense amplifier outputs a logic 1 and logic 0 if the opposite is true ($I_{cell} < I_{ref}$ or $V_t < V_{tref}$). The outputs of the three sense amplifiers are connected to a logic circuit that interprets the two data bits in parallel. Fig. 2.15 depicts the way in which the reference cells are
biased and connected to the flash array cells. The logic circuit interprets the data within each flash cell as indicated by the Table 2.4.

2.5.5 MLC Accurate Charge Retention

Accurate charge storage ensures the charge level or data remains intact over time. This is called data retention [22]. Intel has reported that the difference in the number of electrons in each of the above states is about 3000 electrons [22]. Hence, if data retention of about 10 years is required, then before a change in the state can be achieved the cell must lose about 300 electrons per year.
which would imply a little less than an electron per day. This is a very sensitive issue. If electron loss occurs from even one cell in an array of millions, the data will be corrupted. The inherent storage capacity of a flash cell exists due to the Si-SiO₂ energy barrier which traps electrons in the floating gate. There may be conditions of trapped oxide charge known as intrinsic charge loss which can cause one time shifts in threshold. Random defects in the insulating oxides can lead to charge loss which may again cause a shift in threshold resulting in an erroneous value to be stored in the floating gate. The remaining concern for charge retention is any degradation to the insulating oxides that occurs as a result of the stresses in the device operation. This type of defects or charge losses cause major issues in MLC, and thus is a cause of major concern during design and testing [14]. The fault model discussed in Chapter 4 deals with various intrinsic and external factors which could lead to threshold shift, causing errors in the array.

2.6 Multi-Level Cell Threshold Voltage Distribution

Having discussed charge placement, we can now draw a plot to show the relationship between the number of flash cells in the array and their various Vt levels just as in Section 5.3 for SBC flash memories. Let us assume for simplicity that 75% of the cells in the array core are programmed and 25% of the cells are erased. If every cell in the array has been sensed by either one of the methods described in Sections 6.3 and 6.4, then we can draw a plot between the number of cells and the threshold voltages of all the cells. This is called the Vt distribution graph (Fig. 2.16) [14]. The number of cells in the array is on the Y axis, while the range of the threshold voltages Vt of all the cells is represented on the X axis. Let all the cells which are erased be represented by region 1, and all the cells which are programmed be represented by regions 2, 3 and 4. The distribution depicted arises from the fact that every cell in the array that is programmed need not have a constant Vt, even
though it may be still considered programmed (and the same applies to all cells which are erased).

This is a noted fact especially in MLC flash, since there are various "programmed levels" which may also have a wide distribution around a maximum average. The Vt distribution of the array is as shown and the various MLC levels are "11" (erased), "10" (programmed), "01" (programmed) and "00" (programmed).

![Multi-Level Flash Cells Diagram](image)

**Figure 2.16: Multi-Level Threshold Voltage Distribution.**

For SBC, in Fig. 2.16, a reference voltage is fixed between threshold voltage of the least erased and the least programmed cell. This is the read window / margin, and any voltage below the reference voltage and outside this window, is assumed erased; conversely, any voltage above this reference and outside the window, is assumed programmed. However, when we consider the multi-level flash distribution, clearly there are four states / levels which can be realized. Hence, there must be a very precise reading of these levels by suitably choosing reference voltages in the window between two adjacent levels. The window between state 11 and state 10 for example would be the Vt gap between the least erased cell of state "11" and the least programmed cell of state "10". The above diagram discussing the window budget for effective sensing can be summarized in Table 2.5. This does consider the transient offset of all the states as well as the reference voltage.
The reference voltages indicated by dashed lines in Fig. 2.16 are actually a range of values. Also there will be a transient time until the final reference voltage peak value is achieved for sensing. Thus, considering the transient offset of each reference voltage, the actual window budget is further reduced as shown in Fig 2.17.

<table>
<thead>
<tr>
<th>No.</th>
<th>Vt Window</th>
<th>Ref. Voltage</th>
<th>State Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vev &amp; Below</td>
<td>Vev</td>
<td>Erased</td>
</tr>
<tr>
<td>2</td>
<td>Vev to Vpv1</td>
<td>Vr1</td>
<td>Programmed</td>
</tr>
<tr>
<td>3</td>
<td>Vpv1 to Vpv2</td>
<td>Vr2</td>
<td>Programmed</td>
</tr>
<tr>
<td>4</td>
<td>Vpv2 to Vpv3</td>
<td>Vr3</td>
<td>Programmed</td>
</tr>
<tr>
<td>5</td>
<td>Vpv3 &amp; Above</td>
<td>Vpv3</td>
<td>Programmed</td>
</tr>
</tbody>
</table>

Figure 2.17: Multi-Level Read Window.

The total window budget is between EV and PV3. The window for each state is represented by the lower case letters for each state. For example, VR1 can vary between R1 box and the window b is the distance between this box and the state 10 box. Hence, if a larger number of states are required to be stored within a flash cell, then the corresponding read window between each pair of adjacent states will be reduced. This is a major concern, as in such a scenario any kind of fault
will cause the Vt of the victim flash cell to change, and may cause it to shift another state. In other words, this causes data corruption as if the voltage state changes, and the bits decoded by the sensing circuitry will be different from the intended original programmed value. This is also the basis for the non-trivial fault model discussed in Chapter 4 - The MLC Flash Fault Model.
Chapter 3

The Multi-level Cell Architecture

This chapter deals with selecting an appropriate MLC architecture to apply the march test and diagnosis algorithms. Chapter 2 dealt with the various types of flash memory, namely conventional single bit per cell and advanced multi-level cell technologies. There are many types of SBC architectures. The beauty of MLC technology is that any suitable SBC architecture can be implemented to realize a MLC product. This chapter will discuss a few of the SBC architectures very briefly, and then a suitable architecture will be chosen to test the MLC march algorithm. The MLC architecture strongly governs: the faults that occur and the faults that can be detected.

Therefore, before a comprehensive fault model can be built, a thorough understanding of the MLC architecture is required. After the MLC architecture is discussed, the next step would be to determine a suitable addressing scheme. The last section deals with the addressing scheme that will be used to access the flash array and pass test vectors to program the individual flash cells to various levels.
3.1 The Flash Array Layout - Full Chip Level

A typical chip level diagram of a flash memory is depicted in Fig. 3.1. Different architectures will have various placements of the sub-blocks. Fig 3.1 depicts a typical full chip layout with the flash memory included.

![Diagram of Flash Array Layout](image)

A typical flash chip will have a flash core which is commonly divided into two partitions. The sense amplifiers and decoding circuitry are usually placed between the two sub-cores and are shared between them. There is also a provision for redundancy as a built-in self repair (BISR) scheme. The high voltage, read path, digital logic and voltage pump blocks are optimally placed beneath the array as shown. Usually a square or near-square shape is preferred as this will yield
the maximum number of die per wafer.

3.2 Background: Traditional Flash Memory Architectures

Flash array architectures may be broadly divided into three main categories namely NOR, NAND and AND as shown in Fig. 3.2. The NOR array architecture is the most common one, and a large percentage of flash memory products in the market today have the NOR architecture. The following is a brief list of the various flash architectures.

1. NOR
   - Virtual Ground: AMG (Alternate Metal Ground), Split Gate (Poly-Poly Erase & Merged).
   - Common Ground: Source Injection, Standard, DINOR, EPROM.

2. NAND

3. AND
   - ACEE (Array Contactless EEPROM)
   - AND
   - HiC (High Coupling Ratio)

It is not possible to discuss each of the above architectures in detail. They can be commonly found in literatures. This discussion is limited to the best architecture which can serve as a good platform to build a fault model. After analyzing various faults which occur in single bit per cell flash memory cores, we find that the Industry Standard Common Ground NOR architecture is the best choice. This NOR architecture will be elaborated in the next section.
3.3 A Simple Flash Structure

The flash cells are arranged as a matrix of rows and columns as shown in Fig 3.3. All the gates of the flash cells are connected to the word lines, and the drains of the cells are connected to the bit lines. The sources of all the flash cells need to be grounded. This is done by connecting each of the sources to a local source column which is grounded. However, a more area-saving scheme is to connect the sources of cells from a bunch of columns to a source strap. For the architecture shown in Fig 3.3, there is one global source strap for every 8 local columns.

For simplicity, a 3 x 3 flash cell matrix subset of the array is chosen. The word lines are named A, B and C and the bit lines are numbered 1, 2 and 3. (There is one local source strap for this matrix.) The cells connected to each of them are named A1 through C3 appropriately.
3.4 The Flash Memory Block

3.4.1 The Array Core Division

The size of an array core is dependent on the intended target application. The size of a common flash core can range from 16Mbit to 512Mbit. Recently, flash cores of 1Gbit each have also been fabricated. The test array architecture under consideration in this work is limited to a size of 32Mbits or 32M flash cells. The core is divided into smaller sub-components. The smallest sub-core unit is defined as a block. The array division is depicted in Fig 3.4.

Thus, after calculating the size of each block, a closer look within its inner architecture is required. Each 0.5Mbit block is realized as a matrix of 512 rows and 1024 columns of flash cells. Therefore,
From Fig 3.4: (Top Down Approach):-

*The array core (32M bits) has four strips.*
⇒ *Each strip contains 8M bits or 8M flash cells each.*

*Each strip contains two planes.*
⇒ *Each plane contains 4M bits or 4M flash cells.*

*Each plane contains eight blocks.*
⇒ *Each block contains 0.5M bits or 0.5M flash cells.*

Conversely: (Bottom Up Approach):-

8 blocks make up 1 plane
⇒ 0.5M Bits * 8 blocks = 4M Bits

2 planes make up 1 strip
⇒ 2 planes * 4M Bits = 8M Bits

4 strips make up the array core
⇒ 4 Strips * 8M Bits = 32M Bits

there are 512 word lines connected to the gates, and 1024 bit lines which are connected to the drains of the flash cells as described in the 3 x 3 matrix in Section 4.

3.4.2 The Flash Array Block Internal Architecture

As stated above, each block is 0.5Mbits in size and contains 512 rows and 1024 columns. The 1024 columns are split into 16 groups each containing 64 columns (16 × 64 = 1024 columns) (Fig. 3.5). This is to facilitate easy decoding and addressing (explained shortly). This grouping is to facilitate the simultaneous reading of 16 bits at a time, thus implying that a single memory address will read out 16 bits or 1 word of data at a time.
Figure 3.4: The Flash Array Core Division / Hierarchy.

Each of the 16 groups contains 64 local columns. There are also overhead global columns which are connected to the local columns. There is one global column for every 8 local columns. Hence in each sub-block, there are 8 global columns and 8 local columns/global column. The flash cells are indicated by the black dots present at the intersection of the rows (word lines) and columns (bit lines). When an address is specified at the address bus, the sixteen column decoders select the same bit line number from each of the 16 groups and send them to their respective outputs. These 16 outputs are compared with the outputs of the reference cells to determine the logic value stored in each corresponding flash cell. The resulting value is sent as a 16bit/word output. Alternatively, if the local columns were split into 32 groups of 32 columns each then a single address access would
output 32 bits or 4 words at a time. Thus, the bandwidth of the architecture under consideration will determine the grouping of the columns within each block.

Let there be a total of "T" columns in a block. If these T "columns" are split into "N" groups of a "S" global columns each, then the total number of columns,

\[ T = (\text{No. of Groups}) \times (\text{No. of global columns / group}) = N \times S \]  
\[ T = (\text{No. of Groups}) \times (\text{No. of local columns / group}) \times 8 = 8 \times N \times S \]
3.5 The Flash Memory Addressing Scheme

After the memory architecture has been discussed, a suitable technique must be developed to access this array. Since the array core is split up into various subcomponents, a step-by-step analysis of the array will yield a simple but effective scheme of accessing every flash cell within the core. Referring to Subsections 5.1 and 5.2, now the addressing scheme is discussed with a suitable example for clarity.

3.5.1 The Array Analysis

The array core is divided into four strips of two planes each;

⇒ Total number of planes in the core = 8
⇒ Total no. of bits required to represent each plane = 3 bits.......... (1)

Each plane is divided into 8 blocks;
⇒ Total no. of bits required to represent each block = 3 bits ..........(2)

Each block contains 512 rows; 512 = 2^9;
⇒ Total no. of bit required to represent each row = 9 bits ..........(3)

There are 1024 columns split up into 16 identical groups of 64 columns each. Since 2^6 = 64, we require a total of 6 bits to represent the columns within each of the 16 groups. Moreover, since there are 8 global columns, there will be a total of 8 global columns per group of 64 columns.
⇒ Total no. of bits required to represent each of the global columns = 3 bits ...........(4)

Since there is 1 global column for every 8 local columns;
⇒ Total no. of bits required to represent each local column / global column = 3 bits ...........(5)

Adding the number of bits obtained in results (1), (2), (3), (4) and (5),

3 bits for 8 planes / core
+ 3 bits for 8 blocks / plane
+ 9 bits for 512 rows / block
+ 3 bits for 8 global columns / 64 columns
+ 3 bits for 8 local columns / global column

Total = 21 bits

In summary, a 21-bit address is required to completely access the 32Mbit MLC flash array. This is
graphically represented in Fig 3.6.

Chapter 5 will deal with the testing methodology (march algorithms) to test the flash
array under consideration. Since every march algorithm, involves traversing addresses either in
an increasing or decreasing order, we must now relate the memory architecture to this testing methodology. March algorithms traverse address locations in succession. In other words, the difference between any two successive addresses of a march algorithm will be only one bit. For example, while traversing addresses from address "0" to address "63" (along a row), any two successive addresses will differ only in the local column bit values. Thus by gradually incrementing the local bit line bits, all cells along a row may be traversed. Similarly, while traversing from one row to the next, the corresponding row address bits need to be incremented. This is how every flash cell in the array will be accessed during testing and diagnosis.
Chapter 4

The Multi-level Cell Flash Fault Model

This chapter deals with the various faults that occur in flash memory architectures. These faults are common to both single bit per cell flash memories as well as multi-level cells. There are faults, however, that affect MLC flash architectures to a larger extent than SBC due to the sensitivity of Vt levels within each cell. Certain faults which can be neglected in SBC FMs cannot be overlooked in MLC products, as their impact causes intolerable errors which cannot be rectified by common techniques such as BISR.

The MLC architecture selected was discussed in Chapter 3. The traditional SBC fault model is first discussed in Section 2. These faults can be extended to MLC as well, since their presence has a greater impact due to the reduced Vt window for each state. The common faults that occur in flash memories have been dealt with in Sections 2 and 3. The traditional faults have been discussed in Section 4 and the non-traditional faults are discussed in Section 5. These faults will be explained in detail and their impact on MLC Flash memories will be emphasized.
4.1 The Traditional SBC Fault Model

The high testability of flash memories, allows the detection of defects (faults), which may cause single-cell failures due to *programming defects, data-retention* and *oxide defects*, thus making flash one of the most reliable semi-conductor memories. There are various factors both intrinsic to the flash cell and those around its periphery which cause significant errors to the logic value stored on the floating gate. This section refers to the issues related to SBC flash memories alone. The subsequent sections deal with MLC-related issues, and a suitable MLC fault model will be formulated and discussed extensively.

4.1.1 Program Disturbs

Program Disturbs due to Gate Stress

Each cell in the flash array is subject to high voltages both on the gate and drain. Hence, it is very prone to defects due to *program disturbs*. Program disturbs affect cells sharing either the word line or the bit line of a cell addressed for programming. During programming, all the cells along a particular word line are subject to what is called *gate stress*. There are two types of programming disturbs due to gate stress namely: *DC-Erasing* and *DC-Programming*.

Program Disturb Gate Stress: DC-Erasing: The gate stress described above can cause charge loss in a programmed cell due to the tunneling of electrons from the negatively charged floating gate to the highly positively charged gate (approx. 12V). Hence, if this charge loss is significant, it can definitely cause the Vt to shift from the programmed state to the erased state. Thus, the fault effect is that logic 0 is erroneously converted to logic 1. Since this is done with a high positive DC voltage applied to the control gate, this type of program disturb is called *DC erasing* or a *word line erase disturbance*. 

49
Program Disturb Gate Stress: DC-Programming: The gate stress can also cause the opposite effect in flash cells. The high voltage (approx. 12V) applied to the control gate of a cell can attract electrons and cause them to tunnel across the tunnel oxide from the channel into the floating gate. If the number of electrons that get injected onto the floating gate is significant, then the state of the cell can be converted from an erased state (logic 1) into a programmed state (logic 0). Hence, this type of fault can have an impact on the data value stored in the cell. Again, since this is done with a positive DC 12V gate voltage, this type of program disturb is called DC programming or a word line program disturbance.

Program Disturbs due to Drain Stress

Cells sharing the same drain (bit) line, are subject to high voltage (5-7V) during programming [23], and this is very similar to gate stress. This can causes programmed cells which have electrons in the floating gate to lose some charge due to the positive drain voltage, causing charge loss. This is called a bit line erase disturbance or sometimes a soft-erase. The state of the cell can be changed from a programmed state (logic 0) to an erased state (logic 1). This can cause serious problems just as in the case of the gate stress type program disturbs. Similarly, the opposite case of a bit line program disturbance may also occur. A bit line program disturbance fault occurs when a program operation to a cell causes another cell on the same bit line to also get programmed to the same value. During a program operation on the aggressor cell, a depletion region is formed under the victim cell. The electrons entering this depletion region can be accelerated by the electric field. This causes them to be injected over the gate oxide potential barrier to the floating gate [23]. The effect of this mechanism will be that the victim cell will now be programmed.
These program disturbs affecting the overall reliability of the array, become more important with the increasing or decreasing number of programming cycles, and must be minimized with a tailored choice of the programming voltages and optimal circuit design. However these faults cannot be completely eliminated.

**Read Disturbs**

A read disturb fault occurs on flash cells since the operating conditions are very similar to programming [23] but lower in magnitude [14]. This can cause two kinds of read disturbs on the affected cell in the erased state [14]. Firstly, a flash cell could be programmed due to CHE injection if the drain voltage is not low enough. Secondly, all cells in the erased state will be subjected to a low-voltage gate stress on the control gates of the flash cells. Hot electrons will be injected from the channel into the floating gate even if there is a low gate voltage applied. This again results in programming. In both cases, the oxide current is very low, but this may still cause the cell to read a faulty value [14]. Moreover, tunnel oxide degradation due to cycling may also be one of the main causes for read disturbs. During the life cycle of the flash array, the number of read operations could potentially be infinite and hence these operations wear out the oxide thus facilitating such defects or disturbs. One way to alleviate this problem due to cycling, is to use lower drain voltages (approx. 1v).

### 4.1.2 Over-Erase Fault

Considering the Vt distribution of SBC flash, sometimes there may be cells which get over erased. They cause the Vt distribution to have a tail-like structure due to the Vt dispersion, as they erase much faster than a majority of flash cells in the array. The dispersion of threshold
voltages is due to coupling ratio variations. (The coupling ratio is defined as the ratio between capacitance between two terminals and the total capacitance. Eg: $C_g / C_T$, $C_g$ is with reference to a grounded source). These cause the flash to store incorrect values, since the programming voltage supplied will not be able to shift its Vt into the programmed cell Vt distribution. There is no fixed reason for such cells to erase faster causing *tail bits*. It is assumed that there are certain inherent defects in the oxide crystal structure causing random positive charges in the tunnel oxide. There is a well-known existence of donor-like bulk oxide traps. Electrons get off the floating gate at a faster rate resulting in higher tunnel current density due to these traps. Hence, the tail of the erase distribution can be attributed to structural imperfections, i.e. intrinsic defects, so attempts can be made to minimize them by process improvements. However, if they occur after manufacturing, then they result in the flash cells storing faulty values. This problem will not have much of an effect for SBC, unless the cell has a negative Vt (or an extremely low Vt) which does not increase the Vt to the programmed state even after the maximum program pulses are applied. This type of over erase faults, however, will definitely have an adverse effect in MLC Flash. If the state of a cell is not erased, (say state 01) and due to inherent defects in the cell (oxide) electrons will flow out of the floating gate causing the Vt to drop. If this drop is significant, then the defect will cause the cell Vt breach the window between the 10 and 01 states and the cell now stores a faulty value of 10.

Having discussed the single bit per cell, the multi-level fault model will be formulated and the various types of MLC faults shall be discussed.
4.2 Fault Classification

This section deals with the various types of faults that may occur in the flash array. Faults cannot be broadly classified on the basis of SBC or MLC faults, as there are certain faults which are common to both types of flash memories. Since the architectures of SBC and MLC are the same, the faults that occur in SBC arrays can very easily be extended to MLC arrays. Hence, in general, any type of flash memory (SBC / MLC) fault may be broadly classified into two main categories: traditional/obvious faults and non-traditional faults.

4.2.1 Definitions

Traditional Faults

Traditional faults are those faults which occur in conventional semiconductor memories such as SRAM, DRAM and even SBC flash. They do not require much analysis, as they are very common in literature. If these faults are present, their detection by the algorithm is sufficient. There is no need for a comprehensive proof that these faults exist.

Non-Traditional Faults

Non-traditional faults are faults which occur in SBC, but do not have much impact on the SBC Vt state. The effect of these faults is much more predominant in MLC arrays. They require some form of theoretical analysis to describe their behavior. The presence of these faults may cause the Vt of the MLC to change thus causing errors. Hence these faults are very important and form a major part of the MLC flash fault model.

4.2.2 Flash Memory - Traditional Fault Listing

The following is a brief list of all the traditional faults that will be dealt with in this research.

1. MLC stuck-at faults
a. Stuck-at 00
b. Stuck-at 01
c. Stuck-at 10
d. Stuck-at 11

2. Disturb faults

3. Bridging faults

4. Broken connection faults

4.2.3 Flash Memory - Non-Traditional Fault Listing

The following is a brief list of all the non-traditional faults that will be considered in this research.

1. Series resistance faults

2. Faults due to traps in the oxide

3. Faults due to charge leakage
   a. Faults due to leakage along a column
   b. Faults due to leakage along a row
   c. Faults due to gate-drain coupling (extremely low Vt)

4. Stress induced leakage current (SILC)

5. Faults due to contamination / process residue (ion) faults
   a. Contamination charge loss fault
   b. Contamination charge gain fault
The traditional and non-traditional faults are summarized in Table 4.1

<table>
<thead>
<tr>
<th>Fault Category</th>
<th>S.No.</th>
<th>Type of Fault</th>
<th>Sub-Faults</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>1</td>
<td>Stuck-at Faults</td>
<td>Stuck-at 00</td>
<td>MLC Only</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>&quot;</td>
<td>Stuck-at 01</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>&quot;</td>
<td>Stuck-at 10</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>&quot;</td>
<td>Stuck-at 11</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Disturbance Faults</td>
<td>——</td>
<td>SBC / MLC</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Bridging Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Broken Connection Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td>Non-Traditional</td>
<td>8</td>
<td>Series Resistance Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Faults due to Charge Leakage</td>
<td>Along a Column</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>&quot;</td>
<td>Along a Row</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>&quot;</td>
<td>Gate-Drain Coupling</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Contamination Faults</td>
<td>Charge Loss</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>&quot;</td>
<td>Charge Gain</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Faults due to Oxide Traps</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>SILC Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

4.3 Traditional Faults in Multi-Level Flash Memories

4.3.1 Stuck-at Faults

"Stuck-at" faults which are a well-known problem in standard SRAM memories also exist in flash arrays. For MLC products, there are four new stuck-at faults that exist.

**Stuck-at Faults due to Errors in Charge Pumps**

Flash memory chips are usually single power supply devices. This means that there is only a one power pin/source which is set at Vcc (approx 5v). The positive and negative charge pumps present on-chip, step up or step down the voltage as required by various process in the chip. For example during programming, the charge pumps for each voltage level on chip need to generate the voltage in order to realize multiple voltage levels to store more than one bit. (Eg. 11, 10, 01 and
Due to the high power and heat dissipation that occur in the analog areas of the chip, there is a very good chance that the positive or negative charge pumps may not exhibit this flexibility that MLC flash requires. As a result, the charge pump of each voltage level can be stuck at a particular value. This can be due to a broken connection or destructive capacitive coupling which causes the charge pumps to remain at a particular voltage level. When these error prone charge pumps are connected to the array, the programming voltage levels cannot be varied. Therefore, the flash cell under consideration will always be programmed to a fixed voltage level corresponding to the charge pump error.

**Stuck-at Faults due to MUX Faults**

All MLC flash architectures contain X and Y decoders which have access to all the bit and word lines in the array. Once the decoding is complete, the charge pump outputs (voltage levels corresponding to 11, 10, 01 and 00) are connected to the word lines via a multiplexer. Hence, during programming, due to a short-circuit / fault within the MUX, if the decoded word line is always connected to one particular charge pump output, the cell will be programmed to only one particular voltage level. The decoder would always pass a fixed voltage to the word lines causing the cell to be constantly programmed to the same level whenever selected.

In a MLC product, there may be instances where certain cells may be shorted to Vcc or to a source strap. A connection to Vcc would be similar to the highest programming state voltage level, and hence would result in a stuck-at 00 fault. Similarly a short between the floating gate and the source (strap) would result in depleting all the electrons off the floating gate and this will result in a stuck-at 11. Stuck-at faults at states 10 and 01 are not common, but may occur due to oxide
traps or leakage. Hence, due to the reasons stated above, there may be four such stuck-at faults for a MLC flash array, namely stuck-at 11, stuck-at 10, stuck-at 01 and stuck-at 00. The state diagrams for the above states is depicted in Fig. 4.1.

![State Diagrams for Stuck-at Faults.](image)

**Figure 4.1: State Diagrams for Stuck-at Faults.**

### 4.3.2 Bridging Faults

In general, bridging faults are shorts which occur between two signals within a chip. In a flash array, a bridging fault occurs when there is a short between two word lines or two bit lines. A sample flash array is depicted in Fig. 4.2 to illustrate the impact of a bridging fault in MLC arrays. The concept of bridging faults in flash memories is highly associated with the architecture under consideration. Since the architectures of SBC and MLC flash memories are the same, they have an equal probability of occurring in both types of arrays. Since there are a many voltage states in each MLC cell, the impact of bridging faults in an MLC product cannot be ignored.

Consider the 3 x 3 array in Fig. 4.2. There are three word lines named A, B and C and three bit lines numbered 1, 2 and 3. The flash cells at the intersection of these word and bit lines are named A1 through C3 appropriately. Consider cell B2. This is the correct cell to be programmed. The bit and word lines associated with this cell are indicated by the thick lines. During the programming of cell B2, word line B needs to be set to high voltage in order to activate the channel. Soon afterward the drain voltage (bit line) needs to be supplied with an appropriate voltage. These are the generic
conditions to program a cell (B2) as described in Chapter 2.

Case 1: Two Word Lines Shorted

Now, consider the situation where a bridging fault occurs between word lines A and B as shown in Fig. 4.2 by the thick vertical line. When a voltage is applied to word line B (correct word line), due to the bridge/short between A and B, even line A gets the same voltage. So, now the channels of two cells namely A2 and B2 are activated. When a programming biasing voltage on the bit line 2 is given, since both cells are activated, they are both programmed. This is the result of a bridging fault, and this causes the cell on the associated bit line to be unintentionally programmed. Here, B2 would be analogous to an aggressor cell and A2 would be analogous to a victim cell. The
victim in this case follows the aggressor, if the applied voltage to the aggressor is greater than the Vt value of the victim cell.

Case 2: Two Bit Lines Shorted

Now, consider the situation when a bridging fault occurs between two bit lines 1 and 2 as shown in Fig. 4.2 by the thick horizontal line. When a voltage is applied to word line B, the channels of all cells connected to word line B are activated. So, when bit line 2 is given the biasing voltage, the same voltage gets applied to bit line 1. As a result, cells B1 and B2 get programmed. This is the result of the bit line bridging fault, and causes the cell on the associated word line to be unintentionally programmed. In this case, B2 is analogous to an aggressor cell and B1 is analogous to a victim cell. So, the victim follows the aggressor, if the applied word line voltage is greater than the Vt of the victim cell.

4.3.3 Broken Connection Faults

All flash cells within the array obviously need physical connection to the gate and drain to perform operations such as read, program and erase. A fault such as a broken connection can cause serious repercussions, as one fault may render a large portion of the array unusable. There are three major connections to a flash cell which can render the cell and in some cases all the subsequent cells on the associated line completely unusable. Consider the flash cell in Fig. 4.3. There are three regions where a broken connection fault may occur:

- Broken word line connection (disconnected from gate)
- Broken bit line connection (disconnected from drain)
- Broken source-ground connection (disconnected from source)
Figure 4.3: Broken Connection Fault Possibilities.

**Broken Word Line Connection**

Referring to the flash architecture under consideration (Chapter 3), the word lines run horizontal along the chip. There are two regions where a broken connection could occur:

**Case 1:** Consider Fig. 4.4. If the fault occurs in the location depicted in the figure, then only the cell under consideration can be rendered unusable, as the programming voltage can never reach the gate of the cell. The fault location is local to the cell alone before it reaches the overhead word line. Hence, this cell can never be programmed to any of the levels namely 10, 01 and 00. If this occurs at the time of fabrication, then the cell can never be erased either, i.e., state 11 can never be reached either.

**Case 2:** Consider Fig. 4.5. There are two regions where there is a broken connection on the overhead horizontal word line. If there is a broken connection at location 2, then all cells toward the right (not including the cell under consideration) will be rendered unusable, as the applied
voltage will never reach the gates of the affected cells. If there is a broken connection at location 1, then all cells toward the right (including the cell under consideration) will be rendered unusable (i.e., cannot read or program).

**Broken Bit Line Connection**

Referring to the flash architecture under consideration, the bit lines run vertical along the chip. Hence, again, as in the case of word lines there are two cases to consider:

**Case 1:** Consider Fig. 4.6. If the fault occurs in the location depicted, then only the cell under consideration cannot be used. The channel will be activated but there will be no drain current
which will cause electrons to flow within the channel. Hence, the cell cannot be programmed or read easily, but this cell can be effectively erased as the drain is usually kept floating during a FN erase.

**Case 2:** There are two regions where there is a broken connection on the overhead vertical bit line as indicated in Fig. 4.7. If there is a broken connection at location 1, then all the cells below the cell (including the cell under consideration) will be rendered unusable. If there is a broken connection at location 2, then all cells below the cell (not including the cell under consideration) will be rendered unusable, (cannot read or program). Again just as in case 1, the channels of the cells will be activated by the word line but there will be no drain bias current to inject electrons into the channel of the device. (Again, this cell can be effectively erased as the drain is usually kept floating during a FN erase).

**Broken Source-Ground Connection**

Referring to the flash architecture under consideration (Chapter 3), each flash cell has a connection to ground. Each column in the array can possess a source line running in parallel to
the bit line to ground, or there can be a source-strap type architecture where the sources of all flash cells are connected to one global source strap which is grounded. For the architecture under consideration, there is one global source strap for every eight local columns. If there is a broken connection between any cell and the source strap, then the source of that individual cell will be disconnected from ground, and as a result the cell cannot be read, programmed or erased. If there is a broken connection along the source strap, then all the eight columns of cells connected to it will be disconnected from ground, and will lead to a memory loss of $512 \times 8 = 4096 = 4K$ bits. This broken connection has the most impact as a large portion of the array can be rendered unusable.

4.3.4 Disturb Faults

Single bit per cell (SBC) flash deals with the traditional fault model of disturbs occurring during normal operation. This traditional fault model can be extended to multi-level flash cells (MLC).

Program Disturbs due to Gate Stress

The cells in the flash array are subject to high voltages both on the gate and drain (approx. 3 to 12V). Hence, they are very prone to errors due to program disturbs. Program disturbs affect
cells sharing either the same word line or the same bit line of a cell addressed for programming. During programming, all the cells along a particular word line are subject to what is called gate stress. There are two types of programming disturbs due to gate stress namely: DC - Erasing and DC - Programming.

Program Disturbs due to Gate Stress: DC-Erasing

The gate stress described above can cause charge loss in the programmed cells due to the tunneling of electrons from the negatively charged floating gate to the highly positively charged gate. Hence, if this charge loss is significant, it can definitely cause the Vt value of a cell to shift from the programmed state to other states. The fault is that, a high Vt state is erroneously converted to lower Vt state. Since this is done with a positive DC voltage, this type of program disturb is called DC erasing or word line erase disturb. If a cell is in a programmed state such as any of the 00, 01 or 10 state, due to this charge loss, the net Vt of the cell will reduce causing the Vt to breach the MLC state window budget, and its Vt shifts into the next adjacent lower Vt state. Hence, DC erasing can cause serious problems as, during a subsequent read, an error can occur as the cell is now decoded to store a faulty value.

Program Disturbs due to Gate Stress: DC-Programming

The gate stress can also cause the opposite effect in flash cells. The high voltage (approx. 3 to 12V) applied to the gate of a cell can attract electrons and cause them to tunnel across the tunnel oxide from the channel into the floating gate. If the number of electrons that get injected onto the floating gate is significant, then the state of the cell can be converted from an erased state (state 11) into a programmed state, or any programmed state into a higher Vt state. Hence this type of fault can have an impact on the data value stored in the cell. Again, since this is done with
a positive DC gate voltage, this type of program disturb is also called DC programming or a word line program disturb. If the cell is in an erased state or any of the lower programming states such as 10 or 01 respectively, then due to the inflow of electrons to the floating gate the Vt value can increase. If the MLC state window is breached, the cell Vt value may shift to a new state. The program disturbs that occur in an MLC flash are summarized in Table 4.2 and Table 4.3.

**DC - Erase Faults:**

Table 4.2: DC - Erase Faults

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Original</th>
<th>Disturb Type</th>
<th>Final Disturb State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>DC-Erase</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>DC-Erase</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>DC-Erase</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>DC-Erase</td>
<td>Over-Erased</td>
</tr>
</tbody>
</table>

**DC - Program Faults:**

Table 4.3: DC - Program Faults

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Original</th>
<th>Disturb Type</th>
<th>Final Disturb State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>DC-Program</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>DC-Program</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>DC-Program</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>DC-Program</td>
<td>00</td>
</tr>
</tbody>
</table>

**Program Disturbs due to Drain Stress**

Cells sharing the same drain (bit) line, are subjected to high voltage (5-7V) during programming [23], and this is very similar to gate stress. This can causes programmed cells which have electrons in the floating gate to lose some charge due to the positive drain voltage, causing charge loss. This is called a *bit line erase disturbance* or sometimes a *Soft-Erase*. For example, if the state of a cell is say 01, due to a drain stress fault, the electrons are attracted to the drain because of the positive drain voltage and as a result, electrons from the floating gate are taken off resulting in
state 10. Similarly, the opposite case of a bit line program disturbance may also occur.

A *bit line program disturbance* fault occurs when a program operation to a cell causes another cell on the same bit line to also get programmed to the same value. During a program operation on the aggressor cell, a depletion region is formed under the victim cell. The electrons entering this depletion region can be accelerated by the electric field. This causes them to be injected over the gate oxide potential barrier to the floating gate [23]. The effect of this mechanism will be that the victim cell will now be programmed. In MLC products, this is an even more serious problem as the Vt window budget is much more narrow. Hence, such shifts in Vt could cause errors to show up when the value stored in the cell needs to be decoded. For the above example, electrons can get accelerated onto the floating gate and hence can cause the state to change from state 01 to state 00. This would lead to an error when a read operation is performed on the cell. The program disturb drain stress are summarized in Table 4.4.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Original</th>
<th>Disturb Type</th>
<th>Final Disturb State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>Drain Stress - E</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>Drain Stress - E</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>Drain Stress - E</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>Drain Stress - E</td>
<td>Over-Erased</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>Drain Stress - P</td>
<td>00</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>Drain Stress - P</td>
<td>01</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>Drain Stress - P</td>
<td>10</td>
</tr>
</tbody>
</table>

These programs disturbs affecting the overall reliability of the matrix become more important with the number of programming cycles and must be minimized with a tailored choice of the programming voltages and optimal circuit design. However these faults cannot be completely eliminated.
4.3.5 Read Disturbs

A read disturb fault occurs on flash cells since the operating conditions are very similar to programming [23] but lower in magnitude [14]. This can cause two kinds of read disturbs on the affected cell in the erased state [14]. Firstly, a flash cell could be programmed due to CHE injection if the drain voltage is not low enough. Secondly, all cells in the erased state will be subjected to a low-voltage gate stress on the control gates of the flash cells. Hot electrons will be injected from the channel into the floating gate even if there is a low gate voltage applied. This again results in programming. In both cases, the oxide current is very low, but this may still cause the cell to read a faulty value [14]. For example, if the state of the cell is currently "state 10", then due to the above two reasons, the state of the cell after such a read disturb would "state 01". Moreover, tunnel oxide degradation due to cycling may also be one of the main causes for read disturbs. During the life cycle of the flash array, the number of read operations could potentially be infinite and hence these operations could wear out the oxide thus facilitating such defects or disturbs. One way to alleviate this problem due to cycling, is to use lower drain voltages (approx. 1v).

4.4 Non-Traditional Faults in Multi-Level Flash Memories

4.4.1 Series Resistance Faults

As described in Chapter 2, during erasing, the source must be fabricated differently from the drain in order to withstand the high erase current (Fig. 4.8), where electrons are ejected out of the floating gate and into the source. The material must be such that, the erase current be kept to a minimum value, i.e., the material of the source must be more resistive than a standard CMOS transistor. In other words, the resistance of the source must be large in order to limit the erase
current. However, due to process variations, in highly sensitive MLC flash arrays there could be an instance where the source resistance is too high. This affects the read operation to large extent and can be explained as follows.

As discussed in Chapter 3, the current that flows in the bit line due to a read operation is determined by two factors:

- The Vt of the cell under consideration, and
- The drain - bit line resistance. This decides the value of Icell and this in turn determines the value of the bits decoded to be stored in the array cell.

Mathematically, we have, \( I_{cell} = f(V(Rm), Vt) \). The extra source resistance of the flash cell is shown in Fig. 4.9. This causes the Vt of the cell to change. Hence, the drain current Icell is now represented as:

\[
I_{cell} = f(V(Rm), Vt')
\]

Where \( Vt' = \Delta Vt + Vt \).

Figure 4.8: The Cross Section of a Flash Cell.

68
△$V_t$ is the change in $V_t$, due to the increased source resistance as there will be a higher channel current required in order to program a cell to a required level at a particular $V_g$. Hence, in an MLC product if the current through the cell changes due to this increased threshold voltage $△V_t$, the voltage at node B (Fig. 4.9), would be read differently. The programming time will increase and in certain cases if the maximum number of programming pulses has been reached, then an error is raised due to a program fail and the cell will remain at that incorrect programmed value.

If the window budget (Chapter 2) is small, then the state of cell would be incorrectly placed to a neighboring state and the bits decoded by the logic circuit will change thus leading to an error. Though these faults are very rare as most fabrication processes tend to minimize such increased resistance, they cannot be neglected. In fact, they play an important part in MLC testing, and must be included in the MLC flash fault model.

**Example:** If the cell is erased or in state 11 and is intended to be programmed to state 10,
then due to the increased source series resistance, it would take a longer time to program it to a 10 state. In such an instance, if the maximum number of program pulses has been exceeded, then the cell remains in the same state which is state 11. When the cell is read again, instead of reading a state 10, it is read as 11. This can be summarized in Table 4.5.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Initial State</th>
<th>Intended State</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

4.4.2 Oxide Trap Faults

This section has been extracted entirely from [15].

The Interpoly Oxide shares a Si-SiO2 with the gate and floating gate. Some of the bonds are stretched and dangling and these are believed to produce sufficiently strong localized potential perturbations to generate bound states for electrons or holes with energy levels in the forbidden gap. These levels act as localized electron or hole traps, i.e., they can capture or release carriers. Traps can be of two types:

- *Acceptor traps* which are neutral when empty and negative when occupied.
- *Donor traps* which are neutral when occupied and positive when occupied.

In MOS structures, interface traps are believed to be mostly of acceptor type in the upper half of the band gap and of donor type in the lower half. Carrier trapping and de-trapping can occur through a large number of processes, namely;
1. Thermal capture and emission

2. Photon assisted transitions

3. Particle assisted transitions

4. Elastic tunneling in and out of traps

The steady state occupancy of the traps would change with the bias conditions, according to the difference between their energy and the quasi-Fermi level. Under dynamic conditions, the trap occupancy and trapping time can change by several orders due to bias conditions, oxide quality and temperature. The charge statistically or dynamically accumulated in the traps perturbs the built in potential, and can have a strong impact on the Si-SiO2 interface and of SiO2 insulating film. The stretched bonds and bonds formed by contaminants and hydrogen with the underlying crystalline silicon can be much weaker than well oriented Si-O bonds. Therefore they are easily broken in the interaction with radiation and hot carriers, generating interface traps that degrade the electronic properties of the surface and impact the electrical performance of devices such as Threshold Voltage, transconductance, current etc.

If the trap is of acceptor type within the oxide, it forms a sink for hot electrons to get trapped after a program due to electron tunneling (when the channel gets activated). Hence, the Vt of the cell definitely reduces as the net number of electrons on the floating gate would reduce. Again, if such trapping and releasing (leakage) of these hot electrons continues, a sufficient number of electrons can be removed from the floating gate causing the Vt of a cell to breach the individual MLC state window budget. This would in turn as described in the previous chapter cause the sense amplifiers to read a different value in the cell compared to its prior successful program.
If the trap is of donor type within the tunnel oxide, it forms a potential source of electrons which could be attracted from the control gate and get injected onto the floating gate. If this happens, the Vt of the cell definitely changes as the net number of electrons on the floating gate would increase. If such injection of electrons onto the floating gates continues, the Vt of the cell may breach the individual MLC state window budget and the cell would now store a different state. This would be decoded as a different value as the one originally programmed in the cell.

Hence, these traps can cause errors in the contents of the flash cells based on the bias conditions and the location of the traps. The above discussion can be summarized in Table 4.6.

<table>
<thead>
<tr>
<th>No.</th>
<th>Initial State</th>
<th>Trap Type</th>
<th>Trap Location</th>
<th>Conditions</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>Donor</td>
<td>Tunnel Oxide</td>
<td>WL Activation</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Donor</td>
<td>Tunnel Oxide</td>
<td>WL Activation</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>Donor</td>
<td>Tunnel Oxide</td>
<td>WL Activation</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>Donor</td>
<td>Interpoly / Tunnel Oxide</td>
<td></td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
<td>Acceptor</td>
<td>Interpoly / Tunnel Oxide</td>
<td>Vread on Drain</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>Acceptor</td>
<td>Interpoly / Tunnel Oxide</td>
<td>Vread on Drain</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Acceptor</td>
<td>Interpoly / Tunnel Oxide</td>
<td>Vread on Drain</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>Acceptor</td>
<td>Interpoly / Tunnel Oxide</td>
<td></td>
<td>over-erased</td>
</tr>
</tbody>
</table>

### 4.4.3 Contamination Faults

When the flash chips are fabricated, they can never be 100% defect-free. Due to process variations, introduction of new method, upgrading of the current process technology or even inconsistency in the resist (integral part of any VLSI fabrication procedure), the resulting wafers will always have a yield short of 100%. Usually 97% is an acceptable number for the process technology to be deemed successful. From the 97% (yield per wafer) of die that are good wafers, there can
be instances or localized positions within each die. There is a high probability that due to some intermediate fabrication process, a small amount of foreign agents or resist could be trapped. These unwanted foreign materials could have adverse effects on the reliability of the flash cell. This is highly unwanted in MLC flash. The contaminants, as in the case of traps (Section 5.2), can be acceptor or donor type impurities. There have been references in literature which state that mobile sodium ions (Na+) are present in the furnace while realizing the Si-SiO2 interface. During the formation of the gate oxide and the outer oxide, there could be an introduction of these mobile Na+ ions into the floating gate region. The concentration of sodium increases after the photo resist deposition. This causes undesirable effects and they can be classified into two types of contamination faults: contamination charge loss and contamination charge gain.

**Contamination Charge Loss**

The mobile sodium ions (Na+), once introduced with the oxide encapsulation of the flash cell, are attracted by highly negatively charged floating gate. (The built-in bias makes the flash cell even more vulnerable to such effects [15, 16].) Hence these positively charged nuclei are neutralized by the electrons in the floating gate. As a result, the potential of the floating gate reduces. In MLC flash, if the Vt distribution is very tight or if the read window budget is very small, then the individual gaps between adjacent states of the cell will be too close to tolerate such charge loss. This may cause the Vt of certain tail bits to reduce further and breach the gap between adjacent levels. So, the state will change and the flash cell will be decoded to store a different value from the one originally programmed.
**Contamination Charge Gain**

If a cell which is affected by the above fault is erased and then programmed again, then the ion would no longer be bound to the floating gate and would diffuse away due to concentration gradient or other factors. Hence, when these ions diffuse away, they leave the extra electrons back on the floating gate thus increasing the electron count (after a program has already been done). This would result in an increase in $V_t$. Again, for MLC arrays, this could mean breaching the $V_t$ state gap if the bits are tail bits at the end of a state distribution. This could cause the state of the cell to shift to an adjacent one and the cell records a new value when a read is performed.

The effects due to contamination charge defects have been summarized in Table 4.7.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Initial State</th>
<th>Contamination Type</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>Charge Loss</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>Charge Loss</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>Charge Loss</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>Charge Gain</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>Charge Gain</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>Charge Gain</td>
<td>00</td>
</tr>
</tbody>
</table>

### 4.4.4 Leakage Faults

Faults may occur in the flash array in cells which contain another alternate path to ground, besides their source connection to the ground via the source straps [14, 18, 19]. There can be two paths along which there is a leakage path to ground. This could be either due to leakage along a column (*column leakage*) or due to leakage along a row (*row leakage*) [14].
Column / Bit Line Leakage Faults

Consider all the cells connected to a bit line. Sometimes during the erasing of a cell, it will be noticed that not all cells erase at the same rate. This is due to slight differences (source resistance, L/Zeff variation, doping etc). This variation in erase times can cause serious problems. This causes certain cells to be over-erased [14]. (Such cells need to undergo OER - Over-Erase Repair.)

These over-erased cells cause an unwanted path to ground, and can cause leakage of the floating gate electrons back to the source / ground changing the state of the cell in an MLC array [17] as shown in Fig. 4.10. Hence, it would be desirable to find such leaky columns and perform what is called PER (post erase repair) to bring the Vt of the over-erased cells back to their erased level.

Consider Fig. 4.10 where cell C is over erased, has the Vt value is less than zero ($V_t < 0$). Hence, normal read bias voltage would sink all the bit line current to ground, as the transistor is on and there is a direct path to ground. Now, if a read is performed on a normal cell B, all the current in the bit line will flow to ground. The voltage at the input pin of the sense amplifier will be less than the reference voltage, and the logic value recorded will be a logic 1 (SBC) or logic 11 (MLC). Consequently, the sense amplifier considers all the cells on the selected column to be erased, and such a set up is a called "erase leaky column". This problem can cause serious problems as whatever be the logic state of the cell namely 11, 10, 01 or 00, if there is an over-erased cell in the array, the data in the correctly programmed cells will not be read properly. In other words, there will be read errors as the cells will always read 11. During fault analysis, this can be also
observed as a multiple stuck-at fault, where all the cells along the bit line or column appear to be erased or stuck-at 11. If a cell is programmed to any of the three MLC voltage levels of 10, 01 or 00 and then performed a read, the cell will yield a 11.

In order to detect this, usually all the word lines are either grounded or kept at a very low voltage. The deselected bit lines are also grounded / kept floating. Only the selected bit lines is kept at the required read voltage. This kind of fault will not only cause a read fault as discussed above, but also a program fault due the timing-out of the program-verify algorithm which results
in raising a fail flag. These faults are summarized in Table 4.8.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Correct Programmed Value</th>
<th>Fault Type</th>
<th>Value in cell &quot;read&quot; out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>Leakage Along Column</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>Leakage Along Column</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>Leakage Along Column</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>Leakage Along Column</td>
<td>11</td>
</tr>
</tbody>
</table>

**Row Leakage Faults**

In order to program a particular cell "A" in the array (Fig. 4.11), only the corresponding bit line and word line must be activated. All other bit and word lines must be grounded/de-activated. In such a situation, if there is a short between the selected/activated word line and a neighboring grounded word line, then there will be a leakage path for the control gate current. The magnitude of the leakage current will depend on the resistance of the bridging path (short circuit). The leakage current flowing can be measured using testing probes. If this value is high then it can be guaranteed that the proper voltage will not be supplied to the gate of cell "A" (cell to be programmed). For example, if cell A is to be programmed to "state 00" (Vg is 12v), it will not get the appropriate control gate voltage due to the above reason. Thus, depending on the resistance of the leakage path and the Vt window between state 00 and state 01, cell A will be programmed to a lower level Vt state of 01.

<table>
<thead>
<tr>
<th>No.</th>
<th>Intended Programmed Value</th>
<th>Initial Value</th>
<th>Fault Type</th>
<th>Faulty Programmed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>11</td>
<td>Row Leakage</td>
<td>01 or 10</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>11</td>
<td>Row Leakage</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>11</td>
<td>Row Leakage</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>11</td>
<td>Row Leakage</td>
<td>11</td>
</tr>
</tbody>
</table>
Figure 4.11: Faults due to Leakage along a row.

**Gate-Drain Coupling Faults**

During programming the drain voltage (Vd) is usually high (5-7 V). Sometimes, when a cell gets over erased, the Vt of the cell will be very low. Hence, due to the reduction in threshold voltage, the gate couples to the drain, causing a leakage current (the cell will be ON due to low Vt). This leakage current will cause a reduction in the net voltage at the drain terminal, i.e., the drain voltage is pulled down. If this leakage current is high enough it can slow or even prevent programming. A high leakage current is caused by many flash cells at a very low Vt. Hence tail bits at the end of the Vt distribution can cause such a situation.

To detect this all the word lines are grounded. All the array sources are grounded. Only one single column is selected. The current through the cell is measured and the state of the cell
can be determined. If a cell has a very low Vt, the post erase repair or PER needs to be performed to bring up the Vt to the desired level so that programming may be performed. PER is a kind of Soft programming. Due to this Low Vt and high leakage current, soft programming cannot be achieved and the cell will be stuck-at 11. The above may be summarized in Table 4.10.

<table>
<thead>
<tr>
<th>No.</th>
<th>Initial Value</th>
<th>Intended Programmed Value</th>
<th>Fault Type</th>
<th>Final Faulty Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>GD-Coupling</td>
<td>Over-Erased</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>10</td>
<td>GD-Coupling</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>01</td>
<td>GD-Coupling</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>00</td>
<td>GD-Coupling</td>
<td>11</td>
</tr>
</tbody>
</table>

**Stress Induced Leakage Current (SILC) Faults**

As technology progresses, attempts are usually made to reduce the thickness of the tunnel oxide due to scaling. This can cause serious problems with data retention and endurance. During
testing when the part is cycled, with every ramp a high electric field from the gate and drain could
damage the thin tunnel oxide which by today’s technology would be just a few layers or molecules.
This being extremely thin has very little capacity to withstand high electric fields, and as a re-
sult would develop leakage current paths which increase with each successive voltage ramp of the
program-erase cycling process. This leakage current is called Stress Induced Leakage Current or
SILC [14]. The fault model proposed in literature attributes the SILC to tunneling of electrons
through weak spots related to defects that reduce the tunneling barrier. Hence, the electrons from
the floating gate can escape into the thin tunnel oxide at a much lower voltage than the erase
voltage needed to attract electrons into the source. The Vt of the cell would change. If the state
window budget is breached, then the cell would read an incorrect value. A realistic model would
be to shift to the next lower adjacent state in the MLC Vt distribution diagram. This can be
summarized in Table 4.11.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Original State</th>
<th>Fault Type</th>
<th>Faulty State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>SILC</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>SILC</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>SILC</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>SILC</td>
<td>Over-Erased</td>
</tr>
</tbody>
</table>

Most MLC products usually have extremely thin gate oxides and are subjected to fine tuned
voltages due to accurate charge placement (discussed in chapter 2). Hence this fault model is highly
important specifically in MLC flash array testing as the testing itself would cause these faults.
Chapter 5

The MLC Fault Test Methodology

Chapters 2, 3 and 4 have laid the foundation for multi-level flash memory testing. Chapter 2 dealt with an overview of SBC and MLC flash memories. Chapter 3 discussed the multi-level flash array memory architecture which will be used as a platform to apply test patterns. Finally, Chapter 4 elaborately explained the various faults that might occur in the flash array.

The MLC fault model was formulated by considering the operating conditions activating each fault. After acknowledging the behavior of each fault, the test patterns that detect these faults must now be considered. This chapter begins with a brief review of the inability of conventional march algorithms to test flash memories, and then determines the test patterns / flash march algorithms to detect each type of MLC flash memory faults. The main aim of any test algorithm is to test a maximum number of faults with a minimal number of test patterns or algorithm steps. This will in turn reduce the amount of tester time, thus keeping the cost of fault testing to minimum. This is the driving force behind merging the test patterns / algorithms of various types of faults into one effective test algorithm. This will be dealt with in the final section of test pattern merging.

Note: This chapter only deals with the patterns for flash testing. No emphasis is made to the type
of faults that occur in the MLC array. The patterns for fault diagnosis will be dealt with in the next chapter.

5.1 Traditional March Algorithms

Conventional SRAMs can be efficiently tested using march algorithms. March algorithms, as the name suggests, involve performing a series of write and read operations to a memory array in an increasing or decreasing order of addresses. SRAM-based memory arrays are tested by storing logic 0 or logic 1 within each SRAM cell. The entire array is cycled in an appropriate manner, so as to activate every fault under consideration in the fault model. In other words, the fault model determines the length and complexity of the testing algorithm. A typical march algorithm looks like:

\[
\uparrow \{w1, r1\}; \downarrow \{w0, r0\}.\]

The above march elements signify initially writing a logic 1 and reading a logic 1 to the SRAM cells in an increasing order of addresses, i.e., from address "0" to "N-1". A similar process is repeated by writing and reading logic 0 to all the SRAM cells but in a decreasing order of addresses. Different march algorithms perform different read and write operations in an order suitable to test for various faults.

5.2 Flash-March Algorithms

Flash memory testing is not straight-forward like most semi-conductor memory technologies. The flash architecture and the basic operations determine the testing methodology. Conventional march algorithms cannot be applied to a flash array directly. The reason is that any march algo-
algorithm writes 0 and 1 alternatively or in some pattern to the array under test. However, logic 1 cannot be written or programmed into a flash cell individually. The only way to change the contents of a flash cell to logic 1 is to erase the cell. This is where the problem lies. Flash memories erase entire blocks at a time, implying that a W1 to an individual cell cannot be directly applied. This is a major limitation of the application of traditional march algorithms in flash memory testing. Hence, the algorithm must be first modified to test a flash array effectively. There are instances in literature where conventional algorithms such as the MATS++ have been modified and applied to test certain faults \[6\]. In this case, the dual of the algorithm is formed as stored information is inverted in flash memories, i.e., logic 1 for an erased and logic 0 for a programmed one. This is another reason why generic tests which can be applied to test a non-flash array cannot be directly applied.

There are very few instances in literature of testing flash memories with march-like algorithms \[7\]. This thesis is an attempt to formulate an effective fault model and apply flash march-like algorithms to test the array. Moreover, multi-level flash memories have never been tested using a march-algorithmic testing methodology. Thus, this thesis merges the effectiveness and simplicity of march-like algorithms to test a complex semi-conductor memory technology such as MLC flash.

5.3 Test Pattern Generation

Flash faults cannot be broadly classified on the basis of SBC or MLC faults, as there are certain faults which are common to both types of flash memories. Since the architectures of SBC and MLC architectures are the same, the faults that occur in SBC architectures can very easily be extended or expected to occur in MLC arrays as well. In fact, a fault which does not cause a change
of state in SBC can possibly cause the state of the MLC to change. In the recapitulation from chapter 4, any type of flash memory (SBC / MLC) fault may be broadly classified into two main categories: traditional and non-traditional faults and have been listed in Table 5.1 for convenience.

This chapter first derives the elementary test patterns for each kind of fault. A diagonal march algorithm is then developed which will be implemented to reduce the test patterns for program disturb faults. The elementary test patterns are then merged into the diagonal algorithm. This ends up with a set of test patterns that are very compact and can detect all of the faults listed in Table 5.1.

<table>
<thead>
<tr>
<th>Fault Category</th>
<th>S.No.</th>
<th>Type of Fault</th>
<th>Sub-Faults</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>1</td>
<td>Stuck-at Faults</td>
<td>Stuck-at 00</td>
<td>MLC Only</td>
</tr>
<tr>
<td>&quot;</td>
<td>2</td>
<td>&quot;</td>
<td>Stuck-at 01</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>3</td>
<td>&quot;</td>
<td>Stuck-at 10</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>4</td>
<td>&quot;</td>
<td>Stuck-at 11</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>5</td>
<td>Coupling Faults / Disturbs</td>
<td>——</td>
<td>SBC / MLC</td>
</tr>
<tr>
<td>&quot;</td>
<td>6</td>
<td>Bridging Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>7</td>
<td>Broken Connection Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td>Non-traditional</td>
<td>8</td>
<td>Series Resistance Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>9</td>
<td>Faults due to Charge Leakage</td>
<td>Along a Column</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>10</td>
<td>&quot;</td>
<td>Along a Row</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>11</td>
<td>&quot;</td>
<td>Gate-Drain coupling</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>12</td>
<td>Contamination Faults</td>
<td>Charge Loss</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>13</td>
<td>&quot;</td>
<td>Charge Gain</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>14</td>
<td>Faults due to Oxide Traps</td>
<td>——</td>
<td>&quot;</td>
</tr>
<tr>
<td>&quot;</td>
<td>15</td>
<td>SILC Faults</td>
<td>——</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

5.4 Traditional Fault testing for Multi-Level Flash Memories

5.4.1 Stuck-at Fault Testing

In order to detect all stuck-at faults, the entire array must be cycled through all the programming and the erase states. There will be a ”read” step between each of them. This checks
whether each flash cell has been configured to the intended state or not. By performing a read operation, we can determine if the cell is currently at its intended program value or stuck-at the previous value. This section will deal with the algorithmic steps which detect all stuck-at faults in the flash array.

**SAF Testing Algorithm**

- Initially, the entire array is erased by \( \uparrow (w11) \). Erase is performed on a block basis so for a particular 0.5M block under consideration, all the flash cells within it are set to state 11. All addresses are erased from "0" to "N-1". This is the default state and every algorithm will begin with this step.

- The entire array is now programmed to state "10".

- A read operation will be performed to ensure if the program was successful or not. Traditional march algorithms usually perform a read of the expected value stored in the memory cell. So all cells are read to check for state "10" by \( \uparrow (r10) \).

- If any of the cells does not read 10 and still reads 11, then such cells can be said to be stuck-at 11. Hence, by performing a read operation for the entire block, all stuck-at 11 faults can be detected.

- The entire array is now programmed to state "01" by \( \uparrow (w01) \). Hence, all cells must now contain "01".

- All cells are now read to ensure correct programming to state "01" by \( \uparrow (r01) \).

- If any of the cells do not read 01 and still read the previously programmed value 10, then such cells are said to be stuck-at 10. Hence, by performing a read operation for the entire
block, all **stuck-at 10** faults can be detected.

- The entire array is programmed to state "00" by $\uparrow (w00)$. Hence, all cells must now contain "00".

- All cells are now read to ensure correct programming to state "00" by $\uparrow (r00)$.

- If any of the cells do not read 00 and still read the previously programmed value 01, then such cells are staid to be stuck-at 01. Hence, by performing a read operation for the entire block, all **stuck-at 01** faults can be detected.

- The entire block is erased. Hence, all cells must now contain "11" by $\uparrow (w11)$.

- All cells are now read to ensure successful erase to state "11" by $\uparrow (r11)$.

- If any of the cells do not read 11 and instead detect the previous state 00, then such cells are staid to be stuck-at 00. Hence, by performing a read operation for the entire block, all **stuck-at 00** faults can be detected.

Thus, from the above algorithm, all stuck-at faults from the MLC fault model are detected.

**FMA - SAF:** (Flash March Algorithm - Stack At Faults)

$\uparrow (w11); \uparrow (w10); \uparrow (r10); \uparrow (w01); \uparrow (r01); \uparrow (w00); \uparrow (r00); \uparrow (w11); \uparrow (r11)$

This may be further regrouped as:

$\uparrow (w11); \uparrow \{(w10), (r10)\}; \uparrow \{(w01), (r01)\}; \uparrow \{(w00), (r00)\}; \uparrow (w11); \uparrow (r11)$

This would perform all the above operations cell by cell rather than the entire array at a time. Our final goal is to integrate all individual march algorithms into a single one, it will prove...
to be efficient to split up a large algorithm step into sub-steps so that they may be integrated with sub-steps from other fault algorithms.

5.4.2 Bridging Fault Testing

Based on our MLC flash architecture discussed in Chapter 3, there can be two types of bridging Faults: word line bridging faults and bit line bridging faults. The march algorithm steps for both types of faults are the same. However, the marching address increment step are different. It is because of this address increment that we require three separate march algorithms to test for these kinds of faults. However, they can be merged into a single algorithm eventually.

While a flash cell is programmed, a bridging fault may cause another cell to be programmed on another word or bit line depending on the location of the fault. This will cause an unintentional or a spurious program of another flash cell and the march-algorithm will need to detect this.

BF Testing Algorithm

- Initially the array is erased. All cells within the block must now be in state "11" by ↑ (w11).

- All cells within the flash array are read and subsequently programmed to state "10" by ↑ (r11, w10). This step can also be used to check for the correct erase state, i.e., over-erase faults can be detected. The address is incremented by the appropriate step before each of the above operations can be applied.

- Assume there is a bridging fault between two bit lines (or word lines) L₁ and L₂. When the pattern ↑ (r11, w10) is applied to one cell C₁ (along L₁), then the same programming voltage is also given to cell C₂ (along L₂). Thus when cell C₂ is read by ↑ (r11, w10), instead of
reading a logic value of 11, state 10 will be read. Thus, the bridging fault between L1 and L2 is detected.

- The above procedure is repeated with march elements $\uparrow (r10, w01)$, $\uparrow (r01, w00)$ and $\uparrow (r00)$ shown below. Each of the bridging faults will be repeatedly detected two more times.

- Thus all bridging faults can be detected with the first two march elements of FMA-BF. The march elements after $\uparrow \{(r11), (w10)\}$ are given here only for the purpose of finding a universal (golden) march algorithm.

Thus, in order to detect all bridging faults, the Flash March algorithm applied is:

\[
\text{FMA - BF: (Flash March Algorithm - Bridging Faults)}
\]

\[
\uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow (r00).
\]

Bridging Faults - Local Bit Lines

Two local bit lines are fabricated adjacent to each other. While marching is preformed say in $\uparrow$ direction, if the address is increased by "1", it will result in accessing the next adjacent flash cell (of the next word). This means the next adjacent bit line will be activated in order to program or read a cell. Hence, if there is a short between two such local bit lines, the march algorithm proceeds without any problem in regular increments of 1 from address "0" to address "N-1". Thus, all bridging faults will be detected by implementing the FMA-BF algorithm.

From Chapter 3, we have

Start Address: XXX - XXX - XXXXXXXXXX - XXX - L0 L1 L2, and
Next Address: XXX - XXX - XXXXXXXXX - XXX - L0 L1 (L2+1).

Hence, the march algorithm will be:

**FMA-BF-LBL:** *(Flash March Algorithm-Bridging Fault-Local Bit Line)*

\[
\uparrow (w_{11}); \uparrow \{ (r_{11}), (w_{10}) \}; \uparrow \{ (r_{10}), (w_{01}) \}; \uparrow \{ (r_{01}), (w_{00}) \}; \uparrow (r_{00}); \quad (\Delta n = 1)
\]

\((\Delta n = \text{Address Increment Step})\)

**Bridging Faults - Global Bit Lines**

From Chapter 3, there is one global bit line for every 8 local bit lines. In each sub-block of 64 local columns, there are thus 8 such global columns. The addressing scheme between two adjacent global bit lines is shown below.

Address with global bit lines G0 G1 G2: XXX - XXX - XXXXXXXXX - G0 G1 G2 - XXX

Address with global bit lines G0 G1 (G2+1): XXX - XXX - XXXXXXXXX - G0 G1 (G2+1) - XXX

Consider a bridging fault between global bit line 0 (GBL 0) and global big line 1 (GBL 1). Let the local bit line bits decoded by the address be 7. Then, the cell at (GBL 0, LBL 7) and (GBL 1, LBL 7) will be programmed to the same value. Hence, the march algorithm will have to be modified by decoding the global bit line address and incrementing the global bit line by 1. This would translate to incrementing the address by 8. Thus, the above algorithm will need to have a global bit line increment logic block to determine bridging faults between two global bit lines.

Hence, the march algorithm is:
**FMA-BF-GBL:** *(Flash March Algorithm-Bridging Fault-Global Bit Line)*

\[ \uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow (r00); \quad (\Delta n = 8) \]

Since the error is between global bit lines spaced, eight local bit lines apart, the fault detection analysis will be done once every 8 address increments. It should be noted that FMA-BF-LBL will also detect these kinds of faults.

**Bridging Faults - Word Lines**

From Chapter 3, there are 1024 local columns per block. Since this is split up into 16 groups of 64 local bit lines, there can be a maximum of only 64 words per row. This implies that if word 65 needs to be accessed, then the row address must be incremented. Consider a bridging fault between word line 0 and word line 1. If the flash cell at (WL 0, BL 0) (flash cell in word 1) is required to be programmed, then the cell along (WL 1, BL 0) (flash cell in word 65) will also be programmed due to the bridge. Hence, in order for the word line bridge to be detected, the march algorithm address step must be increased to 64. In other words, the row bits must be incremented by 1.

Thus, the March Algorithm will be:

**FMA-BF-WL:** *(Flash March Algorithm-Bridging Fault-Word Line)*

\[ \uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow (r00); \quad (\Delta n = 64) \]

Again, it should be emphasized that the FMA-BF-LBL algorithms will also detect this kind of faults.
5.4.3 Broken Connection Faults

As discussed in Chapter 4, there are three regions where a broken connection fault may occur:

- Broken word line connection (disconnected from gate) cases 1 and 2.
- Broken bit line connection (disconnected from drain) cases 1 and 2.
- Broken source-ground connection (disconnected from source)

Each type of fault will now be considered and a flash march algorithm will be proposed to test for such kind of faults:

Broken Word Line and Bit Line - Case 1

The broken connections along the bit and word lines of type case 1 are very similar in behavior. Since the vital connections such as the bit and word lines are disconnected from the cell, the cell is rendered completely useless. In other words, no erase, program or read operations may be performed on such cells. The value stored in it will be undefined or may be either stuck-at a particular value due to various operating conditions or faults in its proximity. We shall consider cells which possess such type of faults to have an "undefined" value.

Clearly, the only way we can be sure that no operation has an effect on such cells is to program such cells, through all the programming states and perform a block erase. Therefore, the algorithm for bridging faults can be used for this case as well. Thus, the march algorithm will be:

FMA-BCF-WL1-BL1:
(Flash March Algorithm-Broken Connection Fault-Word Line Case 1 - Bit Line Case 1)

⇑ (w11); ⇑ {(r11), (w10)}; ⇑ {(r10), (w01)}; ⇑ {(r01), (w00)}; ⇑ (r00);

Broken Word Line - Case 2

This type of fault may render all cells along a word line from the fault location until the end of the row, completely inaccessible. If there is a broken connection between cells along word line "L" between column "n" and column "n+1", then all cells from column "n+1" until the last column on word line L will be rendered useless. The same march algorithm for the previous case can be applied again.

FMA-BCF-WL2: (Flash March Algorithm-Broken Connection Fault-Word Line Case 2)

⇑ (w11); ⇑ {(r11), (w10)}; ⇑ {(r10), (w01)}; ⇑ {(r01), (w00)}; ⇑ (r00);

Broken Bit Line - Case 2

This type of fault may render all cells along a bit line from the fault location until the end of the column, completely inaccessible. If there is a broken connection between cells along bit line "L", between row "m" and row "m+1", then all cells from row "m+1" until row 512 along bit line L will be rendered useless. Hence, the march algorithm which is applied, must analyze one every 64 words at a time in order to notice where the fault occurred along a column/bit line. The same march algorithm for the previous case can be applied again.

FMA-BCF-WL2: (Flash March Algorithm-Broken Connection Fault-Bit Line Case 2)

⇑ (w11); ⇑ {(r11), (w10)}; ⇑ {(r10), (w01)}; ⇑ {(r01), (w00)}; ⇑ (r00);
**Broken Connection Fault between Source-Ground**

Chapter 3 dealt with the flash architecture where the source-ground connection of each flash cell was explained. There is one global source-strap line every eight bit Lines where the sources of all flash cells are connected to one global source strap which is grounded.

From Chapter 4, Section 4.4, a broken connection fault between source and ground can render a large part of the array useless. The cells affected by this cannot be erased, since there will be no means to draw out the electrons from the floating gate into the channel. If there is a broken connection along the source strap, then all the eight columns of cells connected to it will be disconnected from ground, and will lead to a memory loss of $512 \times 8 = 4096 = 4K$ bits. This broken connection has the most impact, as a large portion of the array can be rendered unusable. Hence, only a block erase and a read operation should be able to detect such type of faults.

**FMA-BCF-S-G: (Flash March Algorithm-Broken Connection Fault-Source - Ground)**

$\uparrow (w11); \uparrow \{(r11)\}$

### 5.4.4 Disturbance Fault Testing

The cells in the flash array are subject to high voltages both on the gates and the drains. Hence, they are very prone to errors due to "program disturbances". Program disturbances affect cells sharing either the same word line or the same bit line as the intended cell to be programmed. During a program operation, all the cells along a particular word line are subjected to gate stress. In recapitulation, the two types of programming disturbs due to gate stress are DC-erasing and
Program Disturbance: Gate Stress - DC ERASE (DC-E)

Program disturbances due to DC-erasing were explained in detail in Chapter 4, Section 4.4.2. Now the test patterns which will detect such types of disturbances will be considered.

Program Disturbs: DC Erase Detection Algorithm

- Initially, the entire array is erased by $\uparrow (w11)$;

- DC-Erase faults can only be activated if the array is programmed, so the next step would be to program it to the next higher Vt level. While programming a cell $C_1$ to "state 10", another cell ($C_2$) which is currently erased can be over erased, and thus, we need to read $C_2$'s state before programming it by $w_{10}$.

- If the r(11) operation to $C_2$ is false, then a DC-Erase fault of type 4 at $C_2$ is detected. Note that our sense circuit must support the detection of the over-erased state.

- All cells are read to check if the program operation to the previous cell resulted in a "soft erase". All cells are then programmed to "state 01" by $w_{01}$.

- If the r10 operation to the cell is false, then a DC-Erase fault of type 3 is detected.

- This process is repeated until all the types of DC-Erase faults are detected.

Thus, the March Algorithm will be:

**FMA-GS/PD - DC-E:**

(Flash March Algorithm-Gate Stress / Program Disturb - DC Erase:)

$\uparrow (w11); \uparrow \{r(11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow \{(r00)\}$;
Program Disturb: Gate Stress - DC-PROGRAM (DC-P)

Program disturbs due to DC Programming were dealt with in Chapter 4, Section 4.4.3. Now, the test patterns which will detect such types of 'disturbances' will be considered. These types of faults occur during programming of cells.

Program Disturbs: DC Program Detection Algorithm

- Type 1 DC-Program faults can only be activated if the array is erased. Hence, the entire array is initially erased. \( \uparrow (w11); \)

- Each cell must be programmed to state "10". While programming a cell \( C_1 \) to "state 10", another cell \( C_2 \) which is currently erased could be programmed to the same, and hence we need to read the state of \( C_2 \) before programming it by \( \uparrow r11, w10; \).

- If the \( r11 \) operation to cell \( C_2 \) is false then the DC-Program fault of type 1 is detected.

- Each cell is then programmed to "state 01". Before this is done each cell is read to check if the program to the previous cell resulted in programming the current one by \( \uparrow r10, w01; \).

- If the \( r10 \) operation to the cell is false, then the DC-Program fault of type 2 is detected.

- The process is repeated to check "01" state. We emphasize that there is no over-programmed state for "00". So, \( w00 \) and then \( r00 \) will not detect any program disturb fault.

Thus, the March Algorithm will be:

**FMA-GS/PD - DC-P:**

\( (Flash \ March \ Algorithm-Gate \ Stress / \ Program \ Disturb - \ DC \ Program:) \)

\( \uparrow (w11); \uparrow \{r(11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow \{(r00)\}; \)
Program Disturbs: Drain Stress/Disturb (DD)

Drain disturbances due to programming were dealt with in Chapter 4, Section 3.4.4. Now, the test patterns which will detect such types of 'disturbs' will be considered. Test patterns for two types of drain disturbances will be determined. They are bit line erase disturb (BED) and bit line program disturb (BPD).

Drain Stress / Drain Disturbance Testing Algorithms

As described in the fault model chapter, a program operation to a cell causes another cell on the same bit line to be erased (BED) or programmed (BPD) and the same algorithm as DC-Erase and DC-Program can be implemented to test drain disturbs in the flash array. Thus, the drain disturbance algorithm is:

FMA-DD

(Flash March Algorithm - Drain Disturb)

\[ \uparrow (w11); \uparrow \{r(11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\}; \uparrow \{(r00)\}; \]

In order to detect a drain disturb, after a cell is programmed, we need to perform a read operation on the next row along the same bit line to check whether the fault occurred or not. Since there are 64 words present per word line, we need to increment the row address by 1 in order to jump to the next row to perform a read. However, the universal test patterns will still detect this type of faults, since the march algorithm will eventually move to the next word to read the fault effect.
5.5 Non-traditional Faults in Multi-Level Flash Memories

5.5.1 Series Resistance Faults

The series resistance fault behavior was dealt with in Chapter 4, Section 5.1. The fault behavior is independent of the initial state of the flash cell. Hence, we do not need to test for all cases depicted in the fault model. The reason is the change in $V_t$ due to the series resistance ($\Delta V$) will always be constant regardless of the initial state. This is due to the fact that the series resistance is constant for a given cell. Hence, the final state of the flash cell would have an error corresponding to $\Delta V$.

- Initially, the entire array is erased by $\uparrow(w11)$.

- A read operation is performed to check for proper erasure (r11), and each cell is now programmed to state "10" by $\uparrow r11, w10$.

- Each cell is then checked for correct programming by (r10). If the read operation does not yield state 10, then a fault is detected. Each cell is then programmed to state "01" by $\uparrow r10, w01$.

- Each cell is then checked for correct programming by (r01). If the read operation does not yield state 01, then a fault is detected. Each cell is finally programmed to state "00" by $\uparrow r01, w00$.

- Each cell is then checked for correct programming by $\uparrow \{(r00)\}$. If the read operation does not yield state 00, then a fault is detected.

Thus, the March Algorithm can be summarized as under:

FMA-SRF-Testing
5.5.2 Oxide Trap Faults

The faults that arise due to traps in the oxide were discussed in Chapter 4, Section 5.2. Considering the fault behavior table for both acceptor and donor type oxide traps, the fault detection march algorithms will now be developed.

Algorithm to Detect Donor Type Oxide Traps

- Initially, the entire array is erased by \( \uparrow (w11) \).
- A read operation is performed to check for proper erasure by \( (r11) \), and each cell is now programmed to state ”10” by \( \uparrow r11, w10 \).
- Each cell is then checked for correct programming by \( (r10) \). If the read does not yield state 10, then a fault is detected. Each cell is then programmed to state ”01” by \( \uparrow r10, w01 \).
- Each cell is then checked for correct programming by \( \uparrow r01 \). If the read does not yield state 01, then a fault is detected.

Hence, the March Algorithm can be summarized as under:

**FMA-DOTF-Testing**

\( \uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01)\}; \)

Algorithm to Detect Acceptor Type Oxide Traps

- Initially, the entire Array is erased \( \uparrow (w11) \).
A read operation is performed to check for proper erasure by \( r_{11} \), and each cell is now programmed to state "10" by \( \uparrow r_{11}, w_{10} \). If the \( r_{11} \) operation is false, the over-erase fault is detected.

Each cell is then checked for correct programming by \( r_{10} \). If the read does not yield state 10, then a fault is detected. Each cell is then programmed to state "01" by \( \uparrow r_{10}, w_{01} \).

Each cell is then checked for correct programming by \( r_{01} \). If the read does not yield state 01, then a fault is detected. Each cell is finally programmed to state "00" by \( \uparrow r_{01}, w_{00} \).

Each cell is then checked for correct programming by \( \uparrow \{ (r_{00}) \} \). If the read does not yield state 00, then a fault is detected.

Thus, the March Algorithm can be summarized as under:

**FMA-AOTF-Testing**

*(Flash March Algorithm - Acceptor Oxide Trap Fault Testing:)*

\( \uparrow (w_{11}); \uparrow \{ (r_{11}), (w_{10}) \}; \uparrow \{ (r_{10}), (w_{01}) \}; \uparrow \{ (r_{01}, (w_{00})) \}; \uparrow \{ (r_{00}) \}; *

### 5.5.3 MLC Contamination Fault Testing

**Contamination Charge Loss & Charge Gain Testing Algorithm**

- Initially, the entire array is erased by \( \uparrow (w_{11}); \).

- If there are contaminants in the oxide which cause electrons to get off the floating gate, then the state would be different from state 11. Hence, a read operation \( r_{11} \) is performed to check for proper erasure, and each cell is then programmed to state "10" by \( \uparrow r_{11}, w_{10} \).

- Each cell is then checked for correct programming by \( r_{10} \). If the read operation does not yield state 10, then a fault is detected. Each cell is then programmed to state "01" by
An unsuccessful read, i.e., state 11 or 01, would imply a charge loss or gain type contamination fault respectively.

- Each cell is then checked for correct programming by \((r01)\). Each cell is finally programmed to state "00" by \(\uparrow r01, w00\). An unsuccessful read, i.e., state 10 or 00, would imply a charge loss or gain type contamination fault respectively.

- Each cell is checked for correct programming by \((r00)\). An unsuccessful read \((r00)\), i.e., state 01, would imply a charge loss contamination loss. Note that there is no charge gain fault for state "00".

Thus, the March Algorithm can be summarized as under:

**FMA-CL/CG-CF-Testing**

*(Flash March Algorithm - Acceptor Oxide Trap Fault Testing:)*

\[ \uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01, (w00))\}; \uparrow \{(r00)\}; \]

### 5.5.4 Leakage Charge Fault Testing

The faults that occur due to charge leakage were dealt with in Chapter 4, Section 4.4. Faults due to leakage occur in the flash array in cells which contain an alternate path to ground, besides their source connection to the ground (via the source straps). There can be two paths along which charge can leak to ground. This could be either due to leakage along a column or due to leakage along a row.

**Column Leakage**

Chapter 4, Section 4.4.1 dealt with the mechanisms which lead to leakage faults along a column. This section will deal with the march algorithms which will test a flash array for leakage
along a column.

The starting condition for the algorithm is that only the bit lines connected to the cell intended to be programmed are activated. All other bit lines are de-activated in order to notice an alternate leakage path. From the fault behavior of leakage along a column, no matter what state the cell is programmed to, the final state will always be state 11 as all the charge on the floating gate will leak off. Hence, we can use any of the non-”11” states to test this fault. We therefore just use a part of the universal pattern.

- Initially the array is erased by $\uparrow (w11)$;

- If there exists an alternate path to ground along a column, electrons would flow off the floating gate causing the Vt of the cell to drop from the erased state to the over-erased state. Therefore, the Vt would be lower than the erased state. Hence, a read operation is performed and each cell is programmed to ”state 10”. This step is done by $\uparrow (r11), (w10)$;

- A read operation is required to check if the cell programmed correctly. If the read is false, a fault is detected. An unsuccessful read would imply a 11 or an over-erased state which would be indicative of charge loss due to leakage. $\uparrow r(10)$

Thus, the March Algorithm can be summarized as under:

**FMA-LC-Testing**

*(Flash March Algorithm - Leakage along Column Fault Testing:)*

$\uparrow (w11); \uparrow \{(r11), (w10)\}, \uparrow (r10)$;
Row Leakage

Chapter 4, Section 4.4.2 dealt with the mechanisms which lead to leakage faults along a row. This section will deal with the march algorithms which will test a flash array for leakage along a row.

The starting condition for the algorithm is that only the word lines connected to the cell intended to be programmed are activated. All other word lines are de-activated in order to notice an alternate leakage path. Practically, there would be an actual probe which can detect leakage current along another row. However, since this is beyond the scope of any march algorithm, we will not consider the analog leakage affects. The march algorithm that will be discussed will only deal with the effect of the leaky row on the flash cells, through simple read and write operations.

- Initially the array is erased $\uparrow (w11)$.
- All the cells in the array are checked for the correct erase state and programmed to state 10 by $\uparrow \{(r11), (w10)\}$.
- All cells in the array are read for state 10 and then programmed to state 01 by $\uparrow \{(r10), (w01)\}$.
- All cells in the array are read for state 01 and then programmed to state 00 by $\uparrow \{(r01), (w00)\}$.
- All cells in the array are read for state 00 by $\uparrow \{(r00)\}$.

If there exists an alternate path to ground along a row, electrons will flow off the floating gate causing the $V_t$ of the cell to drop from a programmed state to a lower state. Thus, the state would be different from the state programmed. If the read was successful the algorithm proceeds,
else a fault is detected.

Thus, the March Algorithm can be summarized as under:

**FMA-LR-Testing**

*(Flash March Algorithm - Leakage along Row Fault Testing:)*

\[ \uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01, (w00))\}; \uparrow \{(r00)\}; \]

**Gate-Drain Capacitive Coupling Leakage Testing**

Chapter 4, Section 4.4.3 dealt with leakage faults due to extremely low Vt cells caused by gate-drain coupling. This results in prevention of programming. Table 5.2 summarizes the fault behaviour.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Initial State</th>
<th>Intended State</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>Over-Erased</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

To detect this, all the word lines are grounded. All the array sources are grounded. Only one single column is selected. The current through the cell is measured and the state of the cell can be determined. If a cell has a very low Vt, the post erase repair or PER needs to be performed to bring up the Vt to the desired level so that programming may be performed. PER is a kind of *soft programming*. Due to this Low Vt and high gate-drain coupling, soft programming cannot be achieved and the cell will be stuck-at 11. Again as explained earlier, the march algorithm is limited to only detecting the effect of such type of faults.

- Initially, the array is erased by \( \uparrow (w11) \).
• Since the Vt is lower than the erased state, a read operation is performed and each cell is programmed to state 10 by $\uparrow \{(r11), (w10)\}$. If the read operation is false then a fault is detected.

• A read operation is required to check if the cell programmed correctly and the cells are programmed to the next state of 01 by $\uparrow \{(r10), (w01)\}$. If the read operation is false then a fault is detected.

• A read operation is required to check if the cell programmed correctly and the cells are programmed to the next state of 00 by $\uparrow \{(r10), (w00)\}$. If the read operation is false then a fault is detected.

• A read operation is required to check if the cell programmed correctly by $\uparrow r(00)$. If the read operation is false then a fault is detected.

Thus, the March Algorithm can be summarized as under:

**FMA-Gate-Drain-Testing**

*(Flash March Algorithm - Gate-Drain Coupling Testing:)*

$\uparrow (w11); \uparrow \{(r11), (w10)\}; \uparrow \{(r10), (w01)\}; \uparrow \{(r01), (w00)\} \uparrow (r00)$.

**Stress Induced Leakage Current (SILC)**

SILC being an analog effect can only be detected practically by probes in a test lab. Since the simplicity of march algorithm limits the testing of analog parameters such as current, again, only the effect of this leakage current can be seen in the flash cells. Thus, the effects of the leakage can be detected by the change in state of the flash cell. From Chapter 4, Section 5.4.3, SILC fault behavior would be detectable by the same test patterns as leakage charge. For simplicity only the
March algorithm has been listed here.

Thus, the March Algorithm can be summarized as under:

**FMA-SILC-Testing**

*(Flash March Algorithm - Stress Induced Leakage Current Testing:)*

⇑ (w11); ⇑ {(r11), (w10)}; ⇑ {(r10), (w01)}; ⇑ {(r01, (w00))}; ⇑ {(r00)};

### 5.6 Test Pattern Summary

Sections 1 through 5 dealt with the test patterns for each individual fault that occurs in the MLC array. The aim of any test process is to obtain the maximum fault coverage with a minimal set of test patterns. In other words, the number of march elements to test for all types of faults must be minimal. This need stems from the low test time each unit is allowed to have during the bulk testing process. Therefore, the goal of this thesis, in addition to comprehensively discussing all possible types of faults, is to minimize the test patterns for each type of fault into one compound march algorithm. When this algorithm is applied to the flash array it will test for every possible type of fault discussed. This section will deal with combing the march elements of the algorithms discussed in earlier sections.

Table 5.3 is a recapitulation of the march algorithms for both the traditional and non-traditional faults that may exist in an MLC flash array. Now, having summarized each of the test patterns which can detect all the traditional and non-traditional faults efficiently, Section 7 deals with a new diagonal algorithm (modified from [24]) which will employ the above test patterns at strategic points in the algorithm which will not only detect program disturbs but all of the
non-traditional faults discussed so far.

<table>
<thead>
<tr>
<th>No.</th>
<th>Fault</th>
<th>March Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SAF</td>
<td>(\uparrow (w11); \uparrow {(w10), (r10)}; \uparrow {(w01), (r01)}; \uparrow {(w00), (r00)}; \uparrow (w11); \uparrow (r11))</td>
</tr>
<tr>
<td>2</td>
<td>BF-LBL</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>3</td>
<td>BF-GBL</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>4</td>
<td>BF-WL</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>5</td>
<td>BCF-BL(_1)/WL(_1)</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>6</td>
<td>BCF-WL(_2)/BL(_2)</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>7</td>
<td>BCF - SG</td>
<td>(\uparrow (w11); \uparrow (r11))</td>
</tr>
<tr>
<td>8</td>
<td>DC-E</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>9</td>
<td>DC-P</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>10</td>
<td>DD</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>11</td>
<td>Hi Resistance</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>12</td>
<td>D OxTF</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow (r01))</td>
</tr>
<tr>
<td>13</td>
<td>A OxTF</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>14</td>
<td>Contamination</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>15</td>
<td>Leakage Faults</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
<tr>
<td>16</td>
<td>SILC</td>
<td>(\uparrow (w11); \uparrow {(r11), (w10)}; \uparrow {(r10), (w01)}; \uparrow {(r01), (w00)}; \uparrow (r00))</td>
</tr>
</tbody>
</table>

### 5.7 The MLC Diagonal Algorithm FTX

#### 5.7.1 Introduction

The diagonal march algorithm for flash memory test and diagnosis was proposed in [24]. One of its main advantages is that it effectively reduces the test time without sacrificing fault coverage. The diagonal algorithm will now be modified to test an MLC flash array. We have improved this algorithm such that all the non-traditional faults discussed in Chapter 4 can also be easily detected and diagnosed. After both testing (Diagonal-FT) and diagnostic (Diagonal-FD) algorithms have been discussed in detail, the complexity of the algorithms and the test time will be analyzed. The diagonal algorithms will be compared with a few popular testing algorithms to highlight the effectiveness of the modified diagonal algorithm to MLC flash testing.
5.7.2 The Basic Diagonal Algorithm

The diagonal algorithm is mainly used to detect and diagnose the disturbances that occur in an MLC array. From the definition of disturbances, the faulty cell is always on the same word-line or the same bit-line as that of the cell being programmed. Thus programming a cell may activate two types of disturbances. The main idea of this algorithm is to perform read and write operations on the two diagonals of a flash array. Let the total number of rows be \( m \) and the total number of columns be \( n \). The total size of the array is now \( m \times n = mn \). However, for simplicity the array under consideration is a 4 x 4 square array of 16 flash cells in the following discussion. The two diagonals are represented by D1 and D2.

D1 is the diagonal along (-45\(^\circ\)) while D2 is the diagonal along (+45\(^\circ\)). All elements not on diagonal D1 are represented by !D1 and those not on diagonal D2 are !D2. The basic idea of this algorithm from a testing standpoint is to detect all faults along D1 and !D1. This can ensure testing the entire array. Since there are four kinds of disturbances, namely, *word line erase disturbance* (WED), *word line program disturbance* (WPD), *bit line erase disturbance* (BED), *bit line program disturbance* (BPD), the algorithm must be able to detect them along D1 and !D1. If all the above faults are detected along D1 and !D1, then the algorithm is efficient. The diagonal algorithm for SBC flash is presented in [24]. In this chapter, the diagonal algorithm will be modified for MLC flash. In the process, the non-traditional faults will be revisited and their test patterns will be highlighted. This is done to include the non-traditional fault test patterns within the MLC diagonal flash algorithm to detect conventional / traditional faults that may occur within the flash array. As the general rule of any testing algorithm, our goal is to test the maximum number of faults using the minimal number of test patterns. The MLC diagonal algorithm, however, does not
detect broken connection and bridging faults. These will have to be discussed and tested separately.

5.7.3 Implementation of the Diagonal FT Algorithm

Chapter 4 discusses the various disturbs that occur within a flash array. The disturb faults that may occur in an MLC flash array are summarized here for convenience. From Table 6.3, the WED and BED have the same fault behaviour, while the WPD and BPD also share the same fault behavior. Hence, testing for one will automatically test for the other.

<table>
<thead>
<tr>
<th>Type</th>
<th>Original State</th>
<th>WED</th>
<th>BED</th>
<th>WPD</th>
<th>BPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>OE</td>
<td>OE</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Now, the MLC flash algorithm will be discussed and all the steps will be clearly illustrated using tables.

**Step 1**: Initially, the flash array is erased, i.e., all cells in the array under test now contain "11".

This can be depicted in Table 6.4.

**Step 2**: All cells not on diagonal 1 (!D1) are first read (r11). Any cell which does not yield the correct Vt value corresponding to state 11, is detected and over-erased faults are detected. After the read operation (r11), each of the non-diagonal flash cells programmed to the next state "10".

By programming all of the non-diagonal cells, the WED & BED of type 1 and the WPD & BPD
of type 1 faults are activated on the cells on D1. This can be depicted in Table 5.6.

Table 5.6: Diagonal FT - Step 2

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

**Step 3:** All the cells on D1 are read to check if any of the disturb faults were activated during the previous step. A (r11) is performed on all cells along D1. If some cells do not possess the Vt corresponding to state 11, they can be observed. Hence, all the WED & BED of type 1 and WPD & BPD of type 1 faults on D1 are successfully detected in this step. After each cell is read as described above, all cells along D1 are programmed to the next higher voltage level which is ”10”. By programming cells along D1, the WED & BED of type 2 and WPD & BPD of type 2 are activated on the non-diagonal elements (!D1). This can be depicted in Table 6.5.

Table 5.7: Diagonal FT - Step 3

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Step 4:** All cells not on diagonal 1 (!D1) are first read (r10). Any cell which does not yield the correct Vt value corresponding to state 10, can be observed, and the WED & BED of type 2 and WPD & BPD of type 2 faults on !D1 are successfully detected in this step. After the read operation (r10) on !D1, each of the non-diagonal flash cells is programmed to the next state of ”01” (w01). By programming all of the non-diagonal cells, the WED & BED of type 2 and the WPD & BPD of type 2 faults are activated on D1. This can be depicted in Table 6.6.

**Step 5:** All cells along D1 are read (r10). Any cells which do not yield the correct Vt value
corresponding to state 10. are flagged and the WED & BED of type 2 and WPD & BPD of type 2 are detected on D1. After the read operation (r10), each of the diagonal cells are programmed to the next state of ”01” (w01). By programming all of the cells on D1, the WED & BED of type 3 and WPD & BPD of type 3 Faults are activated on !D1. This can be depicted in Table 6.7.

| 10 | 01 | 01 | 01 |
| 01 | 10 | 01 | 01 |
| 01 | 01 | 10 | 01 |
| 01 | 01 | 01 | 10 |

Step 6: All cells not on diagonal 1 (!D1) are first read (r01). Any cell which does not yield the correct Vt value corresponding to state 01, can be observed and the WED & BED of type 3 and WPD & BPD of type 3 faults on the non-diagonal (!D1) are successfully detected in this step. After the read operation (r10), each of the non-diagonal flash cells is programmed to the next state ”00” (w00). By programming all of the non-diagonal cells, the WED & BED of type 3 and the WPD & BPD of type 2 faults are activated on D1. This can be depicted in Table 6.8.

| 01 | 01 | 01 | 01 |
| 01 | 01 | 01 | 01 |
| 01 | 01 | 01 | 01 |
| 01 | 01 | 01 | 01 |

Step 7: All cells along D1 are read (r01). Any cells which do not yield the correct Vt value corresponding to state 01 can be observed and the WED & BED of type 3 and WPD & BPD
of type 3 faults are detected on D1. After the read operation (r01), each of the diagonal cells is programmed to the next state "00" (w00). By programming all of the cells on D1, the WED & BED of type 4 faults are activated on !D1. This can be depicted in Table 6.9.

Table 5.11: Diagonal FT - Step 7

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

**Step 8:** All the non diagonal cells (!D1) are read (r00). If any of the cells does not yield a Vt corresponding to state 00, they can be observed and the WED and BED of type 4 faults are detected. This can be depicted in Table 6.10.

Table 5.12: Diagonal FT - Step 8

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>00</th>
<th>00</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

**Step 9:** At this point, all the cells in the array have been programmed to state "00". The flash cells cannot be programmed further. Hence in order for the algorithm to continue the entire array must be erased. This can be depicted in Table 6.11.

Table 5.13: Diagonal FT - Step 9

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
**Step 10:** All the cells along D1 are now read (r11). All the over erase faults along D1 are detected. After the read is performed, all cells on D1 are programmed to state 00 (w00). This activates the WED & BED of type 1 and WPD & BPD of type 1 faults on the non-diagonal cells (!D1). This can be depicted in Table 6.12.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

**Step 11:** All the cells on !D1 are read for state 11. If any of them are over-erased then the WED and BED of type 1 and WPD & BPD of type 1 faults are detected on !D1. This can be depicted in Table 6.13.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

**Step 12:** All the cells along the second diagonal (D2) are programmed to state 00. This activates all the WED and BED of type 4 on D1 and this can be depicted in Table 6.14.

**Step 13:** All cells along D1 are read. The expected value is 00. If the cells along D1 are not in state 00, then the WED and BED of type 4 along D1 are detected successfully. This can be
Table 5.16: Diagonal FT - Step 12

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>11</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
</tbody>
</table>

depicted in Table 6.15.

Table 5.17: Diagonal FT - Step 13

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>11</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
</tbody>
</table>

The above thirteen steps of the modified diagonal FTX algorithm are summarized in Table 5.18.

Table 5.18: Diagonal March Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$w(11)$</td>
</tr>
<tr>
<td>2</td>
<td>$(!D1) \uparrow r(11), w(10)$</td>
</tr>
<tr>
<td>3</td>
<td>$(D1) \uparrow r(11), w(10)$</td>
</tr>
<tr>
<td>4</td>
<td>$(!D1) \uparrow r(10), w(01)$</td>
</tr>
<tr>
<td>5</td>
<td>$(D1) \uparrow r(10), w(01)$</td>
</tr>
<tr>
<td>6</td>
<td>$(!D1) \uparrow r(01), w(00)$</td>
</tr>
<tr>
<td>7</td>
<td>$(D1) \uparrow r(01), w(00)$</td>
</tr>
<tr>
<td>8</td>
<td>$(!D1) \uparrow r(00)$</td>
</tr>
<tr>
<td>9</td>
<td>$w(11)$</td>
</tr>
<tr>
<td>10</td>
<td>$(D1) \uparrow r(11), w(00)$</td>
</tr>
<tr>
<td>11</td>
<td>$(!D1) \uparrow r11$</td>
</tr>
<tr>
<td>12</td>
<td>$(D2) \uparrow w(00)$</td>
</tr>
<tr>
<td>13</td>
<td>$(D1) \uparrow r(00)$</td>
</tr>
</tbody>
</table>

5.8 Test Pattern Merging

The modified diagonal algorithm described can be modified further to detect all the traditional and non-traditional faults discussed in Chapter 4. Referring to Table 5.3, the patterns indicated may now be analyzed and broken down into smaller patterns to be merged with the
diagonal FTX algorithm. The diagonal algorithm only takes care of over-erase faults and faults due to disturbances. The non-traditional faults (SRF, OXF, CF, LF & SILC) and traditional faults (SAF, BF & BCF) are not detected at this stage. Hence, we need to extract those patterns which will detect these faults and place them at strategic points within the diagonal-FTX algorithm. This will not only ensure 100% fault coverage but also maintain the low complexity of the diagonal algorithm, as compared to conventional march algorithms for testing. The non-traditional faults in Table 5.3 are now effectively grouped together, such that all the faults which have similar test patterns, will be detected at the same time (test pattern minimization). This process is summarized in Table 5.19, which lists the patterns which activate and detect all of the non-traditional faults.

Table 5.19: Non Traditional Fault Activation and Detection Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Faults Activated</th>
<th>Detecting Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>w(00)</td>
<td>Srf1, Aox1, CL1, LC1, LR1, IDL1, Silc1</td>
<td>w(00)r(00)</td>
</tr>
<tr>
<td>w(01)</td>
<td>Srf2, Dox1, Aox2, CL2, CG1, LC2, LR2, IDL2, Silc2</td>
<td>w(01)r(01)</td>
</tr>
<tr>
<td>w(10)</td>
<td>Srf3, Dox2, Aox3, CL3, CG2, LC3, LR3, IDL3, Silc3</td>
<td>w(10)r(10)</td>
</tr>
<tr>
<td>w(11)</td>
<td>Srf4, Dox4, Aox4, CL4, CG3, IDL4, Silc4</td>
<td>w(11)r(11)</td>
</tr>
</tbody>
</table>

Table 5.20 lists the patterns which activate and detect all of the traditional MLC faults described in the fault model. (The march element \( r(\text{prev.pgm.state}) \) is the previous value programed into a flash cell from an earlier step in the diagonal algorithm.) The patterns \( r(XX), w(XX) \) will detect the bridging faults and the patterns \( w(XX), r(XX) \) will detect the stuck-at faults and the broken connection faults in the array.

The Diagonal-FTX march algorithm in Table 5.18 will now be appended by including the detecting patterns from Table 5.19 and Table 5.20. The final MLC diagonal FTX algorithm which
Table 5.20: Traditional Fault Activation and Detection Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Faults Activated</th>
<th>Detecting Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>w(00)</td>
<td>SA 01, BF3, BCF-WL/BL</td>
<td>r(prev_pgm_state),w(00),r(00)</td>
</tr>
<tr>
<td>w(01)</td>
<td>SA 10, BF2, BCF-WL/BL</td>
<td>r(prev_pgm_state),w(01),r(01)</td>
</tr>
<tr>
<td>w(10)</td>
<td>SA 11, BF1, BCF-WL/BL</td>
<td>r(prev_pgm_state),w(10),r(10)</td>
</tr>
<tr>
<td>w(11)</td>
<td>SA 00, BCF-SG</td>
<td>r(prev_pgm_state),w(11),r(11)</td>
</tr>
</tbody>
</table>

will detect all flash memory disturbs as well as all of the traditional and non-traditional faults is represented in Table 5.21.

Note:

A very important point to be noted here is the reason why we implement the diagonal-FTX algorithm. The diagonal algorithm, was mainly aimed at detecting the disturbances efficiently and was extended to detect all of the remaining faults. The complexity and test time of the diagonal algorithm, is a little higher than the universal pattern. However, the universal pattern derived to detect each fault, is by far the most efficient test pattern to successfully detect every fault in the MLC fault model. Each fault discussed thus far, was uniquely considered and the test patterns used to detect each type was the most optimal. Hence, one may argue the use of the diagonal algorithm, since it does not have the lowest MLC complexity. Our reason to use the diagonal algorithm, is fault diagnosis. Chapter 6 derives a very unique MLC diagonal-FDX algorithm, which is very efficient at diagnosing all of the MLC faults. Since the final goal of any test/diagnosis is to determine the cause of the fault, our diagonal diagnosis algorithm in this case determines our testing strategy. In other words, since the final diagnosis algorithm is a diagonal one, we implement our testing strategy along the same lines so that the testing algorithm may be suitably extended to fault diagnosis.
Table 5.21: Modified Diagonal March Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \uparrow w(11) );</td>
</tr>
<tr>
<td>2</td>
<td>( (!D1) \uparrow r(11), w(10), r(10) );</td>
</tr>
<tr>
<td>3</td>
<td>( (D1) \uparrow r(11), w(10), r(10) );</td>
</tr>
<tr>
<td>4</td>
<td>( (!D1) \uparrow r(10), w(01), r(01) );</td>
</tr>
<tr>
<td>5</td>
<td>( (D1) \uparrow r(10), w(01), r(01) );</td>
</tr>
<tr>
<td>6</td>
<td>( (!D1) \uparrow r(01), w(00), r(00) );</td>
</tr>
<tr>
<td>7</td>
<td>( (D1) \uparrow r(01), w(00), r(00) );</td>
</tr>
<tr>
<td>8</td>
<td>( (!D1) \uparrow r(00) );</td>
</tr>
<tr>
<td>9</td>
<td>( \uparrow w(11) );</td>
</tr>
<tr>
<td>10</td>
<td>( (D1) \uparrow r(11), w(00), r(00) );</td>
</tr>
<tr>
<td>11</td>
<td>( (!D1) \uparrow r11 );</td>
</tr>
<tr>
<td>12</td>
<td>( (D2) \uparrow w(00), r(00) );</td>
</tr>
<tr>
<td>13</td>
<td>( (D1) \uparrow r(00) );</td>
</tr>
</tbody>
</table>

5.8.1 Test Length Calculation

As mentioned in previous subsections, there have been many test algorithms proposed for flash memories. However, MLC flash algorithms are very new and not much work has been done in this area. The various march elements of the modified diagonal algorithm is clearly illustrated in Table 5.21. Let us assume that the time taken to read a cell is the same for all possible Vt levels/states. Hence, \( r_{11} = r_{10} = r_{01} = r_{00} \). Similarly, let us assume that the time taken to program a cell to any of the programmed states is the same. Therefore \( w_{10} = w_{01} = w_{00} \). Let the time for any read operation be \( r \), and the time for any write operation or program operation be \( p \). Let the time to erase the entire array be \( e \). Now, possessing the time taken for all the basic flash operations, we may now begin to theoretically derive the complexity or test length of the mlc diagonal algorithm. The Diagonal FTX algorithm (Table 5.21) can now be represented in terms of read \( (r) \), program\( (p) \) and erase \( (e) \) operations. Each of the diagonal FTX march steps and their corresponding complexities have been summarized in Table 5.22.
Table 5.22: Modified Diagonal March Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
<th>Corresponding Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>↑ w(11);</td>
<td>e</td>
</tr>
<tr>
<td>2</td>
<td>(!D1) ↑ r(11), w(10), r(10);</td>
<td>!D1(r+p+r)</td>
</tr>
<tr>
<td>3</td>
<td>(D1) ↑ r(11), w(10), r(10);</td>
<td>D1(r+p+r)</td>
</tr>
<tr>
<td>4</td>
<td>(!D1) ↑ r(10), w(01), r(01);</td>
<td>!D1(r+p+r)</td>
</tr>
<tr>
<td>5</td>
<td>(D1) ↑ r(10), w(01), r(01);</td>
<td>D1(r+p+r)</td>
</tr>
<tr>
<td>6</td>
<td>(!D1) ↑ r(01), w(00), r(00);</td>
<td>!D1(r+p+r)</td>
</tr>
<tr>
<td>7</td>
<td>(D1) ↑ r(01), w(00), r(00);</td>
<td>D1(r+p+r)</td>
</tr>
<tr>
<td>8</td>
<td>(!D1) ↑ r(00);</td>
<td>!D1(r)</td>
</tr>
<tr>
<td>9</td>
<td>↑ w(11);</td>
<td>e</td>
</tr>
<tr>
<td>10</td>
<td>(D1) ↑ r(11), w(00), r(00);</td>
<td>D1(r+p+r)</td>
</tr>
<tr>
<td>11</td>
<td>(!D1) ↑ r11;</td>
<td>!D1(r)</td>
</tr>
<tr>
<td>12</td>
<td>(D2) ↑ w(00), r(00);</td>
<td>D2(p+r)</td>
</tr>
<tr>
<td>13</td>
<td>(D1) ↑ r(00);</td>
<td>D1(r)</td>
</tr>
</tbody>
</table>

If the number of rows and columns in the array are not equal, then the number of diagonal elements is represented by \( \max(m, n) \), where \( m \) and \( n \) are the number of rows and columns respectively. The total number of elements in the array would be \( m \times n \). Therefore, the total number of non-diagonal elements can be calculated as \( mn - \max(m, n) \). Thus the complexity of the algorithm is obtained by adding the complexity of each step of the above algorithm (Table 5.22.)

\[
C = (2)e + (3mn)(2r + p) + (mn - \max(m, n))(r) + (\max(m, n))(2r + p) + (mn - \max(m, n))(r) + (\max(m, n))(p + r) + (\max(m, n))(r)
\]

\[
C = 2e + (8mn + \max(m, n))r + (3mn + 2\max(m, n))p
\]

Hence, from the above result, if \( (m^*n) = N \) ———–(1)

and by approximating \( m=n \),

then \( \max(m,n) = n \) ———–(2)

then \( N = m^*n = n^*n = n^2 \) ———–(3)
Hence, \( n = \sqrt{n^2} = \sqrt{N} \) ——— (4)

The Complexity for erase, program and read is summarized in Table 5.23.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Erase</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Read</td>
<td>( 8N + \sqrt{N} )</td>
</tr>
<tr>
<td>3</td>
<td>Program</td>
<td>( 3N + 2\sqrt{N} )</td>
</tr>
</tbody>
</table>

### 5.8.2 Test Time Calculation

Having calculated the number of program and read cycles, the test time for all the MLC algorithms will now be determined. The following data is used for a 2MBbit flash array [24]:

- The standard erase time is 200 milli seconds
- The standard program time is 1.5 micro seconds
- The standard read time is 1.25 nano seconds

Implementing the above values and calculate the test time for our MLC test algorithm, we have

\[
C = 2e + (8mn + \max(m, n))r + (3mn + 2\max(m, n))p
\]

\[
C = 2e + (8N + \sqrt{N}))r + (3N + 2\sqrt{N})p
\]

\[
T = 2(200ms)+(8(2\text{Megabits})+\sqrt{(2\text{Megabits}}))(1.25ns)+(3(2\text{Megabits})+2\sqrt{(2\text{Megabits}}))(1.5\text{us})
\]

\[
T = (0.4 + 0.0200 + 9.00424) \text{ seconds.}
\]

\[
T = 9.4242 \text{ seconds.}
\]

The new diagonal-FTX algorithm test time is **9.4 seconds**.
Chapter 6

The MLC Fault Diagnosis

Chapters 5 discussed the MLC flash memory testing methodology. The march algorithms discussed and proposed only dealt with testing the chip for the presence of a fault. There was no emphasis on the types of faults. In this chapter, a new diagonal algorithm will be proposed which will diagnose all the traditional and non-traditional faults discussed in the fault model (Chapter 4).

The MLC fault model was formulated by considering the operating conditions activating each fault. After acknowledging the behavior of each fault, the test patterns that detected these faults were considered. This chapter begins with a brief introduction to how MLC faults will be diagnosed, and then the new MLC diagnosis algorithm will be proposed. As described in the fault testing chapter, the main aim of this diagnosis algorithm is to diagnose the maximum number of faults with a minimal number of patterns or algorithm steps. This will in turn, keep the cost of fault diagnosis to a minimum. This is the driving force behind merging the diagnosing patterns / algorithms of various types of faults into one effective test algorithm. This will be dealt with in the final section on pattern minimization. The last section of the chapter deals with calculating the run-time and complexity of the algorithm in terms of read, program and erase operations.
6.1 March Algorithms Modification for Diagnosis

The march algorithms discussed for MLC testing will be used as a starting point for MLC fault diagnosis. From Chapter 4, a typical testing march algorithm looks like this:

$$\uparrow \{w_{00}, r_{00}\};$$

An important point to be noted is that for all testing march algorithms, the read element is always done for the expected state of the flash cell. If a fault occurred, by reading the expected value, only the presence of the fault can be detected. No information about the faulty types can be gathered. Hence, an important modification to the march elements is to perform a read operation for the faulty value. If the read value is matched with the expected faulty value of a defect, then one of the various faults will be identified or diagnosed. For instance, if some of the cells in the array cannot be programmed to state 00 due to acceptor traps in the tunnel oxide, then the final state of the cell after programming to state 00 (w00) will be state 01. Therefore, in the above march algorithm, a $$\uparrow r_{00}$$ will not identify this fault. If march element r00 is replaced by r01, then the acceptor oxide trap fault will be identified (diagnosed) successfully. The diagnosing algorithm would now be:

$$\uparrow \{w_{00}, r_{01}\};$$

Note that the read value presented in the diagnosis march algorithm is the expected value of a hypothetic defect.

6.2 Fault Diagnosis Pattern Generation

The traditional and non-traditional fault model was discussed in Chapter 4. The test patterns for each fault (Chapter 5) will now be modified for fault diagnosis. The behavior of each fault will be summarized for convenience, and the diagnosing march algorithm elements will be
determined. Since the diagnosing algorithms are, (in most cases,) a modification of the testing algorithms, the diagnosing algorithm steps will not be explained in detail. Only the patterns for each individual fault will be summarized based on their fault behavior.

### 6.2.1 Fault Behavior and Diagnosis

Table 6.1 lists all of the MLC faults under consideration, their faulty values and the diagnosing patterns. The pattern for each fault is listed under the assumption that only that particular fault has occurred in the array. Clearly, the fault behavior decides the diagnosing patterns. These patterns are different from fault testing patterns, and the overall number of read operations will be significantly larger. Obviously, these patterns will need to be merged at a later stage to effectively reduce the test time at the same time to cover all of the faults.

From Table 6.1, the faults which have the common behavior (faults numbered 16 through 48), can be grouped appropriately and this process can be summarized in Table 6.2. Even though faults numbered 9 through 15 may also be included in this classification, they are diagnosed separately by the diagonal algorithm and are hence excluded from this grouping. Having summarized each of the test patterns which can detect the traditional and non-traditional faults, the following section proposes a new diagnosis algorithm which will employ the above test patterns at strategic points. The algorithm not only diagnoses program disturbs but all of the remaining traditional and non-traditional faults discussed so far.

### 6.3 The MLC Diagonal Diagnosis Algorithm FDX

#### 6.3.1 Introduction

The diagonal march algorithm for flash memory test and diagnosis was proposed in [24]. One of its main advantages is that it effectively reduces the test time without sacrificing fault...
Table 6.1: Table Summarizing MLC Flash Memory Fault Behavior & Diagnosis

<table>
<thead>
<tr>
<th>No.</th>
<th>Type of Fault</th>
<th>Intended State</th>
<th>Faulty State</th>
<th>Diagnosing Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stuck At 00 (SA 00)</td>
<td>XX</td>
<td>00</td>
<td>↑(w00);↑ r00;</td>
</tr>
<tr>
<td>2</td>
<td>Stuck At 01 (SA 01)</td>
<td>XX</td>
<td>01</td>
<td>↑(w01);↑ r01;</td>
</tr>
<tr>
<td>3</td>
<td>Stuck At 10 (SA 10)</td>
<td>XX</td>
<td>10</td>
<td>↑(w10);↑ r10;</td>
</tr>
<tr>
<td>4</td>
<td>Stuck At 11 (SA 11)</td>
<td>XX</td>
<td>11</td>
<td>↑(w11);↑ r11;</td>
</tr>
<tr>
<td>5</td>
<td>WL-WL Bridging Faults (BF-WL)</td>
<td>&quot;S&quot;</td>
<td>&quot;S&quot;</td>
<td>↑ wXX (n);↑ rXX (n+4);</td>
</tr>
<tr>
<td>6</td>
<td>BL-BL Bridging Fault (BF-BL)</td>
<td>&quot;S&quot;</td>
<td>&quot;S&quot;</td>
<td>↑ wXX (n);↑ rXX (n+1);</td>
</tr>
<tr>
<td>7</td>
<td>Broken Connection (BCF-WL,BL)</td>
<td>XX</td>
<td>&quot;U&quot;</td>
<td>↑ w(state X);↑ r(state X);</td>
</tr>
<tr>
<td>8</td>
<td>Broken Connection (BCF-SG)</td>
<td>XX</td>
<td>&quot;U&quot;</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>9</td>
<td>Gate/Drain Stress (WPD/BPD 1)</td>
<td>01</td>
<td>00</td>
<td>↑ w01;↑ r01;</td>
</tr>
<tr>
<td>10</td>
<td>Gate/Drain Stress (WPD/BPD 2)</td>
<td>10</td>
<td>01</td>
<td>↑ w10;↑ r01;</td>
</tr>
<tr>
<td>11</td>
<td>Gate/Drain Stress (WPD/BPD 3)</td>
<td>11</td>
<td>10</td>
<td>↑ w11;↑ r10;</td>
</tr>
<tr>
<td>12</td>
<td>Gate/Drain Stress (WED/BED 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r10;</td>
</tr>
<tr>
<td>13</td>
<td>Gate/Drain Stress (WED/BED 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>14</td>
<td>Gate/Drain Stress (WED/BED 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>15</td>
<td>Gate/Drain Stress (WED/BED 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>16</td>
<td>Series Resistance (SRF 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r01;</td>
</tr>
<tr>
<td>17</td>
<td>Series Resistance (SRF 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>18</td>
<td>Series Resistance (SRF 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>19</td>
<td>Series Resistance (SRF 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>20</td>
<td>Acceptor Type Oxide Trap (AOx 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r01;</td>
</tr>
<tr>
<td>21</td>
<td>Acceptor Type Oxide Trap (AOx 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>22</td>
<td>Acceptor Type Oxide Trap (AOx 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>23</td>
<td>Acceptor Type Oxide Trap (AOx 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>24</td>
<td>Donor Type Oxide Trap (DOx 1)</td>
<td>01</td>
<td>00</td>
<td>↑ w01;↑ r00;</td>
</tr>
<tr>
<td>25</td>
<td>Donor Type Oxide Trap (DOx 2)</td>
<td>10</td>
<td>01</td>
<td>↑ w10;↑ r01;</td>
</tr>
<tr>
<td>26</td>
<td>Donor Type Oxide Trap (DOx 3)</td>
<td>11</td>
<td>10</td>
<td>↑ w11;↑ r10;</td>
</tr>
<tr>
<td>27</td>
<td>Contamination Charge Loss (CL 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r01;</td>
</tr>
<tr>
<td>28</td>
<td>Contamination Charge Loss (CL 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>29</td>
<td>Contamination Charge Loss (CL 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>30</td>
<td>Contamination Charge Loss (CL 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>31</td>
<td>Contamination Charge Gain (CG 1)</td>
<td>01</td>
<td>00</td>
<td>↑ w01;↑ r00;</td>
</tr>
<tr>
<td>32</td>
<td>Contamination Charge Gain (CG 2)</td>
<td>10</td>
<td>01</td>
<td>↑ w10;↑ r01;</td>
</tr>
<tr>
<td>33</td>
<td>Contamination Charge Gain (CG 3)</td>
<td>11</td>
<td>10</td>
<td>↑ w11;↑ r10;</td>
</tr>
<tr>
<td>34</td>
<td>Column Charge Leakage (LC 1)</td>
<td>00</td>
<td>11</td>
<td>↑ w00;↑ r11;</td>
</tr>
<tr>
<td>35</td>
<td>Column Charge Leakage (LC 2)</td>
<td>01</td>
<td>11</td>
<td>↑ w01;↑ r11;</td>
</tr>
<tr>
<td>36</td>
<td>Column Charge Leakage (LC 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>37</td>
<td>Row Charge Leakage (LR 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r01;</td>
</tr>
<tr>
<td>38</td>
<td>Row Charge Leakage (LR 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>39</td>
<td>Row Charge Leakage (LR 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>40</td>
<td>Gate-Drain coupling Fault (GDC 1)</td>
<td>00</td>
<td>11</td>
<td>↑ w00;↑ r10;</td>
</tr>
<tr>
<td>41</td>
<td>Gate-Drain coupling Fault (GDC 2)</td>
<td>01</td>
<td>11</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>42</td>
<td>Gate-Drain coupling Fault (GDC 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>43</td>
<td>Gate-Drain coupling Fault (GDC 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
<tr>
<td>45</td>
<td>SILC Fault (SILC 1)</td>
<td>00</td>
<td>01</td>
<td>↑ w00;↑ r01;</td>
</tr>
<tr>
<td>46</td>
<td>SILC Fault (SILC 2)</td>
<td>01</td>
<td>10</td>
<td>↑ w01;↑ r10;</td>
</tr>
<tr>
<td>47</td>
<td>SILC Fault (SILC 3)</td>
<td>10</td>
<td>11</td>
<td>↑ w10;↑ r11;</td>
</tr>
<tr>
<td>48</td>
<td>SILC Fault (SILC 4)</td>
<td>11</td>
<td>OE</td>
<td>↑ w11;↑ rOE;</td>
</tr>
</tbody>
</table>
coverage. The diagonal algorithm which was proposed to test an MLC flash array in Chapter 5 will now be improved to diagnose all of the faults discussed in the fault model (Chapter 4). After the diagnostic (diagonal-FDX) algorithm has been discussed in detail, the complexity and the test time will be analyzed and calculated.

### 6.3.2 Motivation

The universal test patterns derived in Chapter 5 are useful only for testing purposes. However, if they are used for diagnosis, then they are not effective. The reason can be explained as follows. Assume the entire array is written to state 01 for example. Due to a disturb, if a cell’s state gets converted to state 10, then the only information gathered from the universal pattern is that an erase disturb is noticed. Since there are four kinds of disturbs namely WED, WPD, BED and BPD, the fault can be either due to a WED or a BED. Thus, at this stage there is no indication if the fault is due to a WED or a BED. By implementing the diagonal algorithm, each time a diagonal cell is programmed (e.g., state 01), read operations are performed on all the cells on the same bit line (to diagnose BED or BPD) and the same word line (to diagnose WED or WPD). Since read operations \( r(01) \) are performed immediately following the write operation, the source of the program fault or the erase disturb fault will be diagnosed successfully. This is the most

---

Table 6.2: MLC Fault Diagnosis Pattern Grouping

<table>
<thead>
<tr>
<th>MLC Fault</th>
<th>Diagnosing Patterns</th>
<th>MLC Fault Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRF 4, AOx 4, CL 4, SILC 4, GDC 4</td>
<td>( \uparrow { w(11), r(OE) } )</td>
<td>NT&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>CG 3, DOx 3</td>
<td>( \uparrow { w(11), r(10) } )</td>
<td>NT&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
<tr>
<td>SRF 3, AOx 3, CL 3, LR 3, SILC 3, GDC 3, LC 3</td>
<td>( \uparrow { w(10), r(11) } )</td>
<td>NT&lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
<tr>
<td>CG 2, DOx 2</td>
<td>( \uparrow { w(10), r(01) } )</td>
<td>NT&lt;sub&gt;4&lt;/sub&gt;</td>
</tr>
<tr>
<td>LC 2, GDC 2</td>
<td>( \uparrow { w(01), r(11) } )</td>
<td>NT&lt;sub&gt;5&lt;/sub&gt;</td>
</tr>
<tr>
<td>SRF 2, AOx 2, CL 2, LR 2, SILC 2</td>
<td>( \uparrow { w(01), r(10) } )</td>
<td>NT&lt;sub&gt;6&lt;/sub&gt;</td>
</tr>
<tr>
<td>CG 1, DOx 1</td>
<td>( \uparrow { w(01), r(00) } )</td>
<td>NT&lt;sub&gt;7&lt;/sub&gt;</td>
</tr>
<tr>
<td>LC1, GDC 1</td>
<td>( \uparrow { w(00), r(11) } )</td>
<td>NT&lt;sub&gt;8&lt;/sub&gt;</td>
</tr>
<tr>
<td>SRF 1, AOx 1, CL 1, LR 1, SILC 1</td>
<td>( \uparrow { w(00), r(01) } )</td>
<td>NT&lt;sub&gt;9&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
important reason, why we claim the diagonal algorithm will efficiently diagnose all of the MLC disturbs in the array

6.3.3 The Diagnostic Algorithm

The diagnosis algorithm is first used to diagnose all disturbance defects that occur in a MLC array. The main idea of this algorithm is to perform read and write operations on the two diagonals of a flash array. Let the total number of rows be $m$, and the total number of columns be $n$. The size of the entire array is now $mn$. However, for simplicity, the array under consideration is a $4 \times 4$ square array of 16 flash cells in the following discussion. The two diagonals are represented by $D_1$ and $D_2$.

$D_1$ is the diagonal along $(-45^\circ)$ while $D_2$ is the diagonal along $(+45^\circ)$. All elements not on diagonal $D_1$ are represented by $!D_1$ and those not on diagonal $D_2$ are $!D_2$. The basic idea of this algorithm from a diagnosis standpoint is to identify all faults along $D_1$ and $!D_1$. This ensures diagnosing the entire array. Since there are four kinds of disturbances, namely, word line erase disturbance (WED), word line program disturbance (WPD), bit line erase disturbance (BED), bit line program disturbance (BPD), the algorithm must be able to diagnose them along $D_1$ and $!D_1$. If all the above faults are identified along $D_1$ and $!D_1$, then the algorithm is efficient.

In the following subsections, the diagonal testing algorithm (Chapter 5) will be modified for fault diagnosis. In the process, the non-traditional faults summarized in Table 6.2 and the traditional fault patterns presented in Table 6.1, will be merged in appropriate steps of the algorithm. Again, our goal is to diagnose all of the faults using the minimal number of march elements patterns.
6.3.4 Implementation of Diagonal FDX Algorithm

Chapter 4 discusses all of the various disturbs that occur within a flash array. The disturb
faults that may occur in an MLC flash array are summarized here for convenience. From Table
6.3, the WED and BED have the same fault behaviour, while the WPD and BPD also share the
same fault behavior. Hence, testing for one will automatically test for the other.

Table 6.3: MLC Disturb Fault Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Original State</th>
<th>WED state</th>
<th>BED state</th>
<th>WPD state</th>
<th>BPD state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>OE</td>
<td>OE</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>3</td>
<td>01</td>
<td>10</td>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Now, the MLC flash algorithm will be discussed and all the steps will be clearly illustrated
using tables. All cells along a word line are represented by "i", while traversing cells along the
word line will have the notation: \(\uparrow_i\). Similarly, all cells along a bit line are represented by "j", and
traversing cells along the bit line, will have the notation: \(\uparrow_j\).

**Step 1:** Initially, the flash array is erased, i.e., all cells in the array under diagnosis now contain
"11". This can be depicted in Table 6.4. This not only initializes the entire array for the remainder
of the algorithm, but also activates the over-erased faults in the array using \(\uparrow w(11)\).

Table 6.4: Diagonal FDX - Step 1

\[
\begin{array}{cccc}
11 & 11 & 11 & 11 \\
11 & 11 & 11 & 11 \\
11 & 11 & 11 & 11 \\
11 & 11 & 11 & 11 \\
\end{array}
\]

**Step 2:** The cells in the array are now read to check if the cells are over-erased. Any cell which
possesses a Vt less than that of state 11 will be read as over erased. Thus, all of the over-erased
faults (if any) in the array are identified by \(\uparrow r(OE)\).

**Step 3:** All cells on diagonal D1 are programmed to state 10. This will activate all WED & BED faults of type 4 and WPD & BPD faults of type 3 on the non-diagonal elements (!D1). After each cell is programmed, all cells on !D1 on the same row and column as the cell being programmed are read. Each read operation performed on each cell of !D1 may be OE or 10 for erase or program disturb defect respectively. If the read value is OE or 10, then the corresponding fault is diagnosed. Thus, all of the WED & BED faults of type 4 and WPD & BPD faults of type 3 on !D1 are successfully identified. This is depicted in Table 6.5 and the march element is 
\[ D1 : \uparrow w(10); !D1 : \{\uparrow_i r(OE), r(10); \uparrow_j r(OE), r(10)\} \].

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>11</th>
<th>11</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Step 4:** All the cells on D2 are now programmed to state 10. This will activate all WED & BED faults of type 3 and WPD & BPD faults of type 2 on D1. After each of the D2 cells is programmed, the cells on D1 which are on the same bit line and word line as the programmed cell, are read. A read operation is performed on each cell on D1. If the read value is 11, then a WED or a BED fault of type 3 is identified. If the read value is 01, then a WPD or a BPD fault of type 2 is identified. Thus all the WED & BED faults of type 3 and the WPD & BPD faults of type 2 on D1 can be successfully identified. This can be depicted in Table 6.6 using march element 
\[ D2 : \uparrow w(10); D1 : \{\uparrow_i r(11), r(01); \uparrow_j r(11), r(01)\} \].

\[126\]
Step 5: All the non-diagonal cells are now programmed to state 10. This is done so that, the algorithm can continue with all cells in the array with the state of 10. This can be depicted in Table 6.7 using the march element $!D1, !D2 : \uparrow w(10)$.

Step 6: All cells on D1 are programmed to state 01. This will activate all WED & BED faults of type 3 and WPD & BPD faults of type 2 on the non-diagonal elements ($!D1$). After each cell is programmed, all cells on $!D1$, on the same row and column as the cell being programmed are read. Each read operation performed on each cell of $!D1$ can be 11 or 01 for erase or program disturb respectively. If the read value is 11 or 01, then the corresponding fault is detected. Thus, all of the WED & BED faults of type 3 and WPD & BPD faults of type 2 on $!D1$ are successfully identified. This is depicted in Table 6.8 using march element $D1 : \uparrow w(01); !D1 : \{\uparrow i, r(11), r(01); \uparrow j, r(11), r(01)\}$.
**Step 7:** All the cells on D2 are now programmed to state 01. This will activate all WED & BED faults of type 2 and WPD & BPD faults of type 1 on D1. After each of the D2 cells is programmed, the cells on D1 which are on the same bit line and word line as the programmed cell, are read. A read operation is performed on each cell on D1. If the read value is 10, then a WED or a BED fault of type 3 is identified. If the read value is 00, then a WPD or a BPD fault of type 2 is identified. Thus, all the WED & BED faults of type 2 and the WPD & BPD faults of type 1 on D1 can be successfully identified. This can be depicted in Table 6.9 using $D2 : \uparrow w(01); \ D1 : \{\uparrow_i r(10), r(00); \uparrow_j r(10), r(00)\}$.

Table 6.9: Diagonal FDX - Step 7

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>10</th>
<th>10</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
</tbody>
</table>

**Step 8:** All the non-diagonal cells are now programmed to state 01. This is done so that, the algorithm can continue with all cells in the array with the state of 01. This can be depicted in Table 6.10 using $!D1, !D2 : \uparrow w(01)$.

Table 6.10: Diagonal FDX - Step 8

<table>
<thead>
<tr>
<th></th>
<th>01</th>
<th>01</th>
<th>01</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
</tbody>
</table>

**Step 9:** All cells on D1 are programmed to state 00. This will activate all WED & BED faults of type 2 and WPD & BPD faults of type 1 on the non-diagonal elements ($!D1$). After each cell is programmed, all cells on $!D1$, on the same row and column as the cell being programmed are read.
Each read operation performed on each cell of $!D1$ can be a 10 or 00 for erase or program disturb respectively. If the read value is 10 or 00, then the corresponding fault is detected. Thus, all of the WED & BED faults of type 3 and WPD & BPD faults of type 2 on $!D1$ are successfully identified. This is depicted in Table 6.11 using $D1 : \uparrow \; w(00); \; !D1 : \{\uparrow_i \; r(10), r(00); \; \uparrow_j \; r(10), r(00)\}$.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

Table 6.11: Diagonal FDX - Step 9

**Step 10:** All the cells on $D2$ are now programmed to state 00. This will activate all WED & BED faults of type 1 on $D1$. After each of the $D2$ cells is programmed, the cells on $D1$ which are on the same bit line and word line as the programmed cell are read. A read operation is performed on each cell on $D1$. If the read value is 01, then a WED or a BED fault of type 1 is identified. Thus, all the WED & BED faults of type 1 on $D1$ can be successfully identified. This can be depicted in Table 6.12 using $D2 : \uparrow \; w(00); \; D1 : \{\uparrow_i \; r(10); \; \uparrow_j \; r(10)\}$.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

Table 6.12: Diagonal FDX - Step 10

**Step 11:** The flash array is erased, i.e., all cells in the array under test now contain "11". This can be depicted in Table 6.13. This not only initializes the entire array for the remainder of the algorithm, but also activates the over-erased faults in the array by $\uparrow \; w(11)$.
Step 12: All the cells on D2 are now programmed to state 00. This will activate all WED & BED faults of type 4 and WPD & BPD faults of type 3 on D1. After each of the D2 cells is programmed, the cells on D1 which are on the same bit line and word line as the programmed cell, are read. A read operation is performed on each cell on D1. If the read value is OE, then a WED or a BED fault of type 4 is identified. If the read value is 10, then a WPD or a BPD fault of type 3 is identified. Thus, all the WED & BED faults of type 4 and the WPD & BPD faults of type 3 on D1 can be successfully identified. This can be depicted in Table 6.14 using $D2 : \uparrow w(00); D1 : \{\uparrow_i r(OE), r(10); \uparrow_j r(OE), r(10)\}$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Step 13: All the non-diagonal cells are now programmed to state 00. This is done so that, the algorithm can continue with all cells in the array are currently at the programmed state of 00. This can be depicted in Table 6.15 using $!D1,!D2 : \uparrow w(00)$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Step 14: All cells on D1 are programmed to state 00. This step will activate WED and BED faults of type 1 on !D1. After each program to the D1 cells, a read operation on each cell of !D1 is
Table 6.15: Diagonal FDX - Step 13

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Flash $\uparrow w(11)$</td>
</tr>
<tr>
<td>12</td>
<td>$\uparrow r(OE)$</td>
</tr>
<tr>
<td>13</td>
<td>$D1 : \uparrow w(10); \uparrow {i r(11), r(01); j r(10), r(01)}$</td>
</tr>
<tr>
<td>14</td>
<td>$D1 : \uparrow w(11)$</td>
</tr>
</tbody>
</table>

performed. If the read value is 01 then the WED and BED faults of type 1 on !D1 are diagnosed. This is depicted in Table 6.16 and the march element is: $D1 : \uparrow w(00); \uparrow \{r(01)\}$.

Table 6.16: Diagonal FDX - Step 14

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>$\uparrow r(01)$</td>
</tr>
<tr>
<td>01</td>
<td>$D1 : \uparrow {i r(11), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>02</td>
<td>$D1 : \uparrow w(00)$</td>
</tr>
<tr>
<td>03</td>
<td>$D1 : \uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
</tbody>
</table>

Thus, all of the flash memory disturbances for all MLC states have been successfully activated and diagnosed. The diagonal - FDX algorithm described above can be summarized in Table 6.17.

Table 6.17: The Diagonal FDX March Algorithm

<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flash $\uparrow w(11)$</td>
</tr>
<tr>
<td>2</td>
<td>$\uparrow r(OE)$</td>
</tr>
<tr>
<td>3</td>
<td>$D1 : \uparrow w(10); \uparrow {i r(11), r(01); j r(10), r(01)}$</td>
</tr>
<tr>
<td>4</td>
<td>$D2 : \uparrow w(10); \uparrow {i r(11), r(01); j r(10), r(01)}$</td>
</tr>
<tr>
<td>5</td>
<td>$\uparrow r(01)$</td>
</tr>
<tr>
<td>6</td>
<td>$D1 : \uparrow w(00); \uparrow {i r(11), r(01); j r(10), r(01)}$</td>
</tr>
<tr>
<td>7</td>
<td>$D2 : \uparrow w(00); \uparrow {i r(11), r(01); j r(10), r(01)}$</td>
</tr>
<tr>
<td>8</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>9</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>10</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>11</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>12</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>13</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
<tr>
<td>14</td>
<td>$\uparrow {i r(10), r(00); j r(10), r(00)}$</td>
</tr>
</tbody>
</table>
6.4 Test Pattern Merging

The modified diagonal-FDX algorithm can be modified further to diagnose all the traditional and non-traditional faults discussed in Chapter 4. The diagnosis patterns for the non-traditional faults were listed in Table 6.2. (Note: The patterns in Table 6.2 are a combination of the broad classification of faults numbered 16 through 48 in Table 6.1.) Table 6.18 summarizes the final diagnosis FDX algorithm which will diagnose all of the faults listed in the fault model. The algorithm steps are listed with the modifications indicated in bold.

The following notation will be used in Table 6.18. Whenever an array element is programmed, in order to diagnose a bridging fault, we need to read the four non-diagonal elements around the diagonal cell being programmed. (All cells on the same word line are indicated by "i" and all cell along the same bit line are indicated by "j" as before.)

Consider Fig. 6.1. If the array cell (i,j) is programmed, then we have to check cell(i-1,j) & cell(i+1,j) along the same word line for a bit line bridging fault. Similarly, cell(i,j-1) & cell(i,j+1) along the same bit line needs to be checked for a word line bridging fault. Let us represent "i-1" by p, "i+1" by q, "j-1" by r and "j+1" by s. Thus for any write operation to cell (i,j), we need to test cells (p,j) & (q,j) for a bit line bridging fault with the programmed cell. Similarly, cells (i,r) & (i,s) need to be diagnosed for word line bridging faults with the programmed cell. For example, if a read r(00) is performed to cell(q,j) & cell(i,s) it is represented by r_q(00) & r_s(00) and so on.

Having described the notations that are used in Table 6.18, we shall consider an example
Figure 6.1: Bridging Fault Diagnosis.

to explain the notation very clearly. The reasoning to diagnose all the faults listed in Table 6.19 is discussed now. Let us consider step 4 in the algorithm as a suitable example.

Step 4: \( D_2 : \uparrow w(10), r(11), r(01), r(10) \);

\[ D_1 : \{ \uparrow_i r(11), r(01), r_p(10), r_q(10); \uparrow_j r(11), r(01), r_r(10), r_s(10) \} \]

In this step of the algorithm, each cell along \( D_2 \) is first programmed by \( w(10) \). A read operation is performed immediately on the cell to read its state. Only one read operation is performed. However, if the read is one of three values namely 11, 01 or 10 then the corresponding faults will be diagnosed. If the read operation yields 11, then SA11 and NT\(_3\) faults along \( D_2 \) is diagnosed. If the read operation yields 01, then NT\(_4\) fault along \( D_2 \) is diagnosed. If the read operation yields 10, then the BCF (for state 10) along \( D_2 \) is diagnosed.

As soon as the above operations are performed on each cell of \( D_2 \), cells along \( D_1 \) on the
same word line and bit line (as the cell on D2) are read. The \( \hat{r}_i \), \( r(11), r(01) \) operation diagnoses any word line disturbances. If the read operation yields 11, then a WED\(_2\) fault on D1 is diagnosed. If the read operation yields 01, then a WPD\(_2\) fault on D1 is diagnosed. While traversing along the word line (\( \hat{r}_i \)), the \( r_p(10) \) and \( r_q(10) \) operations detect any of the bit line bridging faults (along a word line).

Similarly, the \( \hat{r}_j \), \( r(11), r(01) \) operation diagnoses any bit line disturbances. If the read operation yields 11, then a BED\(_2\) fault on D1 is diagnosed. If the read operation yields 01, then a BPD\(_2\) fault on D1 is diagnosed. While traversing along the bit line (\( \hat{r}_j \)), the \( r_r(10) \) and \( r_s(10) \) operations detect any of the word line bridging faults (along a bit line).

This reasoning is followed for all the fault diagnostic steps of diagonal-FDX algorithm, summarized in Table 6.18.

Now, Table 6.19 summarizes all the faults detected by each step of the diagonal-FDX algorithm.

### 6.5 Test Complexity and Test Length Calculation

All the various march elements of the modified diagonal algorithm has been clearly illustrated in Table 6.18. Let us assume that the time taken to read a cell is the same for all possible Vt levels/states. Hence, we have \( r_{11} = r_{10} = r_{01} = r_{00} \). Similarly, let us assume that the time taken to program a cell to any of the programmed states is the same. Therefore, we have \( w_{10} = w_{01} = w_{00} \). Let the time for any read operation be \( r \), and the time for any write operation or program operation be \( p \). Let the time to erase the entire array be \( e \). Possessing the time taken for each of the basic flash operations, we can now begin to theoretically derive the complexity or test length
<table>
<thead>
<tr>
<th>Step</th>
<th>March Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flash $\uparrow w(11)$</td>
</tr>
<tr>
<td>2</td>
<td>$r(OE), r(10)$</td>
</tr>
<tr>
<td>3</td>
<td>$D1 : \uparrow w(10), r(11), r(01), r(10)$; $!D1 : {\uparrow_i r(OE), r(10), \uparrow_j r(OE), r(10)}$</td>
</tr>
<tr>
<td>4</td>
<td>$D2 : \uparrow w(10), r(11), r(01), r(10)$; $D1 : {\uparrow_i r(11), r(01), r_p(10), r_q(10) ; \uparrow_j r(11), r(01), r_r(10), r_s(10)}$</td>
</tr>
<tr>
<td>5</td>
<td>$!D1, !D2 : \uparrow w(10), r(11), r(01), r(10)$; $\uparrow_i r_p(10), r_q(10), \uparrow_j r_r(10), r_s(10)$</td>
</tr>
<tr>
<td>6</td>
<td>$D1 : \uparrow w(01), r(11), r(01), r(00)$; $!D1 : {\uparrow_i r(11), r(01); \uparrow_j r(11), r(01)}$</td>
</tr>
<tr>
<td>7</td>
<td>$D2 : \uparrow w(01), r(11), r(01), r(10), r(00)$; $D1 : {\uparrow_i r(10), r(00), r_p(01), r_q(01); \uparrow_j r(10), r(00), r_r(01), r_s(01)}$</td>
</tr>
<tr>
<td>8</td>
<td>$!D1, !D2 : \uparrow w(01), r(11), r(01), r(10), r(00)$; $\uparrow_i r_p(01), r_q(01), \uparrow_j r_r(01), r_s(01)$</td>
</tr>
<tr>
<td>9</td>
<td>$D1 : \uparrow w(00), r(11), r(01), r(00)$; $!D1 : {\uparrow_i r(10), r(00); \uparrow_j r(10), r(00)}$</td>
</tr>
<tr>
<td>10</td>
<td>$D2 : \uparrow w(00), r(11), r(01), r(00)$; $D1 : {\uparrow_i r(01), r_p(00), r_q(00); \uparrow_j r(01), r_r(00), r_s(00)}$</td>
</tr>
<tr>
<td>11a</td>
<td>Flash $\uparrow w(11)$</td>
</tr>
<tr>
<td>11b</td>
<td>$r(11), r(00)$</td>
</tr>
<tr>
<td>12</td>
<td>$D2 : {\uparrow w(00);$ $D1 : \uparrow_i r(OE), r(10); \uparrow_j r(OE), r(10)}$</td>
</tr>
<tr>
<td>13</td>
<td>$!D1, !D2 : \uparrow w(00), r(11), r(01), r(00)$; $\uparrow_i r_p(00), r_q(00), \uparrow_j r_r(00), r_s(00)$</td>
</tr>
<tr>
<td>14</td>
<td>$D1 : \uparrow w(00);$ $!D1 : \uparrow {r(01)}$</td>
</tr>
</tbody>
</table>

of the MLC diagonal algorithm.

**Calculation of Complexity:**

We need to calculate the complexity for a diagonal read operations due to programming a diagonal.

Since there are $\sqrt{N}$ number of cells along any diagonal, for every write operation to any of the D1 or D2 cells, we will need to check for the cells around each cell for a bridging fault. Each cell along
Table 6.19: Diagonal - FDX algorithm fault detection summary

<table>
<thead>
<tr>
<th>MLC Fault</th>
<th>Algorithm step detecting fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT₁, NT₂, OEF</td>
<td>2</td>
</tr>
<tr>
<td>WED₁, BED₁, WPD₁, BPD₁ on !D₁</td>
<td>3</td>
</tr>
<tr>
<td>WED₂, BED₂, WPD₂, BPD₂ on D₁</td>
<td>4</td>
</tr>
<tr>
<td>NT₃, NT₄, SA11, BCF &amp; BF (by w₁₀)</td>
<td>3, 4, 5</td>
</tr>
<tr>
<td>WED₂, BED₂, WPD₂, BPD₂ on !D₁</td>
<td>6</td>
</tr>
<tr>
<td>WED₃, BED₃, WPD₃, BPD₃ on D₁</td>
<td>7</td>
</tr>
<tr>
<td>NT₅, NT₆, NT₇, SA10, BCF &amp; BF (by w₀₁)</td>
<td>6, 7, 8</td>
</tr>
<tr>
<td>WED₃, BED₃, WPD₃, BPD₃ on !D₁</td>
<td>9</td>
</tr>
<tr>
<td>WED₄, BED₄ on D₁</td>
<td>10</td>
</tr>
<tr>
<td>SA₀₀, BCG-SG</td>
<td>11b</td>
</tr>
<tr>
<td>WED₁, BED₁, WPD₁, BPD₁ on D₁</td>
<td>12</td>
</tr>
<tr>
<td>NT₈, NT₉, SA₀₁, BCF &amp; BF (by w₀₀)</td>
<td>9, 10, 13</td>
</tr>
<tr>
<td>WED₄, BED₄ on !D₁</td>
<td>14</td>
</tr>
</tbody>
</table>

The diagonal can affect a total of 4 cells around it except the corner cells which only affect two cells each. Thus the number of reads to diagnose a bridging fault along a diagonal is

\[4r(\sqrt{N} - 2) + 2r(2) \text{ or } 4r(\sqrt{N} - 1) - - - - - - - (1)\]

Similarly, we need to calculate the number of reads to diagnose bridging faults complexity along !D₁ and !D₂. The total number of non-diagonal elements are \(N - 2\sqrt{N}\). For all the !D₁ and !D₂ cells there will be a total of four read operations in order to diagnose all of the bridging faults. However, the cells along the four borders will only need only three reads. Thus the total non-diagonal read operation complexity is

\[4r(N - 2\sqrt{N}) - r(\sqrt{N} - 2).4 \text{ or } (4N - 12\sqrt{N} + 8)r - - - - - - - (2)\]

Having calculated the two complexities, let us assume an example from Table 6.18. Consider Step four of diagonal-FDX. The write and read operation for the diagonal D₂ yields a complexity...

136
of $\sqrt{N}(p + r)$. The read operations to D1 yield $r(\sqrt{N}).2 + 4r(\sqrt{N} - 1)$. The first term is due to each of the D1 cells being read twice due to the programming of all cells along D2. The second term is the complexity derived to diagnose bridging faults along the diagonal from eq. 2 (derived above). Thus the total complexity for step 4 of the diagonal-FDX algorithm is

$$\sqrt{N}(p + r) + r(\sqrt{N}).2 + 4r(\sqrt{N} - 1)$$

This reasoning is carried out for each step of the algorithm and Table 6.20 lists each step of the algorithm and its corresponding complexity.

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity of Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$E$</td>
</tr>
<tr>
<td>2</td>
<td>$Nr$</td>
</tr>
<tr>
<td>3</td>
<td>$\sqrt{N}(p + r) + r(N - \sqrt{N}).2$</td>
</tr>
<tr>
<td>4</td>
<td>$\sqrt{N}(p + r) + r(\sqrt{N}).2 + 4r(\sqrt{N} - 1)$</td>
</tr>
<tr>
<td>5</td>
<td>$(N - 2\sqrt{N})(p + r) + 4Nr - 12\sqrt{N}r + 8r$</td>
</tr>
<tr>
<td>6</td>
<td>$\sqrt{N}(p + r) + r(N - \sqrt{N}).2$</td>
</tr>
<tr>
<td>7</td>
<td>$\sqrt{N}(p + r) + r(\sqrt{N}).2 + 4r(\sqrt{N} - 1)$</td>
</tr>
<tr>
<td>8</td>
<td>$(N - 2\sqrt{N})(p + r) + 4Nr - 12\sqrt{N}r + 8r$</td>
</tr>
<tr>
<td>9</td>
<td>$\sqrt{N}(p + r) + r(N - \sqrt{N}).2$</td>
</tr>
<tr>
<td>10</td>
<td>$\sqrt{N}(p + r) + r(\sqrt{N}).2 + 4r(\sqrt{N} - 1)$</td>
</tr>
<tr>
<td>11a</td>
<td>$E$</td>
</tr>
<tr>
<td>11b</td>
<td>$Nr$</td>
</tr>
<tr>
<td>12</td>
<td>$\sqrt{N}.p + r(\sqrt{N}).2$</td>
</tr>
<tr>
<td>13</td>
<td>$(N - 2\sqrt{N})(p + r) + 4Nr - 12\sqrt{N}r + 8r$</td>
</tr>
<tr>
<td>14</td>
<td>$\sqrt{N}.p + (N - \sqrt{N})r$</td>
</tr>
</tbody>
</table>

The total complexity of the modified diagonal-FDX algorithm is calculated by adding up the complexities of steps 1 through 14 and is shown in equation (3).

$$C = 2e + (3N + 2\sqrt{N})p + (24N - 23\sqrt{N} + 12)r - - - - - - - - (3)$$

Test Time Calculation:
Having calculated the number of erase, program and read cycles, the test time for the proposed MLC diagnosis algorithm can be determined. We use the following data for a 2MBbit flash array as in [24]:

- The standard erase time is 200 milli seconds
- The standard program time is 1.5 micro seconds
- The standard read time is 1.25 nano seconds

Substituting the above values to the test complexity equation (3), we have the total test time for the MLC diagonal-FDX algorithm as:

\[
T = 2(200ms) + (3N + 2\sqrt{N})(1.5us) + (24N - 23\sqrt{N} + 12)(1.25ns)
\]

\[
= (0.4 + 9.0042 + 0.05999) \text{ seconds}
\]

\[
= 9.4642 \text{ seconds}
\]

Finally, we have the algorithm test time \( T = 9.5 \text{ seconds} \) approximately.
Chapter 7

Conclusion

The major contributions of this thesis are MLC flash memory testing and diagnosis. Effective means to achieve this were elaborated.

In this research, we have developed a novel testing methodology that is used to effectively test a MLC array. This is the first work done in MLC flash testing. No recent work reported in literature deals with testing multi-level arrays using march algorithms. All faults which greatly impact MLC flash memories are detected by test patterns in an efficient way. The diagonal algorithm (Diagonal-FTX) has been introduced to mainly detect all types of flash memory disturbances, namely word line erase disturbance (WED), word line program disturbance (WPD), bit line erase disturbance (BED) and bit line program disturbance (BPD). Special emphasis was made on detecting these faults, since they have the most impact on multi-level cells. Through suitable modifications, the diagonal algorithm was appended to detect not only traditional faults such as stuck-at, bridging and broken connection faults, but also non-traditional faults whose effects cannot be neglected in multi-level flash memories.

The complexity of the test algorithm is \((2e) + (8N + \sqrt{N})r + (3N + 2\sqrt{N})p\). Since, the
test time is greatly influenced by the number of write operations to the flash cells, by keeping the number of program operations to a minimum, a very short test time was achieved. The test time for the diagonal algorithm for fault detection was only \textbf{9.42 seconds} which is acceptable. (As discussed before, the above time was for \(2 \times 10^6\) cells or \(4 \times 10^6\) bits.) Since we are testing something that doubles the number of bits when compared to SBC, this test time is reasonable. Thus, our goal of testing MLC flash arrays within low algorithm complexity and test time has been achieved.

We have also dealt with an efficient fault diagnosis methodology (diagonal-FDX) which completely identifies every type of traditional and non-traditional fault that occurs in a MLC array. This research also has never been carried out to diagnose a multi-level flash array using simple march algorithms. The diagonal-FTX algorithm (Chapter 5) was modified to uniquely distinguish between the various types of disturb, such as WED, WPD, BED and BPD. The diagonal-FDX algorithm also uniquely distinguished all of the traditional faults such as stuck-at, bridging and broken connection faults, and all non-traditional faults with similar fault patterns/behavior were also identified successfully.

The complexity of the diagnosis algorithm is \((2c) + (3N + 2\sqrt{N})p + (39N - 33\sqrt{N} + 12)r\).

Since, the test time is greatly influenced by the number of write operations to the flash cells, by keeping the number of program operations to a minimum, a very short diagnostic time was also achieved. The test time for the diagonal algorithm for fault detection was only \textbf{9.501 seconds} which is very acceptable for diagnosis, since it almost matches the test algorithm time. (As discussed before, the above diagnostic time was for \(2 \times 10^6\) cells or \(4 \times 10^6\) bits.) Thus, our goal of diagnosing MLC flash arrays within low algorithm complexity and time has been achieved.
The benefit of the above methodology is that a complex semiconductor technology, such as MLC flash, may be tested by a simple march algorithm. Thus, by implementing and developing the diagonal FTX and FDX march algorithms, any error-prone, multi-level flash memory array may be effectively tested and fault-diagnosed.
Bibliography


