TRANSISTOR-LEVEL DEFECT-TOLERANT TECHNIQUES FOR RELIABLE DESIGN AT THE NANOSCALE

BY

FARHAN KHAN

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

June 2009

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **FARHAN KHAN** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING.**

Thesis Committee

Dr. Aiman H. El-Maleh (Adviser)

Dr. M. E. S. Elrabaa (Co-adviser)

Dr. Sadiq M. Sait (Member)

Dr. Adnan A. Gutub (Member)

Dr. Abdelhafid Bouhraoua (Member)

Dr. Adnan A. Gutub
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

14/9/09
Date

Dedicated to my beloved father Mr. Abdul Aziz Khan

ACKNOWLEDGMENTS

In the name of Allah, the Most Beneficent, the Most Merciful

All praise be to Allah (The One and The Only Creator of everything) for His limitless blessings. May Allah bestow peace and His choicest blessings on the last prophet, Muhammad (Peace Be Upon Him), his family (May Allah be pleased with them), his companions (May Allah be pleased with them) and his followers.

I would like to express my profound gratitude and appreciation to my thesis committee chairman and adviser, Dr. Aiman Helmi El-Maleh, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas e.g., digital system design, modeling, synthesis, testing, fault-tolerance. Dr. El-Maleh is the one professor who truly made a difference in my life. It was under his tutelage that I developed a focus and became interested in research. He provided me with direction, intellectual support and became more of a mentor than a professor. It was through his persistence, understanding and kindness that I completed my degree. I doubt that I will ever be able to convey my appreciation fully, but I owe him my eternal gratitude.

I am grateful to Computer Engineering department Chairman Dr. Adnan Abdul-Aziz Gutub for providing me an opportunity to work on this thesis project with my adviser. Without his support and understanding at the very initial stages, this thesis work would have been impossible.

I would like to thank my co-adviser Dr. Muhammad El-Rabaa and the other member of my committee Dr. Abdelhafid Bouhraoua for the guidance they provided at all levels of the coursework projects and thesis research. Finally, I would like to thank Dr. Sadiq Mohammed Sait for his very useful suggestions and for taking out time from his very busy schedule to serve as committee member.

I would also like to thank my friends at Computer Engineering Department, particularly Ahmed Al-Masri, Abdul Rahman Elshafei and Syed Usama Idrees for our exchange of knowledge, ideas, skills, and venting of frustration during Master's studies, which helped enrich the experience. I also sincerely appreciate the help provided by my friends Ihab Hawari, Omair Khan, Ali Zaidi, Rahil, Zeeshan, Babar, Monim, Khizer, Zeehasham and Asif during my stay in the university.

I would also like to thank my family for the support they provided me throughout my life and in particular, I must acknowledge my beloved father and best friend, Mr. Abdul Aziz Khan, without whose love and encouragement, I would not have accomplished anything worthwhile in my career.

In addition, I acknowledge that this research would not have been possible without the support and assistance provided by the King Fahd University of Petroleum & Minerals.

TABLE OF CONTENTS

LIST OF TABLES **x**

LIST OF FIGURES **xii**

ABSTRACT (ENGLISH) **xvii**

ABSTRACT (ARABIC) **xix**

1 INTRODUCTION AND MOTIVATION **1**

1.1 Introduction 1

1.2 Motivation 3

1.3 Techniques for Reliable Design of Nanoelectronics 4

1.4 Thesis Objectives 5

1.5 Thesis Contributions 5

1.6 Thesis Organization 7

2 LITERATURE REVIEW **9**

2.1 Introduction 9

2.2 Definitions 10

2.2.1 Defects, Faults and Errors 10

2.2.2 Defect (or Fault) Models 11

2.2.3 Yield 14

2.2.4 Reliability 14

2.2.5 Fault Tolerance 16

2.2.6 Defect Tolerance 17
2.3 Defect-Tolerant Design Techniques 19
2.3.1 Von Neumann's Multiplexing 19
2.3.2 N-tuple Modular Redundancy (NMR) & Triple Modular Redundancy (TMR) 21
2.3.3 Intervowen Redundant Logic & Quadded Logic 24
2.3.4 N-tuple Intervowen Redundancy (NIR) & Triple Intervowen Redundancy (TIR) 29
2.4 Defect Avoidance Design Techniques 33
2.4.1 Mishra & Goldstein's Technique 33
2.4.2 Chen He and M. F. Jacome's Technique 35
2.5 Transient & Soft Error Mitigation Techniques 37
2.5.1 Single Event Upsets and Single Event Transients 38
2.5.2 Single Event Upset Mitigation Techniques 43
2.5.3 Single Event Upset Mitigation Techniques for FPGAs . . . 50
2.5.4 Empirical Model for Soft Error Rate Estimation 54
2.5.5 Soft-Spot Analysis 54
2.6 Defect-Tolerant Crossbar Design Techniques 60
2.6.1 Crossbar Architecture 60
2.6.2 Tahoori's Defect-Tolerant Design Techniques for 2D Crossbars 65
2.6.3 Hogg and Snider's Defect Tolerant Design Technique . . . 69
2.6.4 DeHon and Naemi's Defect Tolerant Design Technique . . 73
2.7 Defect-Tolerant FPGA Design Techniques 76
2.7.1 Field Programmable Gate Array Architecture 76
2.7.2 Categorization of Defect and Fault-Tolerant Techniques for FPGAs 78
2.7.3 Survey of Defect-Tolerant Techniques for FPGAs 82

3 DEFECT-TOLERANT N^2-TRANSISTOR STRUCTURES 84

3.1 Introduction 85

3.2 N^2-Transistor Structures 86
3.2.1 Quadded-Transistor Structure 87
3.2.2 Nona-Transistor Structure 93
3.3 Experimental Results 100
3.3.1 Stuck-Open and Stuck-Short Defect Analysis 103
3.3.2 Bridging Defect Analysis 108
3.3.3 Hybridization of Quadded and Nona-Transistor structures with TIR and TMR 110
3.4 Summary 112

4 TRANSIENT AND SOFT ERROR MITIGATION USING QUADDED-TRANSISTOR STRUCTURE 113
4.1 Introduction 114
4.2 Quadded Modular Redundancy Technique 115
4.3 Quadded-Transistor based SEU Mitigation Technique (QT16) . . 118
4.4 Gate-specific Quadded-Transistor based SEU Mitigation Technique (QT8) 118
4.5 Experimental Results 122
4.5.1 Quadded Modular Redundancy Technique Analysis 122
4.5.2 Quadded-Transistor based SEU Mitigation Technique Analysis (QT16) 147
4.5.3 Gate-specific Quadded-Transistor based SEU Mitigation Technique Analysis (QT8) 148
4.5.4 Reversed Gate-specific Quadded-Transistor based SEU Mitigation Technique Analysis (QT8R) 149
4.5.5 Circuit Reliability Comparison of QT(QMR1), QMR3, TMR9, QT16 and QT8 Techniques 150
4.5.6 Circuit Area Comparison of QT, QMR, TMR, QT16 and QT8 Techniques 152
4.6 Summary 153

5 DEFECT-TOLERANT CROSSBAR DESIGN TECHNIQUE 154

5.1 Introduction 154

5.2 Multi-Crosspoint Architecture 157

5.2.1 Quadded MCP Architecture 157

5.2.2 Nona MCP Architecture 164

5.3 Experimental Results 165

5.3.1 Reliability Analysis 166

5.3.2 Area Analysis 174

5.4 Summary 177

6 DEFECT-TOLERANT FPGA DESIGN TECHNIQUE 179

6.1 Introduction 179

6.2 Defect-Tolerant CLBs for FPGAs 180

6.3 Experimental Results 182

6.3.1 Reliability Analysis 182

6.3.2 Area Analysis 185

6.4 Summary 187

7 CONCLUSION 188

7.1 Conclusion 188

7.2 Future Work 191

REFERENCES 192

VITAE 206

LIST OF TABLES

3.1 Comparison of circuit failure probability between quadded-transistor structure and quadded logic approaches for stuck-open and stuck-short defects. 106

3.2 Comparison of circuit reliability between quadded-transistor structure and quadded logic approaches for stuck-open and stuck-short defects. 107

3.3 Circuit failure probability for the nona-transistor structure approach for stuck-open and stuck-short defects. 108

3.4 Circuit reliability for the nona-transistor structure approach for stuck-open and stuck-short defects. 109

3.5 Comparison of circuit failure probability between quadded-transistor structure and quadded logic approaches for bridging defects. 110

4.1 Comparison of circuit reliability between QMR and TMR techniques for a module size of 1 (i.e., full QT implementation). . . . 143

4.2 Comparison of circuit reliability between QMR and TMR techniques for a module size of 3. 143

4.3 Comparison of circuit reliability between QMR and TMR techniques for a module size of 5. 145

4.4 Comparison of circuit reliability between QMR and TMR techniques for a module size of 7. 145

4.5 Comparison of circuit reliability between QMR and TMR techniques for a module size of 9. 146
4.6 Comparison of circuit reliability between quadded-transistor based technique and TMR9 technique for SEU mitigation. 148
4.7 Comparison of circuit reliability between gate-specific quadded-transistor based technique and TMR9 technique for SEU mitigation. 149
4.8 Comparison of circuit reliability between reversed gate-specific quadded-transistor based technique and QT8 technique for SEU mitigation. 150
4.9 Circuit area comparison of QT, QMR, TMR, QT16 and QT8 techniques. 153

5.1 Comparison of circuit reliability between quadded MCP and monomorphism-based reconfiguration approaches. 172
5.2 Comparison of circuit reliability between nona MCP and monomorphism-based reconfiguration approaches. 173
5.3 Crossbar area in terms of number of crosspoints for the monomorphism-based reconfiguration architecture. 175
5.4 Crossbar area in terms of number of crosspoints for the quadded MCP architecture. 176
5.5 Crossbar area in terms of number of crosspoints for the nona MCP architecture. 177

6.1 Comparison of circuit failure probability between QT CLB and 2 spares based reconfiguration approaches. 186
6.2 Comparison of circuit failure probability between QT CLB and 3 spares based reconfiguration approaches. 186
6.3 Comparison of area in terms of number of transistors and CLBs for QT based CLB approach and 2 and 3 spares based approach. . . . 187

LIST OF FIGURES

1.1 Growth of transistor counts for Intel processors (dots) and Moore's law (vertical log scale). 2

2.1 Von Neumann NAND Multiplexing. 20
2.2 A Triple Modular Redundant (TMR) structure. 23
2.3 Nonredundant complementary half adder implemented with NAND logic. 26
2.4 Quadded implementation of the complementary half adder. 27
2.5 TIR implementation of the complementary half adder. 31
2.6 TMR configuration of the TIR complementary half adder. 32
2.7 Three-level design hierarchy showing abstractions in the form of region, mapping unit and component on the upper level and behavioral abstractions on the lower level. 36
2.8 Upsets hitting combinational and sequential logic. 38
2.9 Single Event Upset (SEU) effect in a SRAM Memory cell. 39
2.10 Single Event Transient (SET) Effect in Combinational Logic based on [53]. 40
2.11 Single Event Upset (SEU) effect in a SRAM Memory cell. 42
2.12 Full Time Redundancy. 45
2.13 TMR implemented in the entire device. 46
2.14 TMR memory cell with single voter. 47
2.15 TMR memory cell with three voters and refreshing. 47

2.16 Full time redundancy scheme for combinational logic combined with full hardware redundancy in the sequential logic. 49
2.17 Full hardware redundancy scheme for combinational and sequential logic. 49
2.18 Duplication to mitigate SET in combinational logic. 50
2.19 Time redundancy to mitigate SET in combinational logic. 51
2.20 Example of INVERTER logic with the code word state preserving (CWSP) in the duplication and time redundancy to mitigate SET in combinational logic. 51
2.21 The effective noise window. 56
2.22 Automatic soft-spot analysis. 59
2.23 Schematic view of a molecular crossbar from two different perspectives. 61
2.24 Implementing the AND/OR function X = A + BC with a diode crossbar and resistor. 63
2.25 (a) 4 x 4 2D nanoscale crossbar (b) Bipartite graph representation. 67
2.26 Resource Allocation: Searching for a monomorphism between circuit and a crossbar graph. 70
2.27 (a) A logic array of NanoPLA (b) Programmed logic array. 74
2.28 (a) Crosses show defective junctions (b) Graph of the OR-term nanowiores and OR functions (c) One possible assignment. 75
2.29 Simplified example of an FPGA with 16 CLBs. 79

3.1 (a) Transistor in original gate implementation, (b) First quadded-transistor structure, (c) Second quadded-transistor structure. . . . 88
3.2 Defect-tolerant N^2-transistor structure. 92
3.3 (a) Transistor in original gate implementation, (b) First nona-transistor structure, (c) Second nona-transistor structure. 94
3.4 Gate reliability comparison between quadded-transistor structure (Q), nona-transistor structure (N) and conventional CMOS. . . . 101

3.5 Reliability obtained both theoretically (t) and experimentally (e) based on quadded-transistor structure and stuck-open and stuck-short defects. 104
3.6 Reliability obtained both theoretically (t) and experimentally (e) based on nona-transistor structure and stuck-open and stuck-short defects. 105
3.7 Comparison of circuit failure probability for an 8-stage cascaded half-adder circuit for stuck-open and stuck short defects. 111

4.1 Quadded-Transistor based technique for permanent defects. 116
4.2 Quadded Modular Redundancy technique for a simple 2-input circuit. 117
4.3 Quadded-Transistor based technique for SEU Mitigation. 119
4.4 Gate-specific connections for NAND gate to mask faulty transistors. 121
4.5 Triple Modular Redundancy technique for single stage of 2-input complementary half adder. 125
4.6 Quadded Modular Redundancy technique for single stage of 2-input complementary half adder. 126
4.7 Comparison of circuit failure probability for a 1-stage complementary half-adder circuit for transient faults. 127
4.8 Comparison of circuit failure probability for a 2-stage cascaded complementary half-adder circuit for transient faults. 127
4.9 Comparison of circuit failure probability for a 4-stage cascaded complementary half-adder circuit for transient faults. 128
4.10 Comparison of circuit failure probability for a 8-stage cascaded complementary half-adder circuit for transient faults. 128
4.11 Comparison of circuit failure probability for a 16-stage cascaded complementary half-adder circuit for transient faults. 129
4.12 Comparison of circuit failure probability for a 32-stage cascaded complementary half-adder circuit for transient faults. 129

4.13 Comparison of area in terms of number of transistors for 1, 2, 4, 8, 16 and 32-stage cascaded complementary half adders for QMR and TMR implementation. 130
4.14 Example Circuit. 134
4.15 Application of modular TMR algorithm on example circuit for a module size of 1. 135
4.16 Application of modular TMR algorithm on example circuit for a module size of 2. 136
4.17 Application of modular TMR algorithm on example circuit for a module size of 3. 137
4.18 Application of modular TMR algorithm on example circuit for a module size of 1. 140
4.19 Application of modular TMR algorithm on example circuit for a module size of 2. 141
4.20 Application of modular TMR algorithm on example circuit for a module size of 3. 142
4.21 Comparison of circuit reliability for QMR and TMR techniques module sizes of 1, 3, 5, 7 and 9. 144
4.22 Comparison of circuit reliability of all approaches for ISCAS benchmarks for injecting 0.1% faults. 151
4.23 Comparison of circuit reliability of all approaches for ISCAS benchmarks for injecting 0.5% faults. 152

5.1 Crossbar implementation for a simple function $X = A + BC$. . . . 158
5.2 Multi-crosspoint architecture using row and column redundancy for a simple function $X = A + BC$ for $k = 2$. 159
5.3 Allowable defect configuration in which the function will remain $X = A + BC$. 160
5.4 Allowable defect configuration in which the function will remain $X = A + BC$. 160

5.5 Allowable defect configuration in which the function will remain $X = A + BC$. 161
5.6 Allowable defect configuration in which the function will remain $X = A + BC$. 161
5.7 Obstructive defect configuration in which the function will become $X = 1$. 162
5.8 Obstructive defect configuration in which the function will become $X = BC$. 162
5.9 Obstructive defect configuration in which the function will become $X = 1$. 163
5.10 Obstructive defect configuration in which the function will become $X = 1$. 163
5.11 Multi-crosspoint architecture using row and column redundancy for a simple function $X = A + BC$ for $k = 3$. 165
5.12 A 3-bit adder which adds two 3-bit numbers (denoted as the bits $A_2A_1A_0$ and $B_2B_1B_0$, respectively) to produce a 4-bit sum (with bits $S_3S_2S_1S_0$). 169
5.13 A 3-bit adder implemented as 2-level logic in a single diode crossbar. 170
5.14 Reliability comparison of quadded, nona and monomorphism-based approaches for 3-bit adder shown in Figure 5.12. 170
5.15 Reliability comparison of quadded, nona and monomorphism-based approaches for 3-bit adder shown in Figure 5.13. 171

6.1 A basic FPGA logic block. 181
6.2 Schematic of 4-input LUT. 181
6.3 Comparison of circuit failure probability for alu4 benchmark. . . . 185

THESIS ABSTRACT

NAME:	Farhan Khan
TITLE OF STUDY:	Transistor-Level Defect-Tolerant Techniques for Reliable Design at the Nanoscale
MAJOR FIELD:	Computer Engineering
DATE OF DEGREE:	June 2009

Nanoelectronics based systems offer an attractive alternative for present day CMOS technology. It is estimated that nanoelectronics can achieve very high densities (billion devices per centimeter square) and operate at very high frequencies. With such high device densities, nanotechnology has the potential to take electronic circuits to the next higher level of integration. Nanoelectronic devices like carbon nanotubes (CNT), silicon nanowires (NWs) and quantum dot cells have already been demonstrated successfully by researchers. These devices are normally manufactured using bottom-up self-assembly fabrication process which results in higher defect densities in comparison to conventional lithography-based VLSI fabrication. Therefore, there is a renewed interest in using hardware redundancy to mask faulty behavior in order to increase reliability of nanoelectronic components.

In this thesis, detailed investigation of a recently proposed transistor-level defect-tolerant technique for nanoelectronics is performed. The investigated technique replaces each transistor by a N^2-transistor structure $(N = 2, 3, .., k)$ and guarantees defect tolerance of all permanent defects of multiplicity $\leq (N - 1)$ in each transistor structure. The theoretical and experimental analysis for the defect tolerance of stuck-open and stuck-short defects for quadded-transistor structure i.e.,$(N = 2)$ is extended for the nona-transistor structure i.e.,$(N = 3)$. Comparison of defect tolerance of transistor structures $(N = 2, 3)$ against other techniques like Triple Intervowen Redundancy (TIR) and Quadded Logic (QL) is carried out experimentally. It is shown that the combinations of defect tolerance at both the transistor level and gate level have significantly improved circuit defect tolerance. For this, combination of Triple Modular Redundancy (TMR) with majority gate implemented with N^2-transistor structure is investigated in this thesis.

Application of N^2-transistor structure for handling soft errors is also investigated and a novel approach based on quadded-transistor structure is proposed. Finally, techniques for the defect tolerance of logic implemented using crossbar switches and FPGAs are also investigated.

Keywords: *Defect Tolerance, Quadded Logic, Quadded-Transitor structure, Triple Modular Redundancy , Triple Intervowen Redundancy, Quadded Modular Redundancy, Defect-tolerant Nanoscale Crossbars, Defect-tolerant FPGAs*

ملخص الرسالة

الاسم: فرحان خان

عنوان الدراسة: تقنية احتمال العيوب على مستوى الترانزستور للتصميم الموثوق بمقياس النانو

التخصص: هندسة الحاسب الآلي

سنة التخرج: يونيو 2009

توفر أنظمة إليكترونيات النانو بديلاً جذاباً لتقنية CMOS المستخدمة في هذه الأيام ، فمن المقدر أن إلكترونيات النانو يمكن أن تحقق درجات عالية من الكثافة (مليار ترانزستور لكل سنتيمتر مربع) وتعمل على ترددات عالية جداً. بهذه الكثافة العالية تملك تقنية النانو المقدرة لتطوير مستوى تكامل الدوائر الالكترونية إلى أعلى المستويات. الكترونيات النانو أجهزة مثل الكربون نانوتيوب (CNT) وأسلاك السليكون المتناهية الصغر (NWs) وخلايا نقطة الكم قد تم بالفعل توضيح البرهنة على عملهم بنجاح من قبل الباحثين. هذه الأجهزة عادة ما تكون مصنوعة من أسفل إلى أعلى باستخدام تقنية التصنيع بالتجميع الذاتي والتي تنتج أجهزة بنسب خلل وأعطال عالية في هذه الأجهزة مقارنة بطريقة الطباعة الحجرية التقليدية في تصنيع دارات التكامل الفائق (VLSI). لذلك هناك اهتمام متجدد لاستخدام دوائر إلكترونية احتياطية أو كنسخ إضافية لحجب أخطاء تلك الدوائر في حال وجودها مما يزيد من فعالية وموثوقية مكونات الكترونيات النانو.

في هذه الأطروحة ، نقدم تحقيقاً مفصلاً لتقنية احتمال العيوب التصنيعية على مستوى الترانزستور والتي تم عرضها مؤخراً. هذه التقنية تقوم باستبدال كل ترانزستور بتشكيلة أو بنية مكونة من ($ن^2$) من الترانزستورات بحسب مستوى السماحية للعيوب المطلوبة بحيث تكون (ن = 2 ، 3 ، 4 ، ...ك) ، هذه الترانزستورات الإضافية تضمن تغلب الدائرة الاكترونية على جميع العيوب الدائمة فيها بتعددية تكون فيها عدد تلك العيوب أقل أو تساوي (ن-1) المختارة في كل بنية واحدة من الترانزستورات. التحليل النظري والتجريبي لاحتمال العيوب عندما يكون هذا العيب عالقاً كدائرة فتح أو عالقاً كدائرة غلق لتشكيلات الترانزستورات الرباعية (ن = 2) قد تم تمديده للتشكيلات التساعية (ن = 3) في هذه الرسالة ، كذلك تمت مقارنة احتمالية العيوب لتشكيلات الترانزستورات (ن = 2 ، 3) بتلك التقليدية مثل التكرار الثلاثي المتشابك (TIR) والدوائر المنطقية الرباعية (QL). عن طريق إجراء التجارب ، يتبين أن التركيبات التي في مقدورها احتمال وجود عيوب على مستوى الترانزيستور أو مستوى البوابات المنطقية أدت إلى تحسين مستوى موثوقية الدوائر الإلكترونية بشكل ملحوظ وملفت ، لهذا فإنه تم بحث وتحقيق القيام بدمج تقنية الوحدات الثلاثية مع تنفيذ بوابة الغالبية بين الإشارات الرقمية بتقنية ($ن^2$) للحصول على موثوقية أعلى للدوائر الرقمية.

واحد من التطبيقات المستخدمة لتشكيلات ($ن^2$) من الترانزستورات هو معالجة الأخطاء الخافتة تم البحث فيه و تم تقديم طريقة جديدة استناداً على تقنية الترانزستور الرباعية المقترحة في هذه الأطروحة ، أخيراً تقدم الأطروحة تحقيقاً على استخدام تقنيات احتمال العيوب للدوائر المنطقية في شبكة الخطوط المستعرضة و مصفوفة البوابات المنطقية القابلة للبرمجة (FPGA).

الكلمات الرئيسية : احتمال العيوب ، المنطق الرباعي ، بنية الترانزستورات الرباعية ، الوحدات الثلاثية المتكررة ، التكرار الثلاثي المتشابك ، الوحدات الرباعية المتكررة ، شبكة الخطوط المستعرضة المقاومة للعيوب ، مصفوفة البوابات المنطقية القابلة للبرمجة المقاومة للعيوب.

CHAPTER 1

INTRODUCTION AND MOTIVATION

1.1 Introduction

In the past few decades, the rapid pace with which microelectronics has progressed is driven by the continual miniaturization of CMOS technology. This miniaturization of CMOS technology is manifested in the popular Moore's law which states that the number of electronic components per chip doubles every 18 months (formerly 2 years). The growth by Moore's law is shown in Figure 1.1. As the CMOS technology enters the nanometer scale, quantum mechanical effects come into play creating many technological challenges for further scaling of CMOS devices [34]. This has triggered research in two dimensions. One dimension of research is the invention and investigation of novel CMOS structures to achieve more scaling in current CMOS technology. The other dimension of work is the exploration of

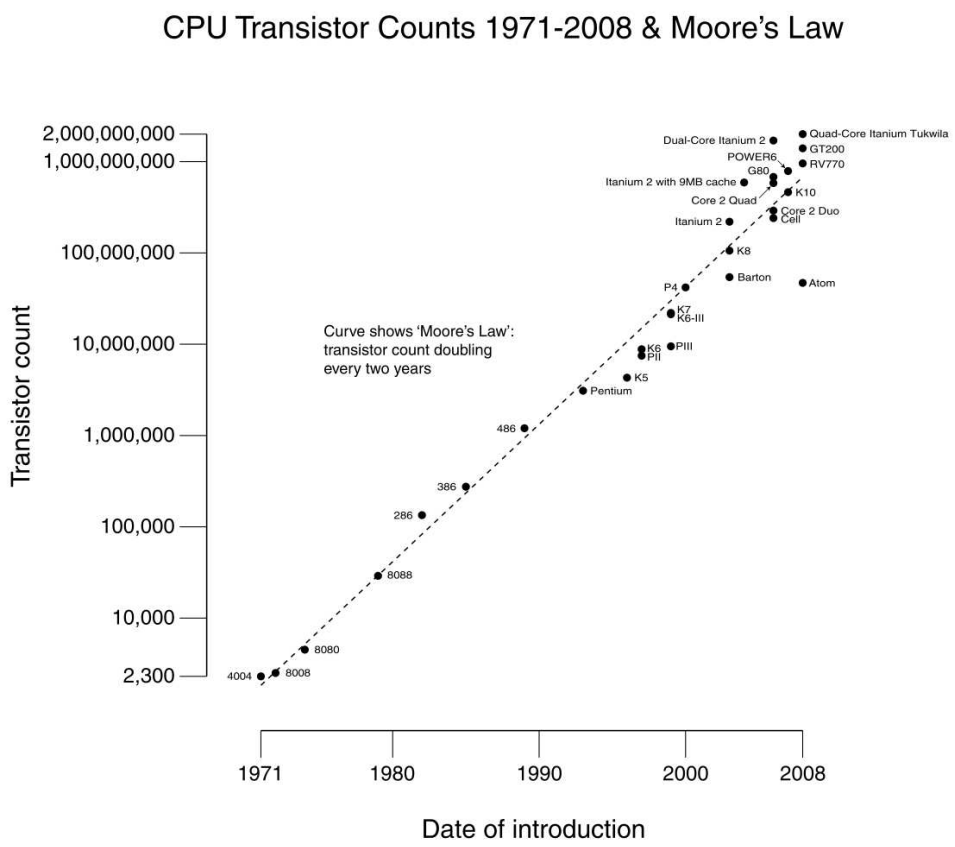

Figure 1.1: Growth of transistor counts for Intel processors (dots) and Moore's law (vertical log scale).

alternative technologies for information processing [34]. Nanoelectronic devices and circuits based on nanotechnology-based fabrication are expected to offer the extra density and performance to take electronic circuits to the next higher level of integration stage. It is estimated that nanoelectronics can achieve very high densities (10^{12} devices per cm^2) and operate at very high frequencies (of the order of THz) [1]. Several research groups have proposed and successfully demonstrated novel nanoelectronic devices at the logic circuit level. These devices include reso-

nant tunneling diodes (RTDs), single electron tunneling (SET) devices, quantum cellular automata (QCA), rapid single flux quantum (RSFQ), supercon, carbon nanotubes (CNTs), silicon nanowires (SiNWs), molecular nanoelectronics, quantum dot cells etc. [2, 3, 4, 5, 34]. These nanoelectronic devices share one or more characteristics such as extremely small dimensions, high switching speed, low power consumption, ease of fabrication and very good scaling potential [34]. It is expected in near future that one or more of these devices will be integrated on a CMOS platform, serving as complementary components to CMOS. Moreover, in the long run, one can expect nanoelectronics to serve as an alternative to present-day CMOS technology [34]. Nanoelectronic devices are normally manufactured using bottom-up self assembly fabrication processes as compared to normal CMOS fabrication which uses top-down lithography based fabrication. Due to fabrication regularity imposed by the self-assembly fabrication process, nanoelectronic devices are presently being manufactured as regular structures like two-dimensional (2-D) crossbars.

1.2 Motivation

Nanoscale devices whether manufactured using self-assembly or lithograpy-based processes have several characteristics which impose limitations on their use in nanoelectronic architectures. The most prominent characteristics are the devices' lower reliability and higher defect rates. This low reliability and higher defect rates of nanoelectronic devices arise from two sources [34].

- One source is the inherent imprecision and randomness in the bottom-up manufacturing process which results in a large number of defective devices during the fabrication process [34].

- The other source is the reduced noise tolerance of these devices which can be responsible for inducing device malfunctions by external influences like EMI, thermal perturbations, cosmic radiations etc [34].

Therefore, permanent defects may emerge during manufacturing process and transient errors can happen during the operation rendering nanoelectronic connections, wires and devices effectively unusable [4, 6, 7]. In order to address issues of unreliability in nanoelectronics and to ensure reliable system design and operation, defect tolerant design techniques need to be devised and applied for emerging nanoelectronic devices.

1.3 Techniques for Reliable Design of Nanoelectronics

The techniques for reliable design of nanoelectronics can be categorized as defect-tolerant and defect avoidance techniques. Defect-tolerant design techniques are based on adding redundancy in the design to mask faulty behavior due to defects or faults. However, defect avoidance techniques are based on identifying defects and bypassing them based on reconfiguration. Both these techniques are discussed in detail in Chapter 2. For defect-tolerant techniques, hardware redundancy can

be added at logic block level, gate level or transistor level. The work investigated in this thesis is based on adding redundancy at the transistor level. The proposed work will be discussed in detail in Chapters 3, 4, 5 and 6.

1.4 Thesis Objectives

The main goal of this work is to develop novel transistor-level defect-tolerant techniques that can be employed at nanoscale to afford enhanced reliability to the nanoelectronic circuits. In addition, defect-tolerant techniques have also been developed for specific nanoscale architecture like crossbars and FPGAs.

1.5 Thesis Contributions

The work presents the results of investigation related to the objectives mentioned in previous section. In particular, the main contributions can be summarized as follows:

- A recently proposed transistor-level defect-tolerant technique called Quadded-Transistor structure [31, 32] is studied in detail and is extended to develop another transistor-level defect-tolerant technique called Nona-Transistor technique. Both theoretical and experimental analysis is performed for tolerating transistor stuck-open and stuck-short defects. Reliability and failure rate analysis of Nona-Transistor technique and Quadded Logic technique for transistor stuck-open and stuck-short defects has proved

that Nona-Transistor technique has outperformed Quadded Logic technique in terms of defect tolerance. Nona-Transistor technique has also shown better reliability than Quadded-Transistor technique at the cost of higher area.

- Hybridization of Nona-transistor technique with TMR is proposed in order to achieve higher reliability following the idea of hybridization of quadded-transistor technique with TMR as proposed in [31, 32].

- A new transistor-level technique is proposed for mitigating transient and soft errors in digital circuits. The proposed technique is based on selective application of the Quadded-Transistor structure and is called Quadded Modular Redundancy(QMR). Simulation-based comparison of QMR with TMR for transient faults has shown that QMR affords more tolerance to transient faults in comparison to TMR. Two more variants of the same technique are also explored. Experimental analysis has shown that the proposed techniques are more efficient in terms of defect-tolerance than TMR but at the cost of higher area.

- A new defect-tolerant architecture for implementing logic circuits on partially defective nanoscale crossbars is proposed. The proposed crossbar architecture called Multi-crosspoint(MCP) architecture uses row and column redundancy in order to achieve higher defect tolerance in nanoscale crossbar-based circuits. A comparison of the proposed architecture is made with the monomorphism based reconfiguration algorithm for defect-tolerant crossbar design for a number of circuits and the experimental analysis has shown that

the MCP architecture performs better than monomorphism based approach on circuits with more dense product terms.

- Transistor-level defect-tolerant FPGA design technique is also explored for realizing reliable Configurable Logic Blocks (CLBs). Simulation based comparison of QT based CLBs is performed with 2-spares and 3-spares based reconfiguration technique which shows that the QT based CLB affords better defect tolerance than 2 spares based technique but is inferior to 3-spares based technique.

1.6 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, a survey of defect-tolerant design techniques available in literature is presented. The chapter covers various techniques concerning the objectives of this work that are reported in literature, discussing relevant algorithms and architectures wherever required. The chapter starts with surveying gate-level defect-tolerant design techniques. After that, relevant literature pertaining to SEU mitigation in digital circuits is presented. Then defect-tolerant crossbar design techniques are briefly surveyed. At the end of the chapter, defect-tolerant FPGA design techniques are briefly surveyed.

Chapter 3 discusses in detail the proposed transistor-level defect-tolerant design techniques for masking the effects of transistor stuck-open, stuck-short and bridging defects. The chapter covers in detail the theoretical and experimen-

tal analysis of Quadded-Transistor and Nona-Transistor techniques along with a discussion on the simulation framework used for experimental analysis.

In Chapter 4, detailed discussion is presented on the proposed transistor-level defect-tolerant technique for mitigating Single Event Upsets in digital circuits. This is followed by a detailed discussion of the proposed defect-tolerant design technique for crossbar-based designs in Chapter 5. Chapter 6 presents the proposed defect-tolerant techniques for design of defect-tolerant FPGAs. This thesis ends with conclusion and some future directions in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, a survey of various defect-tolerant design techniques is reported. The chapter begins with definitions of important terms in the field of defect tolerant digital design. This is followed by a discussion of gate-level defect-tolerant design techniques like NAND Multiplexing, N-Modular Redundancy (NMR), Quadded Logic, N-tuple Intervowen Redundancy (Section 2.3). This is followed by a brief description of defect avoidance design techniques (Section 2.4). Soft and transient error mitigation techniques are discussed in Section 2.5. Section 2.6 describes algorithms for defect-tolerant crossbar design techniques. Defect-tolerant FPGA design techniques are briefly described in Section 2.7.

2.2 Definitions

2.2.1 Defects, Faults and Errors

Many terms are used to describe incorrectness in electronic systems. One may find that the terms defects, errors and faults are used in confusing ways. In the next few sub-sections, the definitions of these terms as defined in the book by Michael Bushnell [49] are presented.

Defect

A defect in the electronic system is the unintended difference between the implemented hardware and its intended design.

Some typical defects in VLSI chips are:

- Process Defects - missing contact windows, parasitic transistors, oxide breakdown.
- Material Defects - bulk defects (cracks, crystal imperfections), surface impurities.
- Age Defects - dielectric breakdown, elctro-migration etc.
- Package Defects - contact degradation, seal leaks.

Defects occur either during manufacture or during the use of devices. Repeated occurrence of the same defect indicates the need for improvement in the manufacturing process or the design of the device.

Fault

A representation of a "defect" at the abstracted function level is called a fault.

The difference between a defect and a fault is rather subtle. They are the imperfections in the hardware and function respectively.

Error

A wrong output signal produced by a defective system (or circuit) is called an error. An error is an "effect" whose cause is some "defect".

Fabrication defects, fabrication errors and physical failures are collectively termed as physical faults [50]. According to their stability in time, physical faults can be classified as:

- *Permanent faults*: They are those which are always present after their occurrence.
- *Intermittent faults*: They are those which exist only during some intervals.
- *Transient faults*: They are one-time occurrence (also known as Single Event Upsets (SEUs) or Single-Event Transients (SETs)) which are caused by a temporary change in some environment factor e.g., due to α-particle radiation etc.

2.2.2 Defect (or Fault) Models

While analyzing the defect tolerance of a circuit, the effect of defects in the circuit needs to be simulated. The effect of a production defect can be complex. Accu-

rate defect modeling based on layout and geometrical considerations is normally not an option when the effect of production defects is to be analyzed. For this reason, several defect(or fault) models have been proposed at different levels of abstraction. In the following, only those defect(or fault) models are defined which are relevant to our work. For other defect (or fault) models, the interested reader may refer to the book by Michael Bushnell [49].

- *Stuck-at defect (or fault) model*: It is based on assigning a fixed (0 or 1) value to a signal line in the circuit. A signal line is an input or an output of a logic gate or a flip-flop. The most popular forms are the single stuck-at faults i.e., stuck-at-1 and stuck-at-0.

- *Stuck-open and Stuck-short defect (or fault) model*: It is used for modeling transistor defects. In this model, a MOS transistor is modeled as an ideal switch and a defect is modeled as the switch being permanently either in the open (never conducting) or the shorted state(always conducting). In general, a MOS logic gate consists of more than one transistor. This defect model assumes just one transistor to be stuck-open or stuck-short. The stuck-open and stuck-short defect (or fault) model is also called transistor defect (or fault) model.

- *Bridging defect (or fault) model*: Usually modeled at the gate or transistor level, a bridging fault represents a short between a group of signals. The logic of the shorted net may be modeled as 1-dominant(OR bridge), 0-dominant(AND bridge) or intermediate, depending upon the technology in

which the circuit is implemented. Non-feedback bridging faults are combinational and their coverage by stuck-at fault tests is normally very high. It is not always the case with the feedback bridging faults that produce memory states in the otherwise combinational logic. Bridging faults are often used as examples of "defect-oriented faults".

- *Crosspoint defect(or fault) model*: It is used for modeling crosspoint defects (or faults) in the programmable logic arrays (PLAs). In the layout of a PLA, input and output variable lines are laid out perpendicular to the product lines. Crossing signal lines either form specific types of connections or remain unconnected at crosspoints, depending on the function implemented. There are two types of crosspoint defects (or faults). A missing crosspoint defect means a missing connection at a crossing where a connection was intended. An extra crosspoint defect means a faulty connection at a crosspoint where no connection was intended. Based on their effect on AND and OR planes of the PLA, the crosspoint defects (or faults) are further classified as shrinkage, growth, appearance and disappearance defects (or faults). A missing crosspoint in the AND plane is called a growth defect (or fault). An extra crosspoint in the AND plane is called a shrinkage defect (or fault). A missing crosspoint in the OR plane is termed as disppearance defect (or fault). An extra crosspoint in the OR plane is termed as appearance defect (or fault).

2.2.3 Yield

Yield can be defined as the ratio of the number of usable items after production to the number of potentially usable items [62]. The main contributor to low yield for ICs is defects during production and fabrication. Yield is an important measure because only usable items are sellable. Low yield can make production prohibitively expensive.

For chip production, the total yield is the product of wafer process yield, device yield and module test yield. Wafer process yield is the ratio of usable wafers. Device yield is the ratio of usable dies after photolithography and module test yield is the ratio of usable chips after packaging. Device yield is the most important component, and the only one that is dependent on the specific circuit [71].

Redundancy techniques, such as the ones explained in Section 2.3 to 2.7, can improve device yield by tolerating a certain amount of defects.

2.2.4 Reliability

The reliability of a system can be defined as the ability to perform the specified function under stated conditions [63].

For hardware systems, the most common way of evaluating reliability is to apply a probabilistic reliability function $R(t)$ that gives the probability that a system is working correctly between time 0 and time t, given certain conditions and correct behavior at time 0. If the failure rate of the system is constant over

time, the reliability function is $R(t) = e^{-\lambda t}$ where λ is the constant failure rate for one unit of time. When λt is small, $R(t) \approx 1 - \lambda t$.

In a system composed from several subcomponents, all of which must be working, the reliability of the system is given as $R = \prod_{c=1}^{n} R_c$ where R_c is the reliability of subcomponent c and n is the number of subcomponents. A defect-tolerant system can continue to operate despite a certain number of defects. For such systems, where not all subcomponents need to be working, more elaborate reliability calculations need to be performed or, more realistically for complex systems, Monte Carlo simulations need to be employed.

An alternative evaluation criterion is Mean Time To Failure ($MTTF$) which is the average time a system will run before failing. $MTTF$ is linked to the failure rate in the following way: $MTTF = (1/\lambda)$. If λ is the failure rate per hour, $MTTF$ is the average number of hours before failing.

When considering how reliable a system is in the presence of production defects, time is not relevant. $MTTF$ is therefore not applicable and reliability is simply $R = (1 - \lambda)$ where λ is the probability of failing under stated conditions. It should be noted that reliability in this case is similar, but not the same as yield. Yield is the percentage of chips that can be sold. Reliability is a probability of working given certain conditions. These conditions need not be directly linked to what actually causes unsellable chips. However, if the stated conditions are realistic and relevant for what constitutes a sellable chip, high reliability will lead to high yield [71].

2.2.5 Fault Tolerance

The term fault-tolerance first appeared in technical literature in 1967, defined as:

A system is fault-tolerant if its programs can be properly executed despite the occurrence of logic faults [64].

The prime motivation at that time was challenges in interplanetary exploration. Today, the importance of fault-tolerance is increasing, and, as mentioned in Chapter 1, fault-tolerance is a long term grand challenge of the semiconductor industry [65].

The objective of fault-tolerance is either to mask, or to recover from, faults once they have been detected [66]. It is in contrast to the second method of achieving system reliability, fault prevention, which seeks to eliminate all faults before the system is put to use. Complete fault prevention is impossible to achieve in practice and high degrees of prevention is costly [72].

Fault-tolerance is therefore commonly used in increasing system reliability, often in combination with partial fault prevention. Much research on reliable systems is concerned with the detection of faults using error detecting and correcting codes or fault-detecting and self-repairing circuits. The tolerance itself is achieved using redundancy techniques. Such techniques can be classified as hardware, time, information or software redundancy [66]. With hardware redundancy there are spare elements available to replace faulty ones. Time redundancy implies that elements still operating may perform the functions that were originally intended to be performed by now faulty elements [67]. With information redundancy, re-

dundant information is added to an existing data set, for instance by using codes that enable detection and correction of errors. Software redundancy uses techniques such as N-version programming [68], where N independently constructed programs run in parallel.

The research work reported in this thesis is concerned with only hardware redundancy to afford defect tolerance in digital circuits particularly at the nanoscale. Hardware redundancy can be implemented as either static or dynamic redundancy. As defined in the next section, out of these two types, our approach is the application of static hardware redundancy at the transistor level.

2.2.6 Defect Tolerance

A defect-tolerant circuit is a circuit that functions correctly even if there are defective subcomponents, for example defective transistors and/or wires. Defect tolerance can be seen as a special case of fault tolerance where only permanent defects are considered. Transient faults that do not result in permanent damage are not an issue [71].

A defect-tolerant circuit is a circuit that is designed to tolerate a certain amount of defective components. The term defect coverage refers to the percentage of all possible defects a defect tolerant system can tolerate. 100% defect coverage means that any possible single defect anywhere in the system is tolerated. Often, defect coverage is less than 100%, either because not all defect types are tolerated or because some parts of the system are not defect- tolerant [71].

In the beginning of the history of digital electronic circuits, logic was built from unreliable vacuum tubes. As a result, there was a significant amount of research on how to build reliable computers from unreliable components and many of the most well known defect tolerance techniques date from the early period of computing. After the introduction of the IC, failure rates dropped drastically and reduced the importance of defect and fault tolerance techniques, except for a few extreme cases such as for space exploration. Recent predictions on failure rates in future production processes have renewed interest in defect tolerance [71].

Defect tolerance is achieved through the use of redundancy techniques. Redundancy techniques relevant for tolerating hardware defects can be classified as static hardware redundancy and dynamic hardware redundancy [71].

- Static hardware redundancy involves having redundant hardware components connected in such way that defects are tolerated without any need to first detect the defects. They are briefly covered in section on Defect-Tolerant Design Techniques.

- Dynamic hardware redundancy involves first detecting a defect and then applying measures, for example reconfiguration, for avoiding the detected defect. They are briefly covered in section on Defect Avoidance Design Techniques.

2.3 Defect-Tolerant Design Techniques

Defect-tolerant digital system design techniques are based on the concept of adding redundancy in order to mask faulty behavior in the nanoelectronic components due to defects, faults or errors. Examples of defect-tolerant design techniques are Von Neumann Multiplexing, N-tuple Modular Redundancy (and its derivative Triple Modular Redundancy), Quadded Logic, N-tuple Intervowen Redundancy (and its derivative Triplicated Intervowen Redundancy).

2.3.1 Von Neumann's Multiplexing

In the 1950s, John von Neumann initiated the study of techniques for the design of reliable systems using redundant unreliable components [9]. In his multiplexing structure, von Neumann considered two types of basic logic, namely majority-voting and NAND logic. He duplicated each logic gate N times and replaced each input with a bundle of N lines, thus, each output bundle also had N lines. For NAND logic, the inputs from the first bundle randomly pair with those from the second bundle to form the input pairs of the duplicated NANDs (as illustrated in Figure 2.1). Instead of requiring all or none of the output bundle's lines to produce correct answers, von Neumann set a certain critical (or threshold) level Δ such that $0 < \Delta < 1/2$. If the number of lines carrying the correct signal was larger than $(1 - \Delta)N$, he interpreted it as a positive state of the bundle, if it was less than ΔN, he considered it a negative state. By using a massive duplication of unreliable components, von Neumann concluded that the construction can be

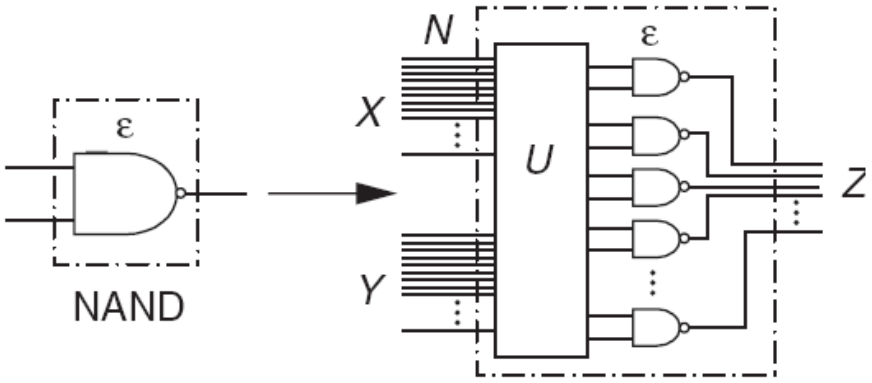

Figure 2.1: Von Neumann NAND Multiplexing.

reliable with a high probability if the failure probability of the gates (denoted by ε) is sufficiently low for example, lower than approximately 10^{-2} [9]. In general, von Neumann's construction requires a large amount of redundancy ($N > 10^3$) and a low error rate for individual gates. These features motivated extensive research efforts in later decades to find the complexity of redundancy required to cope with errors. It is shown in [20] that for deep logic with a gate failure probability $\varepsilon = 0.01$ and $N = 100$, it is possible to achieve circuit failure probability in the order of 10^{-6}. This required amount of redundancy is excessive and is considered impractical. In order to reduce this large amount of redundancy, the works in [21, 22] combine NAND multiplexing with reconfiguration.

Because CMOS devices became dominant in industry and showed an amazing performance in terms of reliability and scalability, chip designers never used von Neumann's multiplexing technique in practice. However, researchers have im-

plemented many redundancy techniques derived from von Neumann's proposal, such as triple modular redundancy (TMR) and error-correcting codes (ECC), in high-reliability applications and in memory circuits [11].

2.3.2 N-tuple Modular Redundancy (NMR) & Triple Modular Redundancy (TMR)

N-tuple modular redundancy (NMR) design of which TMR is the most-used particular case have been used as benchmarks for evaluating fault-tolerant approaches and have been implemented in VLSI for high-reliability applications. NMR techniques, generally implemented at the modular rather than gate level, use redundant components to mask fault effects.

An NMR system (also known as M-of-N system) is a system that consists of N modules and needs at least M of them for proper operation. Thus, the system fails when fewer than M modules are functional. The reliability of an NMR system as computed in [35] is presented next. The assumption is that the failures of the different modules are statistically independent and that there is no repair of failing modules. If $R(t)$ is the reliability of an individual module (the probability that the module is still operational at time t), the reliability of an NMR system is the probability that M or more modules are functional at time t. The system

reliability is therefore given by:

$$R_{NMR}(t) = \sum_{i=M}^{N} \begin{pmatrix} N \\ i \end{pmatrix} R^i(t)[1 - R(t)]^{N-i} \tag{2.1}$$

The assumption that failures are independent is the key assumption to the high reliability of NMR systems. Even a slight extent of positively correlated failures can greatly diminish their reliability. For example, suppose q_{cor} is the probability that the entire system suffers a common failure. The reliability of the system now becomes:

$$R_{NMR}(t) = (1 - q_{cor}) \sum_{i=M}^{N} \begin{pmatrix} N \\ i \end{pmatrix} R^i(t)[1 - R(t)]^{N-i} \tag{2.2}$$

If the system is not designed carefully, the correlated failure factor can dominate the overall failure probability.

The best-known example of NMR type of systems is the triplex, which consists of three identical modules whose outputs are voted on. This is a 2-of-3 system so long as a majority of the modules produce correct results, the system will be functional. In TMR, all the three identical modules perform the same operation, and a voter accepts outputs from all three modules, producing a majority vote at its output as shown in Figure 2.2. In such a structure, $M = 2$ and $N = 3$ and a voter selects the majority vote. If a single voter is used, that voter becomes a

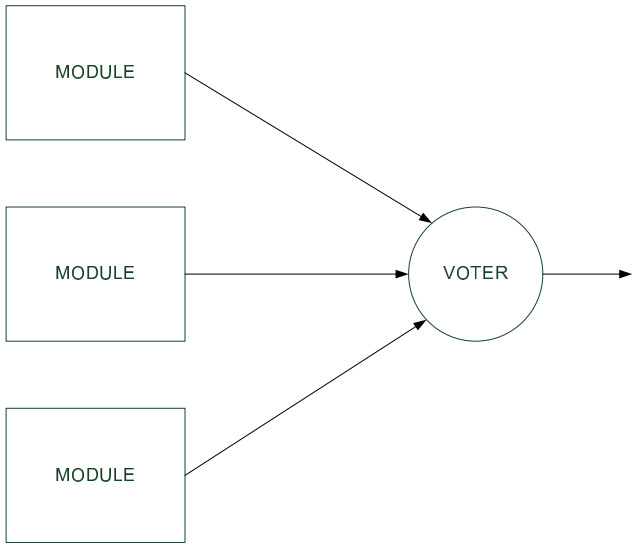

Figure 2.2: A Triple Modular Redundant (TMR) structure.

critical point of failure and the reliability of the TMR structure is:

$$R_{NMR}(t) = R_{voter}(t) \sum_{i=2}^{3} \left(\begin{array}{c} 3 \\ \\ i \end{array} \right) R^i(t)[1 - R(t)]^{3-i}$$

$$R_{NMR}(t) = R_{voter}(t)[3R^2(t) - 2R^3(t)] \tag{2.3}$$

where $R_{voter}(t)$ is the reliability of the voter. As shown in the above equation, the reliability of TMR design is limited by that of the final arbitration unit (i.e., voter), making the approach difficult in the context of highly integrated nanosystems [8]. In TMR, however, the reliability of a module imposes a demanding requirement on a module's size i.e., the modules involved in TMR should be modest in size in relation to the error rate of an individual component in the circuit, in other words, a module with many components will present a serious limit on

the upper bound of the device error rate that TMR can tolerate [11].

A TMR circuit can be further triplicated. The obtained circuit thus has nine copies of the original module and requires two layers of majority gates to collect information at outputs. This process can be repeated if necessary, resulting in a technique called cascaded triple modular redundancy (CTMR). Spagocci and Fountain [12] have shown that using CTMR in a nanochip with many (for example, 10^{11} or 10^{12}) nanoscale devices would require an extremely low device error rate. However, the method might be effective in modest or small circuit modules. Another disadvantage of the CTMR scheme is that it introduces an exponential growth in redundancy as the cascaded layers increase. In [13], it is shown that recursive voting leads to a double exponential decrease in a circuit's failure probability. However, a single error in the last majority gate can cause an incorrect result, hampering the technique's effectiveness.

2.3.3 Intervowen Redundant Logic & Quadded Logic

Pierce [10] generalized von Neumann's and his contemporaries' ideas on fault-tolerant logic to a theory termed interwoven redundant logic. This theory interprets the faults it considers as $0 \rightarrow 1$ and $1 \rightarrow 0$ faults. The error correction mechanism in interwoven redundant logic depends on asymmetries in the effects of these two types of binary errors. The effect of a fault depends on the value of the erroneous input and the type of gate. Consider a NAND gate, for instance. If the binary value of one of its inputs is 0 while it should be 1, possibly because

of a faulty gate or interconnection, the NAND's output value will remain a 1 regardless of the values of other inputs. If an input value is 1 while it should be 0, the output will not be stuck but will depend on other inputs. Thus, there are two types of faults for a NAND gate. One is critical in the sense that its occurrence on one of the inputs leads to a stuck output, the other is subcritical in the sense that its occurrence alone does not cause an output error. Hence, alternating layers of NAND (or NOR) gates can correct errors by switching them from critical to subcritical.

Quadded logic [11, 14, 15] is an ad hoc configuration of the interwoven redundant logic. It requires four times as many circuits, interconnected in a systematic way, and it corrects errors and performs the desired computation at the same time. Researchers have studied quadded logic for use with AND, OR, and NOT logic, and for use with NOR logic. Consider the schematic of a complementary half adder (computing the complements of carry and sum, denoted as cc and cs) shown in Figure 2.3 and its quadded form in Figure 2.4, both implemented with NAND gates (including inverters, considering them a special form of NAND gate).

Interconnection in Quadded Logic

The quadded implementation in Figure 2.4 replaces each NAND gate from Figure 2.3 with a group of four NAND gates, each of which has twice as many inputs as the one it replaces. The four outputs of each group are divided into two sets of two outputs, each providing inputs to two gates in a succeeding stage. The

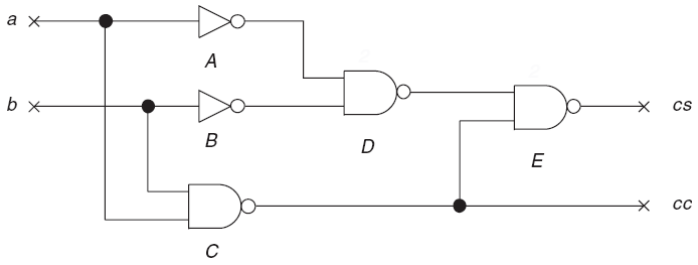

Figure 2.3: Nonredundant complementary half adder implemented with NAND logic.

interconnections in a quadded circuit are hence eight times as many as those used in the nonredundant form. The interconnect patterns in a quadded network are important to the network's capability of error correction, yet the rules are simple. The outputs of four gates, numbered 1 to 4 in Figure 2.4, are divided into two sets. Each set forms a pair of inputs and each pair feeds the two gates with the same numbers as the set in succeeding stages. If the four outputs are divided into two sets of (1,3) and (2,4), for instance, set (1,3) will provide inputs to gates 1 and 3 in the next stage and set (2,4) will provide inputs to gates 2 and 4. There are three possible ways to break four inputs into two sets to form an interconnect pattern: (1,2) and (3,4); (1,3) and (2,4); and (1,4) and (2,3). The rule to arrange these patterns is that the interconnect pattern at the outputs of a stage must differ from the interconnect patterns of any of its input variables.

Error Correction in Quadded Logic

In the pattern of interconnection in quadded logic, any single error introduced in the network is correctable by the network itself, provided that the network is

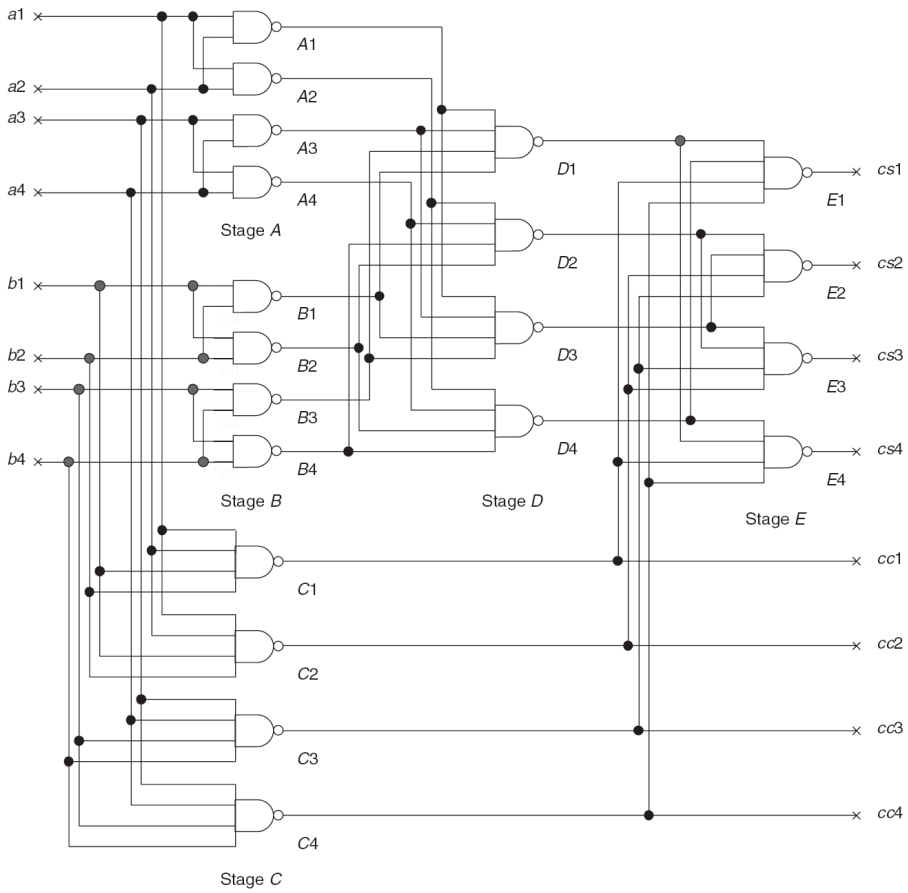

Figure 2.4: Quadded implementation of the complementary half adder.

large enough. In Figure 2.4, assume that output B1 in stage B is wrongly in the 0 state when it should be in the 1 state (a critical $1 \rightarrow 0$ error for the NAND gate). Because of this error, outputs D1 and D3 of stage D must be 1, this can be erroneous, but it would be a subcritical $0 \rightarrow 1$ error. Since outputs D2 and D4 of stage D are not in error (thus in the correct 0 state), the subcritical errors at outputs D1 and D3 are masked at stage E, producing the expected (correct) 1 state at all the outputs of stage E. It is observed that a subcritical $0 \rightarrow 1$ error is even more promptly corrected in the NAND network. In general, a single critical error in a quadded circuit will be eliminated after passing through two stages, and a single subcritical error will be corrected in the next stage after its occurrence. The error correction property of a quadded NAND network is in fact a result of its logical characteristics. Consider the outputs of stage B in Figure 2.4: B1, B2, B3, and B4. After passing through two NAND stages, the outputs of stage B can be represented at stage E by the following Boolean function: $B1B3 + B2B4$. All Bs in this function should be the same in the absence of errors, but any single error in the Bs will not affect the function's correct value. In a quadded circuit, a single error is correctable in at most two logic layers. Errors occurring on the circuit's edge, however, might not be eliminated at outputs (more specifically, a critical error within the last two layers or a subcritical error in the last layer is not correctable at outputs). Therefore, the gates on the edge are critical in the sense that the failure of any critical gate will cause a high probability of failure for the whole circuit. Because a single error is corrected within a rather short

logical path, many multiple errors do not interact. Hence, multiple errors are also correctable in many cases. This is a particular merit of quadded logic.

2.3.4 N-tuple Intervowen Redundancy (NIR) & Triple Intervowen Redundancy (TIR)

Jie Han and Pieter Jonker present a new design of interwoven redundant logic, called random interwoven redundancy, which can serve as the basis for building any realistic circuit. The investigation of the fault tolerance of random interwoven redundant circuits is done through a simulation-based experimental approach [34].

Triplicated interwoven redundancy (TIR) is the simplest form of random intervowen redundancy. Figure 2.5 shows the schematic of a TIR implementation for the complementary half adder in Figure 2.3. The TIR circuit triplicates each NAND gate in the nonredundant circuit, as well as all the interconnections. A TIR circuit thus has three times as many gates and interconnections as the corresponding nonredundant circuit. The interconnections in a TIR circuit are, in principle, arranged randomly. For example, in a TIR circuit comprising two-input NAND gates, for instance, there are six possible pair connections: (1,1), (2,2), (3,3), (1,1), (2,3), (3,2), (1,2), (2,3), (3,1), (1,2), (2,1), (3,3), (1,3), (2,1), (3,2), and (1,3), (2,2), (3,1). The notation (i, j) means that the output of gate i in a triplet of gates, pairs with the output of gate j in another triplet to form the inputs of a gate in the next stage. The total interconnect pattern becomes 36 (or 6×6) if the gate orders of a triplication in the next stage are distinguished.

One method of arranging the interconnections is to randomly adopt one of the 36 connection patterns for all connecting pairs in adjacent layers. As shown in Figure 2.5, the interconnect patterns used in the three layers from inputs to outputs of the circuit are (1,1), (2,2), (3,3), (1,2), (2,3), (3,1), and (1,3), (2,1), (3,2), although the circuit can use any other interconnect pattern. Notice that, if the pattern (1,1), (2,2), (3,3) is used in all layers for all interconnections, the circuit in Figure 2.5 will perform a computation as three independent modules, it will actually work as a TMR circuit, as depicted in Figure 2.6. TIR is hence a generalization of TMR to allow for random interconnections. The randomness in the TIR interconnections is particularly interesting in the physical implementation of molecular electronics, for which stochastic chemical assembly will most likely be the manufacturing method.

The principle of TIR is applicable to arbitrary logic circuits. A general procedure for constructing a TIR circuit is as follows [34]:

1. Start with a nonredundant form of the circuit.

2. Triplicate each gate.

3. Following the interconnect pattern of the nonredundant circuit, randomly select a gate from a triplet to use as an input for a gate that has no other inputs from the same triplet.

4. Repeat Step 3 until all the gates are connected in the TIR circuit

As in TMR, a TIR circuit requires a decision element (a voter) as a restor-

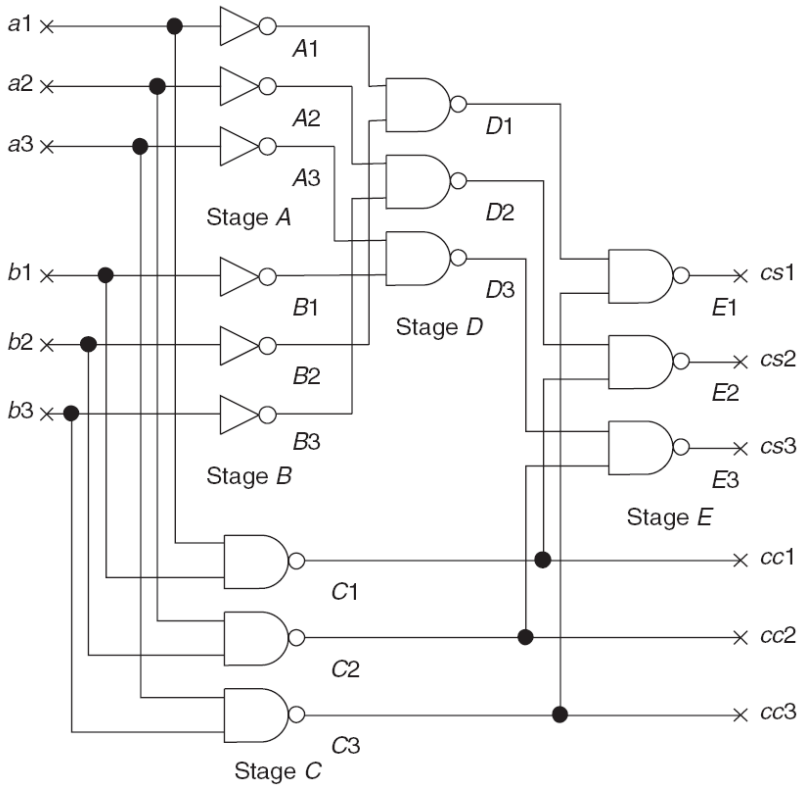

Figure 2.5: TIR implementation of the complementary half adder.

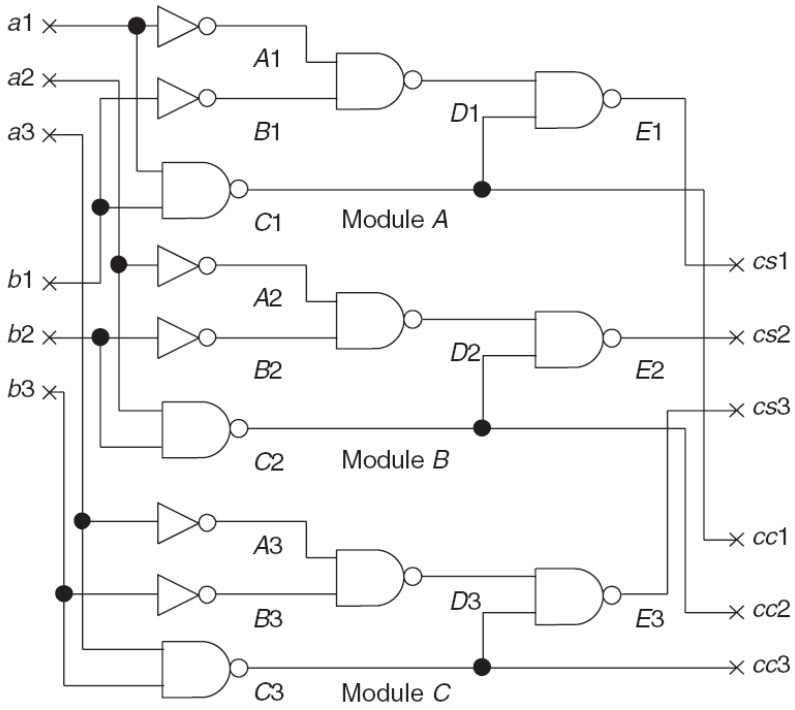

Figure 2.6: TMR configuration of the TIR complementary half adder.

ing device. TIR can be extended to higher orders, namely, N-tuple interwoven redundancy (NIR), similar to the extension of TMR to NMR. Hence, NIR is a generalization of NMR, but with random interconnections [34].

2.4 Defect Avoidance Design Techniques

Unlike defect-tolerant techniques which are designed to work properly despite the presence of defects, defect avoidance techniques are based on a different principle. They are based on the identification of defective modules and replacing them by other redundant modules through reconfiguration. Several researchers have proposed defect-avoidance techniques [8, 16, 17]. Two of them are discussed in this section.

2.4.1 Mishra & Goldstein's Technique

Mishra & Goldstein [17] have proposed a defect avoidance methodology particularly suited for Chemically Assembled Electronic Nanotechnology (CAEN) centered around reconfigurable devices. Their proposed technique first constructs a map of the defects depending on the outcome of testing and defect detection. Then when the device is configured to implement a particular circuit, the defects are avoided by using only the good components of the device. Their testing technique is primarily concerned with finding the defects. Their testing algorithm for finding defects in reconfigurable nanoblocks consists of two phases:

- the probability assignment phase

- the defect location phase

The probability assignment phase assigns each component a probability of being defective and discards the components which have a high probability. This results in a large fraction of defective components being identified and eliminated from further testing. The remaining components are likely to have a small enough defect rate that they can be tested in the defect location phase using a simple method to identify all the defect-free components.

In each phase, the fabric components are configured into test circuits in a particular orientation, or tiling, since each circuit uses only a small number of components, many such circuits can be configured in parallel, or tiled, across the fabric. After finding the defects, defect map is constructed for each nanoblock and then a feasible configuration is synthesized offline realizing the application for each nanofabric instance. Finally, each instance is configured accordingly.

The drawback of this approach is that it is not considered scalable for large nanosystems because it requires mapping, synthesis and configuration at a very fine level of granularity. Moreover, the two-phase group testing strategy used by Mishra & Goldstein for defect mapping requires unlimited connectivity among nanoblocks [8]. The main idea of Mishra and Goldstein's technique for using reconfiguration for achieving defect avoidance has been used by other researches in their work as starting point. An example of such a technique using reconfiguration for defect avoidance is Chen He and Margarida F. Jacome's design paradigm [8] which is discussed in next section.

2.4.2 Chen He and M. F. Jacome's Technique

Chen He et al. have proposed a hierarchy of design abstractions aimed at ensuring scalability not only during a nanosystem's synthesis but also in the defect mapping and configuration phases [8]. The innovative aspect of their approach is that it addresses scalability jointly across these phases. Their approach is based on two key ideas:

- The first idea is to structure designs as hierarchies of carefully dimensioned reconfigurable fabric regions, while decomposing and assigning small functional flows to each such region. By restricting the functionality preassigned to a specific nanofabric region, the scope and complexity of the defect mapping and configuration tasks are limited. Because it requires working with only a set of basic flows assigned to structured fabric regions of limited complexity, it becomes possible to compute configuration alternatives. However, to achieve high yields, it must be ensured that each region has sufficient degrees of freedom for configuration or capacity so that there is high probability that associated flows can be instantiated [8].

- The second idea underlying their approach is to devise efficient defect mapping and configuration methods for such regions. There is no need to map all defects in a region, instead, it is sufficient to establish the existence of a feasible configuration for the region's associated flows.

Figure 2.7 summarizes the three-level hierarchy used to design the nanofabric.

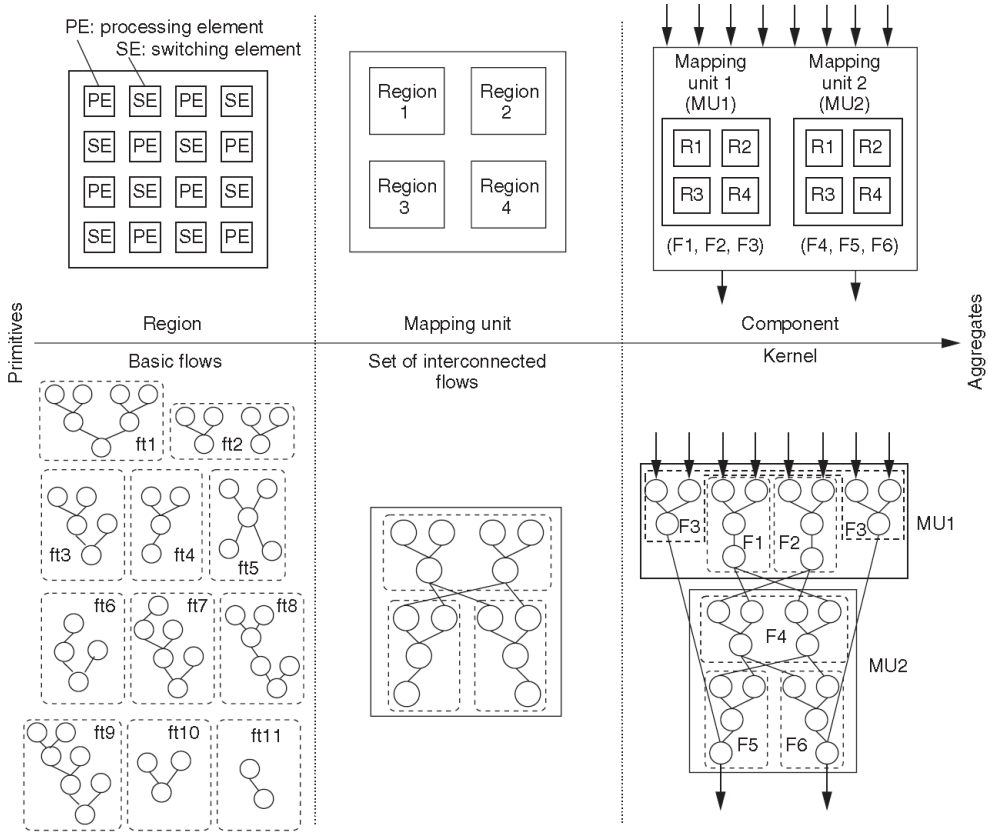

Figure 2.7: Three-level design hierarchy showing abstractions in the form of region, mapping unit and component on the upper level and behavioral abstractions on the lower level.

The proposed approach and design abstraction enables a substantial part of the defect mapping and configuration tasks for structured nanofabrics to be performed within the nanofabric itself. Specifically it is possible to independently map defects in each fabric region and then configure the individual basic flows mapped into such regions around the defects. The approach uses a set of test tiles implementing a TMR configuration to systematically identify a region's defective PEs or connections. The experimental results show that the proposed technique using scalable defect mapping and configuration performs better than TMR-based design methodology.

2.5 Transient & Soft Error Mitigation Techniques

Transient and soft errors due to Single Event Upsets (SEUs) (or Single Event Transients (SETs)) which are caused mainly due to cosmic-ray neutrons or alpha particles, are main reasons behind lower field-level product reliability [33].

In the following subsections, first a brief introduction of SEUs and SETs is presented. After that, SEU mitigation techniques are briefly surveyed. This is followed by a brief survey of SEU mitigation techniques for FPGAs. Then, a brief description of an empirical model for estimation of soft error rate and soft-spot analysis method for the identification of circuit areas most prone to soft-errors is presented.

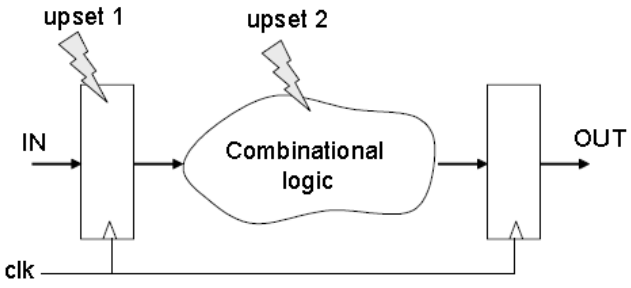

Figure 2.8: Upsets hitting combinational and sequential logic.

2.5.1 Single Event Upsets and Single Event Transients

A single particle can hit either the combinational logic or the sequential logic in the silicon [54]. Figure 2.8 illustrates a typical circuit topology found in nearly all sequential circuits. The data from the first latch is typically released to the combinational logic on a falling or rising clock edge, at which time logic operations are performed. The output of the combinational logic reaches the second latch sometime before the next falling or rising clock edge. At this clock edge, whatever data happens to be present at its input (and meeting the setup and hold times) is stored within the latch.

When a charged particle strikes one of the sensitive nodes of a memory cell, such as a drain in an OFF state transistor, it generates a transient current pulse that can turn on the gate of the opposite transistor. The effect can produce an inversion in the stored value, in other words, a bit flip in the memory cell. Memory cells have two stable states, one that represents a stored 0 and one that represents a stored 1. In each state, two transistors are turned ON and two are turned OFF

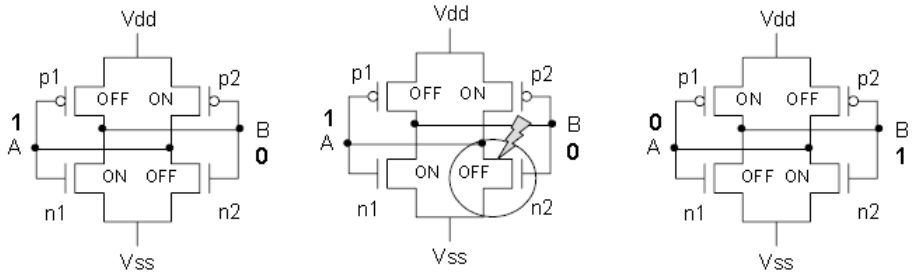

Figure 2.9: Single Event Upset (SEU) effect in a SRAM Memory cell.

(SEU target drains). A bit-flip in the memory element occurs when an energetic particle causes the state of the transistors in the circuit to reverse, as illustrated in Figure 2.9. This effect is called Single Event Upset (SEU), and it is one of the major concerns in digital circuits.

When a charged particle hits the combinational logic block, it also generates a transient current pulse. This phenomenon is called single event transient (SET) effect [56]. If the logic is fast enough to propagate the induced transient pulse, then the SET will eventually appear at the input of the second latch in Figure 2.8, where it may be interpreted as a valid signal. Whether or not the SET gets stored as real data depends on the temporal relationship between its arrival time and the falling or rising edge of the clock.

Figure 2.10 exemplifies the signal paths in a combinational logic. In [57, 58], the probability of a SET becoming a SEU is discussed. The analysis of SET is very complex in large circuits composed of many paths. Techniques such as timing analysis could be applied to analyze the probability of a SET in the combinational

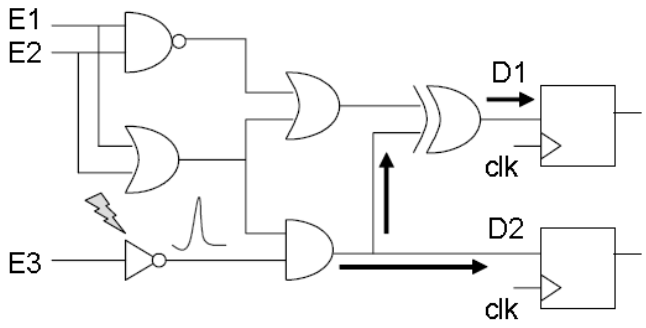

Figure 2.10: Single Event Transient (SET) Effect in Combinational Logic based on [53].

logic being stored by a memory cell or resulting in an error in the design operation, as presented in [59]. Additional invalid transient pulses can occur at the combinational logic outputs as a result of SETs generated within global signal lines that control the function of the logic. An example of this would be SETs generated in the instruction lines to an ALU. In [60], the widths of some induced transient pulses are measured to obtain more precise models for fault-tolerant analysis.

It is worth noting that according to the logic fan-out, a single SET can produce multiple transient current pulses at the output. Consequently, SETs in the logic can also provoke multiple bit upsets (MBU) in the registers once the SETs are captured by the flip-flops.

Performing a more detailed analysis, the sensitive regions of an integrated circuit are the surroundings of the reverse-biased drain junctions of a transistor biased in the OFF state [61], as for instance the drain of the OFF p-channel transistor as shown in Figure 2.11. As current flows through the struck transistor,

the transistor in the ON state (n-channel transistor in Figure 2.11) conducts a current that attempts to balance the current induced by the particle strike. Actually, there are three current components at the struck node. The current induced by the particle strike I_P, the current I_{ON} that flows through the transistor in the on-state, and the current I_C that charges the parasitic capacitances at the node. The current $I_C(t)$ is the current that will charge the node equivalent capacitance and cause the bit flip, and is given by:

$$I_C(t) = I_P(t) - I_{ON}(t) \tag{2.4}$$

If the current induced by the particle strike is high enough, the ON transistor can not balance the current and a voltage change at the node will occur. This voltage change can be propagated to the opposite inverter and lead to the flipping of the bit stored in the memory cell. If the voltage transient is fed back through the opposite inverter, a SEU occurs. If the voltage on the struck node is recovered by the current feed through the ON transistor, no SEU will be observed.

The critical charge has been reduced in new process technologies because of scaling. For constant field scaling, for example, as all physical device dimensions such as gate length L, gate width W, and gate oxide thickness T_{OX}, are reduced, the supply voltage V_{DD} and the threshold voltage V_{TH} are also reduced proportionately. This fact results in proportionately lower drain current (I_{ON}), proportionately lower load capacitance (C), and proportionately lower circuit gate delay ($C * VDD/I_{ON}$). This means that less charge or current is required to store

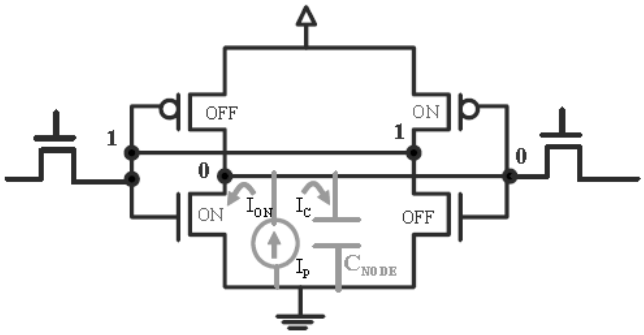

Figure 2.11: Single Event Upset (SEU) effect in a SRAM Memory cell.

information. Consequently, devices are becoming more vulnerable to radiation and this means that particles with small charge, which were once negligible, are now much more likely to produce upset [51].

Traditionally, SEUs have been a threat mostly in aerospace applications, but as discussed in previous paragraph, more recently ICs are becoming more and more sensitive to upsets at ground level due to continual evolution of fabrication technology. Device shrinkages, power supply reduction and increasing operating speeds significantly reduce noise margins and reliability because of the interal noise sources which very deep-submicron devices face . With device dimension shrinking to nanometer scales, this trend is approaching a point at which it will be infeasible to produce ICs that are free from these effects. Therefore, tolerance of soft and transient errors is no longer a matter exclusively for aerospace applications, it is important for the designers of next-generation terrestrial applications as well [47].

2.5.2 Single Event Upset Mitigation Techniques

In [51], Kastensmidt et al. have detailed several SEU mitigation techniques. The authors describe that the first SEU mitigation solution that has been used for many years in spacecraft and other aerospace applications was shielding, which reduces the particle flux to very low levels, but it does not completely eliminate it. This solution was sufficient to avoid errors caused by radiation effects for many years in the past. However, due to the continual evolution of the fabrication technology process, electronic circuits are becoming more and more sensitive to radiation particles, and the charged particles that once were negligible are now able to cause errors in the electronic design. Consequently, extra techniques must be applied to avoid radiation effects.

Several SEU mitigation techniques have been proposed in the last few years in order to avoid faults in digital circuits, including those implemented in programmable logic [51]. They can be classified as:

- Fabrication process-based techniques such as:
  - Epitaxial CMOS processes
  - Advanced process such as silicon-on-insulator (SOI)
- Design-based Techniques
  - Detection Techniques
    - EDC (Error Detection Coding)
    - Self-checker techniques

  - Mitigation techniques
    - Hardware redundancy like Triple Modular Redundancy (TMR), Multiple redundancy with voting
    - Time redundancy
    - EDAC (Error detection and correction coding)
    - Hardened memory cell level
- Recovery Techniques (applied to FPGAs only)
  - Reconfiguration
  - Partial configuration
  - Rerouting design

In the next few subsections, design based mitigation techniques based on time and hardware redundancy are briefly discussed. Mitigation Techniques for FPGAs based on TMR as well as reconfiguration will be briefly surveyed in section 2.7.

Full Time and Hardware Redundancy

The use of full time redundancy in the combinational logic permits voting the correct output value in the presence of a SET. The name full redundancy comes from the complete N-modular redundancy, when N is equal to three, it is triple modular redundancy. In this case, the output of the combinational logic is latched at three different moments, where the clock edge of the second latch is shifted by the time delay d and the clock of the third latch is shifted by the time delay $2d$.

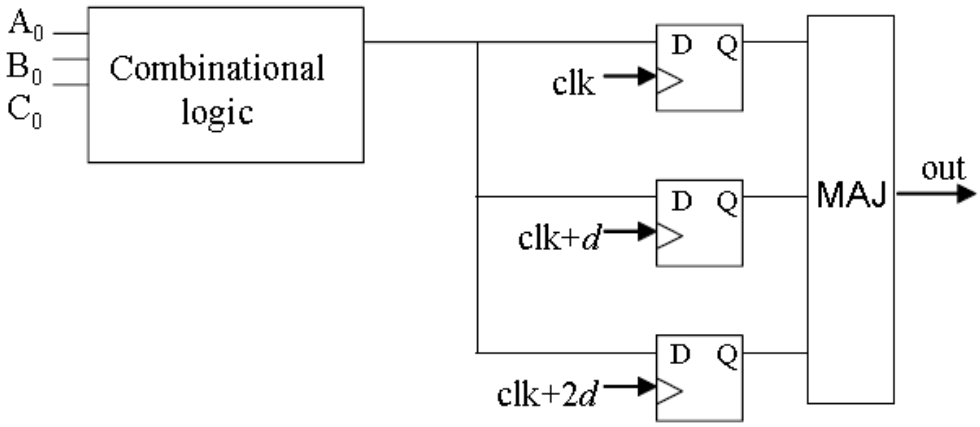

Figure 2.12: Full Time Redundancy.

A voter chooses the correct value. The full time redundancy scheme is illustrated in Figure 2.12. The area overhead comes from the extra sample latches and the performance penalty is given by $clk + 2.d + t_p$, where d depends on the duration of the transient current pulse and t_p is the delay from the majority voter.

In the case of the full hardware redundancy, for instance in the well-known Triple Modular Redundancy (TMR) approach, the logic is triplicated and voters are placed at the output to identify the correct value. The first possibility that was largely used in space applications is the triplication of the entire device, Figure 2.13. This approach uses a voter as a fourth component in the board. It needs extra connections and it presents area overhead. If an error occurs in one of the three devices, the voter will choose the correct value. It protects both combinational and sequential logic against upsets. However, if an upset occurs in the voter, the TMR scheme is ineffective and a wrong value will be present in the

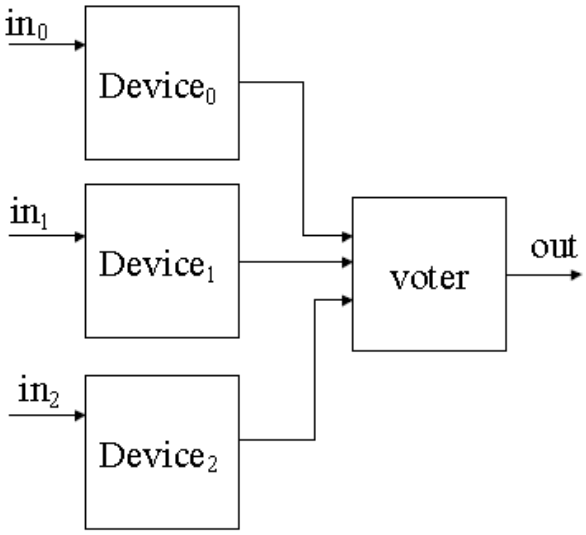

Figure 2.13: TMR implemented in the entire device.

output. Another problem of this approach is the accumulation of upsets, hence an extra mechanism is necessary to correct the upset in each device before the next SEU happens.

A more efficient implementation of the TMR is applied focussing on the sensitive logic, for example the memory cells to protect against SEU, Figure 2.14. However, this solution does not avoid the accumulation of upsets in the sequential logic and the voter is vulnerable to upsets.

In order to restore the corrected value, a solution using three voters with a feedback was proposed [52], Figure 2.15. The upsets in the latches are corrected by extra logic in order to avoid accumulation. The load frequency (refreshing) can be set by the multiplexor control signal.

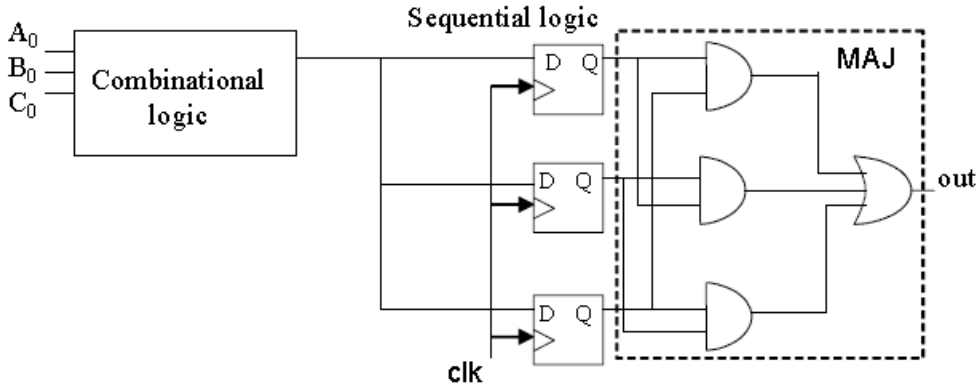

Figure 2.14: TMR memory cell with single voter.

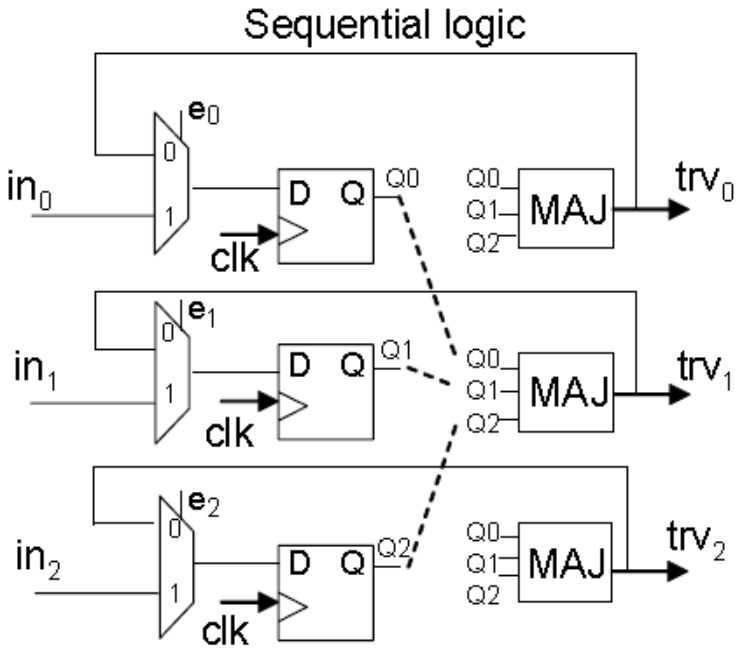

Figure 2.15: TMR memory cell with three voters and refreshing.

The combinational logic must also be protected to avoid SET. There are many possibilities. One is to use time redundancy in the logic as shown in Figure 2.16. Another possibility is to triplicate the combinational logic as well, as shown in Figure 2.17.

Although the last proposed implementation of the TMR (Figure 2.17) presents a larger area overhead compared to time redundancy, since it triplicates all the combinational and sequential logic, it protects the logic against SET and SEU and avoids accumulation of upsets. In addition, it does not have major performance penalties, just the voter propagation time, and it does not need different clock phases.

Another method to mitigate SET in combinational logic is based on duplication and a Code Word State Preserving (CWSP) [53], as illustrated in Figure 2.18. This method does not need voters or comparators. The duplication can be replaced by time redundancy as well, which reduces the area overhead significantly, Figure 2.19. The main contribution of this method is the CWSP stage, which replaces the last gates of the circuit by a particular gate topology, which is able to pass the correct value in the combinational logic in the presence of a SET, Figure 2.20. Additional techniques to cope with SET are presented in [54].

Some application systems concern about multiple upsets. However the problem of multiple upsets must be carefully analyzed. Solutions are not trivial. For N-Modular redundancy, where N is usually an odd integer, solutions with N larger than 3 does not always present gains in reliability compared to the TMR because

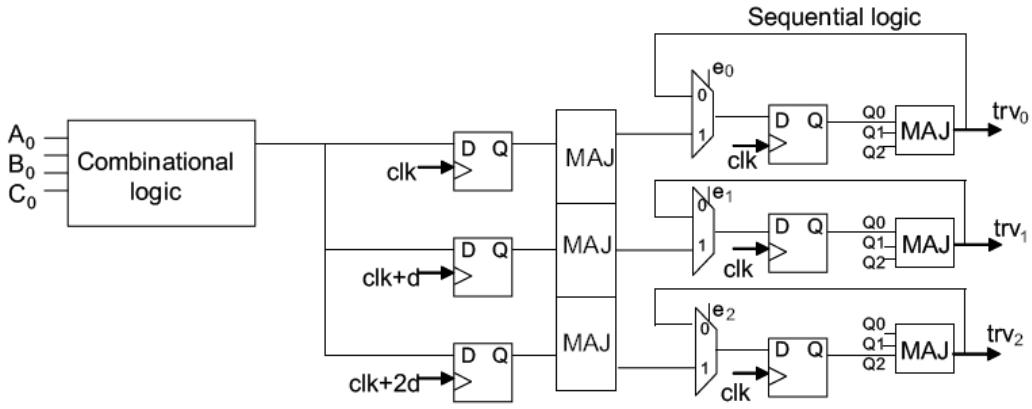

Figure 2.16: Full time redundancy scheme for combinational logic combined with full hardware redundancy in the sequential logic.

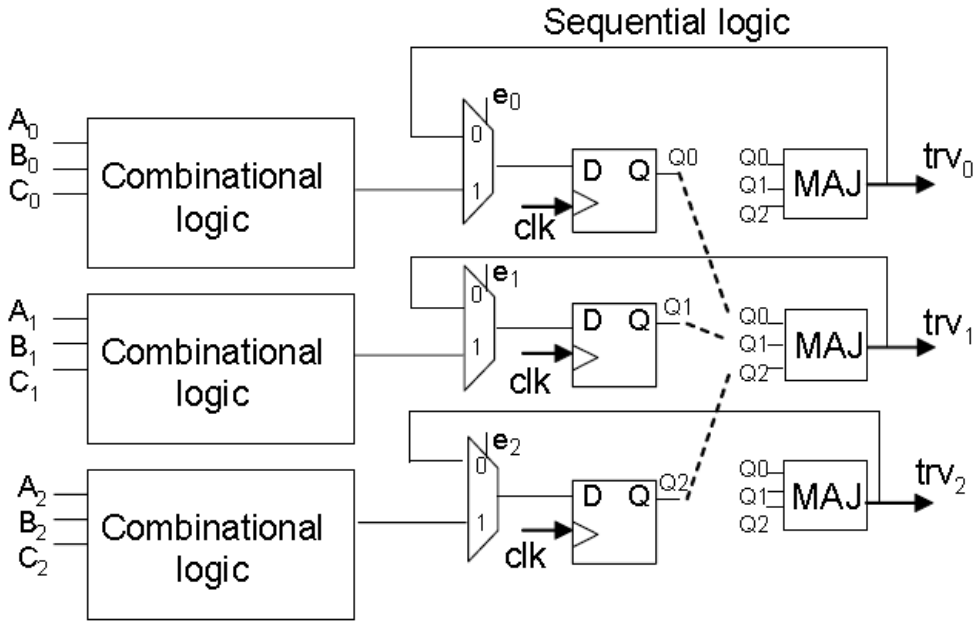

Figure 2.17: Full hardware redundancy scheme for combinational and sequential logic.

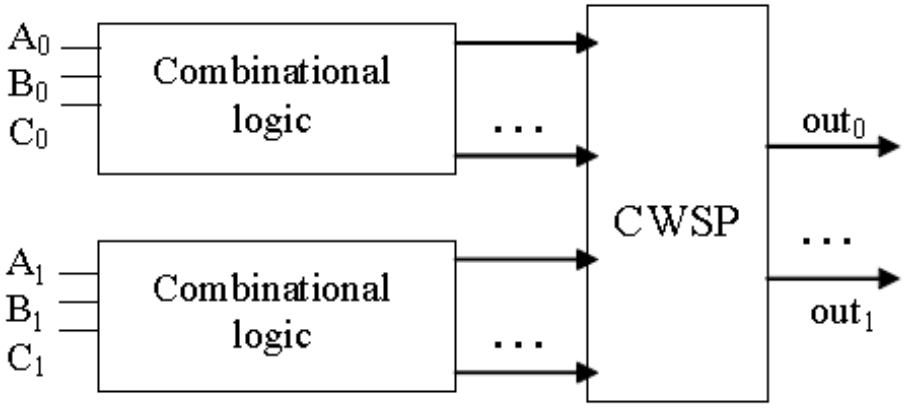

Figure 2.18: Duplication to mitigate SET in combinational logic.

the result depends on the failure rate: λ [55].

For details of other SEU Mitigation techniques reported in literature, refer to the book by Kastensmidt [51].

2.5.3 Single Event Upset Mitigation Techniques for FPGAs

SEU mitigation techniques and other fault-tolerant techniques have been extensively investigated in the context of FPGAs [39, 40, 41, 42, 43, 19, 44, 46, 47, 48]. The reason for this interest is that FPGAs are preferred by designers of aerospace systems because of their ease and flexibility of design and use. Many of these mitigation techniques are variants of Triple Modular Redundancy (TMR) (which is an example of static hardware redundancy) or reconfiguration based schemes.

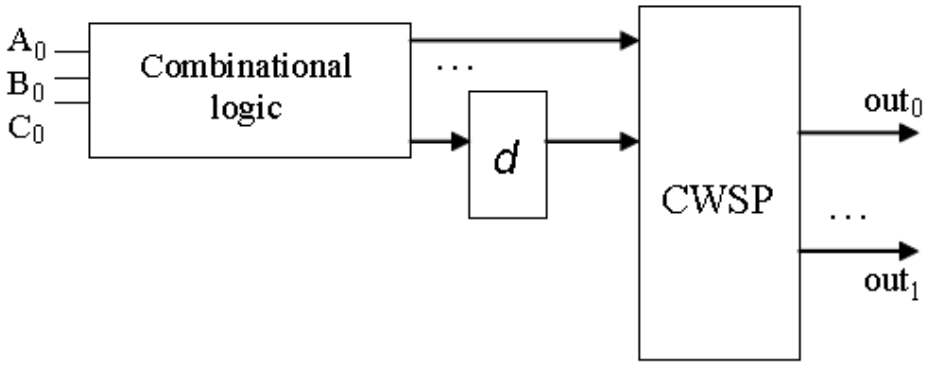

Figure 2.19: Time redundancy to mitigate SET in combinational logic.

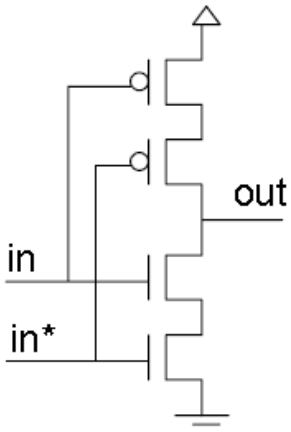

Figure 2.20: Example of INVERTER logic with the code word state preserving (CWSP) in the duplication and time redundancy to mitigate SET in combinational logic.

Categorization of SEU Mitigation Techniques for FPGAs

Kastensmidt et al. have provided the following categorization of fault-tolerant FPGA design techniques for mitigating SEUs [51].

- *SEU mitigation solutions at the architectural level*: These techniques require changing the architecture of the FPGA by replacing the traditional FPGA blocks by hardened and fault-tolerant blocks like hardened memory cells, fault-tolerant CLBs and fault-tolerant interconnection blocks.

- *SEU mitigation techniques at the high level description*: These techniques involve developing a fault-tolerant design at the high level using TMR or similar technique before targeting it into FPGAs.

- *Recovery using Scrubbing*: These techniques involve periodic refreshing of the configuration memory of the FPGA so that transient errors due to SEUs can be cleaned.

Survey of SEU Mitigation Techniques for FPGAs

Asadi et al. presented an evaluation for different single-event-upset (SEU) fault tolerance schemes implemented on FPGAs [39].

The bottleneck in triple modular redundancy (TMR) implementations is the voter. The reliability of the system is always limited by that of the voter. Samudrala et al. proposed implementing TMR on FPGAs. They suggest using tri-state buffers (available on Xilinx Virtex FPGAs) to build SEU-tolerant voters [40]. By doing so, they deal with the bottleneck in the reliability of the TMR. They use a

program to evaluate the nodes with a high probability of errors due to SEUs, and selectively implement TMR at the potential gates.

The work in [41] investigates the ideal positioning of the voters for a TMR system to achieve maximal robustness with minimal overhead. TMR is not the only way to deal with SEUs in FPGAs. Some work in the past has compared the efficiency of TMR with that of error detection (using duplication) followed by recomputing. At low error rates, it is more efficient to use duplication with recomputing.

Tiwari et al. proposed protecting against SEUs using parity bits in the memory blocks of FPGAs [42]. If an error is detected, the memory is re-written. The technique shows a significant power improvement compared to TMR.

Sterpone et al. suggest a place and route algorithm to reduce the susceptibility to SEUs in FPGAs [43].

Duplication and concurrent error detection are used to tolerate transient and permanent faults in FPGAs [47]. The authors use time redundancy as well as duplication with comparison. They also apply recomputation with shifted operands and swapped operands. Their technique requires fewer I/O pads and consumes less power than TMR. However, it takes more time and requires more flip flops.

For details of other SEU mitigation techniques for FPGAs reported in literature, refer to [51].

2.5.4 Empirical Model for Soft Error Rate Estimation

Hazucha et al. have derived an empirical model for estimation of soft error rate (SER) [33]. They fabricated test circuits in a standard 0.6-μm CMOS process. The neutron SER dependence on the critical charge and supply voltage was measured and time constants of the noise current were extracted from the measurements and compared with device simulations in three dimensions. The empirical model was calibrated and verified by independent SER measurements. One limitation of the model is that it is only capable of predicting cosmic-ray neutron SER of a circuit manufactured in the same process as the presented test circuits.

2.5.5 Soft-Spot Analysis

Zhao et al. have proposed a technique called *Soft-Spot Ananlysis* [38]. Their argument is that only few nodes in the design are highly critical and they need to be tolerant to faults. Soft-Spot analysis identifies regions in a circuit that are most susceptible to multiple noise sources and their compound effects so that designers can harden those spots for greater robustness.

For each node N in a given digital circuit, softness S_N is defined as the node's vulnerability to noise, reflected by the node's tendency to allow noise to propagate through it with enough strength and proper timing to eventually cause observable errors. An observable error is one that is latched into a memory element and thus becomes a stable erroneous logic value. Soft-spot analysis determines the magnitude of S_N for all circuit nodes and identifies a collection of soft spots as

the nodes with high softness values.

The authors further argue that not all noise occurring in a digital circuit can eventually cause functional errors. Three well-known masking effects viz. timing masking, electrical masking and logic masking tend to prevent noise from causing observable errors.

Timing Masking

Timing masking means that noise can cause an observable error only if it is captured by a memory element. To be captured, noise must arrive at the memory element's input within sampling window. For a DFF, the sampling window is bounded by setup time t_{su} and hold time t_h around the active clock edge as shown in Figure 2.21. To determine the required time interval for noise at a node to reach a DFF within its sampling window, the authors [38] have defined the effective noise window TW_{eff}^{N} such that only noise existing at node N overlapping with TW_{eff}^{N} can reach at least one DFF during the DFFs sampling window. In other words, if a noise originates or arrives at node N before the start (or after the end) of TW_{eff}^{N}, it will reach all DFFs before the start (or after the end) of their sampling window and will therefore not be captured by any DFF. The TW_{eff}^{N} of a specific path (p) is bounded by start time t_{start}^{Np} and end time t_{end}^{Np}, determined by the worst-case longest delay $(\Delta T^p)_{max}$ and best-case shortest delay $(\Delta T^p)_{min}$ from N to the DFF through p, respectively. If the clock period is T, it is easy to see that $t_{start}^{Np} = T - t_{su} - (\Delta T^p)_{max}$ and $t_{end}^{Np} = T - t_h - (\Delta T^p)_{min}$.

Because there are usually multiple DFFs reachable from node N through many

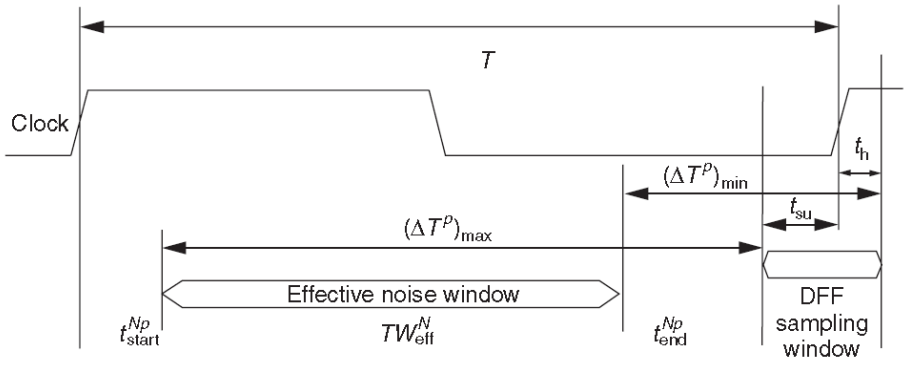

Figure 2.21: The effective noise window.

logic paths, the authors have used the maximum (latest) t^{Np}_{end} and the minimum (earliest) t^{Np}_{start} among all paths to calculate TW^N_{eff}. Let P be the collection of all possible paths through node N:

$$TW^N_{eff} = max_{p \in P} \ [t^{Np}_{end}] - min_{p \in P} \ [t^{Np}_{start}] \tag{2.5}$$

Electrical Masking

Electrical masking means that noise must have enough duration and amplitude to propagate through multiple logic gates. The strength of a single gate's electrical masking effect can be represented by the gate's noise rejection curves (NRCs).

When using the NRC graphs, the noise propagation ratio R^N_e can be defined

as:

$$R_e^N = \left(\frac{A_{sen}}{A_{imm}}\right)_{NRC} \tag{2.6}$$

where A_{sen} is the area of the noise-sensitive region and A_{imm} is the area of the noise-immune region.

Logic Masking

Logic masking refers to the effect that noise ceases to propagate through a gate whose output is solely determined by inputs other than the one carrying the noise. The chances that noises occurring at different nodes will survive multiple levels of logic gates and eventually reach the memory elements depend on the logic structure. Complete determination of the logic masking effect requires exhaustive exploration of the entire input vector space and prohibitively long dynamic simulation time. The authors [38] have developed an efficient logic-path tracing algorithm using the breadth-first search to estimate the propagation probability P_{prop}^N, defined as the ability of a glitch propagating from node N to extend to all reachable DFFs through legitimate logic paths.

Using the above concepts, the softness S_N is evaluated as a function of the timing factor TW_{eff}^N, the electrical factor R_e^N, and the logic factor P_{prop}^N. Therefore, S_N can be expressed as:

$$S_N = W_N \left(TW_{eff}^N \times R_e^N \times P_{prop}^N\right) \tag{2.7}$$

where W_N is an application-specific weighting factor at node N for designers to convey design-related knowledge.

For the identification of soft spots using the above methodology, the authors have developed an automated flow called the automatic soft-spot analyzer (ASSA) shown in Figure 2.22.

The authors have proposed two useful applications of soft-spot analysis: *robustness enhancement* and *robustness insertion.*

- Robustness enhancement increases a circuit's noise immunity by reducing the three masking effects at the identified soft spots through localized and limited design modifications at the gate level. As the analysis identifies soft spots, reducing one or more of the three contributing factors can reduce the spots' softness.

- Robustness insertion judiciously adds circuit-hardening cells at the soft spots to improve the circuits online reliability against transient errors. Spatial and temporal redundancies that protect circuits from noise disturbances have been important techniques for improving circuit online reliability. However, without guidelines, excessive redundancy insertions incur unacceptable design overhead, and the protection might still not be efficient if the most vulnerable circuit elements are underprotected and other circuit elements are overprotected. The goal of robustness insertion is to find an optimal protection scheme to achieve the highest level of robustness improvement

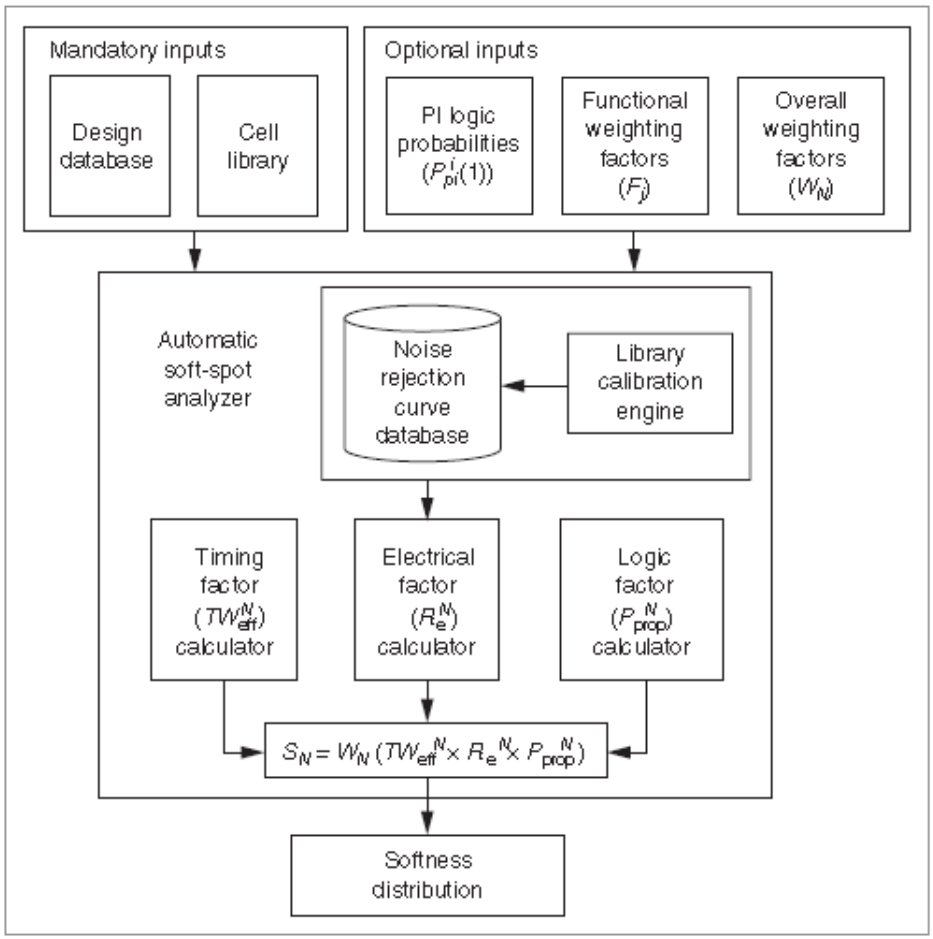

Figure 2.22: Automatic soft-spot analysis.

under given design constraints, using the guidelines of soft-spot analysis and an efficient optimization algorithm.

2.6 Defect-Tolerant Crossbar Design Techniques

Crossbar architectures are one approach to nanoelectronic circuits for memory and logic applications [27, 85]. However, currently feasible manufacturing technologies introduce numerous defects so insisting on defect-free crossbars will give low yields. Instead, defect-tolerant techniques need to be investigated for crossbar based designs in order to ensure correct operation of the circuit even in the presence of defects.

In the following subsections, crossbar architecture and techniques for defect-tolerant crossbar design as reported in literature are briefly discussed.

2.6.1 Crossbar Architecture

The crossbar architecture is a general approach for molecular electronics [27]. A molecular crossbar consists of two parallel planes of molecular wire arrays separated by a thin layer of a chemical species (called the 'interlayer') with particular electrochemical properties as shown in Figure 2.23. Each plane consists of a number of parallel molecular wires (also called 'nanowires'), with each wire in a plane being of the same type. The wires in one plane cross the wires in the other plane

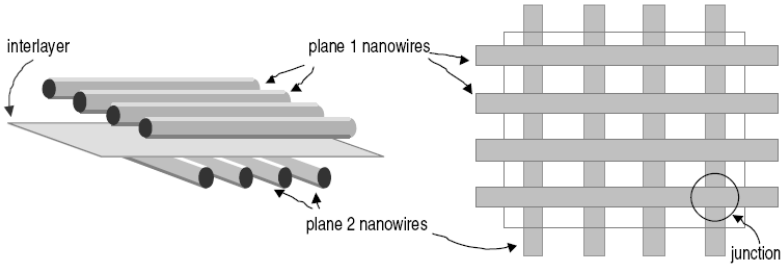

Figure 2.23: Schematic view of a molecular crossbar from two different perspectives.

at a right angle. The region where two perpendicular wires cross is called a junction or crosspoint. Depending on the nature of the interlayer and nanowires, each junction may be configured to implement an electronic device, such as a resistor, diode or field effect transistor, or may be left unconfigured so the two crossing wires forming the junction do not interact electrically.

The crossbar structure is an attractive architecture for molecular electronics since it is relatively simple and inexpensive to fabricate using either chemical self-assembly or nanoimprint lithography [27]. By suitable selection of the type of connections at each crosspoint (e.g., no connection, or a diode in one direction or the other), crossbars can be set to evaluate any logical formula expressed as a combination of AND and OR operations. Figure 2.24 shows one example. To see this, consider the output wire, labeled X. It is connected to ground through a resistor, and via diode junctions to the second and third vertical wires. If both vertical wires are at low voltage (OFF), then the output wire X will also be at low voltage due to its connection to ground. On the other hand, if either of the

connected vertical wires is at high voltage (ON), the diode connection from the high voltage vertical wire(s) will give a high voltage to the output wire (since, by design, the diode resistance in the forward direction is much smaller than the resistor connecting the output wire to ground). If only one of the vertical wires is ON, the high resistance of the diode junction in the reverse direction ensures that the output wire remains at high voltage. Thus this combination of resistors and diode connections makes the output X equal to the logical-OR of the inputs on the two vertical wires. Similarly, the connections from the inputs A, B and C implement logical-ANDs. The crossbar of Figure 2.24 connects each column, through a pullup resistor, to a positive voltage source. With the diode directions shown here, each column implements the logical-AND of its inputs (the horizontal wires). Each output row, connected to ground through a pulldown resistor, implements the logical-OR of the columns connected to it through diode junctions. Although this is not the only way to configure crossbar circuits, it provides a simple functional form in which each output is the logical-OR of a number of terms, each of which is the logical-AND of some inputs. An important limitation of diode/resistor logic is its inability to implement logical inversion (i.e., a NOT gate). However, by presenting the circuit with two wires for each input (i.e., one wire representing the true input value, the other representing its complement), the crossbars can produce internal signals in both the original and complemented forms. Combining these signals using just AND and OR operations then allows evaluating any logical formula. The complemented inputs to the crossbar are

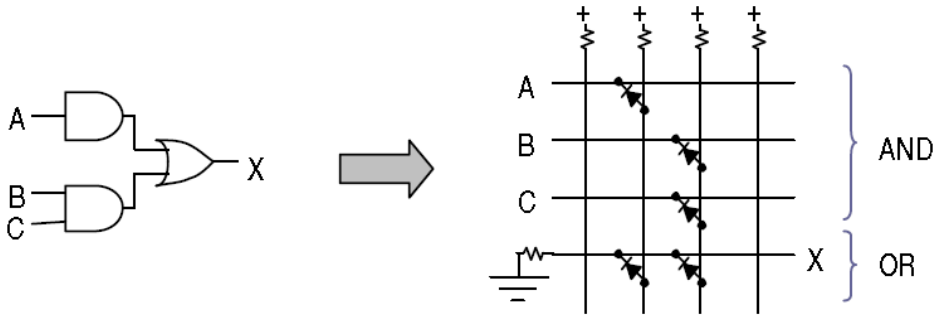

Figure 2.24: Implementing the AND/OR function X = A + BC with a diode crossbar and resistor.

readily produced by the external circuit, fabricated using conventional technology, to which the crossbar is connected for input and output. Thus by doubling the number of wires and presenting all primary inputs in both true and complemented forms, the diode crossbar architecture can implement any logical formula just using combinations of AND and OR operations [27].

There are three crossbar-based architectures reported in the literature:

- NanoFabric
- PLA-based
- CMOS-compatible crossbars

Goldstein and Budiu have proposed a chemically-assembled electronic nanotechnology FPGA-like architecture called NanoFabric [86]. Nano logic arrays, also called Nanoblocks, implement a diode resistor logic (DRL) since crosspoints act as programmable diodes. Since only AND and OR logic can be implemented

by DRL (i.e., no inversion), inputs and their complements are given to nanoblocks, and the output function and its complement are generated. Signal restoration is performed by using a molecular latch at the output of crossbars.

DeHon and Wilson have presented another array-based nanoarchitecture using Programmable Logic Arrays (PLAs) [87]. This architecture allows inversion by using nanowire Field Effect Transistor (FET) devices as buffers. Logic functionality is achieved in the form of two-plane PLAs. Each plane consists of a 2D crossbar, implementing programmable OR array, followed by a restoration and selective inversion array. Therefore, NOR-NOR logic is used in this architecture.

The third architecture is a CMOS-compatible crossbar memory array proposed by Nantero Inc. called NRAM [88]. In this architecture, everything but nanoelectromechanical switches are implemented in CMOS using conventional lithography processes (CMOS-compatible fabrication). The programmable switches are realized by a belt of carbon nanotubes (monolayer fabric of nanotubes). The same technology can also be used to implement programmable logic and interconnection network.

With currently feasible technologies, nanoscale crossbars will contain numerous defective junctions. Thus as a practical matter for implementing logic operations, there is a need to create functioning circuits in spite of defects rather than simply discarding any circuit with even a single defect (which would give unacceptably low yield). For nanoscale crossbar devices, the main type of defect is that introduced during manufacturing (so-called "static defects") rather than during

operation. This is reasonable for plausible technologies, which involve high temperatures during manufacturing, and hence a relative ease of introducing defects, but low temperature during operation, with much less chance of creating new defects. In this situation, an appropriate systems architecture consists of a compiler to arrange for desired circuit behaviors by only using correctly functioning components of a given crossbar circuit, as determined from a testing phase after manufacture [16]. This approach of avoiding known defects gives defect-tolerant crossbar architecture.

2.6.2 Tahoori's Defect-Tolerant Design Techniques for 2D Crossbars

Tahoori et al. have extensively explored the problem of utilizing a defective crossbar for implementing logic [19, 25, 29, 30] . The main idea behind all the approaches is to utilize a partilally defective nanoscale $n \times n$ crossbar as a smaller defect-free $k \times k$ crossbar. The following sections describe briefly the various algorithms proposed by Tahoori et al. for 2D crossbars.

Using Maximum Flow Algorithm

Tahoori et al. have studied the impact of defects on the routability of a 2D crossbar [19]. The 2-D crossbar is represented by a bipartite graph $B = (U, V, E)$. The partition U represents the input nano-wires, while the partition V represents the output nano-wires, E represents the programmable switches in the crossbar,

as illustrated in Figure 2.25. A matching T is a set of edges such that no two edges share the same vertex. If an edge (v_i, v_j) is in the matching, then vertices v_i and v_j are said to be matched. A perfect matching of a graph is a matching such that all vertices are matched. A matching T of size k corresponds to the k signals that can be routed through the crossbar simultaneously. Therefore, the maximum matching of a bipartite graph B represents the so-called routing capacity of the crossbar. In the fault-free case, a perfect matching exists in a fully populated crossbar and thus N input signals can be routed to N outputs. However, in the presence of defects, certain nanowires or switches might become unusable and routability may drop. When the defect density is sufficiently high, the probability of finding a perfect matching will be small. However, the crossbar can still be used as a $k \times k$ ($k < n$) crossbar if a matching of size k can be found with a high probability. In this case, signals can be routed from the k inputs to the k outputs. The proposed technique identifies the probability of finding a $k \times k$ (so that k inputs can be routed to k outputs) crossbar out of a faulty $n \times n$ ($k < n$) crossbar at a specified defect density level. The following metric is used as figure of merit for their proposed technique.

Metric $M_{n,k}^{d}$: The probability of finding a matching of size k in an $n \times n$ crossbar when the defect density is d.

The problem of finding the maximum matching in an arbitrary bipartite graph $B = (U, V, E)$, $|U| = |V| = N, |E| = e$ is done by using the maximum flow algorithm. The authors have obtained experimental results for different sized

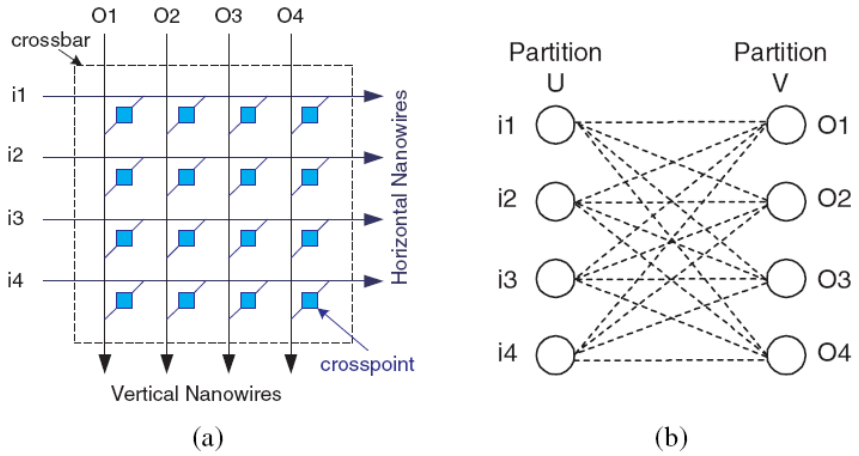

Figure 2.25: (a) 4 x 4 2D nanoscale crossbar (b) Bipartite graph representation.

crossbars and for four types of faults namely switch stuck-open faults, switch stuck-closed faults, nanowire open and nanowire bridging faults. The authors have observed that switch stuck-closed faults in general have a significantly higher impact on the manufacturing yield compared to switch stuck-open faults.

Using Recursive Biclique Algorithm

Tahoori has argued that given the graph model of the defective crossbar as shown in Figure 2.25, the goal of finding a $k \times k$ crossbar within the original crossbar corresponds to finding the maximum biclique in a bipartite graph [25, 29]. The following yield metric is used as figure of merit for the techniques presented in this and next section.

Yield Metric $Y^d_{n,k}$: The probability of finding a biclique (defect-free crossbar) of size $k \times k$ in an $n \times n$ crossbar when the defect density is d.

Finding the maximum biclique in a bipartite graph is an NP-complete problem. In [29], a decision version of this problem is solved which is less complex compared to the original optimization version. Instead of finding the maximum biclique in $G(U, V, E)$, the following decision (Yes/No) problem: "Does $G(U, V, E)$ have a biclique of size $k_1 \times k_2$?" is solved. A recursive algorithm is presented in [29] for solving the aforementioned decision problem.

Using Greedy Biclique Algorithm

Tahoori has proposed a greedy heuristic algorithm for finding the maximum biclique [25, 30]. The approach is to convert the problem of finding a smaller $k \times k$ defect-free crossbar to the problem of finding the maximum independent set in the complement graph. The complement of a graph G is a graph $\bar{G}$ with the same set of vertices such that two vertices of $\bar{G}$ are adjacent if and only if they are not adjacent in G. An independent set S in a graph G is a subset of nodes that are disconnected, i.e. there are no edges between any two nodes in an independent set: $\forall\ u, v \in S, (u, v) \notin E(G)$. The maximum independent set is an independent set with the maximum number of nodes.

Even in the presence of defects (defect density $< 30\%$), the corresponding bipartite graph model of the crossbar is still dense, i.e. $|E| = O(n^2)$. Consequently, the complement graph would be sparse and therefore, a heuristic approach can be effectively used for finding the maximum independence set in the complement graph. This is the main motivation behind converting the maximum biclique problem into the maximum independent set problem in the complement graph

[30].

It is reported in [25] that for given size crossbars, the greedy biclique algorithm is extremely faster (406 times faster) than recursive biclique algorithm for a given value of defect density.

2.6.3 Hogg and Snider's Defect Tolerant Design Technique

Hogg and Snider have examined the implementation of binary adders on defective crossbars [26, 27]. The contribution of their work is follows:

- Two different ways of implementing binary adders have been proposed with one implementation better in defect tolerance than the other.
- An allocation algorithm for mapping a circuit graph (representing the logical formula to be implemented) onto a crossbar graph has been proposed.

Allocation Algorithm

The allocation algorithm uses graphs with annotated edges and nodes to represent both the original circuit to be mapped onto a set of crossbars as well as the crossbars themselves [27] . A wire in the crossbar is represented by a node in the graph, and a junction is represented by an edge between the two nodes representing the wires that define the junction. A perfect crossbar has an edge for every junction. A defective crossbar has edges only for usable junctions. Allocation is accomplished by

1. Creating graphs representing the desired circuit and compound crossbars

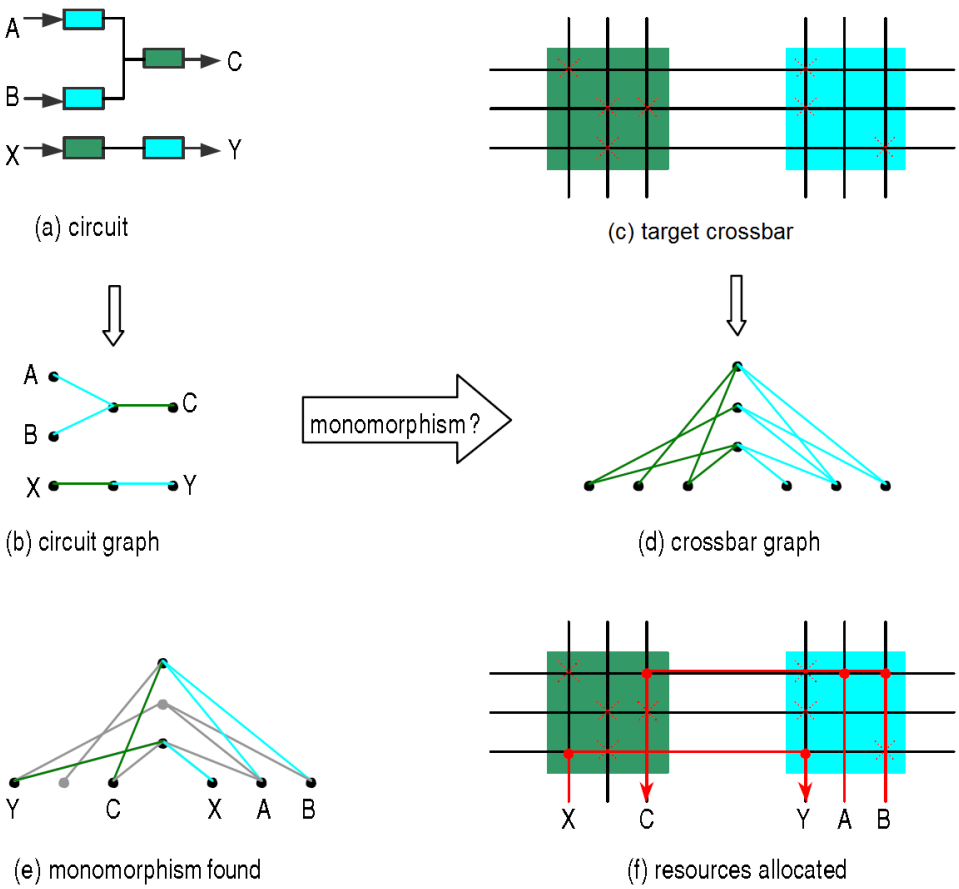

Figure 2.26: Resource Allocation: Searching for a monomorphism between circuit and a crossbar graph.

2. Searching for an embedding or monomorphism between the circuit graph and the compound crossbar graph.

   Figure 2.26 illustrates this in detail.

The steps for allocation algorithm are as follows:

1. For the desired circuit (Figure 2.26(a)) create a circuit graph (Figure 2.26(b)) representing it: wires and junctions in the circuit are represented

by nodes and edges in the circuit graph, respectively.

2. For the desired target compound crossbar (Figure 2.26(c)), create a compound tile graph (Figure 2.26(d)) representing it. As in circuit graph case, wires and junctions in the crossbars are represented by nodes and edges in the compound tile graph, respectively. A defective junction in a crossbar is represented by the absence of its corresponding edge in the crossbar graph.

3. Annotate the edges of the circuit graph and the crossbar graph with annotations representing the functionality of those edges (junctions in the circuit represented by the graph). For example, edges in both graphs representing resistors would all be tagged with identical annotations.

4. Annotate the nodes of the circuit graph and crossbar graph with annotations to constrain matching between the two graphs. This is done to either enforce input/output constraints between the desired circuit and other circuitry that has been or will be mapped to other areas of a large compound tile graph, or enforce directionality constraints on asymmetric junctions, such as diodes, that must have, for example, an input delivered on a horizontal wire and an output driven on a vertical wire; or enforce both.

5. Search for a monomorphism (Figure 2.26(e)) between the annotated circuit graph and the annotated target crossbar graph to do allocation (Figure 2.26(f)), subject to the constraints that node and edge annotations must match. In other words, a node in the circuit graph can only be matched

with a node in the crossbar graph if they both have identical annotations or both have no annotations. Similarly, edges can only be matched if they both have identical (or non-existent) annotations.

6. Use the monomorphism to complete the allocation or mapping of wires and junctions in the desired circuit graph onto wires and junctions of the crossbar. For example, a node, A, in a circuit graph matched to a node, B, in the crossbar graph will be used to allocate the crossbar wire represented by B in the crossbar graph to carry the signal represented by A in the desired circuit. Similarly, an edge, X, in a circuit graph matched to an edge, Y, in the crossbar graph will be used to allocate the junction in the crossbar represented by Y in the crossbar graph for the electrical component represented by X in the desired circuit.

Efficient algorithms for searching for a graph monomorphism are reported in [89, 90, 91].

Using different implementations of the circuits on crossbars, the authors have shown a tradeoff between defect tolerance and circuit area. It is shown that the likelihood that defects are tolerable changes abruptly from one to near zero over a small range of defect rates for a given crossbar size.

2.6.4 DeHon and Naemi's Defect Tolerant Design Technique

DeHon and Naemi have proposed a strategy for tolerating defective crosspoints and a linear-time greedy heuristic algorithm for mapping NanoPLA logic around crosspoint defects [28].

NanoPLAs , like conventional PLAs consist of two programmable NOR planes. Each of the NOR planes consist of two arrays: logic array and buffer/inverter array. The logic array is the programmable part of each NOR plane. Its junctions are the bistable crosspoints. The logic array implements the OR function of its inputs which is why the outputs of this array are called OR-terms. Each of the connected junctions behaves like a diode, and each OR-term is the wired OR logic of its inputs. The output of each OR-term is pulled down weakly. If any of the inputs is high, then it pulls up the OR-term outputs [87].

To implement a specific circuit on a nanoPLA, the logic arrays are programmed. This means that each OR function of a design is mapped to an OR-term nanowire. The logical inputs are the set of inputs to the OR functions. The logical inputs include the primary inputs of the nanoPLA and the signals that are fed back from the other NOR plane. In each OR function, the set of logical inputs that participate in the OR function is called ON-inputs and those that do not participate are called OFF-inputs.

To map each OR function to an OR-term nanowire, the crosspoints of the OR-term nanowire associated with the ON-inputs of the OR function are programmed

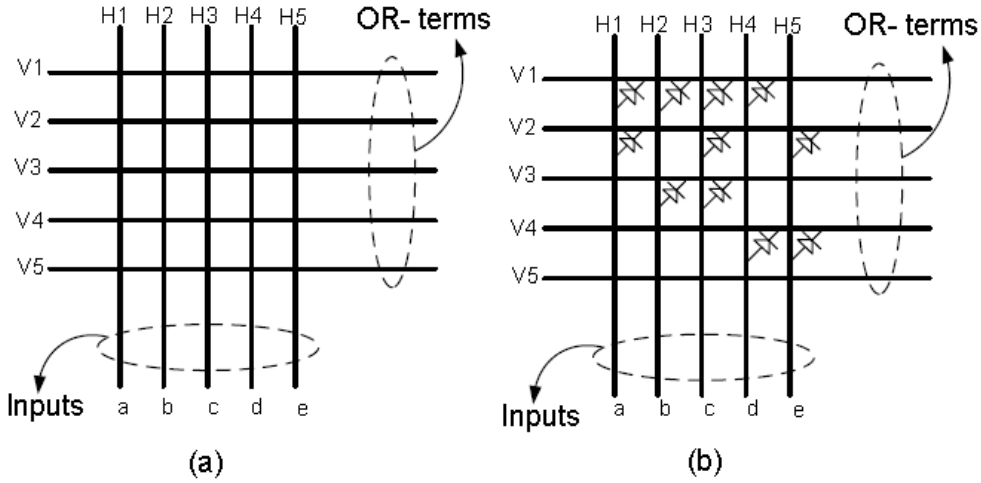

Figure 2.27: (a) A logic array of NanoPLA (b) Programmed logic array.

closed, and crosspoints of OFF inputs are left open. Figure 2.27 shows an example of mapping four OR functions, $f_1 = a + b + c + d$, $f_2 = a + c + e$, $f_3 = b + c$, and $f_4 = d + e$, with logical inputs, a, b, c, d, and e. The logic array inputs a to e are assigned to input nanowires $H1$ to $H5$, respectively. In the case like Figure 2.27(b) where there is no defect in the array, each OR function can be mapped to any nanowire. Here OR functions 1 to 4 are mapped to nanowires $V1$ to $V4$ respectively.

But, nanoscale logic array may contain defective crosspoints so the problem is to find an assignment of the OR functions to the OR-term nanowires. The idea of the proposed algorithm proposed is that since in each OR function there are always some OFF inputs, i.e. some of the junctions will always be left open, if there is a nanowire with defective junctions only at a subset of those positions,

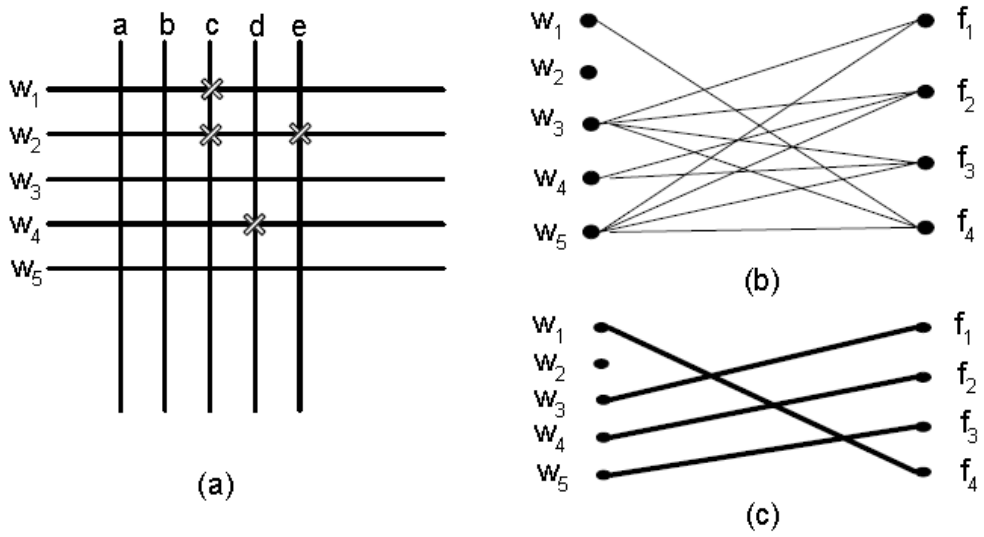

Figure 2.28: (a) Crosses show defective junctions (b) Graph of the OR-term nanowiores and OR functions (c) One possible assignment.

then this defective nanowire can be successfully assigned to the OR function [28]. Let F be the set of OR functions and W be the set of physical OR-term nanowires. The problem is then to find an assignment of OR functions to the nanowires. The problem can be formally stated as finding a bipartite matching from the set F to the set W. Every matching of size $|F|$ on the bipartite graph is a valid assignment of the OR functions to the OR-term nanowires, because it finds an assignment for all of the OR functions in F. Figure 2.28(b) shows a bipartite graph $G(F, W, E)$. Set F is the set of OR functions, and set W is the set of nanowires in the nanoPLA of Figure 2.28(a). Figure 2.28(c) shows one possible matching.

Their proposed greedy heuristic algorithm maps the OR functions on a logic array with defective crosspoints and is based on sorting f_i of F based on the

expected value of their degree.

2.7 Defect-Tolerant FPGA Design Techniques

The regularity of FPGAs makes them suitable candidates for nanotechnology implementation. Reconfigurability in FPGAs has historically been used as means for mitigating the effect of defects. For permanent defects, a recent common practice for the largest two FPGA manufacturers has been to try mapping available designs on working blocks of the FPGA and use it as an "application-specific" FPGA [46].

The much more widely used fault tolerance application for FPGAs is fault tolerance for transient faults due to single event upsets (SEUs). Until recently, SEUs due to charged particles have been a threat mostly in remote and space computers. As explained in Section 2.5, with device dimension shrinking to nanometer scales, the threat of SEUs is now very significant even for terrestrial applications.

In the following sections, first a brief description of FPGA architecture is presented followed by categorization and brief survey of defect-tolerant FPGA design techniques.

2.7.1 Field Programmable Gate Array Architecture

FPGA architecture and terminology vary between vendors. A general and simplifed FPGA architecture is shown in Figure 2.29 . The functional primitives of an FPGA are termed as Confgurable Logic Blocks (CLBs) and the FPGA con-

sists of a regular array of CLBs. Each CLB contains at least one Look Up Table (LUT) and Flip Flop (FF). A LUT consists of an SRAM and has typically four to six inputs addressing the SRAM. A LUT can thus implement any logic function with four inputs. To implement a function that is too large to fit in one CLB, the function is split up and placed in several CLBs, connected through the configurable interconnect. The interconnect consists of lines, Switch Blocks (SBs) and Connection Blocks (CBs). Each switch block is configurable and connects lines entering and leaving the switch block. A connection block has a structure similar to the switch block but connects the lines to the inputs and outputs of a CLB. To be able to connect the configured circuit to the outside world, the FPGA also contains Input/Output Blocks (IOBs). An IOB is often similar in structure to a CLB but has additional circuitry for connecting to a physical pin on the FPGA [71].

A modern FPGA is more complex than the FPGA shown in Figure 2.29. The interconnect is more flexible, with long lines that bypass several switch blocks for reduced delay. The CLBs are often clustered to reduce delay for local connections. Each CLB also contains several configurable multiplexers to increase the flexibility of internal CLB routing and dedicated carry chains to reduce delay when implementing adder circuits. A LUT can also be configured as a small memory block or a shift register. The FPGA may also contain specialized units like dedicated RAM blocks and complete processor cores, all embedded in the array of CLBs. The FPGA is mostly SRAM based (although other types of FPGAs

are also availbele but only SRAM based FPGAs are relevant to this thesis work) which means that all configurable elements are controlled by at least one SRAM cell. The set of all configuration SRAM cells is called the configuration memory of the FPGA. When a FPGA is to be programmed, a bit file containing a value for every SRAM cell in the configuration memory is uploaded to the FPGA. This bit file is the result of an automated design flow, where a circuit described in a Hardware Description Language (HDL) is synthesised, placed, routed and converted to a suitable bit file for the device [71].

2.7.2 Categorization of Defect and Fault-Tolerant Techniques for FPGAs

Defect and Fault-tolerant techniques based on redundancy can be loosely classified into three groups [46]:

- software based redundancy techniques,
- hardware based redundancy techniques and
- run-time based redundancy techniques.

Each of these approaches have their advantages, and typically trade-off between time (critical path delay and processing/application time) and resources (silicon area, external storage, etc).

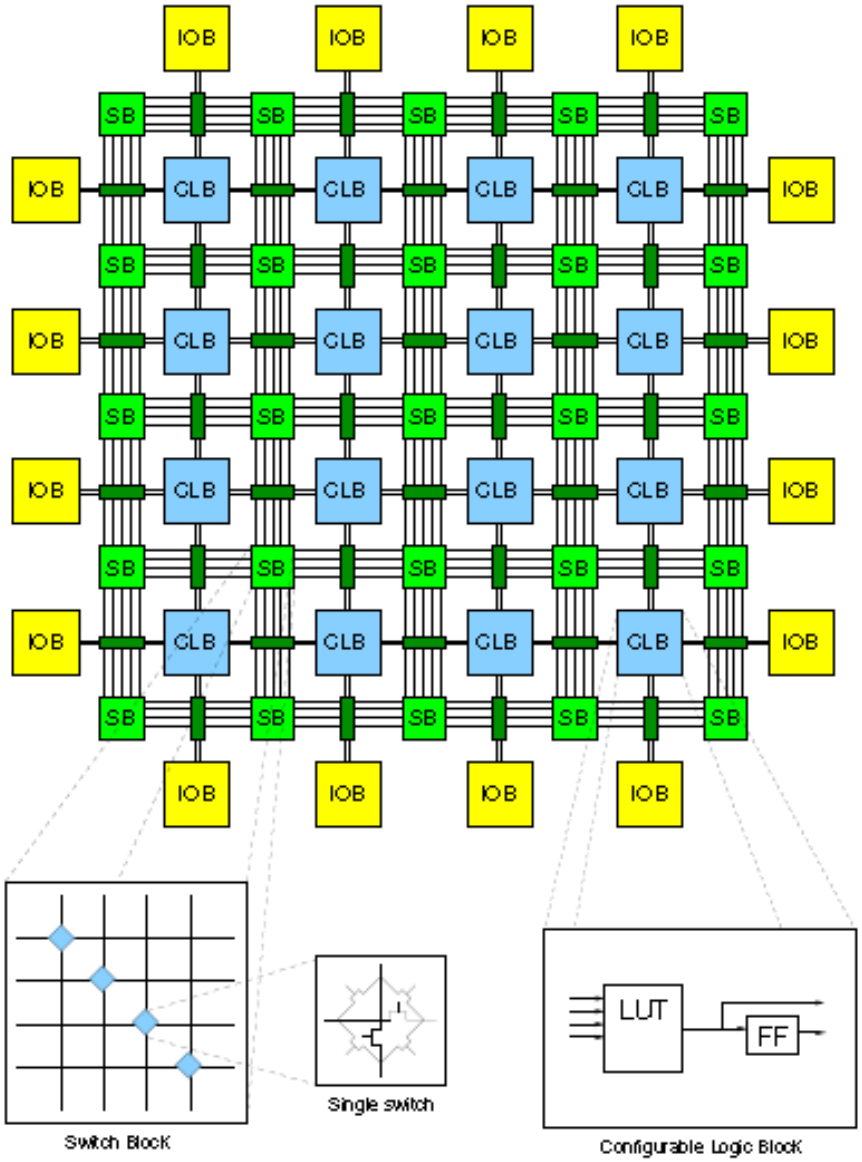

Figure 2.29: Simplified example of an FPGA with 16 CLBs.

Software based Redundancy Techniques

In a software-based redundancy approach, CAD tools are used to map around faulty resources. This method typically has no hardware overhead. The effectiveness and efficiency of correction is dependent on the abilities of the CAD tools. Furthermore, this method is impractical in a production environment because:

1. generating a unique placement and routing solution for each FPGA is time-consuming, and

2. verifying timing of each solution is impossible.

Xilinx solves these problems with their EasyPath [73] technology. Rather than forcing the configuration bitstream to avoid the defects, Xilinx forces the defects to avoid the bitstream. They do this by obtaining the customers final bitstream and selecting chips which contain defects only in the unused portions of the chip. Two other approaches have been proposed to solve these problems. The first method is to precompute a number of placement and routing solutions for a particular design. Each precomputed solution differs by its resource usage. When programming a defective chip, defect correction simply involves selecting the appropriate solution (one that does not use the defective resource(s)) [74, 75]. The second method requires the reservation of spare resources. By carefully avoiding the use of certain resources, it is possible to avoid defects by shifting the entire design [76] by one row or column in the array. Design shifting can be applied in a relatively short amount of time. Without special hardware support, however, shifting results in a slight variance in IO timing. It can also be complicated by heterogeneous (memory

or DSP) blocks in the array. Furthermore, to support multiple defects, they must be perfectly aligned to the spare locations.

Hardware based Redundancy Techniques

Hardware redundancy involves the addition of extra or spare resources. The spare resources allow defective parts to be swapped with empty spare ones. This exchange reduces correction time since the time required to swap is typically less than the time needed to generate a new placement and routing solution. The spare row and column technique is one of the first hardware redundancy approaches and has been successfully applied in industry [77]. This method adds one spare row and one spare column to the layout. It also requires the routing network to be modified. In the event of a defect, the row or column containing the defect is bypassed, and the spare row or column is utilized. The ability to bypass entire rows and columns gives this approach the ability to tolerate defect clusters. Unfortunately, published research does not present the delicate circuit details needed to perform the bypass. Altera patents provide some insight and indicate that additional circuitry is required for bypassing [78]. Redundancy can be implemented at a finer level. For example, additional connections can be added inside the switch block to tolerate one transistor defect per switch block [79]. Unfortunately, this approach is impractical because it significantly alters delay.

Run-time based Redundancy Techniques

Fault tolerance can also be addressed during run-time. As transistor sizes shrink, FPGAs become susceptible to transient faults such as single event upsets [80, 81]. To alleviate this problem, techniques have been developed to detect and correct transient errors through reprogramming or bit scrubbing [82, 83]. However, it is not clear whether these techniques can be extended to correct permanent manufacturing defects; simply reprogramming is insufficient.

Combinations of software and hardware redundancy have also been proposed.

2.7.3 Survey of Defect-Tolerant Techniques for FPGAs

Yu et al. have proposed adding routing resources to facilitate and simplify defect correction and a new switch block design to allow defects to be bypassed by computing a new configuration for a small, localized part of the FPGA [46]. This ensures that areas outside of the neighborhood of the first defect can still tolerate other defects. The affected neighborhood is so small that defect correction can be achieved by modifying the configuration bitstream alone. The defect correction also introduces minimal timing disturbances. The paper also proposes a fault-tolerant design for the switch blocks for yield enhancement. The proposed technique is based on wrapping additional multiplexers around the switch blocks to allow different routes for signals. The additional multiplexers lead to higher probability of finding a route between endpoints.

Huang et al. presented a scheme for evaluating the fault tolerance of different

FPGAs based on reconfigurability of routing resources in the presence of faulty switches [19, 44]. Routability (a realizable route between input and output endpoints) is used as a measure of fault tolerance. As switches fail, the probability of finding a route between endpoints decreases. The paper uses open and short switches as faults.

Yu et al. have also compared coarse and fine-grain redundancy in FPGAs to tolerate defects [48]. For coarse-grain redundancy, they use spare rows and columns, and for fine-grain redundancy they use the fault-tolerant switch block design proposed in [46]. Their findings support using fine-grain redundancy since it offers much higher fault tolerance. They argue that matching the fine-grain fault tolerance using coarse-grain fault tolerance requires double the hardware overhead of fine-grain redundancy, which is about 50%.

CHAPTER 3

DEFECT-TOLERANT N^2-TRANSISTOR STRUCTURES

In this chapter, detailed investigation of a recently proposed transistor-level defect-tolerant design technique called Quadded-Transistor structure is performed. The theoretical and experimental analysis of Quadded-Transistor structure are extended to develop Nona-Transistor structure. Comparison of defect tolerance of proposed transistor-level technique is performed with Quadded Logic technique and the advantages of the proposed technique over other techniques are discussed. Also, hybrid of quadded-transistor and nona-transistor technique with TMR technique is also investigated.

3.1 Introduction

As discussed in Chapter 2, two main approaches have been proposed in the context of reliable nanoelectronics: defect tolerance and defect avoidance. Defect tolerance techniques are based on adding redundancy in the design to tolerate defects or faults. However, defect avoidance techniques are based on identifying the defects and avoiding them possibly through the use of reconfigurable blocks. Recently, traditional fault tolerance techniques such as triple-modular redundancy, triple interwoven redundant logic, and quadded logic have been investigated [11] with the aim to improve the defect tolerance of nanoelectronics design. It has been demonstrated that such techniques are capable of making nanoelectronic circuits more robust to defects.

The previous approaches of defect-tolerance for reliable nanoelectronics have focused on adding redundancy at the functional or unit level such as TMR [12, 13], or gate level such as quadded logic [11]. In this chapter, adding redundancy at the transistor level is investigated and it is shown that it provides higher defect tolerance than unit and gate levels. The work discussed in this chapter is mainly inspired by the work reported in [31, 32] and should be considered as an extension of it.

Adding redundancy at the transistor level itself to improve reliability is not new. Indeed, in [23, 24] transistors were employed to improve the reliability of relay networks. In this chapter, investigation of the effectiveness of transistor-level approach when applied to ISCAS benchmark circuits is performed, since in

[23, 24] bipolar transistors were employed with very simple circuits. Circuit defect tolerance based on N^2-transistor structure with respect to stuck-open, stuck-short and bridging defects is investigated. Furthermore, a comparison is made with recent approaches proposed for defect tolerance in nanoelectronics.

This chapter is organized as follows. The proposed defect-tolerant technique is described in Section 3.2. Experimental results analyzing the defect tolerance of stuck-open, stuck-short and bridging defects are given in Section 3.3. Section 3.4 summarizes and concludes the chapter.

3.2 N^2-Transistor Structures

In this section, the proposed techniques for achieving defect tolerance based on adding redundancy at the transistor-level for electronic circuits are described. Application of transistor-level redundancy for nanoelectronic circuits is also equally significant because it is reported in literature that few of the emerging nanoelectronic technologies show MOSFET like behavior. IBM has recently demonstrated experimentally that carbon nanotubes exhibit electrical characteristics that are similar to that of the state-of-the-art Silicon based MOSFETs [84].

Two transistor structures are described in the following sections.

- Quadded-Transistor Structure
- Nona-Transitor Structure

3.2.1 Quadded-Transistor Structure

The quadded-transistor technique, proposed by El-Maleh et al. in [31] addresses the defect tolerance of transistor stuck-open, stuck-short and bridging defects between gate terminals of transistors. A transistor is considered defective if its expected behavior changes regardless of the type of defect causing it. In order to tolerate single defective transistors, each transistor, A, is replaced by a quadded-transistor structure implementing either the logic function $(A+A)(A+A)$ or the logic function $(AA)+(AA)$, as shown in Figure 3.1. In both of the quadded-transistor structures shown in Figure 3.1(b) and (c), any single transistor defect (stuck-open, stuck-short, AND/OR-bridge) will not change the logic behavior, and hence the defect is tolerated. It should be observed that for NMOS transistors, OR-bridge and stuck-short defects produce the same behavior while AND-bridge and stuck-open defects have the same behavior. Similarly, for PMOS transistors, OR-bridge and stuck-open defects produce the same behavior while AND-bridge and stuck-short defects have the same behavior. Double stuck-open (or their corresponding bridge) defects are tolerated as long as they do not occur in any two parallel transistors (T1&T2 or T3&T4 for the structure in Figure 3.1(b), and T1&T2, T1&T4, T3&T2 or T3&T4 for the structure in Figure 3.1(c)). Double stuck-short (or their corresponding bridge) defects are tolerated as long as they do not occur in any two series transistors (T1&T3, T1&T4, T2&T3 or T2&T4 for the structure in Figure 3.1(b), and T1&T3 or T2&T4 for the structure in Figure 3.1(c)). In addition, any triple defect that does not include two parallel stuck-open

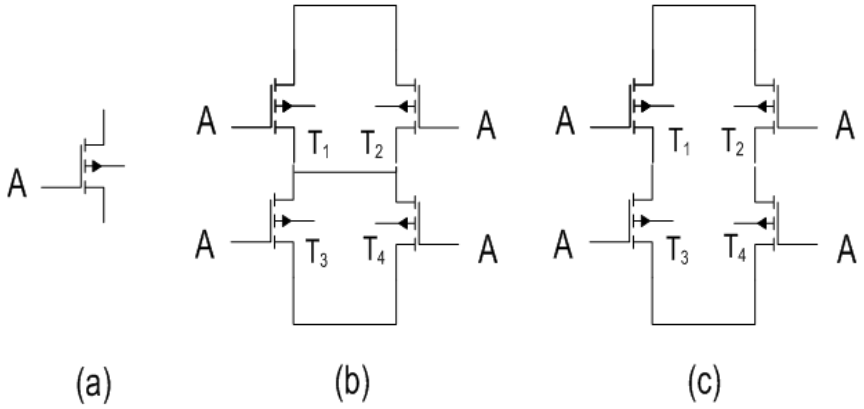

Figure 3.1: (a) Transistor in original gate implementation, (b) First quadded-transistor structure, (c) Second quadded-transistor structure.

defects or two series stuck-short defects or their corresponding bridging defects is tolerated. Thus, one can easily see that using either of the quadded-transistor structures, the reliability of gate implementation could be significantly improved. It should be observed that the effective resistance of the quadded-transistor structures has the same resistance as the original transistor. However, in the presence of a single defect, the worst case effective resistance of the first quadded-transistor structure (Figure 3.1(b)) is $1.5R$ while that of the second quadded-transistor structure (Figure 3.1(c)) is $2R$, where R is the effective resistance of a transistor. This occurs in the case of single stuck-open (or corresponding bridge) defects. For tolerable multiple defects, the worst case effective resistance of both structures is $2R$. For this reason, the first quadded-transistor structure (Figure 3.1(b)) is adopted in our theoretical and experimental work.

Next, the probability of circuit failure given a transistor defect probability

using quadded-transistor structures is determined according to the method given in [31, 32]. A transistor is considered defective if it does not function properly due to manufacturing defects.

Theorem 1 *Given a transistor-defect probability, P, the probability of quadded-transistor structure failure is*

$$P_q = \frac{3}{2}P^2 - \frac{1}{2}P^3$$

Proof This is proved with respect to stuck-open and stuck-short defects as bridge defects have equivalent behaviors to them as explained above.

If there are only two defective transistors in a quadded-transistor structure, then there are four possible pairs of stuck-open and stuck short defects. In all cases, only one of those pair of defects produces an error. Thus, the probability of failure in this case is:

$$\frac{1}{4}\begin{pmatrix} 4 \\ \\ 2 \end{pmatrix} P^2(1-P)^2 = \frac{3}{2}P^2(1-P)^2$$

If three transistors are assumed defective, then there are eight possible combinations of stuck-open and stuck short defects. In all cases, five out of those

combinations produce an error. Thus, the probability of failure in this case is:

$$\frac{5}{8}\begin{pmatrix} 4 \\ 3 \end{pmatrix} P^3(1-P) = \frac{5}{2}P^3(1-P)$$

If four transistors are assumed defective, then in this case there will always be an error and the probability of failure is:

$$1\begin{pmatrix} 4 \\ 4 \end{pmatrix} P^4 = P^4$$

Thus, the probability of quadded-transistor structure failure is:

$$\begin{aligned} P_q &= \frac{3}{2}P^2(1-P)^2 + \frac{5}{2}P^3(1-P) + P^4 \\ P_q &= \frac{3}{2}P^2 - 3P^3(1-P) + \frac{3}{2}P^4 + \frac{5}{2}P^3 - \frac{5}{2}P^4 + P^4 \\ P_q &= \frac{3}{2}P^2 - \frac{1}{2}P^3(1-P) \end{aligned}$$

Theorem 2 *Given a transistor-defect probability, P, and a circuit with N quadded-transistor structures, the probability of circuit failure and circuit reliability*

are

$$P_f = \sum_{i=1}^{N} (-1)^{i+1} \begin{pmatrix} N \\ i \end{pmatrix} (P_q)^i$$

$$R = 1 - P_f = \sum_{i=1}^{N} (-1)^{i+1} \begin{pmatrix} N \\ i \end{pmatrix} (P_q)^i$$

Theorem 2 is based on the inclusion-exclusion principle [97]. The probability of circuit failure may also be computed based on the binomial distribution as

$$P_f = \sum_{i=1}^{N} \begin{pmatrix} N \\ i \end{pmatrix} \left(P_q{}^i\right) (1 - P_q)^{N-i}$$

which gives equivalent results. It is assumed in the work in [31, 32] that circuit reliability represents the probability that the circuit will function correctly in the presence of defects. It should be observed that while the result above represents the exact circuit failure probability for stuck-open and stuck-short defects, it represents an upper bound for bridging defects. This is due to the fact that not all bridging defects that result in a faulty quadded-transistor structure result in a faulty gate behavior. For example, AND-bridging defects between gate terminals of transistors within the same NAND gate do not change the gate behavior regardless of their multiplicity. Similarly, OR-bridging defects between gate terminals of

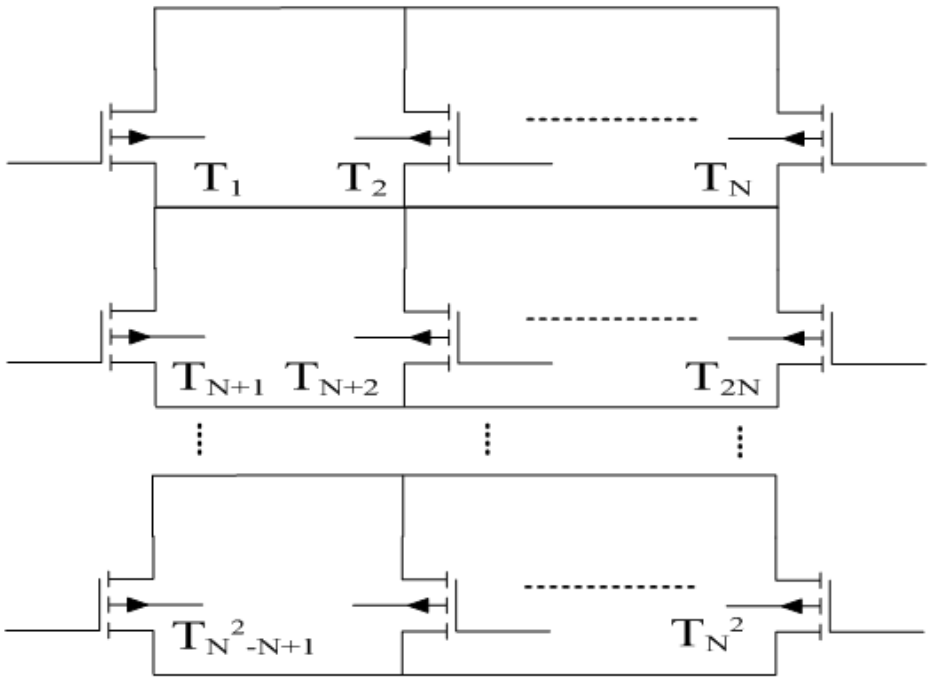

Figure 3.2: Defect-tolerant N^2-transistor structure.

transistors within the same NOR gate do not change the gate behavior regardless of their multiplicity.

The quadded-transistor structure, given in Figure 3.1(b), can be generalized to an N^2-transistor structure, where $N \geq 2$. An N^2-transistor structure is composed of N blocks connected in series with each block composed of N parallel transistors, as shown in Figure 3.2. An N^2-transistor structure guarantees defect tolerance of all defects of multiplicity less than or equal to $(N-1)$ in the structure. Hence, a large number of multiple defects can be tolerated in a circuit implemented based on these structures. It is obvious that quadded-transistor structure is an N^2-transistor structure with $N = 2$.

3.2.2 Nona-Transistor Structure

The nona-transistor structure is an extension of the quadded-transistor structure. In the nona configuration ($N = 3$), each transistor, A, is replaced by a nona-transistor structure implementing either the logic function $(A+A+A)(A+A+A)(A+A+A)$ or the logic function $(AAA)+(AAA)+(AAA)$, as shown in Figure 3.3. In both of the nona-transistor structures shown in Figure 3.3(b) and (c), any single transistor defect (stuck-open, stuck-short, AND/OR-bridge) will not change the logic behavior, and hence the defect is tolerated. Double stuck-open (or their corresponding bridge) defects are also always tolerated. Double stuck-short (or their corresponding bridge) defects are also always tolerated. In addition, any triple defect that does not include three parallel stuck-open defects or three series stuck-short defects or their corresponding bridging defects is tolerated. Thus, one can easily see that using either of the nona-transistor structures, the reliability of gate implementation could be significantly improved.

The nona-transistor structure shown in Figure 3.3(b) is adopted in this work for the same reason of having less resistance in case of tolerable defects in the transistors as explained in the previous section for quadded-transistor structure. Next, the probability of failure for nona-transistor structure where N=3 is determined based on a similar analysis as done for quadded-transistor structure [31, 32].

Theorem 3 *Given a transistor-defect probability, P, the probability of a nona-*

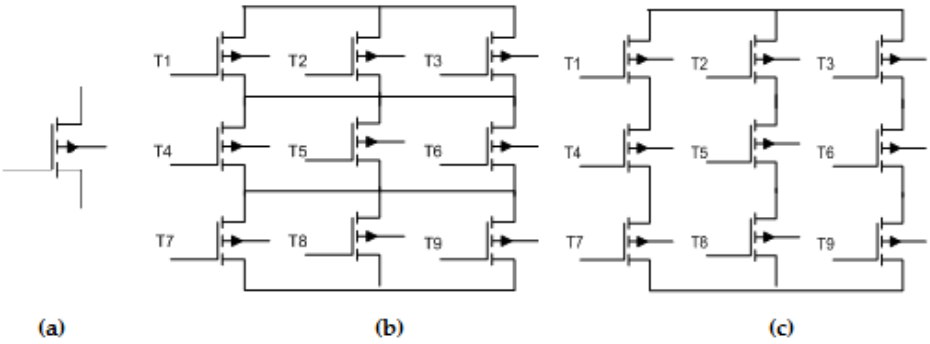

Figure 3.3: (a) Transistor in original gate implementation, (b) First nona-transistor structure, (c) Second nona-transistor structure.

transistor structure failure is

$$P_n = \frac{30}{8}P^3 - \frac{81}{16}P^4 + \frac{27}{8}P^5 - \frac{21}{16}P^6 + \frac{31}{128}P^7 - \frac{19}{128}P^8 + \frac{5}{32}P^9 \qquad (3.1)$$

This is proved as follows.

Proof This is also proved with respect to stuck-open and stuck-short defects.

If there are only two defective transistors in a nona-transistor structure, the defect will always be tolerated.

If three defective transistors are assumed defective in a nona-transistor structure, then there are eight possible combinations of stuck-open and stuck short defects. In all cases, only one of those combinations of defects produces an error for 3 unique parallel (stuck-open) and 27 unique series (stuck-short) defective

transistor structures. Thus, the probability of failure in this case is:

$$(3 \times \frac{1}{8} + 27 \times \frac{1}{8})P^3(1-P)^6$$

If four transistors are assumed defective, then there are sixteen possible combinations of stuck-open and stuck short defects. Among those, only two combinations produce an error for 18 unique parallel transistor structures. Moreover, only three combinations produce an error for 81 unique series transistor structures. Thus, the probability of failure in this case is:

$$(18 \times \frac{2}{16} + 81 \times \frac{3}{16})P^4(1-P)^5$$

If five transistors are assumed defective, then there are thirty two possible combinations of stuck-open and stuck short defects. Among those, only four combinations produce an error for 18 unique parallel transistor structures. Moreover, only eleven combinations produce an error for 27 series transistor structures which are overlapping with parallel transistor structures. Also, nine combinations produce an error for 81 series transistor structures which are non-overlapping with parallel transistor structures. Thus, the probability of failure in this case is:

$$(18 \times \frac{4}{32} + 27 \times \frac{11}{32} + 81 \times \frac{9}{32})P^5(1-P)^4$$

If six transistors are assumed defective, then there are sixty-four possible combinations of stuck-open and stuck short defects. Among those, only fifteen combina-

tions produce an error for 3 unique parallel transistor structures. Moreover, only twenty-nine combinations produce an error for 54 series transistor structures which are overlapping with parallel transistor structures. Also, twenty-seven combinations produce an error for 27 series transistor structures which are non-overlapping with parallel transistor structures. Thus, the probability of failure in this case is:

$$(3 \times \frac{15}{64} + 54 \times \frac{29}{64} + 27 \times \frac{27}{64})P^6(1-P)^3$$

If seven transistors are assumed defective, then there are one hundred and twenty eight possible combinations of stuck-open and stuck short defects. Among those, there are no unique parallel transistor structures. Moreover, only seventy-four combinations produce an error for 1 series transistor structure which is overlapping with parallel transistor structures. Also, seventy-nine combinations produce an error for the other 35 series transistor structures which are overlapping with parallel transistor structures. There are no series transistor structures which are non-overlapping with parallel transistor structures. Thus, the probability of failure in this case is:

$$(1 \times \frac{74}{128} + 35 \times \frac{79}{128})P^7(1-P)^2$$

If eight transistors are assumed defective, then there are two hundred and fifty six possible combinations of stuck-open and stuck short defects. Among those, there are no unique parallel transistor structures. Moreover, only one hundred and fifty eight of those combinations produce an error for 1 series transistor structure which

is overlapping with parallel transistor structures. Also, two hundred and seven combinations produce an error for the other 8 series transistor structures which are overlapping with parallel transistor structures. There are no series transistor structures which are non-overlapping with parallel transistor structures. Thus, the probability of failure in this case is:

$$(1 \times \frac{158}{256} + 8 \times \frac{207}{256}) P^8 (1 - P)$$

If nine transistors are assumed defective, then in this case there will always be an error and the probability of failure is

$$1 \begin{pmatrix} 9 \\ 9 \end{pmatrix} P^9 = P^9$$

Thus, the probability of nona-transistor structure failure is:

$$\begin{aligned}
P_n &= (3 \times \frac{1}{8} + 27 \times \frac{1}{8})P^3(1-P)^6 \\
&\quad +(18 \times \frac{2}{16} + 81 \times \frac{3}{16})P^4(1-P)^5 \\
&\quad +(18 \times \frac{4}{32} + 27 \times \frac{11}{32} + 81 \times \frac{9}{32})P^5(1-P)^4 \\
&\quad +(3 \times \frac{15}{64} + 54 \times \frac{29}{64} + 27 \times \frac{27}{64})P^6(1-P)^3 \\
&\quad +(1 \times \frac{74}{128} + 35 \times \frac{79}{128})P^7(1-P)^2 \\
&\quad +(1 \times \frac{158}{256} + 8 \times \frac{207}{256})P^8(1-P) \\
&\quad +P^9 \\
P_n &= \frac{30}{8}P^3 - \frac{81}{16}P^4 + \frac{27}{8}P^5 - \frac{21}{16}P^6 + \frac{31}{128}P^7 - \frac{19}{128}P^8 + \frac{5}{32}P^9
\end{aligned}$$

Theorem 4 *Given a transistor-defect probability, P, and a circuit with N nona-transistor structures, the probability of circuit failure and circuit reliability are*

$$\begin{aligned}
P_f &= \sum_{i=1}^{N} (-1)^{i+1} \begin{pmatrix} N \\ i \end{pmatrix} (P_n)^i \\
R &= 1 - P_f = \sum_{i=1}^{N} (-1)^{i+1} \begin{pmatrix} N \\ i \end{pmatrix} (P_n)^i
\end{aligned}$$

Based on the analysis of the quadded-transistor and nona-transistor structures, it can be deduced that the probability of failure for an N^2-transistor structure will be $O(P^N)$. The N^2-transistor structure, for $N > 2$, may be applied selectively for critical gates due to its increased overhead.

An interesting advantage of the N^2-transistor structure is that it fits well in existing design and test methodologies. In synthesis, a library of gates implemented based on the N^2-transistor structure will be used in the technology mapping process. The same testing methodology will be used assuming testing is done at the gate level based on the single stuck-at fault model. So, the same test set derived for the original gate-level structure can be used without any change.

Figure 3.4 compares the reliability of several NAND gates of various inputs, including 2, 4 and 8, implemented using the quadded-transistor structure, the nona-transistor structure and conventional (pull-up, pull-down) CMOS implementation for stuck-open and stuck-short defects. As can be seen, the reliability of gates implemented using the quadded-transistor and nona-transistor structures is significantly higher than the reliability of conventional gate implementation. For example, for an 8-input NAND gate, with a probability of transistor failure = 10%, the gate reliability for the nona-transistor structure-based design is 95%, the gate reliability for the quadded-transistor structure-based design is 79%, while the gate reliability for the conventional CMOS implementation is 19%. Furthermore, as the number of inputs increases, the probability of gate failure increases and reliability decreases, as expected.

The gate capacitance that the quadded-transistor structure induces on the gate connected to the input A is four times the original gate capacitance. This has an impact on both delay and power dissipation. However, as shown in [38], a gate with higher load capacitance has better noise rejection curves and hence is more resistant to soft errors resulting in noise glitches.

While the quadded-transistor structure increases the area, this increase is less than other gate-level defect tolerance techniques as will be shown in the experimental results. As with all defect tolerance techniques, the increase in area, power and delay is traded off by more circuit reliability. This is justified given that it is predicted that nanotechnology will provide much higher integration densities, speed and power advantages [34].

3.3 Experimental Results

To demonstrate the effectiveness of the N^2-transistor structure technique, experiments have been performed on a number of largest ISCAS85 and ISCAS89 benchmark circuits (replacing flip-flops by inputs and outputs). Two types of permanent defects are analyzed separately: transistor stuck-open and stuck-short defects, and AND/OR bridging defects. For evaluating circuit failure probability and reliability, the simulation-based reliability model used in [11] is adopted. Circuit reliability based on the quadded-transistor and nona-transistor structures is compared with the approaches in [11] including Triple Interwoven Redundancy (TIR) and Quadded logic. A complete test set T that detects all detectable single

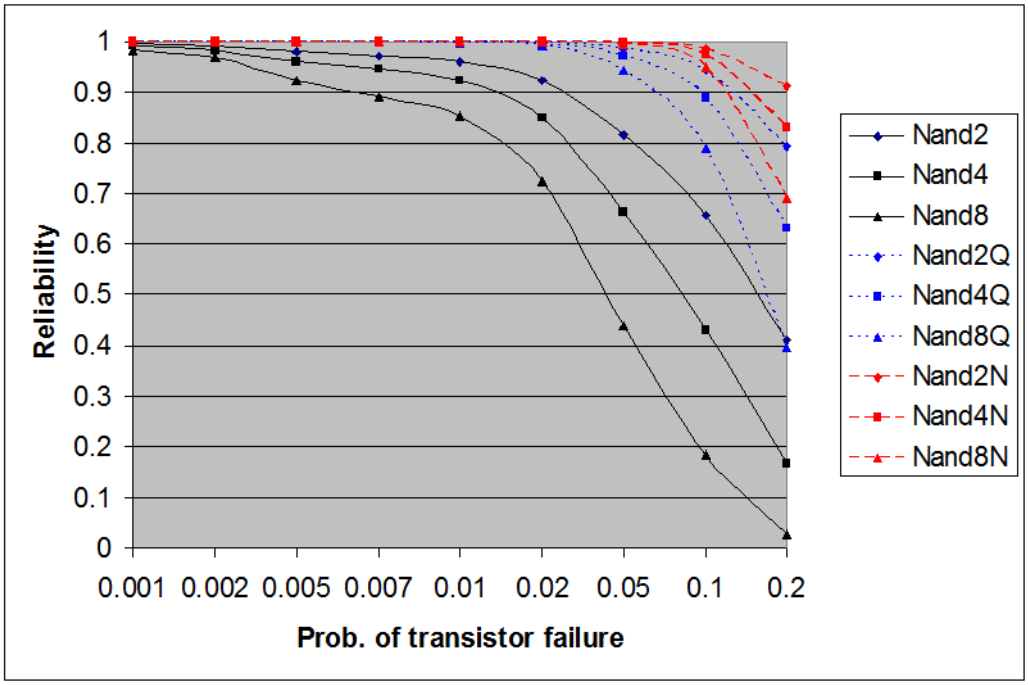

Figure 3.4: Gate reliability comparison between quadded-transistor structure (Q), nona-transistor structure (N) and conventional CMOS.

stuck-at faults in a circuit is used. Test sets generated by Mintest ATPG tool [69] are used. To compute the circuit failure probability, F_m, resulting from injecting m defective transistors, the following procedure is used:

1. Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.
2. Simulate the fault-free circuit by applying the test set T.
3. Randomly inject m transistor defects.
4. Simulate the faulty circuit by applying the test set T.
5. If the outputs of the fault-free and faulty circuits are different, increment K by 1.
6. Decrement I by 1 and if I is not 0 goto step 3.
7. Failure Rate $F_m = K/1000$.

Assuming that every transistor has the same defect probability, P, and that defects are randomly and independently distributed, the probability of having a number of m defective transistors in a circuit with N transistors follows the binomial distribution [11] as shown below:

$$P(m) = \begin{pmatrix} N \\ m \end{pmatrix} P^m (1-P)^{N-m}$$

Assuming the number of transistor defects, m, as a random variable and using the circuit failure probability F_m as a failure distribution in m, the probability of circuit failure, F, and circuit reliability, R, are computed as follows [11]:

$$F = \sum_{m=1}^{N} F_m \times P_m$$
$$R = 1 - F = \sum_{m=1}^{N} F_m \times P_m$$

3.3.1 Stuck-Open and Stuck-Short Defect Analysis

Figures 3.5 and 3.6 show the reliability of some of the ISCAS85 benchmark circuits obtained both theoretically and experimentally based on the above simulation procedure and formulas for stuck-open and stuck-short defects for quadded-transistor and nona-transistor structures respectively. As can be seen, there is almost identical match, clearly validating the derived theoretical results.

For TMR to be effective, a careful balance between the module size and the number of majority gates used needs to be made. For this reason, comparison of the reliability of ISCAS benchmark circuits between the quadded-transistor and nona-transistor structures and quadded logic is being presented in this chapter.

A comprehensive comparison of the probability of circuit failure between the quadded-transistor structure and the quadded logic is given in Table 3.1 for several

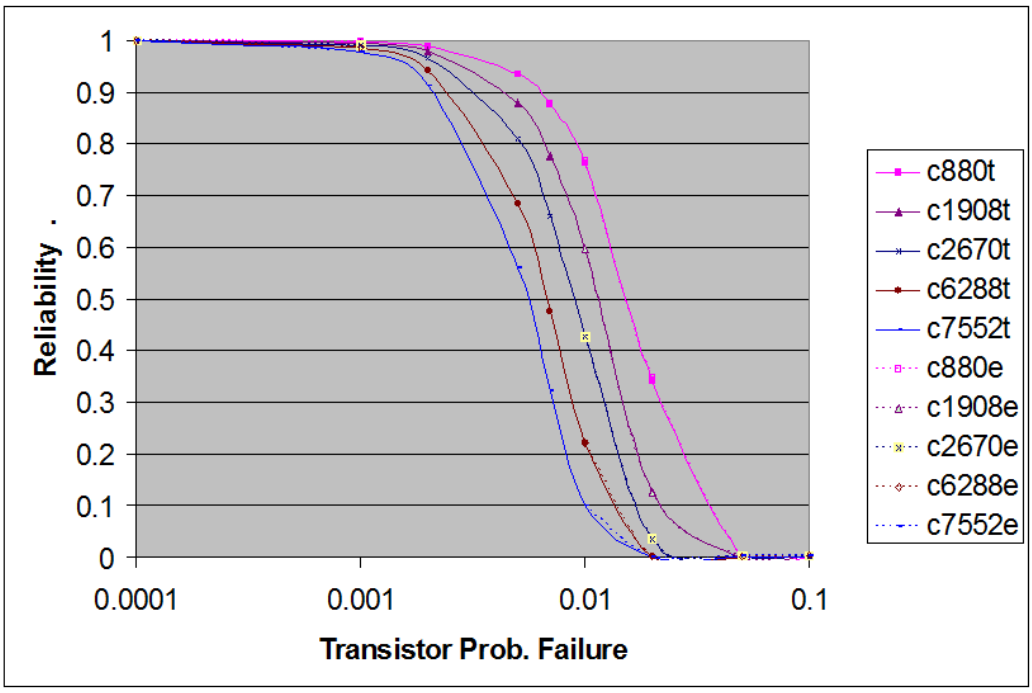

Figure 3.5: Reliability obtained both theoretically (t) and experimentally (e) based on quadded-transistor structure and stuck-open and stuck-short defects.

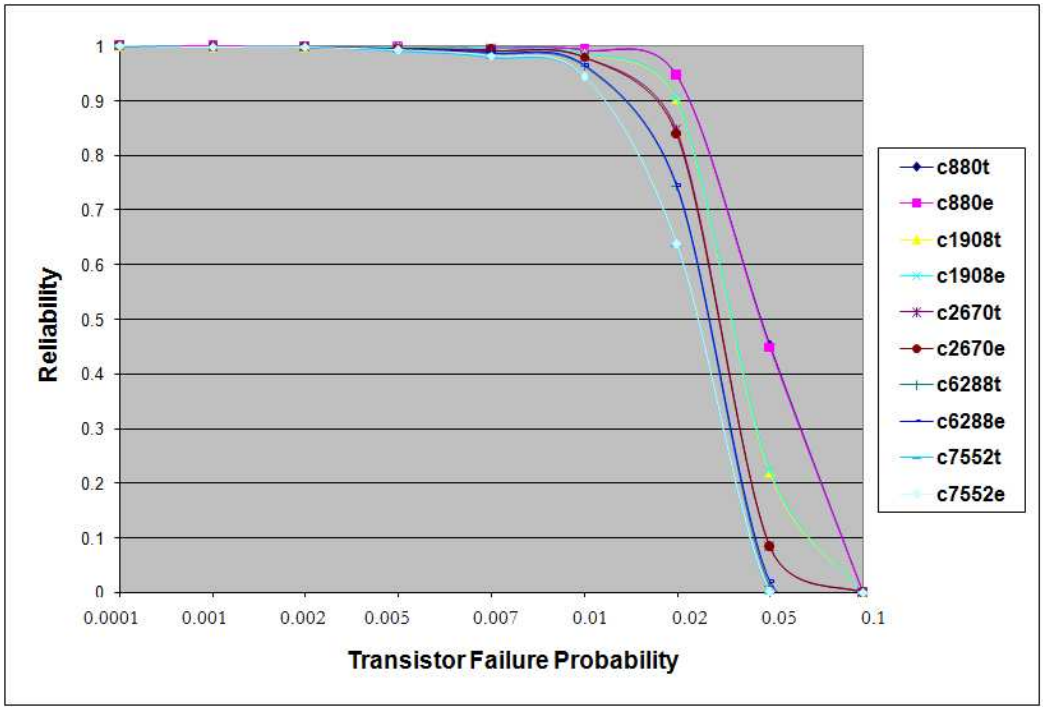

Figure 3.6: Reliability obtained both theoretically (t) and experimentally (e) based on nona-transistor structure and stuck-open and stuck-short defects.

Table 3.1: Comparison of circuit failure probability between quadded-transistor structure and quadded logic approaches for stuck-open and stuck-short defects.

	Quadded-Transistor Structure					Quadded Logic				
Cct.	Trans.	0.25%	0.5%	0.75%	1%	Trans.	0.25%	0.5%	0.75%	1%
c880	7208	0.015	0.060	0.135	0.237	13616	0.452	0.783	0.905	0.978
c1355	9232	0.023	0.082	0.176	0.287	18304	0.531	0.846	0.975	0.995
c1908	13784	0.030	0.115	0.248	0.400	24112	0.673	0.94	0.984	≈ 1
c2670	22672	0.047	0.188	0.375	0.569	36064	0.958	0.999	≈ 1	≈ 1
c3540	30016	0.067	0.238	0.457	0.674	46976	0.59	0.901	0.996	0.999
c5315	45048	0.095	0.341	0.614	0.816	74112	0.991	≈ 1	≈ 1	≈ 1
c6288	40448	0.085	0.307	0.576	0.787	77312	0.685	0.962	0.999	≈ 1
c7552	61600	0.136	0.441	0.732	0.909	77312	0.985	≈ 1	≈ 1	≈ 1
s5378	35608	0.081	0.282	0.521	0.737	59760	≈ 1	≈ 1	≈ 1	≈ 1
s9234	74856	0.166	0.510	0.791	0.939	59760	0.999	≈ 1	≈ 1	≈ 1
s13207	103544	0.212	0.625	0.888	0.980	150448	≈ 1	≈ 1	≈ 1	≈ 1
s15850	128016	0.257	0.697	0.936	0.992	171664	≈ 1	≈ 1	≈ 1	≈ 1

percentages of injected stuck-open and stuck-short defects. For all the circuits, the quadded-transistor technique achieves significantly lower circuit failure probability than the quadded logic technique for the same and for twice the percentage of injected defects. For 10 out of 12 circuits, it achieves lower failure probability with four times the percentage of injected defects. In Table 3.2, the circuit reliability results obtained based on the simulation procedure outlined above for the quadded-transistor structure and quadded logic approaches for several transistor defect probabilities based on stuck-open and stuck-short defects are reported. The effectiveness of the quadded-transistor structure technique is clearly demonstrated by the results as it achieves higher circuit reliability with 4 to 5 times more transistor defect probability. This is in addition to the observation that the quadded-transistor structure technique requires nearly half the area of the quadded logic technique as indicated by the number of transistors.

Table 3.2: Comparison of circuit reliability between quadded-transistor structure and quadded logic approaches for stuck-open and stuck-short defects.

	Quadded-Transistor Structure					Quadded Logic				
Cct.	Trans.	0.0001	0.001	0.005	0.01	Trans.	0.0001	0.001	0.005	0.01
c880	7208	0.999	0.997	0.934	0.767	13616	0.979	0.822	0.283	0.042
c1355	9232	0.999	0.996	0.917	0.713	18304	0.975	0.765	0.187	0.008
c1908	13784	0.999	0.994	0.879	0.596	24112	0.975	0.755	0.261	0.001
c2670	22672	0.999	0.991	0.809	0.427	36064	0.904	0.350	0.001	0
c3540	30016	0.999	0.989	0.755	0.327	46976	0.981	0.805	0.237	0
c5315	45048	0.999	0.984	0.656	0.185	74112	0.853	0.227	0.001	0
c6288	40448	0.999	0.986	0.685	0.222	77312	0.971	0.718	0.024	0
c7552	61600	0.999	0.978	0.562	0.101	77312	0.874	0.292	0	0
s5378	35608	0.999	0.985	0.717	0.263	59760	0.811	0.134	0.001	0
s9234	74856	0.999	0.972	0.496	0.061	59760	0.821	0.140	0	0
s13207	103544	0.999	0.961	0.379	0.023	150448	0.518	0.008	0	0
s15850	128016	0.999	0.953	0.302	0.008	171664	0.576	0.009	0	0

In Table 3.3, the circuit failure probability for the nona-transistor structure technique for several percentages of injected defects based on stuck-open and stuck-short defects is reported and in Table 3.4, the circuit reliability for the nona-transistor structure technique for several transistor defect probabilities based on stuck-open and stuck-short defects is reported. The nona-transistor structure technique achieves higher circuit reliability than the quadded logic technique with 20 times more transistor defect probability. It also achieves higher circuit reliability than the quadded-transistor structure technique with 4 to 5 times more transistor defect probability. This should also be observed that the nona-transistor structure technique requires higher number of transistors as compared to quadded-transistor structure and quadded logic techniques.

Table 3.3: Circuit failure probability for the nona-transistor structure approach for stuck-open and stuck-short defects.

Cct.	Trans.	0.25%	0.5%	0.75%	1%
c880	16218	0	0.001	0.003	0.008
c1355	20772	0	0.001	0.006	0.011
c1908	31014	0	0.001	0.005	0.012
c2670	51012	0.001	0.004	0.006	0.020
c3540	67536	0	0.001	0.013	0.024
c5315	101358	0	0.008	0.019	0.028
c6288	91008	0	0.005	0.008	0.039
c7552	138600	0.001	0.006	0.015	0.049
s5378	80118	0.001	0.003	0.013	0.035
s9234	168426	0	0.004	0.032	0.074
s13207	232974	0	0.004	0.040	0.096
s15850	288036	0.003	0.006	0.051	0.128

3.3.2 Bridging Defect Analysis

In order to analyze the defect tolerance of the quadded-transistor structure and the quadded logic techniques to bridging defects, the same simulation-based model was used. The experiments were performed on the same set of ISCAS circuits. The bridging defects were injected randomly between the gates of the defective transistor and one of its neighbors, located within a window of local transistors in the netlist (± 8 transistors). Both AND and OR bridging defects were injected equally. It should be observed that for injecting m defective transistors due to bridges, only $m/2$ bridges need to be injected.

Table 3.5 shows the results obtained for several percentages of injected bridging defects for the quadded-transistor and the quadded logic techniques. As can be seen, the quadded-transistor structure technique exhibits a much lower failure probability than quadded logic technique. The quadded-transistor structure technique achieves failure rates lower than quadded logic for the same and twice the

Table 3.4: Circuit reliability for the nona-transistor structure approach for stuck-open and stuck-short defects.

Cct.	Trans.	0.0001	0.001	0.002	0.005	0.007	0.01	0.02	0.05	0.1
c880	16218	≈ 1	0.999	0.999	0.999	0.997	0.993	0.948	0.453	0.002
c1355	20772	≈ 1	0.999	0.999	0.998	0.997	0.991	0.934	0.363	0.0005
c1908	31014	≈ 1	0.999	0.999	0.998	0.995	0.987	0.904	0.22	0.00001
c2670	51012	≈ 1	0.999	0.999	0.997	0.992	0.979	0.847	0.083	0
c3540	67536	≈ 1	0.999	0.999	0.996	0.99	0.972	0.803	0.037	0
c5315	101358	≈ 1	0.999	0.999	0.994	0.985	0.959	0.719	0.007	0
c6288	91008	≈ 1	0.999	0.999	0.995	0.987	0.963	0.744	0.011	0
c7552	138600	≈ 1	0.999	0.999	0.992	0.981	0.944	0.637	0.0011	0
s5378	80118	≈ 1	0.999	0.999	0.995	0.988	0.967	0.771	0.02	0
s9234	168426	≈ 1	0.999	0.999	0.991	0.976	0.933	0.578	0.00027	0
s13207	232974	≈ 1	0.999	0.999	0.988	0.967	0.908	0.469	0.00001	0
s15850	288036	≈ 1	0.999	0.999	0.985	0.96	0.888	0.392	0	0

percentage of injected bridging faults. For 0.25% of injected defects, it achieves failure rates nine times less than quadded logic and three times less for 0.5% of injected defects in most of the circuits. It should be observed that for the same percentage of defective transistors, the failure rate for bridging defects is less than that of stuck-open and stuck-short defects. This is due to the fact that not all bridging defects will result in a faulty gate behavior.

Since the defect tolerance of circuits using quadded-transistor structures in the presence of stuck-open and stuck-short defects is a lower bound on those in the presence of bridge defects, the defect tolerance of the nona-transistor structure with respect to bridging defects is not performed.

Table 3.5: Comparison of circuit failure probability between quadded-transistor structure and quadded logic approaches for bridging defects.

	Quadded-Transistor Structure					Quadded Logic				
Cct.	Trans.	0.25%	0.5%	0.75%	1%	Trans.	0.25%	0.5%	0.75%	1%
c880	7208	0.011	0.046	0.084	0.134	13616	0.168	0.279	0.437	0.539
c1355	9232	0.008	0.047	0.095	0.158	18304	0.195	0.339	0.498	0.571
c1908	13784	0.018	0.091	0.201	0.272	24112	0.384	0.690	0.827	0.916
c2670	22672	0.034	0.110	0.229	0.381	36064	0.768	0.945	0.988	≈ 1
c3540	30016	0.043	0.171	0.325	0.496	46976	0.303	0.532	0.683	0.803
c5315	45048	0.058	0.208	0.419	0.631	74112	0.648	0.866	0.953	0.984
c6288	40448	0.041	0.138	0.292	0.452	77312	0.163	0.324	0.480	0.588
c7552	61600	0.088	0.294	0.512	0.699	77312	0.574	0.837	0.935	0.973
s5378	35608	0.060	0.179	0.392	0.671	59760	0.672	0.793	0.924	0.940
s9234	74856	0.079	0.324	0.572	0.802	59760	0.733	0.929	0.982	0.995
s13207	103544	0.119	0.386	0.661	0.853	150448	0.998	≈ 1	≈ 1	≈ 1
s15850	128016	0.110	0.357	0.649	0.846	171664	0.987	≈ 1	≈ 1	≈ 1

3.3.3 Hybridization of Quadded and Nona-Transistor structures with TIR and TMR

In Figure 3.7, comparison of the probability of circuit failure for a given percentage of stuck-open and stuck-short defects between the quadded-transistor structure (QT), nona-transistor structure (NT), quadded logic (QL) [11] and TIR logic [11] is presented. The comparison is made based on an 8-stage cascaded half adder circuit used in [11]. TIR logic is implemented by adding a majority gate for each sum and carry-out signal at each stage. Majority gate is also implemented as a single gate. As can be seen, adding transistor-level defect tolerance generates circuits with significantly less probability of circuit failure than those that add defect tolerance at gate level (QL) and unit level (TMR). This is in addition to smaller area overhead in terms of smaller number of transistors used in the case of quadded-transistor structure. The number of transistors in the quadded-transistor

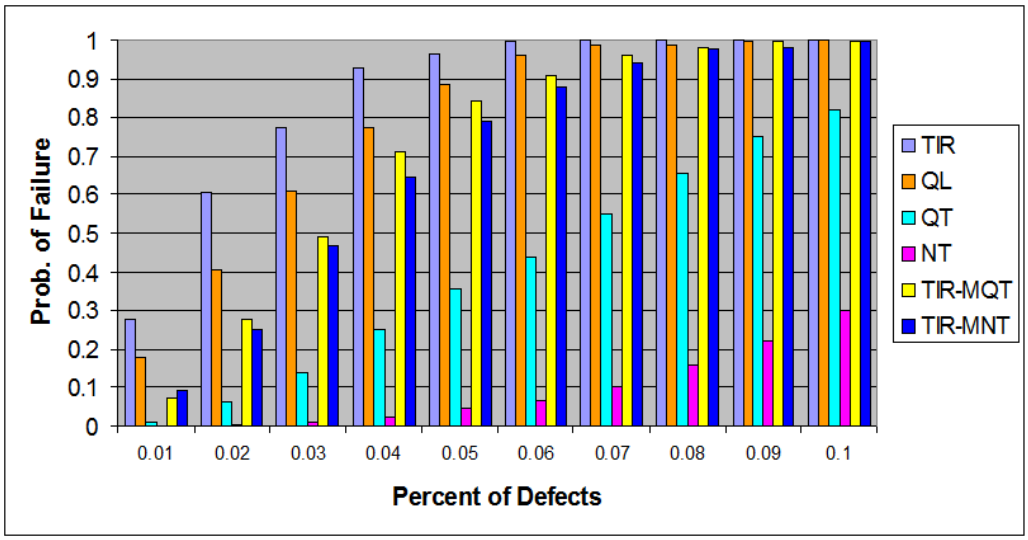

Figure 3.7: Comparison of circuit failure probability for an 8-stage cascaded half-adder circuit for stuck-open and stuck short defects.

structure implementation is 512, while it is 608 for TIR logic, 1024 for quadded logic and 1152 for the nona-transistor structure.

The probability of circuit failure for TIR and TMR logic can be improved by enhancing the reliability of majority gates. A hybrid approach for improved defect tolerance as proposed in [31, 32] is being followed in this work by implementing the majority gates in the 8-stage cascaded half adder TIR logic circuit based on the quadded-transistor structure (TIR-MQT) and the nona-transistor structure (TIR-MNT). As shown in Figure 3.7, the reliability of the implemented circuit is improved compared to TIR circuit at the expense of increased number of transistors (1280 for TIR-MQT and 2400 for TIR-MNT). However, the reliability of the individual modules needs also more enhancements to improve the

overall reliability of the circuit. This shows an interesting potential application of the N^2-transistor structure in improving the reliability of voter-based redundancy techniques.

3.4 Summary

In this chapter, extension of a recently proposed defect-tolerant technique called quadded-transistor technique has been investigated. Quadded-transistor technique is based on adding redundancy at the transistor level. The theoretical and experimental analysis of the investigated technique for stuck-open and stuck-short defects is extended to develop another transistor level technique called nona-transistor structure. The proposed nona-transistor technique provides defect tolerance against a large number of permanent defects including stuck-open and stuck-short defects. Experimental results have demonstrated that the proposed technique provides significantly less circuit failure probability and higher reliability than recently investigated techniques based on gate level (Quadded Logic) and unit level (Triple Modular Redundancy) but with higher overhead in terms of number of transistors. Hybridization of quadded and nona-transistor techniques with TIR (and TMR) is also investigated by implementing only the majority voters with quadded and nona-transistor techniques. The techniques have been investigated theoretically and by simulation using large ISCAS 85 and 89 benchmark circuits and have significantly improved defect tolerance.

CHAPTER 4

TRANSIENT AND SOFT ERROR MITIGATION USING QUADDED-TRANSISTOR STRUCTURE

In this chapter, the proposed transistor-level defect-tolerant design techniques for transient and soft-error mitigation are described in detail. The experimental analysis is presented along with a comparison of the proposed techniques with the most popular technique for mitigating transient and soft errors i.e., TMR (Triple Modular Redundancy).

4.1 Introduction

Integrated Circuits (ICs) are prone to upsets and transients that occur in aerospace due to various charged particles like neutrons etc [41]. These upsets are the main cause behind soft and transient errors in ICs which have to operate in aerospace-related applications. More recently, ICs have also become prone to upsets at ground level because of the continual evolution of fabrication technology for semiconductors. Drastic device shrinkage, power supply reduction and increasing operating speeds significantly reduce noise margins and hence reduce reliability [47]. This trend is approaching a point at which it will be infeasible to produce ICs that are free from these effects. Consequently, defect and fault tolerance is no longer a matter exclusively for aerospace designers, it is important for the designers of next generation ground level products as well [51].

As discussed in Chapter 2, the high-level SEU mitigation techniques used most often today to protect designs against SEUs are based mainly on TMR [41]. The TMR mitigation scheme uses three identical logic circuits (redundant blocks 0, 1, and 2) synthesized in the module. These circuits perform the same task in parallel, with a majority voter circuit comparing corresponding outputs.

In this chapter, a new quadded-transistor based technique for transient and soft error mitigation is proposed. The proposed technique is a majority voter-less technique and is called Quadded Modular Redundancy (QMR). QMR technique has less area overhead as compared to TMR and also affords more reliability.

In addition to QMR, this chapter also proposes another technique based on

direct application of QT(Quadded-Transistor) structure. This technique is called QT based SEU mitigation technique. Reliability analysis has shown that this technique also outperforms TMR in terms of reliability but has more area overhead because all the gates are implemented using QT(Quadded-Transistor) structures. A gate-specific version of QT based SEU mitigation technique is also proposed which has half the area of the QT based SEU mitigation technique.

The rest of the chapter is organized as follows. In the next section, the QMR technique for SEU mitigation is described. After that, Quadded-Transistor based SEU mitigation technique is discussed followed by gate-specific QT based SEU mitigation technique. After that, experimental analysis is presented followed by a summary in the last section that concludes this chapter.

4.2 Quadded Modular Redundancy Technique

In chapter 3, two transistor structures were proposed for tolerating permanent defects in the digital circuits. The same transistor structures can also be used for mitigating SEUs. Out of the two proposed structures, only quadded-transistor structure will be employed in our proposed work for SEU mitigation.

In the normal quadded-transistor (QT) technique as shown in Figure 4.1 for tolerating permanent defects, the gate terminals of all the four transistors representing a single transistor in the present level are fed by the same output line emanating from the logic gate in the previous level which is feeding the logic gate in the present level.

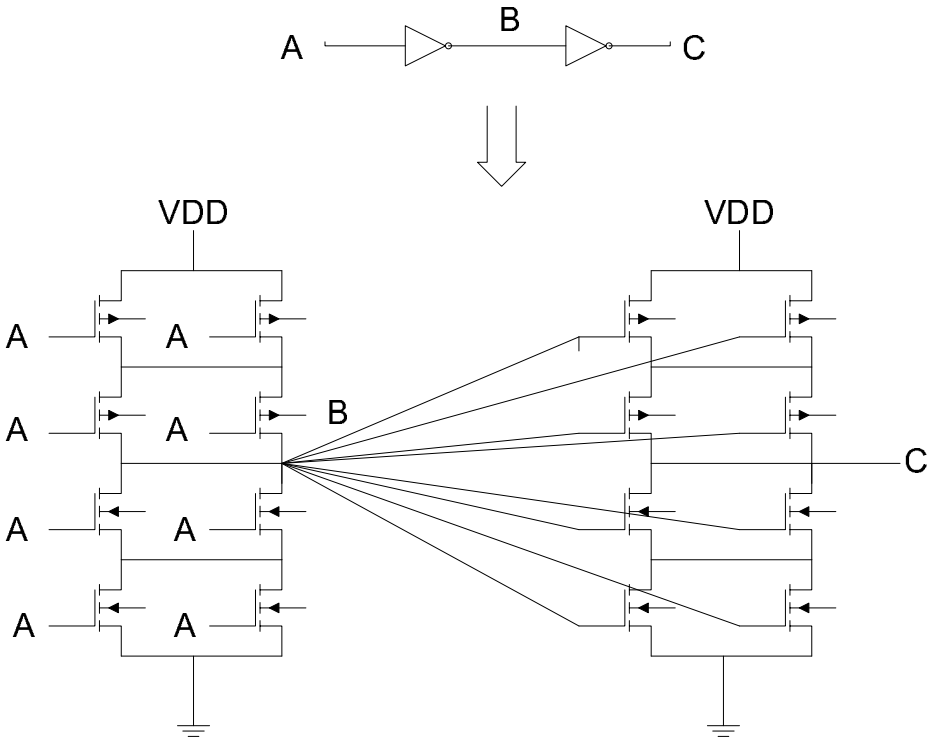

Figure 4.1: Quadded-Transistor based technique for permanent defects.

The normal quadded-transistor technique can be directly applied for SEU mitigation but one characteristic of the quadded-transistor technique as mentioned in the previous chapter is that the gate capacitance that the quadded-transistor structure induces on the gate connected to an input is four times the original gate capacitance. This has an impact on both delay and power dissipation. The main idea behind the proposed Quadded Modular Redundancy (QMR) technique is to selectively implement the quadded-transistor structure in some of the gates.

Quadded Modular Redundancy technique is based on making four copies of each logic module using the conventional CMOS transistor implementation and only selectively implementing the quadded-transistor structure in some of the gates which are referred to as restoring gates in QMR. The restoring gate selected

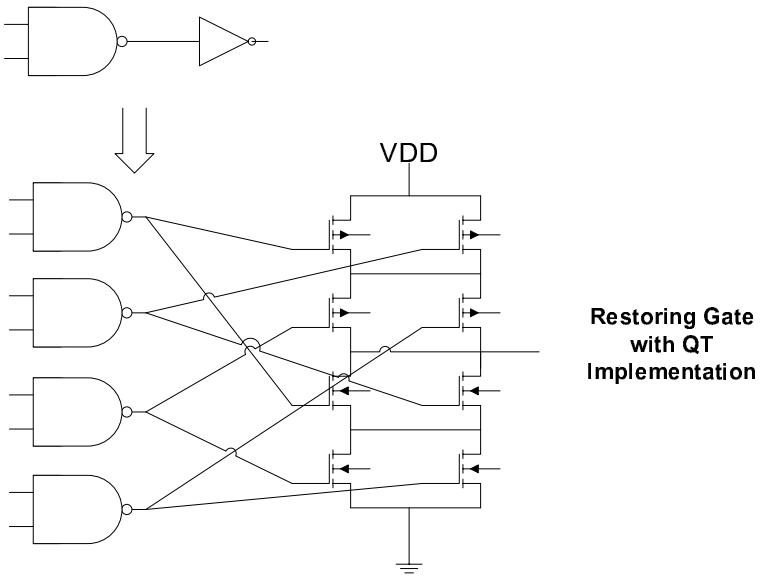

Figure 4.2: Quadded Modular Redundancy technique for a simple 2-input circuit.

for QT implementation receives a separate input to each of its input transistor in the QT structure. This approach is shown in Figure 4.2 for a simple 2-input circuit.

The advantage of this approach over the TMR approach is that it does not require any other restoring module like Majority voter (which is the case with the TMR) eliminating any single point of failure (SPF). The restoring gates are the ones which are implemented using QT structure. Another advantage of the QMR technique is that the resulting circuit will have normal gate capacitance because every transistor is taking input from a separate copy of the gate in the previous level and the circuit delay will not be affected as compared to QT implementation.

4.3 Quadded-Transistor based SEU Mitigation Technique (QT16)

In the proposed QT based SEU mitigation technique as shown in Figure 4.3, the idea is to implement all gates using quadded-transistor structure and also to replicate every gate four times in such a manner that the gate terminals of all the transistors belonging to a quadded structure in the present level are connected from a different copy of the gate in the previous level feeding the gate terminals of transistors in the present level.

The advantage of this technique is that this way permanent defects are taken care of by the quadded-transistor structure and the transient faults are taken care of by the four copies of every gate feeding a different input of the quadded-transistor structure in the next level.

4.4 Gate-specific Quadded-Transistor based SEU Mitigation Technique (QT8)

The technique proposed in the previous section requires high hardware overhead in terms of number of transistors in the sense that four copies of each gate are used and each gate is implemented using quadded-transistor structures. If the specific structures implementing different gates at the transistor levels are taken in account while connecting transistors together, it is possible to minimize the number of gates by half.

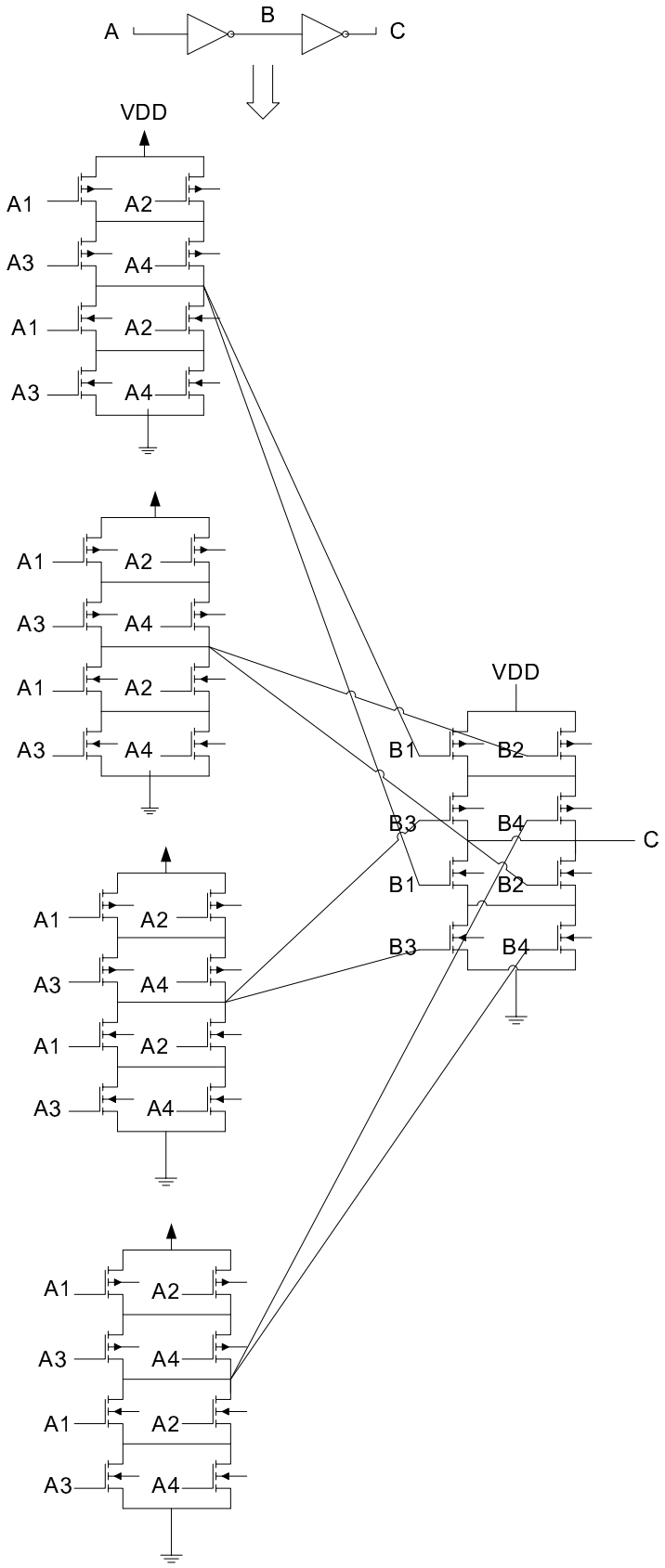

Figure 4.3: Quadded-Transistor based technique for SEU Mitigation.

The modified approach called gate-specific QT based SEU mitigation technique is based on making only two copies of every logic gate in the previous level and making the connections by considering the specific gate to which the connection is made.

The connections for NAND and AND gates are made as follows:

- The outputs from the the same copy of the gate in previous level are connected to the gate terminals of the two parallel PMOS transistors in the quadded-transistor structure in the present level.

- The outputs from the the same copy of the gate in previous level are connected to the gate terminals of the two series NMOS transistors in the quadded-transistor structure in the present level.

The above mentioned connection scheme for a simple NAND gate is shown in Figure 4.4.

The reason to do these connections is that if two NMOS transistors in series in the quadded-transitor structure become OFF when they should be ON, the other two non-defective ON transistors can mask the fault. Following the same line of reasoning, if two parallel PMOS transistors become ON when they should be OFF, the other two OFF transistors can mask the fault.

Another important reason to do these connections is that if two NMOS transistors in series in the quadded-transistor structure become ON when they should be OFF, the fault will not propagate unless all the other inputs in the NAND gate have a value of 1 when the fault occurs.

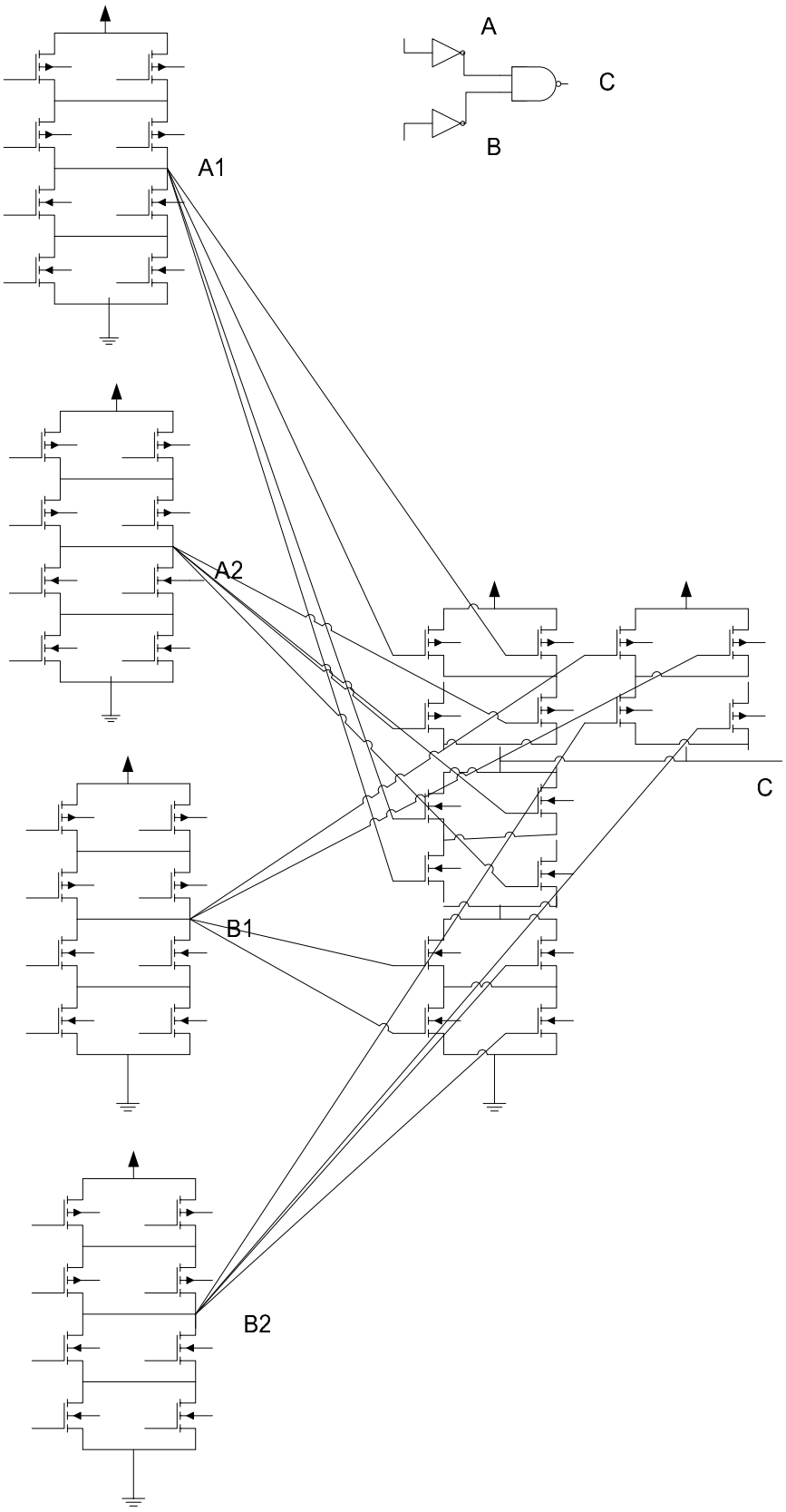

Figure 4.4: Gate-specific connections for NAND gate to mask faulty transistors.

The connections for NOR and OR gates are made as follows:

- The outputs from the the same copy of the gate in previous level are connected to the gate terminals of the two series PMOS transistors in the quadded structure in the present level.
- The outputs from the the same copy of the gate in previous level are connected to the gate terminals of the two parallel NMOS transistors in the quadded structure in the present level.

Following the same line of reasoning as discussed for NAND/AND gates, it is argued that the proposed connection scheme for NOR/OR gates will provide more fault masking. For all the other gate types, any connection scheme can be chosen.

The reliability analysis of all the above mentioned techniques is presented in the next section.

4.5 Experimental Results

In the following sections, the experimental results obtained using simulations for the above mentioned three techniques for SEU mitigation are presented.

4.5.1 Quadded Modular Redundancy Technique Analysis

To demonstrate the effectiveness of the QMR technique for SEU mitigation, experiments have been performed on a number of largest ISCAS85 and ISCAS89

benchmark circuits.

Two types of faults are considered: a transient fault affecting the gate terminal of a transistor erroneously switching it OFF (equivalent of a stuck-open fault) and a transient fault affecting the gate terminal of a transistor effectively switching it ON (equivalent of a stuck-short fault).

The comparison of circuit reliability of QMR technique with the TMR technique is performed using the simulation-based methodology presented in [11] but with random test vectors instead of a complete test set. This is done to simulate the random and transitory nature of the transient faults.

To compute the circuit failure probability, F and reliability R, resulting from injecting m faults in the transistors, the following simulation procedure is used:

1. Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.

2. Generate a random test vector.

3. Simulate the fault-free circuit by applying the random test vector T.

4. Randomly inject m transient faults in transistors.

5. Simulate the faulty circuit by applying the same random test vector T.

6. If the outputs of the fault-free and faulty circuits are different, increment K by 1.

7. Decrement I by 1 and if I is not 0 goto step 2.

8. Failure Probability $F = K/1000$.

9. Reliability $R = 1 - F$.

Using the aforementioned simulation model, a number of experiments were carried out on multistage complementary half adder circuits and a number of largest ISCAS85 and ISCAS89 benchmark circuits.

Comparison of circuit failure probability using QMR and TMR techniques for multistage adders

In Figures 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12, a comparison of circuit failure probability for QMR and TMR techniques for different percentages of faults is shown for 1, 2, 4, 8, 16 and 32-stage complementary half-adder circuits. TMR is implemented at module level i.e., the sum and carry outputs of every stage are being voted upon as shown in Figure 4.5 for a single stage. Similarly QMR is also implemented in sum and carry outputs of every stage with every half-adder stage looking as shown in Figure 4.6. It should be noted that the QMR implementation of a single complementary half-adder stage involves implementing only CS and CC NAND gates using quadded-transistor structure.

It is clear from Figures 4.7 to 4.12 that the QMR approach gives 25% to 50% less circuit failure probability than the TMR approach for all the small percentages of faults normally projected for CMOS process.

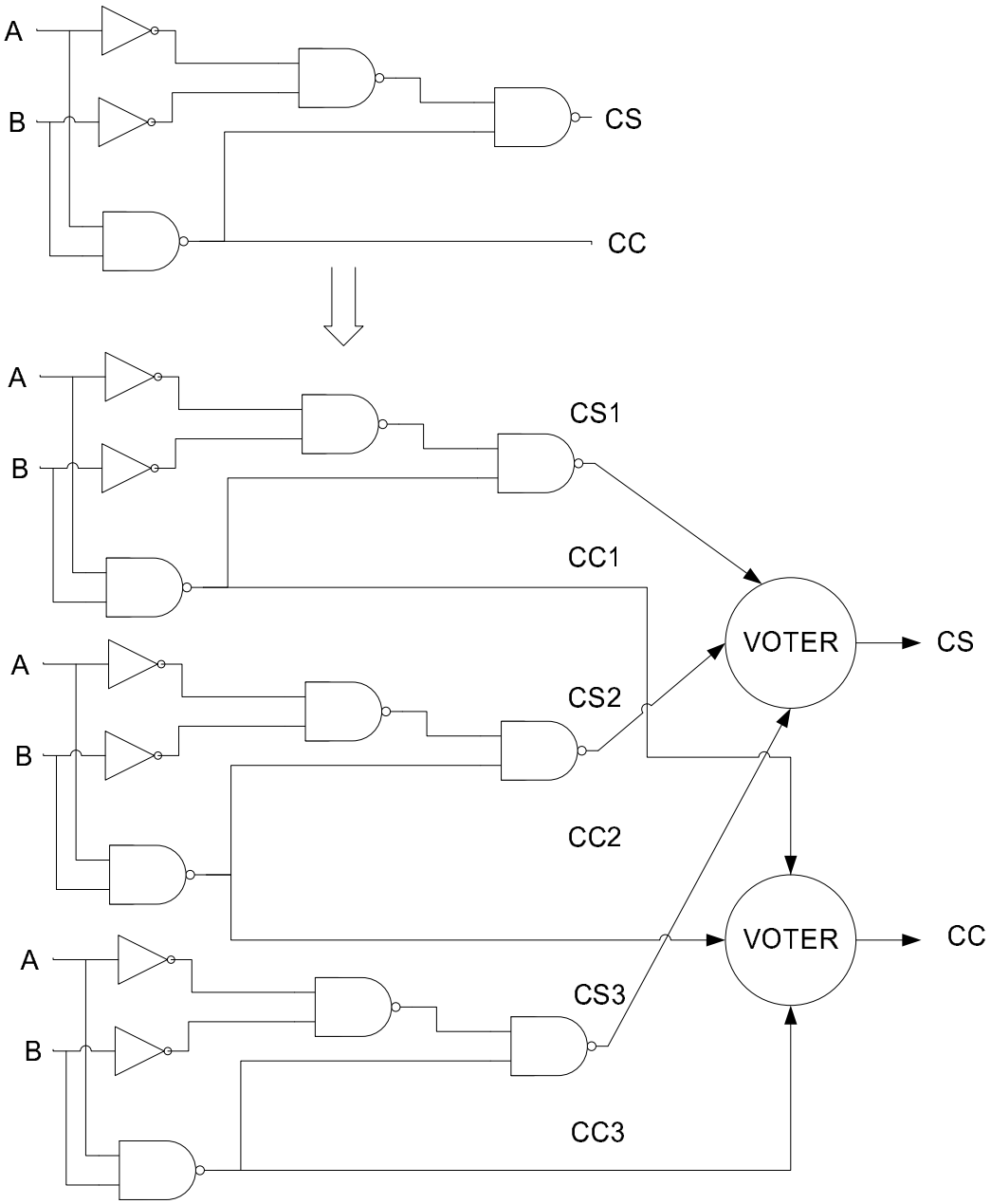

Figure 4.5: Triple Modular Redundancy technique for single stage of 2-input complementary half adder.

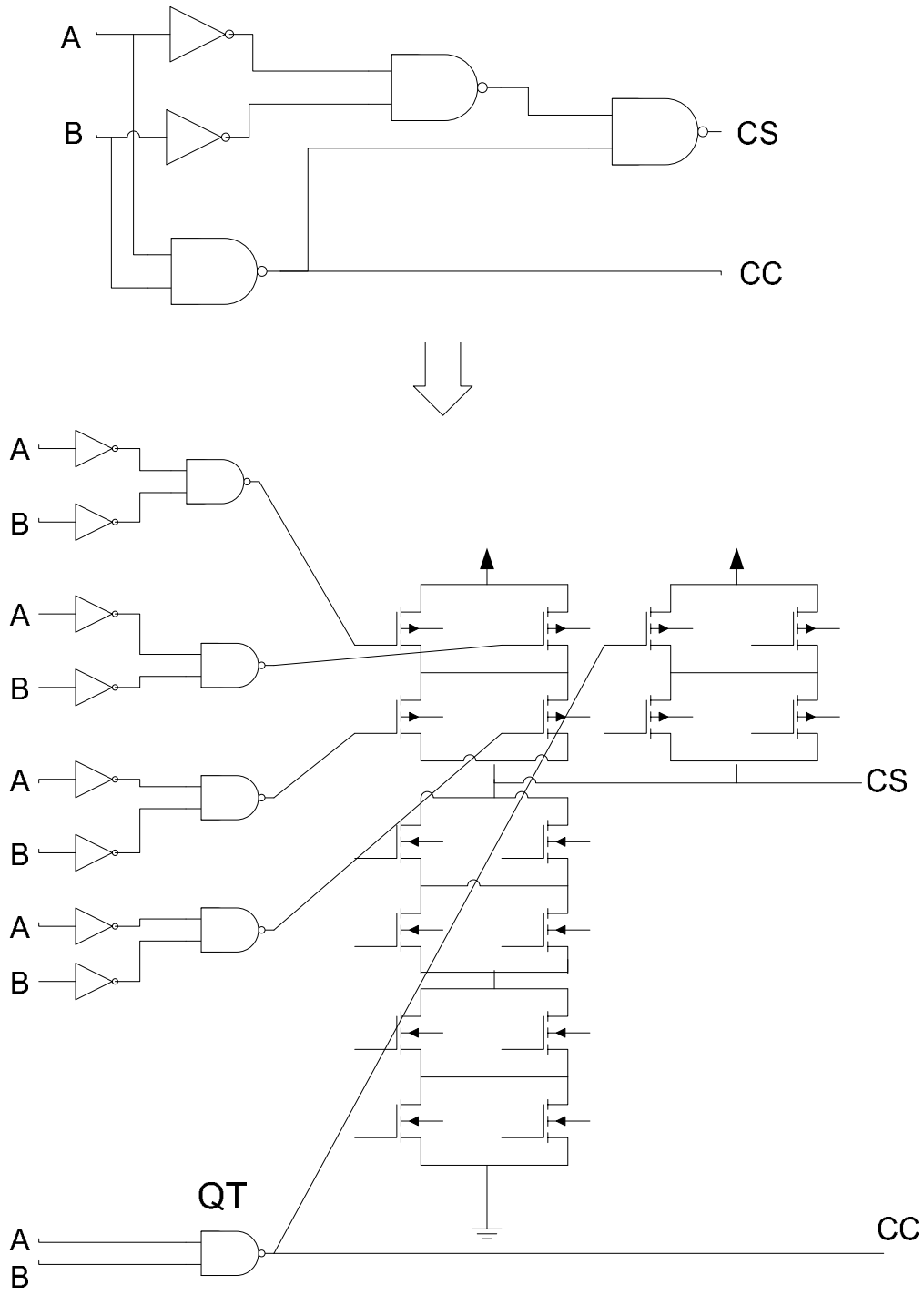

Figure 4.6: Quadded Modular Redundancy technique for single stage of 2-input complementary half adder.

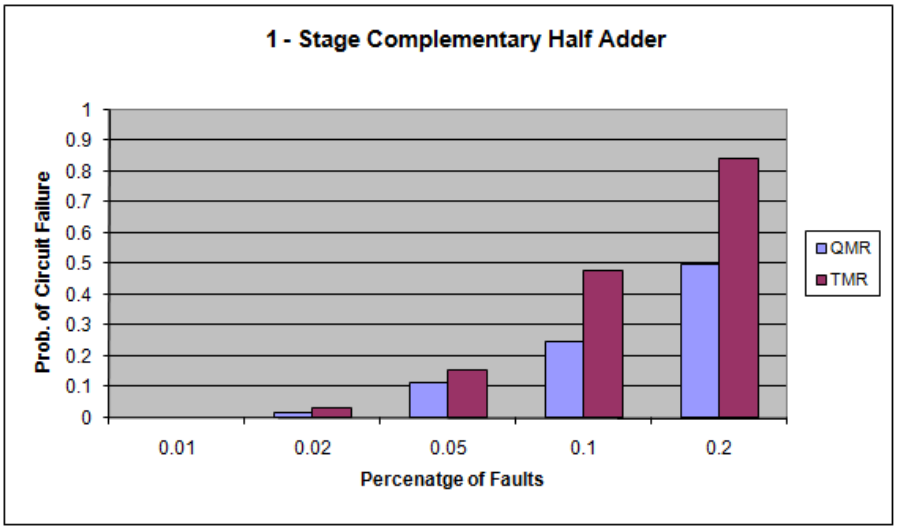

Figure 4.7: Comparison of circuit failure probability for a 1-stage complementary half-adder circuit for transient faults.

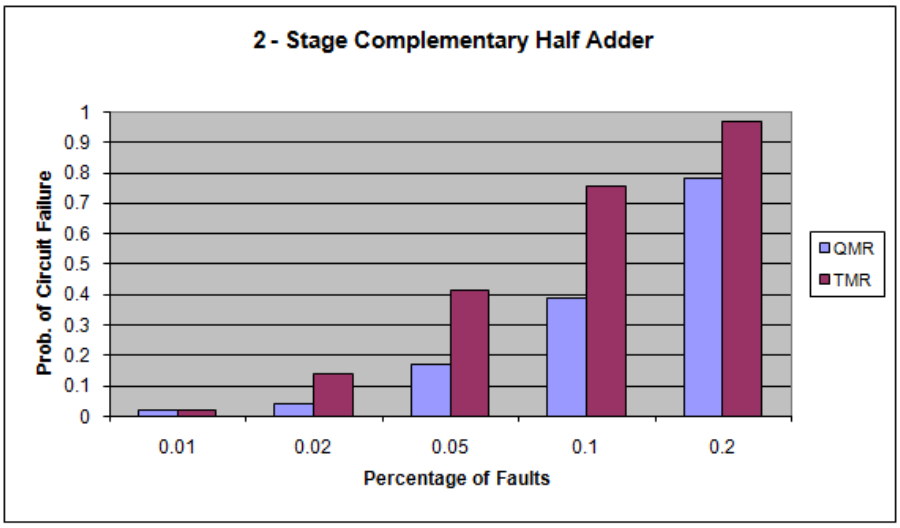

Figure 4.8: Comparison of circuit failure probability for a 2-stage cascaded complementary half-adder circuit for transient faults.

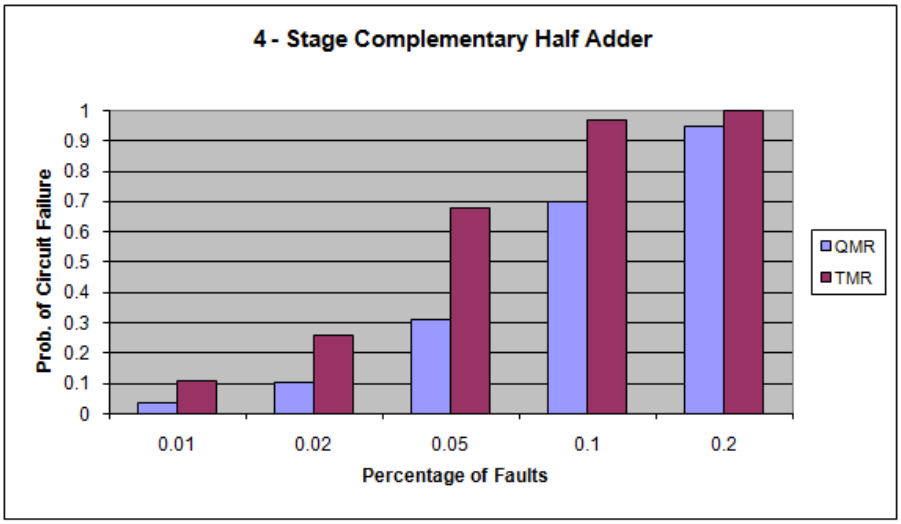

Figure 4.9: Comparison of circuit failure probability for a 4-stage cascaded complementary half-adder circuit for transient faults.

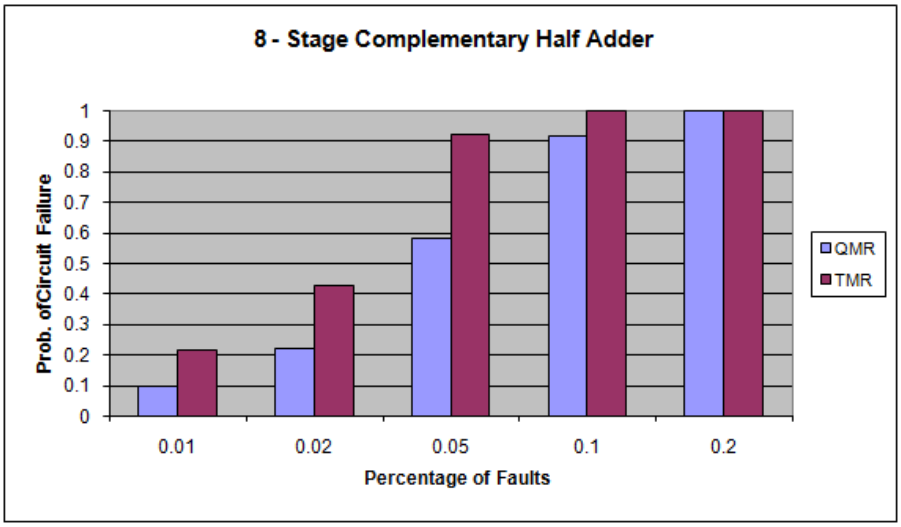

Figure 4.10: Comparison of circuit failure probability for a 8-stage cascaded complementary half-adder circuit for transient faults.

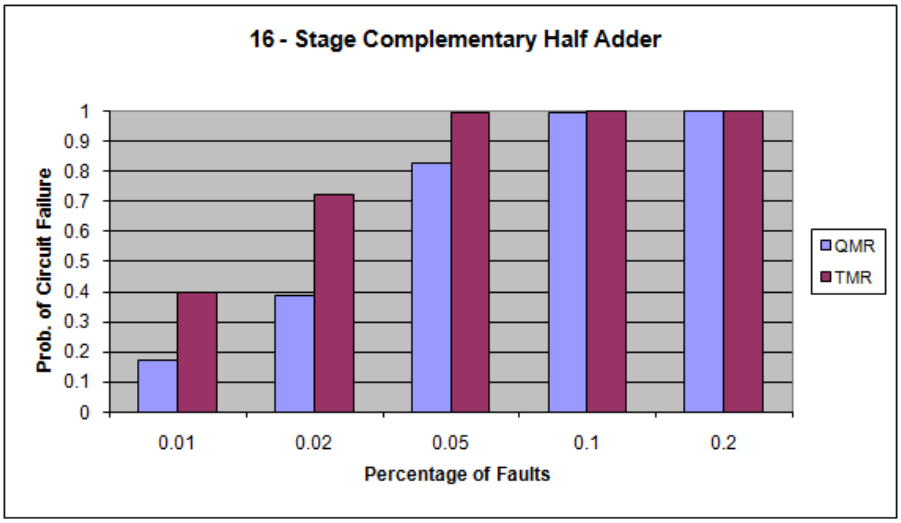

Figure 4.11: Comparison of circuit failure probability for a 16-stage cascaded complementary half-adder circuit for transient faults.

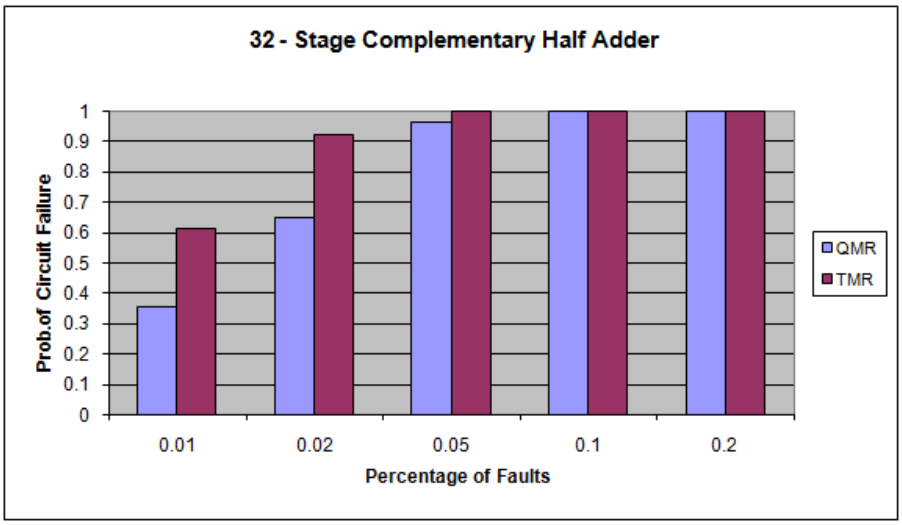

Figure 4.12: Comparison of circuit failure probability for a 32-stage cascaded complementary half-adder circuit for transient faults.

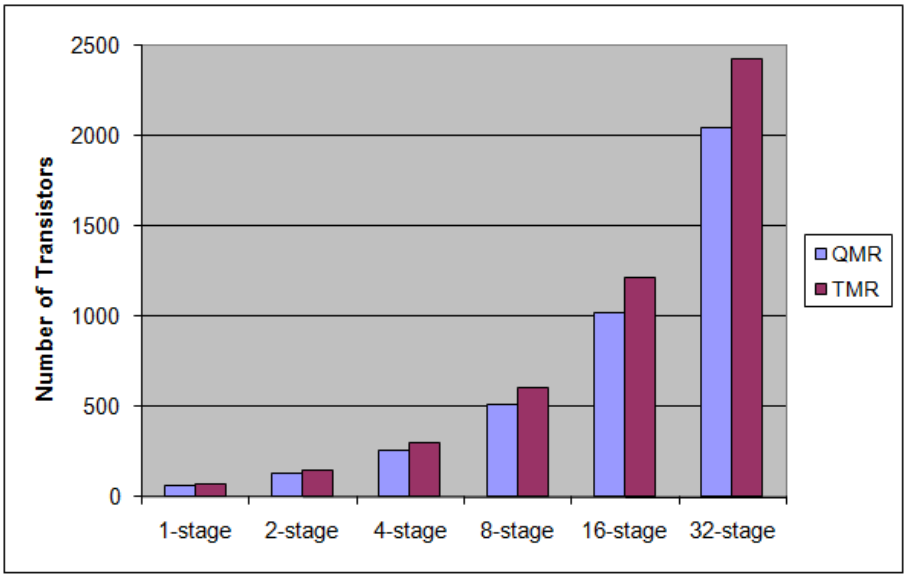

Figure 4.13: Comparison of area in terms of number of transistors for 1, 2, 4, 8, 16 and 32-stage cascaded complementary half adders for QMR and TMR implementation.

Comparison of circuit area using QMR and TMR techniques for multistage adders

Figure 4.13 compares the area in terms of number of transistors for the 1, 2, 4, 8, 16 and 32-stage cascaded complementary half adder circuits. It is observed that the QMR implementation requires 20% less area than the TMR implementation.

Comparison of circuit reliability using QMR and TMR techniques for ISCAS benchmarks

Using the same simulation procedure, the circuit reliability analysis was carried out for some ISCAS85 and ISCAS89 benchmark circuits.

Since in the TMR technique, the module size affects the circuit reliability, therefore, the reliability results were obtained using different module sizes of 1, 3, 5, 7 and 9 for QMR and TMR techniques. For example in TMR, module size of 3 means that a module consisting of a maximum of 3 gates is triplicated and a majority voter is used to vote upon the three outputs from the three modules. Similarly, in QMR, module size 3 means that in a module consisting of a maximum of 3 gates, all the gates are replicated four times except the restoring gate to which the four copies of the other gates feed their outputs and which provides the output of the module. The restoring gate uses the quadded-transistor structure in its implementation.

In order to do the comparison of QMR and TMR with different module sizes, an algorithm reported in [99] for TMR has been adopted. The modular TMR algorithm combines gates for a given module size and puts majority voter at appropriate places in the TMR. It is expalianed in the following section.

Modular TMR Algorithm

Initially, the algorithm defines an array called "unprocessed" that will contain the primary outputs. Starting from the outputs of the circuit, gates are added to

construct a module until one of the following cases occur: reaching the inputs of the circuit, reaching a fanout point, or reaching the module size required. The constructed module will then be triplicated and a majority voter will be inserted for these three copies. All remaining gates where the algorithm stopped at for this iteration will be added to the "unprocessed" array without replicating nodes that already existed. Next, the algorithm will start processing the next node in the unprocessed array doing same as explained in the previous step. This will continue until all nodes in the "unprocessed" array are processed.

Following data structures are defined for the modular TMR algorithm.

- "unprocessed" array that will hold initially all circuit output nodes
- "unprocessed-inputs" array that will hold the input nodes for the currently processed gate
- *Module-Size* variable that will hold the module size required for TMR module construction
- *Level* variable which is the current module size reached in the current iteration

The modular TMR algorithm is as follows:

A Put all outputs in the "unprocessed" array.

B For every node in the "unprocessed" array.

1. Initialize *Level* to 0.

2. Get the gate which outputs the current node.

3. Triplicate the retrieved gate and increment *Level*.

4. Add a Majority voter.

5. Get the inputs of the current processed gate and put them in "unprocessed-inputs" array.

6. For every node in the "unprocessed-inputs" array.

   while(*Level* < *Module-Size* and the current node is Not a Primary Input or a Fanout)

   (a) Triplicate the current gate and increment *Level*.

   (b) Get the inputs of the current processed gate and add them to the "unprocessed-inputs" array.

7. Add the remaining nodes which are still unprocessed from "unprocessed-inputs" array to the "unprocessed" array.

Application of the Modular TMR algorithm for a simple logic circuit of Figure 4.14 is shown in Figures 4.15, 4.16 and 4.17 for modules sizes of 1, 2 and 3 respectively.

For QMR, the modular TMR algorithm is modified in order to convert only the restoring gates of a module to quadded-transistor implementation. The modified algorithm is as follows:

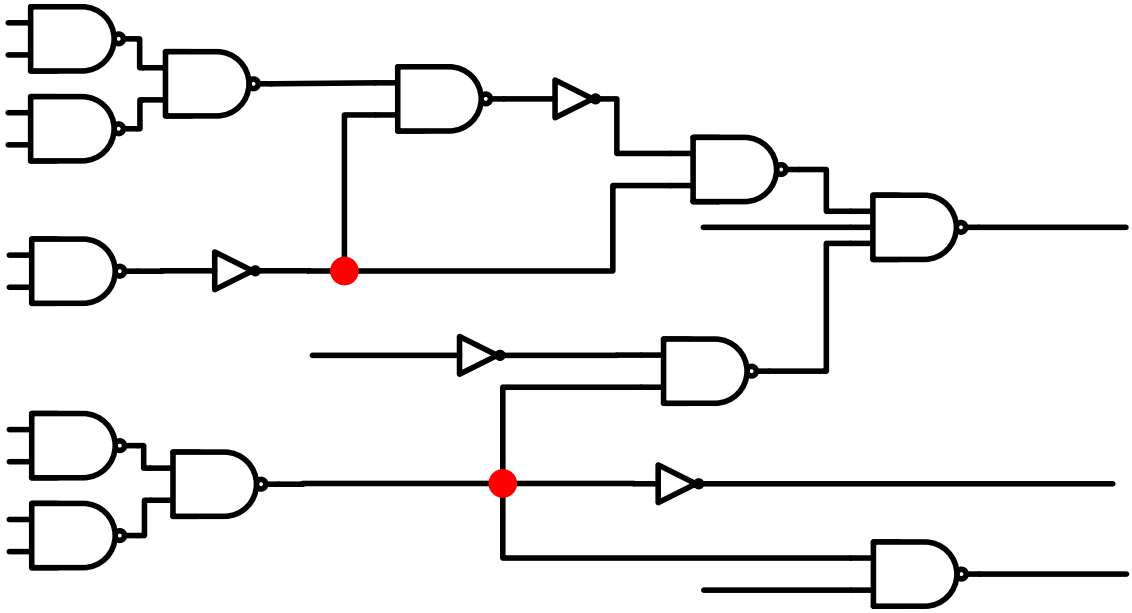

Figure 4.14: Example Circuit.

Modular QMR Algorithm

Initially, the algorithm defines an array called "unprocessed" that will contain the primary outputs. Starting from the outputs of the circuit, gates are added to construct a module until one of the following cases occur: reaching the inputs of the circuit, reaching a fanout point, or reaching the module size required. All the gates in the constructed module will then be replicated four times expect the restoring gate of the module which provides output to other modules. Only the restoring gate will be converted to quadded-transistor implementation. All remaining gates where the algorithm stopped at for this iteration will be added to the "unprocessed" array without replicating nodes that are already existed. Next, the algorithm will start processing the next node in the unprocessed array doing same as explained in the previous step. This will continue until all nodes in the

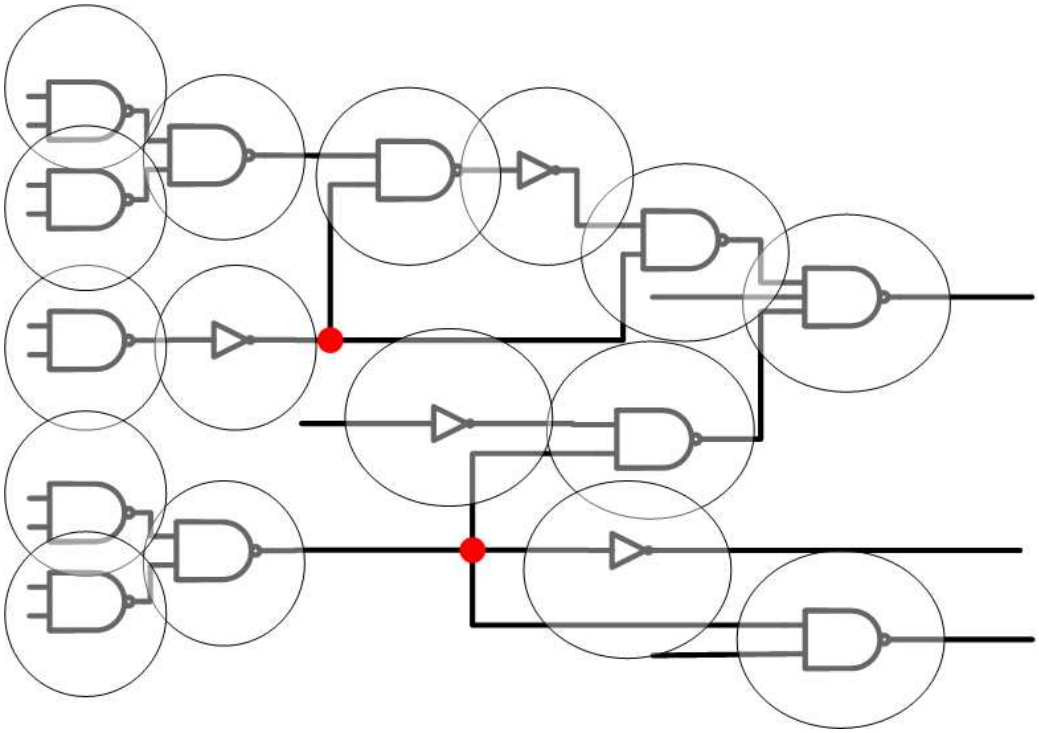

Figure 4.15: Application of modular TMR algorithm on example circuit for a module size of 1.

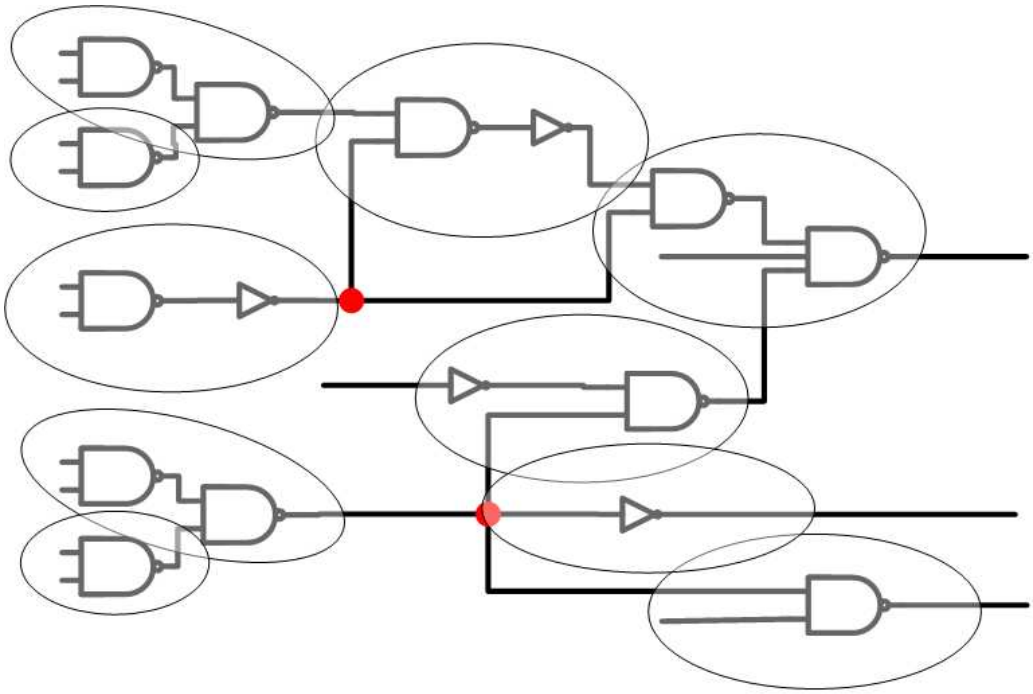

Figure 4.16: Application of modular TMR algorithm on example circuit for a module size of 2.

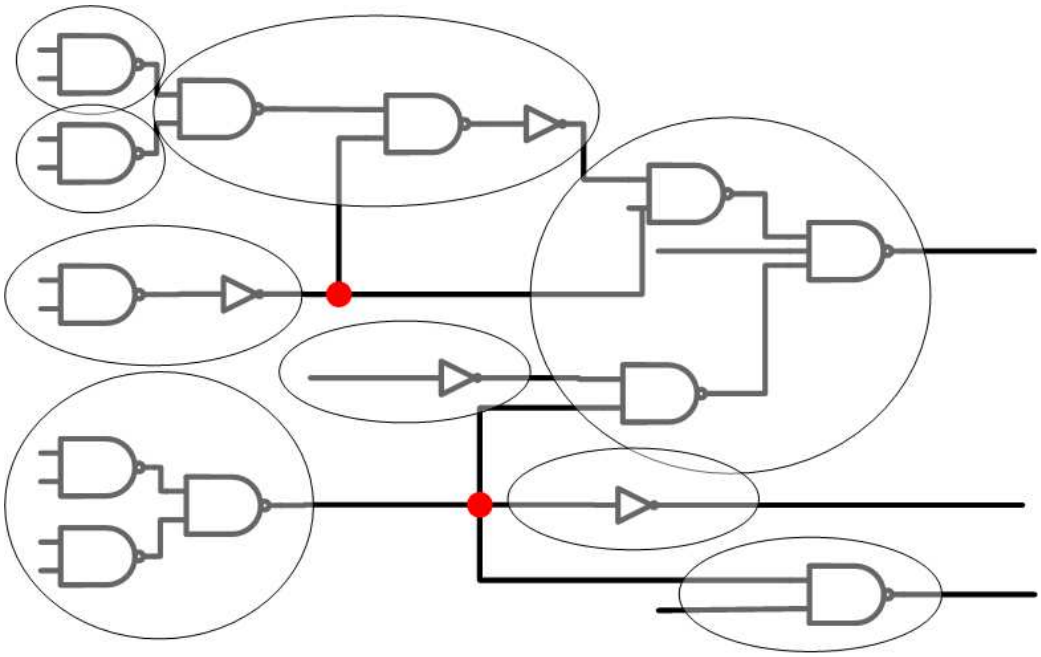

Figure 4.17: Application of modular TMR algorithm on example circuit for a module size of 3.

"unprocessed" array are processed.

Following data structures are defined for the modular QMR algorithm.

- "unprocessed" array that will hold initially all circuit output nodes
- "unprocessed-inputs" array that will hold the input nodes for the currently processed gate
- *Module-Size* variable that will hold that module size required for QMR module construction
- *Level* variable which is the current module size reached in the current iteration

The modular QMR algorithm is as follows:

A Put all outputs in the "unprocessed" array.

B For every node in the "unprocessed" array.

1. Initialize *Level* to 0.
2. Get the gate which outputs the current node. This is the restoring gate of the module.
3. Convert the retrieved restoring gate to QT implementation and increment *Level*.
4. Get the inputs of the current processed gate and put them in "unprocessed-inputs" array.
5. For every node in the "unprocessed-inputs" array.

while(*Level* < *Module-Size* and the current node is Not a Primary Input or a Fanout)

(a) Replicate the current gate four times and increment *Level*.

(b) Get the inputs of the current processed gate and add them to the "unprocessed-inputs" array.

6. Add the remaining nodes which are still unprocessed from "unprocessed-inputs" array to the "unprocessed" array.

Application of the Modular QMR algorithm for a simple logic circuit of Figure 4.14 is shown in Figures 4.18, 4.19 and 4.20 for modules sizes of 1, 2 and 3 respectively. The gates which are marked with **Q** are the ones which will be implemented with quadded-transistor structure.

One interesting observation after analyzing Figure 4.18 is that if module size of 1 is chosen, all the gates of QMR implementation will be implemented with quadded-transistor structure. Therefore, it can be said that the normal QT implementation is a special case of QMR technique with a module size of 1. Alternatively, it can be said that the QMR technique is a generalization of normal QT implementation with selective application of QT structure.

Circuit reliability of the QMR and TMR techniques for some ISCAS85 and ISCAS89 benchmark circuits for different percentages of transient faults and for module sizes of 1, 3, 5, 7 and 9 are compared in Tables 4.1, 4.2, 4.3, 4.4 and 4.5.

As shown in Tables 4.1 to 4.5, the circuit reliability of modular QMR is better than that of modular TMR for all the percentages of injected faults and module

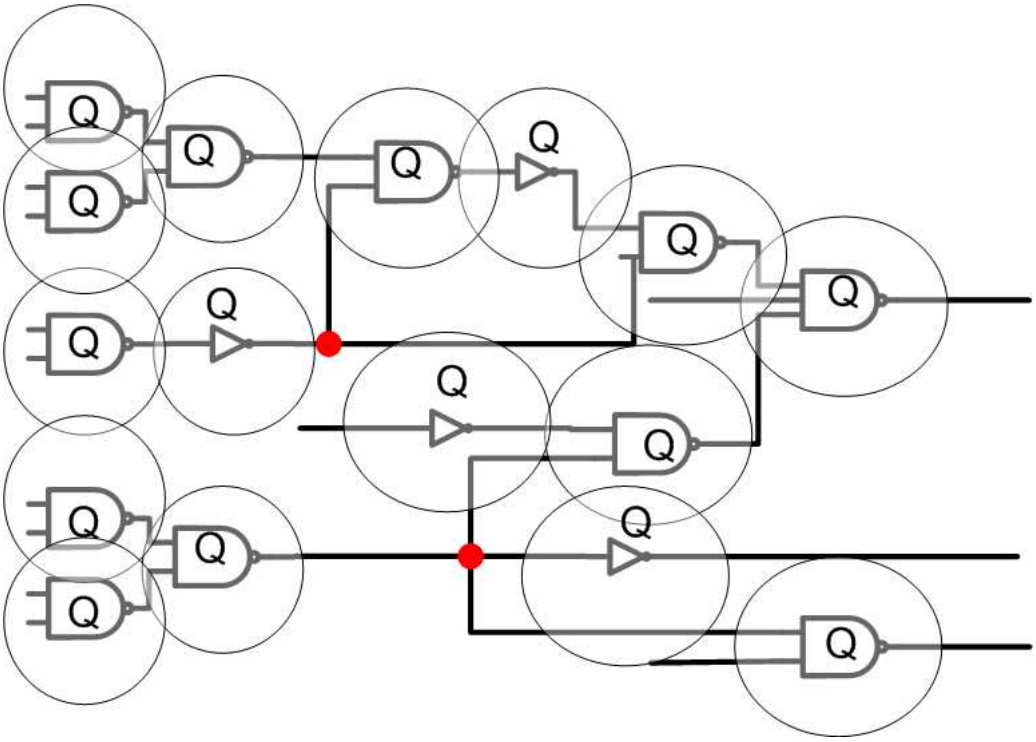

Figure 4.18: Application of modular TMR algorithm on example circuit for a module size of 1.

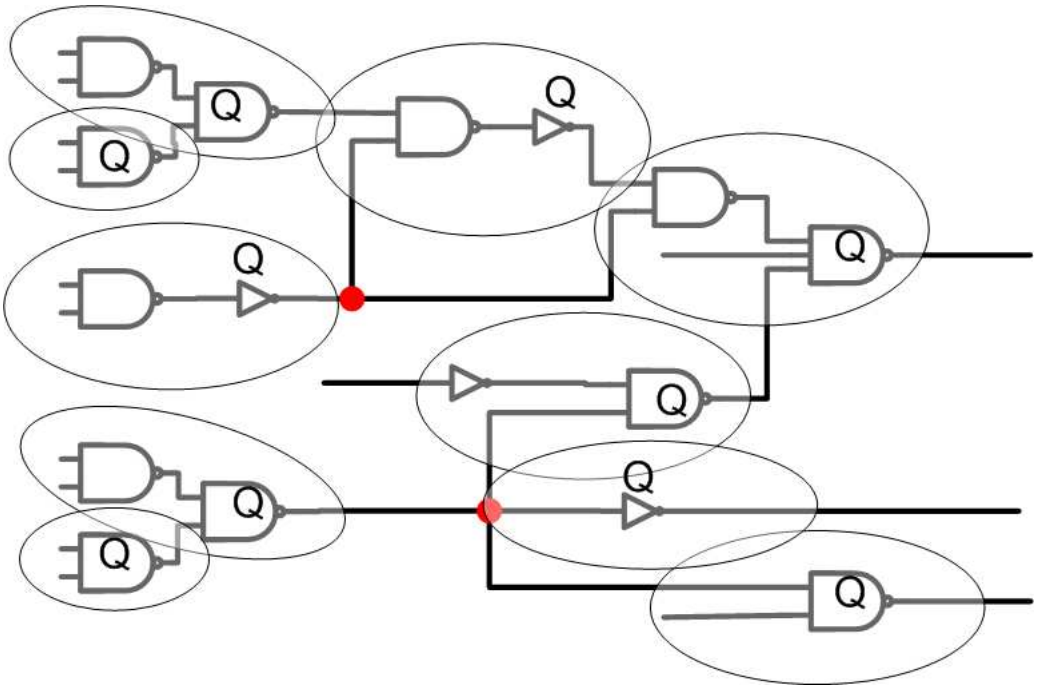

Figure 4.19: Application of modular TMR algorithm on example circuit for a module size of 2.

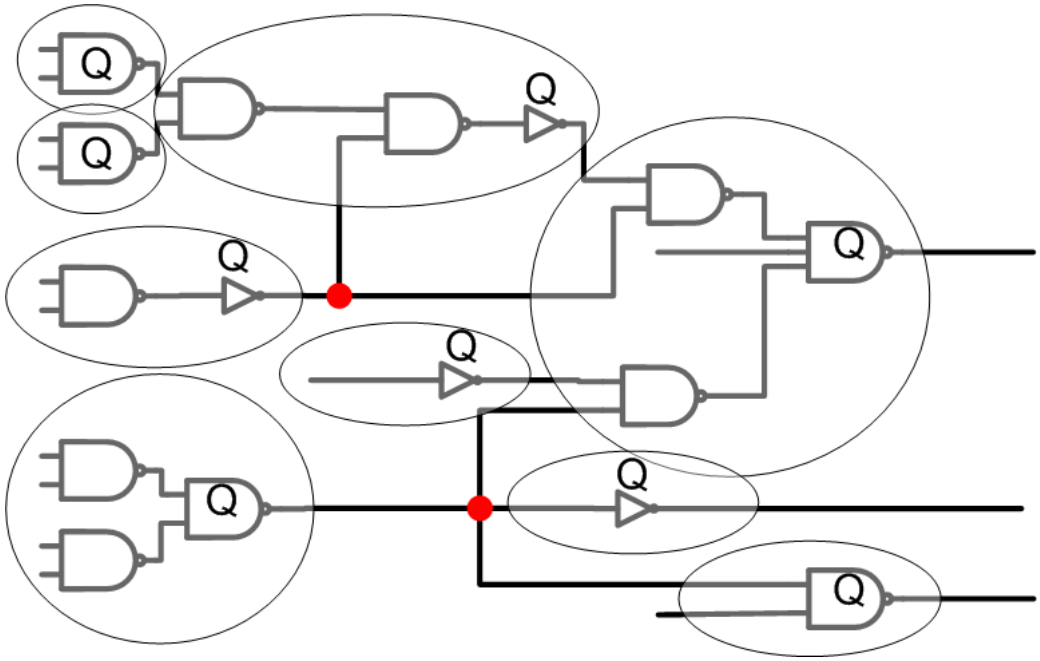

Figure 4.20: Application of modular TMR algorithm on example circuit for a module size of 3.

Table 4.1: Comparison of circuit reliability between QMR and TMR techniques for a module size of 1 (i.e., full QT implementation).

	QMR					TMR				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	7208	1	0.998	0.99	0.943	10768	0.977	0.79	0.307	0.081
c1355	9232	1	1	0.978	0.939	14568	0.972	0.764	0.284	0.051
c1908	13784	1	0.999	0.972	0.928	22658	0.957	0.667	0.149	0.038
c3540	30016	1	1	0.978	0.873	45878	0.953	0.652	0.08	0.007
c5315	45048	1	0.999	0.95	0.833	66084	0.921	0.447	0.015	0
c6288	40448	1	0.997	0.908	0.630	64160	0.826	0.09	0	0
s5378	35608	1	0.998	0.948	0.775	66222	0.879	0.215	0	0
s9234	74856	1	0.998	0.914	0.726	134500	0.792	0.067	0	0

Table 4.2: Comparison of circuit reliability between QMR and TMR techniques for a module size of 3.

	QMR					TMR				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	7208	1	0.997	0.912	0.731	8374	1	0.897	0.494	0.156
c1355	9232	1	0.999	0.929	0.739	11264	0.979	0.812	0.302	0.091
c1908	13784	1	0.998	0.918	0.723	18836	0.981	0.746	0.23	0.077
c3540	30016	1	0.994	0.893	0.624	37072	0.973	0.689	0.102	0.004
c5315	45048	1	0.987	0.748	0.362	54828	0.958	0.563	0.048	0
c6288	40448	1	0.98	0.664	0.19	50720	0.845	0.189	0	0
s5378	35608	1	0.986	0.655	0.206	47614	0.95	0.453	0.007	0
s9234	74856	1	0.951	0.364	0.026	92990	0.9	0.264	0	0

sizes for all the ISCAS benchmarks. In the next section, the impact of module size on the circuit reliability is discussed.

Impact of module size on circuit reliability in QMR and TMR Techniques

In Figure 4.21, circuit reliability of QMR version of c880 benchmark circuit is compared with that of the TMR version of c880 benchmark for module sizes of 1, 3, 5, 7 and 9.

An interesting observation after analyzing Figure 4.21 and the circuit reliability

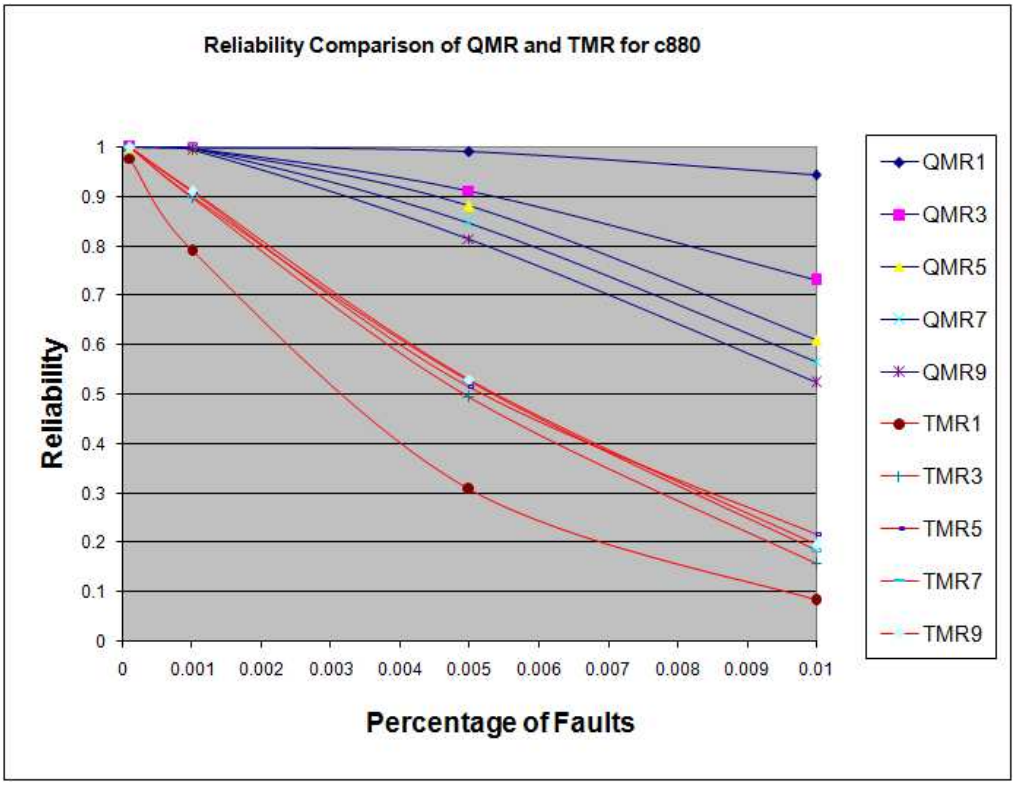

Figure 4.21: Comparison of circuit reliability for QMR and TMR techniques module sizes of 1, 3, 5, 7 and 9.

Table 4.3: Comparison of circuit reliability between QMR and TMR techniques for a module size of 5.

	QMR					TMR				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	7208	1	0.997	0.881	0.609	7562	1	0.908	0.514	0.261
c1355	9232	1	0.99	0.924	0.699	10872	0.978	0.831	0.3	0.076
c1908	13784	1	0.995	0.86	0.601	16316	0.983	0.746	0.232	0.063
c3540	30016	1	0.989	0.834	0.475	34734	0.966	0.691	0.106	0.007
c5315	45048	1	0.98	0.607	0.185	47814	0.954	0.64	0.062	0.002
c6288	40448	1	0.989	0.647	0.204	50720	0.851	0.195	0	0
s5378	35608	1	0.978	0.499	0.074	43840	0.95	0.51	0.021	0
s9234	74856	1	0.928	0.205	0.001	84086	0.924	0.352	0.004	0

Table 4.4: Comparison of circuit reliability between QMR and TMR techniques for a module size of 7.

	QMR					TMR				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	7208	1	0.996	0.846	0.566	7632	1	0.899	0.525	0.182
c1355	9232	1	0.998	0.921	0.698	10816	0.979	0.803	0.298	0.07
c1908	13784	1	0.997	0.846	0.582	15812	0.979	0.733	0.24	0.058
c3540	30016	1	0.985	0.787	0.432	34090	0.981	0.687	0.111	0.004
c5315	45048	1	0.986	0.591	0.178	46772	0.971	0.608	0.055	0
c6288	40448	1	0.982	0.634	0.206	50720	0.848	0.17	0	0
s5378	35608	1	0.955	0.441	0.051	42222	0.95	0.542	0.016	0
s9234	74856	1	0.894	0.073	0	80460	0.92	0.426	0.001	0

reported in Tables 4.1 to 4.5 is that for the QMR approach, there is no change in number of transistors with varying module sizes. The impact of module size is the increase in number of gates which are implemented using quadded-transistor structure, hence smaller modular size results in better reliability. Noting this, it can be claimed that for QMR, smaller module size will be better because more gates will be implemented with the QT structure and hence will provide more transistor-level defect-tolerance.

For TMR, the module has a significant impact on the reliability. The reason is that with smaller module size, there are more majority voters in the circuit

Table 4.5: Comparison of circuit reliability between QMR and TMR techniques for a module size of 9.

	QMR					TMR				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	7208	1	0.993	0.815	0.523	7450	1	0.91	0.53	0.195
c1355	9232	1	0.997	0.904	0.7	10760	0.981	0.834	0.308	0.1
c1908	13784	1	0.997	0.845	0.568	15784	0.99	0.776	0.236	0.073
c3540	30016	1	0.996	0.773	0.386	33502	0.961	0.693	0.114	0.009
c5315	45048	1	0.977	0.576	0.158	46036	0.967	0.656	0.0659	0.002
c6288	40448	1	0.987	0.645	0.174	50720	0.842	0.183	0	0
s5378	35608	1	0.961	0.414	0.037	41480	0.94	0.527	0.027	0
s9234	74856	0.999	0.882	0.053	0	78598	0.948	0.452	0.006	0

thereby increasing the probability that a fault will affect a majority gate effectively resulting in a failure for the whole circuit. Increasing module size results in fewer majority voters and hence improved circuit reliability. At the same time, increasing module size will not result always in higher circuit reliability because a larger module size also increases the possibility of failure of two copies in a triplicated module thereby reducing circuit reliability. Therefore, in TMR, there is a trade-off between module size and the circuit reliability which needs to be considered while choosing a module size for a particular circuit.

As shown in Tables 4.1 to 4.5, the QMR technique for SEU mitigation outperforms TMR technique in terms of both circuit reliability and circuit area (measured in terms of number of transistors). This justifies the use of quadded-transistor based technique not only for masking permanent defects but also for transient faults.

4.5.2 Quadded-Transistor based SEU Mitigation Technique Analysis (QT16)

To demonstrate the effectiveness of the above mentioned quadded-transistor based technique for SEU mitigation, experiments have been performed on a number of largest ISCAS85 and ISCAS89 benchmark circuits. The same transistor-level transient fault model was used for reliability analysis which was used for the QMR and TMR techniques.

To compute the circuit failure probability, F and reliability R, resulting from injecting m faults in the transistors, the same simulation-based procedure as described in the previous section for QMR and TMR is used.

Circuit reliability of the QT16 and TMR9 are compared in Table 4.6 for different percentages of injected faults. The comparison of circuit reliability for quadded-transistor based technique with the TMR using a module size of 9 is performed because it was the best module size for TMR in the previous section.

As shown in Table 4.6, the quadded-transistor based technique for SEU mitigation outperforms TMR technique in terms of circuit reliability but the area overhead in terms of number of transistors is very high. In fact, area is 4 times of the normal QT implementation and 16 times of the original non-redundant circuit. Another characteristic of the QT16 technique is that it will have higher circuit delay because of the higher gate capacitance induced by QT structures so the resulting circuit will be more defect-tolerant but slower than the non-redundant circuit.

Table 4.6: Comparison of circuit reliability between quadded-transistor based technique and TMR9 technique for SEU mitigation.

	QT16					TMR9				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	28832	1	0.998	0.99	0.983	7450	1	0.91	0.53	0.195
c1355	36928	1	1	0.994	0.972	10760	0.981	0.834	0.308	0.1
c1908	55136	0.986	0.803	0.393	0.165	15784	0.99	0.776	0.236	0.073
c3540	120064	0.977	0.873	0.489	0.257	33502	0.961	0.693	0.114	0.009
c5315	180192	0.983	0.859	0.422	0.184	46036	0.967	0.656	0.0659	0.002
c6288	161792	1	0.997	0.995	0.987	50720	0.842	0.183	0	0
s5378	142432	1	0.999	0.98	0.939	41480	0.94	0.527	0.027	0
s9234	299424	1	1	0.975	0.902	78598	0.948	0.452	0.006	0

If the circuit reliability of QT16 and QMR with a module size of 1 is compared, it is observed that QT16 offers less reliability for the given percentages of injected faults. This is due to the 4 times increase in area which has negative impact on reliability as the injected faults are the percentage of the number of transistors in a circuit.

4.5.3 Gate-specific Quadded-Transistor based SEU Mitigation Technique Analysis (QT8)

To demonstrate the effectiveness of the gate-specific quadded-transistor based technique for SEU mitigation, experiments have been performed on a number of largest ISCAS85 and ISCAS89 benchmark circuits using the same transistor-level fault model and simulation procedure used for QMR and TMR analysis.

Circuit reliability of the gate-specific quadded-transistor based approach and TMR9 approach are compared in Table 4.7 for different percentages of transient faults.

Table 4.7: Comparison of circuit reliability between gate-specific quadded-transistor based technique and TMR9 technique for SEU mitigation.

	QT8					TMR9				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	14416	1	1	0.99	0.944	7450	1	0.91	0.53	0.195
c1355	18464	1	1	0.987	0.945	10760	0.981	0.834	0.308	0.1
c1908	27568	1	1	0.974	0.929	15784	0.99	0.776	0.236	0.073
c3540	60032	1	0.999	0.958	0.85	33502	0.961	0.693	0.114	0.009
c5315	90096	1	0.998	0.933	0.757	46036	0.967	0.656	0.0659	0.002
c6288	80896	1	0.957	0.797	0.568	50720	0.842	0.183	0	0
s5378	71216	1	0.998	0.95	0.84	41480	0.94	0.527	0.027	0
s9234	149712	1	0.995	0.847	0.452	78598	0.948	0.452	0.006	0

As shown in Table 4.7, the gate-specific quadded-transistor based technique for SEU mitigation gives us better circuit reliability as compared to the TMR technique even with 10 times higher percentage of injected faults. This clearly shows that the gate-specific quadded-transistor based technique is more efficient than the TMR technique for mitigating SEUs but the area in terms of number of transistors is still higher than TMR. In fact, area is 2 times of the normal QT implementation and 8 times of the original non-redundant circuit. Another characteristic of the QT8 technique is that it will have higher circuit delay because of the higher gate capacitance induced by QT structures so the resulting circuit will be more defect-tolerant but slower than the non-redundant circuit.

4.5.4 Reversed Gate-specific Quadded-Transistor based SEU Mitigation Technique Analysis (QT8R)

To analyze the effectiveness of the proposed gate-specific connections, experiments were also carried out with the reversed connections as well i.e., the connections

Table 4.8: Comparison of circuit reliability between reversed gate-specific quadded-transistor based technique and QT8 technique for SEU mitigation.

	QT8R					QT8				
Cct.	Trans.	0.01%	0.1%	0.5%	1%	Trans.	0.01%	0.1%	0.5%	1%
c880	14416	1	1	0.98	0.939	14416	1	1	0.99	0.944
c1355	18464	1	0.998	0.983	0.93	18464	1	1	0.987	0.945
c1908	27568	1	0.999	0.968	0.89	27568	1	1	0.974	0.929
c3540	60032	1	0.995	0.94	0.835	60032	1	0.999	0.958	0.85
c5315	90096	1	0.998	0.923	0.732	90096	1	0.998	0.933	0.757
c6288	80896	1	0.954	0.782	0.538	80896	1	0.957	0.797	0.568
s5378	71216	1	0.994	0.895	0.639	71216	1	0.998	0.95	0.84
s9234	149712	1	0.992	0.802	0.437	149712	1	0.995	0.847	0.452

proposed in the Section 4.4 for NAND/AND and NOR/OR gates were reversed.

The circuit reliability of the gate-specific quadded-transistor based technique using reversed connections(QT8R) and QT8 are compared in Table 4.8 for several percentages of faults. As expected, the reliability of reversed connections technique(QT8R) is worse as compared to the reliability for gate-specific quadded-transistor based technique(QT8).

4.5.5 Circuit Reliability Comparison of QT(QMR1), QMR3, TMR9, QT16 and QT8 Techniques

Figures 4.22 and 4.23 compare the circuit reliability for all the techniques(QT, QMR, QT16, QT8 and TMR9) for all the ISCAS benchmarks for injecting 0.1% and 0.5% of faults.

It is observed for most of the ISCAS benchmarks that in terms of circuit reliability, QT is better than QT8 which is better than QMR3 which is better than QT16 which is better than TMR9. The reason for choosing QMR3 and

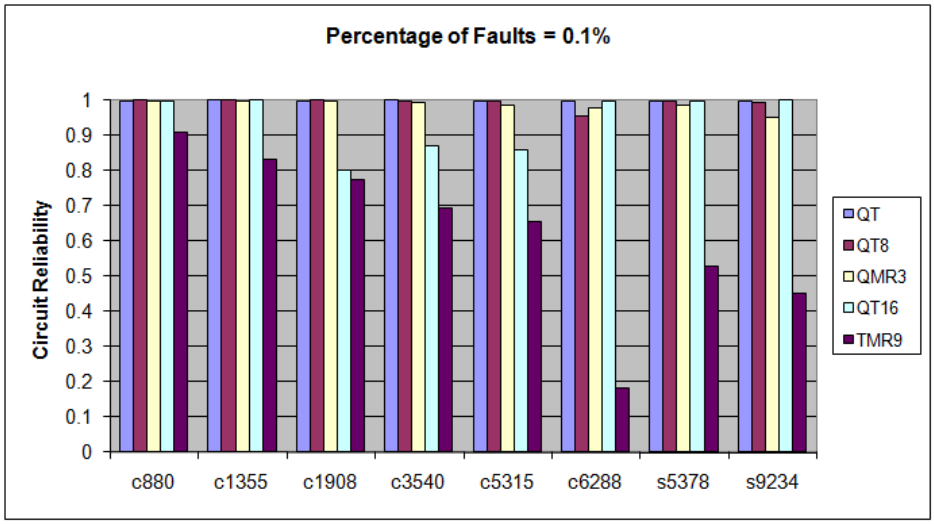

Figure 4.22: Comparison of circuit reliability of all approaches for ISCAS benchmarks for injecting 0.1% faults.

TMR9 is that for most of the benchmarks, QMR3 (i.e., QMR with module size of 3) and TMR9 (i.e., TMR with module size of 9) perform best in terms of circuit reliability.

Although QT based techniques perform better in terms of circuit reliability but as observed in Section 4.2, the implemented designs will involve higher delay because of higher gate capacitance and will perform slower as compared to QMR which only implements quadded-transistor structure in few selected gates.

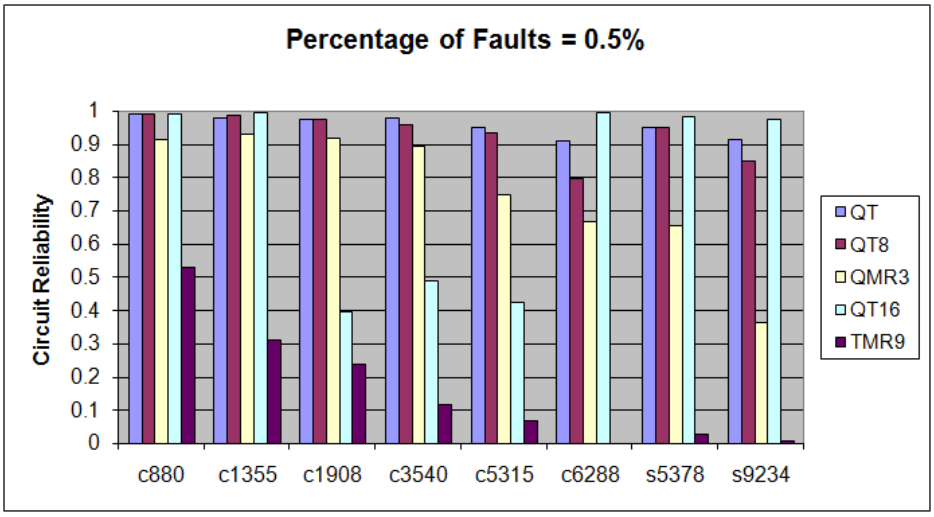

Figure 4.23: Comparison of circuit reliability of all approaches for ISCAS benchmarks for injecting 0.5% faults.

4.5.6 Circuit Area Comparison of QT, QMR, TMR, QT16 and QT8 Techniques

Table 4.9 specifies the area in terms of number of transistors occupied by QT, QMR, QT16, QT8 and TMR1, TMR3, TMR5, TMR7 and TMR9 for all the ISCAS benchmarks.

It is clear from the table that the QT16 requires highest area which is 4 times of QT and QT8 occupies half of the area occupied by QT16 which is 2 times of QT. QMR and QT require same area for all module sizes and TMR area varies with the module size. For the chosen module sizes of 1, 3, 5, 7 and 9, modular QMR is smaller in terms of area than their modular TMR counterpart circuits.

Table 4.9: Circuit area comparison of QT, QMR, TMR, QT16 and QT8 techniques.

Cct.	QT	QMR	QT16	QT8	TMR1	TMR3	TMR5	TMR7	TMR9
c880	7208	7208	28832	14416	10768	8374	7562	7632	7450
c1355	9232	9232	36928	18464	14568	11264	10872	10816	10760
c1908	13784	13784	55136	27568	22658	18836	16316	15812	15784
c3540	30016	30016	120064	60032	45878	37072	34734	34090	33502
c5315	45048	45048	180192	90096	66084	54828	47814	46772	46036
c6288	40448	40448	161792	80896	64160	50720	50720	50720	50720
s5378	35608	35608	142432	71216	66222	47614	43840	42222	41480
s9234	74856	74856	299424	149712	134500	92990	84086	80460	78598

4.6 Summary

In this chapter, transient and soft error mitigation techniques based on adding redundancy at the transistor level have been investigated. Three techniques namely QMR (Quadded Modular Redundancy), QT based SEU mitigation technique and gate-specific QT based SEU mitigation technique have been proposed, discussed and analyzed using a simulation-based methodology. Experimental analysis using a set of large ISCAS85 and ISCAS89 benchmark circuits has demonstrated that the proposed techniques provide significantly less circuit failure probability and higher reliability in comparison with the TMR technique which is the most popular technique reported in literature for SEU mitigation. QMR also requires less area overhead in terms of number of transistors as compared to TMR. The impact of module size on QMR and TMR has also been discussed.

CHAPTER 5

DEFECT-TOLERANT CROSSBAR DESIGN TECHNIQUE

In this chapter, a defect-tolerant technique that utilizes redundancy in the rows and columns of a nanoscale crossbar is described. The reliability analysis based on stuck-open crosspoint defect model indicates an increase in defect-tolerance at the cost of an increase in crossbar area.

5.1 Introduction

As discussed in Chapters 1 and 2, nanoscale technologies are increasingly being explored as an alternative solution to sustaining and possibly surpassing current performance trends of microelectronics. Hybrid technologies, whereby CMOS and

nanotechnologies are integrated to develop various devices, are seen as the next step on the pathway to realizing fully functioning nanoscale devices. Nanowires (NW) and Carbon Nanotubes (CNT) are emerging as the building blocks for future nanoscale technologies [85].

Nanoelectronic devices consist of many unreliable components due to the bottom-up fabrication methods, which makes defect-tolerance a necessity. In this chapter, a defect-tolerant architecture is proposed that is based on adding redundancy in the rows and columns of a molecular switch crossbar used for implementing logic. This is referred to as the Multi-crosspoint (MCP) architecture.

Crossbar architectures [26, 27] have been demonstrated as a proof of concept and have gained widespread acceptance as a design option for constructing nanoscale crossbars based logic circuits [6]. Crossbar architectures can be used for implementing logic functions [26, 27] or for bit storage in the crossbar nanomemory [85]. IBM has recently demonstrated that CNT can exhibit electrical characteristics that are similar to that of the state-of-the-art Silicon-based MOSFETs [84]. Moreover, a nonvolatile random access memory (RAM), implemented with nanoscale molecular switch crossbar arrays, has already been demonstrated to show great potential as a practical memory device [6].

The nanoscale crossbar implementation using diodes and resistors (diode/resistor logic) as explained in Section 2.6.1 is used as the basis for the proposed defect-tolerant architecture in this chapter. This chapter's work focuses on the reliability analysis of the defect-tolerant MCP architecture and its

comparison with the monomorphism-based reconfiguration approach for crossbars [26, 27]. An AND-OR logical model of the crossbar as mentioned in Section 2.6.1 is used for the implementation of logic on the crossbars and a missing crosspoint or stuck-open crosspoint defect model is used for analyzing reliability. The missing crosspoint model or stuck-open switch model has been used in most of the previous works [19, 25, 30, 26] due to the reason that the main types of defects are expected to be introduced during manufacturing of nanoscale crossbars. This is due to the fact that the plausible technologies for manufacturing at the nanoscale involve high temperature leading more probably to inoperative crosspoints rather than shorts in the wires [27, 29].

The novelty of the work proposed in this chapter lies in the presentation of the MCP architecture as an independent defect-tolerant nanoscale crossbar architecture without any prerequisites for defect maps or reconfiguration for defect avoidance. Furthermore, the work in this chapter presents the first attempt to implement and analyze the reliability of logic functions by utilizing the row and column redundancy using the proposed MCP architecture and comparing it against the reliability gains realized in the nanoscale crossbar structures using monomorphism-based reconfiguration algorithm [26, 27]. A row and column redundancy based work for realizing nanoscale crossbar memories is reported in [85].

The remainder of the chapter is organized as follows. In the next section, the proposed MCP architecture is discussed. After that, reliability analysis and

comparison with reconfiguration are presented. The section after that discusses the area analysis. The chapter is summarized in the last section.

5.2 Multi-Crosspoint Architecture

Crossbar architecture lends itself well as a good defect-tolerant architecture because its grid geometry allows for the direct implementation of redundancy [85]. In this section, the effects on the reliability and tolerance of defects in the nanoscale crossbars by having redundancy in the rows and columns of nanowires are discussed.

Defects in nanoscale electronics can take the form of hardware faults that occur during manufacturing or transient faults resulting from such anomalies as random charges in the devices, power supply fluctuations and crosstalk [65]. Defects in this chapter refer to those that occur during the fabrication process leading to defective crosspoints (also called switches or junctions), or other defects that cause crosspoint stuck-open faults.

5.2.1 Quadded MCP Architecture

In the proposed quadded MCP architecture, a single literal A is represented by a literal set $AA + AA$ by taking advantage of the Boolean algebra equality $A = AA + AA$ and by mapping it to a crosspoint set on the crossbar instead of a single crosspoint (as done in the normal implementation). This is achieved by adding one extra row and column in the AND crossbar and only one extra column in the

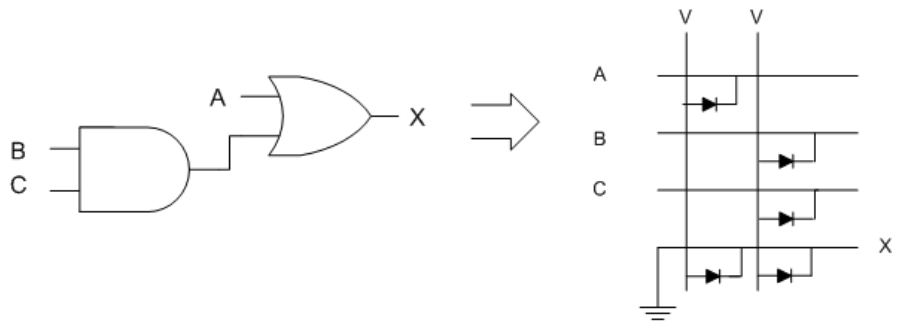

Figure 5.1: Crossbar implementation for a simple function $X = A + BC$.

OR crossbar for a given literal. The resulting MCP architecture for $k = 2$ will be referred to as quadded MCP architecture. The following example will clarify the redundancy scheme proposed in MCP architecture.

For example, in the normal AND-OR based crossbar implementation, a function $X = A + BC$ is implemented in the way shown in Figure 5.1. In comparison, in the MCP architecture, one redundant row and column is used for every literal which is input to the AND crossbar. This can be denoted by $k = 2$ where k represents the redundancy factor in the row and column in the AND portion of the crossbar. So, a function $X = A + BC$ is represented as $X = AA + AA + BBCC + BBCC$. This is illustrated in Figure 5.2. The reason for doing this is to improve defect tolerance of the partially defective crossbar on which the logic functions are mapped.

Defects are thus tolerated when a connection exists in the crosspoint set between at least one of its rows or one of its columns given that the corresponding

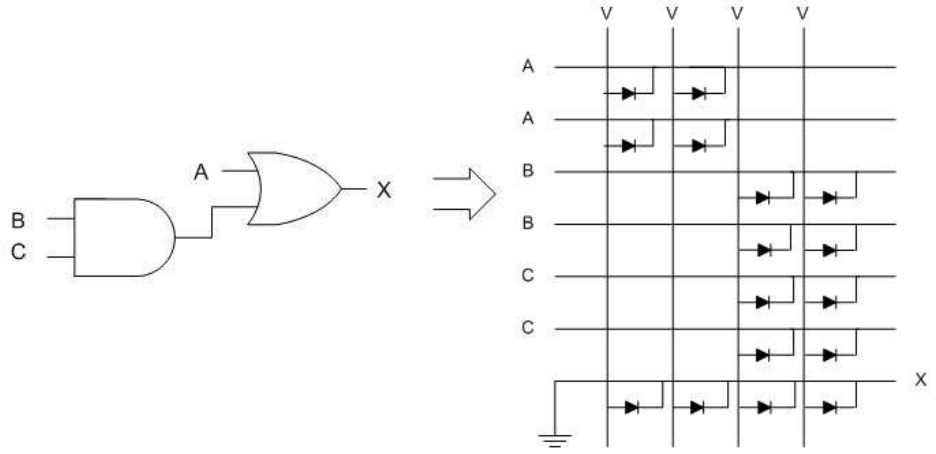

Figure 5.2: Multi-crosspoint architecture using row and column redundancy for a simple function $X = A + BC$ for $k = 2$.

crosspoint in the OR array is intact. This suggests that only specific patterns of defective crosspoints can be tolerated.

Few of the allowable defect configurations in which the defects are tolerated for the function $X = A + BC$ are shown in Figures 5.3, 5.4, 5.5 and 5.6.

Few obstructive defect configurations in which the defects will inhibit the implementation of the function $X = A + BC$ are shown in Figures 5.7, 5.8 and 5.9.

Conditions for failure to implement a given logic function in Quadded MCP Architecture

The obstructive defect configurations shown in Figures 5.7 to 5.10 provide the conditions for failure to implement a logic function in the presence of stuck-open crosspoint defects for the proposed quadde MCP architecture.

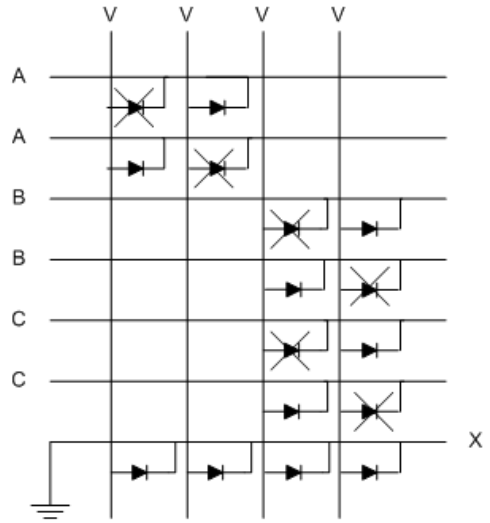

Figure 5.3: Allowable defect configuration in which the function will remain $X = A + BC$.

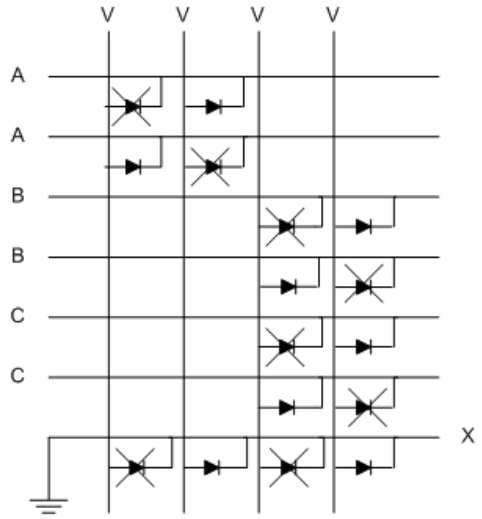

Figure 5.4: Allowable defect configuration in which the function will remain $X = A + BC$.

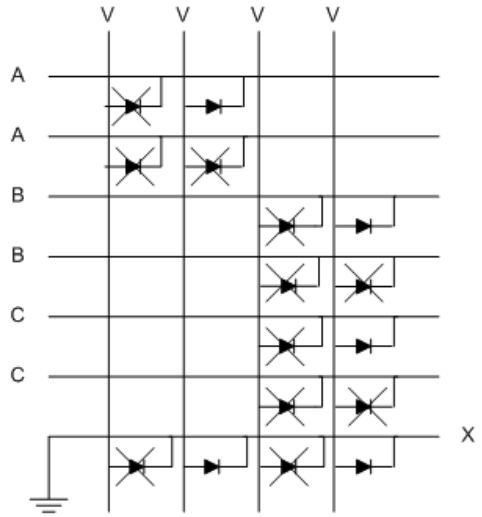

Figure 5.5: Allowable defect configuration in which the function will remain $X = A + BC$.

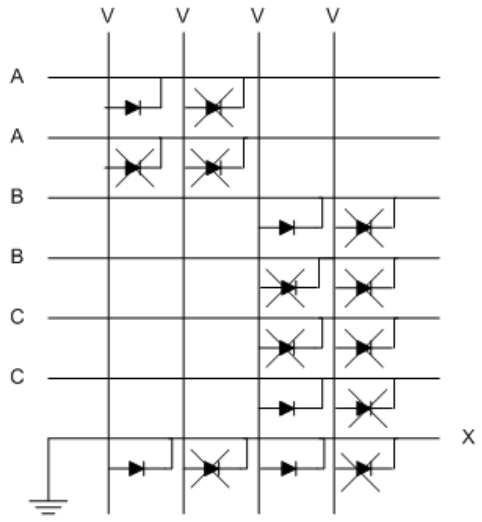

Figure 5.6: Allowable defect configuration in which the function will remain $X = A + BC$.

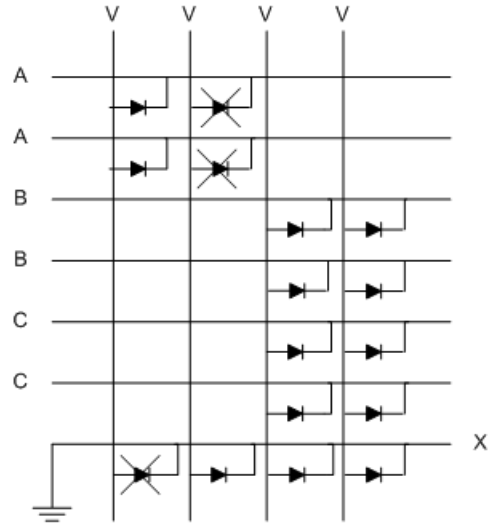

Figure 5.7: Obstructive defect configuration in which the function will become $X = 1$.

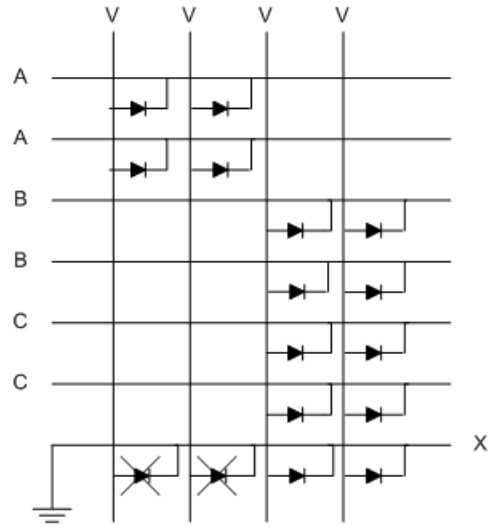

Figure 5.8: Obstructive defect configuration in which the function will become $X = BC$.

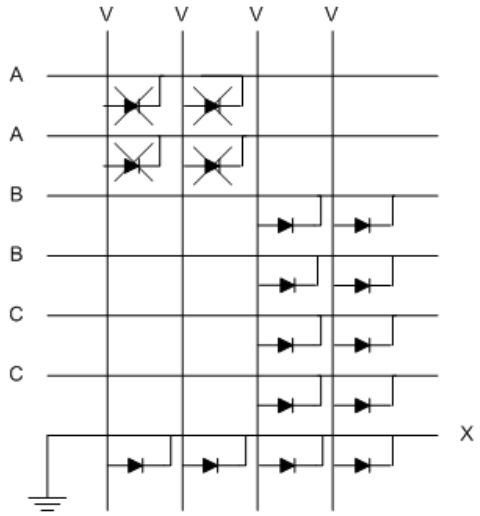

Figure 5.9: Obstructive defect configuration in which the function will become $X = 1$.

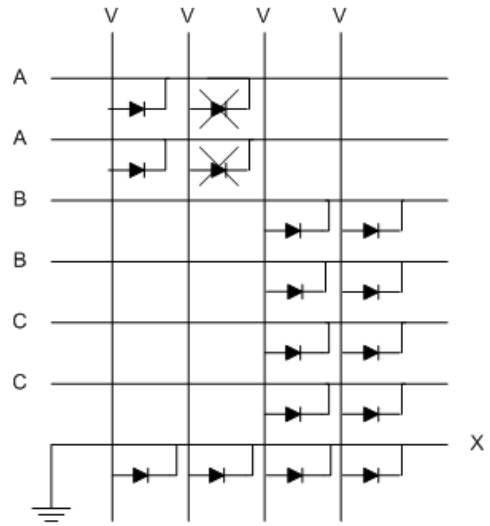

Figure 5.10: Obstructive defect configuration in which the function will become $X = 1$.

The condition for failure in the AND portion of crossbar is the following:

- If there are stuck-open defects in the same column for two adjacent crosspoints in the crosspoint set (for a product term) in the AND portion of the crossbar even if both of the corresponding crosspoints in the OR portion of the crossbar are intact (depicted in Figure 5.10)

The condition for failure in the OR portion of crossbar is the following:

- If there are stuck-open defects in two adjacent crosspoints in the OR portion of crossbar corresponding to the crosspoint set even if all the 4 crosspoints in the corresponding crosspoint set in the AND portion are intact (depicted in Figure 5.8)

The above conditions are shown in the figures for a simple single output function. Similarly, the successful implementation of a multiple output function will require that all individual output functions are successfully implemented on the crossbar.

5.2.2 Nona MCP Architecture

Following the same approach of row and column redundancy, the same function $X = A + BC$ is shown in Figure 5.11 with redundancy of two extra rows and two extra columns in the AND crossbar. For the nona MCP configuration, the redundancy factor $k = 3$.

The above configuration with redundancy factor $k = 3$ will afford more defect tolerance and reliability but the area overhead also will be greater. The conditions

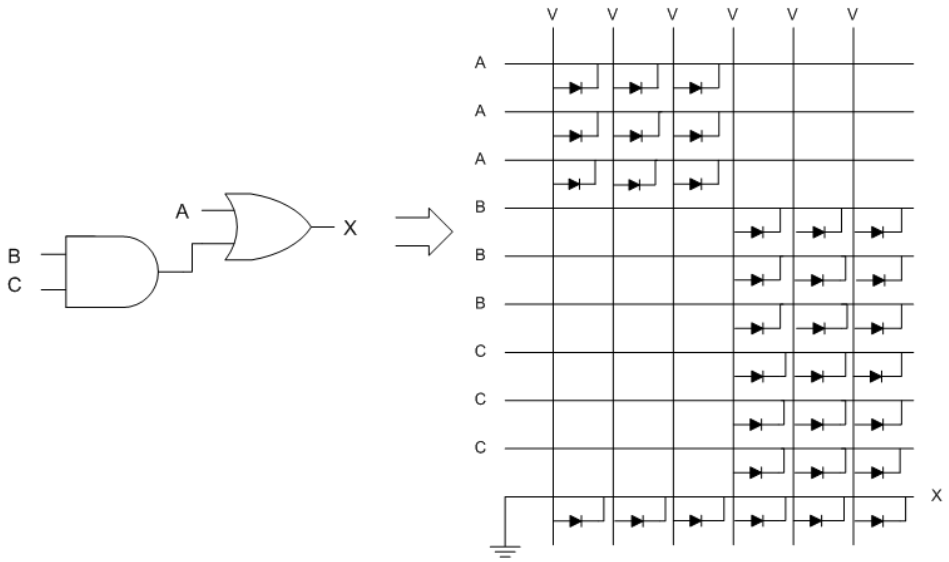

Figure 5.11: Multi-crosspoint architecture using row and column redundancy for a simple function $X = A + BC$ for $k = 3$.

for failure to implement a given logic function in Nona MCP architecture can be obtained similarly by extending the conditions of the quadded MCP architecture shown in previous section.

5.3 Experimental Results

To demonstrate the effectiveness of the row and column redundancy schemes, a number of experiments are performed, firstly on two different implementations of a 3-bit adder and then on 12 MCNC benchmark circuits. The type of defects used in all experiments are the missing crosspoint or switch stuck-open defects. The considered defects are also assumed to be randomly distributed and unclustered.

5.3.1 Reliability Analysis

For evaluating the circuit failure probability and circuit reliability, a modified version of the simulation-based model presented in [11] is used. Reliability comparison of MCP architecture with redundancy factors of $k = 2$ and $k = 3$ was performed with the monomorphism-based reconfiguration algorithm proposed in [27].

The circuit reliability in the experiments is considered as the probability of successfully implementing all output functions of a logic circuit on a given partially defective crossbar [27].

To compute the circuit failure probability F and reliability R, resulting from injecting m defective (stuck-open) crosspoints in the MCP architecture, the following procedure is used:

Inputs to the procedure:

1. Crossbar representation of the circuit to be mapped.

2. Target crossbar on which the circuit is to be mapped.

Procedure:

1. Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.

2. Inject m random defects (stuck-open) in the crosspoints of the target crossbar.

3. If the injected defects result in an obstructive defect configuration in the crossbar inhibiting successful mapping of all output functions, increment K by 1.

4. Decrement I by 1 and if I is not 0 goto step 2.

5. Circuit failure probability $F = K/1000$.

6. Reliability $R = 1 - F$.

To compute the circuit failure probability F and reliability R, resulting from injecting m defective (stuck-open) crosspoints in the crossbar architecture for the monomorphism-based reconfiguration algorithm, the following procedure is used:

Inputs to the procedure:

1. Crossbar representation of the circuit to be mapped.

2. Target crossbar on which the circuit is to be mapped.

Procedure:

1. Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.

2. Inject m random defects (stuck-open) in the crosspoints of the target crossbar.

3. If the injected defects result in failure to find for the circuit a monomorphism on the target crossbar for successful mapping of all output functions, increment K by 1.

4. Decrement I by 1 and if I is not 0 goto step 2.

5. Circuit failure probability $F = K/1000$.

6. Reliability $R = 1 - F$.

Figure 5.12 shows the 3-bit adder circuit and its straight-forward implementation on a crossbar. The ripple-carry logic implementation (shown on top in the figure) translates directly to a diode crossbar implementation (shown at the bottom of figure) using feedback from some of the outputs to the inputs (gray lines). The input wire marked $-A_0$ gives the complement of input bit A_0, and similarly for the other inputs. Note that the carry bit between successive stages of the crossbar implementation must be presented in both original and complemented forms.

Figure 5.13 shows another implementation of the same 3-bit half adder circuit. Although this approach uses more diodes, it consumes less area. Inputs and outputs are labeled as in Figure 5.12.

Figure 5.14 compares the reliability obtained for quadded MCP, nona MCP and monomorphism-based reconfiguration approaches for the adder circuit shown in Figure 5.12 for injecting different percentages of stuck-open crosspoint defects. It is observed that the reliability of monomorphism-based reconfiguration scheme is better than nona MCP architecture whose reliability in turn is better than quadded MCP architecture.

Similarly, Figure 5.15 compares the reliability obtained for quadded MCP, nona MCP and monomorphism-based reconfiguration approaches for the adder circuit

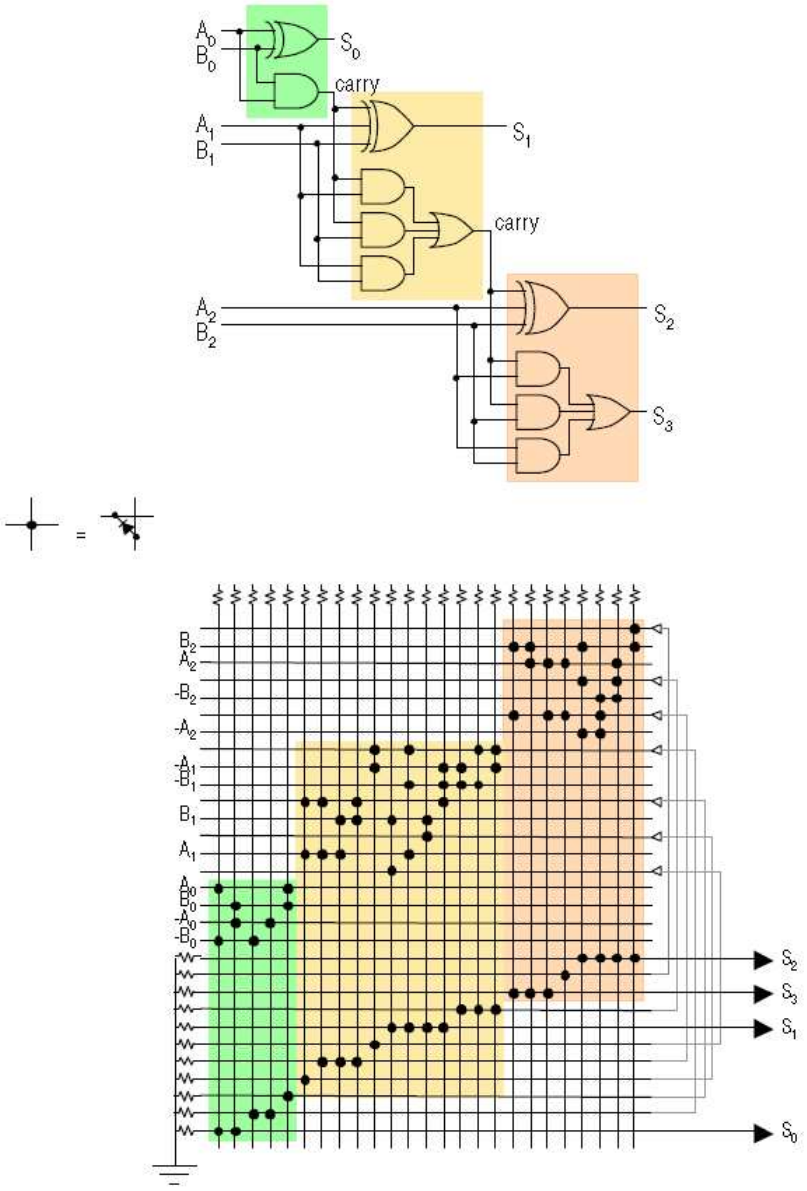

Figure 5.12: A 3-bit adder which adds two 3-bit numbers (denoted as the bits $A_2A_1A_0$ and $B_2B_1B_0$, respectively) to produce a 4-bit sum (with bits $S_3S_2S_1S_0$).

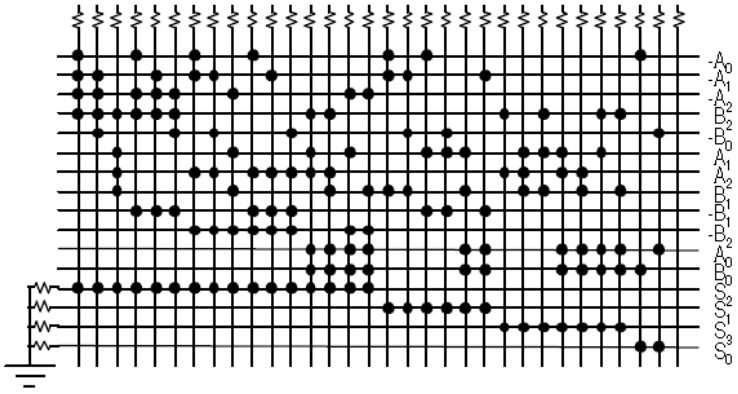

Figure 5.13: A 3-bit adder implemented as 2-level logic in a single diode crossbar.

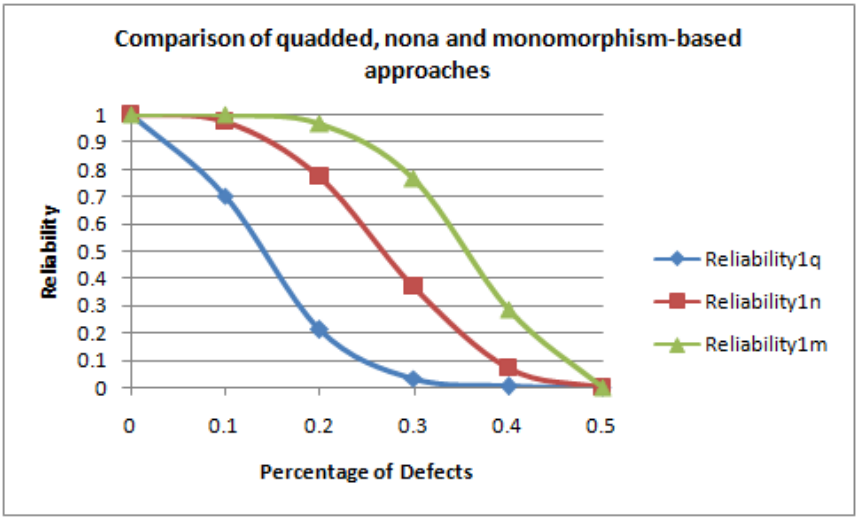

Figure 5.14: Reliability comparison of quadded, nona and monomorphism-based approaches for 3-bit adder shown in Figure 5.12.

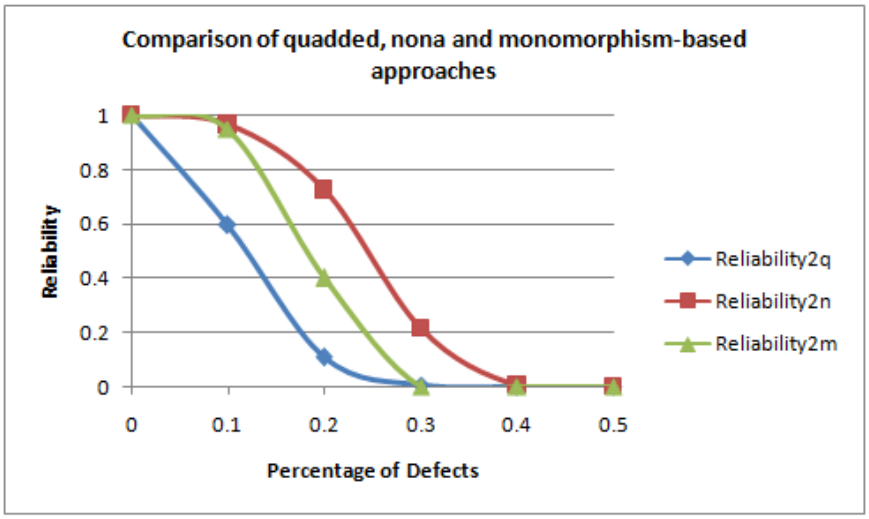

Figure 5.15: Reliability comparison of quadded, nona and monomorphism-based approaches for 3-bit adder shown in Figure 5.13.

shown in Figure 5.13 for injecting different percentages of stuck-open crosspoint defects. Contrary to the trend in Figure 5.14, it is observed that the reliability of nona MCP architecture is better than monomorphism-based reconfiguration scheme whose reliability is in turn better than quadded MCP architecture.

Table 5.1 compares the circuit reliability between quadded MCP and monomorphism-based reconfiguration approaches for 12 MCNC benchmarks.

For the MCNC benchmarks shown in Tables 5.1 and 5.2, the following flow was adopted.

- The MCNC circuits were originally present in PLA format. The first step performed was the logic optimization of the circuits using the ESPRESSO tool [98].

Table 5.1: Comparison of circuit reliability between quadded MCP and monomorphism-based reconfiguration approaches.

	Quadded MCP					Monomorphism				
Cct.	0.5%	1%	5%	10%	20%	0.5%	1%	5%	10%	20%
bench1	0.993	0.972	0.335	0.002	0	1	0.998	0.939	0.473	0
dk17	0.999	0.998	0.887	0.514	0.019	1	0.997	0.945	0.693	0.005
ex1010	0.986	0.922	0.043	0	0	0.967	0.983	0.964	0.58	0
exp	0.996	0.986	0.534	0.037	0	0.884	0.76	0.2	0.002	0
inc	0.997	0.991	0.816	0.328	0.002	1	0.999	0.921	0.477	0.002
m1	0.996	0.991	0.67	0.131	0	0.873	0.782	0.085	0.001	0
m2	0.994	0.962	0.265	0.001	0	0.664	0.377	0.001	0	0
m3	0.989	0.953	0.235	0.003	0	.595	0.322	0	0	0
m4	0.989	0.951	0.165	0	0	0.55	0.13	0	0	0
p82	0.999	0.993	0.832	0.395	0.007	1	0.993	0.852	0.491	0.006
test1	0.991	0.971	0.36	0.005	0	1	1	0.919	0.589	0
test4	0.985	0.935	0.078	0	0	1	0.995	0.68	0.026	0

- The obtained optimized circuits were converted to crossbar implementation by writing a conversion script. The resulting crossbar based representation was used as input to the simulation procedures for MCP reliability analysis and monomorphism based reconfiguration algorithm.

The monomorphism based reconfiguration algorithm which is used in our work for comparison is the one reported in [89] and is part of the VF graph matching library available from [91]. Since graph monomorphism is a complete search method, a CPU time threshold of 30 sec was used to obtain monomorphism. The 30 sec CPU time threshold is also used in [27] from which the adder circuits of Figure 5.12 and 5.13 for comparison have also been taken.

Table 5.2 compares the circuit reliability between nona MCP and monomorphism-based reconfiguration approaches for 12 MCNC benchmarks.

As seen in the circuit reliability results shown in Tables 5.1 and 5.2, the ob-

Table 5.2: Comparison of circuit reliability between nona MCP and monomorphism-based reconfiguration approaches.

	Nona MCP					Monomorphism				
Cct.	0.5%	1%	5%	10%	20%	0.5%	1%	5%	10%	20%
bench1	1	1	0.965	0.75	0.022	1	0.998	0.939	0.473	0
dk17	1	1	0.966	0.958	0.652	1	0.997	0.945	0.693	0.005
ex1010	1	1	0.991	0.777	0	0.967	0.983	0.964	0.58	0
exp	1	0.999	0.982	0.835	0.12	0.884	0.76	0.2	0.002	0
inc	1	1	0.986	0.938	0.455	1	0.999	0.921	0.477	0.002
m1	1	1	0.99	0.883	0.242	0.873	0.782	0.085	0.001	0
m2	1	0.999	0.956	0.674	0.02	0.664	0.377	0.001	0	0
m3	1	1	0.954	0.655	0.003	.595	0.322	0	0	0
m4	1	1	0.959	0.573	0.005	0.55	0.13	0	0	0
p82	1	1	0.995	0.949	0.504	1	0.993	0.852	0.491	0.006
test1	0.999	0.999	0.967	0.769	0.037	1	1	0.919	0.589	0
test4	1	0.998	0.898	0.409	0	1	0.995	0.68	0.026	0

served reliability trend in some MCNC benchmarks is consistent with the one which was obtained for the adder implementation in Figure 5.13 and shown in Figure 5.15 i.e., the best reliability is obtained for nona MCP, second by monomorphism followed by quadded MCP. But in some MCNC benchmarks like exp, $m1$, $m2$, $m3$ and $m4$, quadded MCP even outperformed monomorphism-based approach. Looking into the structure of the benchmarks gave the insight that those benchmark circuits in which the product terms which need to be implemented on the columns of the crossbar are large in terms of the number of literals, the performance of the monomorphism-based reconfiguration algorithm is worse than the MCP because of the relative difficulty in finding the defect free columns in the crossbar to implement denser product terms. For circuits with sparse product terms ie., having few literals in the product terms, the monomorphism approach has more flexibility and it performs better. Also in some of the benchmarks like

*dk*17, *inc* and *p*82, the reliability results of the quadded MCP are close to those of the monomorphism-based reconfiguration algorithm. The obtained reliability results justify the adoption of MCP architecture.

5.3.2 Area Analysis

In order to calculate the area required by the proposed MCP architecture, the approach presented in [27] is followed. It is based on counting the number of crosspoints needed by the AND and OR crossbars and summing the two to form the total area of the crossbar.

The total area in terms of number of crosspoints for the monomorphism-based reconfiguration approach, quadded MCP and nona MCP is shown in Tables 5.3, 5.4 and 5.5. The first column is the name of the benchmark circuit, the second column is the number of rows in the AND portion of crossbar. The third column is the number of product terms which need to be implemented in columns of crossbar. The fourth column is the number of rows in the OR portion crossbar. The fifth and sixth and seventh columns are the number of crosspoints in AND, OR and the full crossbar. It should be noted that smallest size crossbar needed to implement all product terms of a benchmark is used for the monomorphism-based reconfiguration approach.

Reliability results combined with area results show that for increasing degrees of redundancy, the reliability of the MCP architecture improves with acceptable area overheads. As observed in Tables 5.3, 5.4 and 5.5, the quadded MCP has

Table 5.3: Crossbar area in terms of number of crosspoints for the monomorphism-based reconfiguration architecture.

Cct.	Rows in AND CB	Columns	Rows in OR CB	AND CPs	OR CPs	Total CPs
Adder1	19	25	11	475	275	750
Adder2	12	31	4	372	124	496
bench1	18	139	9	2502	1251	3753
dk17	20	18	11	360	198	558
ex1010	20	284	10	5680	2840	8520
exp	16	59	18	944	1062	2006
inc	14	30	9	420	270	690
m1	12	19	12	228	228	456
m2	16	47	16	752	752	1504
m3	16	66	16	1056	1056	2112
m4	16	105	16	1680	1680	3360
p82	10	21	14	210	294	504
test1	16	121	10	1936	1210	3146
test4	16	120	30	1920	3600	5520

on average 3 times more area requirements as compared to monomorphism-based reconfiguration approach and nona MCP has on average 6 times more area requirements as compared to monomorphism-based reconfiguration approach. The average case is taken as the crossbar in which there are equal number of rows in the AND portion and OR portion of the non-redundant crossbar. Keeping the reliability and area results in view, it can be said that there is an area-reliability tradeoff which needs to be considered while using any of the proposed MCP architectures.

The greatest advantage of the MCP architecture is that it does not require any type of defect diagnosis, defect mapping and consequently defect avoidance using monomorphism or any other algorithm employing reconfiguration for avoiding the

Table 5.4: Crossbar area in terms of number of crosspoints for the quadded MCP architecture.

Cct.	Rows in AND CB	Columns	Rows in OR CB	AND CPs	OR CPs	Total CPs
Adder1	34	50	11	1700	550	2250
Adder2	24	62	4	1488	248	1736
bench1	36	278	9	10008	2502	12510
dk17	40	36	11	1440	396	1836
ex1010	40	568	10	22720	5680	28400
exp	32	118	18	3776	2124	5900
inc	28	60	9	1680	540	2220
m1	24	38	12	912	456	1368
m2	32	94	16	3008	1504	4512
m3	32	132	16	4224	2112	6336
m4	32	210	16	6720	3360	10080
p82	20	42	14	840	588	1428
test1	32	242	10	7744	2420	10164
test4	32	240	30	7680	3600	11280

location of defects.

Reconfiguration algorithms for defect-tolerant crossbars of which monomorphism is an example require extensive time for execution and mapping of logic on partially defective crossbars. For monomorphism, it is reported in [27] that it is a complete search algorithm which can have prohibitive computational costs for searching for a solution. Also there is no guarantee that for given defect locations, a solution will always be found. Hence in a mass-production manufacturing context where there can be hundreds of thousands of crossbars implementing logic, the computing and testing time for reconfiguration based methods can be immense. For such cases, the defect-tolerant architectures like the MCP architecture proposed in this chapter have a clear edge in terms of reliability over reconfiguration

Table 5.5: Crossbar area in terms of number of crosspoints for the nona MCP architecture.

Cct.	Rows in AND CB	Columns	Rows in OR CB	AND CPs	OR CPs	Total CPs
Adder1	57	75	11	4275	825	5100
Adder2	36	93	4	3348	372	3720
bench1	54	417	9	22518	3753	26271
dk17	60	54	11	3240	594	3834
ex1010	60	852	10	51120	8520	59640
exp	48	177	18	8496	3186	11682
inc	42	90	9	3780	810	4590
m1	36	57	12	2052	684	2736
m2	48	141	16	6768	2256	9024
m3	48	198	16	9504	3168	12672
m4	48	315	16	15120	5040	20160
p82	30	63	14	1890	882	2772
test1	48	363	10	17424	3630	21054
test4	48	360	30	17280	10800	28080

based defect avoidance methods.

5.4 Summary

In this chapter, defect-tolerant techniques for implementing logic on 2D crossbars based on adding redundancy at the crosspoint level both in the crossbar rows and columns are proposed and discussed. The proposed technique is called Multi-crosspoint(MCP) architecture and the two versions presented in this chapter are called quadded MCP and nona MCP architectures. The proposed techniques provide defect tolerance against a large number of defective(stuck-open) crosspoints. Comparison of the proposed MCP architecture with monomorphism based reconfiguration approach has shown that the nona MCP architecture affords better

reliability for most of the tested benchmark circuits. The comparison has also shown that the quadded MCP architecture is slightly inferior in reliability to the monomorphism-based approach but on some of the benchmark circuits it gives better or equivalent reliability. This improvement in reliability by using the MCP architecture is achieved at a higher overhead but the real advantage of the proposed schemes is that they do not require any defect mapping or algorithmic steps for circuit implementation and defect avoidance and are defect-tolerant by virtue of their redundant geometry.

CHAPTER 6

DEFECT-TOLERANT FPGA DESIGN TECHNIQUE

In this chapter, an investigation is made to find out the answer to the question: "whether it is more feasible in terms of reliability to implement defect-tolerant CLBs (Configurable Logic Blocks) using the quadded-transistor technique or is it feasible to allocate more spare CLBs for the mapping of a given benchmark circuit on FPGA". The experimental analysis is carried out using the VPR tool [93].

6.1 Introduction

As discussed in Chapter 2, the high levels of integration and small submicron device sizes used in present VLSI technologies and projected for future nanoelectronic technologies for FPGAs can result in higher occurrences of defects and operational faults. Thus, there is a critical need for defect tolerance and reconfiguration techniques for FPGAs to increase chip yields as well as system reliability

in the field.

Several methods as mentioned in Section 2.7 have been proposed to tolerate faults in CLBs and interconnect of FPGAs. Most of these methods are based on the concept of defect avoidance using spare CLB and interconnection resources via reconfiguration in which the faults are detected and reconfigured around the defective resources [92, 76].

In this chapter, a transistor-level defect-tolerant design for FPGA CLBs is proposed. The focus of the work in this chapter is on masking permanent defects in CLBs by utilizing the defect-tolerant quadded-transistor structure presented in Chapter 3. The proposed defect-tolerant technique is compared to the spares-based reconfiguration technique using 2 and 3 spare CLBs.

6.2 Defect-Tolerant CLBs for FPGAs

As described in Chapter 2, an FPGA consists of regular structures called Configurable Logic Blocks (CLBs) connected to each other via interconnection consisting of wiring and Switch Blocks (SBs). A basic FPGA logic block is shown in Figure 6.1. Its main components are 4-input LUT, a Flip-Flop and a multiplexer.

The 4-input LUT can be implemented at the transistor level using the multiplexing scheme as shown in Figure 6.2.

In the proposed defect-tolerant CLBs, every transistor T in the CLB is replaced by the Quadded-Transistor structure as shown in Figure 3.1. This increases the size of each CLB four times in terms of number of transistors but as demon-

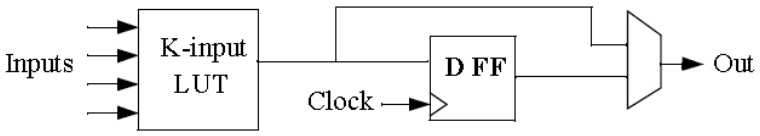

Figure 6.1: A basic FPGA logic block.

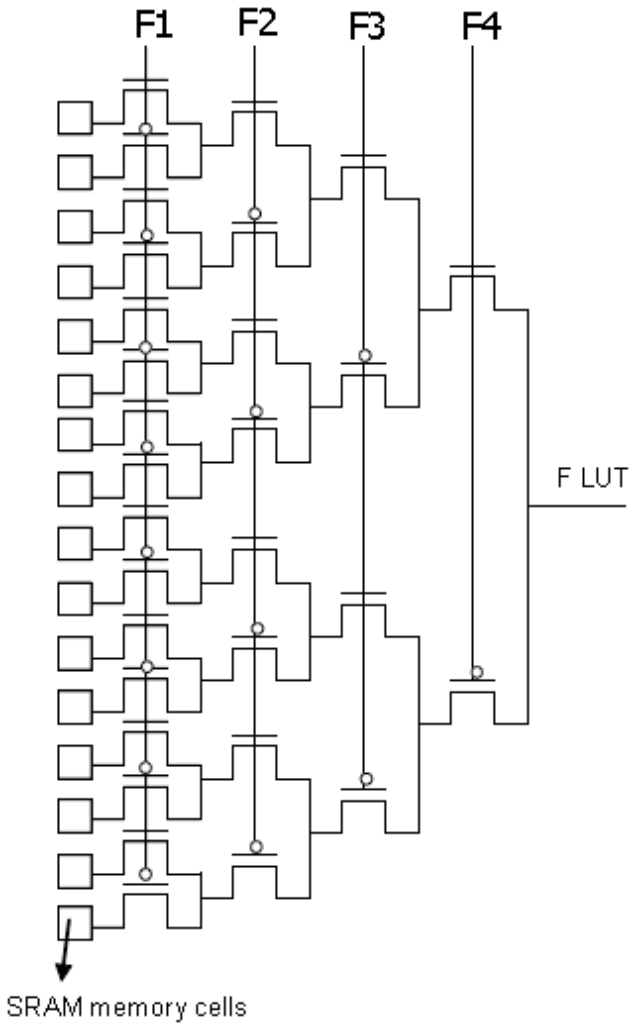

Figure 6.2: Schematic of 4-input LUT.

strated in the next section, the quadded transitor implementation of CLB greatly increases the probability to tolerate permanent defects in each CLB.

6.3 Experimental Results

To demonstrate the effectiveness of the quadded-transistor based CLB, experimental analysis for estimation of circuit failure probability, reliability and area is carried out using 8 MCNC benchmark circuits.

6.3.1 Reliability Analysis

The reliability analysis consists of two major steps.

- Placement of benchmark circuits on FPGA using the VPR Tool.
- Estimation of Circuit Failure Probability and Reliability using fault-injection simulation.

The following flow is used for the FPGA placement of the benchmark circuits using the VPR tool.

- Technology-independent logic optimization of all circuits is performed using the SIS synthesis package [94].
- Then the circuits are technology-mapped into netlists with 4-input LUTs and flip-flops using the FlowMap tool [95] resulting in .blif format netlists of logic blocks.

- The .blif format netlists are then packed into logic blocks using the T-VPack tool [96] resulting in .net format netlists which can be directly read by VPR.

- The netlists of circuits along with a 4-input LUT FPGA architecture description file are input to the VPR tool and the circuit is both placed as well as routed using the VPR tool resulting in .place format file describing circuit placement and .route format file describing circuit routing on FPGA.

The FPGA placements of benchmark circuits generated by VPR is used as the starting point for reliability analysis.

The estimation of circuit failure probability and reliability for the benchmark circuits is carried out using the following fault-injection simulation procedure:

- Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.

- Randomly inject m transistor defects in the original FPGA placed circuit.

- If the injected transistor defects result in any defective CLB, increment K by 1.

- Decrement I by 1 and if I is not 0 goto step 2.

- Circuit Failure Probability $F = K/1000$.

- Reliability $R = 1 - F$.

In order to compare the circuit failure probability and reliability of the quadded-transistor CLBs with the reconfiguration approach using spares, follow-

ing fault-injection simulation procedure is used.

- Set the number of iterations to be performed, I, to 1000 and the number of failed simulations, K, to 0.
- Randomly inject m transistor defects in the original FPGA placed circuit augmented with N spares for each CLB.
- If the injected transistor defects result in a defective CLB along with defects in all N spares allocated for that CLB, increment K by 1.
- Decrement I by 1 and if I is not 0 goto step 2.
- Circuit Failure Probability $F = K/1000$.
- Reliability $R = 1 - F$.

Using the aforementioned simulation procedures, experiments were carried out on 8 MCNC benchmark circuits and comparison was performed with spare based reconfiguration approach using 2 spares ($N = 2$) and 3 spares ($N = 3$). Figure 6.3 shows comparison of circuit failure probability for alu4 MCNC benchmark circuit for different percentages of injected defects.

As shown in the comparison of alu4 benchmark circuit, the QT based CLB implementation has less circuit failure probability than 2-spare based reconfiguration approach but is inferior to the 3-spare based reconfiguration approach. Tables 6.1 and 6.2 compare the circuit failure probability of QT based CLB approach and 2 spares and 3 spares based reconfiguration approach for different percentages

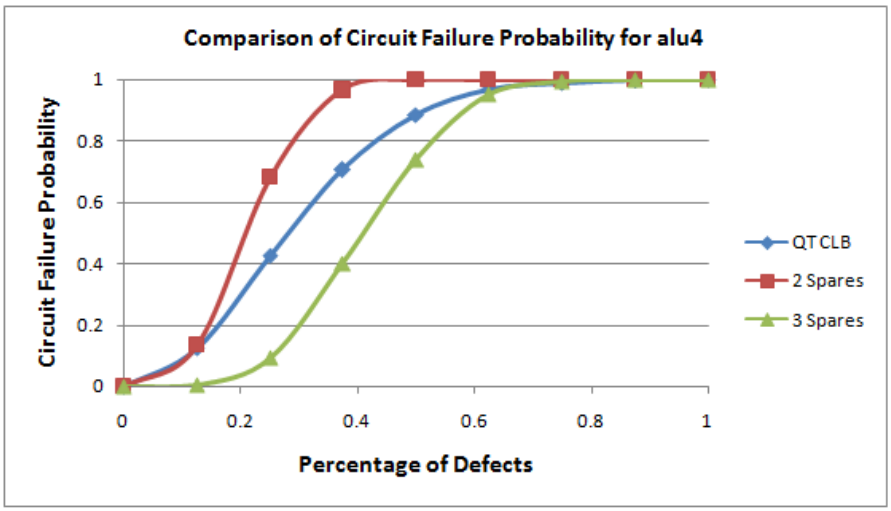

Figure 6.3: Comparison of circuit failure probability for alu4 benchmark.

of injected defects. The same trend is observed in all the 8 MCNC benchmark circuits with the QT based CLB approach performing midway between 2 spares and 3 spares based reconfiguration approach.

6.3.2 Area Analysis

Comparison of circuit area in terms of number of transistors as shown in Table 6.3 reveals that the QT based CLB technique and 3 spares based reconfiguration approach have same area but 2 spares based approach occupies 25% less area in terms of number of transistors. As reported in [92], the spares based reconfiguration not only incurs area for additional number of CLBs used but also requires additional routing resources (i.e.,channels, wiring and bypassing circuits) in order to provide reconfiguration around the defective CLB. One advantage of the QT based CLBs

Table 6.1: Comparison of circuit failure probability between QT CLB and 2 spares based reconfiguration approaches.

	Quadded-Transistor CLB					2 Spares				
Cct.	Trans.	0.12%	0.25%	0.37%	0.5%	Trans.	0.12%	0.25%	0.37%	0.5%
alu4	231344	0.124	0.425	0.705	0.887	173508	0.133	0.682	0.966	1
apex2	285456	0.14	0.484	0.793	0.936	214092	0.179	0.749	0.987	1
apex4	191824	0.101	0.358	0.617	0.84	143868	0.108	0.611	0.95	0.997
diffeq	227544	0.114	0.426	0.698	0.895	170658	0.132	0.696	0.977	1
elliptic	547808	0.284	0.73	0.942	0.999	410856	0.307	0.937	0.999	1
ex1010	698896	0.355	0.812	0.966	0.997	524172	0.4	0.974	1	1
ex5p	161728	0.098	0.315	0.588	0.793	121296	0.15	0.546	0.918	0.993
frisc	540512	0.292	0.72	0.942	0.991	405384	0.35	0.911	0.999	1

Table 6.2: Comparison of circuit failure probability between QT CLB and 3 spares based reconfiguration approaches.

	Quadded-Transistor CLB					3 Spares				
Cct.	Trans.	0.12%	0.25%	0.37%	0.5%	Trans.	0.12%	0.25%	0.37%	0.5%
alu4	231344	0.124	0.425	0.705	0.887	231344	0.006	0.092	0.399	.741
apex2	285456	0.14	0.484	0.793	0.936	285456	0.009	0.118	0.441	0.802
apex4	191824	0.101	0.358	0.617	0.84	191824	0.006	0.085	0.334	0.687
diffeq	227544	0.114	0.426	0.698	0.895	227544	0.008	0.096	0.398	0.723
elliptic	547808	0.284	0.73	0.942	0.999	547808	0.022	0.202	0.645	0.955
ex1010	698896	0.355	0.812	0.966	0.997	698896	0.028	0.254	0.775	0.979
ex5p	161728	0.098	0.315	0.588	0.793	161728	0.006	0.071	0.277	0.609
frisc	540512	0.292	0.72	0.942	0.991	540512	0.014	0.216	0.686	0.968

is that no additional routing resources are needed by the CLBs because the defect tolerance is built into every quadded-transistor structure implementing the CLB. Another advantage of the QT based CLBs is that no defect mapping procedure is required to identify the defective CLBs that should be avoided during the place and route process due to the fact that the defect tolerance is built into every QT based CLB.

Table 6.3: Comparison of area in terms of number of transistors and CLBs for QT based CLB approach and 2 and 3 spares based approach.

	QT CLB		2 spares		3 spares	
Cct.	CLBs	Trans.	CLBs	Trans.	CLBs	Trans.
alu4	1522	231344	4566	173508	6088	231344
apex2	1878	285456	5634	214092	7512	285456
apex4	1262	191824	3786	143868	5048	191824
diffeq	1497	227544	4491	170658	5988	227544
elliptic	3604	547808	10812	410856	14416	547808
ex1010	4598	698896	13794	524172	18392	698896
ex5p	1064	161728	3192	121296	4256	161728
frisc	3556	540512	10668	405384	14224	540512

6.4 Summary

In this chapter, defect-tolerant CLB design technique for FPGAs using the quadded-transistor structure is proposed. The proposed technique incurs additional area overhead in terms of number of transistors but provides appreciable defect tolerance against permanent defects. A comparison of the proposed technique with the spare based reconfiguration using transistor-level fault injection simulation has been performed and the technique is found to be better in defect tolerance than the 2 spares based reconfiguration approach but inferior to the 3 spares based reconfiguration approach. It is also noted that the defect-tolerant CLB approach does not need extra wiring and routing resources as the defect tolerance is present in every quadded-transistor structure. It is expected that in future nanoelectronics based FPGAs, the quadded-transistor structure may afford more defect tolerance to the nanoscale based CLBs.

CHAPTER 7

CONCLUSION

7.1 Conclusion

Defect-tolerant digital system design techniques have recently attracted a considerable amount of interest in the research community. This is due to the fact that in comparison to CMOS, higher defect rates are being projected for future nanoelectronics based digital circuits. The renewed interest in defect tolerance has motivated researchers to re-investigate pre-CMOS era techniques which are mostly gate level and module level. The work reported in this thesis is based on detailed investigation of transistor-level techniques for designing reliable digital circuits. Following are summary and conclusions of this research:

- A recently proposed transistor-level defect-tolerant technique called Quadded-Transistor technique is studied in detail and is extended to develop another transistor-level defect-tolerant technique called Nona-Transistor technique. Both theoretical and experimental analysis are performed for tol-

erating transistor stuck-open and stuck-short defects. Reliability and failure rate analysis of Nona-Transistor technique and Quadded Logic technique for transistor stuck-open and stuck-short defects has proved that Nona-Transistor technique has outperformed Quadded Logic technique in terms of defect tolerance. Nona-Transistor technique has also shown better reliability than Quadded-Transistor technique at the cost of higher area.

- Hybridization of Nona-transistor technique with TMR is proposed in order to achieve higher reliability following the idea of hybridization of quadded-transistor technique with TMR by implementing only majority voters using Nona-transistor structure and it is concluded that combinations of gate-level defect-tolerant techniques like TMR and transistor-level defect-tolerant techniques like Quadded and Nona-tarnsistor structures will give higher defect tolerance.

- A new transistor-level technique is proposed for mitigating transient and soft errors in digital circuits. The proposed technique is based on selective application of the Quadded-Transistor structure and is called Quadded Modular Redundancy(QMR). Simulation-based comparison of QMR with TMR for transient faults and with different module sizes has shown that QMR affords more tolerance to transient faults in comparison to TMR and with less number of transistors. Two more techniques based on QT structure are also proposed. Comparison based on reliability analysis has shown that the proposed techniques are more efficient than TMR for mitigating SEUs but

have higher area overhead.

- A new defect-tolerant architecture for implementing logic circuits on partially defective nanoscale crossbars is proposed. The proposed crossbar architecture called Multi-crosspoint(MCP) architecture uses row and column redundancy in order to achieve higher defect tolerance in nanoscale crossbar-based circuits. Two variants of the MCP architecture called quadded MCP and nona MCP with redundancy factors of 2 and 3 are evaluated using simulations. A comparison of the proposed architecture is made with the monomorphism based reconfiguration algorithm for defect-tolerant crossbar design for a number of benchmark circuits and the experimental analysis has shown that the nona MCP architecture performs better than monomorphism based approach on circuits with more dense product terms. For the MCP architecture, it is concluded that it does not involve any defect diagnosis, mapping and avoidance but provides defect tolerance by virtue of its redundancy only. The MCP architecture has higher overhead in terms of number of crosspoints but is favorable for crossbar based implementations which want to avoid computational time normally required by reconfiguration approaches in searching for a feasible solution for individual crossbars.

- Transistor-level defect-tolerant FPGA design technique is also explored for realizing reliable Configurable Logic Blocks (CLBs). Simulation based comparison of QT based CLBs is performed with 2 spares and 3 spares based technique which shows that the QT based CLB affords better defect toler-

ance than 2 spares based technique but is inferior to 3 spares based technique. It is expected that for future nanotechnology based FPGAs, quadded-transistor structure may be beneficial for masking manufacturing defects in CLBs.

7.2 Future Work

This work can be extended to do further reserach in the following ways:

- Due to the lack of availability of real-world fabrication and defect data, the transistor-level techniques have only been assessed using theoretical and simulation based approaches. A very interesting extension of this work could be the assessment of the impact of the proposed techniques on the reliability of the fabricated nanoelectronic circuits in the presence of real fabrication related defects.

- The proposed Multi-crosspoint (MCP) architecture can be used in the implementation of Hybrid CMOS / Nanoscale Crossbar based FPGAs particularly for designing defect-tolerant nanoscale crossbar based LUTs.

- Defect-tolerant FPGA design technique reported in Chapter 6 has only covered CLBs. Similar work can be explored for designing defect-tolerant Switch Blocks (SBs) and Connection Blocks(CBs) as well as SRAM configuration memory in FPGAs.

REFERENCES

[1] M. Butts, A. DeHon and S. C. Goldstein, "Molecular Electronics: devices, systems and tools for gigagate, gigabit chips," *Proceedings of International Conference on Computer-Aided Design*, pp. 433 - 440, 2002.

[2] T. N. A. Bachtold, P. Harley and C. Dekker, "Logic circuits with carbon nanotube transistors," *Science*, no. 294, pp. 1317 - 1320, 2001.

[3] Y. Cui and C. M. Lieber, "Functional nanoscale electronic devices assembled using silicon nanowire building blocks," *Science*, no. 291, pp. 851 - 853, 2001.

[4] Y. Huang, "Logic gates and computation from assembled nanowire building blocks," *Science*, no. 294, pp. 1313 - 1317, 2001.

[5] P. D. Tougaw and C. S. Lent, "Logical devices implemented using quantum cellular automata," *Journal of Applied Physics*, no. 75, pp. 1818 - 1825, 1994.

[6] Y. Chen, G. Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. Fraser Stoddart, and R. S. Williams, "Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, no. 14, pp. 462 - 468, Apr. 2003.

[7] D. Whang, S. Jin, Y. Wu and C. M. Lieber, "Large-scale hierarchical organization of nanowire arrays for integrated nanosystems," *Nanoletters*, vol. 3, no. 9, pp. 1255 - 1259, Sep. 2003.

[8] Chen He, Margarida F. Jacome and Gustavo de Veciana, "A reconfiguration-based defect-tolerant design paradigm for nanotechnologies," *IEEE Design and Test of Computers*, pp. 316 - 326, July-August 2005.

[9] John von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, pp. 43 - 98, Priceton University Press, 1956.

[10] William H. Pierce, *Failure-tolerant computer design*. Academic Press, 1965.

[11] Jie Han, Jianbo Gao, Yan Qi, Pieter Jonker and Jose A. B. Fortes, "Toward hardware-redundant, fault-tolerant logic for nanoelectronics," *IEEE Design and Test of Computers*, pp. 328 - 339 , July-August 2005.

[12] S. Spagocci and T. Fountain, "Fault rates in nanochip devices," *Proceedings of Electrochemical Society*, vol. 98, no. 19, pp. 582 - 593, 1999.

[13] Darshan D. Thaker, Francois Impens, Isaac L. Chuang, Rajeevan Amirtharajah and Frederic T. Chong, "Recursive TMR: Scaling fault tolerance in the nanoscale era," *IEEE Design and Test of Computers*, pp. 298 - 305, July-August 2005.

[14] J. G. Tryon, *Quadded logic redundancy techniques for computing systems*. pp. 205 - 228, Spartan Books, 1962.

[15] P. A. Jensen, "Quadded NOR logic," *IEEE Transactions on Reliability*, vol. 12, no. 3, pp. 22 - 31, 1963.

[16] J. R. Heath et al., "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, no. 5370, pp. 1716 - 1721, Jun. 1998.

[17] M. Mishra and S. C. Goldstein, "Defect tolerance at the end of roadmap," *Proceedings of International Test Conference*, pp. 1201 - 1211, 2003.

[18] W. B. Culbertson et al., "Defect tolerance on the Teramac custom computer," *Proceedings of IEEE Symposium on FPGA-Based Custom Computing Machines*, pp. 116 - 123, 1997.

[19] J. Huang, M. B. Tahoori, and F. Lombardi, "On the defect tolerance of nanoscale two-dimensional crossbars," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance*, pp. 96 - 104, 2004.

[20] Yan Qi, Jianbo Gao, and Jose A. B. Fortes,"Markov chains and probabilistic computation - A general framework for multiplexed nanoelectronic systems," *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, pp. 194 - 205, Mar. 2005.

[21] Jie Han and Pieter Jonker, "A defect and fault-tolerant architecture for nanocomputers," *Nanotechnology*, vol. 14, pp. 224 - 230, Jan. 2003.

[22] A. S. Sadek, K. Nikoliae, and M. Forshaw, "Parallel information and computation with restitution for noise-tolerant nanoscale logic networks," *Nanotechnology*, vol. 15, pp. 192 - 210, Jan. 2004.

[23] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays," *Journal of Franklin Institute*, vol. 262, pp. 191 - 197, Oct. 1956.

[24] J. J. Suran, "Use of circuit redundancy to increase system reliability," *Proceedings of International Solid-State Circuits Conference*, pp. 82 - 83, Feb. 1964.

[25] M. B. Tahoori, "Application-independent defect tolerance of reconfigurable nanoarchitectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 2, no. 3, pp. 197 - 218, Jul. 2006.

[26] T. Hogg and G. S. Snider, "Defect-tolerant adder circuits with nanoscale crossbars," *IEEE Transactions on Nanotechnology*, vol. 5, no. 2, pp. 97 - 100, Mar. 2006.

[27] T. Hogg and G. S. Snider, "Defect-tolerant logic with nanoscale crossbar circuits," *HP Labs Technical Report*, May 2004.

[28] H. Naeimi and A. DeHon, "A greedy algorithm for tolerating defective crosspoints in NanoPLA design," *Proceedings of International Conference on Field-Programmable Technology*, pp. 49 - 56, 2004.

[29] M. B. Tahoori, "Defects, yield and design in sublithographic nanoelectronics," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005.

[30] M. B. Tahoori, "A mapping algorithm for defect-tolerance of reconfigurable nano-architectures," *Proceedings of International Conference on Computer Aided Design*, 2005.

[31] A. H. El-Maleh, B. M. Al-Hashimi and A. Al-Yamani, "N^2-transistor structure for defect-tolerance at the nanoscale," *Proceedings of European Test Symposium*, 2007.

[32] A. H. El-Maleh, B. M. Al-Hashimi and Aissa Melouki, "Transistor-level based defect-tolerance for reliable nanoelectronics," *Proceedings of Arab International Conference on Computer Systems and Applications*, 2008.

[33] P. Hazucha and C. Svensson, "Cosmic-ray soft error rate characterization of a standard 0.6-m CMOS Process," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 10, pp. 1422 - 1429, Oct. 2000.

[34] Jie Han, "Fault-tolerant architectures for nanoelectronic and quantum devices," *Doctoral Dissertation, Delft University of Technology*, 2004.

[35] Israel Koren and C. Mani Krishna, *Fault-tolerant systems*. pp. 20 - 21, Morgan Kaufmann Publishers, 2007.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. pp. 643 - 693, McGraw-Hill, 2001.

[37] G. Snider, P. J. Kuekes, and R. S.Williams, "CMOS-like logic in defective, nanoscale crossbars," *Nanotechnology*, vol. 15, pp. 881 - 891, 2004.

[38] Chong Zhao, Sujit Dey, and Xiaoliang Bai, "Soft-spot analysis: Targeting compound noise effects in nanometer circuits," *IEEE Design and Test of Computers*, pp. 362 - 375, July-August 2005.

[39] G. Asadi, S. G. Miremadi, H. R. Zarandi, A. Ejlali, "Evaluation of fault-tolerant designs implemented on SRAM-based FPGAs," *10th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 327 - 332, Mar. 2004.

[40] Samudrala, P. J. Ramos, and S. Katkoori, "Selective triple modular redundancy based single-event upset tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957 - 2969, Oct. 2004.

[41] F. L. Kastensmidt, L. Sterpone, L. Carro, M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," *Proceedings of Design, Automation and Test in Europe*, pp. 1290 - 1295, Vol. 2, 2005.

[42] A. Tiwari and K. A. Tomko, "Enhanced reliability of finite-state machines in FPGA through efficient fault detection and correction," *IEEE Transactions on Reliability*, vol. 54, no. 3, pp. 459 - 467, Sept. 2005.

[43] L. Sterpone, M. S. Reorda, M. Violante, "RoRA: A reliability-oriented place and route algorithm for SRAM-based FPGAs," *Research in Microelectronics and Electronics*, vol. 1, pp. 173 - 176, Jul. 2005.

[44] J. Huang, M. B. Tahoori, F. Lombardi, "Probabilistic analysis of fault tolerance of FPGA switch block array," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, Apr. 2004.

[45] J. Huang, M. B. Tahoori, F. Lombardi, "Fault tolerance of switch blocks and switch block arrays in FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 7, pp. 794 - 807, July 2005.

[46] A. Yu, G. Lemieux, "Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement," *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 255 - 262, Aug. 2005.

[47] F. G. Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs," *IEEE Design and Test of Computers*, vol. 21, no. 6, pp. 552 - 562, Nov-Dec 2004.

[48] A. Yu, G. Lemieux, "FPGA defect tolerance: Impact of granularity," *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 189 - 196, Dec. 2005.

[49] Michael Lee Bushnell and Vishwani D. Agrawal, *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Springer, 2000.

[50] Miron Abramovici, Melvin A. Breuer and Arthur D. Friedman, *Digital systems testing and testable design*. IEEE Press, 1990.

[51] Fernanda Lima Kastensmidt, Luigi Carro and Ricardo Reis, *Fault tolerance techniques for SRAM-based FPGAs*. Springer, 2006.

[52] R. Katz, "An SEU-hard flip-flop for antifuse FPGAs," *Proceedings of International Conference On Military And Aerospace Applications Of Programmable Logic Devices*, 2001.

[53] L. Anghel, D. Alexandrescu and M. Nicolaidis, "Evaluation of a soft error tolerance technique based on time and/or space redundancy," *Proceedings of International Symposium on Integrated Circuits and Systems Design*, 2000.

[54] D. Alexandrescu, L. Anghel and M. Nicolaidis, "New methods for evaluating the impact of single event transients in VDSM ICs," *Proceedings of IEEE International Symposium On Defect and Fault Tolerance in VLSI Systems*, pp. 99 - 107, 2002.

[55] Martin L. Shooman, *Reliability of computer systems and networks: Fault tolerance, analysis and design*. John Wiley, 2002.

[56] J. Leavy, "Upset due to a single particle caused propagated transient in a bulk CMOS microprocessor," *IEEE Transactions on Nuclear Science*, vol. 38, no. 9, pp. 1493 - 1499, Dec. 1991.

[57] J. Hass, "Mitigating single event upsets from combinational logic," *Proceedings of NASA Symposium on VLSI Design*, 1998.

[58] J. Hass, "Probabilistic estimates of upset caused by single event transients," *Proceedings of NASA Symposium on VLSI Design*, 1999.

[59] Kartik Mohanram, "Simulation of transients caused by single-event upsets in combinational logic," *Proceedings of International Test Conference*, pp. 1 - 9, 2005.

[60] M. Nicolaidis and R. Perez, "Measuring the width of transient pulses induced by radiation," *Proceedigs of IEEE International Reliability Physics Symposium*, pp. 56 - 59, IEEE Computer Society, 2003.

[61] P. E. Dodd and L. W. Massengill, "Basic mechanism and modeling of single-event upset in digital microelectronics," *IEEE Transactions on Nuclear Science*, vol. 50, pp. 583 - 602, June 2003.

[62] A. V. Ferris-Prabhu, *Introduction to semiconductor device yield modeling*. Artech House, 1992.

[63] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems, design and evaluation*. Digital Press, 2nd edition, 1992.

[64] A. Avizienis, "Design of fault-tolerant computers," *Proceedings of 1967 Fall Joint Computer Conference of AFIPS*, pp. 733 - 743, 1967.

[65] I. R. Committee, "Executive Summary," *International Technology Roadmap for Semiconductors*, 2003. http://public.itrs.net.

[66] P. K. Lala, *Self-checking and fault-tolerant digital design*. Morgan Kaufmann Academic Press, 1992.

[67] M. Chean and J. Fortes, "A taxonomy of reconfiguration techniques for fault-tolerant processor arrays," *IEEE Computer*, vol. 23, no. 1, pp. 55 - 69, Jan. 1990.

[68] L. Chen and A. Avizienis, "N-version programming: A fault tolerance approach to reliability of software operation," *Digest of the 8th International Symposium on Fault-Tolerant Computing*, pp. 3 - 9, 1978.

[69] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," *Proceedings of International Conference on Computer-Aided Design*, pp. 283 - 289, Nov. 1998.

[70] Mihir R. Choudhury and Kartik Mohanram, "Accurate and scalable reliability analysis of logic circuits," *Proceedings of Design Automation and Test in Europe*, pp. 1454 - 1459, 2007.

[71] Asbjorn Djupdal, "Evolving static hardware redundancy for defect-tolerant FPGAs," *Doctoral Dissertation, Norwegian University of Science and Technology*, April 2008.

[72] Morten Hartmann, "Evolution of fault and noise-tolerant digital circuits," *Doctoral Dissertation, Norwegian University of Science and Technology*, April 2005.

[73] Xilinx San Jose, CA., "EasyPath solutions, 2005" *http://www.xilinx.com/products/easypath/.*

[74] W. J. Huang and E. J. McCluskey, "Column-based precompiled configuration technique for FPGA fault tolerance," *Proceedings of IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 137 - 146, 2001.

[75] J. Lach, W. H. Mangione-Smith and M. Potkonjak, "Efficiently supporting fault-tolerance in FPGAs," *Proceedings of ACM International Symposium on FPGAs*, pp. 105 - 115, 1998.

[76] A. Doumar, S. Kaneko, and H. Ito, "Defect and fault tolerance FPGAs by shifting the configuration data," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 377 - 385, 1999.

[77] F. Hatori, T. Sakurai, et al., "Introducing redundancy in FPGAs," *Proceedings of Custom Integrated Circuits Conference*, pp. 7.1.1-7.1.4, 1999.

[78] "Altera Corporation," *United States Patents 6,034,536, 6,166,559, 6,337,578, 6,344,755, 6,600,337 and 6,759,871*, 20002004.

[79] A. Doumar and H. Ito, "Design of switching blocks tolerating defects/faults in FPGA interconnection resources," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 134 - 142, 2000.

[80] G. Asadi and M. B. Tahoori, "Soft error rate estimation and mitigation for SRAM-based FPGAs," *Proceedings of ACM International Symposium on FPGAs*, pp. 149 - 160, 2005.

[81] S. Hareland, "Impact of CMOS process scaling and SOI on the soft error rates of logic processes," *IEEE Nuclear and Space Radiation Effects Conference*, pp. 73 - 74, 2001.

[82] M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving stars: An integrated approach to on-line testing, diagnosis and fault tolerance for FPGAs," *Proceedings of NASA/DoD Workshop on Evolvable Hardware*, 2001.

[83] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *Xilinx Application Notes XAPP216 (v1.0)*, 2000.

[84] Y. Lin, J. Appenzeller, J. Knoch, and P. Avouris, "High-performance carbon nanotube FET with tunable polarities," *IEEE Transactions on Nanotechnology*, vol. 4, no. 5, pp. 481 - 489, Sep. 2005.

[85] A. Coker, V. Taylor, D. Bhaduri, S. Shukla, A. Raychowdhury and K. Roy, "Multijuction fault tolerance architecture for nanoscale crossbar memories," *IEEE Transactions on Nanotechnology*, vol. 7, no. 2, pp. 202 - 208, Mar. 2008.

[86] S. Goldstein and M. Budiu, "NanoFabrics: Spatial computing using molecular electronics," *Proceedings of International Symposium on Computer Architecture*, pp. 178 - 189, 2001.

[87] Andre DeHon and M. J. Wilson, "Nanowire-based sublithographic programmable logic arrays," *Proceedings of ACM International Symposium on FPGAs*, pp. 123 - 132, Feb. 2004.

[88] Nantero Inc., "http://www.nantero.com", 2005.

[89] L. P. Cordella et al., "An improved algorithm for matching large graphs," *Proceedings of the 3rd IAPR-TC-15 International Workshop on Graph-Based Representations*, pp. 149 - 159, 2001.

[90] Yasser El-Sonbaty and M. A. Ismail, "A graph-decomposition algorithm for graph optimal momomorphism," *Proceedings of the 8th British Machine Vision Conference*, 1997.

[91] SIVALab. "VF graph matching library. University of Naples Federico II", 2001.

[92] Fran Hanchek and Shantanu Dutt, "Methodologies for tolerating cell and interconnect faults in FPGAs," *IEEE Transactions on Computers*, vol. 47, no. 1, pp. 15 - 33, Jan. 1998.

[93] V. Betz and J. Rose, "VPR: A new packaging, placement and routing tool for FPGA research," *International Workshop on Field-Programmable Logic and Applications*, pp. 213 - 222, 1997.

[94] E. M. Sentovich et al., "SIS: A system for sequential circuit analysis," *Technical Report No. UCB/ERL M92/41, University of California, Berkeley*, 1992.

[95] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on CAD*, pp. 1 - 12, Jan. 1994.

[96] V. Betz, "VPR and T-VPack user manual (Version 4.30)," *ECE Department, University of Toronto*, Mar. 2000.

[97] L. Comtet, *Advanced combinatorics: The art of finite and infinite expansions.* Revised & Enlarged Edition, Reidel Publishing Co., Dordrecht, Netherland, 1974.

[98] Robert K. Brayton et al., *Logic minimization algorithms for VLSI synthesis.* Kluwer Academic Publishers, 1984.

[99] A. Z. M. Almassri, "Design for defect-tolerant reliable digital systems at the nanoscale," *Master's Thesis, Department of Computer Engineering, KFUPM*, June 2009.

Vitae

- Farhan Khan
- Born in Hyderabad, Pakistan on Junauary 8, 1981
- Received Bachelor of Engineering (B.E.) in Computer Systems from N.E.D University of Engineering and Technology, Karachi, Pakistan in February 2003.
- Received Master of Engineering (M.Engg.) in Computer Systems from N.E.D University of Engineering and Technology, Karachi, Pakistan in June 2006.
- Joined King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia as a Research Assistant in September 2006.
- Completed Master of Science (M.S.) in Computer Engineering in June 2009.
- Email: *farhankhan43@yahoo.com*
- Present Address: Room 212, Bldg. 903, KFUPM, Dhahran 31261, Saudi Arabia.
- Permanent Address: 4-B Staff Colony, Public School, Unit No. 2, Latifabad, Hyderabad 71800, Pakistan.