Portland State University

PDXScholar

Dissertations and Theses Dissertations and Theses

10-26-2007

Ultra Reliable Computing Systems

Chong Ho Lee
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation

Lee, Chong Ho, "Ultra Reliable Computing Systems" (2007). *Dissertations and Theses.* Paper 6156.

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

ULTRA RELIABLE COMPUTING SYSTEMS

by

CHONG HO LEE

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
©2007

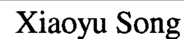
DISSERTATION APPROVAL

The abstract and dissertation of Chong Ho Lee for the Doctor of Philosophy in Electrical and Computer Engineering were presented October 26, 2007 and accepted by the dissertation committee and the doctoral program.

COMMITTEE APPROVALS:

Douglas V. Hall, Chair

Marek A. Perkowski

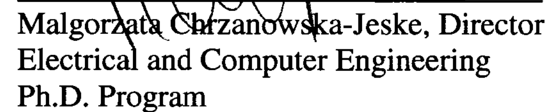


Xiaoyu Song

Dan Hammerstrom

Jong Sung Kim
Representative of the Office of Graduate Studies

DOCTORAL PROGRAM APPROVAL:



Malgorzata Chrzanowska-Jeske, Director
Electrical and Computer Engineering
Ph.D. Program

ABSTRACT

An abstract of the dissertation of Chong Ho Lee for the Doctor of Philosophy in Electrical and Computer Engineering presented October 26, 2007.

Title: Ultra Reliable Computing Systems

For high security and safety applications as well as general purpose applications, it is necessary to have ultra reliable computing systems. This dissertation describes our system of self-testable and self-repairable digital devices, especially, EPLDs (Electrically Programmable Logic Devices). In addition to significantly improving the reliability of digital systems, our self-healing and re-configurable system design with added repair capability can also provide higher yields, lower testing costs, and faster time-to-market for the semiconductor industry.

The digital system in our approach is composed of blocks, which realize combinational and sequential circuits using GALs (Generic Array Logic Devices). We describe three techniques for fault-locating and fault-repairing in these devices. The methodology we used for evaluation of these methods and a comparison with devices that have no self-repair capability was simulation of the self-repair algorithms. Our simulations show that the lifetime for a GAL-based EPLD that uses our multiple self-repairing methods is longer than the lifetime of a GAL-based EPLD that uses a single self-repair method or no self-repair method. Specifically, our work demonstrates that the lifetime of a GAL can be increased by adding extra columns in

the AND array of a GAL and extra output ORs in a GAL. It also gives information on how many extra columns and extra ORs a GAL needs and which self-repairing method should be used to guarantee a given lifetime. Thus, we can estimate an ideal point, where the maximum reliability can be reached with the minimum cost.

ACKNOWLEDGMENTS

As I finish up the doctoral program, I realized that this program is more than just a process for becoming a doctor in engineering. I feel that the philosophy I have come to understand in this lengthy program is what is really important in life, as the word philosophy in Ph. D., may indicate.

I am truly grateful for this day where I have finally reached the point of writing this acknowledgment after countless hours of research, development, and verification, but this dissertation could not have been completed without the support of many people. My utmost gratitude goes to the ones mentioned here.

First of all, I would like to thank my parents whose endless support from near and far allowed me to be the best student and engineer that I could have been, and get here where all my hard effort is finally being printed. The first ones I want to share this moment with are my mother and late father, Dr. Min Jae Lee, who would have been the happiest and the proudest of my achievement.

My family who patiently supported me through the long and arduous process. I regret not being able to spend more time with my little prince, Daniel H. Lee, who stayed strong and persevered with me, and even gave encouragements at times. There are no words that can express my appreciation for his support.

My advisor, Professor Douglas V. Hall, whose teachings gave me insights that are applicable to academic and practical world. He was a teacher in the class, an advisor in the program, and a father figure in life. He guided me with such integrity all the way to the end of the program. He taught me the true meaning of being an engineer, and was a great role model in life.

Professor Marek A. Perkowski who suggested the topic and provided many ideas and guidance. It truly was an honor and an unforgettable experience to learn from a professor of his stature and knowledge. He taught with much passion and strict rules, but always treated students warm-heartedly.

Professor Xiaoyu Song who did not hesitate to provide physical and mental support. He always helped me in difficult situations almost as a friend, and encouraged me to believe that I can complete the program.

It was an honor to have Professor Dan Hammerstrom and Professor Jong Sung Kim as the approval committee members. The points they made in the process will be a tremendous help to completing the thesis.

I will always be grateful for the Assistant Director of International Affairs, Ms. Christina Luther, for her kindness and dedication to guide me through the complicated international student rules and forms.

I would like to share this accomplishment with everyone who supported and waited for this day by my side, especially Mr. Tae Kun Woo and Mr. Hank Lee for their help in the simulation process, which was an integral part of the study.

Finally, I had many difficulties in finding reference materials directly related to this study, but I hope that this paper can be a useful reference to any related studies in the future.

Table of contents

ACKNOWLEDGMENTS .... **i**

List of Tables .... **ix**

List of Figures .... **xi**

1. Introduction **1**

1.1. MOTIVATIONS AND REAL PROBLEMS 2

1.1.1. Two-level Regular-Structured Programmable Logic Devices *2*

1.1.2. Memories *6*

1.2. DOMAIN OF THE WORK 10

1.2.1. General Research Objectives *11*

1.2.1.1. Purposes and Goals 11

1.2.1.2. Hypotheses 16

1.2.2. Contributions and Applications *17*

1.3. OUTLINE 19

2. Prerequisites for the Self-Repair Technology **21**

2.1. INTRODUCTION TO PLDS 21

2.1.1. What is PLD? *21*

2.1.2. Types of PLDs *22*

2.1.3. PAL *24*

2.1.4. GALs Structure *26*

2.2. PROGRAMMING TECHNOLOGY 30

2.2.1. Fusible Link *30*

2.2.2. E^2CMOS Programming Technology *31*

2.2.3. PLDs' Programming Procedure *33*

3. Previous Work on Self-Test and Self-Repair 36

3.1. SELF-HEALING MEMORIES 36

3.1.1. Introduction to Memories; SRAM, DRAM, and Flash Memory *37*

3.1.1.1. SRAM 37

3.1.1.2. DRAM 37

3.1.1.3. Flash Memory 38

3.1.2. Research on the BISTAR Embedded Memories: Trends and Products *39*

3.1.2.1. Why BIST 40

3.1.2.2. Redundancy and Repair 45

3.1.2.3. Self-Testing and Self-Repairing Algorithms 47

3.1.2.4. The STAR (Self-Test and Repair) SRAM Embedded Memory 49

4. Self-Testing and Self-Repairing EPLDs 51

4.1. DESIGN METHODOLOGY OF SELF-REPAIRABLE GALS 52

4.1.1. Fault Model and Assumptions 53

4.1.2. Design Architecture 57

4.1.3. Test Generation and Fault Diagnosis/Location 67

4.2. SELF-REPAIRING METHODOLOGIES 76

4.2.1. Column Replacement with Extra Columns 76

4.2.2. Column Re-Use with Extra Columns 80

4.2.2.1. Column-Column Re-Use with Extra Columns 81

4.2.2.2. Cell-Column Re-Use with Extra Columns 86

4.2.3. Integration of the Column Repair Methods 91

4.2.4. Replacement and Re-use with Extra OR-gates 93

5. Self-Testing and Self-Repairing Switching Circuit 95

5.1. HARDWARE DESIGN MECHANISM OF THE SELF-REPAIRABLE SWITCHING CIRCUIT 98

5.1.1. Fault Model and Assumptions *98*

5.1.2. Design Architecture *101*

5.1.3. Test Generation and Fault Diagnosis/Location *112*

5.2. SWITCHING CIRCUIT SELF-REPAIRING METHODOLOGIES 121

5.2.1. Line Replacement with Extra Lines *121*

6. New Hardware Prototype and Simulator for the Ultra Reliable Computing Systems 127

6.1. ANALYSIS AND SYNTHESIS OF THE HARDWIRED TEST AND THE DIAGNOSIS ALGORITHM 127

6.2. INTRODUCTION TO THE COMPUTER-BASED SIMULATOR 140

7. Evaluation and Analysis of the Simulation Results for the Ultra Reliable Computing Systems 164

7.1. ASSUMPTIONS AND FAILURE RATES 164

7.2. EVALUATION AND ANALYSIS OF THE SIMULATION RESULTS 167

7.3. HARDWARE OVERHEAD AND PERFORMANCE 180

7.3.1. Extra OR-Gate *187*

7.3.2. Extra Line on Switching Circuits *191*

7.3.3. Performance by Available OR-Gate *195*

7.3.4. FPGA and ASIC Design ... *199*

8. Conclusions and Future Works ... **206**

BIBLIOGRAPHY ... **213**

List of Tables

Table 7.1 Average Looping Times of Simulating the Replacement Methodology..... 168

Table 7.2 Average Looping Time of Simulating the Column-Column Re-Use Only. 170

Table 7.3 Average Looping Time of Simulating the Cell-Column Re-Use Only....... 171

Table 7.4 Average Looping Time of Simulating the Column-Column Re-Use and Replacement.. 174

Table 7.5 Average Looping Time of Simulating the Cell-Column Re-Use and Replacement.. 175

Table 7.6 Generic Components 181

Table 7.7 GAL Components... 182

Table 7.8 OLMC's OR-gate Combination Chart.. 183

Table 7.9 Area Overhead and Ratio.. 184

Table 7.10 Performance of a Self-Repairable GAL using Column Replacement Method 185

Table 7.11 Performance of a Self-Repairable GAL using Cell-Column Re-Use Method & Replacement Method 186

Table 7.12 Average Looping Time of Simulating the Cell-Column Re-Use and Replacement 188

Table 7.13 Performance Comparison of GALs with Extra OR-Gates and with Extra Columns 189

Table 7.14 Average Looping Time of SC with 1 and 2 Extra Lines 192

Table 7.15 SC Components 193

Table 7.16 SC Overhead Cell Count 194

Table 7.17 Performance of SC using Extra Line 194

Table 7.18 GAL Overhead –Total Cell Count 196

Table 7.19 GAL Overhead Ratio (Based on the Basic Prototype Proposed) 197

Table 7.20 SC Looping Time Variance on Number of Unused OR-Gates on GAL ... 198

Table 7.21 GAL Overhead Ratio (against a Generic Chipset, GAL) 199

Table 7.22 Comparing Data of FPGA and ASIC Simulation with Extra ORs 200

Table 7.23 Prototype of FPGA and ASIC's Comparison to a Generic System 202

Table 7.24 Unused Extra OR-Gate Efficiency 204

List of Figures

Figure 2.1 Simplified Notation for Input Lines of an AND Gate.................................22

Figure 2.2 Basic Structure of a PROM...23

Figure 2.3 Basic Structure of a PLA..23

Figure 2.4 Basic Structure of a PAL/GAL..24

Figure 2.5 Simplified Logic Diagram of a PAL 26

Figure 2.6 A Programmed Simple Logic Function in a GAL......................................27

Figure 2.7 Structure of the GAL16V8...29

Figure 2.8 (a) Non-Programmed State and (b) Programmed State in Basic Structure of an OR-Array...30

Figure 2.9 (a) Non-Programmed State and (b) Programmed State in Basic Structure of an AND-Array...31

Figure 2.10 E^2CMOS Cell...32

Figure 2.11 PLDs' Programming Procedure 35

Figure 3.1 The BIST Augmented Architecture..43

Figure 3.2. BIST Architecture with Column Repair..46

Figure 3.3. STAR Memory System 50

Figure 4.1 Example of a Logic Diagram Notation 57

Figure 4.2 Cellular Array of EPLDs/Memory in a System; controlled by a FLFRP (Fault-Locating/Fault-Repairing Processor) to repair faults in each EPLD/memory 58

Figure 4.3 Design Architecture for Self-Repairable GAL 60

Figure 4.4 Inner Structure of the Block 62

Figure 4.5 Structure of MAP and SAP Arrays 64

Figure 4.6 Structure of the NC Register 65

Figure 4.7 Generation of the Test Vector Set/Storing Scanning Results into the SAP. 70

Figure 4.8 Comparator Operation for Finding Faults on a Circuit without Faults 73

Figure 4.9 Fault Diagnosis/Location of an Example with Faults 75

Figure 4.10 A Column Replacement Method Example of Multiple Faults 78

Figure 4.11 An Example of the Column-Column Re-Use with Multiple Faults 83

Figure 4.12 An Example of the Cell-Column Re-Use with Multiple Faults 89

Figure 5.1 Design Architecture for the Ultra Reliable Computing System with Self-

Repairable GAL Module and Self-Repairable Switching Circuit Block 103

Figure 5.2 Inner Structure of the Switching Circuit Block 106

Figure 5.3 Structure of MSCI Array 107

Figure 5.4 Structure of the NSC Register 108

Figure 5.5 Structure of the NR Register 111

Figure 5.6 Structure of the MCIR Register 112

Figure 5.7 General Concept of an Example without Faults 115

Figure 5.8 Generation of the Test Vector Set/Storing Scanning Results into SCR7 .. 116

Figure 5.9 Comparator Operation for Finding Faults on an SC without Faults 118

Figure 5.10 Fault Diagnosis/Location of an Example with Faults 120

Figure 5.11 A Line Replacement Method Example of Multiple Faults without a Faulty OR 124

Figure 5.12 A Line Replacement Method Example with a Faulty OR 126

Figure 6.1 Overall Process 129

Figure 6.2 GAL and SC Generation 131

Figure 6.3 System Lifetime Generation 133

Figure 6.4 Fault Generation Simulation 135

Figure 6.5 System Diagnosis Process 137

Figure 6.6 Simulator Reset Process 139

Figure 6.7 Initial Screen of the Simulator 141

Figure 6.8 Creating GAL Options 141

Figure 6.9 Initial MAP and SAP 143

Figure 6.10 Added Faults 143

Figure 6.11 Simulation Configuration 144

Figure 6.12 Simulation Result 145

Figure 6.13 Initial Screen of the Replacement Method 146

Figure 6.14 Replacement Method Result 147

Figure 6.15 Initial Screen of Column Re-Use with Extra Columns Method 148

Figure 6.16 Column-Re-Use with Extra Columns Method Result 149

Figure 6.17 Initial Screen of Cell Re-Use Method 150

Figure 6.18 Cell Re-Use Method Result 152

Figure 6.19 Extra OR-gate Initial Screen 153

Figure 6.20 Extra OR-gate Result 154

Figure 6.21 Switching Circuit Initial Screen 155

Figure 6.22 SC 2 156

Figure 6.23 SC 3 157

Figure 6.24 SC Final 158

Figure 6.25 ASIC Initial Screen 159

Figure 6.26 ASIC 2 160

Figure 6.27 FPGA Initial 161

Figure 6.28 FPGA 2 162

Figure 6.29 Typical Simulation Result 163

Figure 7.1 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 5 and 0.05% Failure Rate 177

Figure 7.2 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 5 and 5.00% Failure Rate 177

Figure 7.3 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 10 and 5.00% Failure Rate 178

Figure 7.4 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 10 and 5.00% Failure Rate 178

Figure 7.5 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 20 and 5.00% Failure Rate 179

Figure 7.6 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 20 and 5.00% Failure Rate 179

Figure 7.7 Performance Ratio/Overhead Ratio of Increased OR-Gate vs. Columns . 190

Figure 7.8 Available OR-Gate and Performance Increase (One Line: No Extra Line & Two Lines: 1 Extra Line in each Pin-to-Pin Connection) 198

1. Introduction

Post-fabrication self-repair of digital circuits that have to work in adverse conditions such as increased cosmic radiation is not a new idea. Refer to the research of the "Hundred Year Spacecraft" in NASA [1] for instance. However, such circuits have not been built in VLSI and so far not much has been published on the subject, except in the area of memories. Although the self-repair problem is already important to current technologies, it will become a necessity when the next scientific revolution of "molecular engineering" or "nanotechnology" creates molecular computers [2,3,4,5] that will be implanted in the human body. In high reliability applications such as these, a circuit should not only test itself but also repair itself as quickly as possible.

There has been a very little research on diagnosis-based self-repair at the logic level. There are a few existing publications presenting various ideas for PLA (Programmable Logic Array) and PLA-based circuits [6,7,8,9,10,11]. However, to our knowledge, the circuits analyzed by other authors were not designed nor even simulated for reliability analysis. We want to systematically develop designs for self-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

repair methodologies on levels of systems, blocks, and logic/layout, with the minimum overhead.

1.1. Motivations and Real Problems

This section describes the reasons why we need to research self-testable and self-repairable digital devices according to regular-structured programmable logic device's realm as well as memories based on EEPROM (Electrically Erasable Programmable Read Only Memory) technology. Also introduced here are the realistic problems in today's world of technology.

1.1.1. Two-level Regular-Structured Programmable Logic Devices

- **GALs; AND-OR Based Arrays**

PLDs (Programmable Logic Devices) have become extremely popular in modern systems since they can be reprogrammed simply and inexpensively without making time-consuming PCB (Printed Circuit Board) changes as was required by earlier

designs that used random logic ICs [12,13,14]. The following citation evaluates the PLD market [25]:

Market for PLDs heats up: (from Semiconductor Business News, Nov. 1997)

- *Industry revenue in PLDs was expected to grow 22.7percent in 1997 to $2.26 billion, compared to $1.84 billion in 1996.*
- *In new market forecast update, the research company predicted world wide sales of user programming logic devices will increase by a 23.2 percent compound annual growth rate, reaching $5.24 billion in 2001.*

PLDs use various programming technologies such as fusible link, E^2CMOS, and others. PLDs allow easy implementation of a variety of logic circuits using EDA (Electronic Design Automation) tools. However, as the complexity of digital devices increases and the chips' geometry shrinks, the probability of developing faulty components (input/output lines, and product terms) also increases as components age [15].

Thus, in-circuit testing of PLDs, especially PLAs (Programmable Logic Array), has become of primary importance and has attracted the attention of large research

community [6,7,8,9,10,11]. In high-reliability applications, the circuit should test itself in real-time and repair itself as early as possible. This requires special logic built into a chip to make it easily testable and diagnosable as well as self-repairable by means of hardware re-programmability. If the high-reliability and quick in-field repair of a digital system is of the primary importance, and a system with only self-testing is not sufficient, then the addition of a self-repairing system would be highly desirable. As far as we know, nothing has been previously published on self-repair of PLDs such as GALs or similar devices.

Computers and other digital systems are subject to any number of faults caused by inadequate quality control during manufacturing, the wear and tear of normal operation, and other. Failures occur in CMOS due to manufacturing defects and due to wear out mechanisms whose effects accumulate over time [22]. External disturbances such as heat, radiation, and electrical and mechanical stress also increase the failure rate [22].

The failure rate and yield calculation of digital ICs has been surveyed in order to understand better the real problem; actually how and which faults are likely to occur in

the real world. Particularly, failure rate of EEPROM is concerned, since our first target model, GAL, uses EEPROM technology with E^2CMOS cells in a programmable AND array. Memories are particularly sensitive to aging as a function of cycling.

Now, we refer to the Early Failure Rate (PPM; Parts Per Million devices or DPM; Defects Per Million devices from 0 year to 1 year) and the Long Term Failure Rate (FIT; Failures In Time from 0 year to 10 years) from data provided by National Semiconductor Corporation. Early failure rates were calculated at 60% confidence using the Chi-Square distribution. Long term failure rates were calculated at 60% confidence using the Arrhenious equation at 0.7eV activation energy and derating the stress temperature to an application temperature of 55°C [23,24]. The data used to calculate the failure rates were obtained from high temperature operating life tests (OPL) performed on product qualification and long term audit (LTA) programs during the period from May 4, 1998 to May 4, 1999 [23]:

0.35μm CMOS: Early Failure Rate = 194 PPM, FITs = 4.66 %

0.50μm CMOS: Early Failure Rate = 141 PPM, FITs = 14.34 %

EEPROM: Early Failure Rate = 489 PPM, FITs = 5.07 %

The values of the PPM and FIT of EEPROM shown above suggest that it is reasonable to invest in a repair mechanism. These data are used to simulate our repair algorithm of the first project in order to get practical and useful information in Chapter 6 and Chapter 7.

1.1.2. Memories

Memory is a big part of any system, and is critical for building a system, today. According to San Jose-based Gartner Dataquest, by 2005, 70 % of the chip's surface will be memory [34]. While the embedded microprocessor and DSP (Digital Signal Processing) cores are essential in defining the system architecture, embedded memory is key to ensure design manufacturability at cost-effective levels. However, we note that memory is a magnet for defects during IC manufacturing because it has twice the defect densities that logic design has [34]. With design productivity doubling only every 39 months, a "design gap" has opened [35]. Design reuse and the availability of semiconductor IP (Intellectual Property) is being cited as the only way to close this gap so that silicon is not underutilized and products are not late to market [35].

The inspection result from [37] shows a fairly high failure rate of DRAM with 0.35μm design rules; the ratio of electrical failures caused by detected defects is 57%, and the results obtained on DRAMs (Dynamic Random Access Memories) can be accurately extrapolated to ASIC (Application Specific Integrated Circuit) products. Gaitonde [38] describes a methodology to accurately predict the probability of fatal faults and yield in array-based ASICs using the DEFAM (DEtect to FAult Mapper); the yield is around 65% under 95% transistor utilization in both the traditional yield estimation and the circuit-based yield estimation, and in 50% transistor utilization, yield based on traditional estimates is 70% and circuit-based yield is 75%. The yield simulator VLASIC (VLSI LAyout Simulation for Integrated Circuits) has been developed for determining functional yield in [39].

Memories are particularly sensitive to aging as a function of cycling. Failure analysis is important in the case of high performance systems, such as air and space assets, where usage of EEPROM [40] is high. The aging process is difficult to study, as it requires observation times of the order of the lifetime of the systems. It has been common practice to rely on the thermal stress as a way to accelerate the aging

process, thus enabling experimental measurements in a typical laboratory time scale. On average software changes in an F/A-18 type asset are implemented every 18-24 months, and during these changes, the EEPROM memory banks of the on-board computers are programmed without avionics removal through a nose wire harness.

During these changes when many memory failures are observed, the verification of performance of the on-board EEPROM devices is needed. The typical failure rates of the EEPROM with respect to the aging problem or wear out over time, under testing of large number of EEPROMs, are 4.35 errors per million device cycles with first error occurring after around 200,000 cycles. The problem of EEPROM in an abnormal condition such as a military temperature range is discussed in [41]. The gate oxide wear out failures and the difference between defect rates in predicted level and in actual level are discussed in [42] and [43], respectively.

When we consider only stand-alone memory, EEPROM, it is a useful device and stores nonvolatile data. However, it suffers from a serious deficiency that is absent in other nonvolatile storage devices. For instance, a magnetic disk has no limit on the number of erase/write cycles for a location, but the finite number of erase/write cycles

for any byte location in EEPROM often limits the disk's performance and utility. Today's technology limits the most commonly used EEPROMs to 100,000 to 1 million erase/write cycles because the erase function degrades the oxide barrier on the silicon and eventually leads to failure [36]. If one byte is erased and written per second in for example, a personal-safety monitor system or full date-and-time stamp system, the location exceeds its endurance rating in 100,000 to 1 million sec, or approximately 27.7 to 277 hours. Thus, the useful life of the equipment containing this device is to three and one half to 35 days at 8 hours per day of usage [36]. In other words, this device must have self-repairing capability so that the lifetime can be extended. This nonvolatile memory, especially Flash memory, is used in many real life applications; BIOS (Basic Input/Output System), digital cellular phones, digital cameras, LAN (Local Area Network), PC cards for notebook computers, digital set-top boxes, embedded controllers, and other devices.

Also, failure mode analysis of electronic devices is of utmost importance in the case of high performance systems, such as air and space systems that have a large amount of memories. It is therefore important to establish the expected lifetime and

to understand the failure modes of these components for procurement and maintenance purposes. With the trend and demand to develop high temperature tolerant electronics, there is an increasing need for repair methodology not only to assure performance reliability, but also to qualify commercial off-the-shelf components for more critical use. This could result in a significant acquisition and maintenance cost saving. If the repair mechanism is used for testing either during the manufacturing process or on outgoing products, the test cost will be decreased and the reliability of the outgoing products will be increased. In other words, it will minimize defect levels at the production test and thereby promote both the cost efficiency and the confidence in the reliability of the outgoing products. Therefore, the second main research target is developing ultra reliable computing systems for high security and safety applications as well as general purpose applications

1.2. Domain of the Work

This section states the objective of the study and describes the scope of the research.

1.2.1. General Research Objectives

The purposes, hypotheses, contributions, and applications of the dissertation are described in this section.

1.2.1.1. Purposes and Goals

The purpose of this research is to introduce ultra reliable computing systems based on nonvolatile EEPROM technology; the EPLD (Electrically Programmable Logic Devices), specially a GAL (Generic Array Logic), and develop a design methodology including hardware architecture, and the fault-detecting, fault-diagnosing, fault-locating, and fault-repairing circuitry that allows us detect, diagnose, locate, and repair automatically of all multiple stuck-at faults in nonvolatile cells and a switching circuit in a system that we propose in this dissertation.

Purpose of the first project

- Introduce the concept of the self-testable and self-repairable EPLDs for high security and safety applications.

- Prove that a self-repairable GAL will last longer in the field.
- Develop a design methodology (the fault-locating and fault-repairing architecture with electrically re-configurable GALs); that will allow us to detect, diagnose, and repair of all multiple stuck-at faults that might occur on E^2CMOS cells in programmable AND plane of a GAL.
- Develop a self-repairing methodology for EPLDs based on our 3 design architectures;
  - Column Replacement with extra columns; the respective faulty elements (E^2CMOS cells/cross-points) are replaced with the new ones (extra columns) by automatic reprogramming of the chip.
  - Column Re-Use with extra columns; the respective faulty elements (E^2CMOS cells/cross-points) are re-used for columns which have been already programmed if terms' personalities would fit the nature of the existing faults.

- OR Replacement with extra columns; the respective faulty ORs are replaced with the new ones (extra ORs) by automatic reprogramming of the chip.

Purpose of the second project

- Introduce the concept of the ultra reliable computing systems for high security and safety applications as well as general purpose applications.
- Prove that the self-healing and re-configurable system design with added repair capability will last longer in the field and can provide higher yields, lower testing costs, and faster time-to-market to the semiconductor industry.
- Introduce the concept of a self-testing and the self-repairing switching circuit based on Demultiplexer structure.
- Develop a design methodology (the fault-locating and fault-repairing architecture with electrically re-configurable GAL modules and self-testing and self-repairing switching circuits); that allows us to detect, diagnose, and repair of all multiple stuck-at faults that might occur on E^2CMOS cells in

programmable AND plane of a GAL, faulty ORs in a GAL, and faulty lines of a switching circuit in a system.

- Develop a self-repairing methodology for switching circuits based on our design architecture; line Replacement with extra lines; the respective faulty interconnection lines are replaced with the new ones (extra lines) by automatic reprogramming of the chip.
- Develop an evaluation methodology; Evaluate and analyze all self-repairing methods and combinations of self-repairing algorithms, Prove that the lifetime for a GAL-based EPLD that uses our self-repairing methods is longer than the lifetime of a GAL-based EPLD that uses a single self-repair method or no self-repair method;
    - Prove how many extra columns and extra ORs a GAL needs and which self-repairing method a GAL uses to guarantee a given lifetime.
    - Prove that our most advanced self-repair algorithm, the cell-column re-use with extra column and column replacement method, gives the best results in all the comparisons with our other algorithms.

- Demonstrate that the lifetime of a device can be increased by self-repair capability.
- Estimate an ideal point, where the maximum reliability can be reached with the minimum cost.

• Develop a computer based simulator for implementing our self-testing and self-repairing hardwired algorithm; Self-repair with redundancy and Self-repair with no redundancy

  - Develop a computer based simulator for implementing a micro-controller, FLFRP (Fault-Locating/Fault-Repairing Processor) that stores fault location and repair-related data.
  - Demonstrate that the lifetime of a GAL can be increased by adding extra columns in an AND array.
  - Demonstrate that the lifetime of a GAL can be increased by adding extra ORs in an AND array.
  - Demonstrate that the lifetime of a switching circuit can be increased by adding extra lines in a switching circuit.

- Introduce the basic concepts of modeling repairable systems as introductory-level knowledge
- Find how many extra columns a GAL needs to reach a lifetime goal in terms of simulation looping time until a GAL is not useful any more.

1.2.1.2. Hypotheses

- The following research hypotheses have been formulated;
- There exists a appropriate built-in self-repair circuitry for reliability, availability, and maintainability
- There exists a efficient self-testing and self-repairing algorithm with no redundancy
- There exists a computer based simulator to represent a repairable device accurately and efficiently.
- There exists a general tool to estimate cost-effect factors for self-testing and self-repairing hardware architecture.

1.2.2. Contributions and Applications

- The proposed research will have following contributions and applications;
- Contribute to high security and safety applications with self-repair capability such as aerospace systems, military systems, and medical instruments.
- Use the self-repairing technique in space, oceanic, and hazardous environments, where replacement of faulty devices cannot be done manually.
- Contribute to general purpose applications with self-repair capability in many real life applications; BIOS (Basic Input/Output System), digital cellular phones, digital cameras, LAN (Local Area Network), PC cards for notebook computers, digital set-top boxes, embedded controllers, and other devices.
- Contribute to the ultra reliable computing systems which will become a necessity for next scientific revolution such as nanotechnology.
- Close design gap, faster time-to-market, lower tests cost, and higher yields to the semiconductor industry.

- Contribute our simulator to simulate the self-repair hardware and to verify how self-repair hardwired algorithms improve the performance of a system in realistic environment.
- Apply the self-repairing method to a FPGA and be expandable for PLAs or EXOR PLAs for ESOP, GRM (General Reed-Muller), FPRM (Fixed Polarity Reed-Muller), and other AND/EXOR canonical forms and AND/EXOR multi-level circuits.
- Applicable to FPGA (Field Programmable Gate Array) design or ASIC (Application-Specific Integrated Circuit) design to shrink the design gap more quickly and affordably by adding redundancy.
- Generalize the stochastic reliable device model and apply it to estimation of cost-effect values.

1.3. Outline

The general statements of the problems are described in Chapter 1. Next chapter introduces several types of PLDs and GALs structure as well as PLDs' programming technology as a prerequisite for our project in this dissertation.

In Chapter 3, the most advanced and related work on memory is described. The literature for memories is reviewed. Especially, BISTAR embedded memories are discussed in industrial point of view with a real product in the market. The BIST and BISR memories from current papers are also examined and evaluated for our project.

In Chapter 4, fault models in the literature are reviewed. Our first project, "self-repairable EPLDs", is explained in detail according to several purposes and contributions; what is the digital system in our approach, and how are the multiple stuck-at faults detected, diagnosed, located, and repaired using either redundancy or no redundancy in a GAL. Our fault model, cross-point stuck-at faults in an E^2CMOS cell of a GAL, and assumptions also are in Chapter 4.

In Chapter 5, the main project is stated; the self-healing and re-configurable system design with added repair capability is described for the ultra reliable

computing systems. For high security and safety applications as well as general purpose applications, a hardware prototype based on EEPROM (Electrically Erasable Programmable Read Only Memory) technology is also introduced. It is deployed to design the general fault-repair circuitry for EEPROM-based digital devices.

In Chapter 6, our computer based simulator is introduced, and our hardwired repairing algorithms are simulated.

In Chapter 7, we analyze and evaluate the simulation results; it demonstrates that the lifetime of a GAL can be increased by adding extra columns in an AND array of a GAL and extra ORs in a GAL, and also gives information on how many extra columns and extra ORs a GAL needs and which self-repairing method a GAL uses to guarantee a given lifetime. Hardware overhead and performance are also discussed in terms of the ratio of efficiency versus cost factor. Thus, we can estimate an ideal point, where the maximum reliability can be reached with the minimum cost.

In Chapter 8, we conclude all the works in this dissertation and discuss potential future work and trends of the semiconductor industry.

2. Prerequisites for the Self-Repair Technology

This chapter introduces PLDs, especially PAL and GAL, the E^2CMOS programming technology, and the PLDs' programming procedure.

2.1. Introduction to PLDs

The background of PLDs is briefly explained, and the PAL/GAL, which is our model of this first project, is described in more detail. The E^2CMOS programming technology and PLDs' programming procedure are also shown concisely in this sub section.

2.1.1. What is PLD?

A PLD (Programmable Logic Device) is normally composed of a specific number of input lines connected through a fixed or programmable array to a set of AND gates, which are in turn connected to a fixed or programmable array of OR gates [14]. The OR gates provide the output signals from the logic array.

Note that a simple array will be used to graphically describe complex PLD structures: we will use special notation shown below, because a typical PLD has many inputs, outputs, and product terms. The modified AND input lines, likewise OR input lines, are also shown to simplify many input lines of an AND gate as in Figure 2.1.

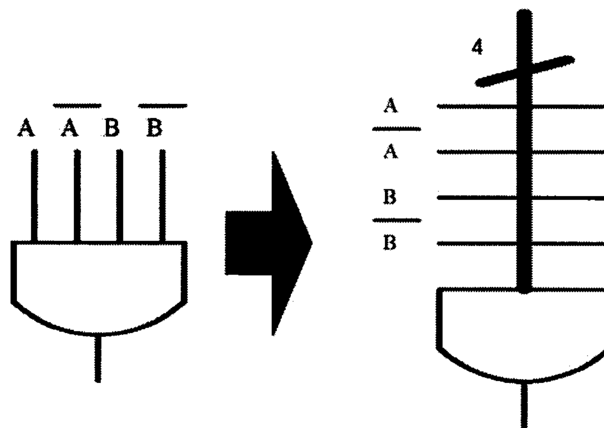


Figure 2.1 Simplified Notation for Input Lines of an AND Gate

2.1.2. Types of PLDs

There are generally four types of PLDs; PROM (Programmable Read Only Memory), PLA (Programmable Logic Array), PAL (Programmable Array Logic), and GAL (Generic Array Logic) [14,44,45].

The PROM has a fixed AND array and a programmable OR array. It is usually used as memory, and a logic diagram of a PROM is shown in Figure 2.2.

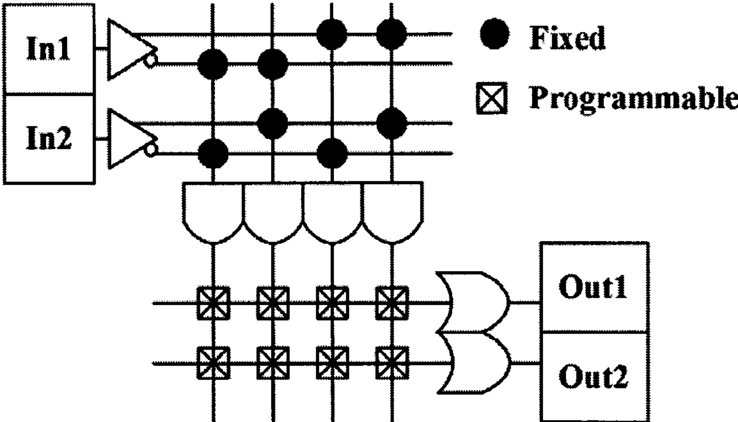


Figure 2.2 Basic Structure of a PROM

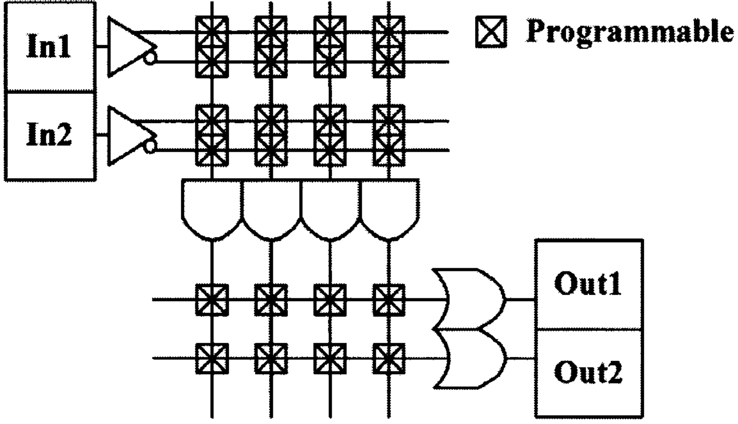


Figure 2.3 Basic Structure of a PLA

Figure 2.3 shows the basic structure of a PLA. The PLA has a programmable AND array and a programmable OR array. It is invented to improve constraints of a fixed AND array in a PROM, and can be programmed by users rather than manufacturers. It is also called to FPLA (Field Programmable Logic Array).

The basic structure of a PAL and a GAL is shown in Figure 2.4.

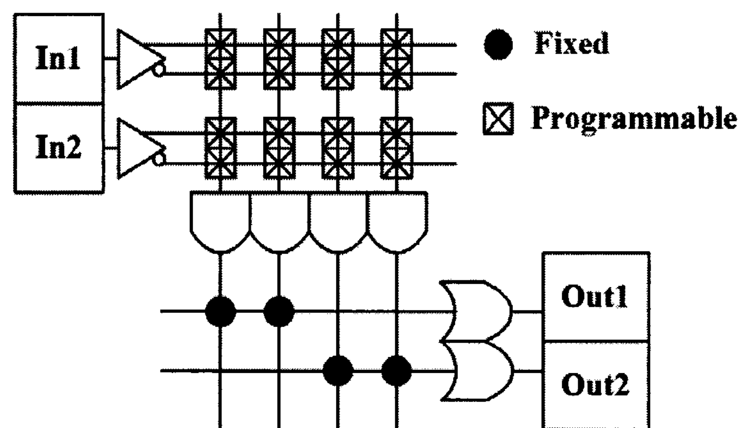


Figure 2.4 Basic Structure of a PAL/GAL

2.1.3. PAL

The Programmable Array Logic (PAL) device is a special case of the PLA. The PAL is one of today's most commonly used types of PLDs [12,14,16]. It has a fixed OR array and a programmable AND array, and is the registered trademark of

Advanced Micro Devices, Inc. (AMD) in the late 1970's. The key innovation of the PAL is the use of fixed OR array and bi-directional input/output pins. The PAL16L8 is probably the today's most commonly used combinational PLD structure.

The PAL16L8 has 64 columns (product terms) and 32 rows (inputs), therefore there are 2048 (64x32) fusible links to be programmed in an AND array. Each of the 64 AND gates in the array has 32 inputs, accommodating 16 variables and their complements, and each of eight OR gates is associated with output pin in the PAL16L8. Each product term can be a function of any subset of the 16 inputs. A simplified logic diagram of a PAL is shown in Figure 2.5. The notation 'X' denotes that the cross-points between each input line and each AND gate is connected with a fusible link.

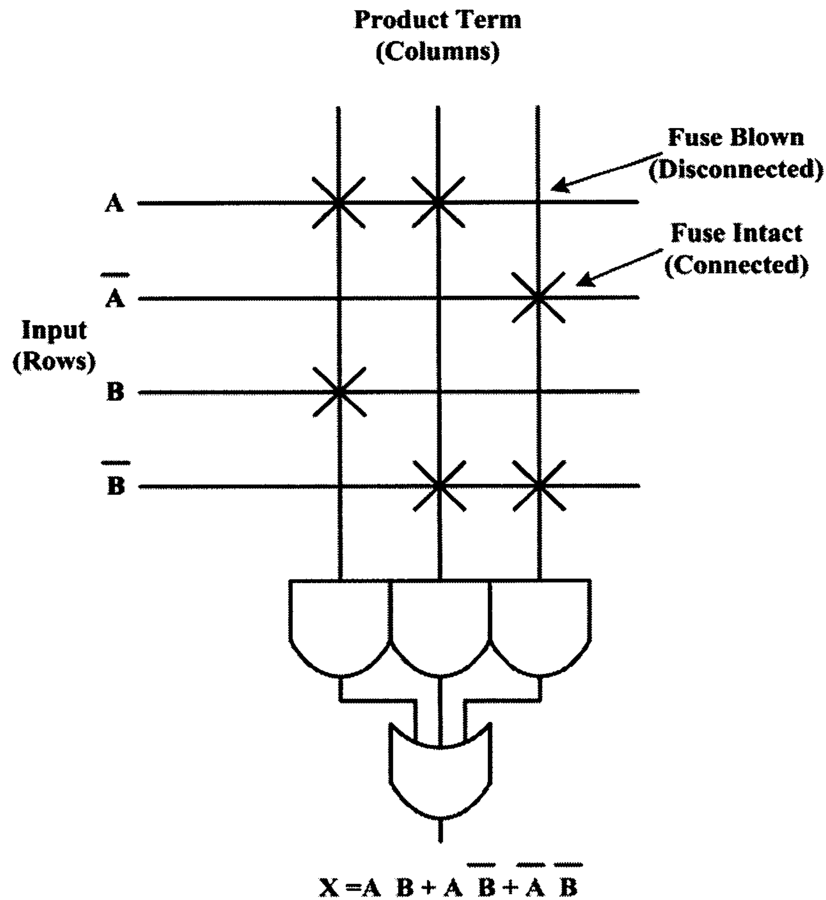


Figure 2.5 Simplified Logic Diagram of a PAL

2.1.4. GALs Structure

Lattice Semiconductor company introduced GAL devices such as the GAL16V8 in the mid 1980's. A GAL16V8 has a fixed OR array and a programmable AND array. The re-programmable array is essentially a grid of conductors forming rows and columns with an electrically erasable CMOS (E^2CMOS) cell at each cross-point,

rather than a fuse as in a PAL [16, 17, 18]. The programmed state of a simple logic function is schematically shown in Figure 2.6.

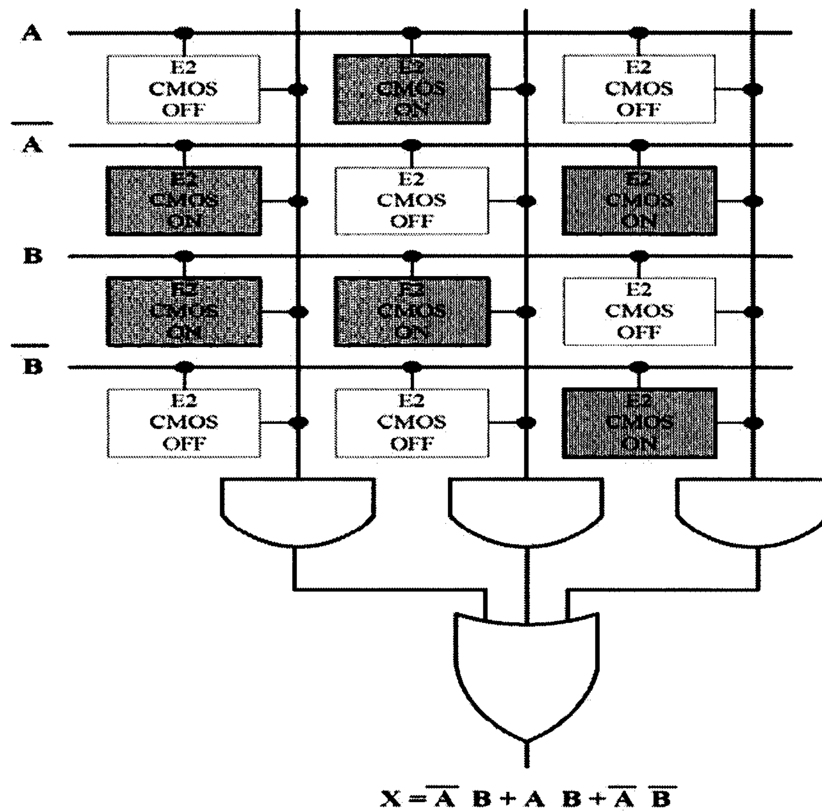


Figure 2.6 A Programmed Simple Logic Function in a GAL

The GAL16V8 provides 3.5ns maximum propagation delay, 250MHz clocking, full programmability, low power consumption, and 100 erase/write cycles [16]. Each column is connected to one input of an AND gate, and each row is connected to an

input variable or its complement. Any combination of input variables or complements can be applied to an AND gate to form any desired product term by programming each E^2CMOS cell to be either 'ON' or 'OFF'. A cell that is ON effectively connects its corresponding row and column, and a cell that is OFF disconnects the row and column. The cells can be electrically erased and reprogrammed. A GAL has the programmable AND array and OLMCs (Output Logic Macro Cells) that contain OR gates and flip-flops [18]. The E^2CMOS cell makes our self-repairing methodology possible. Thus, we choose as our model in this project the GAL16V8, which is a simple low density PLD.

For the more detailed description of pins and configurations the reader is referred to a GAL data book from Lattice Semiconductor, Inc. The sixteen primary inputs which include feedback paths from the OLMC are considered for this project, thus there are 32 input lines, which come with complements of each input variable, in the AND array. The eight primary outputs and eight product terms per an OLMC are considered in our model, thus there are 64 (8X8) product terms in this model. Therefore, the total number of E^2CMOS cells (cross-points) is 2048.

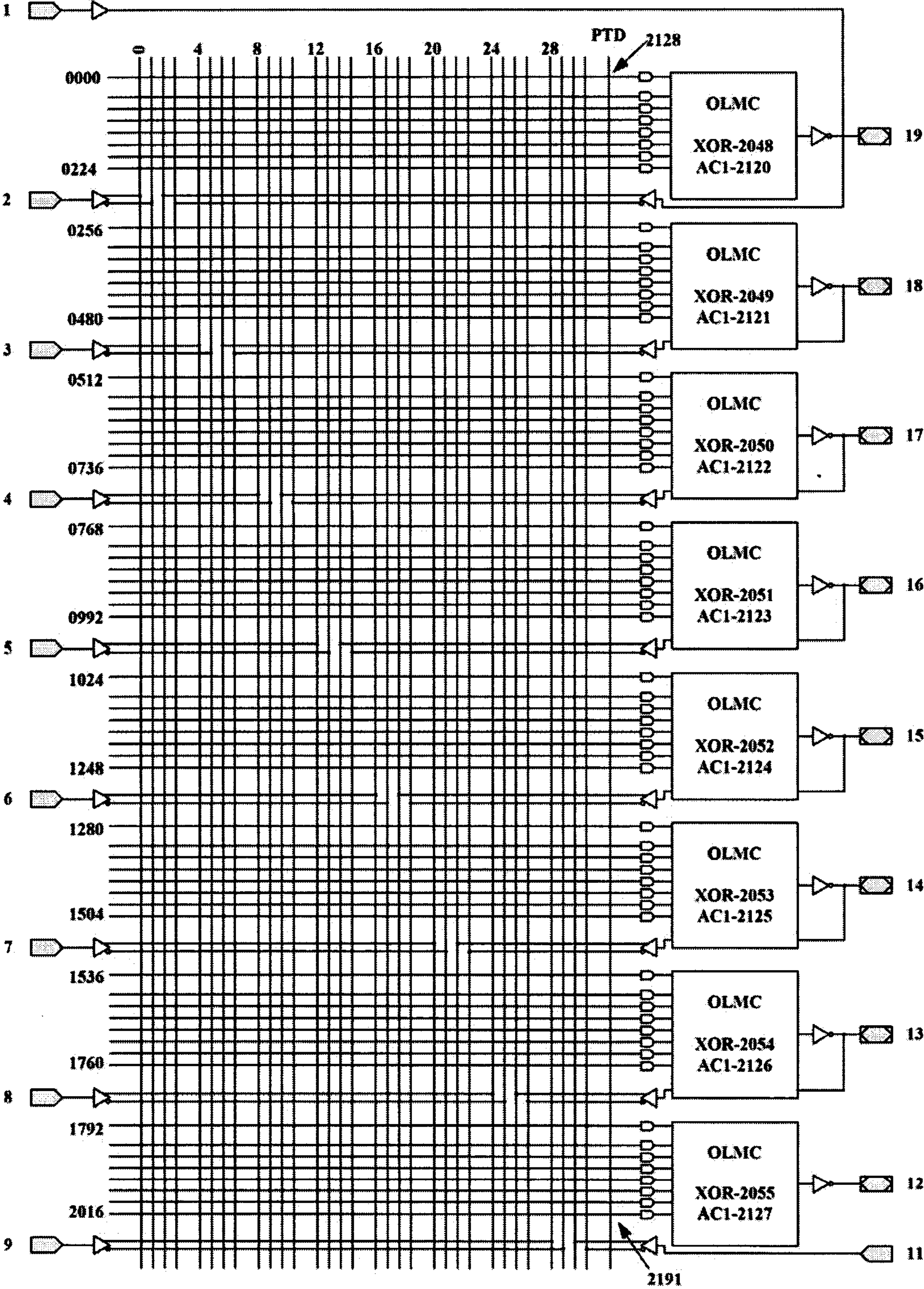


Figure 2.7 Structure of the GAL16V8

2.2. Programming Technology

The PLD has a generalized structure as an array for data inputs and each gate to program PLDs. The OR arrays can implement logical sums and the AND array can implement logical products. These arrays use the fusible link or the E^2CMOS cell to make connections between the data input lines and the gate input lines.

2.2.1. Fusible Link

Figure 2.8 and Figure 2.9 schematically show the basic structure and non-programmed/programmed states of which the PLDs are programmed in an OR array and an AND array with fusible links.

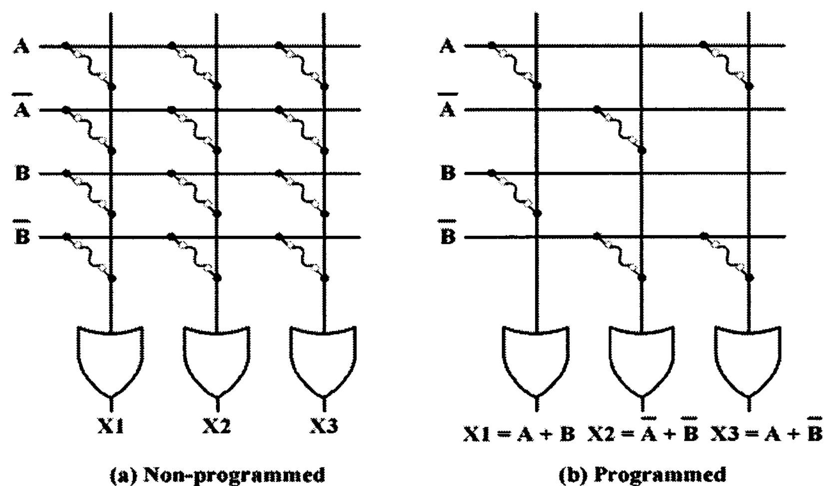


Figure 2.8 (a) Non-Programmed State and (b) Programmed State in Basic Structure of an OR-Array

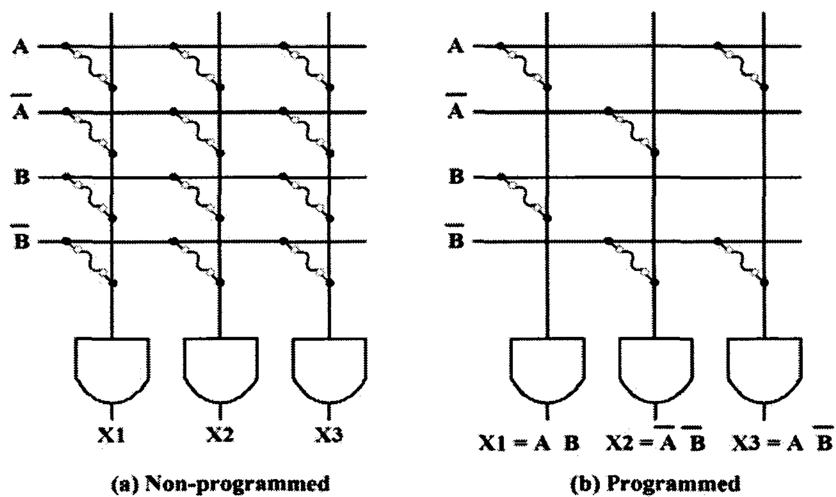


Figure 2.9 (a) Non-Programmed State and (b) Programmed State in Basic Structure of an AND-Array

2.2.2. E²CMOS Programming Technology

The E²CMOS technology is based on a combination of CMOS and NMOS technologies and is used in GALs [45]. The following figure shows an E²CMOS cell structure.

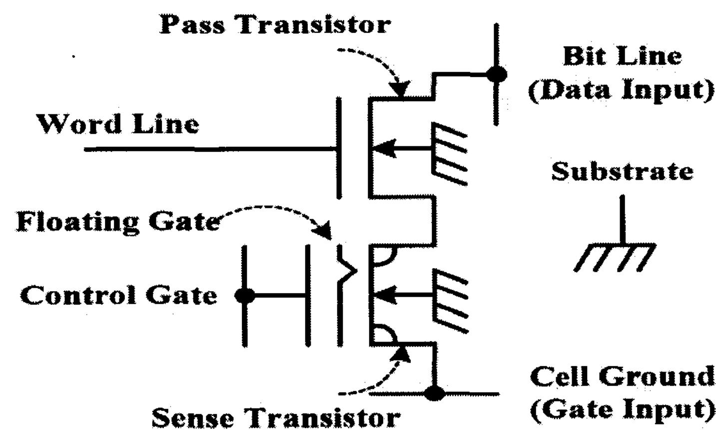


Figure 2.10 E^2CMOS Cell

The cell is programmed by applying a programming pulse to either the control gate or bit line of a cell that has been selected by a voltage on the word line. During the programming cycle, applying a voltage to the control gate to make the floating gate negative first erases the cell. This leaves the sense transistor in the OFF (storing a 1). A write pulse is applied to the bit line of a cell in which a 0 is to be stored. This will charge the floating gate to a point where the sense transistor is ON (storing a 0). The bit stored in the cell is read by sensing presence or absence of a small cell current in the bit line. When a 1 is stored, there is no cell current because the sense transistor is OFF. If a 0 is stored, there is a small cell current because the sense transistor is ON.

Once a bit is stored in a cell, it will remain indefinitely unless the cell is erased or a new bit is written into the cell. If the E^2CMOS is ON, data input can be transmitted to the gate input (AND gate input of a GAL). If the E^2CMOS is OFF, data input cannot be transmitted to the gate input (AND gate input of a GAL). The bit line (data input) and the cell ground (gate input) of Figure 2.10 are corresponding to the horizontal line and vertical line of Figure 2.6, respectively. The word line and the control gate are not shown in Figure 2.6.

2.2.3. PLDs' Programming Procedure

Generally, it needs a programming software (a logic compiler), a personal computer which mounts the software, and the software-driven programmer to program PLDs. A personal computer should satisfy requirements of software and programmer, like microprocessor type, memory amount, OS, and etc. ABEL, CUPL, OrCAD-PLD, LOGiC, PLDesigner, TANGO-PLD, and others can be used as the programming software. These software packages operate and synthesize the logic designs, transform these logic designs into intermediate files, produce JEDEC (Joint Electronic Device

Engineering Council) files, and simulate and debug these logic designs. The JEDEC file which has cross-point's programming information is also called Cell Map or Fuse Map. Boolean equation, truth table, state machine, schematic, timing waveform, hardware description, and etc. are used as a method of presenting logic designs to the programming software. Finally, according to the Fuse Map of JEDEC file, software-driven programmer implements PLDs after a PLD is put on programmer socket, called ZIF (Zero Insertion Force) socket.

To implement logic designs on PLDs, designer has to specify them with Boolean equations or other specifications. The produced input or source file is put on programming software, and debugged by syntax error checking. These are compiled and the compiler minimizes the logic. The logic designs are simulated with a set of test vectors. The programming software provides JEDEC file, and this file is downloaded into a programmer. The Fuse Map has cross-point information, in a GAL case, it will tell whether E^2CMOS cell is ON or OFF. The flowchart of this procedure is shown in Figure 10.

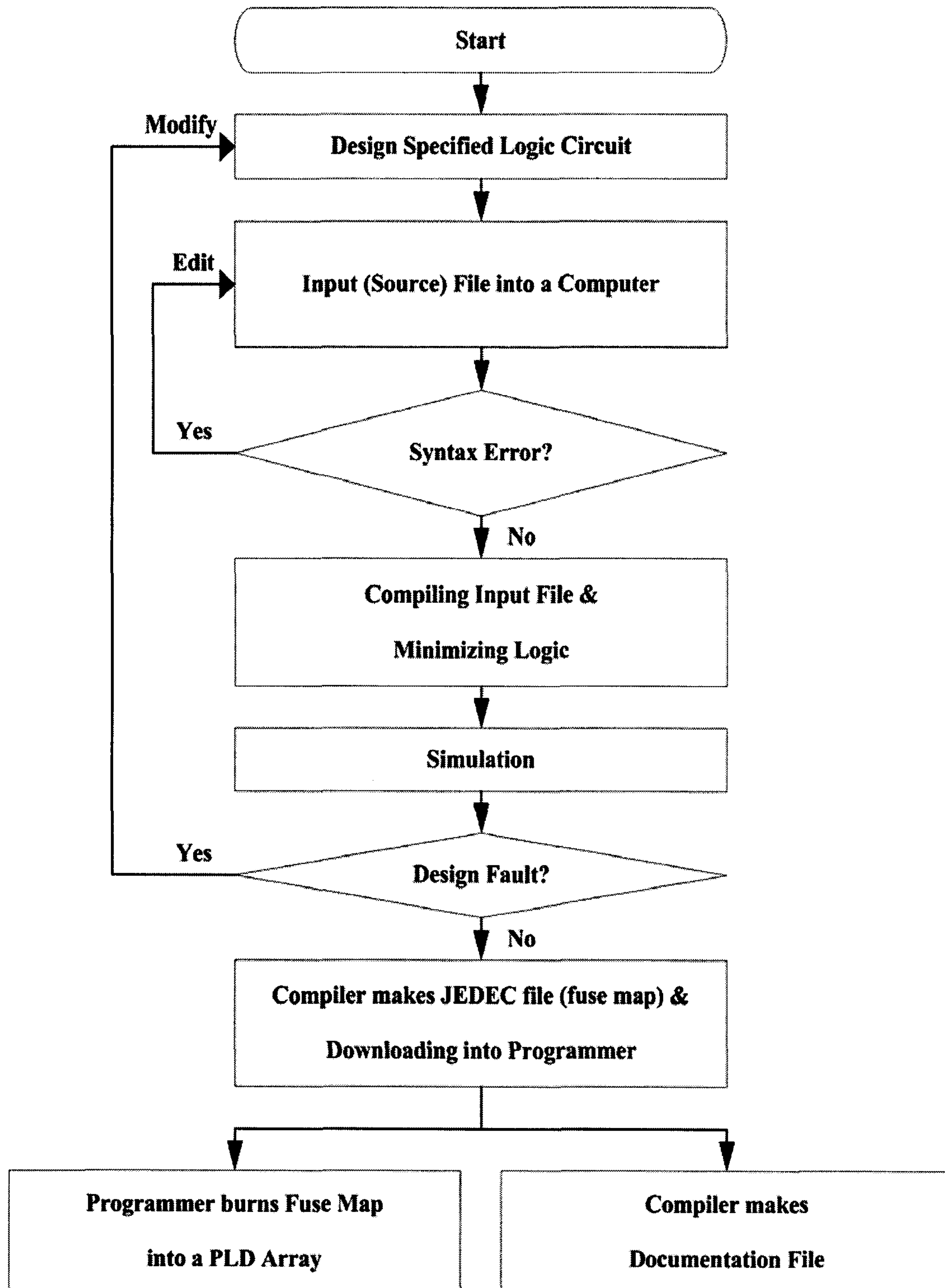


Figure 2.11 PLDs' Programming Procedure

3. Previous Work on Self-Test and Self-Repair

3.1. Self-Healing Memories

Most of the research activities on self-repair techniques were focused on FPGA [65,66,67,68,69]. BIST and BISR schemes have been proposed as potential solutions to the problem of repairing memories, mainly at the manufacturer level [70,71,72,73,74,75,76]. Recently, new techniques have been introduced to perform a memory repair through self-reconfiguration of the addressing space [77,78,79]. The paper [78] considers a column-only repair strategy to simplify the spare allocation procedure, and the paper [80] introduces on-line BIST RAM architecture based on cell-only redundant space allocation at the user level. In this section, several types of memories are described and the self-healing memories are reviewed from the literature. The self-testing and self-repairing algorithms are examined as well as the hardware architecture designs. The trend and product of the BISTAR embedded memories are introduced from the industry in this section.

3.1.1. Introduction to Memories; SRAM, DRAM, and Flash Memory

3.1.1.1. SRAM

SRAM is random access memory that retains data bits in its memory as long as power is being supplied. Unlike dynamic RAM, which stores bits in cells consisting of a capacitor and a transistor, SRAM does not have to be periodically refreshed. Static RAM provides faster access to data but is more expensive than DRAM. SRAM is used for a computer's cache memory and as part of the RAM digital-to-analog converter on a video card.

3.1.1.2. DRAM

Dynamic random access memory is the most common kind of RAM for personal computers and workstations. Memory is the network of electrically charged points in which a computer stores quickly accessible data in the form of 0s and 1s. Random access means that the PC processor can access any part of the memory or data storage space directly rather than having to proceed sequentially from some starting place.

DRAM is dynamic in that, unlike SRAM, it needs to have its storage cells refreshed or given a new electronic charge every few milliseconds. Static RAM does not need refreshing because it operates on the principle of moving current that is switched in one of two directions rather than a storage cell that holds a charge in place. DRAM stores each bit in a storage cell consisting of a capacitor and a transistor. Capacitors tend to lose their charge rather quickly; thus, the need for recharging.

3.1.1.3. Flash Memory

Flash memory (sometimes called "flash RAM") is a type of constantly powered nonvolatile memory that can be erased and reprogrammed in units of memory called blocks. It is a variation of EEPROM that, unlike flash memory, is erased and rewritten at the byte level, which is slower than flash memory updating. Flash memory is often used to hold control code such as the basic input/output system (BIOS) in a personal computer. When BIOS needs to be changed (rewritten), the flash memory can be written to in block (rather than byte) sizes, making it easy to update. Flash memory gets its name because the microchip is organized so that a section of memory cells are

erased in a single action or "flash." The erasure is caused by Fowler-Nordheim tunneling in which electrons pierce through a thin dielectric material to remove an electronic charge from a floating gate associated with each memory cell. Intel offers a form of flash memory that holds two bits (rather than one) in each memory cell, thus doubling the capacity of memory without a corresponding increase in price.

3.1.2. Research on the BISTAR Embedded Memories: Trends and Products

Several parallel processors which have self-repair property based on processor/switch level reconfiguration have been presented in literature and some of them have been built [51,52,53]. The research on memory faults has been published in which the faults were localized and ICs were repaired in the production process by laser trimming or other techniques. However, laser repair is becoming increasingly expensive and is requires dedicated expertise. Now, the fuse based hard repair turns into soft repair, namely Built-In Self-Repair (BISR), which includes the storage of repair data and controls the soft reconfiguration mechanism. Yervant Zorian said the

integrated BIST and self-repair on the chip is becoming more feasible in the growing numbers of ICs with embedded DRAM and system-level LSI integrated circuits [63]. Today's large embedded static/dynamic RAM and flash memories are cases in point [54]. A highly reconfigurable Built-In Self-Test, Diagnosis, and Repair (BISTDR) solution for embedded DRAMs has been deployed by Genesys Testware and is a part of Memory BIST core. Embedded memories are the most dense components within a system-on-chip, accounting for up to 90% of its real estate. Memories also the most sensitive to process defects [55]. The following two subsections are mainly described in [55,64].

3.1.2.1. Why BIST

The complexity of today's ICs demands that embedded memory testing be taken further than traditional pass/fail testing. As the geometries of ICs become increasingly concentrated, new techniques such as diagnostic testing and built-in self-repair must be implemented into the devices. Many embedded memories are designed with built-in redundancy, which provides spare rows and columns that can replace failing

locations. Redundancy enables the manufacturer to repair a number of otherwise defective devices to ensure maximum production yield [55].

These issues are being addressed by the use of built-in self-test (BIST). BIST is the methodology of choice for testing embedded memories within SoC. It offers a simple and low-cost means to test for failures of embedded memories without significantly impacting device performance.

While it has been used primarily for production pass/fail testing, BIST can be extended to provide the diagnostic data required for process monitoring and repair. Although the area overhead required by the BIST circuitry is increased, designing the diagnostic circuitry into the BIST provides many advantages in terms of time for both setup and test [55,56,57]. The overview of memory testing procedures and the BIST are found in [55,58,59,60,61,62,64].

- **BIST advantages**
  - Lower cost of test
  - Better fault coverage

- Possibly shorter test times
- Tests can be performed throughout the operational life of the chip

- **BIST disadvantages**

  - Silicon area overhead
  - Access time
  - Requires the use of extra pins
  - Correctness is not assured

- **BIST Circuitry**

In the basic BIST architecture, each memory is tested by a BIST block that supplies a series of patterns to the memory, usually march tests or checkerboard patterns, and then compares the outputs against a set of expected responses. Because the patterns are highly regular, the outputs from the memories can be compared directly to the reference data using a comparator. This ensures that an incorrect response from the memory will be immediately flagged as a test failure.

Two schemes have proven popular. The first uses the BIST circuitry to identify each failing location and then to serially scan out the fail data. Figure 3.1 illustrates

how this process is added to the basic BIST architecture. The BIST controller is augmented with additional circuitry and has an additional debug enable input (debugz) and scan output (scan_out).

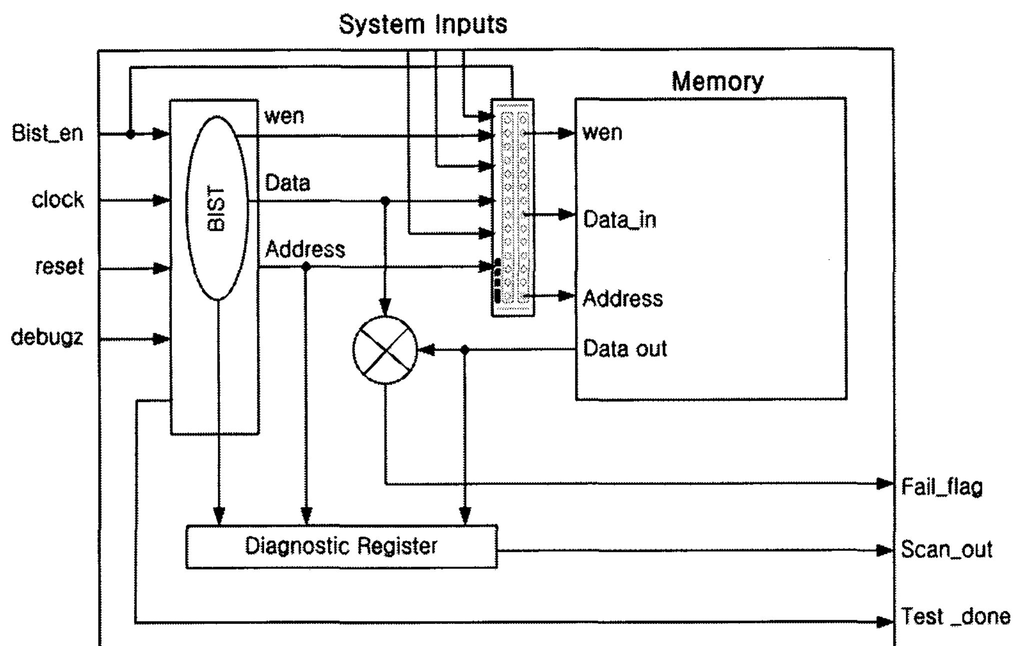


Figure 3.1 The BIST Augmented Architecture

The BIST controller operates in two modes: production BIST and diagnostic BIST. In production BIST, the BIST controller performs the default test and quickly identifies a passing or failing device. The failing devices then are run off-line using the

diagnostic BIST mode. In this mode, when the BIST controller detects a mismatch, it will suspend the application of the test, and the failing data will be serially scanned out of the controller through the scan_out port. The failing data scanned out consists of the data output from the memory, the address at which the failure occurred, and if required, the actual operation within the test algorithm being applied. This scheme requires only two additional ports per BIST controller and an interrupt handling mechanism in the tester software to detect the failure and capture the fail data. And because only the failing data is extracted from the device, this method ensures that additional application time is minimized. The total testing time depends on the number of failures in the test and may not be identical from device to device. For devices with few defects such as those manufactured in mature processes, this method is an effective means of extracting fail data.

A second mechanism makes use of a small diagnostic data bus and repeats the BIST operation many times to view a different slice of the memory output for each run [57]. Because the entire memory bit map is captured, a large requirement is placed on ATE (Automatic Test Equipment) pattern memory, usually resulting in a longer test

application time than the scan method. The application time, although longer, is uniform for each corresponding memory regardless of the defects present. A memory test option (MTO), if available on the ATE, can relieve the large memory requirements of the pattern memory [56]. The MTO generates patterns algorithmically and can be configured to create identical patterns to the BIST and be used as the reference source.

3.1.2.2. Redundancy and Repair

The use of redundancy and repair is not uncommon today, but the process of repair is tedious because it traditionally has required the use of external lasers. Built-In Self-Repair goes hand in hand with BIST as it simply takes advantage of the on chip processor to route around bad memory bits rather than using expensive and slow lasers to burn out bad memory rows. Typically, only a log of the address locations for each failure is required. For self-repair, only defective address locations must be logged. These are decoded on-chip to identify the failing rows or columns.

Figure 3.2 shows architecture for the BIST required to interface with self-repair (column repair) [55]. On each mismatch, the fail flag latches the address value onto

the decoder which identifies the failing column. At the end of the test, the number of defective columns is compared to those available within the redundancy.

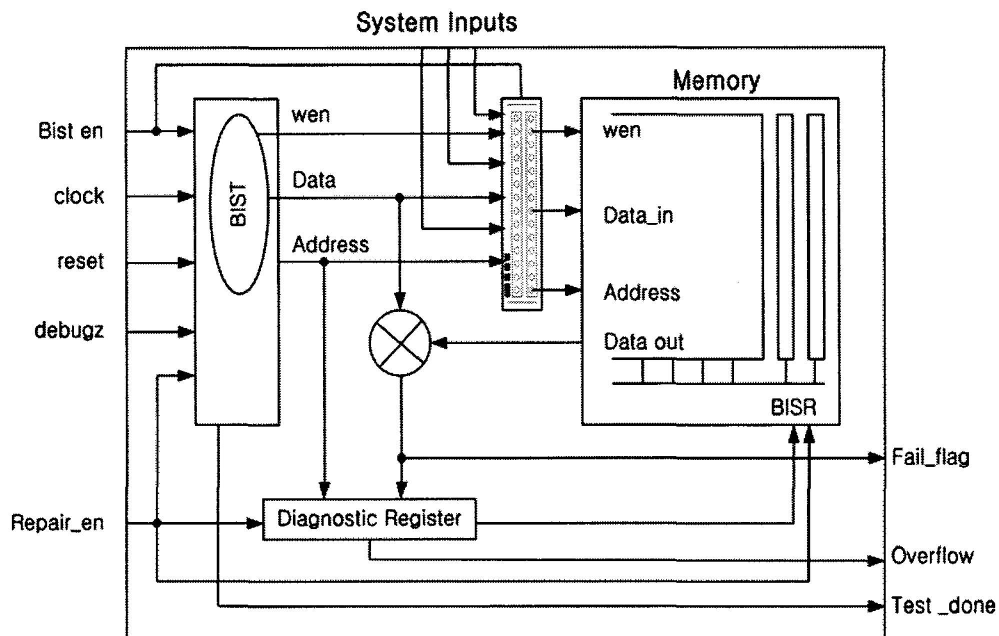


Figure 3.2. BIST Architecture with Column Repair

During the repair process, the repair circuitry is loaded with the values of the columns to be repaired. Once the repair is completed, the BIST is rerun to ensure that the repair produces a functional device.

3.1.2.3. Self-Testing and Self-Repairing Algorithms

The deterministic test algorithms for DRAM chips and Dipilp Bhavsar's algorithm are examined in this sub section.

A truly random design will surely be much smaller than a deterministic one. It would be necessary to build a LFSR (Linear Feedback Shift Register) to produce pseudo-random patterns, but this is much smaller than the counters needed for a deterministic test. It is also important to note that this would not be a truly random test but a repeatable test based on the initial seed. The standard methods of data testing is either to operate in parallel on multiple memory blocks and then check if they concur on a read, or use a set of known seeds and have a compacted form of the output stored as reference output values [64,81,82].

Deterministic memory test algorithms fall into two main categories [64,81,82];

- **March Tests:**
    - A March test is a finite sequence of tests to be preformed on every cell in the memory array before moving on to the next cell.

- All cells of the array are subjected to these same tests and traversed in order, either forwards or backwards.
- Runs in time ranging from 4n to 17n.
- Can cover all address faults, stuck-at faults, coupling faults (independent), inked coupling faults, transition faults, and transitions faults linked with coupling faults.

- **Neighborhood Pattern Sensitive Tests (NPST):**
  - A NPSF tests every cell of the memory in relation to its set of 5 or 9 neighboring cells (including the base cell).
  - Tests run as long as 195n.
  - Test covers the class of active, passive, and static neighborhood pattern sensitive faults.
  - These include stuck-at faults and all coupling and transitional faults between physically adjacent memory cells.

Bhavsar's algorithm [83];

- It generates and analyzes the required failure-bitmap information on the fly during self-test and then automatically repairs and verifies the repaired RAM arrays.
- It is concept of the condensed maximally repairable sparse failure array.
- It is divide-and-conquer strategy;
    - First, partition a large RAM array into small, identical segments
    - Second, repair each segment independently of the others.
    - Next, provide a spare row and a spare column for each segment.

3.1.2.4. The STAR (Self-Test and Repair) SRAM Embedded Memory

Virage Logic Corp. claimed development of the industry's first on-chip self-test and repair memory solution for the system-on-chip design. This STAR memory system does not need a expensive ATE and larger-repair tools. Currently, there are three versions of the STAR SRAM (single port 4Mbit and 512Kbit, and dual port

256Kbit). Its test and repair algorithm comes in the form of hardwired logic gates and will fix as much as 99% of the bad bits in an SRAM [78,84]. The test and repair architecture takes advantage of redundant rows and columns of memory. There are 4 test and repair logic components; foundry-specific BIST algorithm, built-in self-diagnostics, repair and redundancy allocation logic, and reconfiguring algorithm for the row and columns to be topologically efficient. The STAR memory system architecture is shown in Figure 3.3.

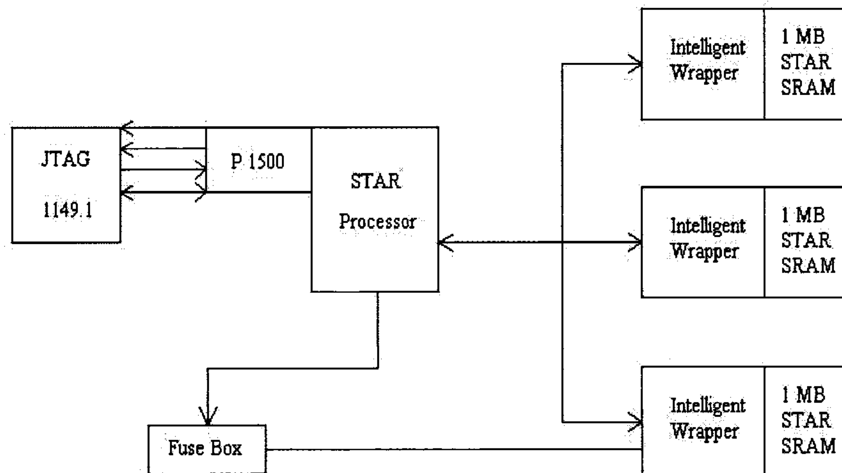


Figure 3.3. STAR Memory System

4. Self-Testing and Self-Repairing EPLDs

The concept of the self-testable and the self-repairable EPLDs (GAL-based structure) for high security and safety applications is described as our first project in this chapter. We developed the fault-locating and fault-repairing architecture with electrically re-configurable GALs to allow detection, diagnosis, and repair of all multiple stuck-at faults that might occur on E^2CMOS cells in a programmable AND plane of a GAL. Three self-repairing methods are presented based on our design architecture: a column replacement method with extra columns; and two column re-use methods with extra columns, one using the whole column's re-use (simply called column-column re-use) and the other using the only cell's (E^2CMOS cell's) re-use (simply called cell-column re-use). In first case the respective faulty elements (E^2CMOS cells, also called cross-points) are replaced with the new ones (extra columns) by automatic reprogramming of the chip. In the latter two respective faulty elements (E^2CMOS cells/cross-points) are re-used for whole columns or only cells which have been already programmed, if a terms' personalities would fit the nature of the existing faults. All three self-repairing methods use a FLFRP (Fault-

Locating/Fault-Repairing Processor), diagnosis/repair bus and the memory that stores fault location and repair-related data. In Chapter 7, we propose an evaluation methodology for this project. Our methodology is based on simulating the self-repair algorithms. It shows that the lifetime for a GAL-based EPLD that uses our self-repairing methods is longer than the lifetime of a GAL-based EPLD that uses a single self-repair method or no self-repair method. It demonstrates that the lifetime of a GAL can be increased by adding extra columns in an AND array of a GAL, and also gives information on how many extra columns a GAL needs to guarantee a given lifetime. Thus, we can estimate an ideal point, where the maximum reliability can be reached with the minimum cost.

4.1. Design Methodology of Self-Repairable GALs

In this section, we develop a design methodology for self-repairing an E^2CMOS cell in the programmable AND plane of a GAL, describe our fault model and assumptions, and develop universal test set for detecting and locating faults on each cross-point (E^2CMOS cell) [47, 48, 49].

4.1.1. Fault Model and Assumptions

Fault modeling is concerned with the systematic and precise representation of physical faults in a form suitable for simulation and test generation [19]. Such a representation usually involves the definition of abstract or logical faults that produce approximately the same erroneous behavior as the actual physical faults [19]. Good fault models should be straightforward, accurate, and easy to use. The most widely used fault model is the stuck-at fault model, which has been used for fault analysis and test generation in all types of logic circuits [19, 20]. While exact coverage figures are difficult to obtain, substantial empirical evidence shows that for general combinational or sequential logic circuits implemented with common MOS (Metal Oxide Semiconductor) or bipolar technologies, the stuck-at fault model provides good coverage of permanent physical faults [21, 46]. The most dominant failure modes in CMOS are shorts and opens [22]. In a switch-level representation of a CMOS circuit, MOS transistors are modeled as switches that conditionally transfer signals. The stuck-open fault model assumes that a faulty transistor never switches on (permanently disconnected), while a stuck-on fault model assumes that a faulty transistor never

switches off (permanently connected) [22]. The fault model for the memory cell array is presented in [21] and [46]; one or more cells are stuck-at 0 or 1. The functional defect (functional level fault model), which is very useful for describing a wide variety of faults, is introduced in [46]. Functional defects are those that will cause a functional failure or degradation in functional performance either immediately or in the short term [43].

The cross-point stuck-at faults, which are located in E^2CMOS cells, are considered as our fault model because E^2CMOS cells of an AND array make up a large percentage of a GAL and the E^2CMOS cells' array has the same structure as an EEPROM (Electrically Erasable Programmable Read Only Memory). Each E^2CMOS cell (cross-point) of a programmable AND array of a GAL, which is located between a vertical line (row, input line) and a horizontal line (column, product term), may be ON or OFF permanently, caused by an aging problem or by other factors referred to in Chapter 1. It is called the cross-point stuck-at-1 (simply, s-a-1) if the E^2CMOS cell of a particular cross-point, which should be programmed as OFF, is ON. If the E^2CMOS cell of a particular cross-point, which should be programmed as ON, is OFF, it is

called the cross-point stuck-at-0 (simply, s-a-0). Only these faults will be considered in this first project because we assume that there are no faults found in a GAL after the manufacturing process. Note that horizontal lines and vertical lines represent product terms and data inputs, respectively in a data book, but rows will be represented as data inputs and columns will be used as AND gates product terms in the rest of figures in the following sections.

The following assumptions will be used for the design, self-repair and evaluation methodologies. The GAL is initially fault free after manufacturing. The primary input/output fault does not exist in a GAL, and also every AND gate input line does not have faults. For GALs, the cross-point stuck-at faults are considered much more probable than the stuck-at faults in wires. The cross-point faults (s-a-0, s-a-1) as defined above are only taken into account in this first project. If the E^2CMOS cell is ON, binary data '1' will be stored in memory, and if the E^2CMOS cell is OFF, binary data '0' will be stored in memory.

In order to replace a faulty column, several extra columns are built into each OR gate (OLMC) in a GAL. There are at least two E^2CMOS cells programmed as ON for

a pair of input variables, like A and complement of A. Thus, an AND gate product term (column) can be discarded safely from the OR gate without changing an OR function even though the faults are appeared in certain cross-points of that column. When a fault occurs in a certain column of a particular OLMC, this faulty column can be replaced with an extra column only in that OLMC. When an E^2CMOS cell is OFF, it can be called 'programmed as OFF.' This cell disconnects a primary input from an AND gate input, and '1' will be on the AND gate input. It will be described with just an intersection between a row and a column in a logic diagram. When an E^2CMOS cell is ON, it can be also called 'programmed as ON'. This cell connects a primary input to an AND gate input. It is denoted by 'X' between rows and columns in a logic diagram.

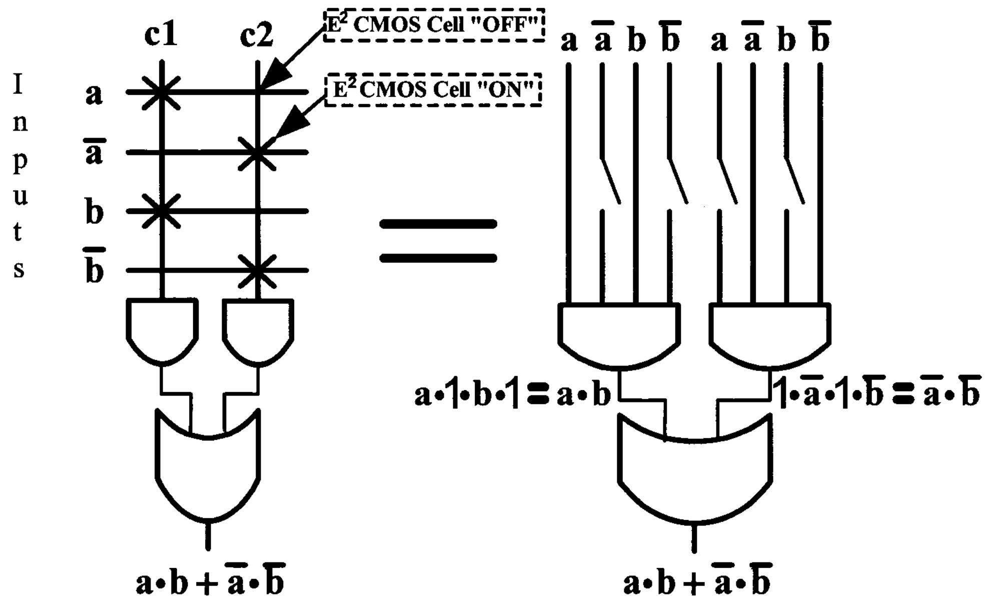


Figure 4.1 Example of a Logic Diagram Notation

4.1.2. Design Architecture

The complete digital system in our first approach is shown in Figure 4.2, and it is a network of blocks realized as separate integrated circuits. Each block is a two-level realization of a Boolean function, and is realized with a GAL. FSMs (Finite State Machines) are composed of Boolean Logic and registers located in OLMCs. This seems to be reasonable since, as the EPLD/FPGA (Field Programmable Gate Array)

technology advances, functions and machines of even greater sizes can be implemented in them. Reprogramming of a GAL serves to replace defective gates in

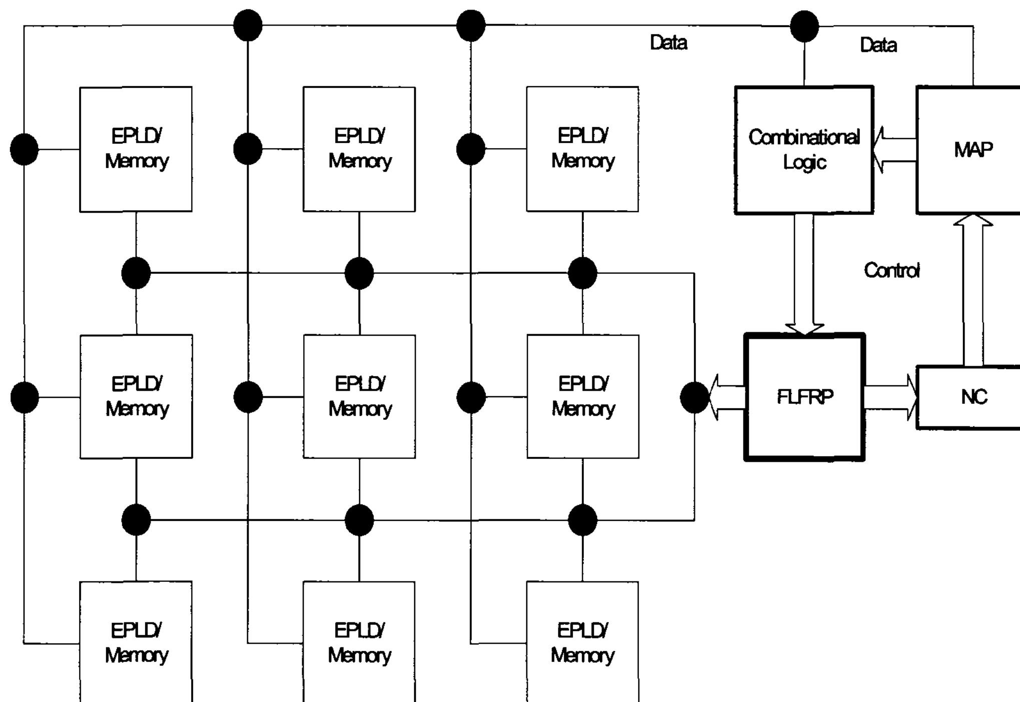


Figure 4.2 Cellular Array of EPLDs/Memory in a System; controlled by a FLFRP (Fault-Locating/Fault-Repairing Processor) to repair faults in each EPLD/memory

that GAL. The proposed method assumes that when the redundant area to create gates (i.e. the extra columns) in a block is exhausted and there are no more columns to replace a newly found defective gate, the controller will signal a global go/no-go signal. The block can be a single module (a chip, a board) or different modules. In the

first case, the module should be replaced, and in the second case, the modules and their connections should be tested independently. We assume that each block includes just one self-repairable GAL. Thus, each block has a programmable AND array with several extra columns and fixed OR (OLMC) plane.

At this point, we consider the problem of getting the maximum usefulness from each block in a system at the lowest level, when the system degrades over time with new faults arising in a block that has already been tested in the manufacturing process.

The following figure shows the general scheme of the design architecture for a self-repairable GAL. It uses extra columns in the AND plane, Scan Registers (SCR1 and SCR2), Diagnosis/Repair Bus, and a FLFRP which has a MAP (Memory AND Plane), a SAP (State AND Plane), a NC (Next Column) Register, three Scan Registers (SCR3, SCR4, and SCR5), a Comparator, a Central Fail-Safe Maintenance Controller, and Data Path/Addressing Unit. The FLFRP can be realized with a micro-controller.

In the normal operation mode, the primary input data are fed into an AND plane and the product terms of the AND plane are just transparent through the SCR2 into the OR plane. In the test mode, the SCR1 will have a test vector fed from the controller of

the FLFRP. The SCR2 then receives a scanning result corresponding to a test vector from the SCR1. This result is sent on the Diagnosis/Repair bus to the SAP of the FLFRP. The Diagnosis/Repair bus is assumed to include control, data, and address bus.

n = # of Inputs (Rows) (n = 32 in a GAL16V8) (Fixed)
k = # of Outputs (OLMCs) (k = 8 in a GAL16V8) (Fixed)
m = # of Products Terms (Columns) (m = 64 in a GAL16V8) with Extra Columns in a GAL (Variable)
y = # of Products Terms (Columns) (y = 8 in a GAL16V8) with Extra Columns in a OLMC (Variable)

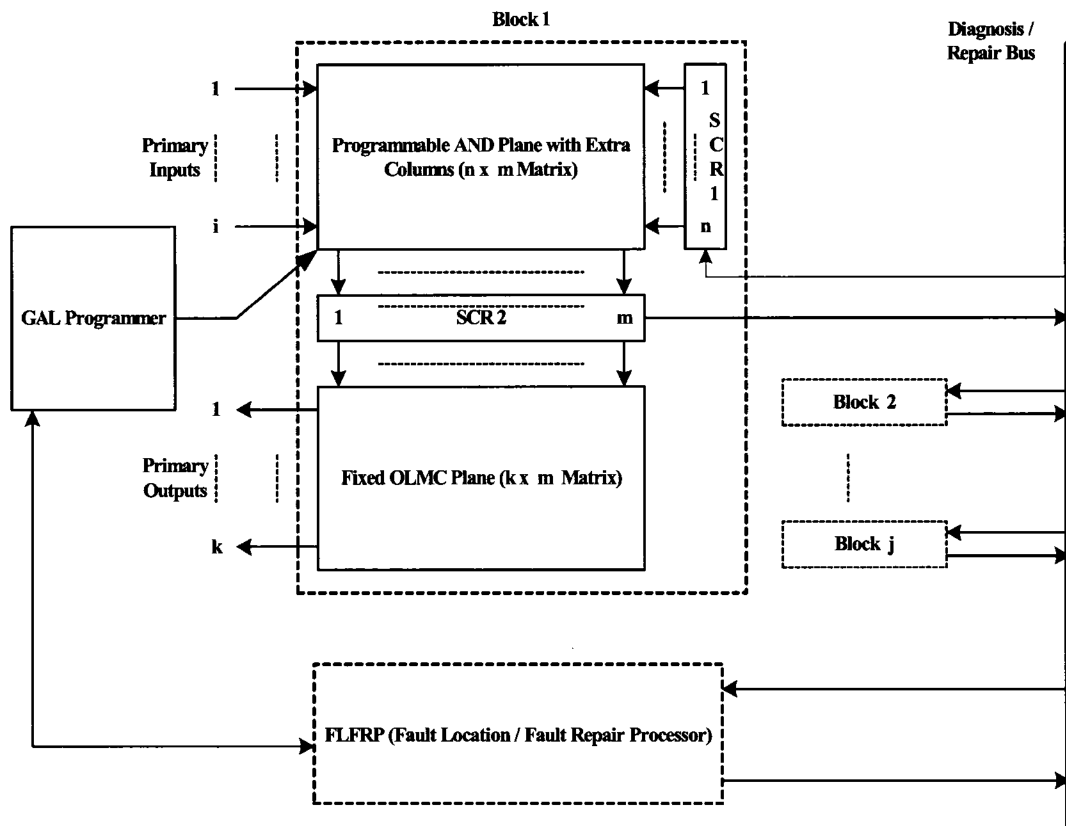


Figure 4.3 Design Architecture for Self-Repairable GAL

The bus is serial to decrease the pin-out of the chip. The 'i' denotes the number of primary inputs in an AND plane of a GAL. The 'n' denotes the total number of inputs in an AND plane of a GAL, including feedback and complement of the primary inputs. The 'k' denotes the number of primary outputs in an OR plane of a GAL. The 'm' denotes the number of product terms (columns) in a GAL. The 'y' denotes the number of product terms that each OR gate has in an OR (OLMC) plane, thus each OR gate has the same value of 'y'. The detailed inner structure of the block is shown in Figure 4.4.

There are 'n x m' cross-points, which are programmable with E^2CMOS cells in the AND plane. The 'n' is said to be the number of rows (inputs), and the 'm' is said to be the number of columns (product terms). The 'i (=16)' includes the feedback input from the OLMCs, thus the total number of rows (inputs) is 32 (= n). Each bit of the n-bit SCR1 corresponds to 1 row. The extra column means that there are several extra product terms in each OR gate of the OR plane of a GAL. We assume that there are the same number of extra columns in each OR gate, also called the OR group. If there are 8 OR gates in an OLMC plane of a GAL and 'y' number of columns, which includes

extra columns, in each OR gate of a fixed OR plane, the total number of columns is 'm' = '8' x 'y' as shown in Figure 4.13. It is the same as the length of the SCR2.

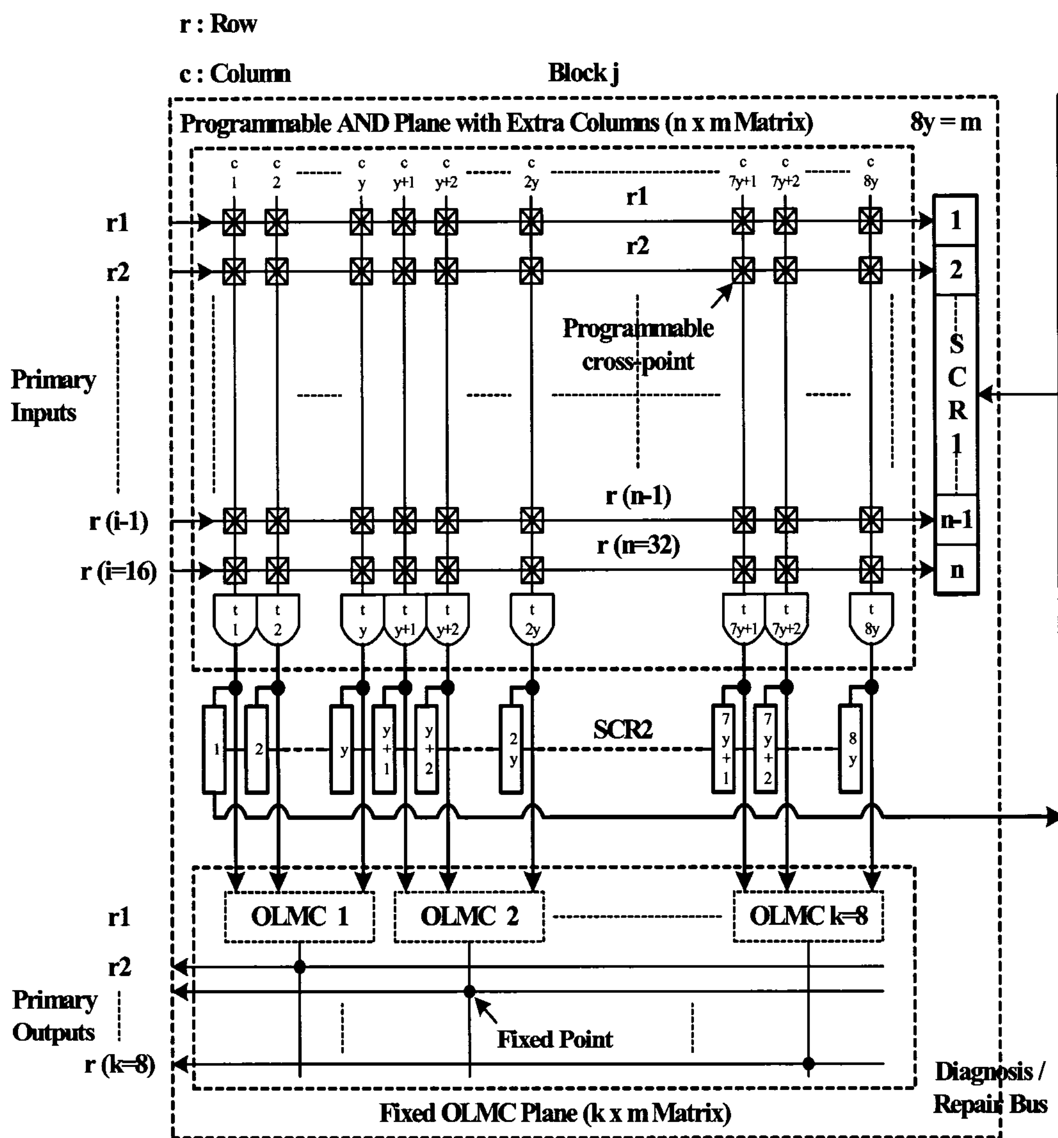


Figure 4.4 Inner Structure of the Block

The MAP stores original personalities of data being programmed into the AND array. It is the same as the information of the Fuse Map, thus this MAP is an 'n' x 'm' matrix. The SAP is exactly the same size array as the MAP. It has information about the actual state of each cross-point of an AND plane after testing. It will show the state of stuck-at faults on the cross-points. It includes both the actual state as well as the stuck-at fault information. It will be compared with the MAP to find and locate faults. SCR3 and SCR4 load the data column by column from the SAP and the MAP, respectively. The comparator compares these two data with each other. The comparison operation can be realized in few ways. We use the 'minus' operation. The SCR4 is subtracted from the SCR3 bit by bit. If the result of subtraction is '0', then the cross-point, corresponding to that bit is correct which means that it has no fault. If the result is '-1', then it has stuck-at-0 fault, and if the result is '1', then it has stuck-at-1 fault in the AND plane. An EXOR (Exclusive OR) gate could be used to detect the existence and location of faults, but it could not tell whether the fault occurred was stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1). Therefore, the proposed design uses the 'minus' operation circuit instead of the EXOR logic. The MAP and the NC

register data will use these compared results to reprogram the AND plane. Due to the inverters, the SCR5 has the complementary data from the SCR2. Since the SAP should have exact information of each E^2CMOS cell (cross-point) after testing, the MAP should be compared with the SAP that includes the inverted data from the SCR2. This will be described in more detail in sub section 4.1.3.

The structure of MAP and SAP is shown in Figure 4.5. If a certain bit has '1', the corresponding cross-point (E^2CMOS cell) is 'ON', and if a certain bit has '0', the corresponding cross-point (E^2CMOS cell) is 'OFF'.

Inputs	c 1	c 2	------------------	c 8y-1	c 8y
r1	1	0	------------------------------	1	1
r2	0	1	------------------------------	1	1
⋮	⋮	⋮	---------------+---------------	⋮	⋮
r (n - 1)	1	0	------------------------------	1	1
r (n = 32)	0	1	------------------------------	1	1

r : Row c : Column 1 : ON 0 : OFF 8y = m

Figure 4.5 Structure of MAP and SAP Arrays

The NC (Next Column) register informs about the status of each column in each OR group. As shown in Figure 4.6, each row represents an OR gate, and each column represents an input of an OR gate.

Primary Outputs	c 1	c 2	------------------------	c y-1	c y
or1	0	0	------------------------	1	1
or2	-2	0	------------------------	0	1
⋮	⋮	⋮	⋮	⋮	⋮
or (k - 1)	-1	0	------------------------	1	1
or (k = 8)	1	1	------------------------	1	1

0: Column being used **1: AvailbleColumn to replace or re-use**

-1: Unavailable Column to replace or re-use **-2: Available Column to replace only**

c: Column in a OR gate **or: OR Group**

Figure 4.6 Structure of the NC Register

There is 'y' number of columns in the NC since each OR gate has 'y' number of columns. There is 'k' number of ORs in this NC. In the GAL16V8, 'k' will be 8 since it has 8 OLMCs. The intersection of a row and a column being '0' in the NC means that the corresponding column of that OR is being used. If it is '1', then that column is useful for reprogramming a new logic function, replacing, or re-using a faulty column. It can be also considered as an available extra column to repair faulty columns. The '-1' means that the column corresponding to that location of the AND plane can not be used to repair a faulty column or to reprogram a new logic function. The '-2' means that if this column becomes faulty, it can only be replaced with an extra column, because it is assumed that the column was already repaired with the Column Re-Use method, and thus it can be only repaired with the Column Replacement method rather than with the Column Re-Use method. This self-repair method will be explained in section 4.2.

4.1.3. Test Generation and Fault Diagnosis/Location

All stuck-at faults, mentioned in section 4.1.1, on cross-points of a GAL can be determined by a pattern. We have a universal test set for detecting faults. We use well-known walking 0 technology.

This test vector set, which is provided by the FLFRP, can find whether the fault is an 's-a-0' or an 's-a-1'. It can determine the exact location in which the cross-point is faulty. All multiple faults of a column in an AND plane can be detected and located by this test set. The length of a test vector is n bits and the 'n' test vectors are needed to test if there are 'n' inputs because only one bit has '0' value in a test vector and it should affect each row. It detects the faults row by row in the AND plane. The FLFRP initializes SCR1 to '011...11', and this initial test vector will be shifted 'n-1' times from left to right. In this first test vector, the first bit is only '0' and others are '1's. The value of '0' in the first bit is fed into the first row (input), and it will affect the cross-point that is programmed (connected) as a value of ON. In other words, the result of this AND gate should be '0', but if the output of this AND gate is a '1', it will be a s-a-0 fault. On the other hand, the value of '0' fed into the disconnected cross-point does

not affect the output of the AND gate. However, if the output of this AND gate is a value of 0, it will be an s-a-1 fault. Thus, all faults are detected by this universal test set, and it is shown as follows.

❖ ***Test Vector Set (Test Pattern)***

1	***2***	***3***	***...***	***n-1***	***n (inputs)***
0	*1*	*1*	...	*1*	*1*
1	*0*	*1*	...	*1*	*1*
1	*1*	*0*	...	*1*	*1*
.	.	.	...	.	.
1	*1*	*1*	...	*0*	*1*
1	*1*	*1*	...	*1*	*0*

A simple example in which an AND plane has 16 cross-points (4 inputs times 4 columns) is shown in Figure 4.7. The procedure of generating the test set in SCR1, and next storing the scanning result from SCR2 into the SAP is illustrated as an animated sequence in this Figure. In Figure 4.7, the symbol 'X' denotes a programmed value of ON in the respective intersection of a row and a column.

This example assumes that there are no faults in the AND plane. The first test vector '0111' generated by the FLFRP is stored in the SCR1. This vector is fed into each input line as shown in Figure 4.7. The first bit of the SCR1, '0' cannot affect the

output of the first AND gate since the cross-point of c1 (the first column) and r1 (the first row/input) is OFF. Thus, the first bit of SCR2 has '1'. The second bit of SCR2 has '0' since the first bit '0' of the SCR1 is fed on the second AND gate according to ON of the cross-point of c2 and r1, and so on. The scanning result '1001' of the SCR2 is transmitted through the diagnosis/repair bus to the SCR5 after inverting this data as shown in Figure 4.7. This inverted data '0110' is stored on the first row of the SAP.

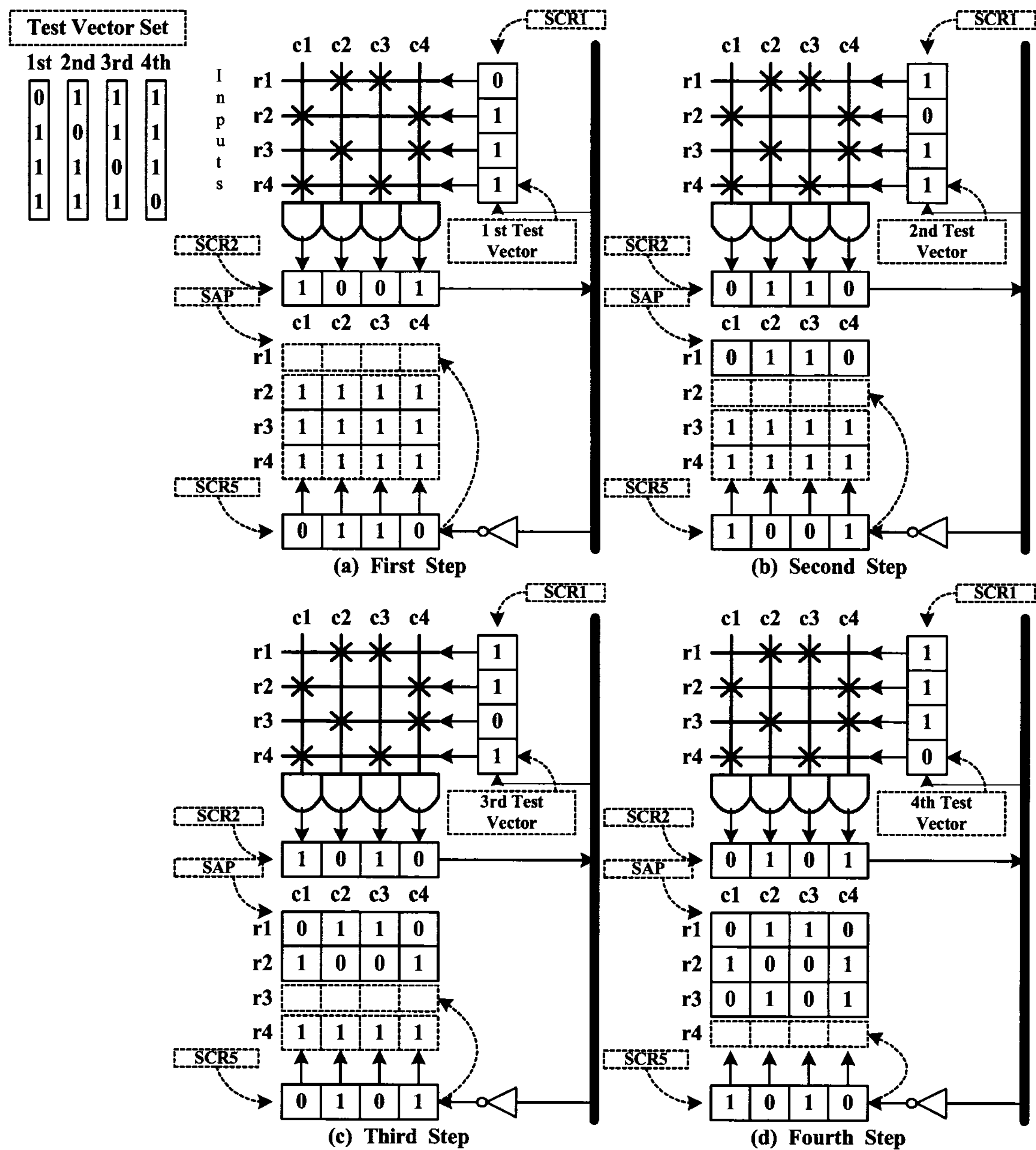


Figure 4.7 Generation of the Test Vector Set/Storing Scanning Results into the SAP

This data '0110' in the SAP is the same as the first one in the MAP since the first row has no faults. This '0110' is also interpreted as the programmed state of that row.

Figure 4.7 shows the procedure of generating the test set and storing the scanning result into the SAP.

Figure 4.8 illustrates the comparison operation for finding faults in a circuit without faults. Figure 4.9 shows the fault diagnosis and location of a simple example with faults. After getting all data in the SAP, these data are compared with the MAP entries. For the compare operation in Figure 4.8, the SCR4 receives the c1 from the MAP, the SCR3 gets the c1 from the SAP, and the c1 of the MAP is subtracted from the c1 of the SAP, and so on. As shown in Figure 4.8, all results from this bit-by-bit 'minus' operation are '0's, since there are no faults in the AND plane. It also means that the MAP is exactly the same as the SAP according to all '0's from the comparison operation. An example of a circuit with some faults is shown in Figure 4.9. The originally programmed personality of the AND plane is the same as in the previous example, but there exist multiple faults (s-a-0 faults and s-a-1 faults) in the AND plane in this case. All multiple faults are diagnosed and located using the same method explained above.

In this case, there are multiple faults in the AND plane. The intersections of c3 & r2, and c4 & r2 have s-a-1 and s-a-0 faults, respectively. When a test vector '1011' is inserted from the SCR1, the result '0101' is obtained in the SCR2. The third bit of SCR2 will be '0' since that cross-point has s-a-1. It means that the second bit of the test vector, '0', can be transmitted to the third AND gate since that cross-point (E^2CMOS cell) is ON (s-a-1). The fourth bit of SCR2 will be '1' since that cross-point has s-a-0 which means that the second bit of the test vector, '0' cannot be transmitted to the fourth AND gate since that cross-point is OFF (s-a-0). Thus the diagnostic result '0101' of the SCR2 is obtained, and it will be inverted to store it into the SAP through the SCR5. The fault locating process is based on comparisons as shown in Figure 4.8.

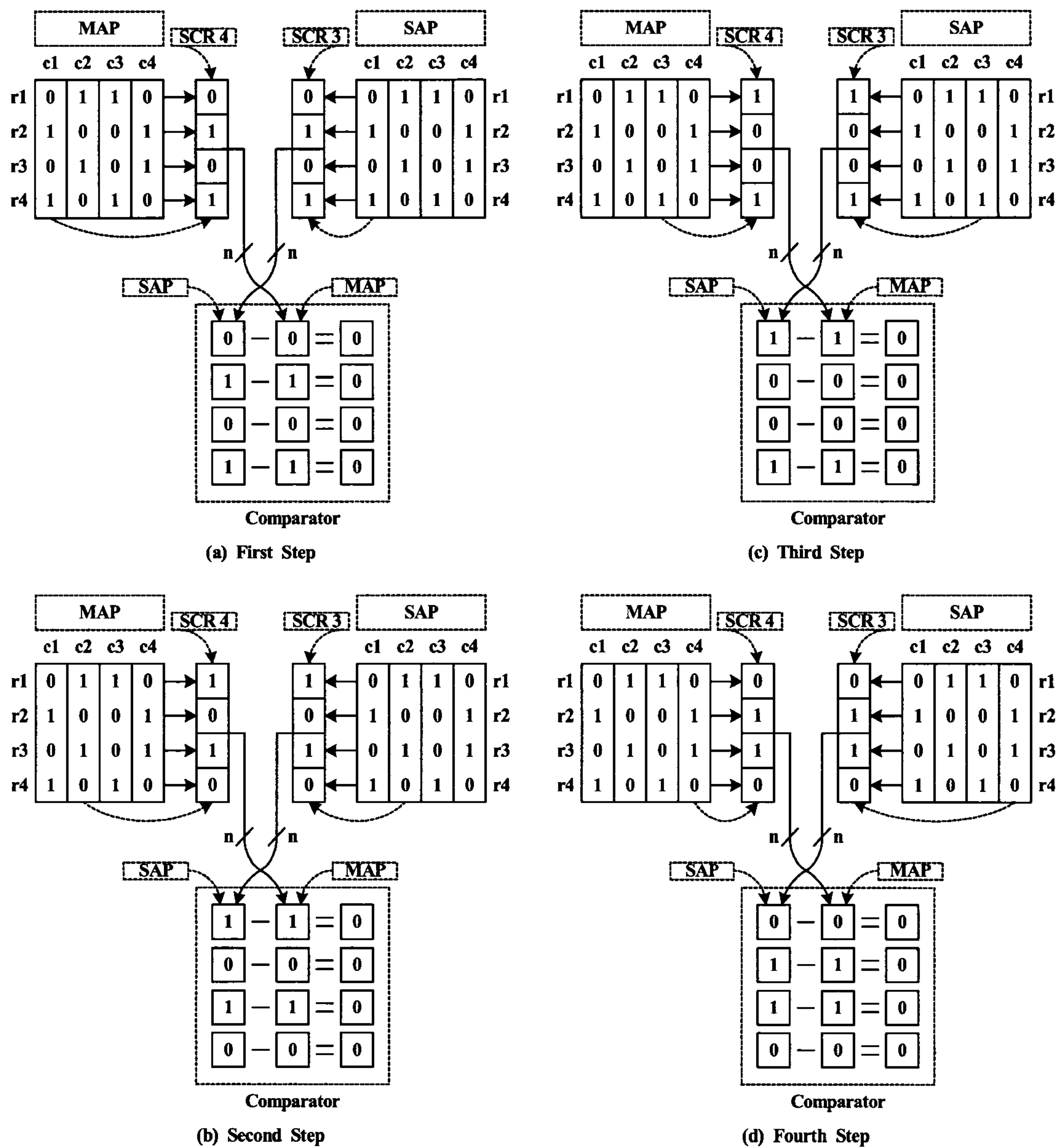


Figure 4.8 Comparator Operation for Finding Faults on a Circuit without Faults

Note that the faults that are the same type as data programmed in E^2CMOS cells cannot be detected and located. In other words, a s-a-0 fault cannot be detected and

located if the s-a-0 fault occurs in an E^2CMOS cell programmed as OFF, and a s-a-1 fault cannot be detected and located if the s-a-1 faults occurs in an E^2CMOS cell programmed as ON. It means that the results of the comparisons are all '0s' after the minus operation since the value on that location of the SAP and the value on that location of the MAP are the same as '0' and '1' in the s-a-0 fault's case and the s-a-1 fault's case, respectively. However, it does not affect the operation of the GAL at all. The controller needs to obtain this result from the minus operation of the comparator to update and control the MAP and the NC register. The detailed operation of the controller is implemented as the computer-based simulation program in our simulator. It is explained in more detail in Chapter 6 and Chapter 7.

The symbolic notation that will be used in the following examples is the following:

" Symbol Notation "	
╳	Programed into ON (Connected) CrossPoint
┼	Programed into OFF (Disconnected) CrossPoint
⊕	Stuck at 0 Faulty CrossPoint
⊗	Stuck at 1 Faulty CrossPoint

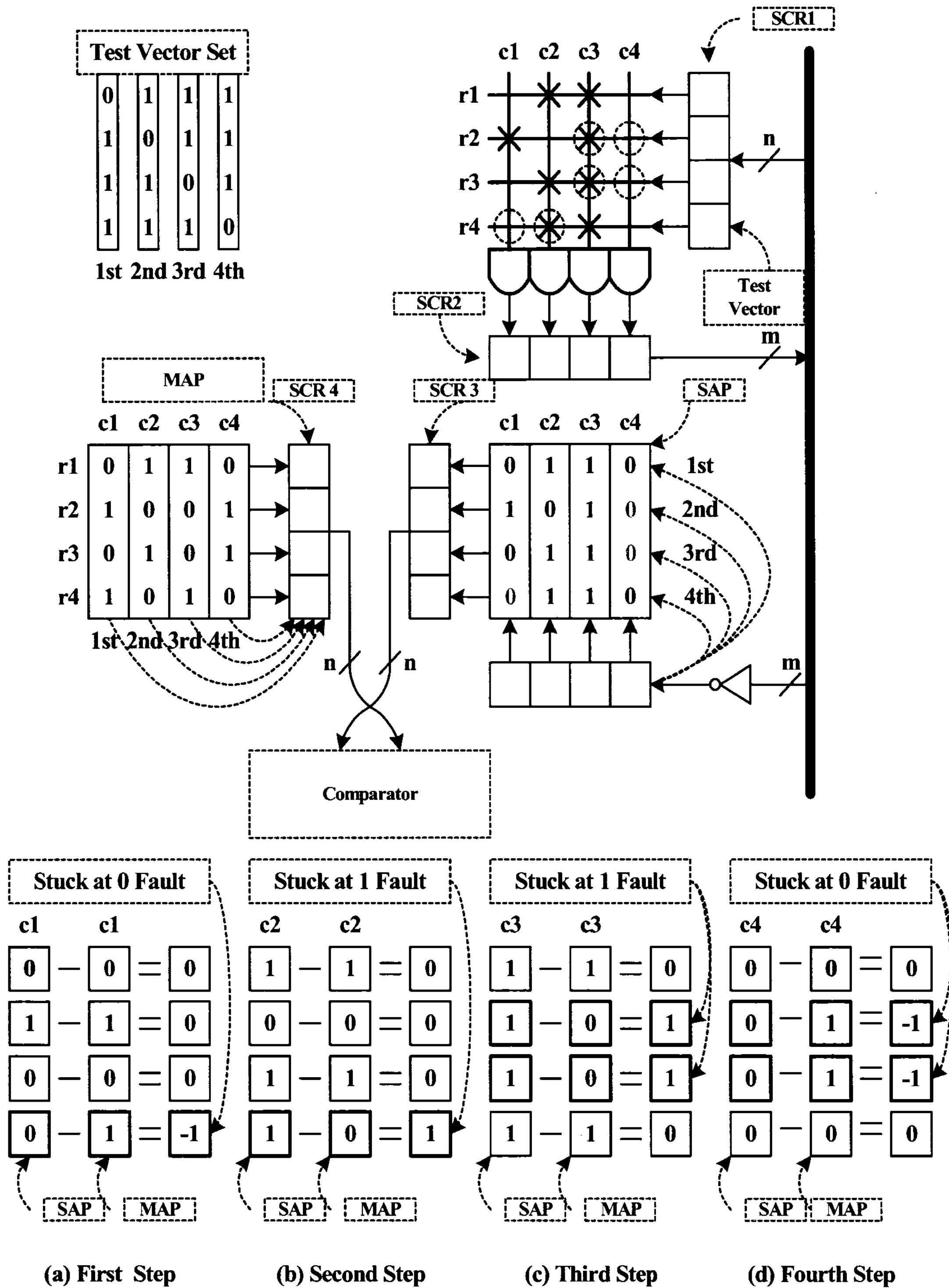


Figure 4.9 Fault Diagnosis/Location of an Example with Faults

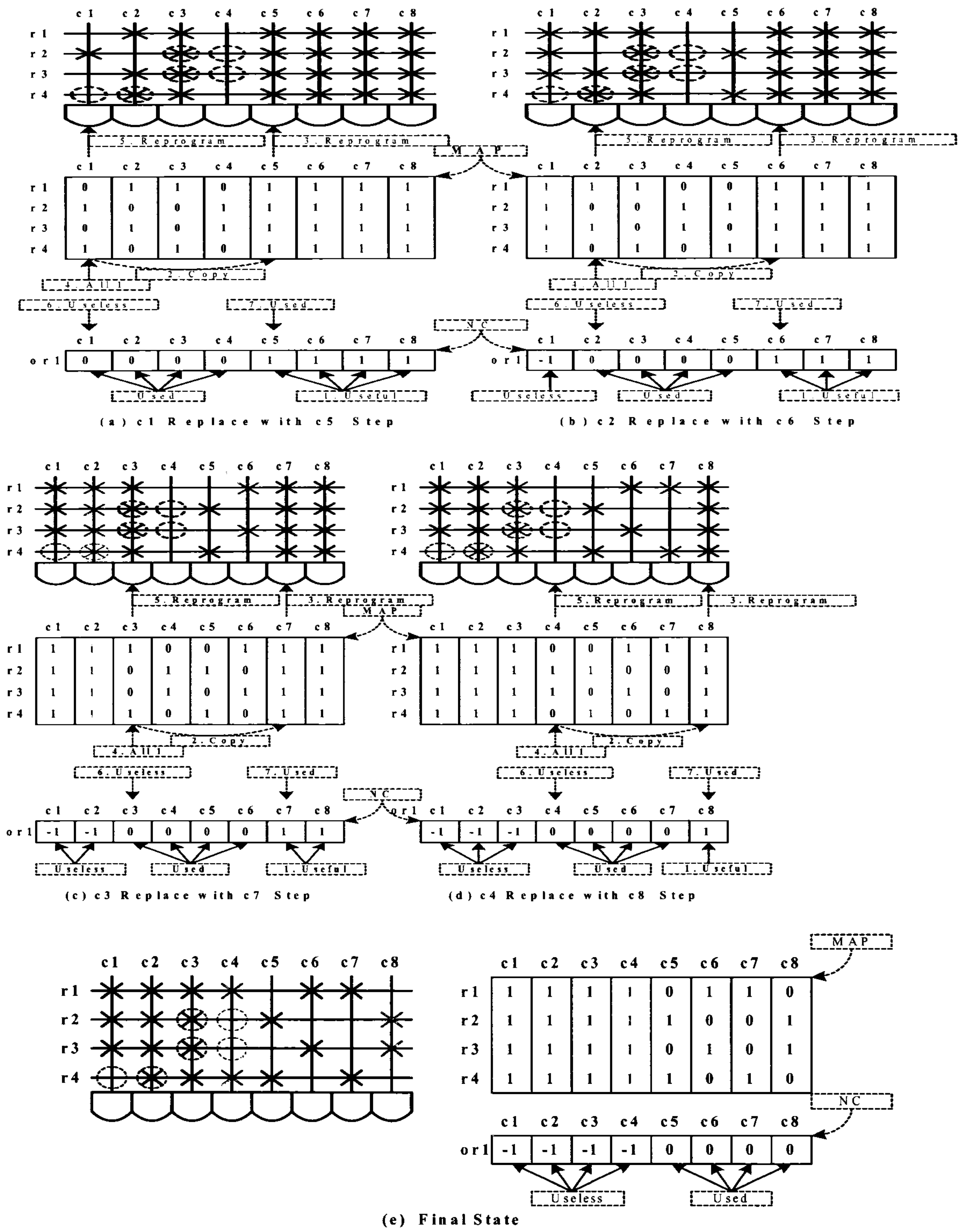
4.2. Self-Repairing Methodologies

Three self-repairing methodologies (column replacement method, column-column re-use method, and cell-column re-use method) are introduced in this section [47, 48, 49]. These methods should be applied after generating all test vectors and creating the SAP. Extra columns will be used to replace faulty extra ones even if the spare columns include faults.

4.2.1. Column Replacement with Extra Columns

The column replacement method starts after obtaining the SAP. First, it finds the column that has a logic value '1' in the NC register. The faulty column is copied at the column location of the MAP corresponding to the column that has logic value '1' in the NC register. That column is reprogrammed into the AND plane according to the MAP, and the faulty column is reprogrammed with all '1s'. Therefore, the column that is reprogrammed with all '1s' cannot affect OR gate function that includes this faulty column. However, if all cross-points in a certain column are faulty as s-a-0, it cannot be reprogrammed with all '1s'. Thus, it requires that at least one pair of literals of the

same variables, for instance A and complement of A, must be reprogrammed as ON not to affect the OR function. It would be reasonable to have this assumption since it is extremely rare to have all s-a-0 in a certain column. Finally, the NC register is updated as '-1' for the reprogrammed faulty column with all '1s', and is updated as '0' for the replaced column. The column replacement method example with multiple faults is shown in Figure 4.10. There are 4 rows (input lines) and 8 columns (product terms) in an OR group of an AND plane.



(a) c1 Replace with c5 Step

(b) c2 Replace with c6 Step

(c) c3 Replace with c7 Step

(d) c4 Replace with c8 Step

(e) Final State

Figure 4.10 A Column Replacement Method Example of Multiple Faults

The originally programmed data (data of the MAP) is the same as in the previous example described in sub section 4.1.3. The symbolic notation is also the same as before. The numbered dotted box describes the sequence of this method. All cross-points that have not been programmed are initialized as logic values '1'. There are 4 extra columns (c5, c6, c7, and c8) initialized as '1' in this example. Thus, the MAP has logic value '1' on each location corresponding to those columns. The SAP was already obtained before applying the self-repairing method; thus it is not shown in this example. However, it can be described as just actual state of this AND plane, as shown in the figure. In Figure 4.10 (a), the MAP shows the initial state that is the information programmed for c1, c2, c3, and c4, and the initial state of extra columns for c5, c6, c7, and c8. The NC register has initial data which are '0' for c1, c2, c3, and c4 since they are being used, and are '1' for c5, c6, c7, and c8 since they are available for replacing faulty columns.

Note that the self-repair should be initiated from a column which is designated as '0' in the NC register. This means that if the stuck-at faults (s-a-0 or s-a-1) occur in a column that has '-1' or '1' in the NC register, these faults are ignored for self-

repairing. In other words, the columns being used are only considered for self-repairing. If any data of the NC register has been changed in a self-repairing step, the SAP should be re-generated to compare the MAP updated after all self-repairing procedures on c1 through c8. If the all data of the SAP are the same as the data of the MAP, the GAL is correct, which means that all the repairing procedures is done and the life time of the GAL has been increased. Otherwise, the GAL should be diagnosed and repaired again until the all data of the SAP are the same as the data of the MAP.

4.2.2. Column Re-Use with Extra Columns

The column re-use method actually does not require extra columns, but it is called "column re-use with extra columns" (or simply column re-use) since it operates on an AND plane that has in addition some extra columns. There are two column re-use methods; column-column re-use and cell-column re-use. The cell-column re-use method has more advanced repairing algorithm than the column-column re-use method since the cell-column re-use, if possible, exchanges a faulty GAL column with a column that needs the same programming as supplied by the faulty cell only on the faulty column while the column-column re-use requires a same programming column

as supplied by the whole faulty column to exchange a faulty column. These methods will be explained in more detail with illustrative examples in the following subsections.

4.2.2.1. Column-Column Re-Use with Extra Columns

For self-repairing of a faulty column, this method first attempts to exchange the faulty column with a column which is already in use in the OR gate and has programming the same as that of the faulty column. The column acquired in this exchange can then be reprogrammed as needed for its new location. This test can be achieved by comparing the column in the SAP that needs to be exchanged with all column information in the MAP. For instance, if the data in c1 of the SAP is the same as the data in c4 of the MAP, then these two columns can be exchanged in this method. It means that original c1 data is stored in the c4 location of the MAP, and the original c4 data is stored in the c1 location of the MAP by reprogramming the original c1 data on the location c4 in the AND plane and vice versa. After that, the corresponding cell (c1) in the NC register will be updated as '–2.' This is to mark that the column replacement method should be applied to that column (c1), instead of the column re-

use method, if some faults are discovered in this column (c1) in a later test operation mode.

An example in which multiple faults have occurred in an AND plane is shown in Figure 4.11. There are 4 rows (input lines) and 8 columns (product terms) in an OR group of an AND plane. The originally programmed data (data of the MAP) is the same as in the previous example described in sub section 4.1.3.

All cross-points, which have not been programmed, are initialized as logic value '1'. There are 4 extra columns (c5, c6, c7, and c8) initialized as '1'. Thus, the MAP has logic value '1' on each location corresponding to those columns. The SAP was already obtained before applying the self-repairing method; thus it is not shown in this example. However, it can be described as just the actual state of this AND plane, as shown in Figure 4.11. In Figure 4.11 (a), the MAP shows the initial state that is the information programmed for c1, c2, c3, and c4, and the initial state of extra columns for c5, c6, c7, and c8. The NC register has initial data which are '0' for c1, c2, c3, and c4 since they are being used, and are '1' for c5, c6, c7, and c8 since they are useful for replacing or re-using faulty columns.

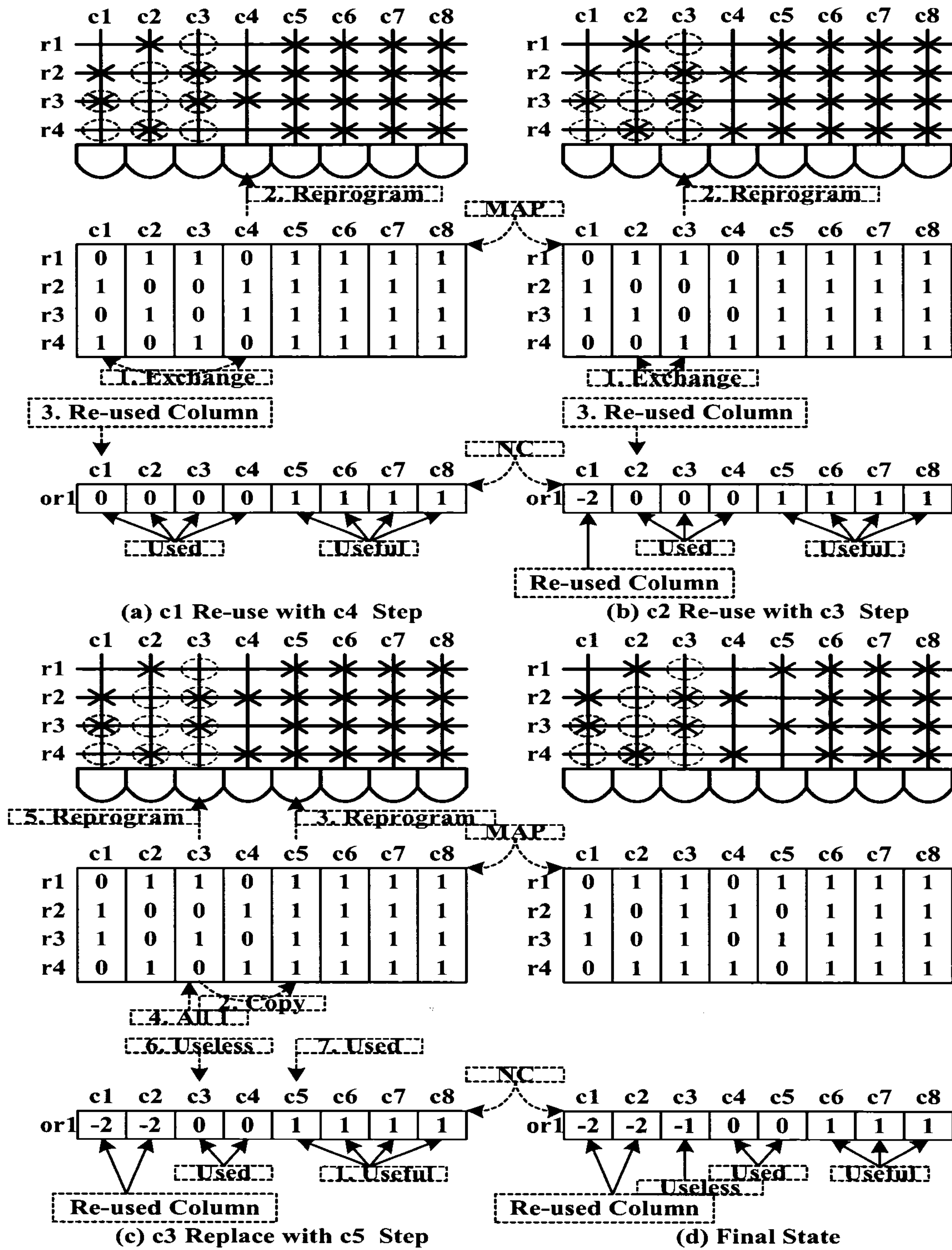


Figure 4.11 An Example of the Column-Column Re-Use with Multiple Faults

There are multiple faults in an OR group which has 8 product terms, as shown in this figure. The numbered dotted boxes describe the sequence of the column re-use in the self-repairing method. Four steps are required to repair the faults in this example.

In Figure 4.11 (a), c1 of the SAP stores '0110' since the intersection of c1 & r3, and c1 & r4 have s-a-1 and s-a-0 faults, respectively. It can be shown to be different from the c1 of the MAP, '0101'. The c1 '0110' of the SAP finds which data is the same as in the MAP. The c4 '0110' of the MAP is found as the same data of the c1 '0110' of the SAP. Thus, they can be exchanged in the MAP. The c4 '0101' exchanged data from c1 of the MAP will be programmed in c4 of the AND plane, and the c1 '0110' exchanged data from c4 of the MAP will be programmed in c1 of the AND plane. Now, the c1 of the MAP has '0110' as the original data for the c4 and the c1 has been re-used for the c4 of the MAP instead of replacing the c1 with an extra column, c5, even though the c1 has multiple stuck-at faults. Although the personalities of these two columns are switched with each other, the function of this OR gate will not be changed. The c1 of the NC register will be updated as '-2' which shows that this column has been re-used, so that it must be only replaced with a new extra column if

this column, c1, becomes faulty again. The same procedure will be done with the c2 '1001' of the SAP in Figure 4.11 (b). In this second step, the column c2 has been re-used with the c3 of the MAP although the s-a-0 fault on the intersection of c2 & r2 cannot be located or even detected since that location of the SAP and the corresponding MAP cell is the same data as the logic value '0'. However, it does not affect the function of the GAL since this column-column re-use method is looking for the exact same information with the whole data of the c2 '1001' of the SAP in the MAP, which means that the c3, 1'0'01 of the MAP is found as the same data of the c2, 1'0'01 of the SAP. In Figure 4.11 (c), the c3 '0110' of the SAP is the same as the c1 '0110' of the MAP, but they cannot be exchanged since the c1 of the NC register is marked as '-2'. Thus, the c3 '1010' of the MAP, after exchanging with the c2 of the MAP, must be replaced with a new extra column, c5. The final state is shown in Figure 4.11 (d).

4.2.2.2. Cell-Column Re-Use with Extra Columns

In new advanced column re-use method, simply called 'cell-column re-use', for self-repairing of a faulty column, if a faulty column is found, an attempt is first made to exchange that column with another column that needs the programming found in the faulty cell of the faulty column rather than the whole column programming in the faulty column. For instance, if the second cell data in c1 (MAP c1 has 0101, SAP c1 has 0001) of the SAP is a logic value '0' as a s-a-0 fault, and the second cell data in c2 (1010) of the MAP is a logic value '0' as an originally programmed data, then these two columns can be exchanged in this new method since the '0' in the second cell of SAP c1 is the same as the data in second cell of the MAP c2, despite of difference between whole information, 0001 of the SAP c1 and whole data, 1010 of the MAP c2. This property is the main difference from the previous re-use method. It means that with this method it is more probable to exchange other columns programmed already. Consequently, it is likely to save more extra columns. The column gained in this exchange is then reprogrammed to replace the faulty column. If column re-use is not possible, the faulty column must be replaced with an extra column built in the OR gate

as shown in the column-column re-use. Thus, the corresponding cell (c1) in the NC register will be updated as '–2' to mark that the column replacement method should be applied to that column (c1), instead of the column re-use method, if some faults are discovered in this column (c1) in a later test operation mode.

An example in which multiple faults have occurred in an AND plane is shown in Figure 4.12. There are 4 rows (input lines) and 8 columns (product terms) in an OR group of an AND plane. The originally programmed data (data of the MAP) and the faulty information (data of the SAP) is the same as in the previous example described in sub section 4.1.3 and 4.2.2.1, respectively.

In Figure 4.12 (a), the MAP shows the initial state that is the information programmed for c1, c2, c3, and c4, and the initial state of extra columns for c5, c6, c7, and c8. The NC register has initial data which are '0' for c1, c2, c3, and c4 since they are being used, and are '1' for c5, c6, c7, and c8 since they are useful for replacing or re-using faulty columns.

There are multiple faults in an OR group which has 8 product terms, as shown in this figure. The numbered dotted boxes describe the sequence of the column re-use in

the self-repairing method. Four steps are also required to repair the faults using the same example in the column-column re-use. However, this method does not use any extra columns for self-repairing of the faulty columns while the column-column re-use uses an extra column to repair all faults. In Figure 4.12 (a), c1 of the SAP stores '0110' since the intersection of c1 & r3, and c1 & r4 have s-a-1 and s-a-0 faults, respectively. It can be shown to be different from the c1 of the MAP, '0101'. The third (1) and fourth data (0) in c1 '0110' of the SAP finds which data of these locations (3rd and 4th cell) are the same as in the MAP. The c2 '1010' of the MAP, which has '1' in 3rd cell and '0' in 4th cell, is found as the same data of the c1 '0110' of the SAP, which has '1' in 3rd cell and '0' in 4th cell. Thus, they can be exchanged in the MAP. The c2 '0101' exchanged data from c1 of the MAP will be programmed in c2 of the AND plane, and the c1 '1010' exchanged data from c2 of the MAP will be programmed in c1 of the AND plane. Now, the c1 of the MAP has '1010' as the original data for the c2 and the c1 has been re-used for the c2 of the MAP. The c2 '0101' cannot be programmed correctly in c2 of the AND plane because of a s-a-0 fault in 2nd and 3rd cell, and a s-a-1 fault in 4th cell in c2 '1001' of the SAP.

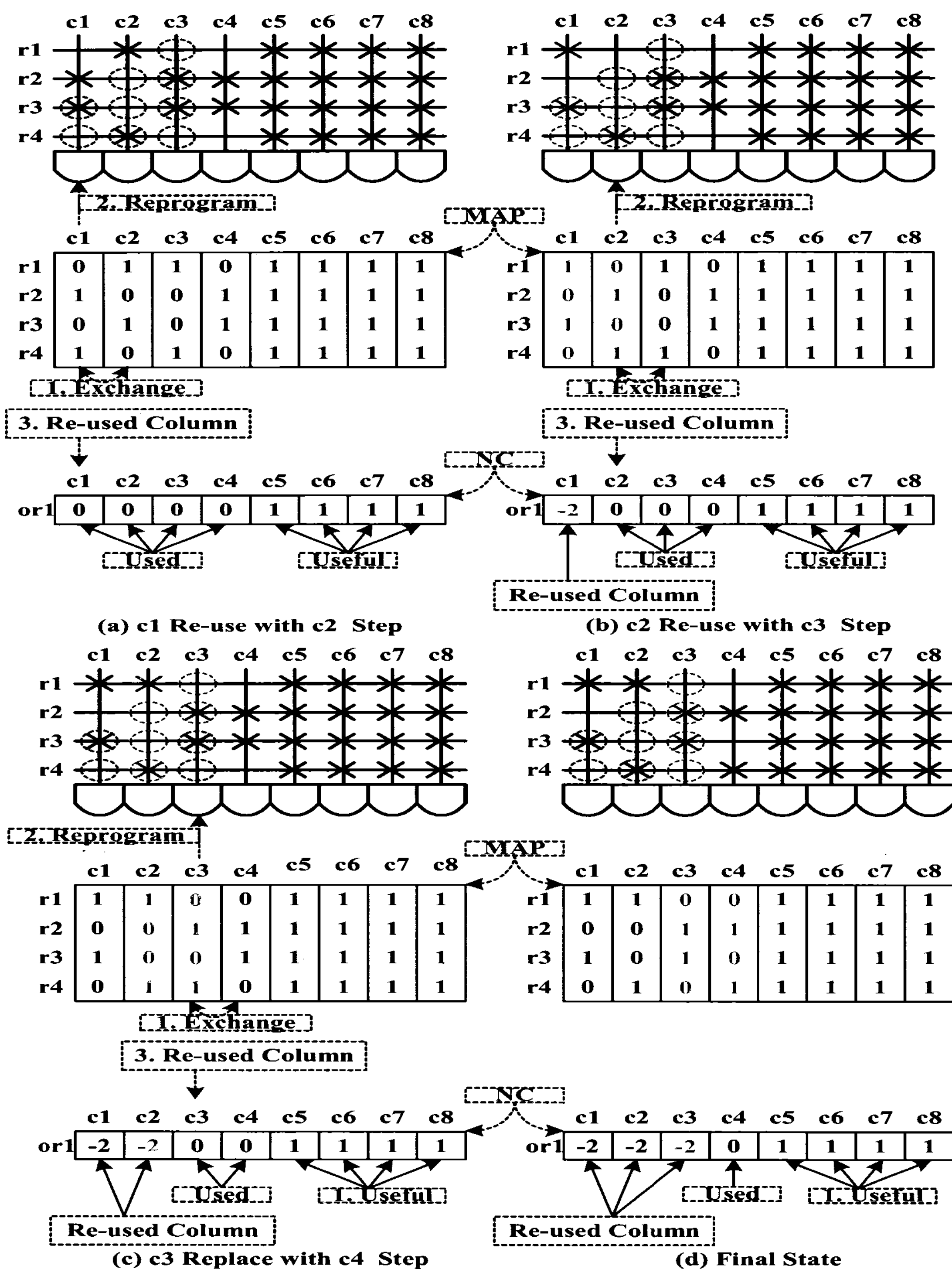


Figure 4.12 An Example of the Cell-Column Re-Use with Multiple Faults

At this step, this problem is just ignored because only c1 is considered if it can be re-used. The c1 of the NC register will be updated as '-2' which shows that this column has been re-used, so that it must be only replaced with a new extra column if this column, c1, becomes faulty again. The same procedure will be done with the c2 '1001' of the SAP in Figure 4.12 (b). In this second step, the column c2 has been re-used with the c3 of the MAP with same data in 3rd (0) and 4th (1) of both the SAP c2 and the MAP c3. Although the s-a-0 fault on the intersection of c2 & r2 cannot be located or even detected since that location of the SAP and the corresponding MAP cell is the same data as the logic value '0', the c2 can be re-used with the c3 safely. The c2 programmed incorrectly in the AND plane in previous step has resolved. In Figure 4.12 (c), the c3 '0110' of the SAP, which has all faulty cells, is the same as the c4 '0110' of the MAP, thus they are exchanged. Thus, the c4 '0101' of the MAP, after exchanging with the c3 of the MAP, is reprogrammed in the c4. The final state is shown in Figure 4.12 (d).

In this method as well as the column-column re-use, the SAP should be re-generated to compare the MAP updated after all self-repairing procedures on c1

through c8 as described in sub section 4.2.1. If data of the re-produced SAP is not the same as data of the MAP gained finally, the cell-column re-use should be applied into the GAL with the SAP newly acquired after test generation. Thus, this cell-column re-use method might have more test and repair time for faulty columns according to hidden faulty personalities, i.e. a s-a-0 fault occurred in a cell programmed as '0', or a s-a-1 fault occurred in a cell programmed as '1'. On the other hand, this method facilitates the use of columns being used and reduces the use of extra columns.

4.2.3. Integration of the Column Repair Methods

Since the Column-Column Re-Use with Extra Columns algorithm works within the Cell-Column Re-Use with Extra Columns algorithm, the algorithms can be divided into two major types; the Column Re-Use with Extra Columns and Column Replacement with Extra Columns.

The Column Re-Use with Extra Columns algorithm only uses columns that have been programmed (previously used) to repair, and the Column Replacement with Extra Columns algorithm uses columns that has not been programmed (new) to repair.

These algorithms run independently and each algorithm does not have an effect on each other. Thus, the combined algorithms can be used on a system and maximize the performance.

There are two basic ways to run the integrated algorithm. It can perform the Replacement method first, and then perform the Re-Use method, or vise versa.

The first method (Replacement first then Re-Use) is not ideal for extending the usage of each column. The initial state of OR-gates on GALs consists of programmed columns and extra columns. The columns repaired by Re-Use cannot be used again because its inability to detect hidden faults. However, the columns repaired by Replacement can be used again by Re-Use algorithm. Thus, when Replacement algorithm is used, it must use Replacement algorithm only until all the extra columns are used. When Re-Use algorithm is used without any extra columns, the number of columns that can be used to repair is equal to the total number of columns per OR-gate.

The second method (Re-Use first then Replacement) on the other hand, Re-Use algorithm can use the existing columns to repair as well as the columns that were used

by Replacement repair. In this case, the number of times columns can be used per OR-gate is: (Used column + Extra column x 2).

An OR-gate with 16 columns (8 original and 8 extra) for example, the first method yields 16 columns that can be used to repair, and the second method; 8 + 8x2 = 24. The number of column usage per GAL under the same specifications and circumstances is directly related to a system's lifetime or performance. In this case, the second method's performance value is expected to be 1.5 times better than the first method's. For this reason the integrated (combined) algorithm used in this dissertation is the second method, Re-Use first then Replacement.

4.2.4. Replacement and Re-use with Extra OR-gates

The concept of adding extra OR groups (simply called ORs) in a GAL is introduced in this section briefly. It is used to replace with ORs and increases the reliability of the GAL as well as the system with the switching circuit, which has extra interconnection lines in a system. The initial purpose of adding extra ORs in a GAL is to implement more reliable system for a GAL itself even though it has large

hardware overhead. However, the real objective is that when the line being used and the extra line in a switching circuit are both defective or when all columns in a AND plane of a GAL are exhausted, the system must be self-repairable.

This method also uses a FLFRP (Fault-Locating/Fault-Repairing Processor), diagnosis/repair bus, and the memory that stores fault location and repair-related data. The NR register shown in Figure 5.5 is used for the actual state of the OR. The size of each NR corresponding to a particular GAL is the same number of ORs in each GAL.

The detailed procedure of using the extra OR is shown in Chapter 5 since it is functional with the self-repairable switching circuit.

5. Self-Testing and Self-Repairing Switching Circuit

As initiation into the next problem to broaden the first project, the concept of the self-testing and the self-repairing switching circuit based on a Demultiplexer structure is described as our second project. It is used to connect and reconfigure GALs in our system proposed in Chapter 4 and enhances the effectiveness and the lifetime of the system together with added ORs (Extra ORs) group in a GAL module. The purpose of adding switching circuits between GAL modules is to implement larger system which is too large to fit in a single GAL. At this point, a connection circuit requires being self-testable and self-repairable since the whole system including an array of GAL modules is useless if connections between GALs fail.

We develop the fault-locating and fault-repairing architecture with Demultiplexer-based structure with an AND gate logic to allow detect, diagnose, and repair of all multiple stuck-at faults that might occur on wires in a switching circuit between GAL modules. A self-repairing method is presented based on our switching circuit design architecture; a line replacement method with extra lines (simply called line replace). This method also uses a FLFRP (Fault-Locating/Fault-Repairing Processor),

diagnosis/repair bus, and the memory that stores fault location and repair-related data. We also propose an evaluation methodology for this project and it is described in Chapter 6. This methodology is based on simulating the self-repair algorithm.

The switching circuit developed in this chapter is self-testable and self-repairable for faulty lines with extra lines in a switching circuit between GALs of a system. An input pin of the switching circuit is connected to an output pin of a GAL and an output pin of the switching circuit is connected to an input pin of the other GAL through a line in a switching circuit. The active line in a switching circuit, which means that the wire is used to connect GALs, is duplicated with spare wires. If the original line is defective, then the switching circuit will use a spare line. If both an original line and an extra line are faulty, then the system starts to look for an available OR group at the GAL connected to the input pin of the switching circuit. The programmed information (column information) of the OR connected to the faulty input pin of the switching circuit is replaced with an available OR, an extra OR, in that GAL, and the OR is marked as unavailable OR, which means that the OR is no more useful. The OR that has reprogrammed information on the extra OR is rerouted with the switching circuit

and it keeps the same output pin connection in the switching circuit. It is that the only input side of the switching circuit is changed but the output side of the switching circuit is kept the same connection to the GAL waiting a signal from the switching circuit. Thus, the function of the whole system can work properly. If an OR of a GAL connected to the input of a switching circuit is no more useful, which means that the columns of that OR group are faulty and cannot be repaired using column repair methods, then the information (column information) of that OR is copied to an extra OR in that GAL. Even though the connection line in the switching circuit has no fault, the switching circuit reroutes the connection between GALs since the GAL at the input side of the switching circuit is malfunctioning.

The fundamental idea of adding the self-testing and self-repairing switching circuit on the self-repairable GAL structure shows the concept of self-testable and self-repairable digital devices and a reliable computing system prototype. The concept may also be applicable to FPGA (Field Programmable Gate Array) design or ASIC (Application-Specific Integrated Circuit) design to shrink the design gap more quickly and affordably by adding redundancy. In addition, this self-testing and self-repairing

methodology for the switching circuit might maximize the safety and security issues of the systems in addition to the self-testing and self-repairing methodology for the EPLDs.

5.1. Hardware Design Mechanism of the Self-Repairable Switching Circuit

In this section, we develop a design methodology for self-repairing a wire (line) in the switching circuit between GALs, describe our fault model and assumptions, and develop universal test set for detecting and locating faults on each interconnection line in a switching circuit.

5.1.1. Fault Model and Assumptions

The line stuck-at faults, which are located in interconnection wires inside a switching circuit, are considered as our fault model because lines of a switching circuit make up a large percentage of a switching circuit as does the E^2CMOS cells' array in a GAL. The stuck-open fault model, which can be tested with stuck-at fault test, or the

bridge fault model, which is more applicable as line width get smaller and covered by stuck-at fault, could be a fault model for the interconnection wires in a switching circuit. At this point, we only consider that the signal fed into the input pin of the switching circuit is transmitted to output pin of the switching circuit correctly. Thus, we assume that the faults on the interconnection line are covered by stuck-at fault model since we are interested in the output signal of the switching circuit for go or no go to next GAL module. Each interconnection line of a switching circuit, which is located between a Demultiplexer (simply called DEMUX) and an AND gate, may be ON or OFF permanently, caused by factors referred to in chapter 1. It is called the line stuck-at-1 (simply, s-a-1) if the interconnection line of a particular route between GAL modules is stuck at 1 functionally. If the interconnection line of a particular route between GAL modules is stuck at 0 functionally, it is called the line stuck-at-0 (simply, s-a-0). Only these faults will be considered in this second project because we assume that there are no faults found in a switching circuit after the manufacturing process.

The following assumptions will be used for the design, self-repair and evaluation methodologies for the switching circuit. The switching circuit is initially fault free

after manufacturing. The primary input/output and control lines (buffer enable, DEMUX enable/select) fault do not exist in a switching circuit, and also every logic gate (DEMUX, AND gate, tri-state buffer) in a switching circuit do not have faults. For switching circuits in our system, all multiple s-a-0 and s-a-1 faults are detectable and repairable. The interconnection line faults (s-a-0, s-a-1) as defined above are only taken into account in this second project.

In order to replace or reroute a faulty interconnection line, one extra line is built into each interconnection between an input pin and an output pin in the switching circuit. There are tri-state buffers to block the faulty interconnection to be transmitted into the next GAL module. Thus, the output signal is discarded safely from the switching circuit even though the faults appear in certain interconnection lines of that switching circuit. When a fault occurs in a certain line of a particular switching circuit, this faulty line can be replaced or rerouted with an extra line only in that switching circuit. When the enable signal of a tri-state buffer has logic value '0', the output signal of the AND has no effect to the next GAL module, and if it has logic value '1', the output signal of the AND gate operates to transmit to the next GAL module. The

information of the tri-state buffer will be stored in the RTB (Register for Tri-State Buffer) located in the FLFRP described in Chapter 4. The RTB has data to control AND gate outputs, not shown in Figure 4.11.

5.1.2. Design Architecture

The complete digital system in our second approach is shown in Figure 5.1. The system is an array of self-repairable GAL modules and self-repairable switching circuit blocks realized both as separate integrated circuits. It is the same as our first approach shown in Figure 4.11. Each GAL module is a two-level realization of a Boolean function, and is realized with a self-testing and self-repairing GAL using extra columns and extra ORs. Each switching circuit module has DEMUX and AND gate logic blocks, and has interconnection lines between them with extra lines for repairing or rerouting the faulty lines. This is our new hardware prototype for the ultra reliable computing system, and seems to be more reliable and robust system since every block can be self-testable and self-repairable in the system. The potential synergy between two self-repairing modules (GAL and switching circuit) makes the

system ideal candidate for the EPLD/FPGA technology advances. The proposed method assumes that when the redundant lines to reroute or replace lines (i.e. the extra lines) in a switching circuit block are exhausted and there are no more lines to replace a newly found defective line, the controller will signal a global go/no-go signal. We assume that the failure of the switching block is less probable than the GAL module. We assume that each switching block is a single module (a chip, a board) and each GAL module includes just one self-repairable GAL which has a programmable AND array with several extra columns and fixed OR (OLMC) plane with extra ORs.

In this stage, we consider the problem of getting the maximum usefulness of the whole system at the highest level, when the system degrades over time with new faults arising in a GAL module and a switching block that have already been tested in the manufacturing process.

The following figure shows the general scheme of the design architecture for a self-repairable switching circuit as well as our system proposed as a reliable computing system. It uses extra lines between the DEMUX logic block and the AND gate, Scan Registers (SCR6 and SCR7), Diagnosis/Repair Bus, and a FLFRP which

has a MSCI (Module-Switching Circuit Information), NSC (Next Switching Ciruit), NR (Next OR), MCIR (Module-Connection Information Reference), NSA (Next State AND), RD (Register for DEMUX), RB (Register for Buffer), RTB (Register for Tri-state Buffer), and NDS (Next DEMUX Select), a Comparator, a Central Fail-Safe Maintenance Controller, and Data Path/Addressing Unit. The FLFRP can be realized with a micro-controller.

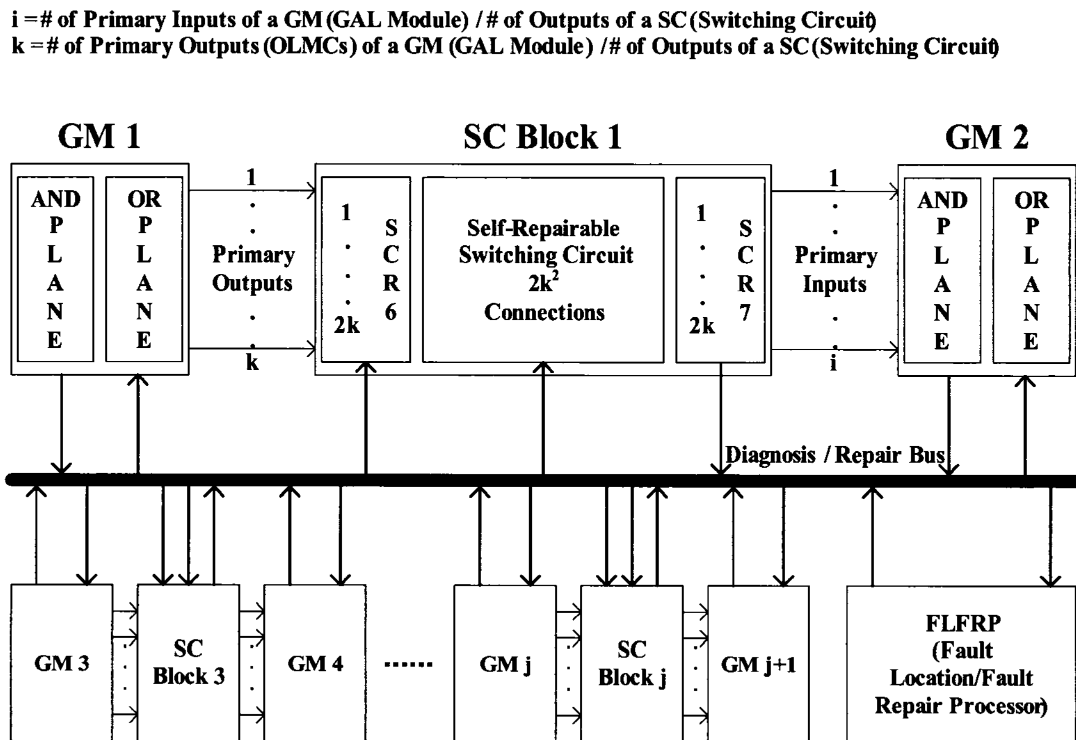


Figure 5.1 Design Architecture for the Ultra Reliable Computing System with Self-Repairable GAL Module and Self-Repairable Switching Circuit Block

In the normal operation mode, the primary output data form a GAL module are fed into a switching circuit and the connections of the switching circuit are just transparent through the SCR7 into the next GAL module. In the test mode, the SCR6 will have a test vector fed from the controller of the FLFRP. The SCR7 then receives a scanning result corresponding to a test vector from the SCR6. The data of SCR6 is sent to the RD and the scanned result data of SCR7 is sent to the RB through the Diagnosis/Repair bus. The Diagnosis/Repair bus is assumed to include control, data, and address bus.

The bus is serial to decrease the pin-out of the chip. The 'k' denotes the number of primary inputs and outputs of a switching circuit, corresponding to the primary outputs of a GAL and the primary inputs of a next GAL module, respectively. Each primary input pin of the switching circuit has two DEMUXs and corresponding two AND gates. First DEMUX connected with an input pin of the switching circuit is corresponding to first AND gate in each output pin of the switching circuit. Second DEMUX connected with an input pin of the switching circuit is corresponding to second AND gate in each output pin of the switching circuit, and so on. That means each DEMUX has 1 input

and 'k' outputs, and each AND gate has 'k' inputs and 1 output to connect into a tri-state buffer in the switching circuit. Thus, there are '$2k^2$' connection lines, which are replaceable with faulty lines in the switching circuit. The detailed inner structure of the switching circuit block is shown in Figure 5.2.

The '2k' is said to be the number of DEMUXs, the number of AND gates, the number of tri-state buffers in a switching circuit. Each bit of the 2k-bit SCR6 and SCR7 corresponds to 1 DEMUX and 1 AND gate output, respectively. The extra line means that there are several extra lines in each input pin of the switching circuit. We assume that there is one extra line in each input pin of the switching circuit, which means that only one extra line exists for replacing a faulty line on that input pin. We assume that the switching circuit should have the same pin-to-pin configuration (the input pin to output pin connection) as originally programmed system information in a system. Thus, 'k-1' numbers of extra lines in a DEMUX are used for rerouting in case of a faulty OR, which means that the faulty OR needs to be replaced with an extra OR since extra columns are exhausted.

Switching Circuit Block j

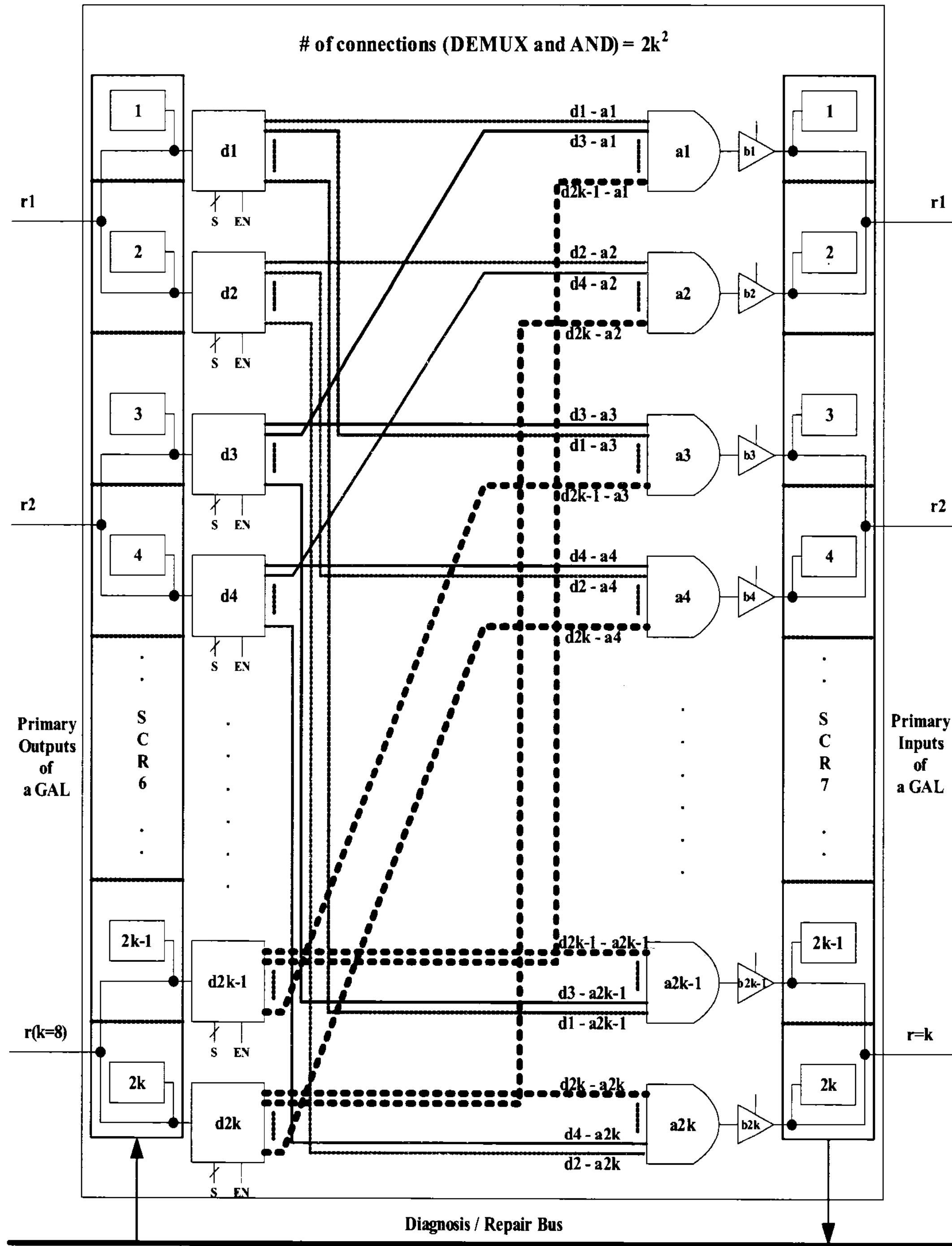


Figure 5.2 Inner Structure of the Switching Circuit Block

In / Out	GM 1	GM 2	————	GM *l*-1	GM *l*
GM 1	×	SC 1	————	SC 1	SC 2
GM 2	SC 3	×	————	SC 3	SC 4
|	|	|	—+—	|	|
GM *l*-1	SC 4	SC5	————	×	SC 5
GM *l*	SC 6	SC7	————	SC 8	×

Out GM : Master GAL module fed into a SC **SC: Switching Circuit between modules**
In GM: GAL module connected by the out GM ***l*** **= # of GAL modules in a system**

Figure 5.3 Structure of MSCI Array

The MSCI stores original connection information between GAL modules in a system by which switching circuit is being used to connect GALs as show in Figure 5.3. The output GAL module (simply called Out GM) is said to be a master GAL fed into a switching circuit (simply called SC) and Input GAL module (simply called In GM) is a GAL module connected by the out GM. In Figure 5.3, SC3 is used to connect GM2 with both GM1 and GMl-1. 'l' is the number of GAL modules in a system, thus this MSCI is an 'l' x 'l' matrix.

PI (row): Primary Input in a SC (Output of a GAL module)
a (column): Connetion lines of an AND gate in a SC
PO: Primary Output of a SC (2 AND gates per a PO)
k = # of primary input/output lines of a SC

	PO(1)		PO(2)		———	PO(*k*)	
Out / In	a 1	a 2	a 3	a 4	———	a 2*k*-1	a 2*k*
PI(1)	0	1	0	0	———	1	1
PI(2)	1	0	1	0	———	1	1
|	|	|	|	|	—+—	|	|
PI(*k*-1)	0	0	1	0	———	1	0
PI(*k*)	1	1	0	0	———	0	1

0 : Stuck-at fault (s-a-0 or s-a-1) **1 : No Stuck-at fault**

Figure 5.4 Structure of the NSC Register

The structure of the NSC is shown in Figure 5.4. It has similar information to the SAP in case of self-repairable GAL. The NSC informs about the information for the stuck-at faults. It is updated after testing an SC, depending upon the stuck-at faults (s-a-0 or s-a-1) or no stuck-at faults. The size of the NSC is the number of inputs in an SC times the number of AND gates in an SC. It has information about the actual state of each AND gate of an SC after testing. It will show the state of stuck-at faults on the

interconnection lines. It includes both the actual state as well as the stuck-at fault information. RD will be compared with the RB to find and locate faults using minus operation. The RD is the same size of the SCR6 and the RB is the same size of the SCR7. Both RD and RB are not shown in the Figure. The RD is subtracted from the RB bit by bit and the results are stored in NSA not shown in the Figure. If the result of subtraction is '0', then the line, corresponding to that bit is correct which means that it has no fault. If the result is '-1', then it has stuck-at-0 fault, and if the result is '1', then it has stuck-at-1 fault in a particular line of that AND gate. Thus, the NSA has the actual state information of lines being connected into an AND gate of an SC. The NSC is updated by the NSA after detecting and locating faults in lines of an SC. The NSC shows the state of AND/line, and which AND gate can be useful or which AND gate is no more useful to keep a connection from a primary input to a primary output as originally programmed information in an SC. If a particular cell in the NSC is '0', then the corresponding line of the AND gate has stuck-at fault, either stuck-at 0 or stuck-at 1. If a stuck-at 0 fault has occurred in a particular line, then the whole column of the NSC, corresponding that line of the AND gate, will be updated as logic value '0'. It

means that the AND gate is no more useful because of the stuck-at 0 fault. If a stuck-at 1 fault has occurred in a particular line, then only the cell of the NSC, corresponding that line of the AND gate, will be updated as logic value '0'. It means that the AND gate is still useful for the connection of input pin and output pin programmed in the SC since the stuck-at 1 fault will not affect to the extra line. This will be described in more detail with an example in sub section 5.2.1.

The structure of NR is shown in Figure 5.5. If a certain bit has '0', the corresponding OR in a master GAL is being used, and if a certain bit has '-1', the corresponding OR in a master GAL is available to replace a faulty OR which means that the OR is extra OR in the GAL. If a certain bit has '1', the corresponding OR in a master GAL is unavailable OR to replace faulty one which means that the OR is faulty.

OR#	Status
OR 1	-1
OR 2	0
|	|
OR(*k*-1)	1
OR(*k*)	1

0 : OR group being used

1 : Available OR to replace (Extra OR)

-1 : Unavailable OR to replace (Faulty OR)

Figure 5.5 Structure of the NR Register

The MCIR register informs about the status of connections of each AND gate in each output pin of an SC. As shown in Figure 5.6, If a certain bit has '0', the corresponding connection is being used with the AND gate in an SC, and if a certain bit has '-1', the corresponding connection is available connection with the AND gate in an SC, which means that the connection line in the SC can be used for the extra line. If a certain bit has '-2', the corresponding connection is unavailable since that connection line has stuck-at 0 fault.

	PO(1)		PO(2)		————	PO(*k*)	
Out	a 1	a 2	a 3	a 4	————	a 2*k*-1	a 2*k*
Status	0	-2	0	-1	————	-1	-1

0 : Connnection being used with the AND gate in a SC
-1 : Available Connection with the AND gate in a SC
-2 : Unavailable Connection with the AND gate in a SC

Figure 5.6 Structure of the MCIR Register

5.1.3. Test Generation and Fault Diagnosis/Location

All stuck-at faults, mentioned in section 5.1.1, on connection lines of an SC can be determined by a pattern. We have a universal test set for detecting faults.

This test vector set, which is provided by the FLFRP, can find whether the fault is an 's-a-0' or an 's-a-1'. It can determine the exact location in which the line is faulty as stuck-at 1. All multiple faults of lines in an AND gate can be detected by this test set. The combination of s-a-0 and s-a-1 is not located by this test vector. However, the detection of the faults is enough for replacing and rerouting faulty ones in case of stuck-at 0 and 1 combination. The AND gate should be discarded if it has stuck-at 0 fault in any line of an AND gate in an SC. The length of a test vector is 2k (k is

number of input pin of an SC) bits and the '2' test vectors are needed to test. It detects the faults line by line in the SC. The FLFRP initializes SCR6 to '111...11', and this initial test vector will detect stuck-at 0 faults on the connection lines in an SC. The value of '1s' in the SCR6 is fed into the DEMUXs, and it will affect the AND gate outputs as a value of 1. In other words, the result of this AND gate should be '1', but if the output of this AND gate is a '0', it will be a s-a-0 fault. On the other hand, the value of '0s' is fed into the DEMUXs line by line using the select of a DEMUX, but if the output of this AND gate is a value of 0, it will be a s-a-1 fault. Thus, all multiple faults are detected and s-a-1 fault is located by this universal test set, and it is shown as follows.

❖ ***Test Vector Set (Test Pattern)***

1	***2***	***3***	***...***	***n-1***	***2k (k inputs)***
1	*1*	*1*	...	*1*	*1*
0	*0*	*0*	...	*0*	*0*

A simple example in which a system has two GMs and 1 SC is shown in Figure 5.7. Each GM has 2 OR group (simply called OR), 2 original programmed columns, 2 extra columns. The SC of this example will be used for the rest of examples in this

chapter. It has 2 input pins (12 and 13) and 2 output pins (2 and 3). There are 4 DEMUXs, 4 AND gates, 4 tri-state buffers in the SC. The input pin 12 is connected to both DEMUX1 (d1) and DEMUX2 (d2) and connected with output pin 2 through AND1 (a1)/buffer1 (b1) and AND1 (a1)/buffer1 (b1), respectively, likewise input pin 13 and output pin 3 connection. The d2-a2 connection line is an extra connection line for the d1-a1 connection line and the d4-a4 connection line is an extra connection line for the d3-a3 connection line.

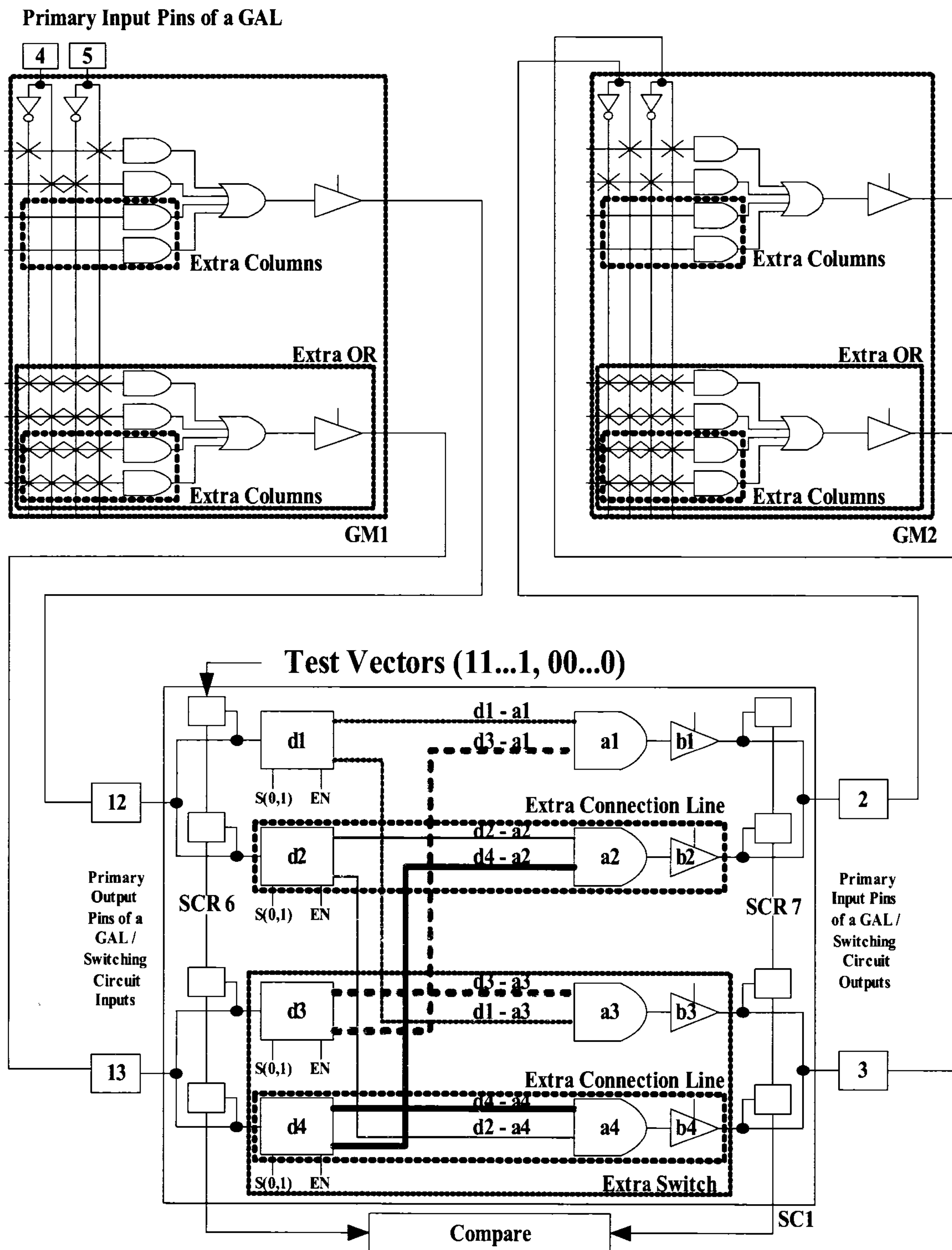


Figure 5.7 General Concept of an Example without Faults

The procedure of generating the test set in SCR6, and next storing the scanning result into the SCR7 is illustrated as an animated sequence in Figure 5.8.

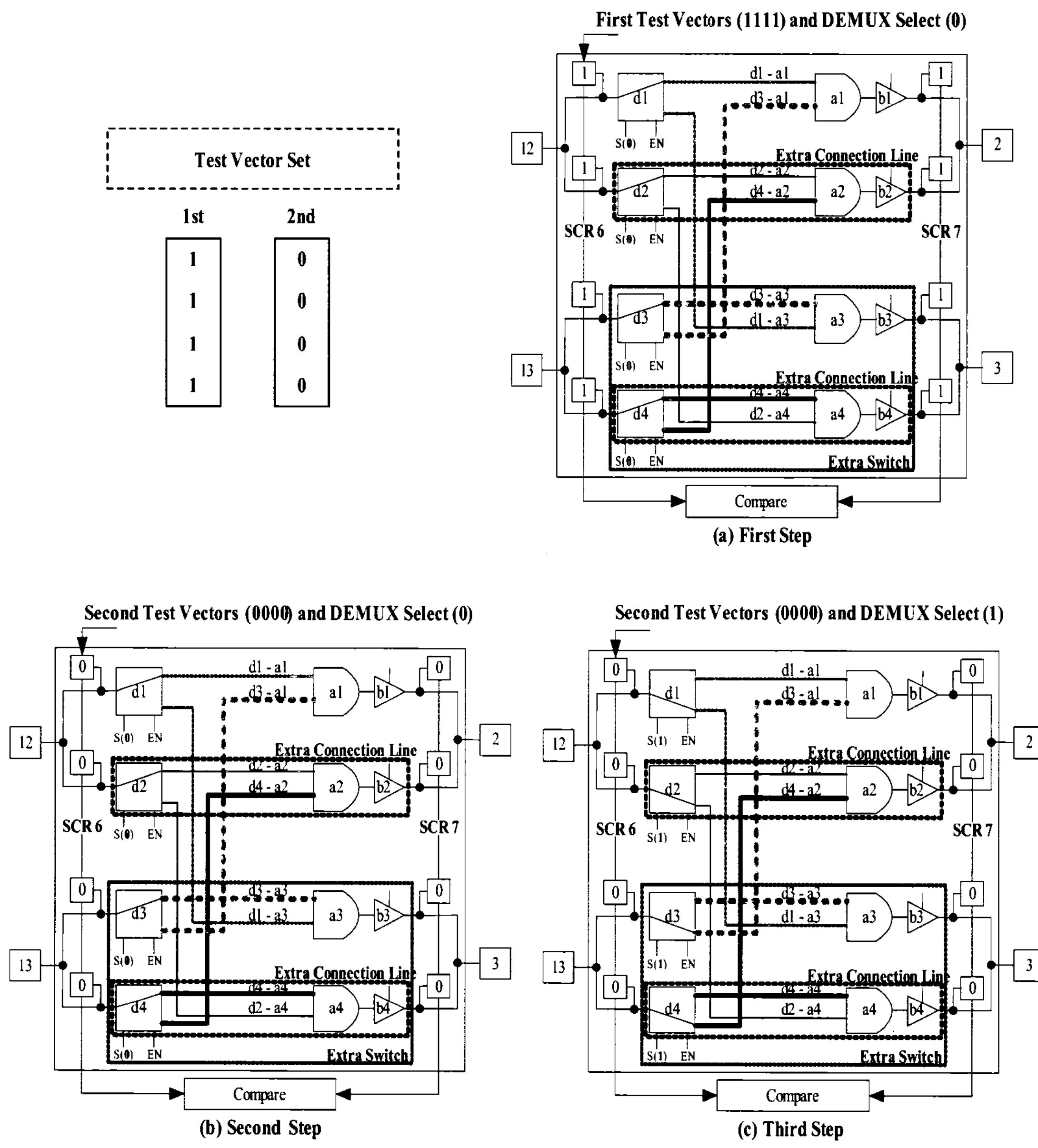


Figure 5.8 Generation of the Test Vector Set/Storing Scanning Results into SCR7

This example assumes that there are no faults in the SC. The first test vector '1111' generated by the FLFRP is stored in the SCR6. This vector is fed into each connection line as shown in Figure 5.8. If the data bits of the SCR7 has '0', then that AND gate should be discarded since a line connected to that AND has s-a-0. The DEMUX select needs to be applied either '0' or '1' since the stuck-at o fault is only considered in this step. The second test vector '0000' generated by the FLFRP is stored in the SCR6. If the data bits of the SCR7 has '1', then that particular line should have s-a-1 fault after applying both logic value '0' and '1' in the DEMUX select. This is the same as a simple stuck-at fault test in the 2-input AND gate. The scanning result '1111', '0000' of the SCR7 is transmitted through the diagnosis/repair bus to the RB.

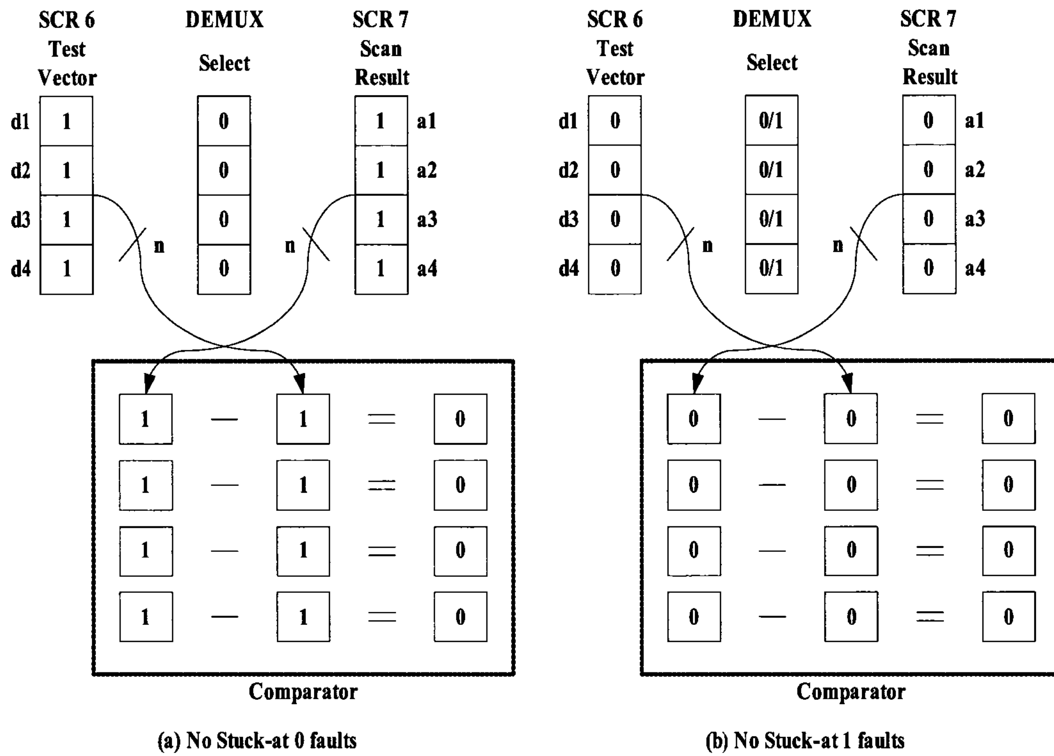


Figure 5.9 Comparator Operation for Finding Faults on an SC without Faults

Figure 5.9 illustrates the comparison operation for finding faults in an SC without faults. Figure 5.10 shows the fault diagnosis and location of a simple example with faults. After getting all data in the RB, these data are compared with the RD entries. For the compare operation in Figure 5.9, the RD receives the data from the SCR6, the RB gets the data from the SCR7, and the RD is subtracted from RB bit by bit. As shown in Figure 5.9, all results from this bit-by-bit 'minus' operation are '0's, since

there are no faults in the connection lines. An example of a circuit with some faults is shown in Figure 5.10. The SC structure is the same as in the previous example, but there exist multiple faults (s-a-0 fault and s-a-1 fault) in the lines in this case. All multiple faults are diagnosed and located using the same method explained above.

In this case, there are multiple faults in the SC. The connection d4-a2 and d1-a1 have s-a-0 and s-a-1 faults, respectively. When a test vector '1111' is inserted from the SCR6, the result '1011' is obtained in the SCR7. The second bit of SCR7 will be '0' since that d4-a2 line has s-a-0. When a test vector '0000' is inserted from the SCR6, the result '1000' is obtained in the SCR7. The first bit of SCR7 will be '1' since that d1-a1 line has s-a-1 when the DEMUX select is 0. It means that the first bit of the test vector, '0', cannot be transmitted to the first AND gate since that line is s-a-1. The fault detecting and locating process is based on comparisons as shown in Figure 5.10.

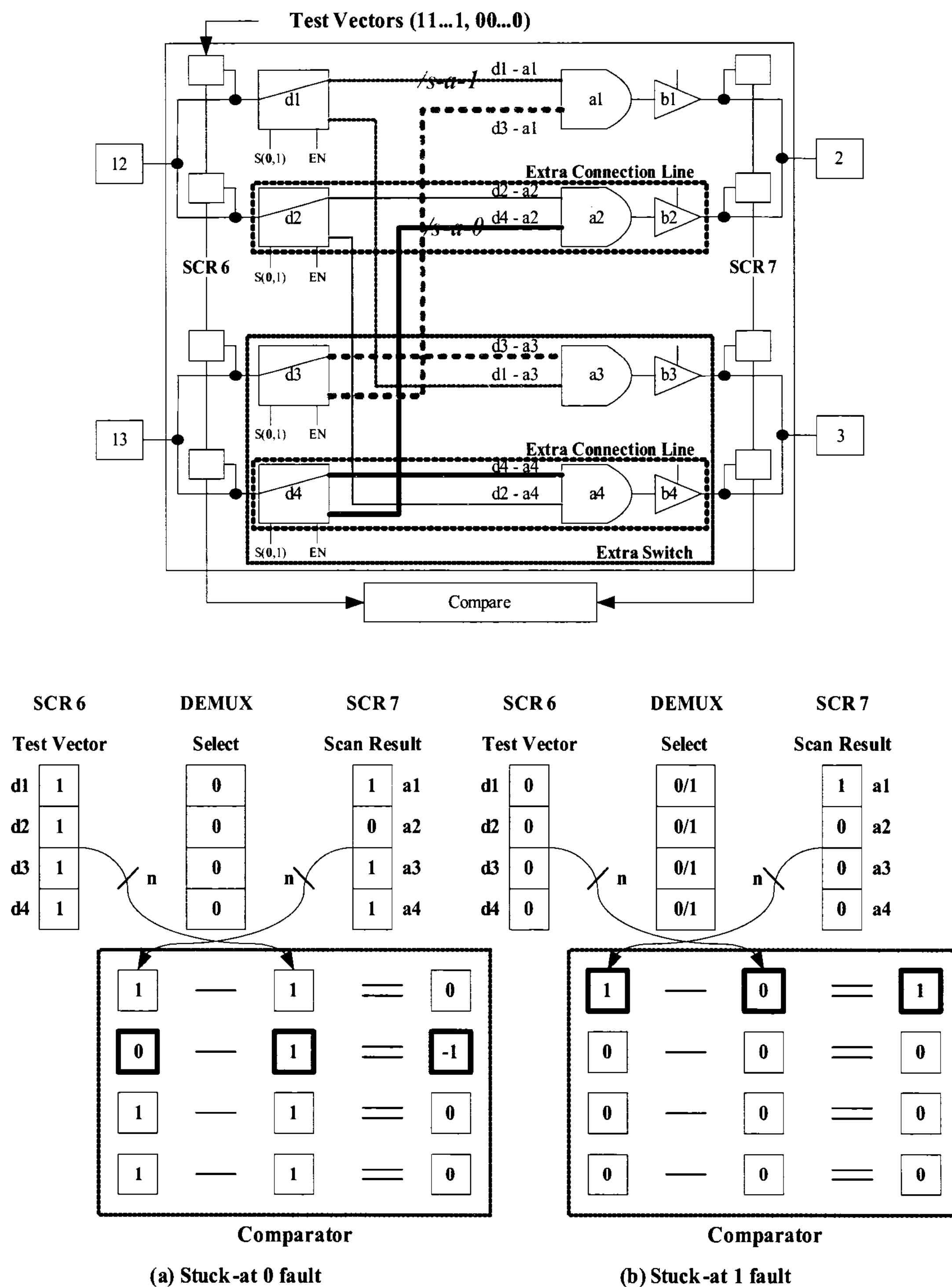


Figure 5.10 Fault Diagnosis/Location of an Example with Faults

5.2. Switching Circuit Self-Repairing Methodologies

A self-repairing methodology (line replacement method) is introduced in this section. This method should be applied after generating all test vectors and creating NSA, NSC, NR, and MCIR.

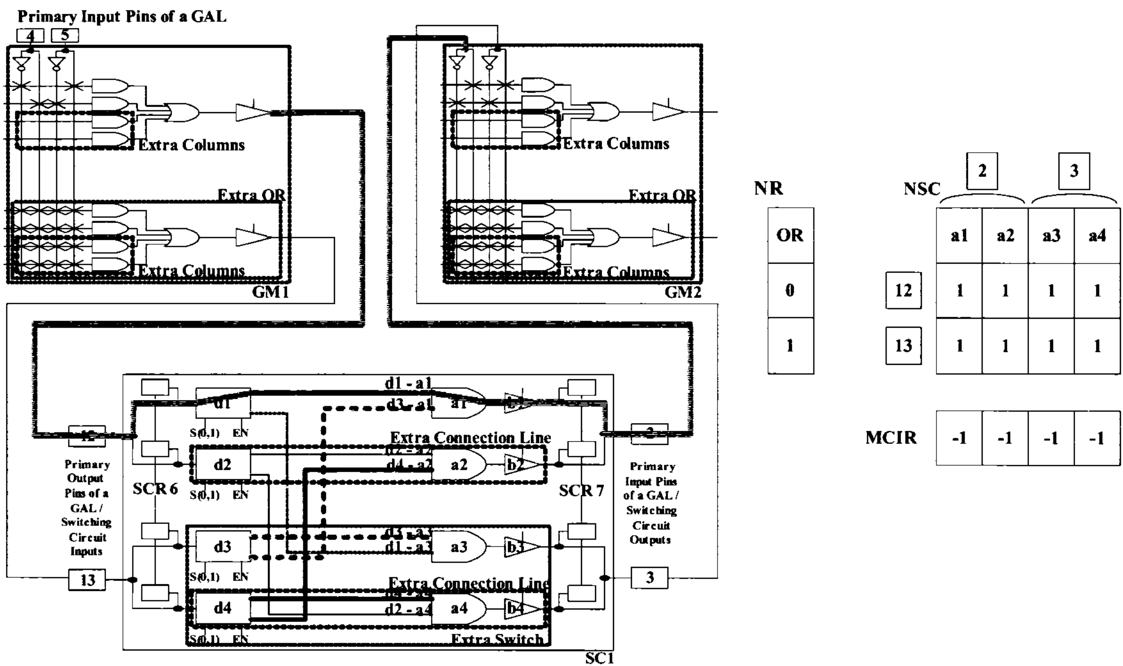
5.2.1. Line Replacement with Extra Lines

The line replacement method example with multiple faults (s-a-0 and s-a-1) is shown in Figure 5.11. This example does not have a faulty OR. The dotted boxes describe the extra devices (extra columns, extra OR, and extra line) in each part (AND plane of an OR, OR group of a GAL, and SC in a system). In Figure 5.11 (a), the GM1 is connected to GM2 through input pin 12 and output pin 2 (d1-a1 connection line) in the SC1. A function of the first OR in GM1 will affect to the function of the first OR in GM2. Thus, the first bit of the NR has logic value '0' which means that this first OR of the GM1 is being used. The d2-c2 is an extra line for d1-a1 line and d4-a4 is an extra line for d3-a3 in the SC1. There are no faults at first step, thus the NSC has all logic value 1s. The MCIR is initialized as logic value '-1' as available connections

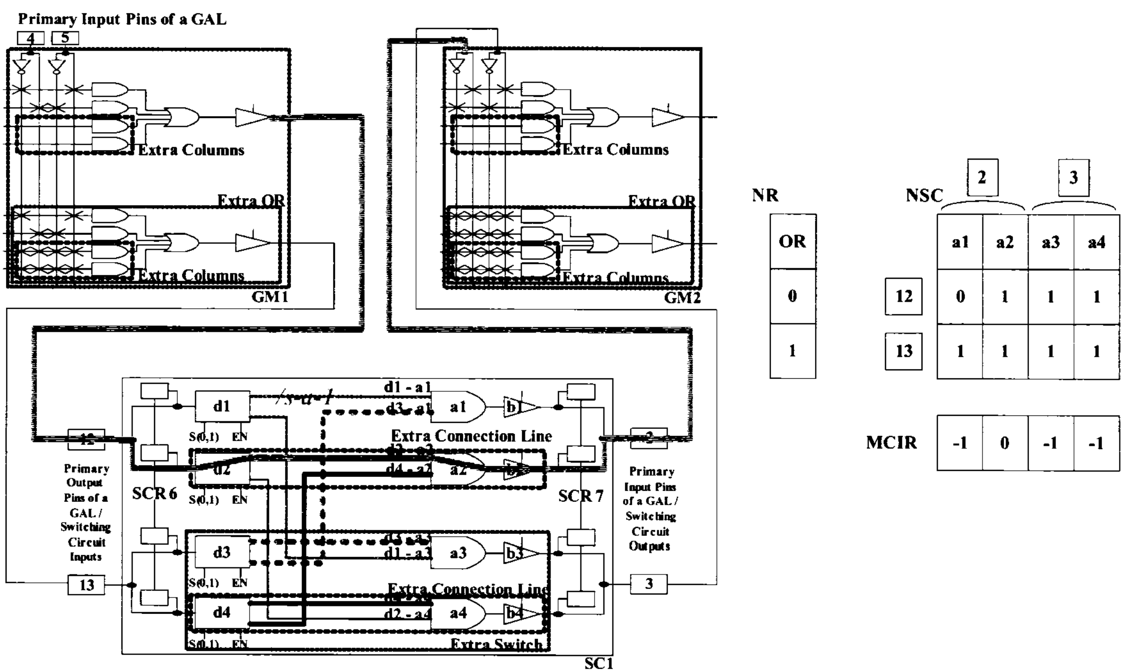
with the AND gate in an SC1. In Figure 5.11 (b), s-a-1 fault is occurred on d1-a1 line. The NSA detect s-a-1 fault and update the NSC as logic value '0'. The intersection of 13 & a1 of the NSC has logic value '1' since the d3-a1 connection line is still available even though the d1-a1 has s-a-1 fault. It is for keeping the pin-to-pin configuration so that the d3-a1 line can be used when the first OR is faulty and that OR is replaced with the second OR in GM1. According to the s-a-1 fault on the d1-a1 line, that line will be replaced with the d2-a2 extra line and the second bit of the MCIR is updated as logic value '0' as being connected. In Figure 5.11 (c), an s-a-0 fault has occurred on the d4-a2 connection line. Thus, the a2 AND gate cannot be used which means that all inputs of the a2 should be disconnected. The b1 which is tri-state buffer will block the output signal from the a2 AND gate, then the a2 column of the NSC has all logic value '0'. The second bit of the MCIR is updated as logic value '-2' as every connection through the a2 is not available. Now, the d2-a2 connection line cannot be used since the a2 is no more useful. Thus, the extra OR of the GM1 will be used for replacing the original programmed OR in the GM1 (first OR group) and makes a new connection with the d3-a1 connection line. Finally, the SC1 keeps the output pin 2 and will not affect to the

GM2 function. The MCIR is updated as logic value '0' in the first bit for informing what the d3-a1 connection (input pin 13 to output pin 2 connection) is being used. The NR is updated as logic value '-1' since the first OR in the GM1 is now unavailable.

The line replacement method example with a faulty OR is shown in Figure 5.12. In this example, there are no faults in the SC1. The MCIR informs that the a1 is being used for connecting the GM1 to the GM2 through input pin 12 and output pin 2 of the SC1 with the information of the NR register. If the OR programmed in the GM1 is faulty even though there are no faults in lines of the SC1, that OR should be replaced with an extra OR in that GAL, GM1, if an extra OR is available. As shown in Figure 5.12, the OR is replaced with the second OR and rerouted with the d3-a1 line. The self-repairing of the system can be done with both extra ORs and extra lines.

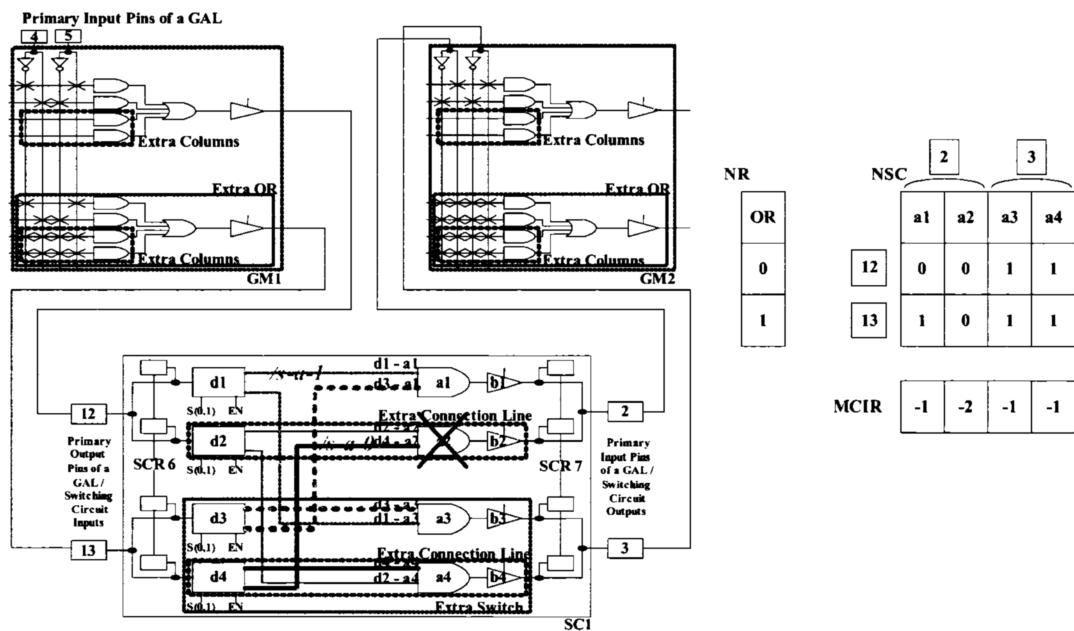


(a) Initial State (No Stuck-at faults ; 12 - a1 - 2 Connection)

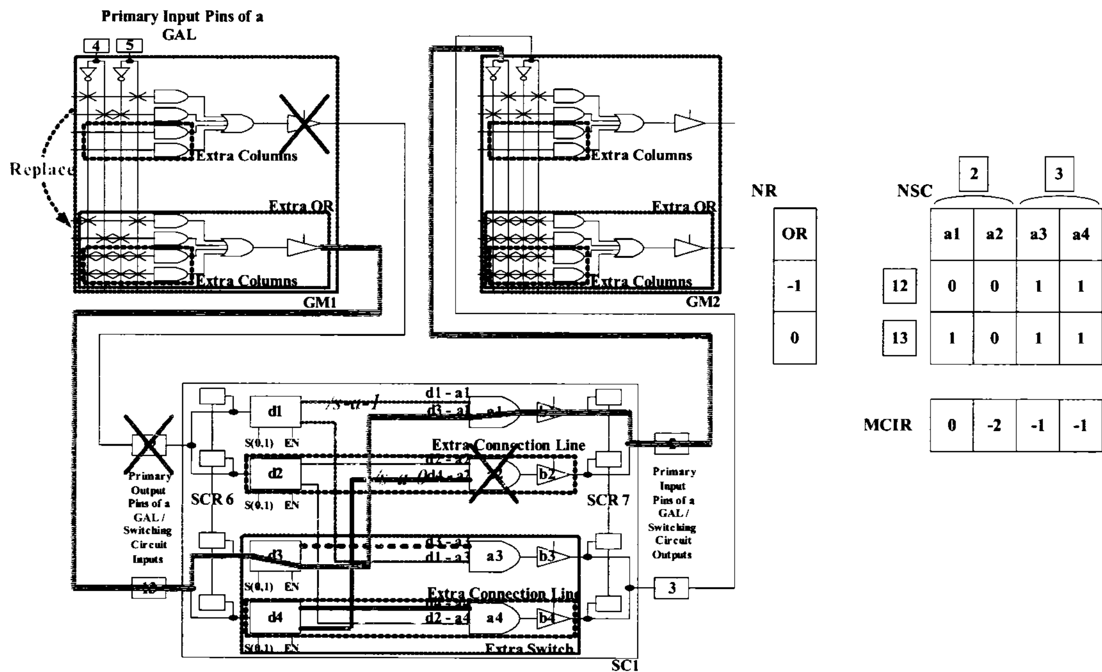


(b) Second State (Stuck-at 1 faults in d1-a1 line; 12 - a2 - 2 Connection)

Figure 5.11 A Line Replacement Method Example of Multiple Faults without a Faulty OR

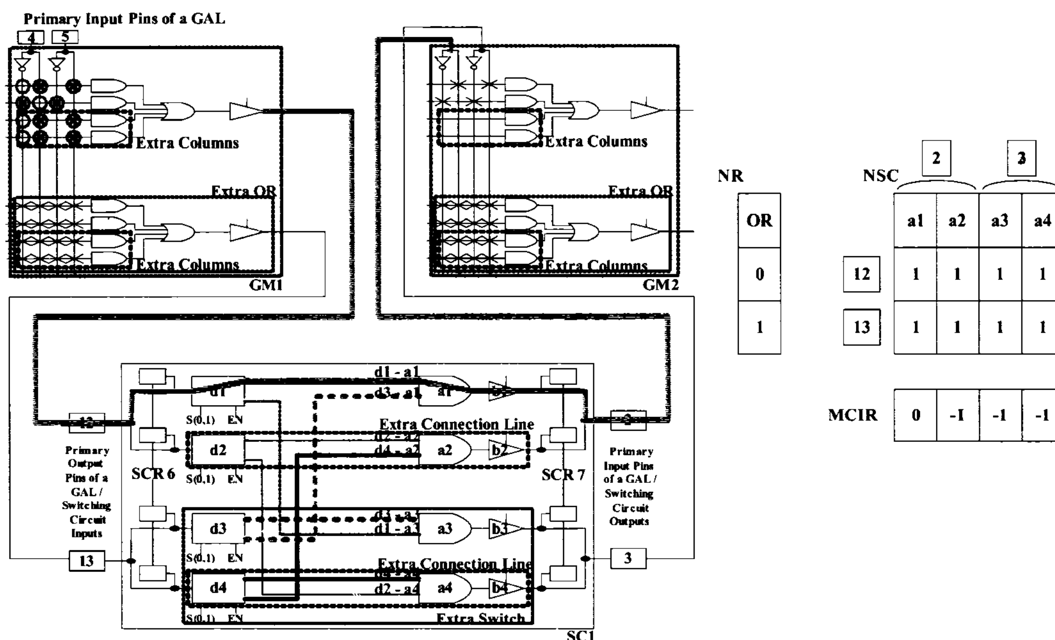


(c) Third State (Stuck-at 0 faults in d 4-a2 line)

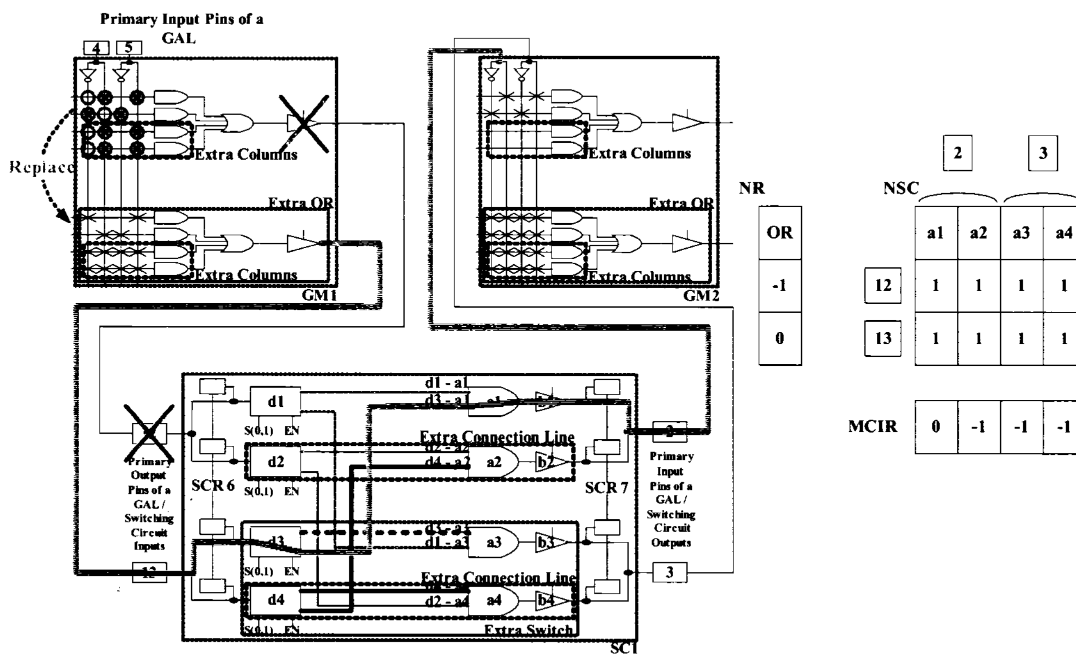


(d) Final State (13 - a1 - 2 Connection)

Figure 5.11 Line Replacement Method Example of Multiple Faults without a Faulty OR (Continued)



(a) Initial State (No faults in a SC & multiple faults in an AND plane of the GM 1; 12 - a1 - 2 Connection)



(b) Final State (Replace first OR group of GM 1 with second OR group of GM1; 13 - a1 - 2 Connection)

Figure 5.12 A Line Replacement Method Example with a Faulty OR

6. New Hardware Prototype and Simulator for the Ultra Reliable Computing Systems

This chapter describes the simulator used in this research in depth. In depth analysis of the simulator's algorithm structure and its development as well as how the simulator is used is presented.

6.1. Analysis and Synthesis of the Hardwired Test and the Diagnosis Algorithm

In order to prove the self-test and self-repair algorithm which is intended for hardware programming, a software program was created to simulate the hardware. The objective of the simulator is to verify how each algorithm improves the performance of a system in realistic environment. The program is made to apply different failure rate and fault limit to support GALs and SCs of different sizes and structures. The simulator can also be configured with different size GALs and SCs for different test cases which may have different sized GALs, and the SCs' size that varies with the design of the system.

First, the simulator's flow chart is explained here. The numbers inside the processes on the flow chart represent the index of the detail flow chart. Processes without this index number indicate that it is a basic process and has a brief description inside of the process. The simulator was built using Microsoft Visual Studio 2005 as the tool and MFC (Microsoft Foundation Classes).

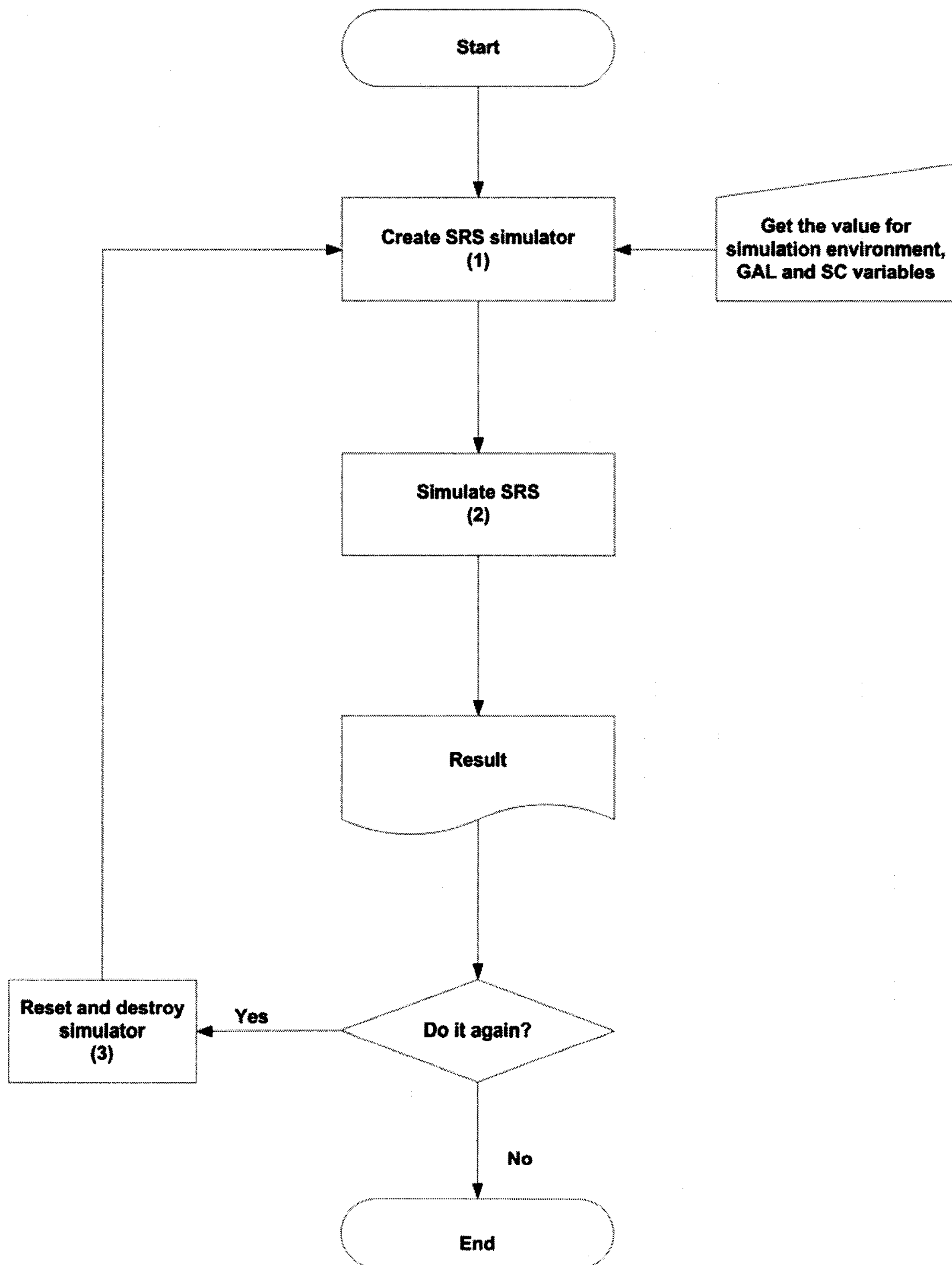


Figure 6.1 Overall Process

After the program is launched, GALs are created first according to the Self-Repair System (SRS) design, and then the SCs are created if the GALs need to be connected. Once the algorithm, GAL, and SC' fault limit and failure rate is set the simulation can be started. Existing GALs and SCs may be deleted once the result is obtained. Or, the GALs and SCs can be reconfigured with another data set to be used for another test. Figure 6.1 illustrates the overall building process of the simulator.

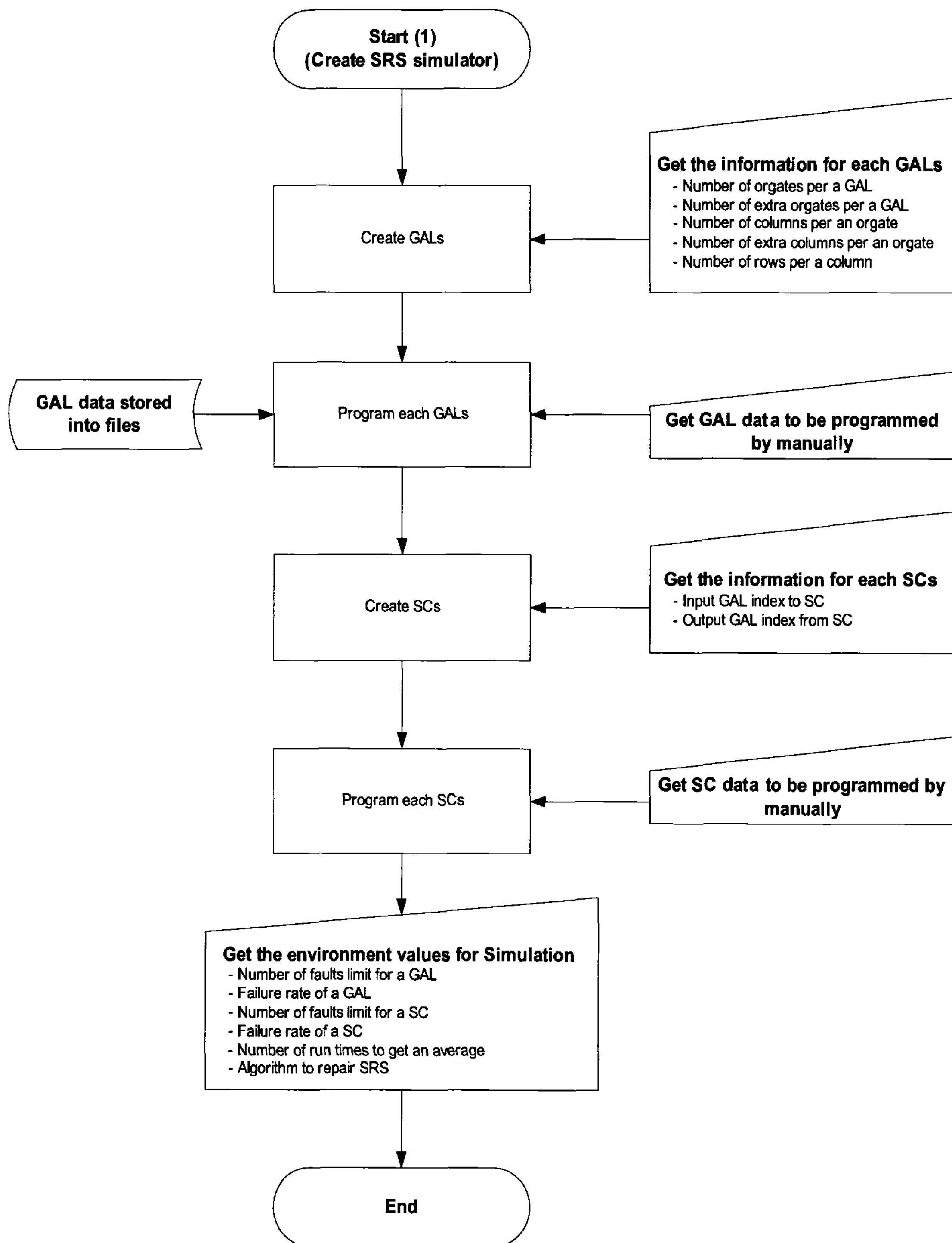


Figure 6.2 GAL and SC Generation

Figure 6.2 shows the order and how GALs and SCs are created. First, GALs are created to the specifications and then programmed. The programming can be done manually or read from a data file such as a JEDEC file. SCs are created after all GALs are created. An SC is created when a connection between GALs is needed. Once the index of the input GAL and output GAL is put in the configuration, the size of the SC is determined by the number of output pins on the input GAL. An SC can then be programmed manually. After all components are put in place and the variables (type of algorithm, fault limit, failure rate, and run time) are set, the program is ready to start the simulation.

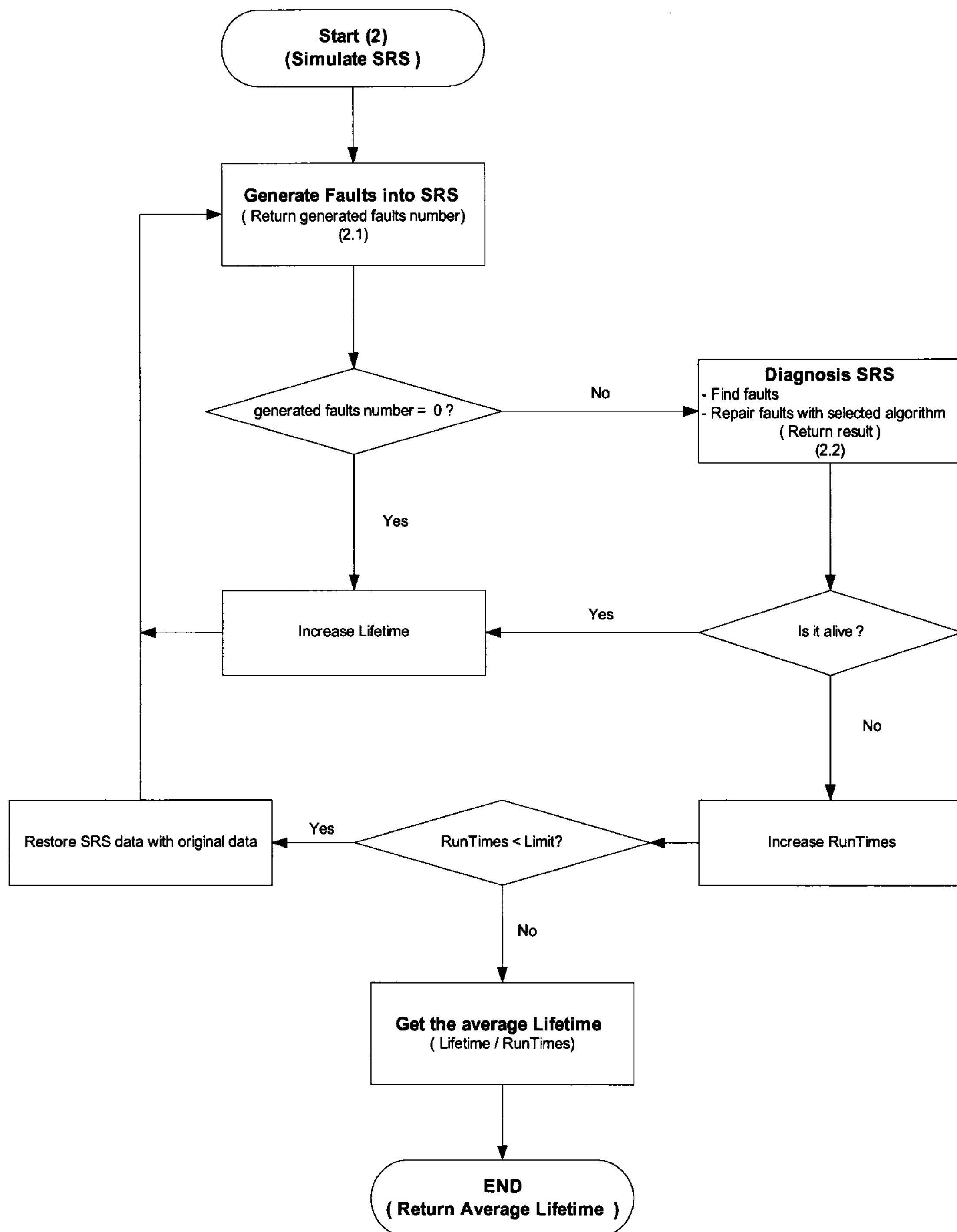


Figure 6.3 System Lifetime Generation

Figure 6.3 is a flow chart for generating the lifetime of the SRS. A fault is generated by the fault limit and the failure rate given to each GAL and SC. If no fault occurs the lifetime is incremented, and then run the fault generation simulation again (2.1). If fault occurs, it attempts to diagnose (2.2) and repair with the given algorithm. If the fault is repaired, it increments the lifetime and goes back to the fault generation simulation (2.1). If repair is not possible with the given algorithm all components on the SRS is reset and run again as many times as the specified run time. The Average Lifetime is obtained by dividing the lifetime by the run time.

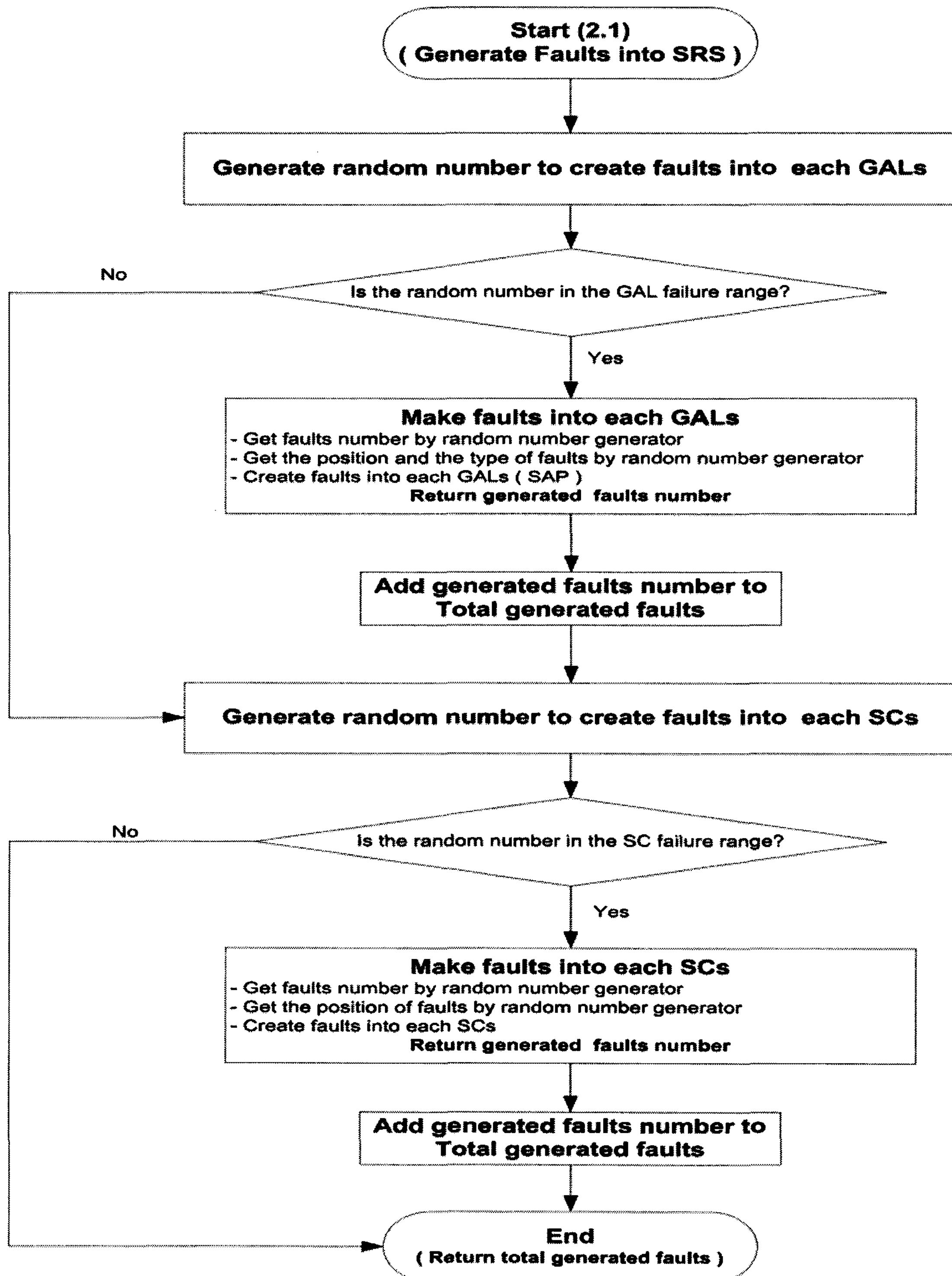


Figure 6.4 Fault Generation Simulation

Figure 6.4 is the fault generation simulation flow chart. First, faults are generated on each GAL. The srand() function is used as the random number generator and the current system time is used as the seed key. A random number is generated and if the number falls within the failure rate range then another random number is generated to determine the number of faults. Another random number is then generated to determine the fault location and the type of fault. Faults on SC is generated the same way. If a generated random number does not fall within the fault range of both GAL and SC, the processor returns 0 and returns the fault number otherwise.

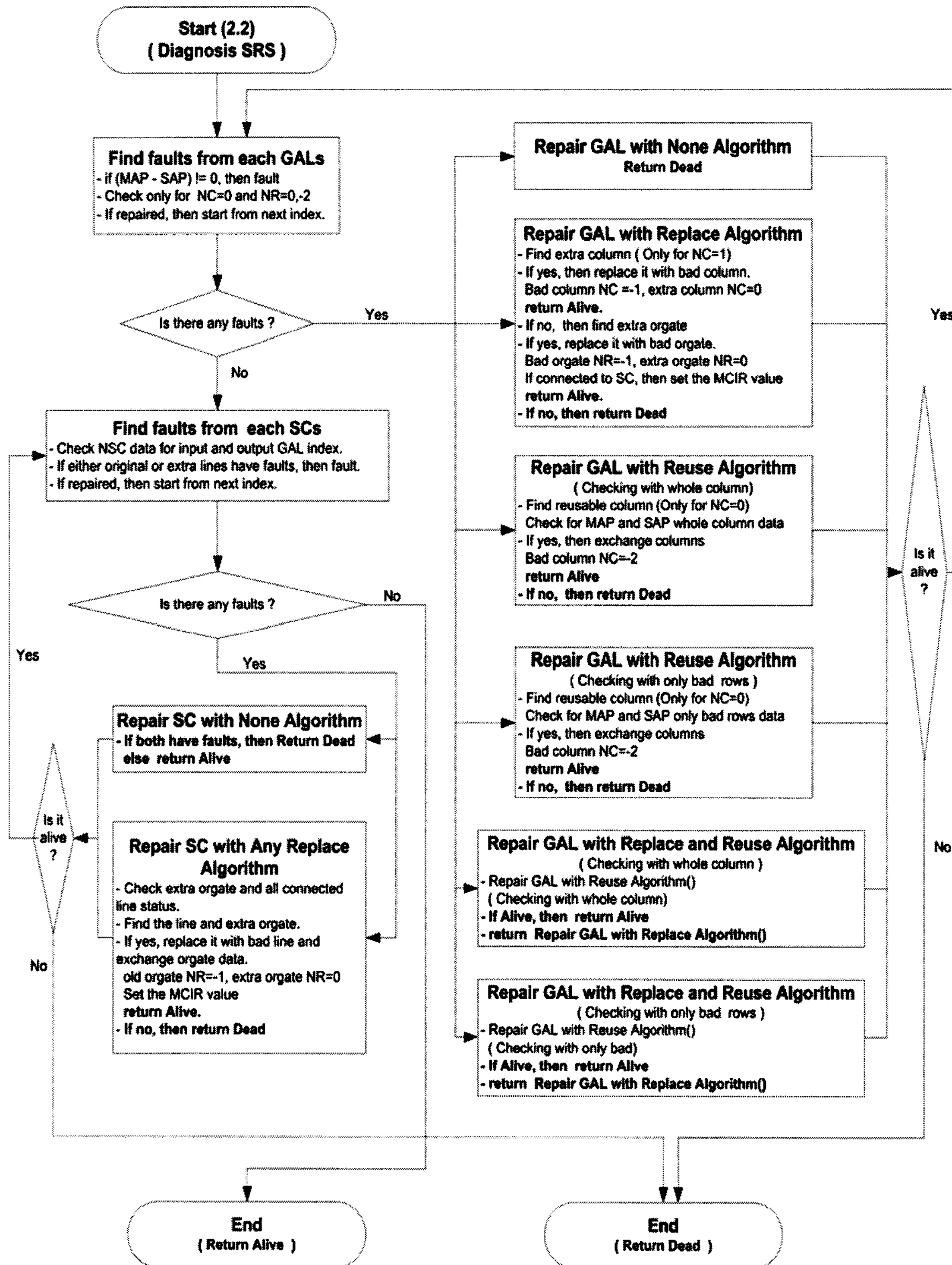


Figure 6.5 System Diagnosis Process

Figure 6.5 is the system diagnosis process. It first detects for faults and then repairs the fault according to the given algorithm. In a GAL fault is detected by searching '-' operation in MAP and SAP, and when the fault is detected it is repaired by the given algorithm. If repair was not possible it returns "Dead" and continues to search for faults in SC otherwise. In the case where extra line is used, the repair process begins if both the original and the extra line has faults. When the repair for both GAL and SC is successful, the process returns "Alive" and returns "Dead" otherwise. The details of the algorithm can be found in Chapter 4.

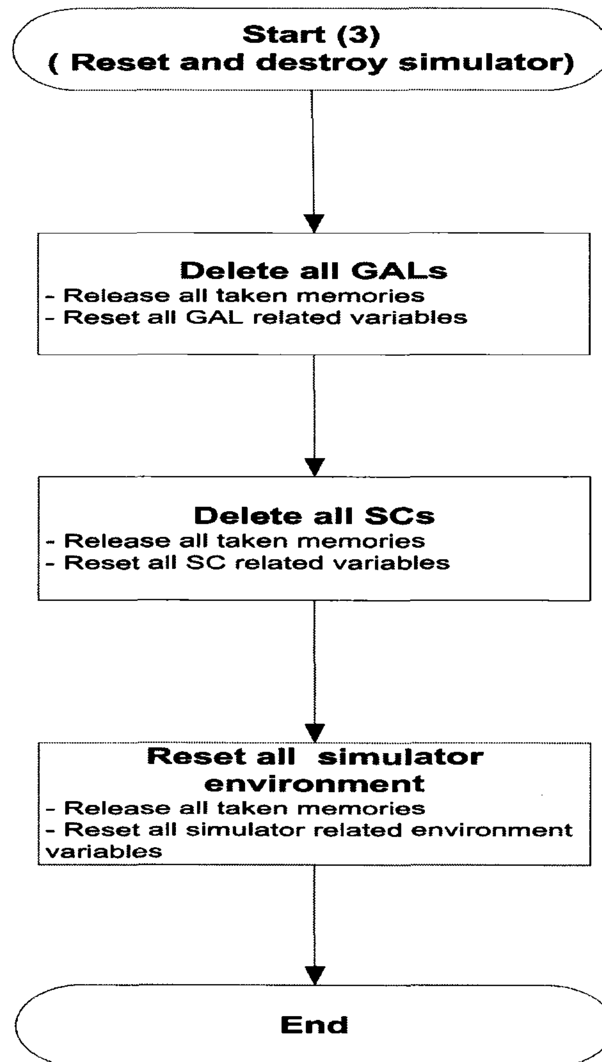


Figure 6.6 Simulator Reset Process

Figure 6.6 shows the SRG reset process. All of the memory used are released, variables are reset, and the temporary memory for GAL and SC are reset, and then it prepares for the next simulation.

6.2. Introduction to the Computer-Based Simulator

This section briefly explains the usage of the simulator program, and verifies the legitimacy of the algorithm and the hardware simulation by running the simulation with the testing vectors of each algorithm. Also described is how modules (GAL and SC) are tested individually on the simulator. Finally, how FPGA and ASIC type system is crated in the simulator and how it carries the implementation of what this research suggests. In the FPGA-like structure, an SC must be placed between GAL modules. On the other hand, in the ASIC-like structure, a particular GAL module can be shared by other GAL modules with an SC. Thus, the ASIC-like design type can reduce the number of SCs in a system design and also the size of each module can be different from other GALs' size. However, the ASIC-like design may not be flexible in designing a system since each module is fixed in terms of extra devices such as extra columns in an AND plane of a GAL module, extra ORs in a GAL module, or extra lines in an SC.

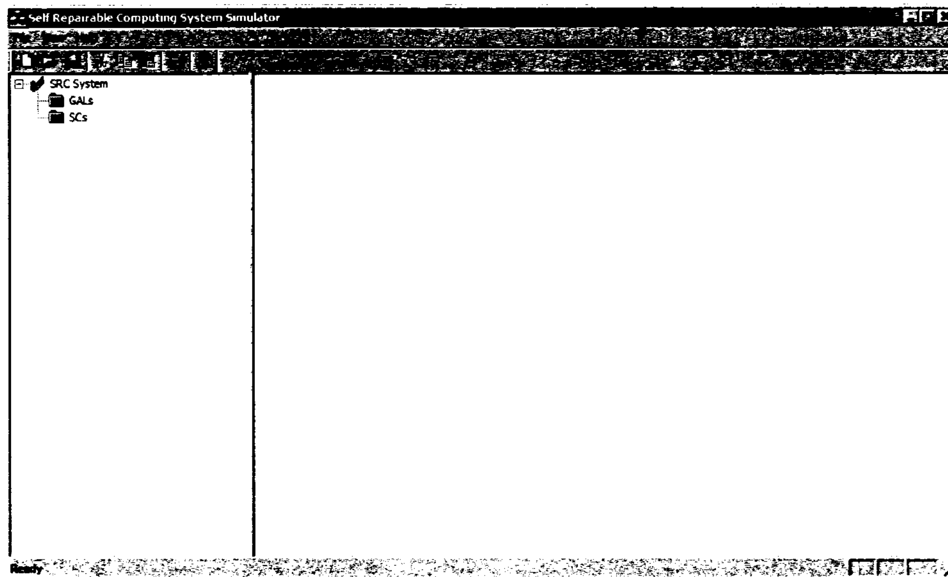


Figure 6.7 Initial Screen of the Simulator

Figure 6.7 shows the initial screen of the simulator.

To create a GAL, right-click the mouse on the GAL folder and select "Add a GAL", and the dialog box appears (Figure 6.8).

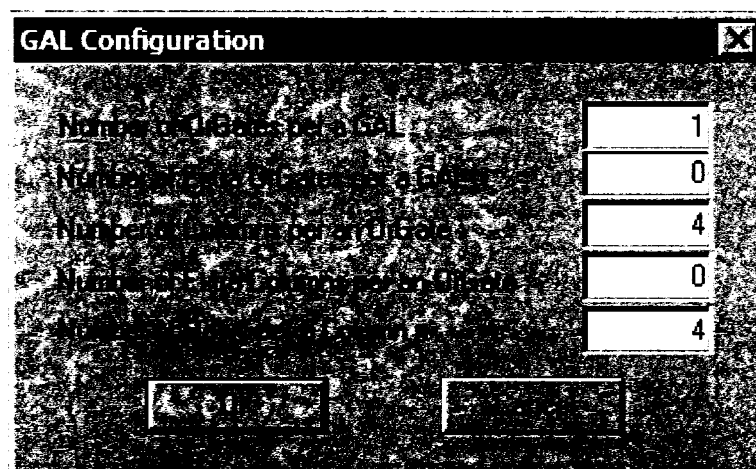


Figure 6.8 Creating GAL Options

In Figure 6.8, configure the size of GAL and extra devices. "Number of Or-gate per GAL" is the total number of OR-gates in the GAL, and "Number of Extra OR-gate" is the number of extra OR-gates out of the total OR-gates. "Number of Column per OR-gate" represents the total number of columns per OR-gate, and the Extra Columns are how many of the total columns are used as extra. "Number of Row per Column" represents the primary input.

Once a GAL is created, the left panel displays the GAL folder with indexed GAL icons, and the right panel displays the detail information. Each column on the right panel corresponds to OR-gate index, Column index of the OR-gate, MAP (M), and SAP (S). Index on column represents input (row) index, and NC represents the current state of the NC register.

To program the GAL, right-click the GAL to be configured on the left panel and select "Program GAL".

Figure 6.9 is the screen of a programmed GAL.

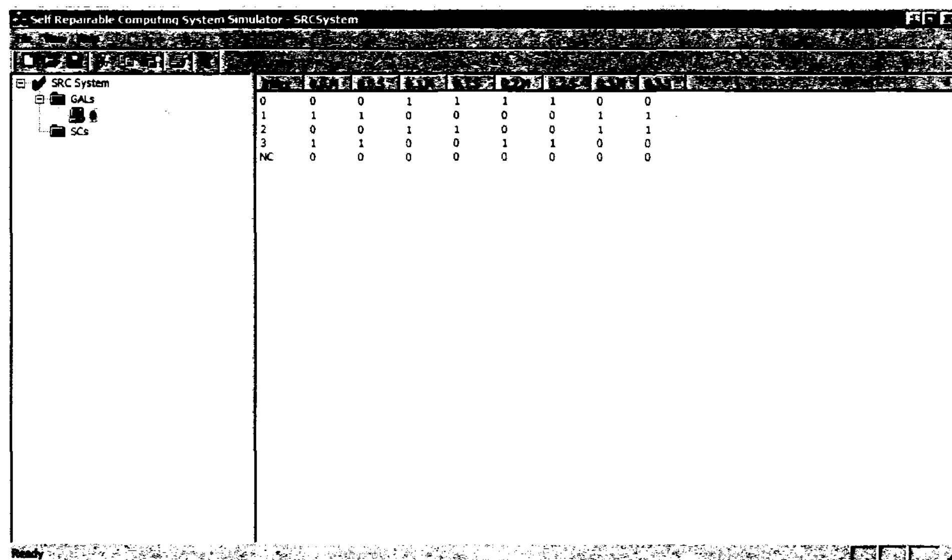


Figure 6.9 Initial MAP and SAP

Click on the GAL icon on the left panel and select "Make Faults into a GAL" to generate faults.

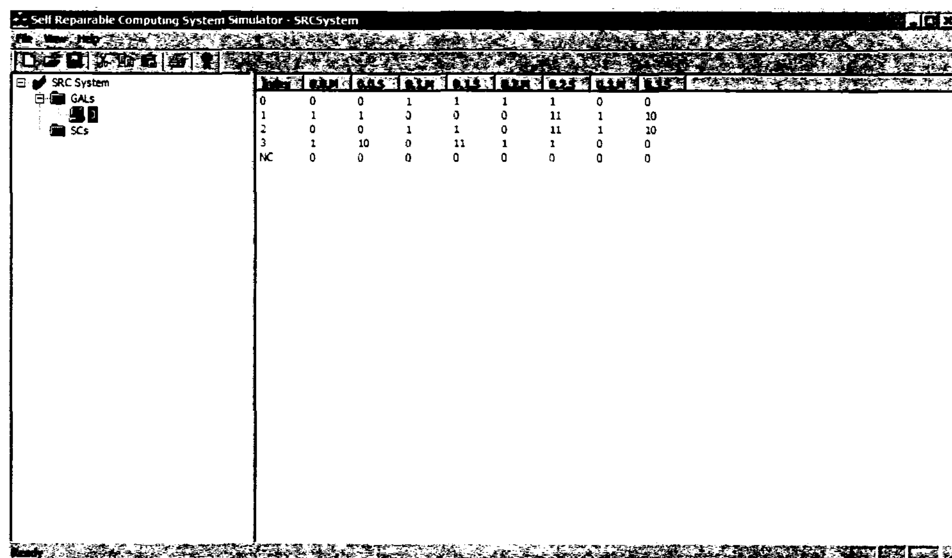


Figure 6.10 Added Faults

Figure 6.10 shows the screen of a programmed GAL with fault variables selected. The example shows 6 faults in Figure 6.10. Faults are indicated by the extra digit on the SAP column and if the extra digit is 0 it is a "stuck-at 0" fault and 1 is " stuck-at 1" fault.

In order to start simulation, the environment variable must be configured. Click on the SRG System icon and select "Configure..." then Figure 6.11 appears. Configure GAL's fault limit, failure rate, SC's fault limit, failure rate, run time, algorithm, and whether extra lines will be used or not.

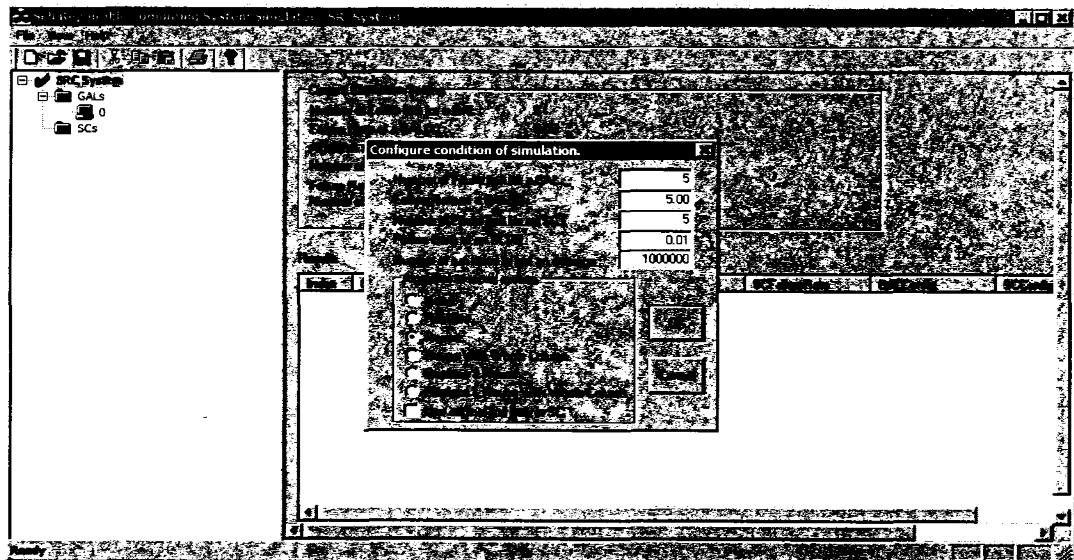


Figure 6.11 Simulation Configuration

From the left panel, when SRG System icon is clicked and “Diagnosis” is selected, chosen algorithm can be verified. The NC registers’ value is also displayed after a repair by the algorithm.

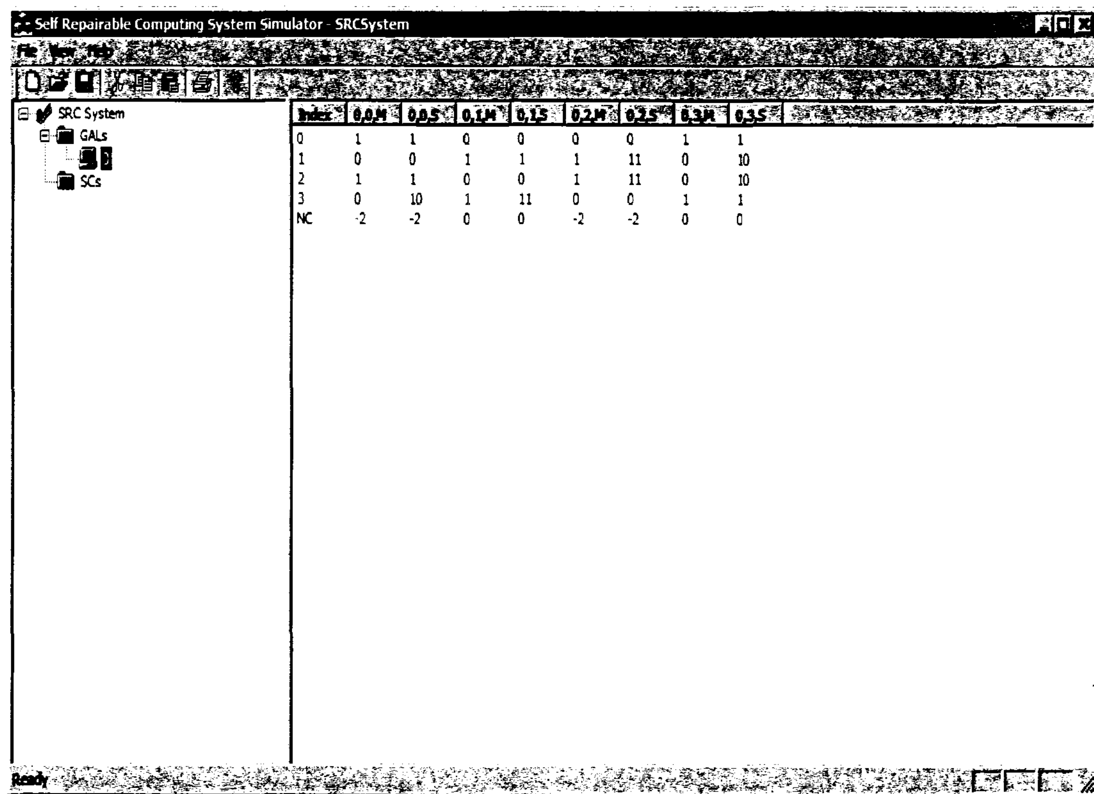


Figure 6.12 Simulation Result

Figure 6.13 is a sample of Replacement method simulation. Shown GAL has 1 OR-gate with 8 columns and 4 primary inputs. 4 of these columns are used as extra. A total of 6 faults are programmed for this simulation.

SRC System
GALs
0
SCs

Index	0.0M	0.0S	0.1M	0.1S	0.2M	0.2S	0.3M	0.3S	0.4M	0.4S	0.5M	0.5S	0.6M	0.6S	0.7M	0.7S
0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
1	1	1	0	0	0	11	1	10	1	1	1	1	1	1	1	1
2	0	0	1	1	0	11	1	10	1	1	1	1	1	1	1	1
3	1	10	0	11	1	1	0	0	1	1	1	1	1	1	1	1
NC	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Ready

Figure 6.13 Initial Screen of the Replacement Method

Figure 6.14 is the result of the simulation run in Figure 6.13. All columns with faults have been programmed (replaced) to the extra columns and all the faulty columns are programmed to 1 and disabled as NC value is changed to -1.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
SCs

Index	0,0M	0,0S	0,1M	0,1S	0,2M	0,2S	0,3M	0,3S	0,4M	0,4S	0,5M	0,5S	0,6M	0,6S	0,7M	0,7S
0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1
2	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1
3	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0
NC	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0

Ready

Figure 6.14 Replacement Method Result

Figure 6.15 shows the Column Re-Use with extra column algorithm test. A GAL has 1 OR-gate with 8 columns and 4 primary inputs. 4 of the 8 columns are used as extra columns. Column 0 has 'stuck-at 1' fault at row 2 and 'stuck-at 0' fault at row 3. Column 1 has 'stuck-at 0' fault at row 1, 2, and 'stuck-at 1' fault at row 3. Column 1 has 'stuck-at 0' fault at row 1, 2, and 'stuck-at 1' at row 3. Column 2 has 'stuck-at 0' fault at row 0, 3, and 'stuck-at 1' at row 1 and 2. No fault occurred on Column3.

SRC System
GALs
0
SCs

Index	0,0M	0,0S	0,1M	0,1S	0,2M	0,2S	0,3M	0,3S	0,4M	0,4S	0,5M	0,5S	0,6M	0,6S	0,7M	0,7S
0	0	0	1	1	1	10	0	0	1	1	1	1	1	1	1	1
1	1	1	0	10	0	11	1	1	1	1	1	1	1	1	1	1
2	0	11	1	10	0	11	1	1	1	1	1	1	1	1	1	1
3	1	10	0	11	1	10	0	0	1	1	1	1	1	1	1	1
NC	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 6.15 Initial Screen of Column Re-Use with Extra Columns Method

Figure 6.16 is the result of the Column Re-Use with extra column algorithm repair simulation run in Figure 6.15.

Column 0 became '0110' after the fault and Re-used with column 3 where the value was the same, '0110'. Column 1 became '1001' after the fault and Re-used with column 2. However, column 2 did not have a matching column so it used an extra column to repair. All the faults have been repaired at this point. The result shows that column 0 and 1 were repaired by re-use and column 2 was repaired by replacement by using an extra column.

Index	0,0M	0,0S	0,1M	0,1S	0,2M	0,2S	0,3M	0,3S	0,4M	0,4S	0,5M	0,5S	0,6M	0,6S	0,7M	0,7S
0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
1	1	1	0	10	1	1	1	1	0	0	1	1	1	1	1	1
2	1	11	0	10	1	1	0	0	1	1	1	1	1	1	1	1
3	0	10	1	11	1	1	1	1	0	0	1	1	1	1	1	1
NC	-2	-2	-2	-2	-1	-1	0	0	0	0	1	1	1	1	1	1

Figure 6.16 Column-Re-Use with Extra Columns Method Result

Figure 6.17 shows the Cell Re-Use with extra column algorithm test. A GAL has 1 OR-gate with 8 columns and 4 primary inputs. 4 of the 8 columns are used as extra columns. Column 0 has 'stuck-at 1' fault at row 2, and 'stuck-at 0' at row 3. Column 1 has 'stuck-at 0' fault at row 1 & 2, and 'stuck-at 1' fault at row 3. Column 2 has 'stuck-at 0' at row 0 & 3, and 'stuck-at 1' at row 1 & 2. Column has no faults.

SRC System
GALs
0
SCs

Index	0,0,M	0,0,S	0,1,M	0,1,S	0,2,M	0,2,S	0,3,M	0,3,S	0,4,M	0,4,S	0,5,M	0,5,S	0,6,M	0,6,S	0,7,M	0,7,S
0	0	0	1	1	1	10	0	0	1	1	1	1	1	1	1	1
1	1	1	0	10	0	11	1	1	1	1	1	1	1	1	1	1
2	0	11	1	10	0	11	1	1	1	1	1	1	1	1	1	1
3	1	10	0	11	1	10	0	0	1	1	1	1	1	1	1	1
NC	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Ready

Figure 6.17 Initial Screen of Cell Re-Use Method

Figure 6.18 shows the result of the repair using the Cell Re-Use with extra column algorithm shown in Figure 6.17

Column 0 has 'stuck-at 0' fault at row 3 and 'stuck-at 1' fault at row 2, thus this column is reused with column 1 which has 1 and 0 at row 2 and 3, respectively. On the second step, since column 1 is already reused by column 0, the faults on row1, 2, and 3 are detected. Thus, the column 1 is reused with column 2 which has matching data on row 1, 2, and 3.

Since column 2 also exchanged data with column 1, it finds the matching rows on column 3 for the rows with detected faults, and then reuses with column 3. Column 3 did not have any faults, thus no additional repair is needed. Columns 0, 1, and 2 were reused and the corresponding NC registers are changed to -2, and column 3 is still usable and no extra column was used.

Self Repairable Computing System Simulator - SRCSystem

SRC System
GALs
SCs

Index	0,0,M	0,0,S	0,1,M	0,1,S	0,2,M	0,2,S	0,3,M	0,3,S	0,4,M	0,4,S	0,5,M	0,5,S	0,6,M	0,6,S	0,7,M	0,7,S
0	1	1	1	1	0	10	0	0	1	1	1	1	1	1	1	1
1	0	0	0	10	1	11	1	1	1	1	1	1	1	1	1	1
2	1	11	0	10	1	11	0	0	1	1	1	1	1	1	1	1
3	0	10	1	11	0	10	1	1	1	1	1	1	1	1	1	1
NC	-2	-2	-2	-2	-2	-2	0	0	1	1	1	1	1	1	1	1

Ready

Figure 6.18 Cell Re-Use Method Result

Figure 6.19 is a sample test within a GAL using the algorithm with extra OR gates. The GAL is made up of 2 OR-gates with one of the two being an extra OR-gate. The OR-gates have 4 columns and no extra columns were used, and each column has 4 primary inputs.

OR-gate 0 has 'stuck-at 0' fault at column 0 & row 2, and 'stuck-at 1' fault at column 2 & row 0.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
SCs

Index	0,0,M	0,0,S	0,1,M	0,1,S	0,2,M	0,2,S	0,3,M	0,3,S	1,0,M	1,0,S	1,1,M	1,1,S	1,2,M	1,2,S	1,3,M	1,3,S
0	1	1	0	0	0	11	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
2	1	10	0	0	0	0	0	0	1	1	1	1	1	1	1	1
3	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1
NC	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Ready NUM

Figure 6.19 Extra OR-gate Initial Screen

Figure 6.20 is the result of the repair using an extra OR-gate in Figure 6.19 OR-gate 0 is disabled and all the data is replaced on to OR-gate 1. If a system has extra OR-gates, then the system can be repaired as shown in Figure 6.20, and improvement on the system performance can be expected.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
SCs

Index	0,0,M	0,0,S	0,1,M	0,1,S	0,2,M	0,2,S	0,3,M	0,3,S	1,0,M	1,0,S	1,1,M	1,1,S	1,2,M	1,2,S	1,3,M	1,3,S
0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1
NC	-1	-1	-1	-1	-1	-1	-1	-1	0	0	0	0	0	0	0	0

Ready NUM

Figure 6.20 Extra OR-gate Result

Figure 6.21 is a simulation of a switching circuit repair.

First, 2 GALs with 4 OR-gates and 2 columns on each OR-gate are created. Each GAL has 2 extra OR-gates and a switching circuit, SC 0 is put in place to connect GAL 0 and GAL 1. Figure 6.21 shows the connection status of SC 0.

MCIR InPin# and MCIR OutPin# show that SC 0 connects GAL 0 and GAL 1. OR-gate 0 and 1, each are connected using 0 to 0 and 1 to 1 line, and extra OR-gates 2 and 3 are not connected at this point. SC 0's output pin# 2 and 3 are currently disabled.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
0
1
SCs

Input GAL Index: 0 Output GAL Index: 1

InPin\OutPin	0(a1)	0e(a2)	1(a3)	1e(a4)	2(a5)	2e(a6)	3(a7)	3e(a8)
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
MCIR InPin#	0	1	2	3				
MCIR OutPin#	0	1	-1	-1				
SC OutPin#	0	1	2	3				
OutPin Status	1	1	0	0				
Total Con#	4							
Real Con#	2							
Extra Con#	2							
Killed Con#	0							

Ready NUM SCRL

Figure 6.21 Switching Circuit Initial Screen

Two ‘stuck-at 1’ faults are occurred on both the original line and the extra line on pin 0-to-pin 0 line (simply denoted by 0-to-0) in SC-0. No faults were created on the GAL.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
0
1
SCs

Input GAL Index: 0 Output GAL Index: 1

InPin\OutPin	0(a1)	0e(a2)	1(a3)	1e(a4)	2(a5)	2e(a6)	3(a7)	3e(a8)
0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

MCIR InPin#	0	1	2	3
MCIR OutPin#	0	1	-1	-1
SC OutPin#	0	1	2	3
OutPin Status	1	1	0	0

Total Con#	4
Real Con#	2
Extra Con#	2
Killed Con#	0

Ready NUM SCRL

Figure 6.22 SC 2

After the repair algorithm is applied, Figure 6.23 shows that no OR-gates are disabled on GAL 0, and all the data are moved to OR-gate 2. Faults occurred on line 0 to 0, thus OR-gate 0's data cannot be sent to SC 0's output pin#, therefore the data is moved to OR-gate 2 and starts to use OR-gate 2.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
0
1
SCs
0

Index	0,0,M	0,0,S	0,1,M	0,1,S	1,0,M	1,0,S	1,1,M	1,1,S	2,0,M	2,0,S	2,1,M	2,1,S	3,0,M	3,0,S	3,1,M	3,1,S
0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1
3	1	1	1	1	0	0	0	0	1	1	0	0	1	1	1	1
NC	-1	-1	-1	-1	0	0	0	0	0	0	0	0	1	1	1	1

Ready NUM SCRL

Figure 6.23 SC 3

Figure 6.24 shows the status of SC 0 after the repair. Observing MCIR input# and MCIR output#, 0 to 0 line is changed to -2, and discarded, and 2 to 0 line is used. Because OR-gate 0's data are moved to OR-gate 2 and SC 0's line connection is changed, there is no functional problems on GAL. The 1 to 1 line is keep being used. If a fault should occur on the line, it can switch to extra line, and if another fault should occur on the extra line, then extra OR-gate can even be used to repair the fault on the line if extra OR-gate is available.

Self Repairable Computing System Simulator - SRCSystem

File View Help

SRC System
GALs
0
1
SCs

Input GAL Index: 0 Output GAL Index: 1

InPin\OutPin	0(a1)	0e(a2)	1(a3)	1e(a4)	2(a5)	2e(a6)	3(a7)	3e(a8)
0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
MCIR InPin#	0	1	2	3				
MCIR OutPin#	-2	1	0	-1				
SC OutPin#	0	1	2	3				
OutPin Status	1	1	0	0				
Total Con#	4							
Real Con#	2							
Extra Con#	1							
Killed Con#	1							

Ready NUM SCRL

Figure 6.24 SC Final

Figure 6.25 and Figure 6.26 shows an ASIC type design with 4 GALs, 2 SCs designed by the simulator. Each GAL has 8 OR-gate x 8 columns x 32 inputs.

Figure 6.25 ASIC Initial Screen

The SC 0 simultaneously connects GAL0 & GAL1, and GAL 0 & GAL 2. The SC 1 connects GAL 1 & GAL 3. The SC 0 has 8 original lines and 8 extra lines which matches the number of OR-gates on GAL 0. The SC 1 has the same number of lines as SC 0, and matches the number of OR-gates on GAL 1.

Figure 6.26 ASIC 2

Figure 6.27 and Figure 6.28 shows a FPGA type design model with 4 GALs, 3 SCs designed by the simulator. Each GAL has 8 OR-gates x 8 columns x 32 inputs.

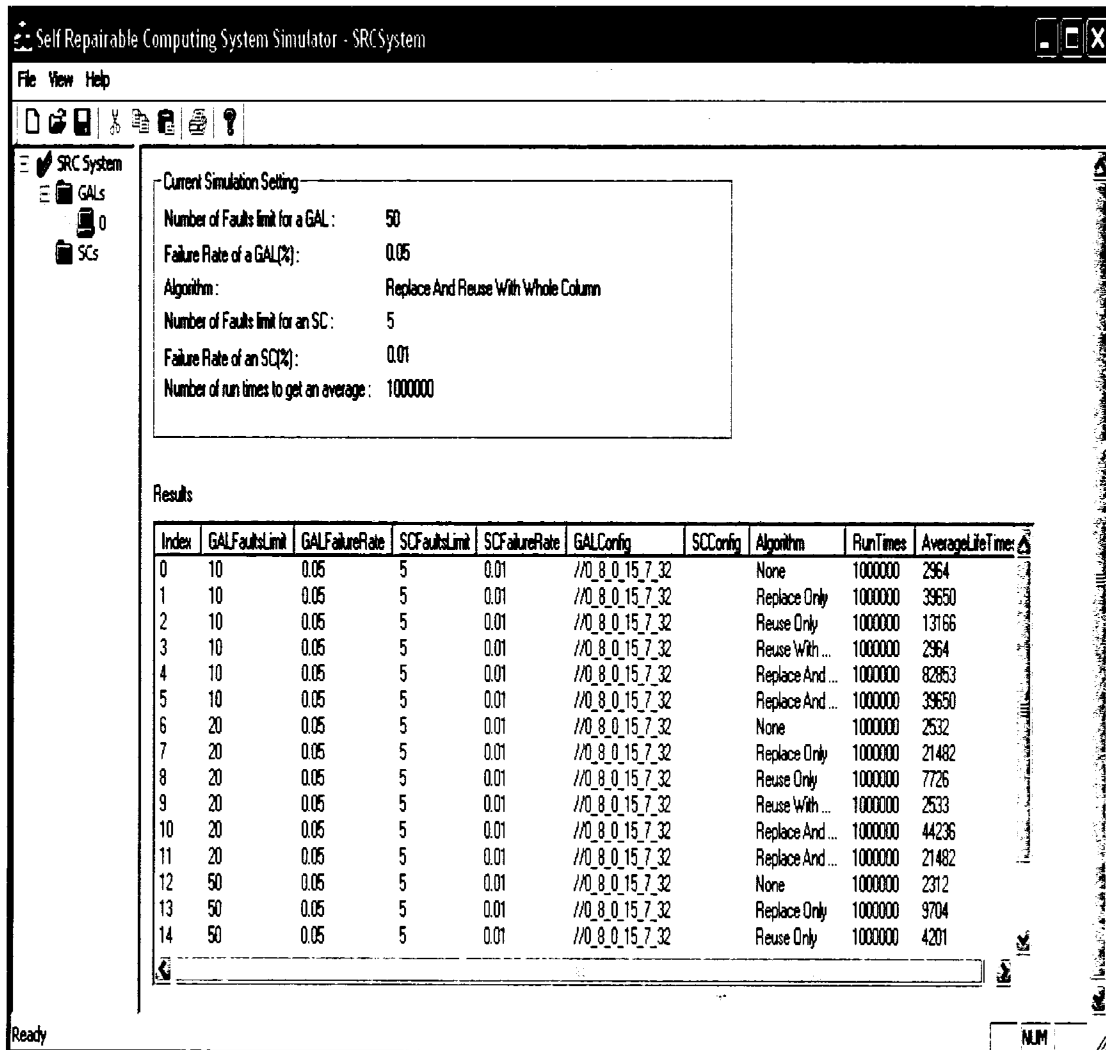
Index	NumOfOrgates	NumOfExtraOrgates	NumOfColumns	NumOfExtraColumns	NumOfRows
0	8	0	8	0	32
1	8	0	8	0	32
2	8	0	8	0	32
3	8	0	8	0	32

Figure 6.27 FPGA Initial

SC 0 and SC1 have 8 original lines and 8 extra lines which match the number of OR-gates on GAL 0. The SC 2 has the same number of lines as SC 0, and matches the number of OR-gates on GAL 1.

Figure 6.28 FPGA 2

This is the extent of the description on FPGA and ASIC type design simulation in this section, however, more detailed performance analysis is presented in Chapter 7.



Index	GALFaultsLimit	GALFailureRate	SCFaultsLimit	SCFailureRate	GALConfig	SCConfig	Algorithm	RunTimes	AverageLifeTime
0	10	0.05	5	0.01	//0_8_0_15_7_32		None	1000000	2964
1	10	0.05	5	0.01	//0_8_0_15_7_32		Replace Only	1000000	39650
2	10	0.05	5	0.01	//0_8_0_15_7_32		Reuse Only	1000000	13166
3	10	0.05	5	0.01	//0_8_0_15_7_32		Reuse With ...	1000000	2964
4	10	0.05	5	0.01	//0_8_0_15_7_32		Replace And ...	1000000	82853
5	10	0.05	5	0.01	//0_8_0_15_7_32		Replace And ...	1000000	39650
6	20	0.05	5	0.01	//0_8_0_15_7_32		None	1000000	2532
7	20	0.05	5	0.01	//0_8_0_15_7_32		Replace Only	1000000	21482
8	20	0.05	5	0.01	//0_8_0_15_7_32		Reuse Only	1000000	7726
9	20	0.05	5	0.01	//0_8_0_15_7_32		Reuse With ...	1000000	2533
10	20	0.05	5	0.01	//0_8_0_15_7_32		Replace And ...	1000000	44236
11	20	0.05	5	0.01	//0_8_0_15_7_32		Replace And ...	1000000	21482
12	50	0.05	5	0.01	//0_8_0_15_7_32		None	1000000	2312
13	50	0.05	5	0.01	//0_8_0_15_7_32		Replace Only	1000000	9704
14	50	0.05	5	0.01	//0_8_0_15_7_32		Reuse Only	1000000	4201

Figure 6.29 Typical Simulation Result

Figure 6.29 is the final result screen at the end of a simulation. This screen shows a summary of the GAL and SC's configuration, algorithm used failure rate, and fault limit.

7. Evaluation and Analysis of the Simulation Results for the Ultra Reliable Computing Systems

In this chapter, simulation results are analyzed to evaluate our self-repairing methods and to prove that our reliable system based on the self-repairable EPLDs, proposed in this dissertation, will last longer in the field. The simulation program is programmed using Microsoft Visual Studio 2005 (MFC) and a GAL16V8 [47, 48, 49]. Our computer-based simulator developed for test, diagnosis, and repair, including the function as a fault processor, is explained in more detail in Chapter 6.

7.1. Assumptions and Failure Rates

Assumptions for our evaluation methodology are the following. Each cross-point in the AND plane has the same probability of stuck-at faults (s-a-0 and s-a-1) occurring. Consequently, each column and OR gate has also the same probability of stuck-at faults to occur. Each interconnection line in a switching circuit has also the same probability of stuck-at faults (s-a-0 and s-a-1) occurring. The failure of one cross-point in an AND plane and one line in an SC does not affect the likelihood that another cross-point and line will fail, respectively. Thus, the failure of one cross-point

or one line is assumed to be independent of the failure of another cross-point or line, respectively. Likewise for columns, ORs, and switching circuit blocks. In GAL operation without self-repairing, if at least one cross-point stuck-at fault appears in a column, then that column will be faulty. This means that if at least one column fails in an OR gate, then that OR gate will be faulty, and if at least one OR gate is faulty in a GAL, then that section of the GAL is useless. In switching circuit operation without self-repairing, if at least one line stuck-at fault appears in an interconnection line, then that line will become faulty. This means that if at least one line fails in an SC, then that SC will become faulty, and if at least one SC is faulty in a system, then that whole system is useless even though a GAL module is not faulty. Due to the excessively large overhead, we do not consider the case when the number of extra columns and the number of extra ORs are greater than the number of original columns and the number of original ORs, respectively.

The PPM, 0.05% (489/1,000,000 ≈ 0.05%) and FIT, 5.00% (5.07% ≈ 5.00%) for EEPROMs obtained from National Semiconductor Corporation are adopted as the PPM and the FIT of a GAL module in our simulation. In our simulation, the MAP is

generated using 100% AND array utilization; thus all E^2CMOS cells are programmed using a random number generator. The connection of lines in an SC is generated with all GAL modules being used in a system; thus all original lines are programmed using a random number generator. After creating the description of the MAP and the NSC, the faults are simulated in an AND array and in an SC by using a random number generator, respectively; the maximum number of cross-points that have faults at a time are limited to be less or equal to 5, 10, 20, 50, 100, 1000, and 2048, and the maximum number of lines of the SC that have faults at a time are limited to be less or equal to 2 and 4. It is assumed that extra columns, extra ORs, and extra lines can also have faults. The extra columns will be increased in a multiple of 8 since there are 8 OLMCs in a GAL and each OLMC has the same number of extra columns as in extra ORs. The extra lines of each pin-to-pin connection will be increased in a multiple of 2 since each input pin of an SC has only one extra line, corresponding to an output pin of the SC. The results of the failure rate analysis are only valid under assumption that the assumed stuck-at fault model adequately represents all physical defects that can occur in a GAL device and an SC block.

7.2. Evaluation and Analysis of the Simulation Results

The same personality being originally programmed in the AND plane is applied into all self-repairing methodologies according to the number of extra columns. In other words, 0 number of extra columns' cases in each method (first rows of the following tables) are simulated with the same information of the AND plane. All 8 number of extra columns' cases in each method are simulated with same information of the AND plane which is the same as the programming of the 0 number of extra columns case and so on. In short, each case (extra columns 0, 8, 16, 24, 32, 40, 48, 56, and 64) has same programming information of the AND plane. It allows to evaluate the effect on the GAL's lifetime according to self-repairing methods under the same condition, the same programming data is used in the AND plane. It gives also how standardize the simulation results for each self-repairing method regardless of originally programmed data of the AND plane.

Table 7.1 displays average looping time using the column replacement method until a GAL is useless after running simulation in 1,000,000 times at each case. These results show that with more extra columns, longer survival times in the field can be

obtained (except two cases that the number of faults limit is less than 1000 and 2048 under 5% failure rate of a GAL (FIT)). However, these two cases will be very rare, even in less than 50 and less than 100 under either 0.05% or 5% failure rate of a GAL. The results on 0.05% failure rate of a GAL are better than on 5% failure rate of a GAL.

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	2709	27	2425	24	2300	23	2229	22	2206	22	2187	21	2183	21
8	7537	75	4791	47	3312	33	2572	25	2367	23	2203	22	2191	21
16	14909	149	8796	87	5296	52	3167	31	2624	26	2225	22	2201	22
24	23888	238	13703	137	7873	78	4079	40	2978	29	2254	22	2215	22
32	34163	341	19329	193	10829	108	5298	52	3443	34	2286	22	2231	22
40	45665	456	25563	255	14102	140	6669	66	4048	40	2323	23	2248	22
48	58136	580	32359	323	17629	176	8137	81	4792	47	2365	23	2267	22
56	71271	714	39650	396	21482	214	9704	97	5627	56	2410	24	2287	22
64	85843	855	47384	473	25536	255	11378	113	6479	64	2455	24	2310	23

Table 7.1 Average Looping Times of Simulating the Replacement Methodology

Table 7.2 displays average looping time using column-column re-use method only until a GAL is useless after running simulation in 1,000,000 times at each case. This case does not take into account of extra columns since the extra columns are useful only with the replacement method. However, failure rates applied into the AND plane

can be distributed to the extra columns equally as originally programmed columns. These results show also that with more extra columns, longer survival times in the field can be obtained in cases of the number of faults limit less than 5, 10, 20 and 50 under 0.05% failure rate (PPM), and less than 5 only under 5% of a GAL (FIT).

However, overall average looping time in this table is less than the average looping time of the replacement method except the cases of the number of faults limit less than 5, 10, and 20 under 0.05% failure rate in case of 0 number of extra columns. It means that in no extra columns, column-column re-use method is better than the replacement method for repairing faulty columns. On the other hand, the column-column re-use has difficulty to find an exact same personality that must be exchanged with a whole faulty column among columns being used. It shows that it is very dependable on the number of faults occurred at a time in an AND array rather than the number of extra columns added.

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	2708	27	2425	24	2300	22	2229	22	2207	22	2188	21	2183	21
8	2861	28	2496	24	2330	23	2240	22	2212	22	2188	21	2184	21
16	3018	30	2567	25	2362	23	2252	22	2218	22	2189	21	2185	21
24	3184	31	2642	26	2394	23	2265	22	2223	22	2190	21	2184	21
32	3347	33	2719	27	2428	24	2276	22	2229	22	2190	21	2185	21
40	3513	35	2798	27	2461	24	2287	22	2235	22	2191	21	2185	21
48	3683	36	2879	28	2496	24	2299	23	2241	22	2191	21	2185	21
56	3853	38	2964	29	2533	25	2312	23	2246	22	2192	21	2185	21
64	4026	40	3045	30	2569	25	2324	23	2252	22	2192	21	2186	21

Table 7.2 Average Looping Time of Simulating the Column-Column Re-Use Only

Table 7.3 displays average looping time using cell-column re-use method only until a GAL is useless after running simulation in 1,000,000 times at each case.

Extra columns are not used for repairing faulty columns as the column-column re-use case and have the same failure rates as the original columns. These results show also that with more extra columns, longer survival times in the field can be obtained (except three cases that the number of faults limit is less than 100, 1000, and 2048 under 5% failure rate of a GAL (FIT)).

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	12654	126	7708	77	4864	48	3076	30	2587	25	2222	22	2200	22
8	14074	140	8483	84	5276	52	3226	32	2646	26	2228	22	2202	22
16	15515	155	9272	92	5686	56	3379	33	2710	27	2233	22	2205	22
24	16961	169	10049	100	6101	60	3538	35	2773	27	2237	22	2207	22
32	18401	183	10825	108	6501	65	3702	36	2836	28	2243	22	2210	22
40	19816	198	11619	116	6913	69	3864	38	2903	29	2248	22	2212	22
48	21273	212	12392	124	7333	73	4032	40	2973	29	2252	22	2214	22
56	22654	226	13166	131	7726	77	4201	41	3043	30	2257	22	2217	22
64	24039	241	13934	139	8135	81	4363	43	3119	31	2263	22	2219	22

Table 7.3 Average Looping Time of Simulating the Cell-Column Re-Use Only

There are no linear increases of the lifetime under two cases that the number of faults limit is less than 1000, and 2048 under 0.05% failure rate of a GAL according to lots of faulty cells at a time and difficulty of finding a re-usable column. For instance, in worst case, if all faults occur in a certain column, then it is very hard to find a re-usable column in that AND array even though comparing only faulty cell locations with a column being used. In cases of the number of faults limit less than 5, 10, 20, 50, and 100 under either 0.05% or 5% failure rate of a GAL, the looping time of these

cases not using any extra columns (not using the replacement method) is longer than the looping time of the replacement method using extra columns up to 16 extra columns. In 0 number of extra columns case, using cell-column re-use method only (12654, 126; average looping time under the number of fault limits ≤ 5 in Table 7.3 makes a self-repairable GAL last over 4.5 times longer than using only replacement method (2709, 27; average looping time under the number of fault limits ≤ 5 in Table 7.1) under both 0.05% and 5% failure rate. It means that the cell-column re-use method itself makes a self-repairable GAL more reliable than the use of the column replacement method or the use of the column-column re-use method even without any spare columns. However, in cases adding greater or equal to 24 extra columns under either 0.05% or 5% failure rate of a GAL, overall average looping time in this table is less than the average looping time of the replacement method. It means that replacing faulty columns with extra columns is better than re-using columns being used for faulty cells in adding more extra columns. It shows the limitation of using only the cell-column re-use method to have longer lifetime (average looping time) in all the cases, comparing with the replacement method.

Table 7.4 shows the simulation results using a combination of two methods, column-column re-use method and replacement method. These results are very similar with the replacement case, which means that the column-column re-use is not much useful. However, the column-column re-use method makes a GAL more reliable in cases of the small number of faults and primary inputs (rows) and the large number of programmed columns being used. The result in this case is little better than on the Table 7.1, but both table have almost same average looping time.

Table 7.5 displays average looping time using cell-column re-use method and replacement method until a GAL is useless after running simulation in 1,000,000 times at each case. These results also show that with more extra columns, longer survival times in the field can be obtained. The results on 0.05% failure rate of a GAL are better than on 5% failure rate of a GAL as all other tables. The combination of cell-column re-use method and replacement method makes a self-repairable GAL most reliable.

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	2708	27	2427	24	2299	22	2228	22	2206	22	2188	21	2183	21
8	7537	75	4793	47	3312	33	2571	25	2367	23	2203	22	2190	21
16	14909	149	8795	87	5297	52	3165	31	2624	26	2226	22	2201	22
24	23890	239	13702	136	7873	78	4080	40	2979	29	2254	22	2215	22
32	34163	341	19331	193	10831	108	5298	52	3442	34	2286	22	2231	22
40	45666	456	25563	255	14100	140	6668	66	4048	40	2323	23	2248	22
48	58135	580	32361	323	17628	176	8136	81	4793	47	2365	23	2267	22
56	71270	714	39650	396	21482	214	9704	97	5627	56	2410	24	2287	22
64	85844	855	47383	473	25535	254	11377	113	6478	64	2456	24	2308	23

Table 7.4 Average Looping Time of Simulating the Column-Column Re-Use and Replacement

In 0 number of extra columns case, using cell-column re-use method (12653, 126) makes a self-repairable GAL last over 5 times longer than using only replacement method (2709, 27) in Table 7.1 under both 0.05% and 5% failure rate.

As a result, our new hardware self-repairing algorithm, the cell-column re-use with extra columns (with the replacement method), is the best solution to self-repair faulty columns including hidden faulty cells occurred in an AND plane (both originally programmed columns and extra columns) of a GAL. If the average looping

time can be converted to the lifetime of a GAL, it would provide an indication how many extra columns a GAL should have in order to guarantee certain reliability and lifetime.

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	12653	126	7710	77	4866	48	3079	30	2588	25	2222	22	2200	22
8	31147	311	17772	177	10129	101	5129	51	3400	33	2285	22	2230	22
16	50068	499	28078	280	15484	154	7346	73	4443	44	2348	23	2260	22
24	69274	692	38521	385	20983	209	9600	96	5640	56	2413	24	2289	22
32	88973	888	49286	492	26587	265	11897	119	6814	68	2481	24	2321	23
40	108933	1089	60208	602	32326	323	14282	142	8003	80	2555	25	2354	23
48	129496	1295	71466	714	38188	382	16711	167	9229	92	2632	26	2386	23
56	150846	1506	82853	829	44236	442	19167	191	10496	104	2711	27	2422	24
64	171992	1719	94653	945	50346	503	21709	217	11764	117	2795	27	2458	24

Table 7.5 Average Looping Time of Simulating the Cell-Column Re-Use and Replacement

Finally, as shown in Figure 7.1 through Figure 7.6, a self-repairable GAL using the cell-column re-use method with extra columns (simply called cell re-use) will last longer than one using the replacement method or no self-repair method in the field. Figure 7.1 through Figure 7.6 only shows three cases that the number of faults limit is

less or equal to 5, 10, and 20 under both 0.05% and 5% failure rate since for other cases that the number of faults limit is less or equal to 50, 100, 1000 and 2048, under both, failure of a GAL will be very rare. The simulation results for the cell-column re-use method with extra columns are better than the results for the replacement method in all cases.

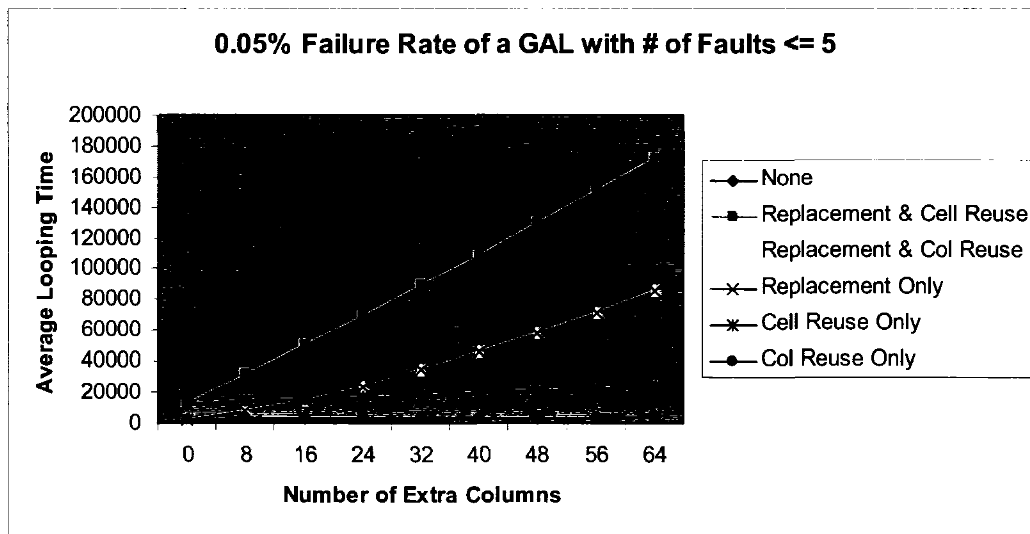


Figure 7.1 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 5 and 0.05% Failure Rate

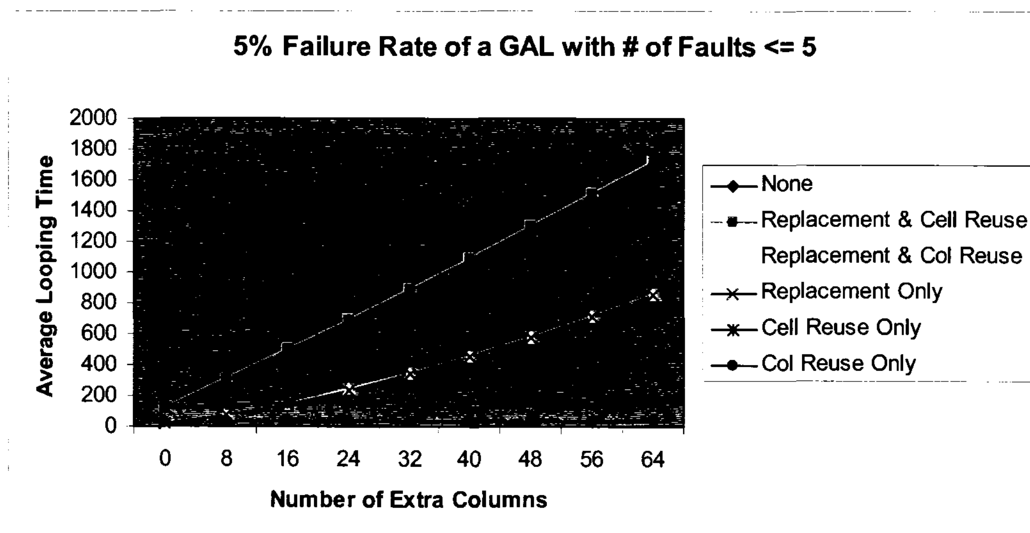


Figure 7.2 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 5 and 5.00% Failure Rate

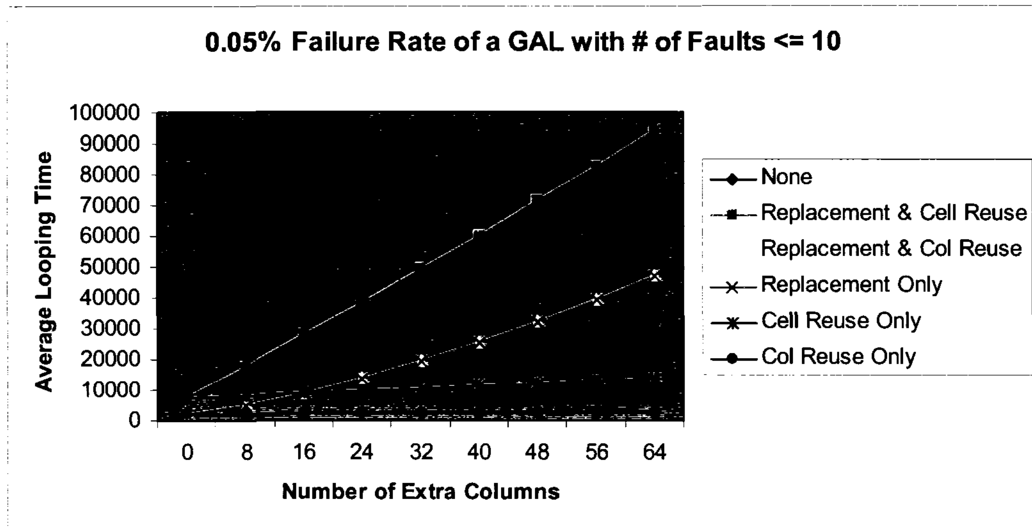


Figure 7.3 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 10 and 5.00% Failure Rate

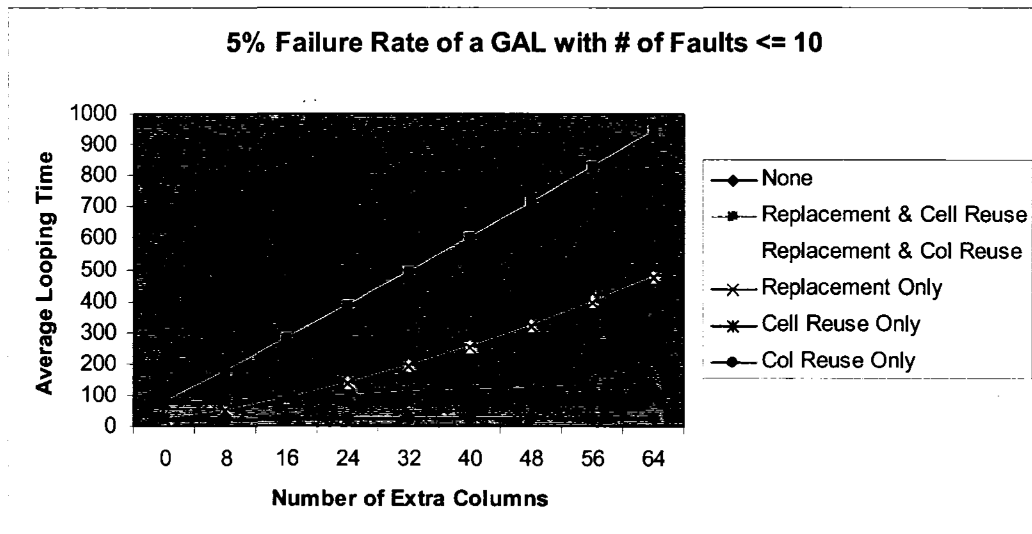


Figure 7.4 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 10 and 5.00% Failure Rate

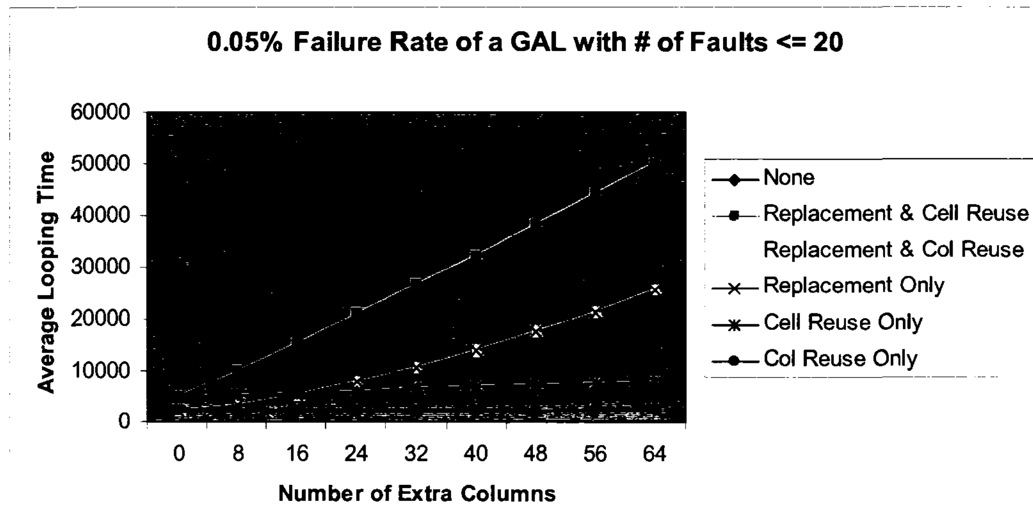


Figure 7.5 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 20 and 5.00% Failure Rate

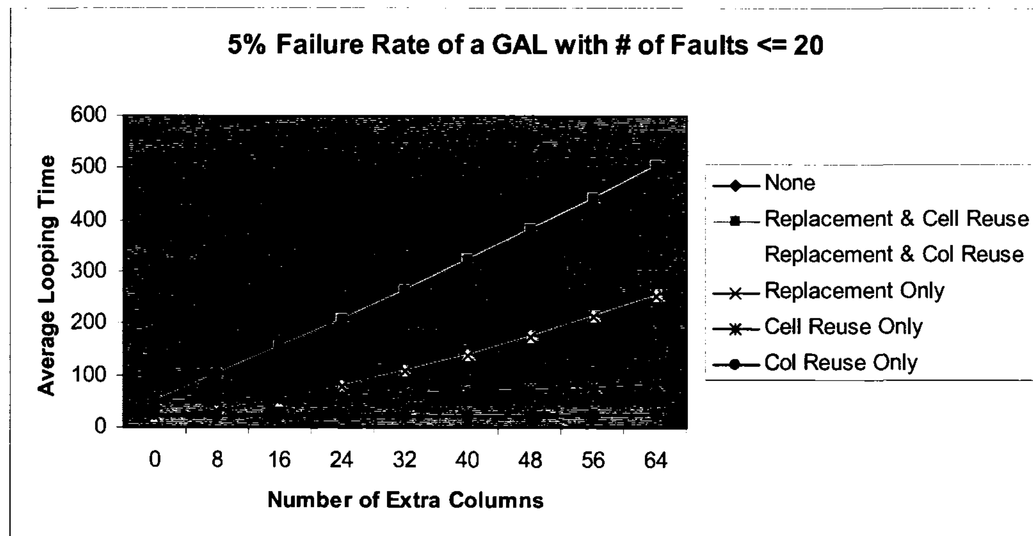


Figure 7.6 Comparison of Average Looping Time for All Methodologies with the Fault Limit less than or equal to 20 and 5.00% Failure Rate

Column re-use method with extra columns is the same as the replacement method as shown in Figure 7.1 through Figure 7.6 under both 0.05% and 5% failure rate of a GAL. Figure 7.1, Figure 7.3, and Figure 7.5 are on different vertical scales from Figure 7.2, Figure 7.4, and Figure 7.6 respectively. The column re-use algorithm is most effective when columns in each OR-gate have similar personality or the input number is small. However, these simulations did not obtain such results. That is the reason for the performance of the column re-use being so close to the performance of no algorithm on these graphs.

7.3. Hardware Overhead and Performance

Now, we consider the hardware overhead in a GAL. In the measurement of area overhead, the LSI Logic Corporation's 0.5-micron LCA/LEA500K array-based products are used for synthesis [50]. All area measurements are expressed in cell units, excluding the interconnection wires. This measurement is separated into two parts, an AND plane and others (OLMCs, Input/Output Buffers/Inverters, a SCR1 (Serial-In-Parallel-Out Shift Register), and a SCR2 (Parallel-In-Serial-Out Shift Register)). In

measurement of an AND plane size, each cross-point section consumes 1 cell, i.e. 2048 (32 inputs x 64 columns) cell units are measured for an AND plane in case of no extra columns. If there are 8 extra columns added in an AND plane, the AND plane has 2304 (32 inputs x 72 columns) cell units, and so on.

Basic Component Item	Cell count
AND Plane (Each cross-point section)	1
2 input EXOR gate	3
D flip-flop	6
1-of-2 Multiplexer	4
1-of-3 Multiplexer	5
1-of-4 Multiplexer	6
Tri-state buffer	3
Inverter	1
4 input OR gate	3
2 input OR gate	2
4 input AND gate	3
2 input AND gate	2
1-to-2 DEMUX	4
1-to-4 DEMUX	6

Table 7.6 Generic Components

Table 7.6 is the list of basic components and the cell counts that are used to represent the hardware overhead cost.

OLMC Component	Component count	Cell count
1-of-2 Multiplexer	2	8
1-of-3 Multiplexer	1	5
1-of-4 Multiplexer	1	6
D flip-flop	1	6
2 input EXOR gate	1	3
Tri-state buffer	1	3
Inverter	1	1
2 input OR gate	K(0[9~10, 12~13, 15~16 input], or 1[8, 11, 14 input])	2*K
4 input OR gate	K(2[8 input], 3[9~11 input], 4[12~14 input], or 5[15~16 input])	3*K
SCR1 Component		
D flip-flop	Number of primary Input(K)	6*K
SCR2 Component		
D flip-flop	Total columns of the GAL(K)	6*K
AND Plane Of a GAL		
Each cross-point section	Total columns of the GAL(K) * Primary input numbers of the GAL(L)	K*L
Other components		
Inverter/Input/Output	Number of output(K) * 2 + 2	2*K + 2
GAL components		
AND Plane	1	
OLMC	1	
SCR1	1	
SCR2	1	
Other GAL components	1	

Table 7.7 GAL Components

Table 7.7 is the list of GAL components. OLMC's OR-gates combination is determined by the number of OR-gate's input as shown in Table 7.8. This table was designed to minimize the number of OR-gates used in OLMC.

Number of columns in an OR-gate	2-input OR-gate	4-input OR-gate
8	1	2
9	0	3
10	0	3
11	1	3
12	0	4
13	0	4
14	1	4
15	0	5
16	0	5

Table 7.8 OLMC's OR-gate Combination Chart

For total area overhead of a GAL, the size of an AND plane is added to the sizes for OLMCs, Input/Output Buffers/Inverters, a SCR1 (Serial-In-Parallel-Out Shift Register), and a SCR2 (Parallel-In-Serial-Out Shift Register). For example, if there are 16 extra columns in an AND plane, the GAL has total number of 3602 cell units; 2560 cell units (32 x (64+16), AND plane) + 352 cell units (44 x 8, 8 OLMCs) + 18 cell

units (Input/Output Buffers/Inverters) + 192 cell units (6 x 32, 32-bit SCR1) + 480 cell units (6 x 80, 80-bit SCR2)).

Table 7.9 provides the area overheads and ratios. These calculations have an acceptable error rate because GAL has a regular structure so that assumption of no connection overhead is realistic.

The ratios from Table 7.9 will be also used to calculate the reliability/cost defined as the ratio of looping time divided by the ratio of area overhead at each case.

# of Extra Columns	# of Cells	Ratio
0	2970	1.00
8	3298	1.11
16	3602	1.21
24	3906	1.32
32	4210	1.42
40	4538	1.53
48	4842	1.63
56	5146	1.73
64	5450	1.84

Table 7.9 Area Overhead and Ratio

Table 7.10 and Table 7.11 show the performance of a self-repairable GAL using column replacement only and cell-column re-use & replacement respectively, as a function of the number of extra columns. These tables represent the ratio of efficiency versus cost factor, so a larger number represents better performance. Theses results show that all cases where the number of fault limits are larger than 50 (shown in table as ≤ 50, ≤ 100, ≤ 1000, and ≤ 2048) are not realistic.

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	2.52	2.51	1.79	1.77	1.30	1.30	1.04	1.03	0.97	0.95	0.91	0.95	0.91	0.90
16	4.56	4.57	3.00	3.00	1.91	1.87	1.18	1.17	0.98	0.98	0.84	0.87	0.83	0.87
24	6.70	6.70	4.29	4.34	2.60	2.58	1.39	1.38	1.03	1.00	0.78	0.80	0.77	0.80
32	8.87	8.89	5.61	5.66	3.31	3.30	1.67	1.66	1.10	1.09	0.74	0.74	0.72	0.74
40	11.06	11.08	6.92	6.97	4.02	3.99	1.96	1.97	1.20	1.19	0.70	0.72	0.68	0.69
48	13.15	13.16	8.18	8.25	4.70	4.69	2.24	2.26	1.33	1.31	0.66	0.67	0.64	0.64
56	15.14	15.22	9.41	9.50	5.38	5.36	2.51	2.54	1.47	1.47	0.63	0.66	0.60	0.60
64	17.22	17.21	10.62	10.71	6.03	6.03	2.77	2.79	1.60	1.58	0.61	0.62	0.58	0.60

Table 7.10 Performance of a Self-Repairable GAL using Column Replacement Method

# of Extra Col.	# of Fault Limits													
	≤ 5		≤ 10		≤ 20		≤ 50		≤ 100		≤ 1000		≤ 2048	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	2.23	2.23	2.09	2.08	1.88	1.90	1.51	1.54	1.19	1.19	0.93	0.90	0.92	0.90
16	3.28	3.28	3.01	3.01	2.63	2.66	1.98	2.01	1.42	1.46	0.87	0.87	0.85	0.83
24	4.16	4.17	3.80	3.80	3.28	3.31	2.37	2.43	1.66	1.70	0.83	0.83	0.79	0.76
32	4.95	4.96	4.50	4.50	3.84	3.88	2.72	2.79	1.85	1.91	0.79	0.77	0.74	0.74
40	5.65	5.67	5.12	5.13	4.36	4.42	3.04	3.11	2.03	2.10	0.75	0.75	0.70	0.69
48	6.27	6.30	5.68	5.68	4.81	4.88	3.33	3.41	2.19	2.25	0.73	0.72	0.66	0.64
56	6.86	6.88	6.19	6.20	5.23	5.30	3.58	3.66	2.33	2.39	0.70	0.71	0.63	0.63
64	7.39	7.41	6.67	6.67	5.62	5.70	3.83	3.93	2.47	2.54	0.68	0.67	0.61	0.59

Table 7.11 Performance of a Self-Repairable GAL using Cell-Column Re-Use Method & Replacement Method

The case of adding 64 extra columns for the assumption of the number of faults, less or equal 5 is the most efficient from Table 7.10 (replacement method) both under the 0.05% failure rate (17.22) and under the 5% failure rate of a GAL (17.21) and from Table 7.11 (cell-column re-use method with extra columns) both under the 0.05% failure rate (7.39) and under the 5% failure rate of a GAL (7.41). These performance values may mislead the readers to think that the column replacement method is better than the cell-column re-use & replacement method, but these values are measured

independent of each other and have no meaning in comparing the two methods' values. These values represent how much improvement is added with each additional column within the method. For example, replacement method's average looping time went from 2709 to 85843 (3170% increase) when 64 extra columns are added. Under the same condition, cell-column re-use with replacement methods values went from 12653 to 171992 (1360% increase). The values on Table 7.10 and Table 7.11 represent the percentage values, not the average looping time.

The performance of either column-column re-use only or cell-column re-use only is not taken into account in this sub section since both cases do not use extra columns, and the efficiency for the column-column re-use with extra columns is not shown since it is almost same as the result using the replacement method only.

7.3.1. Extra OR-Gate

Algorithm used in this simulation is the cell-column re-use and replacement with fault limit ≤ 5, and the failure rates set at 0.05% and 0.5%. Each OR-gate in the GAL is configured with 8 columns with no extra columns, and each column has 32 inputs.

The number of OR-gate is then incremented to see what effect this change had on the system. The result is then used in hardware overhead analysis to find out if the performance upgrade justifies the additional overhead.

# of Extra OR-Gates	# of Fault Limits		# of Extra Columns	# of Fault Limits	
	≤ 5			≤ 5	
	0.05%	5%		0.05%	5%
0	12653	126	0	12653	126
1	14088	141	8	31147	311
2	15523	155	16	50068	499
3	16975	169	24	69274	692
4	18405	184	32	88973	888
5	18471	195	40	108933	1089
6	21249	198	48	129496	1295
7	21434	226	56	150846	1506
8	24162	228	64	171992	1719

Table 7.12 Average Looping Time of Simulating the Cell-Column Re-Use and Replacement

Table 7.12 is the looping time of a GAL with incremented extra OR-gates (left side of the table). And, the looping time of a GAL with incremented extra columns is on the right side of the table. The increments on both sides are comparable hardware

overhead cost. The looping time was generated from running the simulation 1,000,000 times.

The correlation between the extra OR-gates in GALs found in the simulation result proved that the increased number of extra OR-gates had a very little effect on the performance regardless of the algorithm, failure rate, or the fault limit.

# of Extra OR-Gates	# of Fault Limits ≤ 5		# of Extra Columns	# of Fault Limits ≤ 5	
	0.05%	5%		0.05%	5%
0	1.000	1.000	0	1.000	1.000
1	0.952	0.957	8	2.227	2.233
2	0.909	0.911	16	3.276	3.278
3	0.871	0.870	24	4.160	4.173
4	0.835	0.838	32	4.947	4.958
5	0.747	0.791	40	5.649	5.671
6	0.771	0.721	48	6.271	6.298
7	0.702	0.743	56	6.862	6.880
8	0.718	0.681	64	7.388	7.415

Table 7.13 Performance Comparison of GALs with Extra OR-Gates and with Extra Columns

Table 7.13 shows that the average looping time is increased as the number of OR-gate is increased, but when compared to the performance of the test set with increased

number of extra columns with the equivalent overhead increase, the efficiency value significantly falls behind increasing only the columns within the OR-gates. The performance ratio was actually decreased. The Performance ratio/overhead ratio graph, Figure 7.7, clearly shows that adding extra OR-gate on GALs does not translate to increase in performance. This result suggests that adding extra columns on OR-gates is a more effective way of increasing GAL's performance rather than adding extra OR-gates on a GAL.

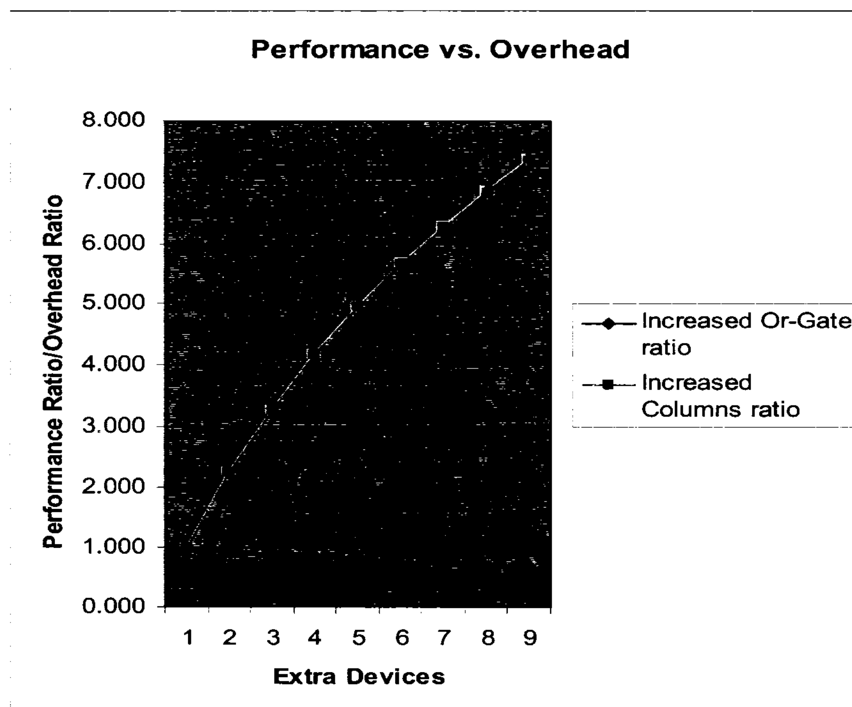


Figure 7.7 Performance Ratio/Overhead Ratio of Increased OR-Gate vs. Columns

7.3.2. Extra Line on Switching Circuits

In order to find out the isolated looping time (simulated lifetime) of SCs (switching circuit), 8-input and 8-output SC is put in the simulation. An SC commonly consists of lines, and lines have a very low failure rate, compared to EEPROM based devices. Since having more than two lines is meaningless in terms of performance enhancement due to the low failure rate on lines, only one extra line is added to obtain more useful data. The fault limit is set at ≤ 2 and ≤ 4, and the failure rate at 0.01% and 0.10%. Also, two 8 (number of an OR gate per a GAL) x 8 (number of columns per an OR) x 32 (number of inputs) GALs were put in place since the simulator does not allow having only SCs. The GALs' failure rate and fault limit are set to 0, in order to isolate and extract SC related data. The looping time was generated from running the simulation 1,000,000 times.

The looping time was increased by more than 5 times when an extra line was added at ≤ 2 fault limits, and more than 3 times at ≤ 4 fault limits.

# of Extra Lines	# of Fault Limits		# of Fault Limits	
	≤ 2		≤ 4	
	0.01%	0.10%	0.01%	0.10%
0	15891	1592	13426	1347
1	80636	8110	51609	5153

Table 7.14 Average Looping Time of SC with 1 and 2 Extra Lines

An SC consists of SCR6, SCR7, and other logic gates. Calculation for each component cost is listed in Table 7.15. An example calculation is, if a 10-input SC is used, the overhead calculation would be:

SCR6 (60) + SCR7 (60) + Other Logic Gates for SC (300) = 420

If an extra line is added with above example, the total hardware overhead is 840.

SCR6		
D flip-flop	Number of input(K)	6*K
SCR7		
D flip-flop	Number of output(K)	6*K
Other logic gates for SC	**Component count**	**Cell count**
1-to-2 DEMUX	Number of input(K) * L(0[9~10, 12~13, 15~16 input], or 1[8, 11, 14 input])	4*K*L
1-to-4 DEMUX	Number of input(K) * M(2[8 input], 3[9~11 input], 4[12~14 input], or 5[15~16 input])	6*K*M
2 input AND gate	Number of input(K) * J(0[9~10, 12~13, 15~16 input], or 1[8, 11, 14 input])	2*K*J
4 input AND gate	Number of input(K) * N(2[8 input], 3[9~11 input], 4[12~14 input], or 5[15~16 input])	3*K*N
Tri-state buffer	Number of outputput(K)	3*K
Switching Circuit		
SCR6	Number of lines (n = 1 or n = 2)	n*SCR6
SCR7	Number of lines (n = 1 or n = 2)	n*SCR7
Other logic gates for SC	Number of lines (n = 1 or n = 2)	n*total

Table 7.15 SC Components

	Input of SC	8	9	10	11	12	13	14	15	16
Total Number of Lines										
1		312	378	420	528	612	663	756	900	960
2		624	756	840	1056	1224	1326	1512	1800	1920

Table 7.16 SC Overhead Cell Count

Table 7.16 is SC hardware overhead table sorted by the number of inputs and lines.

# of Extra Lines	# of Fault Limits ≤ 2		# of Fault Limits ≤ 4	
	0.05%	5%	0.05%	5%
0	1	1	1	1
1	2.537	2.547	1.922	1.913

Table 7.17 Performance of SC using Extra Line

Table 7.17 shows the increase in performance by adding an extra line, and the increase was bigger in the lower fault limit setting. This simulation result can assess the effective value of the extra line.

The result in Table 7.14 shows that the extra line improved the looping time by 5 times (Fault Limit = 2) and 4 times (Fault Limit = 4) the result of the set with no extra line. It also proved that the performance increase ratio is twice the value of the overhead increase ratio.

7.3.3. Performance by Available OR-Gate

This simulation is to find out the correlation of SCs and extra OR-gates since SCs' fault occurrence is affected by the OR-gates usage on a GAL. In reality, there exists unused OR-gates in GALs, and these unused OR-gates can be used as extra OR-gates in the prototype system of this project. Therefore, it is possible to generate faults on SC and see how much extra the OR-gate usage has effect on the increase of SC's lifetime.

An SC uses OR-gates to repair faults, thus the simulation can also verify if this increase in SC lifetime influences the lifetime of the whole system. In this simulation, two 8 x 8 x 32 GALs and an SC connecting these two GALs are created. The GAL is given fault limit = 0, failure rate = 0%, and the SC is given fault limit = 2

and fault rate = 5%. The looping time was generated from running the simulation 1,000,000 times.

	Number of **OR-Gates**	8	9	10	11	12	13	14	15	16
Number of **Columns** per an OR-Gate	2386									
8		2962	3464	3998	4564	5162	5792	6454	7148	7874
9		3274	3833	4428	5059	5726	6429	7168	7943	8754
10		3578	4193	4848	5543	6278	7053	7868	8723	9618
11		3898	4571	5288	6049	6854	7703	8596	9533	10514
12		4210	4940	5718	6544	7418	8340	9310	10328	11394
13		4514	5300	6138	7028	7970	8964	10010	11108	12258
14		4834	5678	6578	7534	8546	9614	10738	11918	13154
15		5146	6047	7008	8029	9110	10251	11452	12713	14034
16		5450	6407	7428	8513	9662	10875	12152	13493	14898

Table 7.18 GAL Overhead –Total Cell Count

Table 7.18 shows the hardware overhead of a generic GAL (2386 cell count: 8 OR-gates x 8 columns x 32 inputs) and self-repairable GALs used in this project.

	Number of **OR-Gates**	8	9	10	11	12	13	14	15	16
Number of **Columns** per an OR-Gate										
8		1.00	1.17	1.35	1.54	1.74	1.96	2.18	2.41	2.66
9		1.11	1.29	1.49	1.71	1.93	2.17	2.42	2.68	2.96
10		1.21	1.42	1.64	1.87	2.12	2.38	2.66	2.94	3.25
11		1.32	1.54	1.79	2.04	2.31	2.60	2.90	3.22	3.55
12		1.42	1.67	1.93	2.21	2.50	2.82	3.14	3.49	3.85
13		1.52	1.79	2.07	2.37	2.69	3.03	3.38	3.75	4.14
14		1.63	1.92	2.22	2.54	2.89	3.25	3.63	4.02	4.44
15		1.74	2.04	2.37	2.71	3.08	3.46	3.87	4.29	4.74
16		1.84	2.16	2.51	2.87	3.26	3.67	4.10	4.56	5.03

Table 7.19 GAL Overhead Ratio (Based on the Basic Prototype Proposed)

Table 7.19 shows the hardware overhead ratio of GALs based on suggested prototype: 8 OR-gates x 8 Columns x 32 inputs

The data on Table 7.20 proved that the increase in the number of extra OR-gates allows the SCs to have more chances to repair its faults, and significantly improves the SCs lifetime and the system's performance as shown on Figure 7.8.

Number of Unused OR-Gates on x8 OR-Gate in a GAL	Number of Lines	
	1	2
0	31	162
1	35	175
2	40	192
3	46	213
4	57	246
5	74	294
6	108	383
7	212	624

Table 7.20 SC Looping Time Variance on Number of Unused OR-Gates on GAL

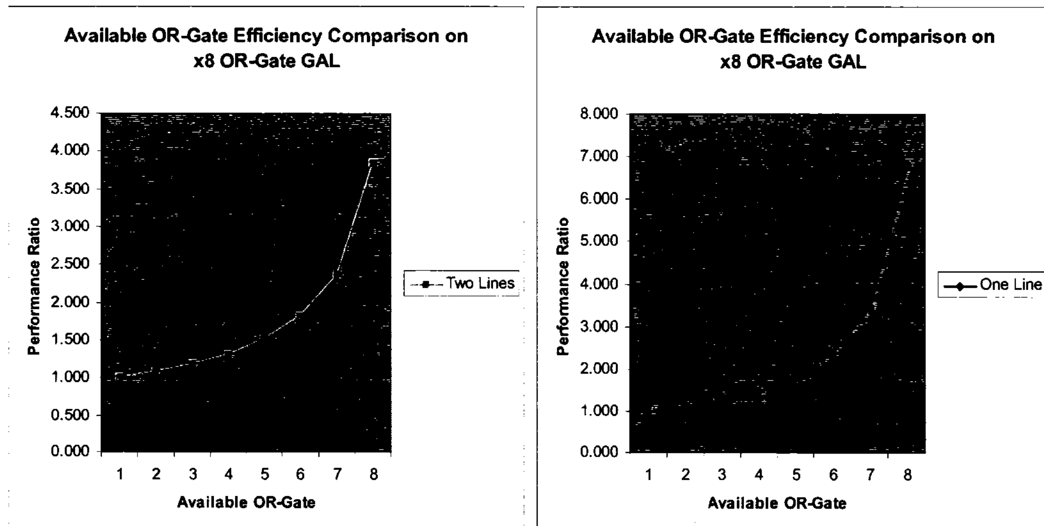


Figure 7.8 Available OR-Gate and Performance Increase (One Line: No Extra Line & Two Lines: 1 Extra Line in each Pin-to-Pin Connection)

7.3.4. FPGA and ASIC Design

One chipset is created on the simulator to compare the two most commonly used chip structures, FPGA and ASIC. This simulation will also gauge the effect of the number of extra OR-gates in GALs of the whole system. The looping time will be generated from running the simulation 1,000,000 times.

- FPGA = GAL (8 x 16 x 32) x 4 + SC x 3
- ASIC = GAL (8 x 16 x 32) x 4 + SC x 2

	Number of **OR-Gates**	8	9	10	11	12	13	14	15	16
Number of **Columns** per an OR-Gate	1.00									
8		1.24	1.45	1.68	1.91	2.16	2.43	2.70	3.00	3.30
9		1.37	1.61	1.86	2.12	2.40	2.69	3.00	3.33	3.67
10		1.50	1.76	2.03	2.32	2.63	2.96	3.30	3.66	4.03
11		1.63	1.92	2.22	2.54	2.87	3.23	3.60	4.00	4.41
12		1.76	2.07	2.40	2.74	3.11	3.50	3.90	4.33	4.78
13		1.89	2.22	2.57	2.95	3.34	3.76	4.20	4.66	5.14
14		2.03	2.38	2.76	3.16	3.58	4.03	4.50	4.99	5.51
15		2.16	2.53	2.94	3.37	3.82	4.30	4.80	5.33	5.88
16		2.28	2.69	3.11	3.57	4.05	4.56	5.09	5.66	6.24

Table 7.21 GAL Overhead Ratio (against a Generic Chipset, GAL)

Table 7.21 is the hardware overhead ratio of GALs based on a generic GAL.

	Average Looping Time (Lifetime)		Hardware Area		Lifetime Ratio		Hardware Overhead Ratio		Performance	
Number Extra OR-Gates Per a GAL	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA
0	905.00	770.00	23048.00	23672.00	1.00	1.00	1.00	1.00	1.00	1.00
1	1028.00	877.00	27140.00	27896.00	1.14	1.14	1.18	1.18	0.96	0.97
2	1154.00	984.00	31392.00	32232.00	1.28	1.28	1.36	1.36	0.94	0.94
3	1275.00	1092.00	36164.00	37220.00	1.41	1.42	1.57	1.57	0.90	0.90
4	1404.00	1200.00	41096.00	42320.00	1.55	1.56	1.78	1.79	0.87	0.87
5	1523.00	1309.00	46152.00	47478.00	1.68	1.70	2.00	2.01	0.84	0.85
6	1643.00	1421.00	51632.00	53144.00	1.82	1.85	2.24	2.25	0.81	0.82
7	1774.00	1526.00	57572.00	59372.00	1.96	1.98	2.50	2.51	0.78	0.79
8	1895.00	1633.00	63432.00	65352.00	2.09	2.12	2.75	2.76	0.76	0.77

Table 7.22 Comparing Data of FPGA and ASIC Simulation with Extra ORs

Each system consists of 4 GALs and 3 Switching Circuits connecting the GALs. SCs are located in between GAL0 and GAL1, GAL0 and GAL2, and GAL1 and

GAL3. In order to follow the FPGA design specifications which has a fixed layout, 3 SC must be used. However, ASIC's more flexible requires only two SCs.

The FPGA and ASIC simulation is not an attempt to simply find out if the self-repair methods used here can be applied, nor simply achieve comparative analysis on the performance of the two different types of chipsets.

First, a generic system is built and tested for looping time performance and the result is 6. This represents the current lifetime on a system that consists of 4 GALs and an SC connecting the GALs.

The prototype is simulated with increasing OR-gates to attain the value that is more intuitive and be compared to a generic system. The simulation results with no extra OR-gates was 905 and 770 for the prototype ASIC and FPGA, respectively. That is 150 times and 128 times of the looping time of the generic chipset. When the overhead ratio is added to the comparison, the performance rating is improved by 66 times and 56 times the generic system's as shown on Table 7.23.

	Average Looping Time (Lifetime)		Hardware Area		Lifetime Ratio		Hardware Overhead Ratio		Performance	
Number Extra OR-Gates Per a GAL	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA
Generic Systems	6	6	10096	10408	1	1	1	1	1	1
0	905.00	770.00	23048.00	23672.00	150.83	128.33	2.28	2.27	66.07	56.43
1	1028.00	877.00	27140.00	27896.00	171.33	146.17	2.69	2.68	63.74	54.53
2	1154.00	984.00	31392.00	32232.00	192.33	164.00	3.11	3.10	61.86	52.96
3	1275.00	1092.00	36164.00	37220.00	212.50	182.00	3.58	3.58	59.32	50.89
4	1404.00	1200.00	41096.00	42320.00	234.00	200.00	4.07	4.07	57.49	49.19
5	1523.00	1309.00	46152.00	47478.00	253.83	218.17	4.57	4.56	55.53	47.83
6	1643.00	1421.00	51632.00	53144.00	273.83	236.83	5.11	5.11	53.54	46.38
7	1774.00	1526.00	57572.00	59372.00	295.67	254.33	5.70	5.70	51.85	44.59
8	1895.00	1633.00	63432.00	65352.00	315.83	272.17	6.28	6.28	50.27	43.35

Table 7.23 Prototype of FPGA and ASIC's Comparison to a Generic System

It is easily noticed that the looping time is increased when more extra OR-gates are added, but the performance rating does not increase as much due to the increase in overhead cost. This means that if looping time of two identical system (one of the

two as a backup) is compared a single system with the number of extra OR-gates that match the overhead cost of the two systems, the single system outperforms the two systems method. This brought focus to the importance of the extra OR-gates in the system. In a complicated system with many connections, the possibility of having faults to occur in SCs increases, and having the extra OR-gates becomes more imperative in a system as the extra OR-gates can repair faults on GALs as well as SCs. Additionally, the necessity of extra GALs becomes less important on small systems or systems with low failure rate SCs.

	Average Looping Time (Lifetime)		Hardware Area		Lifetime Ratio		Hardware Overhead Ratio		Performance	
Number of Unused OR-Gates on x8 OR-Gate in a GAL	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA	ASIC	FPGA
Generic Systems	6	6	10096	10408	1	1	1	1	1	1
0	905	770	23048	23672	150.83	128.33	2.28	2.27	66.07	56.43
1	953	820	23048	23672	158.83	136.67	2.28	2.27	69.58	60.09
2	1005	874	23048	23672	167.50	145.67	2.28	2.27	73.37	64.05
3	1071	942	23048	23672	178.50	157.00	2.28	2.27	78.19	69.03
4	1149	1027	23048	23672	191.50	171.17	2.28	2.27	83.89	75.26
5	1259	1139	23048	23672	209.83	189.83	2.28	2.27	91.92	83.47
6	1411	1308	23048	23672	235.17	218.00	2.28	2.27	103.01	95.85
7	1676	1598	23048	23672	279.33	266.33	2.28	2.27	122.36	117.10

Table 7.24 Unused Extra OR-Gate Efficiency

Table 7.24 shows the performance of each type based on proportion of programmed OR-gates. As the proportion of unused extra OR-gate increases, the prototype's performance increases. In the best scenario, the prototypes performance was 122 (ASIC) and 117 (FPGA) times that of a generic system.

The final analysis indicates that when the prototype developed in this research was tested with the best algorithm and was compared to a current generic system, the performance was improved by 66 (ASIC) and 56 (FPGA) times the generic system's.

8. Conclusions and Future Works

The design and implementation of the self-testable and self-repairable digital devices, especially, EPLDs for high security and safety applications as well as general purpose applications was completed. New hardware prototypes, both FPGA type and ASIC type, based on EEPROM technology for the ultra reliable computing systems, which will become a necessity for next scientific revolution such as nanotechnology, were proposed in this dissertation. This self-healing and re-configurable system design with added repair capability can provide higher yields, lower testing costs, and faster time-to-market to the semiconductor industry.

Our first step was gathering information on previous work on this topic and analyzes the existing technique to extract and filter out what eventually became the foundation of our research.

One of the most focused technologies in the field of self-repairable studies, GAL devices, which also is our initial target device, had to be researched in depth. We developed a design methodology for self-repairing of a GAL which is a type of EPLD. A fault-locating and fault-repairing architecture with electrically re-configurable

GALs was presented. It uses universal test sets, fault-detecting logic, and self-repairing circuits with spare devices. Our design method allows detecting, diagnosing, and repairing of all multiple stuck-at faults which may occur on E^2CMOS cells in the programmable AND plane. Three self-repairing methodologies for product terms/columns in the AND plane of a GAL were developed based on our design architecture; column replacement with extra columns and two variations of column re-use with extra columns. The "column replacement" method with extra columns discards each faulty column entirely and replaced it with an extra column. In contrast, the "column re-use" method, using the whole column (the column-column re-use), exchanges a faulty GAL column with a column that needs the same programming as the faulty column, and then reprograms the freed column to replace the faulty one. In the "column re-use" method, using the only cells (E^2CMOS cells) (the cell-column re-use), the respective faulty elements (E^2CMOS cells/cross-points) were re-used for whole columns or only cells which had been already programmed if terms' programming would fit the nature of the existing faults.

In addition, a self-repairing methodology for an OR group was also developed to create more robust and reliable GALs. A faulty OR group in a GAL was discarded and replaced with an extra OR group in the GAL if the columns in an OR group of a GAL were entirely faulty, and no more column repair methods were applicable in that OR group.

The proof of concept and application of the self-testing and the self-repairing switching circuit based on Demultiplexer structure was also developed in this dissertation. It was used to connect and reconfigure GALs in our system. It was also self-testable and self-repairable for faulty lines with extra lines in a switching circuit between GALs of a system.

Our design methodology (the fault-locating and fault-repairing architecture with electrically re-configurable GAL modules and self-testing and self-repairing switching circuits), developed in this dissertation, allows us to detect, diagnose, and repair of all multiple stuck-at faults that might occur on E^2CMOS cells in programmable AND plane of a GAL, faulty ORs in a GAL, and faulty lines of a switching circuit in proposed system. We developed a self-repairing methodology for switching circuits

based on our design architecture; line Replacement with extra lines; the respective faulty interconnection lines were replaced with the new ones (extra lines) by automatic reprogramming of the chip.

According to the extra devices added in our system, even though the power consumption is considered as one of the limitations, the power will be down when transistors (extra devices) are not needed or a little added power might be consumed for the whole system. Another limitation in our design is the reliability of memories/registers of our system such as MAP/SAP. They are different architecture from the GAL modules and the switching circuit blocks in the system. The current memory technology has fault detection and correction techniques such as hamming code and ECC (Error Correction Code or Error Checking and Correcting) which allows data that is being read or transmitted to be checked for errors. ECC is increasingly being designed into data storage. It covers this consideration.

To test our architecture, we designed and developed a flexible simulator. Our simulator used Microsoft Visual Studio 2005 as the tool and MFC (Microsoft Foundation Classes) is used for the programming language.

In order to evaluate the self-test and self-repair algorithms which are intended for hardware programming, a software program was created to simulate the hardware. The objectives of the simulator are to verify how each algorithm improves the performance of a system in realistic environment. The simulator program allows us to apply different failure rate and fault limit to support GALs and SCs of different sizes and structures. The simulator can also be configured with different size GALs and SCs for different test cases which may have different sized GALs, and the SCs' size that varies with the design of the system.

We developed the evaluation methodology in this dissertation. It was based on simulating our self-repair algorithms. We proved that the lifetime for a GAL-based EPLD that used our self-repairing methods was longer than the lifetime of a GAL-based EPLD that used a single self-repair method or no self-repair method. It demonstrated that the lifetime of a GAL was increased by adding extra columns in an AND array of a GAL and extra ORs in a GAL, and also gave information on how many extra columns and extra ORs a GAL needed and which self-repairing method a GAL used to guarantee a given lifetime. Our computer based simulation program

demonstrated the basic concepts of modeling repairable systems as introductory-level knowledge for repairable systems. Our simulator developed in this dissertation can be used to design more effective and less costly fault-testing and fault-repairing hardware architecture. It proved that our most advanced self-repair algorithm, the cell-column re-use with extra column and column replacement method, gave the best results in all the comparisons with our other self-repairing algorithms. Our computer based simulator implemented our self-testing and self-repairing hardwired algorithm; Self-repair with redundancy and Self-repair with no redundancy. Thus, we proposed guide lines to estimate an ideal point, where the maximum reliability can be reached with the minimum cost.

Future Works

- Extend our self-repairing methodologies into a Fault-Location/Fault-Repair Processor and memories in our system proposed in this dissertation.
- Develop more advanced hardwired self-testing and self-repairing algorithm
- Enhance our self-healing and re-configurable system design to fit in the Embedded System applications and Nanotechnology.
- Analyzing variability of lifetimes from our simulation using a math tool such as MATLAB and SPSS in Statistics.
- Finding an adopted method and making a general formula; induce a Mathematical Model/Stochastic Model in order to compare with the results from our simulator.

BIBLIOGRAPHY

[1] A. Avizienis, "Hundred Year Spacecraft," *The First NASA/DoD Workshop on Evolvable Hardware,* pp.233-239, July 1999.

[2] E. Drexler, "Engines of Creation," *Anchor Books*, 1986.

[3] G. Tempesti, D. Mange, and A. Stauffer, "A Self-Repairing FPGA Inspired By Biology," *The Third IEEE International On-Line Testing Workshop*, pp.191-195, 1997.

[4] D. Mange, M. Goeke, D. Madon, A Stauffer, G. Tempesti, and S. Durand, "Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties," *In Towards Evolvable Hardware*, Springer-Verlag, pp.197-220, 1996.

[5] D. Mange and A. Stauffer, "Introduction to Embryonics: Towards New Self-Repairing and Self-Reproducing Hardware Based on Biological-like Properties," *Artificial Life and Virtual Reality*, John Wiley, pp.61-72, 1994.

[6] D. L. Ostapko and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Array (PLA)," *IEEE Trans. on Computers, Vol. C-28, No. 9,* pp.617-627, September 1979.

[7] K. S. Ramanatha and N. N. Biswas, "An On-Line Algorithm for the Location of Cross Point Faults in Programmable Logic Arrays, " *IEEE Trans. on Computers, Vol. C-32, No. 5*, pp.438-444, May 1983.

[8] J. E. Smith, "Detection of Faults in Programmable Logic Arrays, " *IEEE Trans. on Computers, Vol. C-28, No. 11*, pp.845-853, November 1979.

[9] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Array with Universal Tests," *IEEE Trans. on Computers, Vol. C-30, No. 11*, pp.823-829, November 1981.

[10] W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," *IEEE Trans. on Computers, Vol. C-30, No. 11*, pp.829-833, November 1981.

[11] R. Treuer, H. Fujiwara, and V. K. Agarwal, " Implementing a Built-In Self-Test PLA Design," *IEEE Design and Test*, pp.37-48, April 1985.

[12] M. Abramovici, M. A. Breuer, and A. D. Friedman, "Digital Systems Testing and Testable Design," *IEEE Press*, pp.593-626, 1990.

[13]C. L. Wey and F. Lombardi, "On the Repair of Programmable Logic Arrays," *IEEE Trans. on Computers, Vol. 9*, pp.649-652, 1986.

[14] B. Avi, "A Tour of PLDs," *http://www.ee.cooper.edu/course_pages/past_courses/EE151/PLD1*, 1997.

[15] K. S. Son and D. K. Pradhan, "Design of Programmable Arrays for Testability," *1980 IEEE Test Conference*, pp.163-166, 1980.

[16] "PLA and FPGA Devices," http://www.elec.uq.oz.au/~e3390/lectures/lect14/sld002.htm, 1998.

[17] "Sequential Logic Design with PLDs," *http://www.elec.uq.oz.au/~e3390/lectures/lect14/sld014.htm*, 1998.

[18] "Lattice Semiconductor Corporation, "Lattice Semiconductor Data Book 1996," Lattice Semiconductor Corporation, pp.365-392, 1996.

[19] J. P. Hayes, "Fault Modeling," *IEEE Design & Test of Computers*, pp. 88-95, April 1985.

[20] W. Maly, "Realistic Fault Modeling for VLSI Testing," *Proc. of the 24th ACM/IEEE Design Automation Conference*, pp. 173-180, 1987.

[21] M. Marinescu, "Simple and Efficient Algorithms for Functional RAM Testing," *1982 International Test Conference*, pp. 236-239, November 1982.

[22] S. C. Ma, "Testing BiCMOS and Dynamic CMOS Logic," *Center for Reliable Computing Technical Report*, No. 95-1, June 1995.

[23] National Semiconductor Corporation, "Quality Network – Failure Rates of Major Processes at National Semiconductor, National Semiconductor Failure Rate Trends, and National Semiconductor Reliability," *http://207.82.57.10/quality/pages*, May 1999.

[24] D. Sellers, "Quality and Reliability," Quality and Reliability Hand Book – Space Electronics Inc., *http://www.spaceelectronics.com/Spaceprod/reliability/qr.html*, June 1999.

[25] J. Worchel, "Market for Programmable Logic Devices Heats," Semiconductor Business News, 1997.

[26] U. Kalay, D. V. Hall, and M. A. Perkowski, ``Easily Testable Multiple-Valued Galois Field Sum-of-Products Circuits", *Multiple Valued Logic Journal,* pp.507-528, Vol. 5, 2000.

[27] U. Kalay, M. A. Perkowski and D. V. Hall, ``A Minimal Universal Test Set for Self-Test of EXOR-Sum-Of-Products Circuits," *IEEE Tr. on Computers,* Vol. 49, pp.267–276, March 2000.

[28] U. Kalay, D. V. Hall, and M. A. Perkowski, "Highly Testable Boolean Rings," *Proc. ISMVL'99*, pp.17-21 May 1999.

[29] U. Kalay, N. Venkataramaiah, A. Mishchenko, D. V. Hall, and M. A. Perkowski, ``Highly Testable Finite State Machines Based on EXOR Logic", *PACRIM'99 7th* IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.23-25, August 1999.

[30] A. Sarabi, N. Song, M. Chrzanowska-Jeske, M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Proc. DAC'94,* pp.321-326 June 1994.

[31] N. Song, and M. A Perkowski, ``A New Approach to AND/OR/EXOR Factorization for Regular Arrays," *Proc. 1998 Euromicro,* pp.269-276, August 1998.

[32] N. Song, M. A. Perkowski, "A New Design Methodology for Two-Dimensional Logic Arrays," *Proc. of IEEE International Workshop on Logic Synthesis, IWLS '93,* pp.1-17, May 1993.

[33] U. Kalay, M. A. Perkowski, D. V. Hall, B. Steinbach, and S. A. Shahjahan, ``Rectangle Covering Factorization of ESOPs," *Proceedings of RM'99.*

[34] A. Kablanian, "Memory is dominating SoC Design," *Electronic News*, February 2001.

[35] V. Ratford, "Moving the Market to Embedded Memory," *Embedded Technology News*, March 2001.

[36] P. Buitenkant, "Overcoming Erase/Write-Endurance Limitations in EEPROMs," *EDN*, pp.95-98, September 2000.

[37] P. Bichebois, "Impact of Physical Defects on the Electrical Working of Embedded DRAM with 0.35mm Design Rules," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 124-130, November 1996.

[38] D. D. Gaitonde, W. Maly, and D. M. H. Walker, "Fault Probability Prediction for Array Based Designs," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 30-47, November 1996.

[39] H. Walker, "VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits," *IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 4*, pp. 541-556, October 1986.

[40] A. Gutierrez, "Batch-Mode Accelerated Reliability Tester for Electronic Memories (BART)," *http://www.intersci.com/eeprom.htm*, June 1999.

[41] E. R. Hnatek and B. R. Wilson, "An Evaluation of the 2816 EEPROM, " *1982 International Test Conference*, PP. 225-235, November 1982.

[42] K. Seshan, T. J. Maloney, and K. J. Wu, "The Quality and Reliability of Intel's Quarter Micron Process," *Intel Technology 3rd quarter Journal, http://www.intel.co.uk/technology/itj/q31998/articles/*, July 1998.

[43] F.G. Cockerill, "Quality Control for Production Testing," *1982 International Test Conference*, pp. 308-314, November 1982.

[44] R. H. Katz, "Contemporary Logic Design." Benjamin/Cummings Publishing Company, Inc., pp.160-172, 1992.

[45] T. L. Floyd, "Digital Fundamentals 6th Edition," Prentice Hall, pp.344-771, 1997.

[46] R. Nair, S. M. Thatte, and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories, " *IEEE Trans. on Computers, Vol. C-27, No. 6*, pp. 572-576, June 1978.

[47] C. H. Lee, M. A. Perkowski, D. V. Hall, and D. S. Jun, "Self-Repairable EPLDs: Design, Self-Repair, and Evaluation Methodology," *The Second NASA/DoD Workshop on Evolvable Hardware,* pp.183-193, July 2000.

[48] C. H. Lee, D. V. Hall, M. A. Perkowski, and D. S. Jun, "Self-Repairable GALs," Journal of System Architecture; The EUROMICRO Journal, Vol. 47, Issue 2, pp.119-135, February 2000.

[49] C. H. Lee, M. A. Perkowski, D. V. Hall, and D. S. Jun, "Self-Repairable EPLDs II: Advanced Self-Repairing Methodology," The Proceedings of Congress on Evolutionary Computation 2001, Vol. 1, pp.616-623, May 2001.

[50] LSI Logic Corporation, "LCA/LEA500K Array-Based Products Databook," *Document DB04-000002-03, Fourth Edition*, May 1997.

[51] L. Guerra, M. M. Potkonjak, and J. Rabey, "High Level Synthesis for Efficient Built-In Self Repair," *International Workshop on Defect and Fault Tolerance in VLSI Systems*, pp.41-48, October 1993.

[52] I. Hong, M. M. Potkonjak, and R. Karri, "Heterogeneous BISR-approach using System Level Synthesis Flexibility," *Proceedings of the ASP-DAC '98*, pp.289-294, May 1998.

[53] Y. Tang and X. Song, "Diagnosis for Arbitrarily Connected Parallel Computers," *IEEE Trans. oOn Computer*, Vol. 48, No. 7, July 1999.

[54] Y. Zorian, "Yield Improvement and Repair Trade-Off For Large Embedded Memories," *Proc. Of 3rd Design, Automation and Test in Europe (DATE 2000)*, Invited paper, March 27-30, 2000.

[55] I. Burgess, "Test and Diagnosis of Embedded Memory Using BIST," *EE-Evaluation Engineering*, Http://www.evaluationengineering.com/archive/articles/0300mem.htm, Feb. 14, 2002.

[56] A. L. Crouch, "Design-For-Test for Digital ICs and Embedded Core Systems," *Prentice-Hall PBR*, 1999.

[57] A. L. Crouch, M. Mateja, T. L. McLaurin, J. C. Potter, and D. Tran, "The Testability Features of the 3rd Generation Coldfire Family of Microprocessors," *Proceedings of the International Test Conference*, 1999.

[58] A. V. Goor, "Testing Semiconductor Memories, theory and practice," 1991.

[59] K. K. Saluja, G. S. Song, K. Kinoshita, "Built-in Self-Testing RAM, A Practical Alternative," *IEEE Design and Test*, pp.42-51, February 1987.

[60] T. Takeshima, *et al.*, "A 55-ns 16-Mb DRAM with Built-in Self Test Function Using Microprogram ROM," *IEEE Journal of Solid-State Circuits, Vol.25, No.4, pp.903-909*, August 1990.

[61] K. Koike, *et al.*, "A 30ns 64Mb DRAM with Built-in Self-Test and Repair Function," *IEEE ISSCC*, pp.150, 1992.

[62] P. Mazumder, and J. Patel, "An Efficient Built-in Self Testing for Random-Access Memory," *IEEE Transactions on Industrial Electronics*, Vol.36, No.2, pp.246-253, May 1989.

[63] J. Robertson, "Self-repairing ICs become feasible with greater integration, say experts," Http://www.siliconstrategies.com/stories/7k05bist.htm, February 15, 2002.

[64] D. Barkin, "Built-in Self Test of Dram Chips," Http://www.eecs.harvard.edu/cs245/papers/davidb.html, Feb. 2002.

[65] W. Mangione-Smith and B. Hutchings, "Configurable computing: The road ahead," *Reconfigurable Architectures Workshop*, 1997.

[66] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Efficiently supporting fault-tolerance in FPGAs," *ACM/SIGDA Sixth int. Symp. Field-Programmable Gate Arrays*, 1998.

[67] M.J. Wirthlin and B. L. Hutchings, "A dynamic instruction set computer," *Proc. IEEE Symp. FPGA's for Custom Computing Machines*, 1995.

[68] E. Tau, D. Chen, and I. Eslick *et al.*, "A first generation DPGA implementation," *Proc. Third Canadian Workshop on Field-Programmable Devices*, pp.138-143, 1995.

[69] R. Bittner and P. Athanas, "Wormhole run-time reconfiguration," *ACM/SIGDA Sixth int. Symp. Field-Programmable Gate Arrays*, 1997.

[70] H. Koile et al., "A 30nsec 64Mb DRAM with built-in self-test and repair function," *Int. Solid State Circuits Conf.*, pp.150-151, 1992.

[71] R. Trueuer and V. K. Agarwal, "Built-in self-diagnosis for repairable embedded RAMs," *IEEE Design and Test of Computers*, pp.24-33, 1993.

[72] T. Chen and G. Sunada, "Design of a self-testing and self-repairing structure for highly hierarchical ultra large capacity memory chips," *IEEE Trans. VLSI Syst., vol. 1*, pp.88-97, June 1993.

[73] J. R. Day, "A fault-driven, comprehensive redundancy algorithm," *IEEE Design and Test of Computers*, pp.35-44, June 1985.

[74] R. W. Haddad, A. T. Dahbura, and A. B. Sharma, "Increased throughput for the testing and repair of RAMs with redundancy," *IEEE Trans. Computers, vol. 40*, pp.154-166, Feb. 1991

[75] N. Hasan and C. L. Liu, "Minimum fault coverage in reconfigurable arrays," *Digest of Papers, vol.FTCS-18*, pp.348-353, June 1998.

[76] D. K. Bhavsar, "An algorithm for row-column self-repair of RAM's and its implementation on the alpha 21 264," *IEEE Int. Test Conf.*, pp.311-318, 1999.

[77] O. S. Bair *et al.*, "Method and apparatus for configurable build-in self-repairing of Asic memories design," *US Patent 5 577 050*, Nov. 19, 1999.

[78] A. Kablanian *et al.*, "Built-in self repair system for embedded memories," *US Patent 5 764 878*, Jun. 9, 1998.

[79] I. Kim, Y. Zorian, and G. Komoriya *et al.*, "Built-in self repair for embedded high density SRAM," *IEEE Ent. Test Conf.*, pp.1112-1119, 1998.

[80] A. Benso, S. Chiusano, G. D. Natale, and P. Prinetto, "An On-Line BIST RAM Architecture With Self-Repair Capabilities," *IEEE Transactions on Reliability, Vol 51, No.1*, pp.123-128, March 2002.

[81] R. David, A. Fuintes, and B. Courtois, "Random Pattern Testing Versus Deterministic Testing of RAM's," *IEEE Transactions on Computers*, Col.36, No.5, pp.637-650, May 1989.

[82] M. Franklin, and K. K. Saluja, "An Algorithm to test RAMS for Physical Neighborhood Pattern Sensitive Faults," *IEEE International Test Conference*, pp.675-684, 1991.

[83] J. Desposito, "Innovative Algorithm Allows Row-Column Self-Repair Of RAMs," *International Test Conference 1999 Proceedings*, Vol.47, No.25, December 6, 1999.

[84] A. Kablanian, "The STAR SRAM Embedded Memory," *Semiconductor News*, July 2001.

[85] C. R. Cassady, and E. A. Pohl, “Introduction to Repairable-System Modeling,” *2002 Annual RELIABILITY and MAINTAINABILITY Symposium*, Tutorial Notes, January 2002.

[86] J. J. McCall, “Maintenance Policies for Stochastically Failing Equipment: A Survey,” *Management Science*, Vol.11, No.5, pp.493-524, 1965.

[87] W. P. Pierskalla, “A Survey of Maintenance Models: The Control and Surveillance of Deteriorating Systems,” *Naval Research Logistics Quarterly*, Vol.23, No.3, pp.353-388, 1976.

[88] S. Osaki, and T. Nakagawa, “Bibliography for Reliability and Availability of Stochastic Systems,” *IEEE Transactions on Reliability*, Vol.25, pp.284-287, 1976.

[89] Y. S. Sherif, and M. L. Smith, “Optimal Maintenance Models for Systems Subject to Failure – A Review,” *Naval Research Logistics Quarterly*, Vol.28, pp.47-74, 1981.

[90] C. Valdez-Flores, and R. M. Feldman, “Survey of Preventive Maintenance Models for Stochastically Deteriorating Single-Unit Systems,” *Naval Research Logistics*, Vol.36, No.4, pp.419-446, 1989.

[91] D. I. Cho, and M. Parlar, “A Survey of Maintenance Models for Multi-Unit Systems,” *European Journal of Operational Research*, Vol.51, pp.1-23, 1991.

[92] R. Dekker, “Applications of Maintenance Optimization Models: A Review and Analysis,” *Reliability Engineering and System Safety*, Vol.51, No.3, pp.229-240, 1996.

[93] R. E. Barlow, and F. Proschan, “Mathematical Theory of Reliability,” *John Wiley & Sons, Inc.*, 1965.

[94] W. F. Rice, C. R. Cassady, and J. A. Nachlas, "Optimal Maintenance Plans under Limited Maintenance Time," *Industrial Engineering Research '98 Conference Proceedings*, 1998.

[95] C. R. Cassady, W. P. Murdock, and E. A. Pohl, "A Deterministic Selective Maintenance Model for Complex Systems," *Recent Advances in Reliability and Quality Engineering*, pp.311-325, 2001.

[96] C. R. Cassady, E. A. Pohl, and W. P. Murdock, "Selective Maintenance Modeling for Industrial Systems," *Journal of Quality in Maintenance Engineering*, Vol.7, No.2, pp.104-117, 2001.

[97] C. R. Cassady, W. P. Murdock, and E. A. Pohl, "Selective Maintenance for Support Equipment Involving Multiple Maintenance Actions," *European Journal of Operational Research*, Vol.129, No.2, pp.252-258, 2001.

[98] H. Ascher, and H. Feingold, "Repairable Systems Reliability," *Marcel Dekker, Inc.*, 1984.

[99] S. M. Ross, "Introduction to Probability Models, Fourth Edition," *Academic Press, Inc.*, 1989.

[100] R. E. Barlow, "System Reliability Analysis: Foundation," *Electronic Systems Effectiveness and Life Cycle Coasting, NATO ASI Series*, Vol. F3, pp.2-24, 1983.

[101] L. J. Bain and M. Engelhardt, "Introduction to Probability and Mathematical Statistics Second Edition," *PWS-Kent Publishing Company*, pp.1-170, 1992.

[102] A. Dubi, "Modeling of Realistic Systems With the Monte-Carlo Method – Unified System Engineering Approach," *2002 Annual RELIABILITY and MAINTAINABILITY Symposium*, Tutorial Notes, January 2002.

[103] J. B. Dugan, "Fault-Tree Analysis of Computer-Based Systems," *2002 Annual RELIABILITY and MAINTAINABILITY Symposium*, Tutorial Notes, January 2002.