UNIVERSITY OF SOUTHAMPTON

Defect and Fault Tolerance Techniques for Nano-Electronics

by

Aissa Melouki

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

April 2011

UNIVERSITY OF SOUTHAMPTON
Department of Electronics and Computer Science

Doctor of Philosophy

Defect and Fault Tolerance Techniques for Nano-Electronics

ABSTRACT

Aissa Melouki

Nanotechnology-based devices are believed to be the future possible alternative to CMOS-based devices. It is predicted that the high integration density offered by emerging nanotechnologies will be accompanied by high manufacturing defect rates and high operation-time fault rates. This thesis is concerned with developing defect and fault tolerance techniques to address low manufacturing yield due to permanent defects and reduced computational reliability due to transient faults projected in nanoscale devices and nanometre CMOS circuits.

The described research makes four key contributions. The first contribution is a novel defect tolerance technique to improve the manufacturing yield of nanometre CMOS logic circuits. The technique is based on replacing each transistor by an N^2-transistor structure ($N \geq 2$) that guarantees defect tolerance of all $(N-1)$ defects. The targeted defects include stuck-open, stuck-short and bridging defects. Extensive simulation results using ISCAS benchmark circuits, show that the proposed technique achieves manufacturing yield higher than recently proposed techniques and at a reduced area overhead.

The second contribution is two new repair techniques, named Tagged Replacement and Modified Tagged Replacement, to improve the manufacturing yield of nanoscale crossbars implementing logic circuits as look-up tables (LUTs). The techniques are based on highly efficient repair algorithms that improve yield by increasing the resolution of repair. Simulation results show that the proposed techniques are able to provide higher levels of defect tolerance and have lower redundancy requirements than recently reported techniques. Another popular crossbar-based circuit implementation is nanoscale programmable logic arrays (PLAs). The third contribution is a probabilistic defect tolerance design flow that improves the manufacturing yield of nanoscale PLAs and significantly reduces post-fabrication test and diagnosis time. This is achieved by limiting defect diagnosis to the nanowire level rather than the crosspoint level as in previously proposed graph-based techniques.

The final contribution involves improving both manufacturing yield and computational reliability of nanoscale crossbars implementing logic circuits as LUTs. This is achieved by combining Hamming and Bose-Chaudhuri-Hocquenghem (BCH) codes together or with N-Modular Redundancy and Bad Line Exclusion techniques. Simulation results show a significant improvement in fault tolerance by the proposed techniques (targeting fault rates upto 20%) when compared to previously reported single coding schemes.

Contents

List of Acronyms xii

Acknowledgements xv

1 Introduction 1
- 1.1 Crossbar Architecture . . . 2
  - 1.1.1 Nanoscale LUT Architecture . . . 3
  - 1.1.2 Nanoscale PLA Architecture . . . 4
- 1.2 Permanent and Transient Faults . . . 7
  - 1.2.1 Permanent Defects in Nanometre CMOS . . . 7
  - 1.2.2 Permanent and Transient Faults in Nanoscale Devices . . . 8
- 1.3 Defect Tolerance and Fault Tolerance Background . . . 9
- 1.4 Thesis Organisation . . . 12
- 1.5 Contributions . . . 13

2 Literature Review 15
- 2.1 Fault Tolerance and Defect Avoidance Techniques . . . 15
- 2.2 Reconfiguration . . . 16
  - 2.2.1 Static Reconfiguration . . . 16
  - 2.2.2 Dynamic Reconfiguration . . . 19
  - 2.2.3 Hybrid Fault Tolerance Technique . . . 20
- 2.3 Hybrid Nano/CMOS Architecture . . . 21
- 2.4 Defect Tolerance Techniques for Nanometre CMOS . . . 24
- 2.5 Motivation and Objectives . . . 25

3 Nanometre CMOS Defect Tolerance Technique 28
- 3.1 Overview of the N^2-Transistor Structure . . . 29
- 3.2 Quadded Transistor Structure . . . 29
  - 3.2.1 Theoretical Analysis . . . 31
- 3.3 Nonuple Transistor Structure . . . 35
  - 3.3.1 Theoretical Analysis . . . 36
- 3.4 Other Defect Tolerance Techniques . . . 40
  - 3.4.1 Triple Modular Redundancy (TMR) . . . 41
  - 3.4.2 Triplicated Interwoven Redundancy (TIR) . . . 43
  - 3.4.3 Quadded Logic . . . 44
- 3.5 Experimental Results . . . 45
  - 3.5.1 Stuck-Open and Stuck-Short Defect Analysis . . . 46
  - 3.5.2 Bridging Fault Analysis . . . 50

3.6 Conclusion . . . 56

4 Repair Techniques for Nano/CMOS Architecture 57
4.1 Introduction . . . 58
4.2 Repair Most Technique . . . 61
4.2.1 Theoretical Analysis of Failure Rate (Repair Most Technique) . . . 63
4.2.2 Effect of Don't Cares . . . 65
4.3 Proposed Tagged Replacement Technique . . . 67
4.3.1 Theoretical Analysis of Failure Rate (Tagged Replacement Technique) . . . 70
4.4 Proposed Modified Tagged Replacement Technique . . . 73
4.4.1 Theoretical Analysis of Failure Rate (Modified Tagged Replacement Technique) . . . 79
4.5 Yield Analysis of Large Circuits . . . 80
4.6 Estimation of Repair Cost . . . 82
4.7 Effect of Defect Distribution Model . . . 83
4.8 Conclusion . . . 85

5 Defect Tolerance Design Flow for Nanoscale PLA Architecture 86
5.1 Introduction . . . 87
5.2 Previous Work . . . 89
5.3 Proposed DT Design Flow . . . 90
5.4 Motivational Example . . . 93
5.5 Nanowire Defects . . . 95
5.5.1 Proposed Probabilistic DT Technique for Nanowire Defects . . . 95
5.5.2 Experimental Results (Nanowire DT Technique) . . . 98
5.6 N-bit Adder Circuit Implementation . . . 98
5.7 Open-Junction Defects . . . 102
5.7.1 Open-Junction Defect Masking in Nanoscale PLA . . . 102
5.7.2 Proposed Probabilistic DT Technique for Open-Junctions . . . 104
5.7.3 Yield Optimisation . . . 107
5.7.4 Experimental Results (Open-Junction DT Technique) . . . 107
5.8 Choice of Defect Rates P_{wire} and P_{oj} . . . 109
5.9 Conclusion . . . 112

6 Coding-based Fault Tolerance for Nano/CMOS Architecture 114
6.1 Introduction . . . 115
6.2 Failure Rate Calculation . . . 116
6.3 Error Correcting Codes (Related Work) . . . 117
6.3.1 Hamming Code . . . 118
6.3.2 Bose-Chaudhuri-Hocquenghem (BCH) Code . . . 121
6.4 N-Modular Redundancy (Related Work) . . . 122
6.5 Proposed Hybrid Techniques . . . 124
6.6 Combined Two-Dimensional Coding: Technique 1 . . . 125
6.6.1 Hamming - BCH . . . 125
6.6.2 Hamming - Systematic BCH with Check Bits . . . 126
6.7 N-Modular Redundancy with ECC: Technique 2 . . . 132
6.7.1 NMR with Hamming . . . 134

6.7.2 TMR with BCH . . . 135
6.8 Bad Line Exclusion with ECC: Technique 3 . . . 138
6.9 Overheads of the Proposed Fault Tolerance Techniques . . . 142
6.10 Conclusion . . . 145

7 Conclusions and Future Work **147**
7.1 Thesis Contributions . . . 148
7.2 Future Work . . . 150

A Tools and Benchmark Circuits **151**
A.1 Software Tools . . . 151
A.2 ISCAS Benchmark Circuits . . . 151
A.3 Other Benchmark Circuits . . . 153

B Error Correcting Codes **158**
B.1 Error Correcting Codes . . . 158
B.2 Hamming Code . . . 159
B.3 Bose-Chaudhuri-Hocquenghem (BCH) Code . . . 160
B.4 BCH Codec Synthesis Tool . . . 161

Bibliography **163**

List of Figures

1.1 Molecular crossbar consists of two sets of orthogonal nanowires 3
1.2 Diode-based crossbar forming look-up table (LUT) circuit 4
1.3 1-bit full adder mapped to a diode-based crossbar. (a) circuit schematic (b) adder look-up table (c) LUT-based implementation 5
1.4 Implementing logic function $X = A + BC$ using diode crossbar and resistors 5
1.5 Generalised PLA architecture 6
1.6 PLA-based implementation of a 1-bit full adder 6
1.7 Illustration of stuck-at faults 7
1.8 Illustration of bridging faults 8
1.9 Non-deterministic fabrication methods may lead to (a) perfectly aligned mesh of wires, (b) alignment problems, or (c), (d) defective components . 9
1.10 Defect and fault tolerance techniques are necessary to achieve acceptable yields and computational reliabilities 10
1.11 The computational architectures targeted in the thesis 11

2.1 Static reconfiguration - Hierarchy of design abstractions [He et al., 2005a,b] 17
2.2 Static reconfiguration - A representative set of basic flows [He et al., 2005a,b] 17
2.3 Static reconfiguration - (a) four TMR testing tiles (b) two feasible configurations of a basic flow [He et al., 2005a,b] 18
2.4 Gate-level redundancy to enhance yield in nanometre CMOS circuits [Sirisantana et al., 2004] 24
2.5 Gate level redundancy with switches to enhance yield in nanometre CMOS circuits [Kothe et al., 2006] 25
2.6 Architecture of the defect-tolerant CMOS gate [Ashouei et al., 2008] . . . 26

3.1 Defect-tolerant N^2-transistor structure 29
3.2 DTQT transistor level diagram (a) Transistor in original gate implementation (b) DTQT structure implementing logic function $(A + A).(A + A)$ (c) DTQT structure implementing logic function $(A.A) + (A.A)$ 30
3.3 NAND gate (a) Normal (b) DTQT 30
3.4 Quadded transistor structure - circuit failure rate obtained both theoretically and experimentally based on simulations of some of ISCAS circuits 34
3.5 Yield comparison between quadded-transistor structure (Q) and complementary CMOS 35
3.6 Nonuple structure transistor level diagram 36
3.7 Nonuple transistor structure - circuit failure rate obtained both theoretically and experimentally based on simulations for some of ISCAS circuits 39
3.8 Yield comparison between quadded-transistor structure (Q), nonuple-transistor structure (N) and complementary CMOS 40

3.9 TMR architecture (a) Original circuit (b) TMR circuit 42
3.10 TMR architecture implemented at the logic gate level 42
3.11 TIR architecture (a) Original circuit (b) TIR circuit 43
3.12 Quadded Logic architecture: (a) Original circuit, (b) Quadded logic circuit 45
3.13 Failure rate of ISCAS c880 implemented using DTQT due to stuck-at faults 47
3.14 Failure rates of c1355 implemented using DTQT, TMR and Quadded structures due to stuck-at faults 47
3.15 Comparison of area overhead between DTQT, Quadded Logic and TMR . 48
3.16 Comparison of circuit failure probability for an 8-stage cascaded half-adder circuit for stuck-open and stuck short defects 49
3.17 Failure rate of ISCAS c880 implemented using DTQT due to bridging defects 53
3.18 Failure rates of c1355 implemented using DTQT and Quadded structures due to bridging faults 55

4.1 Proposed defect-unaware design flow 59
4.2 Implementation of LUT based Boolean logic approach using nano/CMOS architecture 60
4.3 Illustration of Repair Most technique 61
4.4 Repair Most technique (a) Implementation for LUT based nano/CMOS architecture (b) Repair mechanism by setting value of $c_{th} = 2$ and $r_{th} = 0$. The columns (rows) with number of defects $> c_{th}$ (r_{th}) are excluded . . 62
4.5 Repair Most - The order of excluding rows and columns significantly affects the yield 62
4.6 Repair Most - Failure rate obtained both theoretically and experimentally 64
4.7 Repair Most - Effect of varying c_{th} on failure rate 64
4.8 Repair Most - Impact of Don't Cares on failure rate 66
4.9 Repair Most - Failure rate obtained both theoretically and experimentally in the presence of Don't Cares 66
4.10 Tagged Replacement Technique: Implementation of a $2^N \times N$ LUT using 1-bit CMOS tags 68
4.11 Comparison between Repair Most and Tagged Replacement techniques . . 69
4.12 Plot of Failure rate Vs Defect rate using Tagged Repair technique for different LUT sizes with (a) 25% redundancy (b) 50% redundancy and (c) 100% redundancy in rows and columns 70
4.13 Tagged Replacement - Defect-free bits in columns should be aligned to ensure successful instantiation of LUTs 71
4.14 Tagged Replacement - Failure rate obtained both by theory and simulation for a $2^4 \times 4$ LUT and 100% redundancy 73
4.15 Tagged Replacement - Impact of Don't Cares on failure rate 73
4.16 Illustration of the principle of the Modified Tagged Replacement technique 74
4.17 Modified Tagged Replacement Technique (a) Allocated $(2^N + r_{sp}) \times (N + c_{sp})$ nanofabric for the implementation of the $2^N \times N$ LUT (b) Implementation of the LUT using Modified Tagged Replacement technique in the case of $\alpha = 1$ 75
4.18 Modified Tagged Replacement Technique (a) $\alpha = 0$ (Tagged Replacement technique) (b) $\alpha = 1$ (c) $\alpha = 2$ 76
4.19 Failure rate comparative study for $2^4 \times 4$ LUT with 100% redundancy . . 77

4.20 Plot of Failure rate Vs Defect rate using Modified Tagged Repair technique ($\alpha = 1$) for different LUT sizes with (a) 25% redundancy (b) 50% redundancy and (c) 100% redundancy in rows and columns 78
4.21 Modified Tagged Replacement - Failure rate vs. Defect rate for different values of α 78
4.22 Modified Tagged Replacement ($\alpha = 1$) - Effect of Don't Cares on failure rate 80
4.23 Failure probability of synthesised ISCAS'85 benchmark circuits using Modified Tagged Replacement technique ($\alpha = 1$) 81
4.24 Defect distribution in nanofabrics: (a) random defect distribution (b) clustered defect distribution using Gaussian distribution function (c) row/-column defect distribution. Defect rate is assumed to be 10% in a 100×100 crossbar 84
4.25 Effect of defect distribution on defect tolerance for a $2^4 \times 4$ LUT with 100% redundancy using Modified Tagged Repair technique ($\alpha = 1$) under (a) random defect distribution (b) Gaussian/clustered defect distribution (c) Row/Column defect distribution 85

5.1 An illustration of the possible physical defects in nanoscale PLAs 88
5.2 Defect-aware design flow [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004] 91
5.3 Proposed design flow 92
5.4 Defect-aware DT technique [Hogg and Snider, 2007] (a) Creating circuit graph where nanowires and active junctions are represented by nodes and edges respectively (b) Creating crossbar graph where defective junctions and broken nanowires are represented by removing their edges (c) Graph monomorphism is searched between circuit and crossbar graphs for successful implementation 94
5.5 Proposed DT technique (a) Original circuit (b) Defective crossbar (c) Mapping circuit onto defective crossbar where redundant nanowires are used to compensate for the missing variables and product terms caused by defective junctions and broken nanowires 95
5.6 Implementation of a $N \times (I + O)$ PLA with N_{sp} spare product wires, I_{sp} spare input wires and O_{sp} spare output wires 96
5.7 Schematic of a 3-bit adder 100
5.8 (a) A multiple-stage (MS) N-bit adder implemented as a ripple-carry logic chain of 1-bit adders. (b) PLA implementation of a multiple-stage 3-bit adder 100
5.9 (a) A single-stage (SS) N-bit adder. (b) PLA implementation of a single-stage 3-bit adder 101
5.10 A multiple-stage 3-bit adder implemented using three equally-sized nano-blocks (MSB) 102
5.11 Different levels of fault masking of function $f = ab + cd$ in nanoscale PLA architecture 104
5.12 Open junction defects leading to erroneous output of logic functions implemented in nanoscale PLAs. (a) Original function $F = AB + C$ (b) Wrong logic function $F1 = AB$ (c) Wrong logic function $F2 = A + C$ . . . 106
5.13 Yield optimisation by programming more junctions without affecting the logical formulas of output wires 108

5.14 Illustration of open-junction defect rates distribution across fabricated nanoscale PLAs . . . 112

6.1 Targeted hybrid nano/CMOS architecture overview . . . 116
6.2 Hamming - Failure rate vs. Error rate for various LUT sizes . . . 119
6.3 Hamming - Failure rate obtained theoretically and experimentally . . . 120
6.4 Hamming - Effect of don't cares on failure rate . . . 121
6.5 BCH - Padding . . . 121
6.6 BCH vs. Hamming . . . 122
6.7 Row Coding Hamming vs. Column Coding BCH . . . 123
6.8 Triple Modular Redundancy (TMR) . . . 123
6.9 TMR - Failure rate vs. Error rate for different LUT sizes . . . 124
6.10 2D Coding (a) Hamming & BCH (b) Hamming & Systematic BCH with check bits - $2^4 \times 4$ LUT . . . 126
6.11 Failure rate comparison between 1D and 2D coding techniques . . . 127
6.12 Flow chart to illustrate the multiple-step decoding process of the proposed 2D coding technique . . . 128
6.13 2D Coding - Effect of varying LUT size on failure rate . . . 129
6.14 2D Coding - Failure rate obtained both theoretically and experimentally . 131
6.15 2D Coding - Effect of Don't Cares on Failure rate . . . 131
6.16 2D Coding - Failure rate obtained theoretically and experimentally in the presence of 50% DCCs . . . 132
6.17 TMR & Hamming - Failure rate obtained theoretically and experimentally 134
6.18 Combining TMR with Hamming helps reduce the number of errors per codeword before decoding . . . 135
6.19 TMR & Hamming - Impact of don't cares on circuit failure rate . . . 137
6.20 Comparison between the failure probabilities of the investigated fault tolerant techniques . . . 137
6.21 Hamming with Bad Line Exclusion . . . 138
6.22 Hamming & Bad Line Exclusion - Failure rate obtained for different percentages of spare rows . . . 139
6.23 Percentage of Redundancy needed to achieve 0% failure rate for different LUT sizes . . . 140
6.24 Hamming & Bad Line Exclusion - Variation of failure rate in the presence of Dont Care entries. Inclusion of 50% DCCs improves fault tolerance by almost two times . . . 141
6.25 Hamming & Bad Line Exclusion - Failure rate obtained both theoretically and experimentally in the presence of 25% spares and 50% DCCs . . . 142
6.26 Time Multiplexing Strategy . . . 145

A.1 ISCAS c17 - Original schematic . . . 153
A.2 ISCAS c17 - Equivalent schematic . . . 153
A.3 ISCAS c17 - Single-stage PLA implementation . . . 154
A.4 Schematic of a 3-bit multiplier circuit . . . 154
A.5 Multiple-stage PLA implementation of a 3-bit multiplier . . . 155
A.6 1-bit ALU: (a) circuit schematic (b) output depends on control signals S_0 and S_1 . . . 156
A.7 Schematic of a 4-bit ALU circuit . . . 156

A.8 Multiple-stage PLA implementation of a 4-bit ALU 157

B.1 Binary BCH decoder structure 161
B.2 File structure of the BCH Codec Synthesis tool 162

List of Tables

3.1 Only one of the four possible pairs of defective trasistors in DTQT structure produces an error. . . . 31
3.2 Comparison of circuit failure probability between quadded transistor structure and quadded logic approaches for stuck-open and stuck-short defects 51
3.3 Comparison of yield between quadded transistor structure and quadded logic approaches for stuck-open and stuck-short defects . . . 52
3.4 Circuit failure probability of ISCAS circuits implemented using nonuple transistor structure due to stuck-open and stuck-short defects . . . 53
3.5 Comparison of circuit failure probability between quadded-transistor structure and quadded logic approaches for bridging defects . . . 54
3.6 Circuit failure probability of ISCAS circuits implemented using nonuple transistor structure due to bridging defects . . . 56

4.1 Repair Most - The required amount of spares and c_{th} to achieve failure rates less than 5% . . . 65
4.2 Tagged Replacement technique - Redundancy required to achieve failure rates less than 5% . . . 70
4.3 Modified Tagged Replacement technique- Failure rate obtained for different LUT sizes and different values of α . . . 79
4.4 ISCAS'85 benchmark circuits synthesised into smaller $2^N \times N$ LUTs . . . 81
4.5 Comparative repair cost of the proposed techniques with the Repair Most technique in terms of targeted defect rate . . . 82

5.1 Targeted vs. actual yields obtained using the proposed probabilistic nanowire DT technique . . . 99
5.2 Manufacturing yields when no DT technique is applied to nanoscale PLAs 102
5.3 Stuck-open defect manifestation in nanoscale PLA . . . 103
5.4 Targeted vs. actual yields obtained using the proposed probabilistic open-junction DT technique . . . 110
5.5 Targeted vs. actual yields obtained using the proposed probabilistic open-junction DT technique for various benchmark PLAs . . . 111

6.1 Summary of the proposed techniques . . . 125
6.2 Hamming decoder - Distribution of erroneous bits in the output word - $2^4 \times 4$ LUT . . . 132
6.3 NMR & ECC - Distribution of errors in $2^4 \times 4$ LUTs . . . 136
6.4 Area, Delay and Energy overheads of CMOS Components Assuming a $0.12\mu m$ CMOS technology, $2^4 \times 4$ LUT and a Clock Frequency of $25MHz$ 143

6.5 Expected Area, Delay and Energy Overheads of CMOS Components at Future 32nm CMOS Technology 143
6.6 Area/Useful bit of the proposed techniques 144

A.1 ISCAS'85 benchmark suite 152
A.2 ISCAS'89 benchmark suite 152

List of Acronyms

2D	Two Dimensional
ATPG	Automatic Test Pattern Generator
BCH	Bose Chaudhuri Hocquenghem Code
CMOL	Molecular CMOS
CMOS	Complementary Metal Oxide Semiconductor
CNT	Carbon Nanotube
CONAN	COnfigurable Nanostructures for reliAble Nano electronics
CTMR	Cascaded Triple Modular Redundancy
DCC	Don't Care Condition
DT	Defect Tolerance
DTNT	Defect Tolerant Nonuple Transistor
DTQT	Defect Tolerant Quadded Transistor
ECC	Error Correcting Code
FPGA	Field-Programmable Gate Array
FPNI	Field-Programmable Nanowire Interconnect
FT	Fault Tolerance
GND	Ground voltage
LUT	Look-Up Table
nm	nanometer
MRF	Markov Random Field
MS	Multiple-Stage
MSB	Multiple-Stage Multipe-Block
μJ	micro Joule
μs	micro Second
MU	Mapping Unit
μW	micro Watt
NMR	N-Modular Redundancy
NRAM	Carbon Nanotube RAM
NT	Nonuple Transistor
NW	Nanoscale Wire
P	Probability
PE	Processing Element

PLA	Programmable Logic Array
QCA	Quantum-dot Cellular Automata
QL	Quadded Logic
QMR	Quintuple-Modular Redundancy
QT	Quadded Transistor
R	Reliability
RAM	Random Access Memory
SE	Switching Element
SEC-DEC	Single Error Correcting and Double Error Detecting Code
SiNW	Silicon NanoWire
SS	Single-Stage
TIR	Triple Interwoven Redundancy
TMR	Triple Modular Redundancy
Tb	Terabits
V_{dd}	Supply voltage
VHDL	Very high speed integrated circuit Hardware Description Language
Y	Yield

Declaration of Authorship

I, *Aissa Melouki*, declare that this thesis entitled:

Defect and Fault Tolerance Techniques for Nano-Electronics

and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done during the period of candidature of the intended qualification at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as listed in page 13, Section 1.5 (Chapter 1).

Signed:

Date:

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Bashir M. Al-Hashimi, for the privilege of working under his supervision. I am deeply indebted to him for his constant support and guidance throughout this PhD project. I would also like to thank him for his help during the preparation of this thesis and all my publications. Dr Saket Srivastava is also to be thanked for his valuable contribution to this research. Dr Aimane El-Maleh from King Fahd University in Saudi Arabia also deserves thanks for his valuable time helping me with this research. I am thankful to Dr. Tom Kazmierski and Dr Koushik Maharatna for their constructive comments during my nine month and MPhil transfer examinations. I am also grateful to the Algerian Ministry of Higher Education and Scientific Research for funding this project.

I would like to thank my fellow mates in the lab, Syed Saqib Khursheed, Amit Acharyya, Mustafa Imran Ali, Rishad A. Shafik, Sheng Yang, Dafong Zhou, Urban Ingelsson, Karim El Shabrawy, Hamed Shahidipour, Anton Kulakov, Evangelos Mazomenos, Shida Zhong and Ghaithaa Manla for their support in both technical and non-technical discussions.

Finally, I am thankful to my parents, brothers (Abd El-Latif, Arbi, Lakhdar, Ibrahim, Abd El-Aziz and Khaled) and friends (Nadjib Mammeri, Ahmed Maache, Samir Rihani, Issam Maamria, Issam Souilah, Hamza A. Rouabah, Abdeldjalil Belouettar and Zakaria Mihoubi) for their encouragement and support which helped me greatly throughout this PhD programme.

To my beloved brother Oussama Abd El-Hafid...

Chapter 1

Introduction

The continuous decrease in transistor feature size has been pushing the CMOS technology to its physical limits caused by ultra-thin gate oxides, short channel effects and doping fluctuations across the chip [Tehranipoor, 2007, Tahoori, 2005a, Sirisantana et al., 2004]. Photolithography techniques used today are fast approaching their limits that will make it impossible to scale them further [Tehranipoor, 2007, Mishra and Goldstein, 2003]. Furthermore, as CMOS devices are scaled down into the nanoscale regime ($<$ 100nm), new challenges are being introduced including low fabrication yield and computational reliability.

To be able to continue the size and speed improvement trends according to Moore's Law, where the number of transistors doubles every 18 months, research investments are growing on a wide range of other emerging devices besides CMOS. Various new technologies have been suggested including Quantum-dot Cellular Automata [Lent et al., 1994, Huang et al., 2007] and chemically assembled nanotechnology [Brown and Blanton, 2007, Mishra and Goldstein, 2003, Copen Goldstein and Budiu, 2001] which uses bottom-up self-assembly and self-alignment techniques [DeHon et al., 2005] to construct circuits using nanometre scale devices such as Silicon NanoWires [Cui and Lieber, 2001, Huang et al., 2001] and Carbon Nanotubes [Bachtold et al., 2001]. All these nanotechnologies hold the ultimate promise of producing feature sizes of 20nm or less and thus extremely high device densities of upto 10^{12} devices/cm^2 [Nikolic et al., 2002, Mishra and Goldstein, 2003, Ziegler and Stan, 2003, Tahoori, 2005b, He et al., 2005b, Rao et al., 2006]. Due to the fabrication regularity imposed by the chemical self-assembly and self-alignment process, it is projected that only regular structures such as the two-dimensional crossbar, outlined in Section 1.1, can be manufactured [Huang et al., 2004, Tahoori, 2005b].

One significant disadvantage of chemically assembled nanotechnology-based fabrication is that it is likely to have significantly higher defect densities than CMOS technology and therefore an increase in yield loss [Mishra and Goldstein, 2003, Brown and Blanton, 2007, Copen Goldstein and Budiu, 2001, Jacorne et al., 2004, Garcia and Orailoglu,

2008]. With defect rates exceeding 10%, as is being predicted in logic circuits based on nanoscale devices [Wang and Chakrabarty, 2007, Stan et al., 2003, Mishra and Goldstein, 2003, Brown and Blanton, 2007], the manufacturing cost can be prohibitively high and discarding a defective chip will no longer be possible. Devices and interconnect at the nanoscale level will be much more susceptible to transient faults [Nikolic et al., 2002, Jacorne et al., 2004, Nepal et al., 2006] which poses a reliability problem. Furthermore, CMOS-based design methodologies and tools take high reliability for granted, applying such methods and tools directly to logic design based on nanoscale devices is expected to lead to exceedingly low yields [Jacorne et al., 2004, He et al., 2005b]. Therefore, a paradigm shift in design methods is necessary, placing defect and fault tolerance at the forefront which is the main focus of this thesis.

The aim of this chapter is to provide preliminary information for the subsequent chapters in the thesis. It is organised as follows. Section 1.1 gives an overview of the crossbar architecture and describes crossbar-based implementation of logic circuits using look-up table (LUT) and programmable logic array (PLA) representations since they will be used in later chapters. Section 1.2 outlines the projected permanent and transient faults in nanometre CMOS and nanoscale devices. The necessity of defect and fault tolerance techniques in nanocircuit designs is highlighted in Section 1.3. The contribution of each chapter is summarized in Section 1.4, and finally Section 1.5 presents the list of publications generated from the work presented in this thesis.

1.1 Crossbar Architecture

The crossbar architecture is a general approach for molecular circuits [Stan et al., 2003, Hogg and Snider, 2007, Brown and Blanton, 2007, Ziegler and Stan, 2002b, 2003, Tahoori, 2005b, Wang and Chakrabarty, 2007, Bhaduri et al., July, 2005, Chen et al., 2003b, Copen Goldstein and Budiu, 2001]. Fig. 1.1 shows a generic nanoscale two-dimensional crossbar architecture that consists of a lower plane of nanowires crossed perpendicularly by an upper plane of nanowires. Both planes consist of parallel and uniformly-spaced nanometre-sized wires such as carbon nanotubes or silicon nanowires. The region where the two perpendicular wires cross each other is called a junction or a crosspoint. Depending on the nature of the nanowires, junctions can be configured to implement either a resistor or a diode [Hogg and Snider, 2007, Ziegler and Stan, 2002b, 2003]. In both resistor-based and diode-based crossbars, each crosspoint acts as a reprogrammable switch that can be programmed by applying a higher voltage to the corresponding horizontal and vertical wires which causes the resistance of the junction to change [Ziegler and Stan, 2002b, 2003, Tahoori, 2005b]. In the resistor-based crossbar, the two-terminal crosspoint can be electrically configured to behave as a low-resistance device or as a high-resistance device. This device can be switched to a low-resistance state (the on-state) by applying an appropriate positive voltage bias.

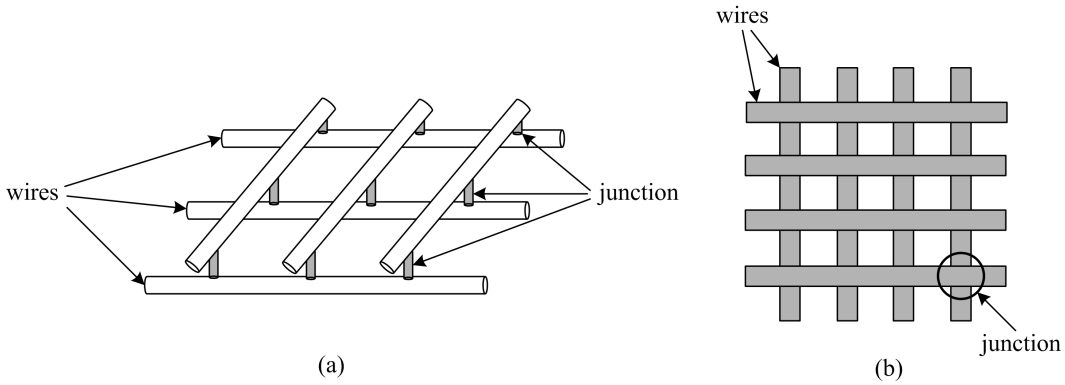


FIGURE 1.1: Molecular crossbar consists of two sets of orthogonal nanowires

Likewise, the device can be switched to a high-resistance state (the off-state) by applying an appropriate reverse voltage bias. If a junction is remained unconfigured, its corresponding crossing wires do not interact electrically. Similarly, in the diode-based crossbars, the diode-like crosspoints are programmed by applying a high voltage to the perpendicular nanowires [Tahoori, 2005b, DeHon et al., 2005, Naeimi and DeHon, 2007, Copen Goldstein and Budiu, 2001]. Because the crosspoints are reprogrammable, their reconfiguration aspect can be exploited in the testing and self-diagnosis of nanoscale crossbar-based architectures [Mishra and Goldstein, 2003, Copen Goldstein and Budiu, 2001, Tahoori, 2005a, Huang et al., 2004].

1.1.1 Nanoscale LUT Architecture

Because of the regular structure of nanoscale devices and their high density as compared to current lithography-based circuits [Hogg and Snider, 2007], fabricating ultra-large memory systems is regarded as a leading target application of the nanoscale crossbar architecture [DeHon et al., 2005, Chen et al., 2003a, Jeffery et al., 2004, Naeimi and DeHon, 2007, Li and Zhang, 2009]. By using each crosspoint as an active memory cell, crossbar circuits can be operated as rewritable memory system with a density of 6.4 Gbits/cm^2 [Chen et al., 2003a, Wang and Chakrabarty, 2007]. In [Ziegler and Stan, 2002b, 2003, Singh et al., 2007, Paul et al., 2007, Rad and Tehranipoor, 2006a, Ma et al., 2008], the authors proposed a memory-based logic implementation of logic circuits. By combining an address decoder with a diode-based crossbar forming the memory array, logic circuits can be implemented assuming look-up table (LUT) representation of logic functions, as illustrated in Fig. 1.2. In an LUT-based implementation of logic functions, the inputs to the function are decoded to generate an address that is used to access a particular location in the memory array. This memory location contains either '1' or '0' depending on whether the function being implemented is '1' or '0' for the given input combination [Paul et al., 2007]. An example of the LUT-based logic implementation is shown in Fig. 1.3(c) where a 1-bit full adder is mapped to the nanoscale crossbar.

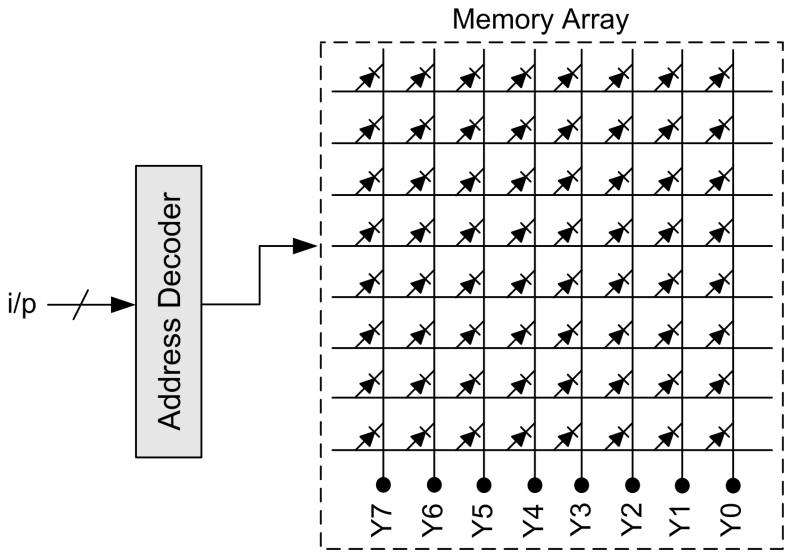
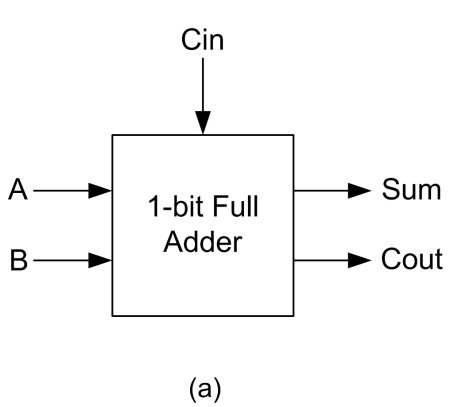


FIGURE 1.2: Diode-based crossbar forming look-up table (LUT) circuit

1.1.2 Nanoscale PLA Architecture

Another implementation of logic functions that have been addressed in [Ziegler and Stan, 2002b, 2003, Hogg and Snider, 2007, Brown and Blanton, 2007, Wang and Chakrabarty, 2007, Joshi and Al-Assadi, 2007, Chakraborty et al., 2008, Rad and Tehranipoor, 2006b, Copen Goldstein and Budiu, 2001] is the programmable logic array (PLA) representation. The crossbar architecture can be set to evaluate any logical formula expressed as a combination of AND and OR operations. Fig. 1.4 shows an example where the function $X = A + BC$ is implemented as a PLA structure. The output X is connected to the ground through a resistor and via a diode junction to the second and third vertical wires. If both vertical wires are at low voltage (i.e. OFF), then the output X will also be off due to its connection to ground. On the other hand, if either of the connected vertical wires is at high voltage (i.e. ON), the diode connection from the high voltage vertical wire will give a high voltage to the output wire. This is because the resistor connecting the output wire to ground is much bigger than the diode resistance in the forward direction. If 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. Therefore, 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 AND.

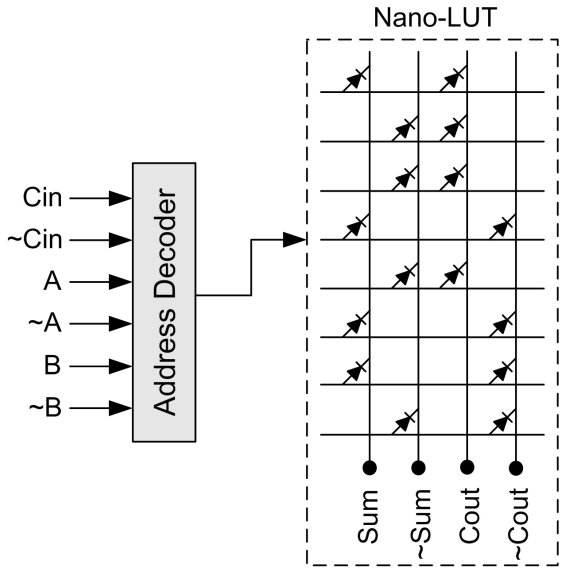
The configuration method suggested in [Ziegler and Stan, 2002b, 2003, Hogg and Snider, 2007, Brown and Blanton, 2007, Joshi and Al-Assadi, 2007] can be generalised to map any logic function whose output is the sum (logical OR) of products (logical AND). As shown in Fig. 1.5, a PLA has a set of input lines $I_1 ... I_l$, a set of product lines $P_1 ... P_m$, and a set of output lines $O_1 ... O_n$. Logic values are applied to the input lines and based on the connections between the input lines and product lines, the logic AND function



(a)

Cin	A	B	Sum	Cout
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

(b)



(c)

FIGURE 1.3: 1-bit full adder mapped to a diode-based crossbar. (a) circuit schematic (b) adder look-up table (c) LUT-based implementation

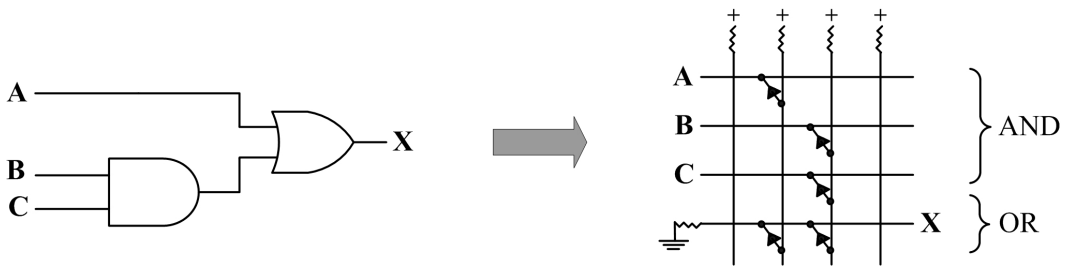


FIGURE 1.4: Implementing logic function $X = A + BC$ using diode crossbar and resistors

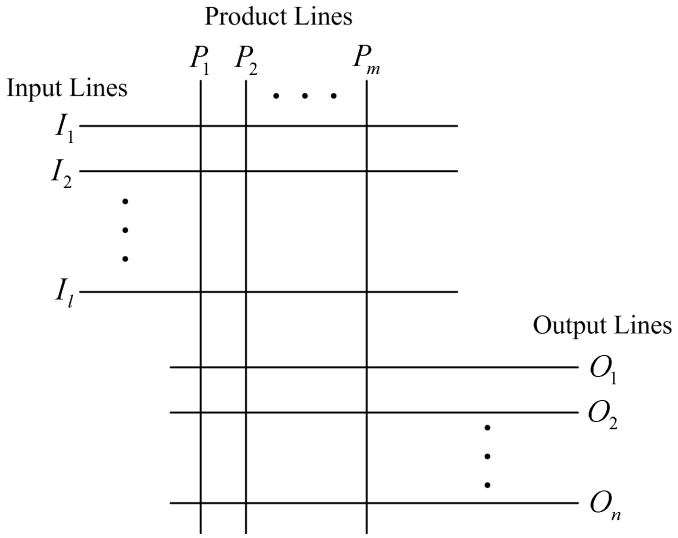


FIGURE 1.5: Generalised PLA architecture

is performed on the input lines connected to a particular product line. For example, if connections were made from the input lines I_1 and I_2 to product line P_1, the value of P_1 would equal $I_1.I_2$. The set of connections between input lines and product lines is therefore known as the *AND plane*. Similarly, based on the connections between product lines and output lines, the logic OR function is performed on the product lines connected to a particular output line. For instance, if product lines P_1 and P_2 are connected to output line O_1, the value of O_1 equals $P_1 + P_2$. The set of connections between product lines and output lines is therefore known as the *OR plane*. The PLA-based implementation of the 1-bit full adder shown in Fig. 1.3(a) is presented in Fig. 1.6 where: $Sum = C_{in}AB + C_{in}\bar{A}\bar{B} + \bar{C_{in}}A\bar{B} + \bar{C_{in}}\bar{A}B$ and $C_{out} = C_{in}A + C_{in}\bar{A}B + \bar{C_{in}}AB$.

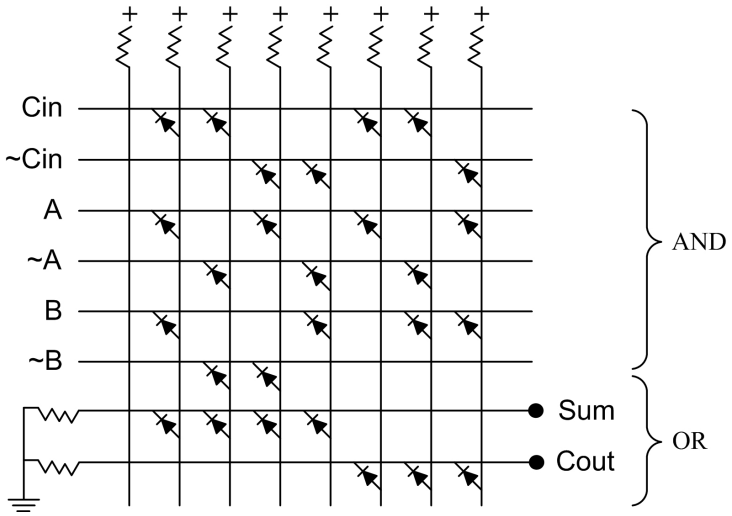


FIGURE 1.6: PLA-based implementation of a 1-bit full adder

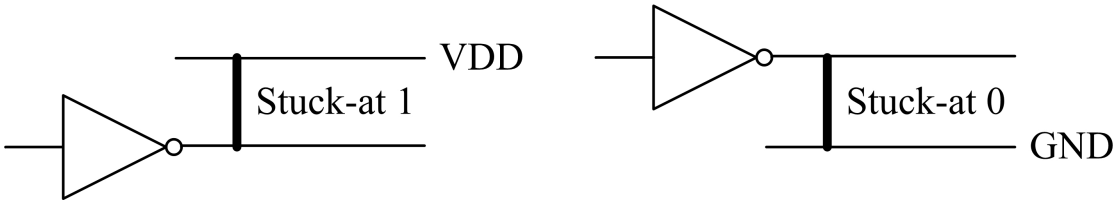


FIGURE 1.7: Illustration of stuck-at faults

1.2 Permanent and Transient Faults

The defect and fault tolerance techniques presented later in the thesis target permanent defects and transient faults in nanometre CMOS and nanoscale devices. A brief description of these defects and faults is given in this section.

1.2.1 Permanent Defects in Nanometre CMOS

Silicon CMOS devices that are scaled beyond the 100nm feature sizes are expected to exhibit high rates of manufacturing defects [Sirisantana et al., 2004] including stuck-open, stuck-short and bridging defects which are mainly caused by the high integration density.

Stuck-Open and Stuck-Short Defects

In deep submicron CMOS, some physical defects can unintentionally cause a logic signal to be connected to one of the power rails i.e. V_{dd} or GND, forcing the logic node to be clamped at the voltage of the rail causing a stuck-at fault [Abramovici et al., 1990]. This defect is referred to as stuck-open (or stuck-at 0) in case the node is connected to the ground rail and it is referred to as stuck-short (or stuck-at 1) if a node is clamped to V_{dd} as illustrated in Fig. 1.7.

Bridging Defects

Bridging faults [Maly, 1987, Ferguson and Shen, 1988, Dalpasso et al., 1993, Polian et al., 2005] represent another major class of physical defects in deep submicron CMOS. They are formed during manufacturing process by a redundant metal connecting two nodes of a design with one another thereby deviating the circuit behaviour from ideal as illustrated in Fig. 1.8. Bridging defects can be classified into intra-gate and inter-gate defects [Nakura et al., 2009, Fan et al., 2006]. Intra-gate bridging defect is due to redundant metal inside a logic gate for example between gate and drain of two transistors of a NOR gate. On the other hand, inter-gate bridging defect is due to a redundant metal between two wires of different logic gates.

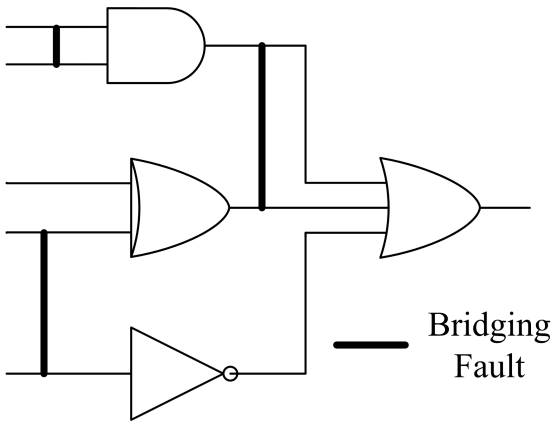


FIGURE 1.8: Illustration of bridging faults

1.2.2 Permanent and Transient Faults in Nanoscale Devices

High manufacturing defect rate is one of the significant issues of the emerging chemically assembled nanotechnology devices and their defect density will be considerably higher than the defect density in current CMOS [Garcia and Orailoglu, 2008, Wang and Chakrabarty, 2007, Jacorne et al., 2004, Copen Goldstein and Budiu, 2001]. Unlike traditional CMOS technology, chemically assembled nanotechnology involves self-assembly and self-alignment methods. This approach is most effective at creating regular structures such as the crossbar structure. However, like any manufacturing process, the fabrication of chemically assembled nanotechnology devices is imperfect. Unlike the ideal mesh of wires (shown in Fig. 1.9(a)) expected in a crossbar architecture, a nanoblock may have serious problems with spacing and alignment of wires (Fig. 1.9(b)) that can cause defective components (Fig. 1.9(c), Fig. 1.9(d)) [Brown and Blanton, 2007, Stan et al., 2003]. The nanofabrication process employs nanowires which are a few atoms long in the diameter. Hence, broken nanowires are expected to be fairly common in nanocircuits. Furthermore, the contact area between nanowires contains only a few tens of atoms. With such small cross-section and contact areas, fragility of these devices is orders of magnitude more than devices currently being fabricated using conventional lithography techniques, resulting in higher junction defect rates [Tahoori, 2005a]. According to [DeHon and Naeimi, 2005, Huang et al., 2004, Naeimi and DeHon, 2004, Garcia and Orailoglu, 2008, Zheng and Huang, 2009, Sun and Zhang, 2007], physical defects in nanoscale devices are split into two categories: defects in nanowires and defects in programmable crosspoints.

- **Nanowire defect**: Nanowire defects are caused by broken wires that cannot conduct current properly from one end of the wire to the other [DeHon and Naeimi, 2005].
- **Crosspoint defect**: A defect in a crosspoint can lead to either a non-programmable stuck-open junction or a non-programmable stuck-closed junction.

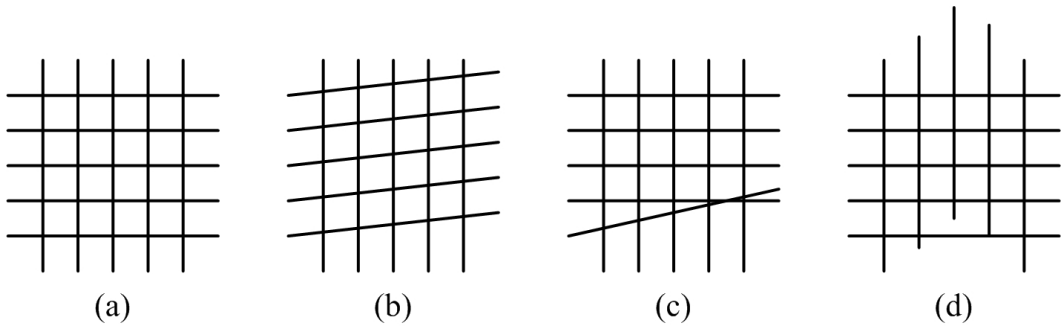


FIGURE 1.9: Non-deterministic fabrication methods may lead to (a) perfectly aligned mesh of wires, (b) alignment problems, or (c), (d) defective components

Stuck-open junction: A programmable junction can be switched between the on- and off-states; however a non-programmable open junction cannot be programmed into the on-state (i.e. inability to activate the crosspoint to make a diode connection).

Stuck-closed junction: Also known as stuck-short junction defect. A non-programmable closed-junction cannot be programmed into the off-state.

In addition to manufacturing defects, nanoscale devices will also be vulnerable to high transient fault rates [He et al., 2005b, Jacorne et al., 2004, Nepal et al., 2006, Zhao et al., 2005, Bahar, 2006]. This is because, according to the 2001 International Technology Roadmap for Semiconductors [ITRS, 2001], with feature sizes shrinking to nanometre scale and clock frequencies reaching the multi GHz level, effects of various noise sources are becoming stronger than ever. Furthermore, to reduce dynamic power dissipation, supply voltage V_{dd} is aggressively scaled down to the subvoltage range. The resulting reduction in noise margin will expose computation in nanosystems to higher transient fault rates [Nepal et al., 2006, Bahar, 2006, Zhao et al., 2005]. Nanoscale devices are also vulnerable to radiation effects that can cause single-event upsets which in turn can introduce logical faults in nanoscale circuits [Thaker et al., 2005, Zhao et al., 2005].

1.3 Defect Tolerance and Fault Tolerance Background

Designing circuits with nanometre CMOS or nanoscale devices imposes significant manufacturing yield and computational reliability challenges. New yield and reliability improvement techniques that account for component unreliability are necessary to make such technologies commercially viable. At present, there is a significant research effort worldwide to devise novel system architectures and design paradigms that can effectively address the high defect and fault rates intrinsic to nanotechnologies (nanometre CMOS, nanoscale devices). In this thesis, the issue of enhancing the manufacturing yield and computational reliability of nanometre CMOS and nanoscale devices is addressed through the use of defect and fault tolerance techniques.

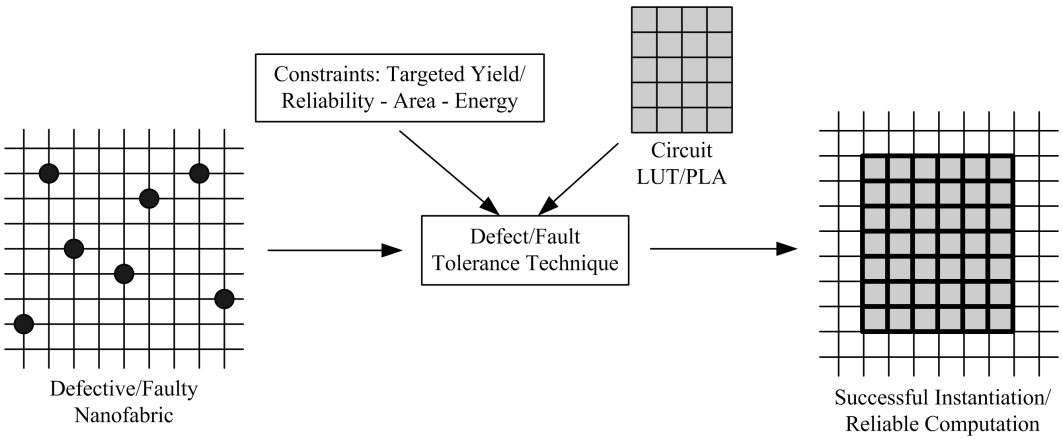


FIGURE 1.10: Defect and fault tolerance techniques are necessary to achieve acceptable yields and computational reliabilities

Fig. 1.10 illustrates the need for defect and fault tolerance techniques to combat the numerous permanent and transient faults in future nanofabrics. However, the techniques to be used should minimise the overheads incurred to achieve the necessary yield and reliability and to meet the design constraints such as the area and energy overheads. Defect and fault tolerance dictates the use of allocated redundant resources to compensate for the defective and faulty components and hence achieving high yields and reliability. The main objective of the thesis is to devise highly efficient defect and fault tolerance techniques that are capable of targeting higher defect and fault rates than the previously reported techniques while minimising area and energy overheads.

Below are the definitions of the parameters used in the evaluation of the proposed techniques.

Defect Tolerance: is the ability of a circuit to operate perfectly in the presence of faulty components that result from physical defects.

Manufacturing Yield: is the probability of successfully instantiating a circuit onto a defective fabric with a certain physical defect rate.

$$Yield = \frac{Total\ repairable\ fabrics}{Total\ number\ of\ fabricated\ fabrics} \tag{1.1}$$

Fault Tolerance: is the ability of a circuit to operate perfectly in the presence of faulty components that result from both physical defects and transient faults.

Reliability: is the probability that a fabric, with a certain defect rate, can successfully implement a circuit and the ability of this circuit to operate perfectly in the presence of transient faults during operation lifetime.

$$Reliability = \frac{Total\ correctly\ functioning\ fabrics}{Total\ number\ of\ fabrics} \tag{1.2}$$

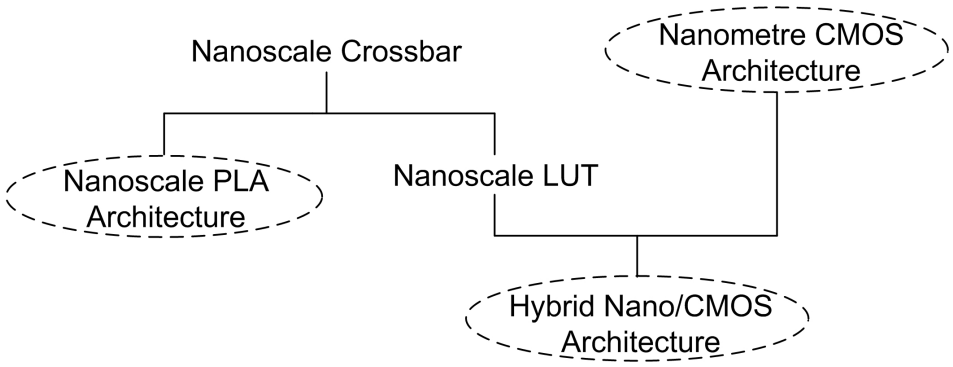


FIGURE 1.11: The computational architectures targeted in the thesis

Targeted Defect (Fault) Rate: is the defect (fault) rate range within which a certain defect (fault) tolerance technique is capable of maintaining its defect (fault) tolerance below a certain rate.

Defect Mapping: is the post-fabrication test and diagnosis overhead required by certain defect tolerance techniques to locate permanent defects in order to avoid them. This parameter is used in Chapter 5 to compare the proposed design flow with the previously proposed techniques.

Targeted Architectures

The defect and fault tolerance techniques proposed in the thesis target various computational architectures which are listed below:

- Nanometre CMOS architecture (Chapter 3)
- Nanoscale PLA architecture (Chapter 5)
- Hybrid Nano/CMOS architecture (Chapters 4 and 6)

The nanometre CMOS architecture uses the static complementary (pull-up, pull-down) CMOS implementation of logic circuits. In the nanoscale PLA architecture, logic circuits are implemented as nanoscale PLAs, as outlined in Section 1.1.2, using nanoscale devices only. However, in hybrid Nano/CMOS architecture, the high performance but unreliable nanoscale devices are complemented with the low performance but more reliable nanometre CMOS. Nanoscale crossbars implementing logic circuits as LUTs, as outlined in Section 1.1.1, are combined with nanometre CMOS components that are used to implement the highly critical functions in a circuit. To facilitate the reading of the thesis, Fig. 1.11 summarises the targeted architectures.

1.4 Thesis Organisation

Chapter 2 - Literature Review

This chapter presents an overview of recently reported defect and fault tolerance techniques and computational architectures targeting nanometre technologies. The chapter also outlines the research objectives that are addressed in this thesis to develop highly efficient defect and fault tolerance techniques for nanometre CMOS and nanoscale devices.

Chapter 3 - Nanometre CMOS Defect Tolerance Technique

This chapter presents a new defect tolerance technique targeting the physical defects in nanometre CMOS. It is based on adding redundancy at the transistor level by replacing each transistor by an N^2-transistor structure ($N \geq 2$) that guarantees defect tolerance of all $(N-1)$ defects. Two particular cases of this technique named quadded transistor ($N = 2$) and nonuple transistor ($N = 3$) structures are evaluated in this chapter in terms of tolerating stuck-at and bridging defects.

Chapter 4 - Repair Techniques for Nano/CMOS Architecture

This chapter presents two new repair techniques, called Tagged Replacement and Modified Tagged Replacement techniques, that provide high level of defect tolerance for hybrid nano/CMOS computational architecture implementing logic circuits as LUTs. The targeted physical defects include both junction and nanowire defects. This chapter shows that the proposed techniques are capable of handling defect rates upto 20% at low redundancy overheads unlike the recently proposed repair technique that requires higher redundancy overhead to handle defect rates of only upto 10%.

Chapter 5 - Defect Tolerance Design Flow for Nanoscale PLA Architecture

This chapter presents a novel probabilistic design flow that improves the manufacturing yield of nanoscale crossbars implementing logic circuits as PLAs. It comprises of two defect tolerance techniques that tackle nanowire and junction defects. Unlike in the recently proposed techniques that require extensive crosspoint-by-crosspoint verification prior to mapping of logic functions onto the crossbars, defect tolerance in the proposed design flow is achieved at a reduced testing overhead by limiting defect diagnosis to the nanowire level rather than the crosspoint level. This chapter also investigates the impact of the geometry and the density of active junctions in PLAs on the area overhead of the proposed design flow.

Chapter 6 - Coding-based Fault Tolerance for Nano/CMOS Architecture

This chapter presents three fault tolerance techniques for hybrid nano/CMOS architecture implementing logic circuits as LUTs. These techniques are based on error correcting

codes including Hamming and BCH codes and they integrally address the tolerance of both manufacturing defects and transient faults. The aim of this chapter is to improve both manufacturing yield and computational reliability of nanoscale LUTs. Simulation results in this chapter show a significant improvement in fault tolerance by the proposed techniques (targeting fault rates upto 20%) when compared to the previously reported single coding schemes. Chapter 6 also evaluates the area, latency and energy overheads incurred by the CMOS components.

Chapter 7 - Conclusions and Future Work

This chapter summarizes the contributions presented in this thesis and outlines a number of worthy and important areas for future research.

Brief descriptions of the benchmark circuits and tools used in the experiments referred throughout the thesis are given in Appendix A. More information on the error correcting codes used in Chapter 6 are given in Appendix B.

1.5 Contributions

The contributions of the research work presented in this thesis have been published as follows:

Book Chapter

1. El-Maleh, A.H., Al-Hashimi, B.M., **Melouki, A.**, *Transistor-level based defect tolerance for reliable nanoelectronics*, Springer book "Robust Computing with Nanoscale Devices", Chao Huang (Editor), Pages 29-49, 2010.

Journal Publications

2. El-Maleh, A.H., Al-Hashimi, B.M., **Melouki, A.**, Khan, F., *Defect Tolerant N^2-Transistor Structure for Reliable Nanoelectronic Designs*, IET Computers & Digital Techniques, Vol. 3, No. 6, Pages 570-580, November, 2009.

3. Srivastava, S., **Melouki, A.**, Al-Hashimi, B.M., *Tagged Repair Techniques for Defect Tolerance in Hybrid nano/CMOS Architecture*, IEEE Transactions on Nanotechnology, Vol. PP, No. 99, 2010.

4. **Melouki, A.**, Srivastava, S., Al-Hashimi, B.M., *Fault Tolerance Techniques for Hybrid CMOS/Nano Architecture*, IET Computers & Digital Techniques, Vol. 4, No. 3, Pages 240-250, May, 2010.

5. **Melouki, A.**, Al-Hashimi, B.M., *Probabilistic Defect Tolerance Design Flow for Nanoscale PLA Design*, submitted to IEEE Transactions on Nanotechnology.

Conference Publications

6. El-Maleh, A.H., Al-Hashimi, B.M., **Melouki, A.**, *Transistor-level based defect tolerance for reliable nanoelectronics*, IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), 31st March to 4th April 2008, Doha, Qatar.

7. Srivastava, S., **Melouki, A.**, Al-Hashimi, B.M., *Repair Techniques for Hybrid Nano/CMOS Computational Architecture*, IEEE Conference on Nanotechnology, 26th to 31st July 2009, Genoa, Italy.

8. Srivastava, S., **Melouki, A.**, Al-Hashimi, B.M., *Defect Tolerance in Hybrid Nano/CMOS Architecture using Tagging Mechanism*, IEEE/ACM Symposium on Nanoscale Architectures, 30th to 31st July 2009, San Francisco, USA.

Chapter 2

Literature Review

This chapter outlines the architectures and techniques proposed by researchers in the area of defect and fault tolerance for nano-electronics. The remainder of this chapter is organised as follows. Section 2.1 outlines the reported fault tolerance and defect avoidance techniques. Section 2.2 discusses static and dynamic reconfiguration methodologies and highlights the drawbacks of this approach. Computational architectures which are based on hybrid Nano/CMOS fabrics are presented in Section 2.3. In Section 2.4, recently reported defect tolerance techniques targeting nanometre CMOS are outlined. Section 2.5 provides the motivation for the research carried out in this thesis and outlines the objectives to be addressed in this thesis.

2.1 Fault Tolerance and Defect Avoidance Techniques

Errors in nanoelectronic components are split into permanent and transient errors (Chapter 1, Section 1.2). Several techniques exist for overcoming the effects of erroneous devices and they all use the concept of redundancy. However, some techniques are more efficient for dealing with transient errors (Fault Tolerance techniques) and some are for permanent defects (Defect Tolerance/Avoidance techniques).

Fault tolerance techniques enable the design of highly reliable nanosystems by using hardware redundancy to tolerate permanent and transient errors. Instead of locating the faulty elements in a circuit, their effect on its logical behaviour is masked by the redundant elements. Redundancy can be applied at different levels in the design including system, unit or gate levels. Classical redundancy-based fault tolerance techniques such as N-Modular Redundancy, NAND Multiplexing and Quadded Logic are presented in Chapter 3, Section 3.4. Although there has been a renewed interest in these techniques to target the high error rates in nanoelectronic circuits [Han et al., 2005], some researchers regard these techniques as inefficient due to the high and time varying fault

rates expected in nanotechnology [Rao et al., 2006, Jacorne et al., 2004, He et al., 2005b]. Another category of fault tolerance techniques is Error Correcting Codes. Various coding schemes were proposed in [Singh et al., 2007, Sun and Zhang, 2007, Jeffery et al., 2004, Biswas et al., 2007] as an alternative approach to hardware redundancy to distinguish error-free from erroneous data in hybrid Nano/CMOS memory systems.

Defect tolerance for nanosystem designs is considered essential to obtain acceptable manufacturing yields. Most of the proposed defect tolerance techniques targeting physical defects are based on defect avoidance through reconfiguration and defect mapping [Mishra and Goldstein, 2003, Jacorne et al., 2004, He et al., 2005a,b, He and Jacome, 2007]. First, the defect avoidance scheme requires that the information of the location of defects be obtained through a test process and stored in a defect database called defect map. Then, a feasible configuration realising the application to be implemented is synthesised by the reconfiguration-based defect avoidance technique by circumventing the located defects. Both static and dynamic reconfiguration-based techniques that appeared in the literature review are outlined next.

2.2 Reconfiguration

2.2.1 Static Reconfiguration

A reconfiguration-based defect avoidance methodology for defect-prone nanofabrics was proposed in [Jacorne et al., 2004, Bhaduri et al., July, 2005, He et al., 2005a,b, He and Jacome, 2006, 2007]. It consists of first obtaining a defect map of the targeted nanofabric, and then configuring the desired functionality around the defective components. In this approach, the targeted nanofabric is structured as hierarchies of carefully dimensioned reconfigurable fabric regions, while decomposing and assigning small functional flows to each region. Fig. 2.1 illustrates the three-level hierarchy used to design the nanofabric. The basic configuration unit of the nanofabric, called a region, is a grid consisting of 8 processing elements (PEs) and 8 switching elements (SEs). Each region of the nanofabric can be configured to execute a small behavioural segment, called a basic flow. A set of representative basic flows is shown in Fig 2.2, where each node represents an arithmetic/logic operation to be executed by a PE and edges represent data transfers between operations performed by the SEs. In [He et al., 2005a,b, He and Jacome, 2007], the authors proposed a nanoscale crossbar-based implementation of PEs by implementing the various operations as LUTs, as outlined in Chapter 1, Section 1.1.1.

From Fig. 2.1, it can be observed that the larger the allocated redundant resources within a region, the larger the number of alternative configurations for its associated basic flow will be, and hence the higher the probability of successfully instantiating it on that region. However, to achieve a sufficiently high probability of successful configuration in

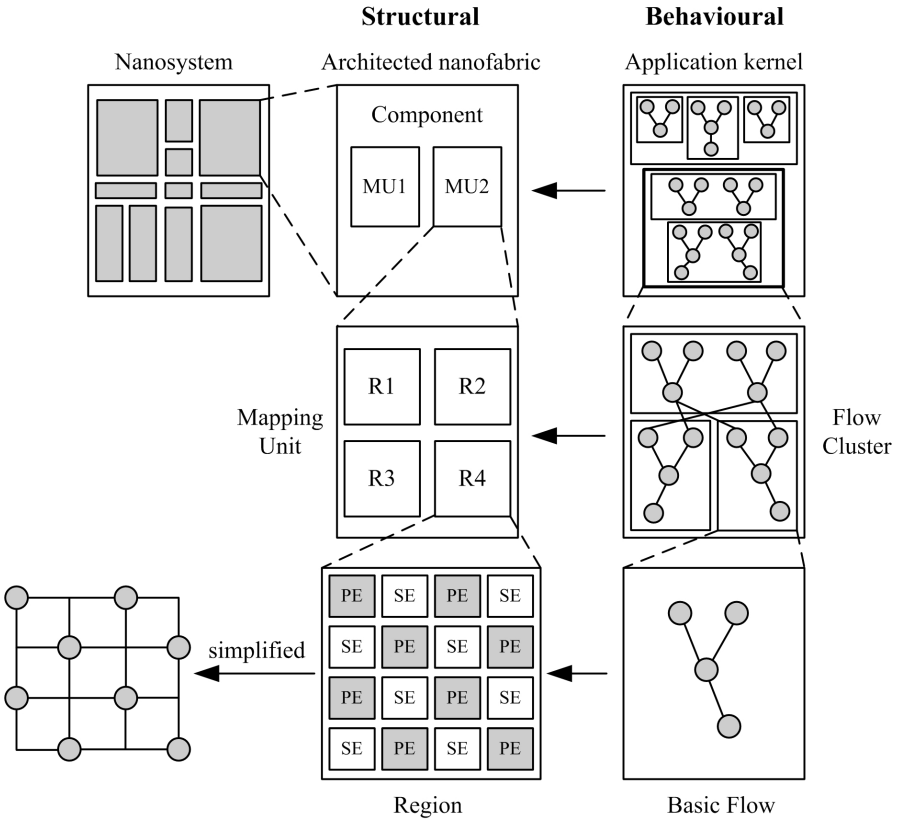


FIGURE 2.1: Static reconfiguration - Hierarchy of design abstractions [He et al., 2005a,b]

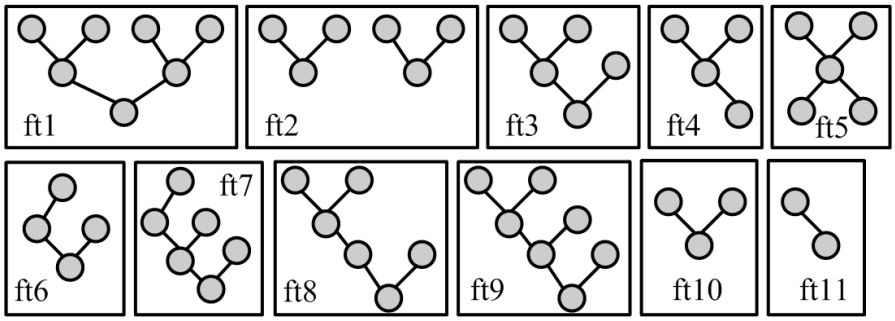


FIGURE 2.2: Static reconfiguration - A representative set of basic flows [He et al., 2005a,b]

a scalable way, an additional hierarchal level was introduced in the fabric architecture, denoted mapping unit (MU), as illustrated in Fig. 2.1. A flow cluster comprising of m basic flows can be instantiated in a MU containing n regions, where $m \leq n$. Thus, the mapping unit abstraction creates a second level of redundancy. Finally, the MUs are grouped together to form a component that is capable of implementing an application kernel. For example, the application kernel shown in Fig. 2.1 comprises of two MUs, each with four regions. The top three flows of the kernel are assigned to MU1 and the bottom three flows are assigned to MU2.

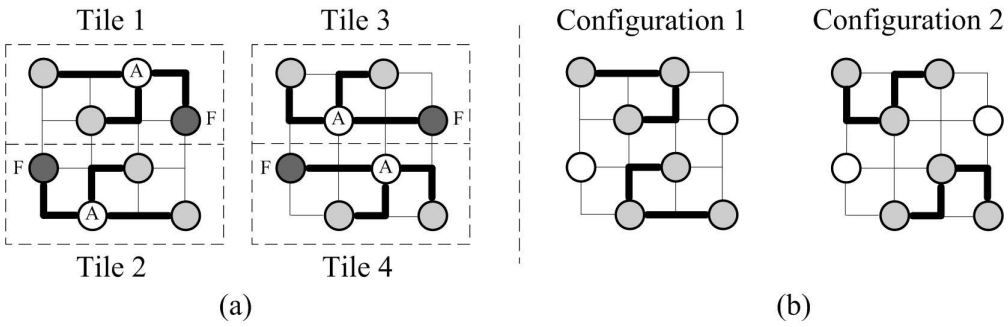


FIGURE 2.3: Static reconfiguration - (a) four TMR testing tiles (b) two feasible configurations of a basic flow [He et al., 2005a,b]

Routing among the elements in the hierarchy shown in Fig. 2.1 is supported as follows: within a region, the switching elements are used to route between adjacent PEs. At the MU level, each region is surrounded by a routing track on each of its sides and a switch block is placed between the routing tracks. Because there is a small number of tracks within a MU, and MUs contain a small number of internal regions, efficient look-up table-based algorithms can be used to explicitly program the switch blocks with the shortest paths between any two regions. At the component level, inter-MU routing uses long-lines for signals that are run next to the MUs and switch boxes for routing between MUs. The routing strategy is similar to that used for intra-MU routing [Jacorne et al., 2004, He et al., 2005b].

When designing a reconfigurable nanofabric to implement the components of a given nanosystem and targeting a specific manufacturing yield, first, the redundant resources required by each component needs to be allocated i.e. determine the number of MUs and the number of regions within each MU, as well as the number of routing tracks forming the interconnect structure. The kernel to be executed by each such component also needs to be decomposed into a number of basic flows. Finally, subsets of the basic flows have to be assigned to the component's MUs, so as to achieve the optimum performance, i.e. the best average component latency over all nanofabric chips, while meeting the targeted yield [He and Jacome, 2006, 2007].

The defective components and/or connectivity within a region are systematically identified by a suite of pre-specified test tiles. A test tile corresponds to configuring and operating a set of PEs to perform a function whose output allows the detection of possible defects. Tiles implementing a Triple Modular Redundancy (TMR) configuration were proposed in [Jacorne et al., 2004, He et al., 2005a,b]. Each tile includes four processing elements, where one plays the role of an arbiter for the outputs of the other three. Such small tiles can be configured systematically, and will usually permit locating faulty PEs or connections. A TMR testing tile can identify a faulty PE/connectivity if the arbiter and two other PEs/connections are operational. Fig. 2.3 shows four such tiles with the arbiters labelled 'A'.

Another reconfiguration-based defect avoidance approach was proposed in [Mishra and Goldstein, 2003, Copen Goldstein and Budiu, 2001, Goldstein et al., 2003]. First, this approach maps defects on a large reconfigurable grid of components, each of which is a mesh of active crosspoints that can be configured as a logic gate or circuit. Then, a feasible configuration realising the desired circuit for each nanofabric is obtained by avoiding the defects. The proposed defect mapping technique is based on a testing algorithm that consists of two phases: a probability assignment phase and a defect location phase. In each phase, a set of fabric components are configured to act as tester circuits. In the probability assignment phase, the outputs of the tester circuits are analysed and each component in the fabric is assigned a probability of being defective. The components with very high defect probability are excluded in this phase. In the defect location phase, the remaining components are rigorously tested to pin-point the defect-free ones and eliminate the errors introduced by the first phase.

Despite the fact that reconfiguration-based defect-avoidance techniques offer a promising solution to achieve acceptable levels of defect tolerance, there are major difficulties associated with using the defect-map strategy in mass production of nanoscale devices. For instance, due to the random nature of defect occurrence, there will be different defect maps for different nanofabrics. Locating the physical defects using different testing algorithms will certainly be very time consuming for highly dense nanoscale fabrics and this will pose a major bottleneck in the manufacturing process [Tahoori, 2005a, Rad and Tehranipoor, 2006b]. Another issue related to the defect map-based strategies is that the defect map size will be too large and for reliability reasons, it will be prohibitively expensive to store it in a reliable CMOS scale memory [Tahoori, 2005a, Rad and Tehranipoor, 2006b]. This problem becomes worse if defect mapping and reconfiguration are performed at the finest level of granularity i.e. nanodevice level as outlined in [Jacorne et al., 2004, He et al., 2005b, Rad and Tehranipoor, 2006b]. Coarse-grained techniques will not alleviate this issue. There will be a considerable loss in utilisable blocks of the nanofabric if the test resolution is decreased to test larger blocks instead of low level molecular switches and nanowires. Another problem associated with the reconfiguration-based techniques is that defect mapping and configuration must be performed on a per-chip basis which poses a major scalability challenge [He et al., 2005b, Tahoori, 2005a]. Finally, the requirement of performing placement and routing for every chip depending on its unique defect map to avoid the defects also poses another bottleneck during the configuration phase of each nanofabric [Rad and Tehranipoor, 2006b].

2.2.2 Dynamic Reconfiguration

CONAN (COnfigurable Nanostructures for reliAble Nano electronics) is a design methodology that was introduced in [Cotofana et al., 2005] to allow system designers to build

reliable systems out of unreliable nanoscale components. It utilises a hierarchy of abstraction levels to allow coping with both system complexity and reliability and fault tolerance constraints. Each level of abstraction embeds its own support for reliability. The authors proposed regular and decentralized structures to be the main source of reliability of the system hierarchy. These structures are based on simple basic computation cells designed to be robust against physical and transient errors through the redundancy-based fault tolerance techniques applied at the gate level. In order to reconfigure the circuit at runtime to react on transient faults, the computation cells are clustered into regular and reconfigurable nanoclusters. This allows the self diagnosis of the nanocluster to identify the faulty subcell and its exclusion with simple rerouting i.e. dynamic reconfiguration.

Another example of an architecture that uses dynamic reconfiguration is the Cell Matrix [Durbeck and Macias, 2001, Bahar, 2006]. Instead of using static defect mapping and reconfiguration process, the Cell Matrix design uses a dynamic defect/fault discovery and recovery. The idea is to design a distributed and parallel system such that failure detection is done through a set of local tasks running while the system is running. The Cell Matrix is a fine-grained reconfigurable fabric composed of simple homogenous cells and interconnects between adjacent cells. Cells can be configured to operate in either *data* or *configuration* mode. Cells operating in the configuration mode monitor neighbour's activities, detect erroneous behaviour, disable defective cells and relocate the damaged portions of the circuit to other locations.

While both CONAN and Cell Matrix architectures allow the construction of autonomous and self-repairing circuits using simple and locally interconnected homogeneous fabric, the dynamic reconfigurability integrated in these architectures adds another level of complexity to circuit design and also requires a high overhead in interconnection and control.

2.2.3 Hybrid Fault Tolerance Technique

A hybrid fault tolerance technique that combines N-Modular Redundancy (NMR) with both static and dynamic reconfiguration was proposed in [Rao et al., 2006]. According to the authors, although the traditional NMR can exploit the large device densities offered by nanoelectronic devices to tolerate the high fault rates associated with them, this approach is inflexible and rigid when the fault rates are high and time varying. Hence, they proposed a dynamically adaptive NMR approach that enhances the fault tolerance capability of the NMR technique by introducing flexibility and adaptability to the high and time varying fault rates in nanoscale fabrics.

A new nanofabric topology that supports sharing of redundancies in the NMR approach so as to adapt to the time varying fault rates was developed. It is a grid structure con-

sisting of voters and redundant computational units which are embedded in the entire structure. The computational units are implemented using nanoscale PLAs as outlined in Chapter 1, Section 1.1.2 and the voters are built using the more reliable CMOS components. Based on the proposed topological structure, two reconfiguration algorithms were developed to deal with fault tolerance loss caused by manufacturing defects and operation-time online faults. The manufacturing defects are bypassed through a post-fabrication static reconfiguration process so as to minimize the impact of physical defects on the NMR fault tolerance capability. At run time, the flexible NMR approach maintains its online fault tolerance capability through dynamic reconfiguration. In the proposed flexible redundancy structure [Rao et al., 2006], there are shareable backup units among the voters. Initially, the computational units are distributed evenly between the voters. Depending on the NMR technique, the N computational units that are allocated for each voter out of its accessible units are denoted as *in-use* units while the remaining units are denoted as *backup* units. These backup units are utilised as in-use units by the neighbouring voters, representing the flexible additional redundancy for the voter on an as needed basis. To maintain the desirable fault tolerance level, each voter needs to maintain a predefined fault tolerance capability, representing a minimum resource requirement necessary for survival. However, defective or faulty units represent a diminution in resources, necessitating a redistribution of the available resources for the related voters. When the number of fault-free in-use units of a voter falls below the predefined threshold, the voter acquires the backup units from its neighbours to recover from this loss in fault tolerance capability. When the voter tries to utilize a backup unit, the neighbouring voter that is currently using the unit releases it. Then, the newly acquired unit is reconfigured to perform the computation required by the acquiring voter.

In this dynamically adaptive NMR approach, the dynamic reconfiguration algorithm is applied on a per-fault basis. It is invoked whenever a voter compares the results of its in-use units and identifies a faulty unit. However, static reconfiguration algorithm is carried out only once and can be performed off-line. After the manufacturing and testing of the nanofabric, a defect map containing the location of defective units is generated. This map is used by the static reconfiguration unit to minimize the number of failing voters through a global resource reorganisation as outlined in Chapter 2, Section 2.2.1.

2.3 Hybrid Nano/CMOS Architecture

There is a growing consensus that nanoscale devices cannot completely replace CMOS technology in the short term [Sun and Zhang, 2007, Singh et al., 2007]. The prohibitively low reliability of nanodevices dictates that they must be interfaced with CMOS circuits to tolerate the inevitable high defect and fault rates associated with them. This leads to a new paradigm of hybrid nanodevice/CMOS nanoelectronics [Sun and Zhang, 2007,

Jeffery et al., 2004, DeHon et al., 2005, Bahar et al., 2007, Singh et al., 2007, Ziegler and Stan, 2002a, 2003, Strukov and Likharev, 2005b, Bahar, 2006, Zhang et al., 2007, Rao et al., 2006, Rad and Tehranipoor, 2006a] that can perform reliable computing using the highly unreliable nanoscale devices. The idea behind this architecture is to combine the reliable, albeit low performance, CMOS components with the high performance, albeit unreliable, nanodevices.

There have been a number of promising architectures proposed for the hybrid nano/CMOS design paradigm. One of the first architectures using hybrid nano/CMOS design paradigm was proposed in [Ziegler and Stan, 2002a, 2003]. It is a double layered architecture where nano-crossbars act as memory or large logic arrays whereas the underlying CMOS provides routing, signal gain and latching capabilities. In [DeHon et al., 2005, DeHon, 2005], the authors proposed a large-scale memory architecture consisting of smaller submemories called banks. Each bank is composed of a set of crossed nanoscale wires. A CMOS-based infrastructure is used to integrate the nanoscale banks into larger assemblies by performing logic, multiplexing and decoding for interbank connectivity. Similarly, in [DeHon, 2003, 2005], a hybrid nano/CMOS programmable logic architecture based on nanoPLAs was designed such that logic computation and interconnect are provided by nanowires. In this architecture, nanoPLAs play a dual role by being the basic building block of logic circuits and also serve as a switching block between adjacent nanoPLA blocks. The limitations of nanowires length [DeHon, 2005] bounds the size of the nanoPLAs that can be built. Consequently, to scale up to large capacity logic devices, modest size nanoPLA blocks must be interconnected. Hence, nanoPLA blocks are extended to include inputs and outputs to other nanoPLA blocks in order to assemble them into a large logic array. These nanoPLAs will be built on top of a lithographic substrate. The lithographic circuitry and wiring provides a reliable structure to probe the nanowires to map their defects and to configure the logic.

Another architecture that combines CMOS with nanodevices is Molecular-CMOS or CMOL logic cells [Strukov and Likharev, 2005b, 2006, Bahar, 2006, Ma et al., 2005]. A CMOL logic cell combines four CMOS transistors with two levels of parallel nanowires, with molecular-scale nanodevices formed between the nanowires at every crosspoint. In CMOL, the CMOS subsystem serves as a reliable medium for connecting nanoscale devices, providing long interconnect and I/O functions. According to [Bahar, 2006, Ma et al., 2005], the most straightforward application of CMOL would be for memories. It is projected that a CMOL-based memory chip about 2×2 cm^2 in size will be able to store about 1Tb (Terabits) of data. CMOL logic cells have also been proposed for building FPGA-like architectures for the implementation of logic circuits [Strukov and Likharev, 2005b, Ma et al., 2005, Strukov and Likharev, 2006, Bahar, 2006] where the CMOL cell acts as a configurable logic block, similar to that found in an FPGA. Field-programmable nanowire interconnect (FPNI) [Snider and Williams, 2007] is a hybrid nano/CMOS architecture that is based on CMOL. In this architecture, logic is done

only in CMOS, whereas nanowires are used for routing and interconnect. The CMOS layer, which comprises of logic gates, buffers and other component, is divided into an array of square cells with each cell connected to only one input pin and one output pin. A sparse nanoscale crossbar is placed on top of the CMOS gates. The crossbar is rotated so that each nanowire is connected to only one input or output pin of a given CMOS cell. The underlying CMOS logic is interconnected by configuring the appropriate junctions into resistors.

In [Zhang et al., 2006, 2007], a hybrid nano/CMOS reconfigurable architecture was developed called NATURE. It consists of CMOS reconfigurable logic, interconnect fabric and Carbon Nanotube (CNT)-based on-chip configuration memory. This architecture addresses two primary challenges in existing CMOS-based FPGAs: logic density and efficiency of run-time reconfiguration. NATURE solves these problems by using non-volatile nanotube RAMs (NRAMs) as on-chip reconfiguration storage. NRAM is a memory technology developed by Nantero [Nantero] which is expected to be considerably denser than DRAM and have similar speed to SRAM. By using NRAMs, NATURE increases logic density by more than an order of magnitude.

Other architectures that exploit the advantages of complementing nanoscale devices with CMOS were reported in [Sun and Zhang, 2007, Jeffery et al., 2004]. The authors proposed coding-based techniques targeting the fault-prone hybrid nano/CMOS memory systems where the nanoscale devices are responsible for the bulk of information processing and storage, while the CMOS circuit performs testing, decoding, global interconnect and some other critical functions. Similarly, in [Singh et al., 2007], it is believed that the early practical uses of Carbon Nanotube-based electronics will be built on top of hybrid CMOS/CNT processes. A computational architecture which is based on a hybrid CMOS/CNT fabric was outlined in this paper. The high density of CNT devices is exploited to implement logic circuits as LUTs. The content of the LUTs is protected by using a coding technique and the corresponding CMOS decoder is used to correct any faulty bits. Moreover, in building a reliable circuit from unreliable nanodevices using the NMR-based fault tolerance technique proposed in [Rao et al., 2006] and outlined in Chapter 2, Section 2.2.3, the reliability of the design is limited by that of the final gate driving the output which is the arbitration unit. The authors suggested mapping the proposed topology onto a hybrid nano/CMOS system in which the computation units are implemented using unreliable nanoelectronic crossbar PLAs and the arbitration units are implemented using the significantly more reliable CMOS.

While there are tremendous efforts devoted to nanoscale devices and fabrication, there is a growing interest in hybrid nano/CMOS systems, as illustrated in this section. However, very little has been done to exploit the advantages of combining both technologies in order to build a highly reliable system, given the high defect rates in nanodevices. New architectures and defect tolerant techniques should be developed to overcome the high defect rates barrier and to exploit the combination of CMOS and nanotechnology to its

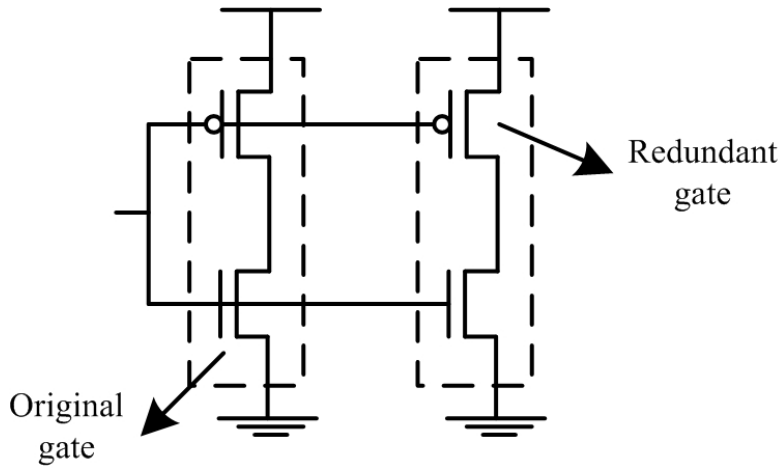


FIGURE 2.4: Gate-level redundancy to enhance yield in nanometre CMOS circuits [Sirisantana et al., 2004]

limits. The techniques to be proposed in Chapters 4 and 6 will further illustrate how CMOS components help achieve highly reliable nano-systems.

2.4 Defect Tolerance Techniques for Nanometre CMOS

Designing highly efficient defect tolerance techniques for nanometre CMOS is also a subject of extensive research. Authors in [Sirisantana et al., 2004] proposed adding transistor-level redundancy by having parallel transistors as permanent back-up as shown in Fig. 2.4. While this approach helps enhance yield by up to 3.2 times, the percentage of defective transistors that it can tolerate is only 0.01%. Moreover, the redundant gates cause higher current drive in the absence of defects and hence a significant increase in power consumption (up to 45%). Another drawback of this technique is that it can only handle stuck-open defect types.

To address a wider range of physical defects, the authors in [Kothe et al., 2006] suggested a modified architecture to that in [Sirisantana et al., 2004] where power switches are included between transistors and V_{dd}/GND rails, as shown in Fig. 2.5. The aim of adding switches is to separate faulty transistors from the supply lines during fault diagnosis. The main advantage of this technique is that it covers more defect types including stuck-open, stuck-short and bridging faults. However, the repair capabilities of this approach is accompanied by a higher design complexity and a triplication of the number of transistors. Given a uniform distribution of defects, logic gates become three times more vulnerable to faults. It should also be noted that adding power switches makes logic gates slower and require additional power overheads to activate them. Furthermore, this technique requires an efficient diagnosis mechanism to bypass the defective gates.

In [Ashouei et al., 2008], a reconfiguration-based defect tolerance architecture for CMOS logic gates was presented. It takes advantage of the functional redundancy in static

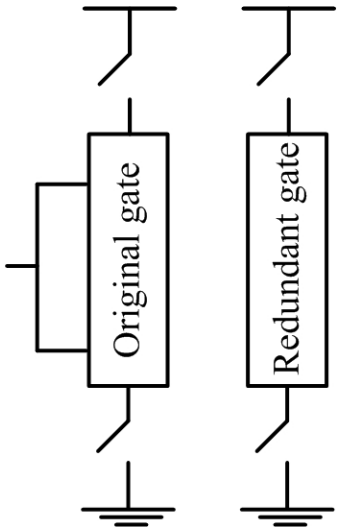


FIGURE 2.5: Gate level redundancy with switches to enhance yield in nanometre CMOS circuits [Kothe et al., 2006]

CMOS where functions are implemented twice by N-transistor and P-transistor networks. Fig. 2.6 illustrates the architecture of the proposed defect-tolerant CMOS gate which operates as follows: during the normal operation, i.e. when there is no fault, switch1 and switch2 are connected to V_{dd} and GND respectively. In this case, switch3 (switch4) connects the gate of the pull-up (pull-down) transistor to the V_{dd}(GND) to turn it off. During the defect tolerance operation, if a faulty transistor is in the pull-up P-network, the logic gate is reconfigured to a pseudo NMOS gate by replacing the faulty P-network with a single P-transistor which acts as a resistive pull-up. Switch1 is turned off to disconnect the faulty P-network from the power supply and switch3 connects the gate of pull-up transistor to the GND. The setting of switch2 and switch4 remains the same as the normal operation. Similarly, a fault in the N-network is recovered from by replacing it by a single N-transistor acting as a resistive pull-down. Switch2 is turned off and switch4 turns on the pull-down transistor. This converts the logic gate to a pseudo NMOS gate. Although this technique can effectively tolerate hundreds of manufacturing defects in a nanometre CMOS fabric, it suffers from major drawbacks. First, the reliability of this technique is limited by the accuracy of the diagnosis technique to perfectly determine the location of defective components. Second, there is a significant delay and leakage overheads due to the reconfiguration of logic gates. Finally, in designs with several million gates, this technique can only target a defect ratio of up to 0.01% which reflects the limitation of its efficiency.

2.5 Motivation and Objectives

From the detailed literature review, it is clear that a significant effort has been made by researchers around the world to develop efficient yield and reliability improvement

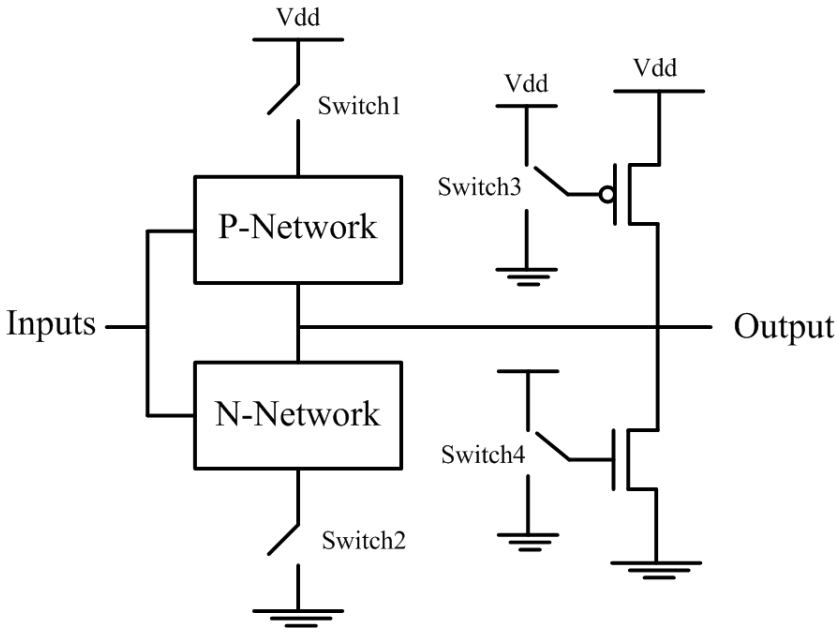


FIGURE 2.6: Architecture of the defect-tolerant CMOS gate [Ashouei et al., 2008]

techniques for nanoscale devices and nanometre CMOS. Several techniques have been proposed to tackle the projected high permanent and transient fault rates. These techniques can be classified into two main categories: reconfiguration and redundancy-based techniques. Each one has advantages and disadvantages, however at this stage, it is not clear which approach is better. Therefore, in light of the high defect and fault rates of nanometre CMOS and nanoscale devices, it is believed that more research is necessary to devise highly efficient techniques that are capable of tolerating higher defect and fault rates and also overcoming the drawbacks of the previously proposed techniques.

Defective and faulty fabrics necessitates designing architectures with a collection of interchangeable resources that are used in post-fabrication programming to configure components so that they use only functional resources, thus avoiding the defective resources. Reconfiguration-based techniques require using defect-mapping to locate the faulty resources. However, as feature sizes shrink to reach the nanometre scales, defect rates inevitably increase and using reconfiguration-based techniques to tolerate them will impose major challenges in the manufacturing process, as explained in Chapter 2, Section 2.2.

Granularity is another important aspect of defect and fault tolerance. Some defect and fault tolerance techniques improve yield and reliability by allocating redundancy at the lowest level of granularity i.e. fine-grained redundancy, whereas other techniques allocate redundancy at higher levels of abstraction i.e. coarse-grained redundancy. In [Yu and Lemieux, 2005] and [Breuer et al., 2004], the authors proved that by adding redundancy at finer levels of granularity, yield and reliability can be significantly improved at reduced area overheads as compared to coarse-grained redundancy.

The objectives of the research reported in this thesis are as follows:

- Design techniques that either do not require defect diagnosis and mapping before repair (Chapters 3 and 6) or limit test and diagnosis to higher levels of granularity to cut on post-fabrication costs by reducing testing overhead (Chapters 4 and 5).
- Defect and fault tolerance should be addressed at the lowest level of abstraction (i.e. transistor level in Chapter 3 and nanodevice or nanowire level in Chapters 4, 5 and 6) to provide robust tolerance.
- Devise efficient defect and fault tolerance techniques capable of dealing with higher defect and fault rates as compared to recently reported techniques for promising architectures including nanometre CMOS, nanoscale PLA and hybrid Nano/CMOS architectures, outlined in Chapter 1, Section 1.3.
- Defect and fault tolerance should be achieved at a reduced area and energy overheads.
- Validate the developed techniques through extensive simulations using the appropriate defect and fault models.
- Validate the developed techniques by deriving the mathematical equations that predict their performance.
- Measure the overheads incurred by the proposed techniques including area and energy overheads.

Chapter 3

Nanometre CMOS Defect Tolerance Technique

There is a wide consensus [Ashouei et al., 2008, Sirisantana et al., 2004, Kothe et al., 2006] that production yield will be a significant issue in nanometre CMOS because of manufacturing defects. In this chapter, a new novel defect tolerant technique that adds redundancy at the transistor level and provides built-in immunity to permanent defects including stuck-open, stuck-short and bridging defects (Chapter 1, Section 1.2.1) is investigated. In order to increase the manufacturing yield, the proposed technique is based on replacing each transistor by an N^2-transistor structure ($N \geq 2$) that guarantees defect tolerance of all $(N-1)$ defects. After modelling the structure, developing the appropriate failure rate simulations and analysing the effectiveness of the proposed structure with respect to recently reported techniques, the N^2-transistor structure proved to be a new and a potential defect tolerance technique for nanometre CMOS.

The chapter is organised as follows: Section 3.1 gives an overview of the N^2-transistor structure. In Sections 3.2 and 3.3, two particular cases of the N^2-transistor structure called Quadded Transistor ($N = 2$) and Nonuple Transistor ($N = 3$) structures are presented. The previously proposed fault tolerance techniques are outlined in Section 3.4. Experimental results are reported in Section 3.5 and finally Section 3.6 concludes the chapter.

The author's contribution in this chapter includes developing the mathematical equations that predict the performance of both Quadded Transistor ($N = 2$) and Nonuple Transistor ($N = 3$) structures in the presence of stuck-at faults and also the analysis of the defect tolerance capabilities of both structures to bridging defects.

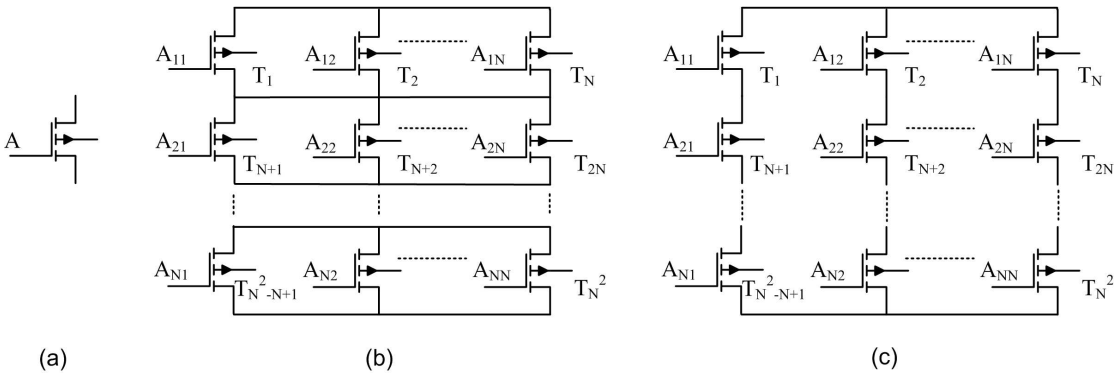


FIGURE 3.1: Defect-tolerant N^2-transistor structure

3.1 Overview of the N^2-Transistor Structure

An N^2-transistor structure is composed of N blocks connected in series with each block composed of N parallel transistors where $N \geq 2$ as shown in Fig. 3.1. In Fig. 3.1(b), the N^2-transistor structure implements the logic function $(A_{11} + A_{12} + ... + A_{1N}).((A_{21} + A_{22} + ... + A_{2N})...((A_{N1} + A_{N2} + ... + A_{NN})$, whereas in Fig. 3.1(c), the following logic function is implemented: $(A_{11}.A_{21}...A_{N1}) + ((A_{12}.A_{22}...A_{N2}) + ... + ((A_{1N}.A_{2N}...A_{NN})$. The N^2-transistor structure guarantees the 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 this structure. In Sections 3.2 and 3.3, two new defect tolerance techniques called Quadded Transistor and Nonuple Transistor structures which are based on the N^2-transistor structure for $N = 2$ and $N = 3$ respectively are proposed.

3.2 Quadded Transistor Structure

The Defect Tolerant Quadded Transistor (DTQT) structure can tolerate single transistor defects by replacing every transistor in the circuit by four transistors implementing one of the two logic functions (A+A).(A+A) or (A.A)+(A.A). Fig. 3.2 illustrates the two versions of this technique implementing both logic functions. An example of this structure is shown in Fig. 3.3 where a standard 2-input NAND gate is implemented using the proposed DTQT technique.

Analysing the DTQT structure, it can be observed that both structures shown in Figs. 3.2(b) and (c) can tolerate any single transistor defects. A transistor is considered defective if its expected behaviour changes regardless of the type of the defect causing it. Any single stuck-open, stuck-short or AND/OR-bridge defect will not change the logic behaviour of the Boolean functions. It should also be observed that for NMOS transistors, OR-bridge and stuck-short defects produce the same behaviour while AND-bridge and stuck-open defects have the same behaviour. Similarly, for PMOS transistors,

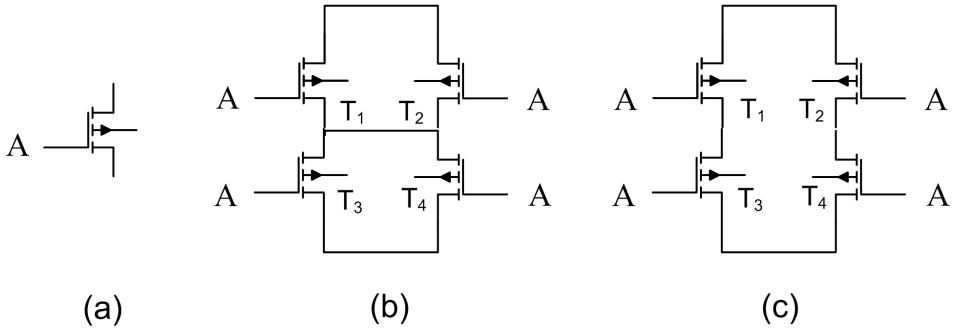


FIGURE 3.2: DTQT transistor level diagram (a) Transistor in original gate implementation (b) DTQT structure implementing logic function $(A + A).(A + A)$ (c) DTQT structure implementing logic function $(A.A) + (A.A)$

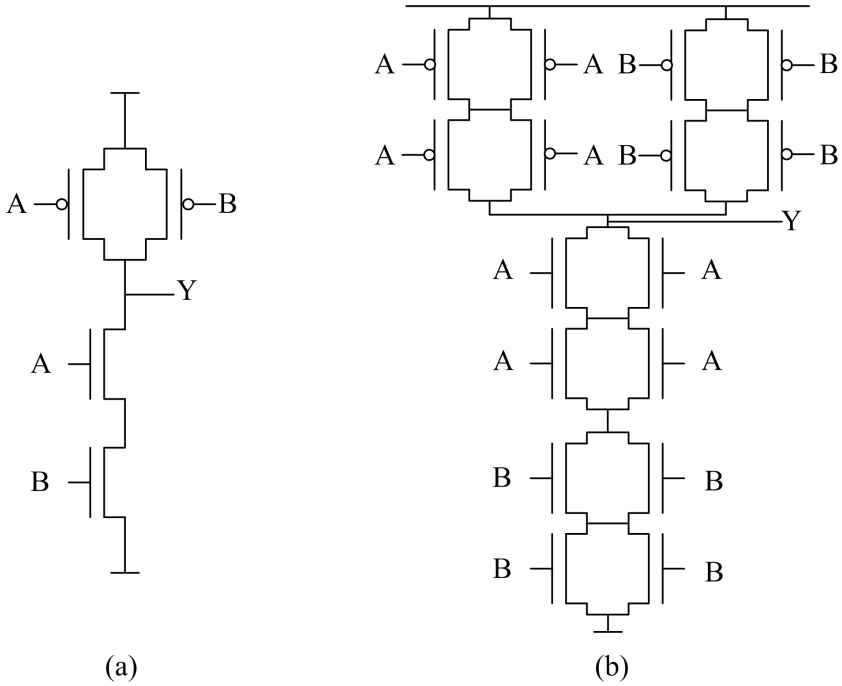


FIGURE 3.3: NAND gate (a) Normal (b) DTQT

OR-bridge and stuck-open defects produce the same behaviour while AND-bridge and stuck-short defects have the same behaviour.

Double stuck-open (and their corresponding bridge) defects are tolerated as long as they do not occur in any two parallel transistors T_1&T_2 or T_3&T_4 in Fig. 3.2(b) and T_1&T_2, T_1&T_4, T_3&T_2 or T_3&T_4 in Fig. 3.2(c). Double stuck-short (and their corresponding bridge) defects are also tolerated as long as they do not occur in any two series transistors T_1&T_3, T_1&T_4, T_2&T_3 or T_2&T_4 in Fig. 3.2(b) and T_1&T_3 or T_2&T_4 in Fig. 3.2(c). Furthermore, any triple defects that do not include two series stuck-short transistors or two parallel stuck-open transistors or their corresponding bridging defects are also tolerated. Therefore, it can be concluded that the manufacturing yield can be significantly improved by employing the quadded-transistor structure.

Another interesting remark about the DTQT structure is that its effective resistance is the same as the resistance of the single transistor R. In Fig. 3.2(b), the total resistance of the two resistors in parallel is $R/2$ and the effective resistance of the structure is: $R/2 + R/2 = R$. In Fig. 3.2(c), the total resistance of the two resistors in series is: $2 \times R$ and the effective resistance of the structure is: $1/(1/2R + 1/2R) = R$. However, in the presence of a single defect, the worst case effective resistance of the first quadded structue (Fig.3.2(b)) is $1.5R$ while that of the second structure (Fig. 3.2(c)) is $2R$. 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 $2R$. Therefore, the quadded transistor structure shown in Fig. 3.2(b) is adopted in this chapter for the analysis of the proposed technique.

3.2.1 Theoretical Analysis

In this section, the failure rate of the proposed DTQT structure is theoretically examined. The mathematical equations to be derived predict the probability of circuit failure implemented using the quadded transistor structure in the presence of a certain defect rate. Given a transistor defect probability P, the probability of circuit failure P_f based on the quadded transistor structure can be determined as follows:

- The total probability of DTQT failure P_q is the sum of the probability of failure when having one, two, three or four defects i.e.

$$P_q = P(1) + P(2) + P(3) + P(4) \tag{3.1}$$

- $P(1) = 0$ because all single transistor defects are tolerated by the DTQT technique.
- 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 pairs of defects produces an error as illustrated in Table 3.1. In Fig. 3.2(b), if transistors T_1 and T_4 are defective, then there are four possible combinations of defects which are listed in Table 3.1. Only one of these combinations leads to an error which is T_1 and T_4 being both stuck-short.

T_1	T_4
stuck-open	stuck-open
stuck-open	stuck-short
stuck-short	stuck-open
stuck-short	stuck-short

TABLE 3.1: Only one of the four possible pairs of defective trasistors in DTQT structure produces an error.

Thus the probability of failure in this case is:

$$\begin{aligned} P(2) &= \frac{1}{4} \times \binom{4}{2} P^2(1-P)^2 \\ &= \frac{3}{2} P^2(1-P)^2 \end{aligned} \tag{3.2}$$

- If three transistors are assumed to be 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:

$$\begin{aligned} P(3) &= \frac{5}{8} \times \binom{4}{3} P^3(1-P) \\ &= \frac{5}{2} P^3(1-P) \end{aligned} \tag{3.3}$$

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

$$\begin{aligned} P(4) &= 1 \times \binom{4}{4} P^4(1-P)^0 \\ &= P^4 \end{aligned} \tag{3.4}$$

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

$$\begin{aligned} P_q &= \frac{3}{2} P^2(1-P)^2 + \frac{5}{2} P^3(1-P) + P^4 \\ &= \frac{3}{2} P^4 - 3P^3 + \frac{3}{2} P^2 + \frac{5}{2} P^3 - \frac{5}{2} P^4 + P^4 \\ &= \frac{3}{2} P^2 - \frac{1}{2} P^3 \end{aligned} \tag{3.5}$$

The above equations were derived with respect to stuck-open and stuck-short defects as bridging faults have equivalent behaviour to these faults as outlined earlier.

Given the transistor defect probability P and a circuit, with T transistors, to be implemented using T quadded transistor structures, the overall probability of circuit failure follows a binomial distribution as given by the following equation:

$$P_f = \sum_{i=1}^{T} \binom{T}{i} P_q^i (1-P_q)^{T-i} \tag{3.6}$$

Hence, the yield is:

$$\begin{aligned} Y &= 1 - P_f \\ &= 1 - \sum_{i=1}^{T} \binom{T}{i} P_q^i (1 - P_q)^{T-i} \end{aligned} \tag{3.7}$$

Equation 3.6 is based on the binomial distribution theorem. The probability of failure P_f of a circuit with T transistors can also be computed using the inclusion-exclusion principle [Comtet, 1974] as follows:

$$P_f = \sum_{i=1}^{T} (-1)^{i-1} \binom{T}{i} P_q^i \tag{3.8}$$

Both equations 3.6 and 3.8 produce equivalent results.

Both equations 3.6 and 3.8 compute the factorials of large numbers such as the number of transistors T. Equation 3.9 allows the computation of circuit failure probability without calculating these factorials.

$$P_f = 1 - (1 - P_q)^T \tag{3.9}$$

The graphs in Fig. 3.4 show the failure rates for a number of ISCAS benchmark circuits obtained both theoretically based on equation 3.6 and experimentally based on the simulation procedure outlined in Section 3.5.1. As can be seen, there is an identical match, clearly validating the derived theoretical equations. While equations 3.6 and 3.7 represents the exact failure probability and yield of circuits in the presence of stuck-open and stuck-short defects, it represents the upper bound for bridging faults because not all bridging defects in a faulty quadded transistor structure result in a faulty gate behaviour. For instance, AND-bridging defects between gates of transistors within the same NAND gate do not change the gate behaviour regardless of their multiplicity. Similarly, OR-bridging defects between gates of transistors within the same NOR gate do not change the gate behaviour regardless of their multiplicity.

Fig. 3.5 compares the yield ratio of several NAND gates of various inputs including 2, 4 and 8 input gates implemented using the quadded transistor structure and the conventional complementary (pull-up, pull-down) CMOS. Yield was computed after injecting various percentages of stuck-open and stuck-short defects with equal probabilities. As can be seen, the level of defect tolerance exhibited by the DTQT structure is significantly higher than the defect tolerance obtained in the absence of the proposed technique. As an example, for an 8-input NAND gate and a defect probability of 10%, the yield obtained by the quadded transistor structure is 79% whereas the conventional CMOS

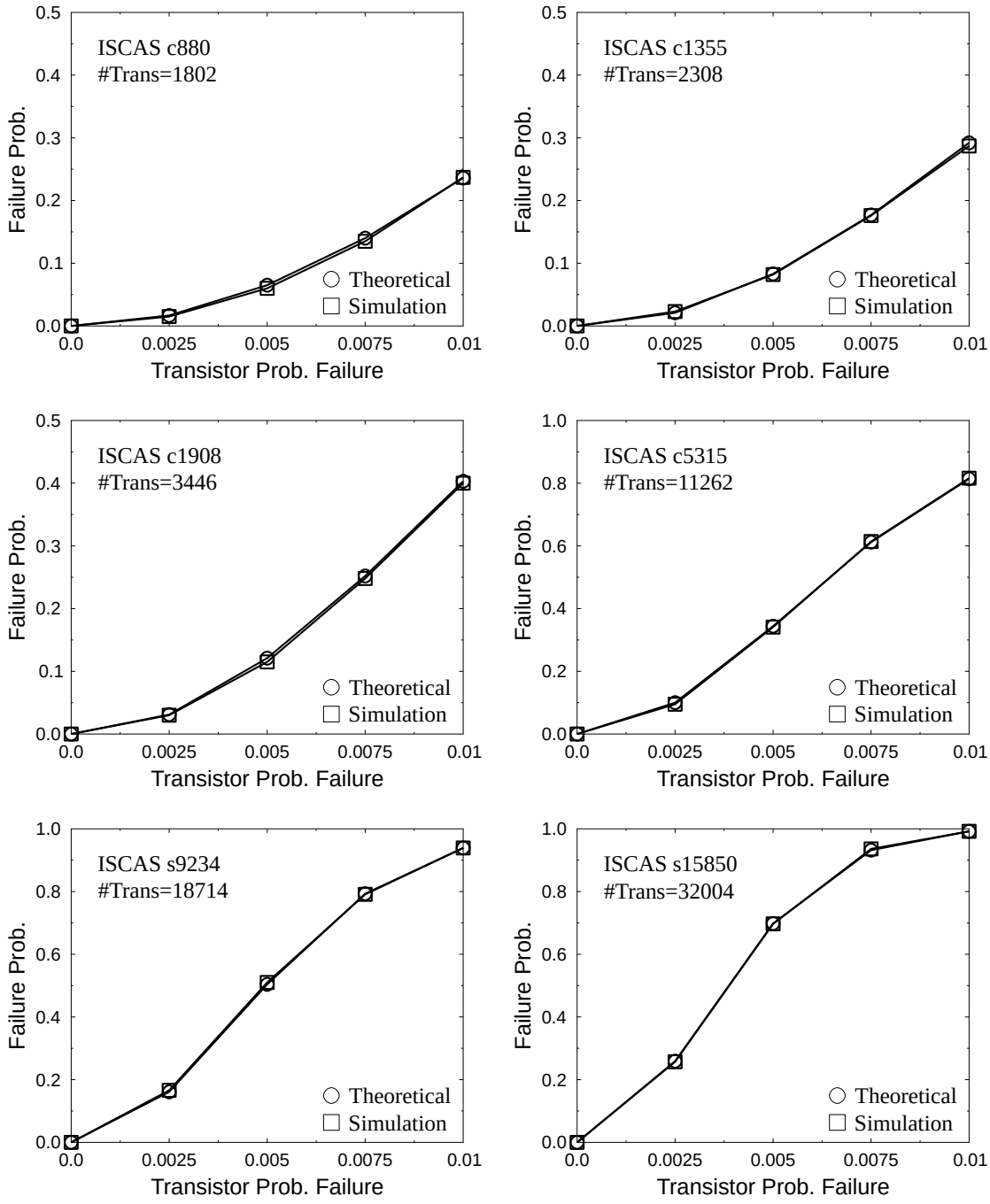


FIGURE 3.4: Quadded transistor structure - circuit failure rate obtained both theoretically and experimentally based on simulations of some of ISCAS circuits

implementation achieves only 19%. It should also be observed that as the size of the gate increases, the yield ratio of the DTQT technique decreases. This is due to the increased number of defects that are present in bigger logic gates.

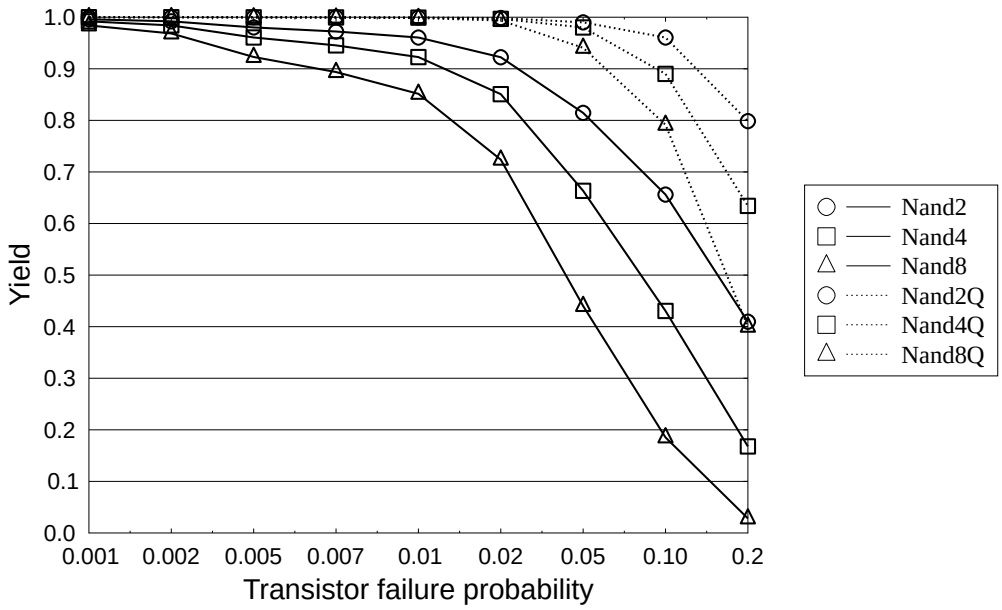


FIGURE 3.5: Yield comparison between quadded-transistor structure (Q) and complementary CMOS

3.3 Nonuple Transistor Structure

To further improve the manufacturing yield, another defect-tolerant structure called Defect Tolerant Nonuple Transistor (DTNT) structure is proposed which is a particular class of N^2-Transistor structure with $N = 3$. In this structure, each transistor is replaced by nine transistors implementing either the logic function $(A + A + A)(A + A + A)(A + A + A)$ or the logic function $(AAA) + (AAA) + (AAA)$, as illustrated in Fig. 3.6. In both structures shown in Fig. 3.6(b) and (c), any single transistor defect (stuck-open stuck-short or AND/OR-bridge) will not affect the logic behaviour of the structure, and hence it is tolerated. Double stuck-open (and their corresponding bridge) and double stuck-short (and their corresponding bridge) defects are also tolerated. Furthermore, 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. Therefore, it can be concluded that using either of the nonuple transistor structures in circuit implementation will lead to a significant improvement in yield.

The nonuple transistor structure shown in Fig. 3.6(b) is adopted in this chapter for the same reason of having less resistance than the structure shown in Fig. 3.6(c) in case of tolerable defects in the transistors as explained in Section 3.2 for the quadded transistor structure.

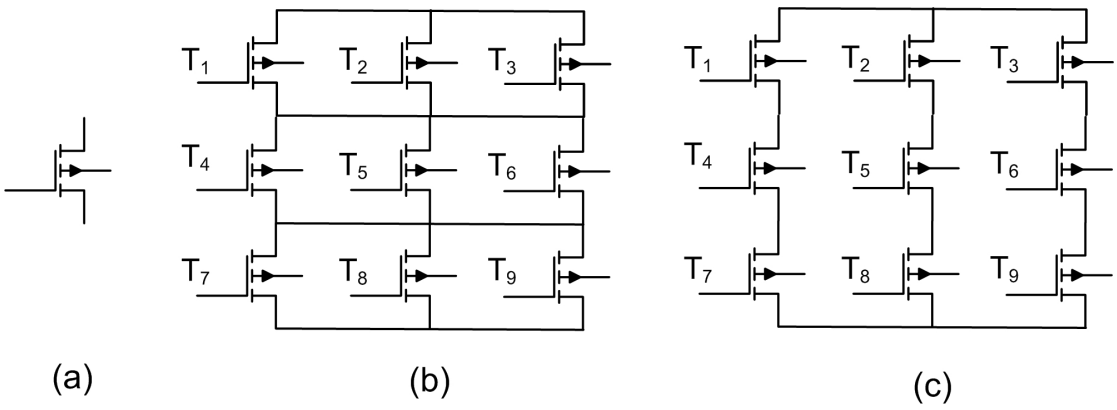


FIGURE 3.6: Nonuple structure transistor level diagram

3.3.1 Theoretical Analysis

In this section, the mathematical equations that relate the yield exhibited by the nonuple transistor structure and the probability of transistor being defective due to stuck-open or stuck-short faults are derived.

Theoretically, given a transistor defect probability P, the probability of circuit failure P_f based on nonuple transistor structure can be computed as follows:

- The total probability of failure of this structure P_n is the sum of the probability of failure when having upto nine defects i.e.

$$P_n = P(1) + P(2) + P(3) + P(4) + P(5) + P(6) + P(7) + P(8) + P(9) \quad (3.10)$$

- $P(1) = 0$ because all single transistor defects are tolerated.
- $P(2) = 0$ because all double transistor defects are tolerated.
- If there are three defective transistors in a nonuple transistor structure, then there are eight (2^3) possible combinations of stuck-open and stuck short defects. In all cases, only one of these 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:

$$\begin{aligned} P(3) &= (3 \times \frac{1}{8} + 27 \times \frac{1}{8})P^3(1-P)^6 \\ &= \frac{30}{8}P^3(1-P)^6 \end{aligned} \quad (3.11)$$

- If four transistors are assumed to be defective, then there are sixteen (2^4) possible combinations of stuck-open and stuck short defects. Among those, only two combinations produce an error for eighteen 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:

$$\begin{aligned} P(4) &= (18 \times \frac{2}{16} + 81 \times \frac{3}{16}) P^4 (1-P)^5 \\ &= \frac{279}{16} P^4 (1-P)^5 \end{aligned} \tag{3.12}$$

- If there are five defective transistors, then there are thirty-two (2^5) 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 11 combinations produce an error for 27 series transistor structures which are overlapping with parallel transistor structures. Also, 9 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:

$$\begin{aligned} P(5) &= (18 \times \frac{4}{32} + 27 \times \frac{11}{32} + 81 \times \frac{9}{32}) P^5 (1-P)^4 \\ &= \frac{1098}{32} P^5 (1-P)^4 \end{aligned} \tag{3.13}$$

- If six transistors are assumed to be defective, then there are sixty-four (2^6) possible combinations of stuck-open and stuck short defects. Among those, only fifteen combinations produce an error for three 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 twenty-seven series transistor structures which are non-overlapping with parallel transistor structures. Thus, the probability of failure in this case is:

$$\begin{aligned} P(6) &= (3 \times \frac{15}{64} + 54 \times \frac{29}{64} + 27 \times \frac{27}{64}) P^6 (1-P)^3 \\ &= \frac{2340}{64} P^6 (1-P)^3 \end{aligned} \tag{3.14}$$

- If seven transistors are defective, then there are one hundred and twenty eight (2^7) 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:

$$P(7) = (1 \times \frac{74}{128} + 35 \times \frac{79}{128})P^7(1-P)^2$$
$$= \frac{2839}{128}P^7(1-P)^2 \tag{3.15}$$

- If eight transistors are assumed to be defective, then there are two hundred and fifty six (2^8) 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:

$$P(8) = (1 \times \frac{158}{256} + 8 \times \frac{207}{256})P^8(1-P)$$
$$= \frac{1814}{256}P^8(1-P) \tag{3.16}$$

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

$$P(9) = P^9 \tag{3.17}$$

- Therefore, the overall probability of nonuple-transistor structure failure is:

$$P_n = \frac{30}{8}P^3(1-P)^6 + \frac{279}{16}P^4(1-P)^5 + \frac{1098}{32}P^5(1-P)^4 + \frac{2340}{64}P^6(1-P)^3$$
$$+ \frac{2839}{128}P^7(1-P)^2 + \frac{1814}{256}P^8(1-P) + P^9$$
$$= \frac{30}{8}P^3 - \frac{81}{16}P^4 + \frac{27}{8}P^5 - \frac{21}{16}P^6 + \frac{31}{128}P^7 - \frac{19}{128}P^8 + \frac{5}{32}P^9 \tag{3.18}$$

Similarly, given the transistor defect probability P and a circuit with T transistors, to be implemented using T nonuple transistor structures, the overall probability of circuit failure follows a binomial distribution as given by equation 3.6 by replacing P_q given by equation 3.5 with P_n given by equation 3.18, as follows:

$$P_f = \sum_{i=1}^{T} \binom{T}{i} P_n^i (1-P_n)^{T-i} \tag{3.19}$$

Circuit failure probability P_f can also be computed using the following equation:

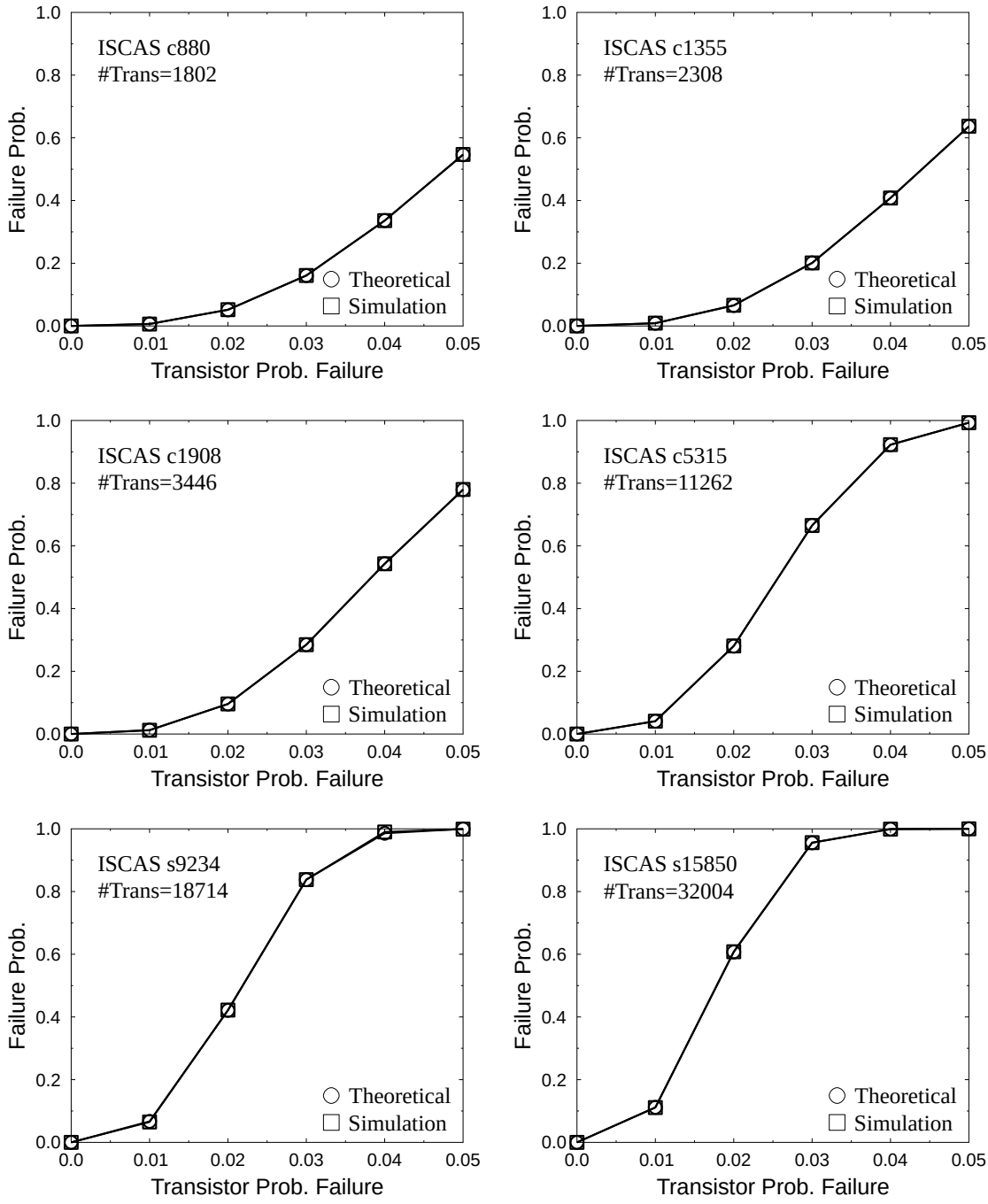


FIGURE 3.7: Nonuple transistor structure - circuit failure rate obtained both theoretically and experimentally based on simulations for some of ISCAS circuits

$$P_f = 1 - (1 - P_n)^T \tag{3.20}$$

In Fig. 3.7, the theoretical and the simulation-based failure rates of a number of ISCAS benchmark circuits implemented using the nonuple transistor structure are shown. As can be observed, there is an identical match between the predicted and actual results which validates the derived theoretical equations.

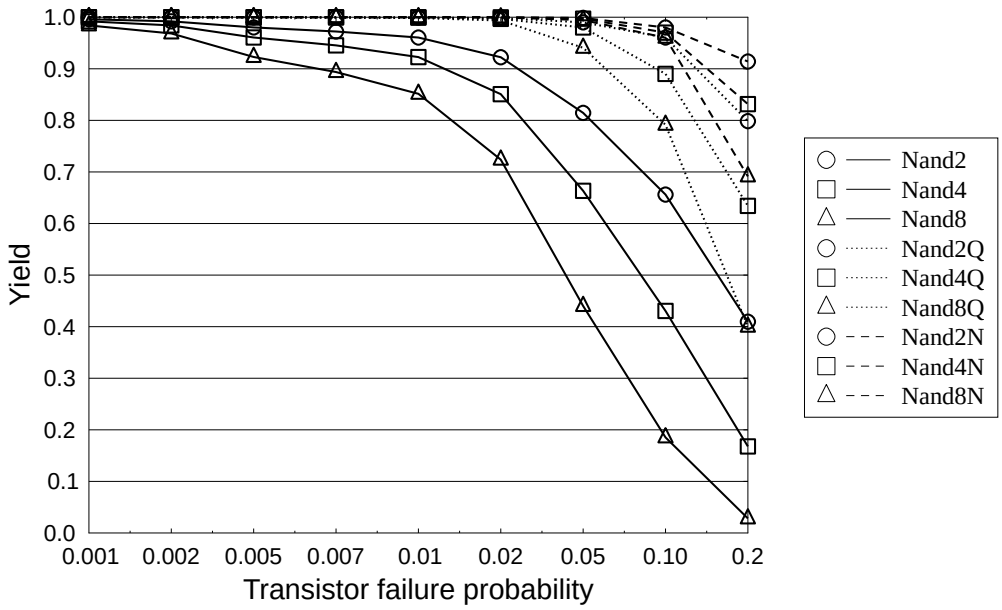


FIGURE 3.8: Yield comparison between quadded-transistor structure (Q), nonuple-transistor structure (N) and complementary CMOS

From equations 3.5 and 3.18, it can be concluded that the failure probability of the proposed N^2-transistor structure is of the order $O(P^N)$. It can also be concluded from the two equations that as the number of transistors N increases, the failure probability of the structure decreases but at the cost of more redundancy and hence more area overhead. The N^2-transistor structure, for $N > 2$, can be applied selectively for critical gates due to its increased area overhead.

In Fig. 3.8, the improvements in defect tolerance against both stuck-open and stuck short defects that can be achieved by both quadded transistor and nonuple transistor structures are compared. As can be observed, the nonuple transistor technique exhibits higher yield values as compared to the quadded transistor structure in the presence of high defect rates. For instance, for an 8-input NAND gate and a defect probability of 10%, the yield obtained by the nonuple transistor structure is more than 95%, whereas the yield achieved by quadded transistor structure is 79%.

3.4 Other Defect Tolerance Techniques

In a seminal 1956 paper [von Neumann, 1956], when the absence of reliable electronic components motivated the need for fault-tolerant designs, von Neumann proposed a modular-based redundant design approach called the multiplexing technique. This latter is based on massive duplication of unreliable devices and randomised imperfect inter-

connects. The author proved that by providing a sufficiently low failure rate of a single component, the multiplexing structure can be reliable with high probability [Thaker et al., 2005, Han et al., 2005], i.e. design reliable circuits from unreliable components.

There has been a renewed interest in using hardware redundancy to mask faulty behaviour in nanoelectronic components. Examples of these techniques are Triple Modular Redundancy (TMR) [Han et al., 2005, Thaker et al., 2005, Nikolic et al., 2002], and Quadded Logic [Han et al., 2005]. In [Han et al., 2005], both techniques were investigated with the aim to improve the reliability of nanoelectronic design. It has been demonstrated that such techniques are capable of making nanoelectronic circuits more robust to defects. Both TMR and Quadded Logic techniques are used as reference points to compare the yield values exhibited by these two techniques with those of the proposed N^2-transistor structures.

3.4.1 Triple Modular Redundancy (TMR)

Triple Modular Redundancy technique [Nikolic et al., 2002, Thaker et al., 2005, Han et al., 2005, Han and Jonker, 2004] is based on triplicating each module of a given size followed by an arbitration unit as illustrated in Fig. 3.9(b). Similarly, all the input signals are also duplicated three times and randomly fed to the three modules and the final output is driven by a majority voter (denoted V in Fig. 3.9(b)) that decides the correct result out of the three intermediate outputs. The voter unit is used to generate a single output that is a majority vote of the three system outputs. Therefore, if a single unit fails, the other two will outvote it, and the voter output remains correct [Nikolic et al., 2002]. However, the reliability of TMR is limited by that of the final arbitration unit, making the approach difficult to implement in the context of highly integrated nanosystems [He et al., 2005b]. The reliability of TMR technique is equal to the multiple of the probability of having at least two out of the three instances of TMR correctly functioning and the reliability of the majority voter as given by equation 3.21.

$$R_{TMR} = R_{voter} \sum_{k=2}^{3} \binom{3}{k} R_{mod}^{k} (1 - R_{mod})^{3-k} \tag{3.21}$$

where R_{voter} is the reliability of the voter and R_{mod} is the probability that each module of TMR is correctly functioning.

The TMR process can be repeated by combining three of the TMR units with another majority voter to form a second order TMR unit with even higher reliability [Nikolic et al., 2002, Han et al., 2005, Thaker et al., 2005]. The obtained circuit thus has nine copies of the original module and two layers of majority gates. This process can be further repeated if necessary, resulting in a technique called Cascaded Triple Modular Redundancy (CTMR) or Recursive Triple Modular Redundancy (RTMR). In [Thaker

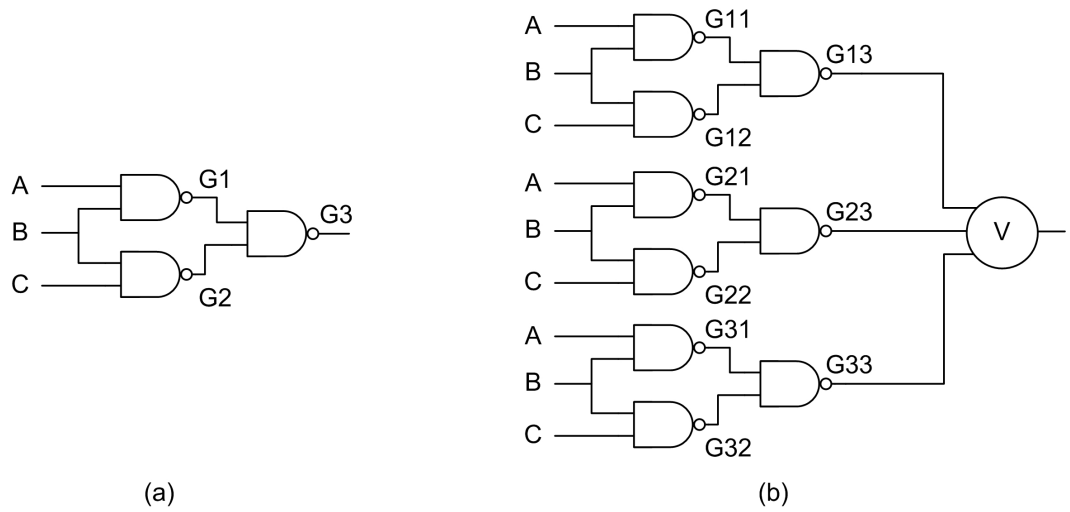


FIGURE 3.9: TMR architecture (a) Original circuit (b) TMR circuit

et al., 2005], it was proved that recursive voting technique leads to a double exponential decrease in a circuit's failure probability. However, a single error in the last majority voter will cause an error in the final output and hence hampering the techniques effectiveness. Another disadvantage of the CTMR is that it introduces an exponential growth in redundancy as the number of cascaded layers increases. Furthermore, it was shown in [Spagocci and Fountain, 1999] that using CTMR technique in nanoscale chips with large numbers of nanoscale devices will require an extremely low device error rate.

In order to improve its efficiency, the TMR technique was implemented at the logic gate level as shown in Fig. 3.10(b) rather than the unit level as shown in Fig. 3.9(b). Each gate in the circuit is triplicated and its outputs are fed to a majority voter. This implementaiton incurs more area overhead due to the increased number of voters as compared to the implementation shown in Fig. 3.9(b).

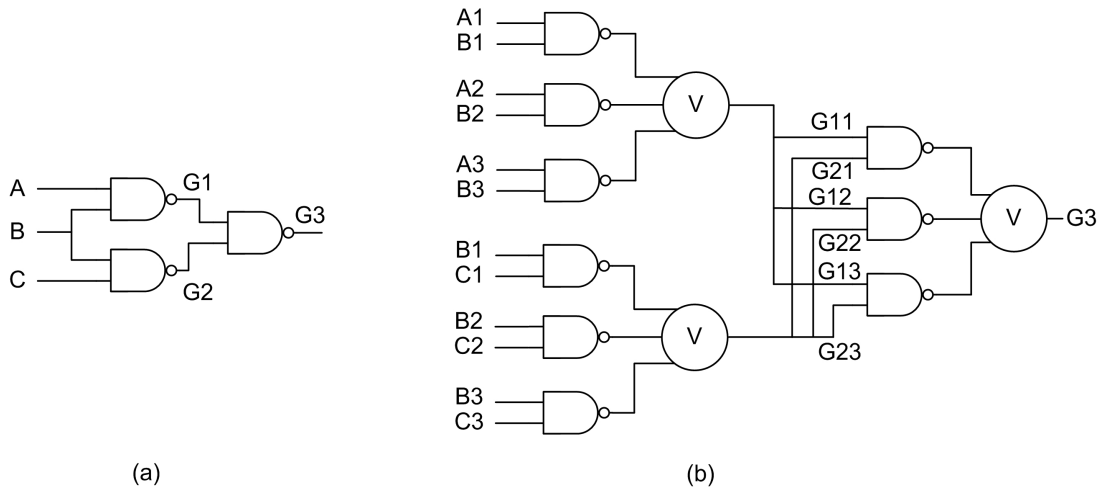


FIGURE 3.10: TMR architecture implemented at the logic gate level

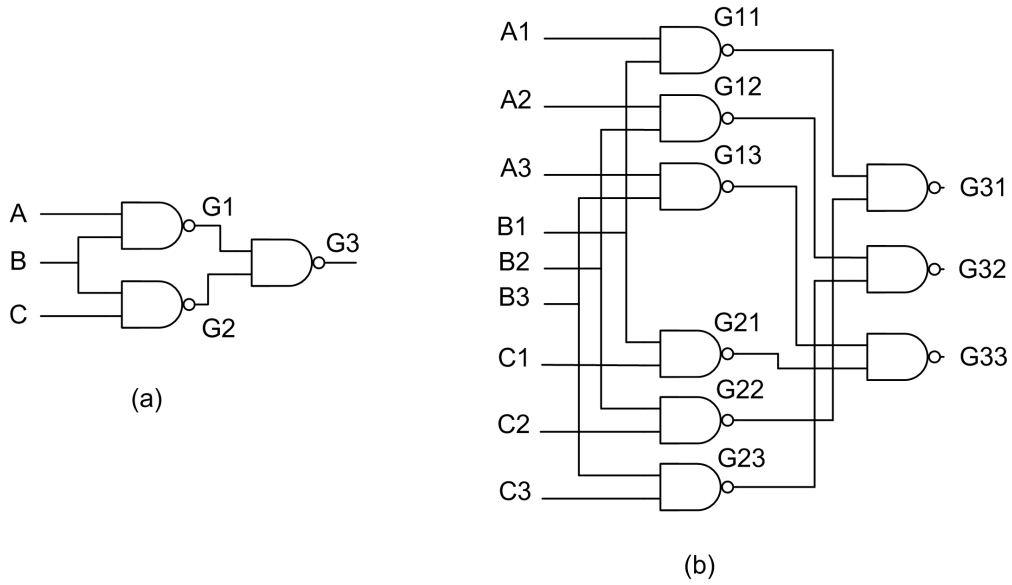


FIGURE 3.11: TIR architecture (a) Original circuit (b) TIR circuit

3.4.2 Triplicated Interwoven Redundancy (TIR)

The idea of triplicated interwoven redundancy (TIR) [Han et al., 2005, Han and Jonker, 2004, Han, 2004] originates from von Neumann's multiplexing technique [von Neumann, 1956] and the general concept of interwoven redundant logic [Pierce, 1965]. Fig. 3.11 shows the schematic of a TIR implementation of a logic circuit. In a TIR circuit, all the gates in the non-redundant circuit are triplicated as well as the interconnections. Hence, a TIR circuit has three times as many gates and interconnections as the corresponding non-redundant circuit.

The interconnections in a TIR circuit are in principle arranged in a random way. However, in practical implementation, the random interconnections can be substituted by arbitrarily selected static ones that have specific routes [Han et al., 2005, Han and Jonker, 2004]. In a TIR circuit made up of 2-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)}. A pair (i, j) means that the output of gate i in a triplication is paired with the output of gate j in another triplication to form the inputs of a gate in the next stage. The total interconnect patterns become 36 (6×6) if a distinction is made among the gate orders of a triplication in the next stage. One method to arrange the interconnections is to randomly adopt one of the 36 connection patterns for all connecting pairs in adjacent layers. As shown in Fig. 3.11(b), the interconnect patterns used in the two layers from inputs to outputs of the circuit are {(1,1), (2,2), (3,3)} and {(1,2), (2,3), (3,1)}. However, any other interconnect patterns can be used to link the inputs to the output.

It is worth mentioning that if the pattern {(1,1), (2,2), (3,3)} is used in all layers for

all interconnections, then the three modules in Fig. 3.11(b) will independently perform computation which is equivalent to a TMR circuit as shown in Fig. 3.9(b). Therefore, it can be concluded that the TIR technique is a general class of TMR implemented with random interconnections and the TMR is a particular configuration of TIR with regular interconnections. As in TMR, a TIR circuit requires a voter as a restoring device. TIR can also be extended to higher orders to give N-tuple interwoven redundancy (NIR), which is similar to the extension of TMR to NMR. Hence, NIR is a generalisation of NMR but with random interconnections. Similarly to TMR, TIR technique was also implemented at the logic gate level rather than the unit level in the simulations to improve its defect tolerance capability.

3.4.3 Quadded Logic

Quadded logic is a redundancy-based fault tolerance technique that was proposed in [Han et al., 2005, Tryon, 1962, Jensen, 1963]. This technique requires four times the circuit size to enable the correction of errors during the computation time. It can be implemented using AND, OR, NOR or NAND logic. To illustrate quadded logic, Fig. 3.12(a) shows the schematic of a logic circuit and its quadded form is shown in Fig. 3.12(b) both implemented with NAND gates. The implementation of NAND quadded logic requires replacing each NAND gate 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 the succeeding stage. The interconnections in a quadded circuit are hence eight times as many as those used in the non-redundant form.

In general, in a circuit implemented using quadded logic technique, any single critical error that occurs in the circuit will be eliminated after passing through two stages, and a single subcritical error will be corrected in the next stage after its occurrence i.e. errors in a quadded circuit are corrected at most after two logic stages [Han et al., 2005]. However, errors occurring on the circuit's edge might not be corrected. In other words, critical errors that occur within the last two layers and sub-critical errors that occur within the last layer will not be corrected. Therefore, the gates on the edge of a quadded logic circuit will impose a critical limit to the circuit fault tolerance i.e. any failure in the edge of a quadded logic circuit will cause the whole circuit to fail [Han et al., 2005].

In quadded logic, the interconnect patterns have a significant effect on the error correction capability of the technique. To ensure single error tolerance, the interconnect pattern at the outputs of a stage must differ from the interconnect pattern of any of its inputs [Han et al., 2005]. The interconnect set pattern can be used to determine the set of gates to which an interconnect is connected. The inputs of every equivalent four gates, such as $G11$,$G12$,$G13$ and $G14$ in Fig. 3.12(b), are divided into two sets. Each set

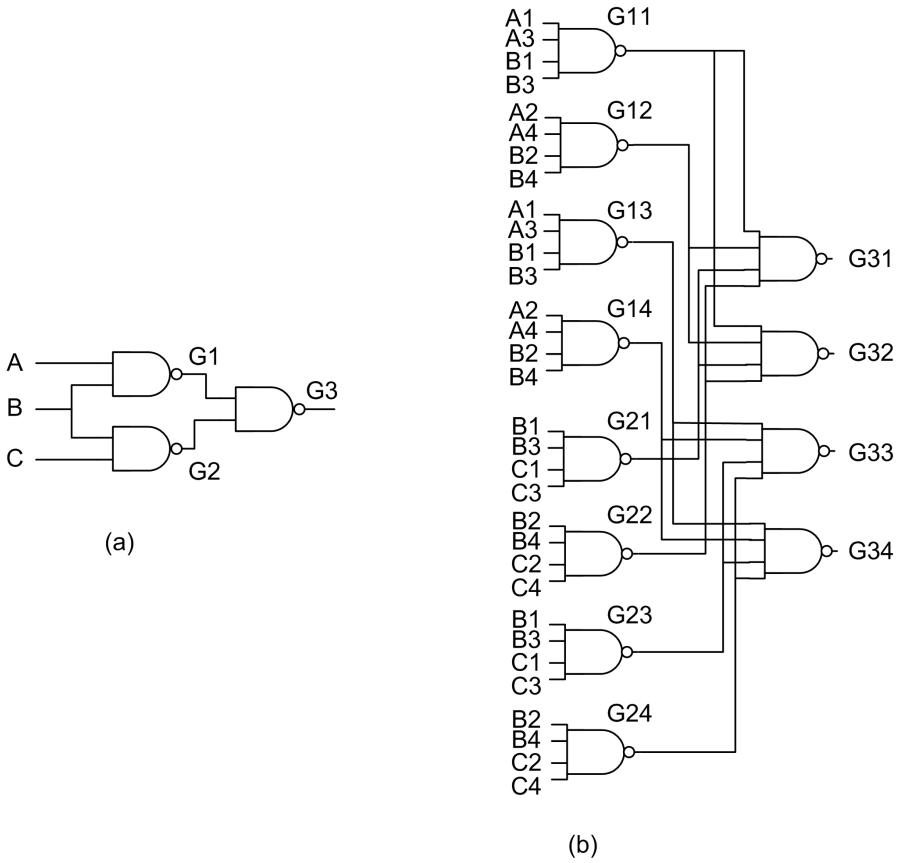


FIGURE 3.12: Quadded Logic architecture: (a) Original circuit, (b) Quadded logic circuit

forms a pair of inputs and each pair feeds the two gates with the same numbers as the set. For example, the set pattern for input A with respect to gate $G1$ is (1,3),(2,4) which indicates that inputs $A1$ and $A3$ should be connected to gates $G11$ and $G13$, while inputs $A2$ and $A4$ should be connected to gates $G12$ and $G14$. The interconnect set pattern for gate $G3$ is (1,2),(3,4), which is different from the set pattern of any of its inputs. Note that the output $G31$ is equal to $A1A3B1B3 + A2A4B2B4 + B1B3C1C3 + B2B4C2C4$. This guarantees the tolerance of any single error in any of the interconnects of the inputs A, B, and C.

3.5 Experimental Results

To demonstrate the effectiveness of the N^2-transistor structure technique, experiments were performed on a number of the largest ISCAS'85 and ISCAS'89 benchmark circuits, details of which can be found in Section A.2, Appendix A. Two types of permanent defects are analysed separately: transistor stuck-open and stuck-short defects, and AND/OR bridging defects (Chapter 1, Section 1.2.1). The defect tolerance capabilities of both the quadded transistor (QT) and the nonuple transistor (NT) structures are compared with the techniques outlined in Section 3.4 including Triple Modular Re-

dundancy (TMR), Triple Interwoven Redundancy (TIR) and Quadded Logic (QL). For comparison purposes, these redundancy-based techniques are used as defect tolerance rather than fault tolerance techniques in this chapter.

3.5.1 Stuck-Open and Stuck-Short Defect Analysis

To evaluate circuit failure probability and yield, the simulation-based model used in [Han et al., 2005] was adopted. A complete test set T that detects all detectable single stuck-at faults in a circuit was used. Test sets generated by Mintest ATPG tool [Hamzaoglu and Patel, 1998] were used. To compute the circuit failure probability F_m, resulting from injecting m defective transistors, the following procedure was 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 $F_m = K/1000$

Fig. 3.13 shows the variations in the failure rate of ISCAS c880 benchmark circuit, implemented using the proposed quadded transistor structure, in the presence of upto 100 randomly injected stuck-at faults. Both stuck-open and stuck-short faults were randomly injected with equal probabilities. The simulation procedure outlined above was used to compute the failure rate. As can be observed, the failure rate is close to 0% for small numbers of injected defects. However, as the number of injected defects increases above 40 which is equivalent to 0.5% of the circuit size, the failure rate starts to increase rapidly.

In Fig. 3.14, ISCAS c1355 benchmark circuit was built using three different defect-tolerance techniques including quadded logic, TMR and the proposed quadded transistor structure. The figure compares the efficiency of these techniques in the presence of upto 100 stuck-at defects. It is clearly shown that the DTQT structure exhibits the best performance as compared to the other techniques whereas the TMR technique exhibits the worst tolerance to the injected defects. For example, in the presence of only 20 stuck-at faults, TMR completely fails which is demonstrated by a failure rate of 100%, however the proposed DTQT technique achieves a failure rate of nearly 0%. The quadded logic

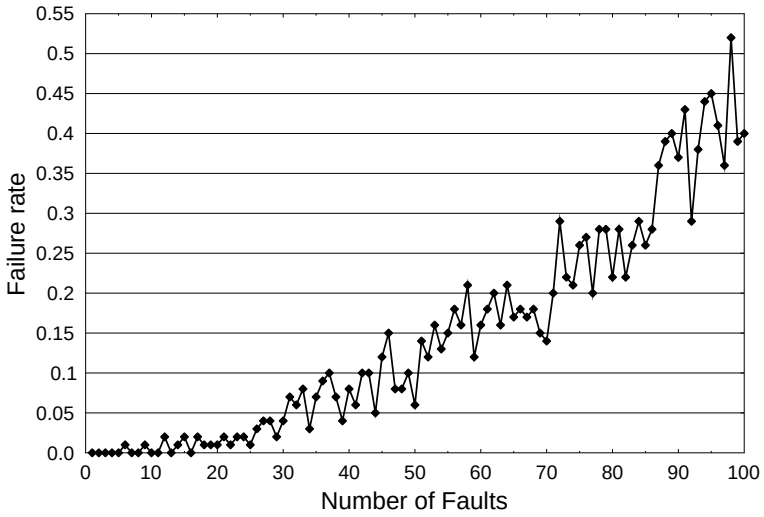


FIGURE 3.13: Failure rate of ISCAS c880 implemented using DTQT due to stuck-at faults

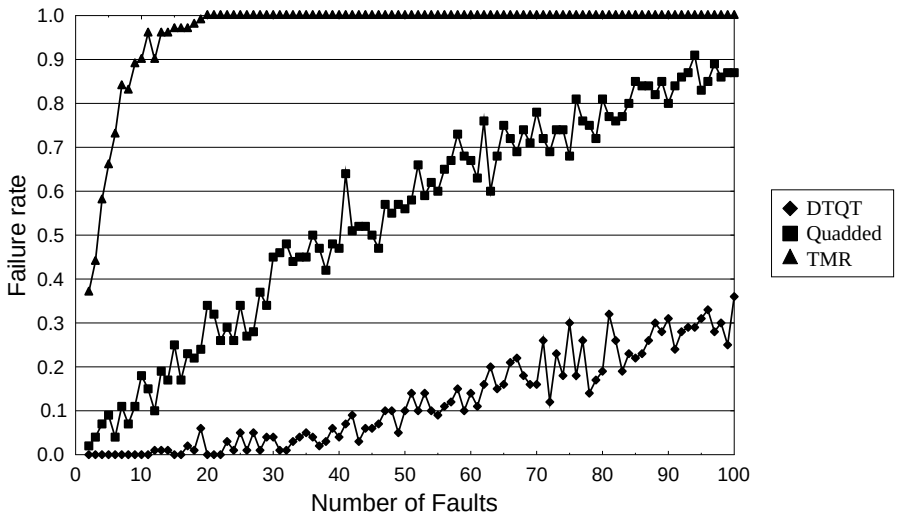


FIGURE 3.14: Failure rates of c1355 implemented using DTQT, TMR and Quadded structures due to stuck-at faults

technique also exhibits low yield values. The failure rate obtained by this technique is very high when 100 defects are injected (close to 90%) whereas less than 30% of the simulations fail when using the proposed DTQT structure. Comparing Figs. 3.13 and 3.14, it can be observed that the quadded transistor technique achieves lower failure rate in the case of c1355 as compared to c880. This is because of the difference in the size of the two benchmark circuits. Injecting 100 faults into c880, composed of only 7208 transistors for instance, represents a higher defect rate as compared to injecting

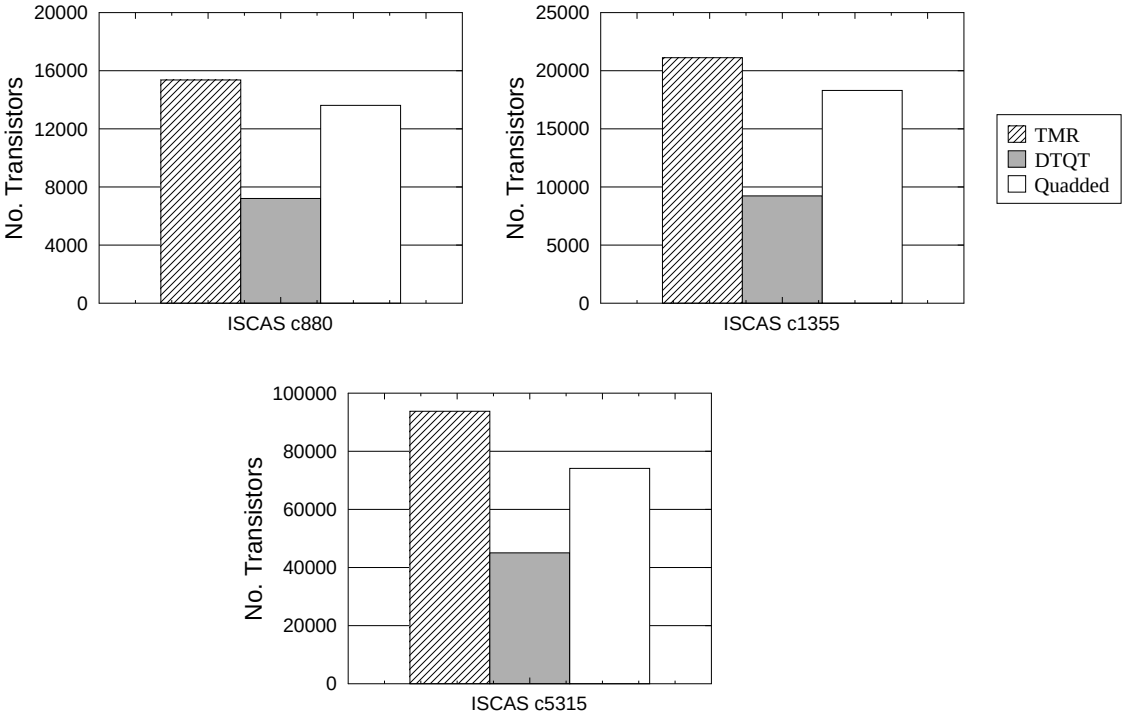


FIGURE 3.15: Comparison of area overhead between DTQT, Quadded Logic and TMR

the same number of faults into c1355 which is composed of 9232 transistors.

Fig. 3.15 compares the area overhead incurred by the TMR, quadded logic and quadded transistor techniques for three ISCAS benchmark circuits (c880, c1355 and c5315). As can be seen, the proposed DTQT structure which adds redundancy at the transistor level incurs the lowest area overhead as compared to quadded logic and TMR techniques that add redundancy at the logic gate level. The largest area overhead is incurred by TMR followed by quadded logic technique. It is worth noting that the number of transistors required by the proposed quadded transistor structure is less than half of that of TMR which indicates the efficiency of the proposed DTQT technique in terms of achieving higher yield values at a reduced area cost.

In Fig. 3.16, the probability of circuit failure of the quadded transistor structure (QT), nonuple transistor structure (NT), quadded logic (QL) and TIR logic are compared in the presence of different percentages of stuck-open and stuck-short defects. The percentage of defects is calculated by dividing the number of defects injected over the total number of transistors present in the circuit. The comparison is made based on an 8-stage cascaded half adder circuit used in [Han et al., 2005]. As can be observed, the failure rate obtained by the quadded transistor structure implementation is significantly lower than the failure rates obtained by the TIR and quadded logic techniques especially in the case of low defect rates. For instance, for a defect rate of 2%, the quadded

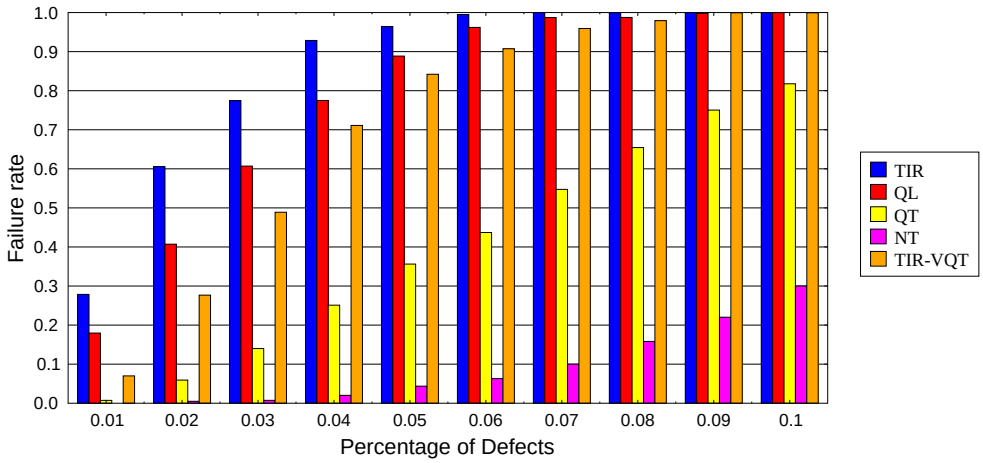


FIGURE 3.16: Comparison of circuit failure probability for an 8-stage cascaded half-adder circuit for stuck-open and stuck short defects

transistor structure achieves a failure rate as low as 6% whereas the quadded logic and TIR techniques achieve failure rates of 41% and 61% respectively. Furthermore, the yield of the proposed N^2-transistor structure can be further improved by using the nonuple transistor structure. As can be observed from Fig. 3.16, the failure rate values obtained by the nonuple transistor structure are significantly lower than the values exhibited by the quadded transistor structure for all the defect rates under consideration. As an example, for a defect rate of 5%, the failure rate obtained by the quadded transistor structure is 35.6% whereas it is less than 5% for the nonuple transistor structure.

As outlined in Section 3.4.1, the probability of circuit failure for TIR and TMR techniques can be significantly improved by improving the reliability of the majority voters. The majority voters in the 8-stage cascaded half adder circuit built using TIR technique was implemented based on the quadded transistor structure (TIR-VQT). As shown in Fig. 3.16, the effectiveness of the implemented circuit in combating stuck-open and stuck-short defects is significantly improved as compared to the TIR technique especially in the presence of low defect rates. However, this improvement is obtained at the cost of an increase in the number of transistors (608 transistors for TIR technique and 1280 transistors for TIR-VQT technique). It can also be observed from Fig. 3.16 that as the defect rate is increased, the failure rate of the TIR-VQT technique increases rapidly. This is due to the failures in the individual modules of the TIR technique which also need further improvement in terms of defect tolerance. Therefore, a balance between the size of the modules and the majority gates used in the TIR-VQT technique need to be made in order to achieve acceptable levels of defect tolerance at reduced area overheads.

Comparison

A comprehensive comparison of the probability of circuit failure between the quadded transistor structure and the quadded logic is given in Table 3.2 for several percentages of injected stuck-open and stuck-short defects. As can be observed, for all the ISCAS benchmark circuits, the quadded transistor technique achieves significantly lower circuit failure probabilities than the quadded logic technique for the same and for twice the percentage of injected defects. For ten out of the twelve circuits under consideration, DTQT achieves lower failure probabilities with four times the percentage of injected defects. For instance, for the circuit c1355, the failure rate obtained by the proposed quadded transistor structure is 28.70% at a defect rate as high as 1%, whereas the quadded logic technique achieves a higher failure rate of 53.10% at a defect rate of only 0.25%. In Table 3.3, the yield results obtained by the quadded transistor structure and the quadded logic technique for several percentages of injected stuck-open and stuck-short defects are reported. The effectiveness of the quadded transistor structure in tolerating higher defect rates is clearly demonstrated by the yield values exhibited by this technique as opposed to quadded logic technique. The proposed technique achieves higher yield values with 4 to 5 times more transistor defect probability. For example, in the case of c880 circuit, the proposed quadded transistor structure achieves 93.40% of yield in the presence of 0.5% of defects whereas the quadded logic technique achieves only 82.20% of yield in the presence of 0.1% of defects.

In Table 3.4, the circuit failure rates of the nonuple transistor structure are reported for several transistor defect probabilities including stuck-open and stuck-short defects. As can be observed, the level of yield exhibited by this structure is significantly improved when compared to the results shown in Table 3.2. This is demonstrated by the failure rates achieved by the nonuple transistor structure which are more than ten times smaller as compared to the quadded transistor structure. As an example, for the s15850 circuit and in the presence of 1% of defects, the obtained failure rate is 99.20% when using DTQT structure, however this value is reduced to only 11.10% when the nonuple transistor technique is used. It is worth noting that the nonuple transistor structure increases the area overhead by 2.25 times as compared to the quadded transistor technique.

3.5.2 Bridging Fault Analysis

To analyse the defect tolerance of quadded-transistor structure to bridging defects (Chapter 1, Section 1.2.1), the same simulation-based model outlined in Section 3.5.1 is used. Both inter-gate and intra-gate bridging defects are taken into consideration. The experiments were performed on the same set of ISCAS benchmark circuits. The bridging defects were injected randomly between the gates of the defective transistor and one of its neighbours, located within a window of local transistors (± 8 transistors). Both AND and OR bridging defects were injected with equal probabilities. It should

Circuit	Quadded-Transistor structure [proposed]					Quadded Logic [Han et al., 2005]				
	#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	96816	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	103488	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

TABLE 3.2: Comparison of circuit failure probability between quadded transistor structure and quadded logic approaches for stuck-open and stuck-short defects

Circuit	Quadded-Transistor structure [proposed]					Quadded Logic [Han et al., 2005]				
	0.01%	0.1%	0.2%	0.5%	1%	0.01%	0.1%	0.2%	0.5%	1%
c880	0.999	0.997	0.989	0.934	0.767	0.979	0.822	0.651	0.283	0.042
c1355	0.999	0.996	0.986	0.917	0.713	0.075	0.765	0.575	0.187	0.008
c1908	0.999	0.994	0.979	0.879	0.596	0.975	0.755	0.558	0.061	0.001
c2670	0.999	0.991	0.967	0.809	0.427	0.904	0.350	0.112	0.001	0.000
c3540	0.999	0.989	0.956	0.755	0.327	0.981	0.805	0.614	0.007	0.000
c5315	0.999	0.984	0.935	0.656	0.185	0.853	0.227	0.034	0.001	0.000
c6288	0.999	0.986	0.941	0.685	0.222	0.971	0.718	0.465	0.024	0.000
c7552	0.999	0.978	0.912	0.562	0.101	0.874	0.292	0.077	0.000	0.000
s5378	0.999	0.985	0.948	0.717	0.263	0.811	0.134	0.015	0.001	0.000
s9234	0.999	0.972	0.894	0.496	0.061	0.821	0.140	0.001	0.000	0.000
s13207	0.999	0.961	0.856	0.379	0.023	0.518	0.008	0.000	0.000	0.000
s15850	0.999	0.953	0.825	0.302	0.008	0.576	0.009	0.000	0.000	0.000

TABLE 3.3: Comparison of yield between quadded transistor structure and quadded logic approaches for stuck-open and stuck-short defects

Circuit	#Trans.	0.25%	0.5%	0.75%	1%	2%	5%
c880	16218	0.000	0.001	0.003	0.006	0.052	0.547
c1355	20772	0.001	0.002	0.004	0.009	0.066	0.637
c1908	31014	0.000	0.002	0.006	0.012	0.096	0.780
c2670	51012	0.001	0.003	0.008	0.021	0.153	0.917
c3540	67536	0.001	0.004	0.011	0.027	0.197	0.963
c5315	101358	0.001	0.005	0.018	0.041	0.281	0.993
c6288	91008	0.000	0.005	0.016	0.036	0.256	0.989
c7552	138600	0.001	0.007	0.024	0.056	0.363	0.998
s5378	80118	0.001	0.004	0.014	0.032	0.229	0.980
s9234	168426	0.002	0.009	0.029	0.065	0.422	0.999
s13207	232974	0.002	0.012	0.040	0.091	0.531	≈ 1
s15850	288036	0.002	0.015	0.049	0.111	0.608	≈ 1

TABLE 3.4: Circuit failure probability of ISCAS circuits implemented using nonuple transistor structure due to stuck-open and stuck-short defects

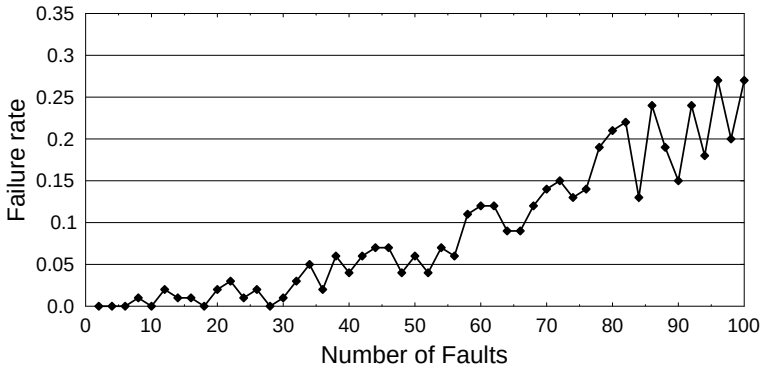


FIGURE 3.17: Failure rate of ISCAS c880 implemented using DTQT due to bridging defects

be observed that for injecting m defective transistors due to bridges, only $m/2$ bridges need to be injected.

Fig. 3.17 shows the variations in failure rate obtained by the proposed quadded transistor structure for the ISCAS c880 circuit in the presence of bridging defects. As can be observed, the DTQT technique exhibits a better tolerance to bridging faults than to stuck-at faults. This is illustrated by the fact that only 25% of simulations fails when 100 transistors are considered defective in the c880 circuit due to bridging faults, however in Fig. 3.13, more than 50% of the simulations fail due to stuck-at faults. In Fig. 3.18, the efficiency of quadded transistor and quadded logic techniques in combating bridging defects are compared. The DTQT structure outperforms the quadded logic approach regardless of the number of injected bridging defects. Generally, the DTQT structure achieves failure rates that are half the values obtained by the quadded logic. For instance, when 50 bridging faults are injected, the circuit failure rate of the quadded transistor structure is 17.5%, however, for the quadded logic implementation it is close to 35%.

Circuit	Quadded-Transistor structure [proposed]					Quadded Logic [Han et al., 2005]				
	#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	96816	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	103488	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

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

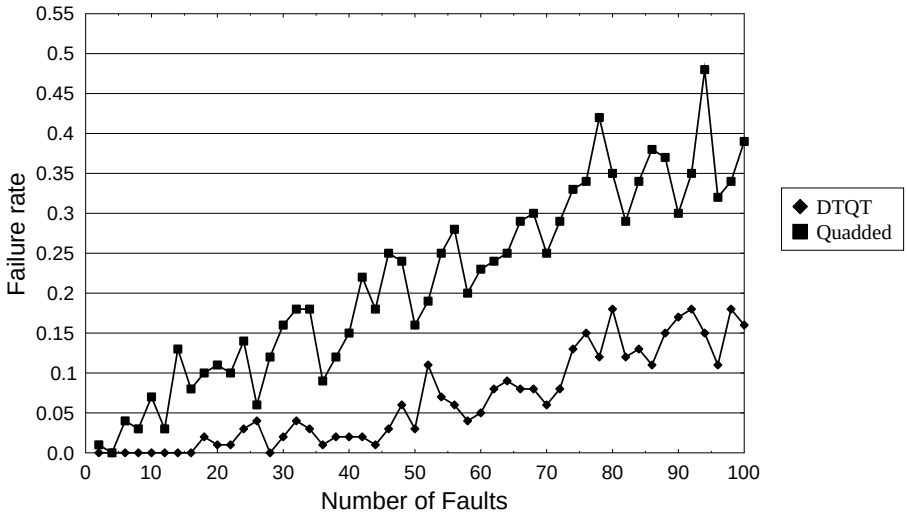


FIGURE 3.18: Failure rates of c1355 implemented using DTQT and Quadded structures due to bridging faults

Table 3.5 shows the results obtained for several percentages of injected bridging defects for the quadded transistor and quadded logic techniques. The quadded transistor structure exhibits a much lower failure probability than the quadded logic technique. The failure rates achieved by the quadded transistor structure are lower than the values obtained by the quadded logic technique for the same and twice the percentage of injected bridging faults. As an example, for 0.25% of injected defects, the proposed DTQT technique 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 rates for bridging defects are less than those of stuck-open and stuck-short defects (see Table 3.2). This is because not all bridging defects result in a faulty gate behaviour.

In Table 3.6, the circuit failure rates obtained by the nonuple transistor structure for different bridging defect rates are tabulated. As compared to the results shown in Table 3.5, this technique achieves an unprecedented level of defect tolerance against bridging defects. This is demonstrated by failure rate values that are significantly lower than the values obtained by the quadded transistor structure. While the quadded transistor structure becomes inefficient in tolerating bridging defect rates higher or equal to 1% especially as the circuit size increases, the nonuple transistor structure is capable of achieving more than 97% of yield values regardless of the circuit's size. It is worth noting that by comparing Tables 3.4 and 3.6, it can be observed that for the same defect probability, the nonuple transistor structure technique exhibits better tolerance to bridging defects than to stuck-at faults.

Circuit	#Trans.	0.25%	0.5%	0.75%	1%
c880	16218	0.000	0.000	0.000	0.002
c1355	20772	0.000	0.000	0.000	0.007
c1908	31014	0.000	0.000	0.000	0.009
c2670	51012	0.000	0.003	0.004	0.008
c3540	67536	0.000	0.001	0.005	0.008
c5315	101358	0.000	0.005	0.003	0.019
c6288	91008	0.000	0.001	0.010	0.017
c7552	138600	0.000	0.005	0.009	0.023
s5378	80118	0.000	0.001	0.007	0.017
s9234	168426	0.000	0.004	0.005	0.020
s13207	232974	0.000	0.001	0.008	0.013
s15850	288036	0.000	0.002	0.009	0.018

TABLE 3.6: Circuit failure probability of ISCAS circuits implemented using nonuple transistor structure due to bridging defects

3.6 Conclusion

In this chapter, a new defect tolerance technique called the N^2-transistor structure that is based on adding redundancy at the transistor level was investigated. Two particular cases of this technique named quadded transistor and nonuple transistor structures were evaluated in terms of tolerating both stuck-at and bridging faults. Thorough simulation procedures were applied to a large set of ISCAS'85 and ISCAS'89 benchmark circuits to obtain accurate results. Experimental results have demonstrated that the N^2-transistor structure exhibits an outstanding tolerance to both stuck-at and bridging defects. The yield values exhibited by the proposed technique are significantly higher than the previously reported techniques that are based on gate-level redundancy such as quadded logic and TMR. This improvement in defect tolerance is achieved at a reduced area overhead as compared to the other reported techniques.

Chapter 4

Repair Techniques for Nano/CMOS Architecture

In the previous chapter, a defect tolerance technique that improves manufacturing yield in nanometre CMOS architecture was proposed. This chapter presents two new repair techniques, named Tagged Replacement and Modified Tagged Replacement, that provide high level of defect tolerance for hybrid nano/CMOS computational architecture. Nanoscale LUT-based implementation of logic circuits is considered in this chapter. The proposed techniques proved their capability of handling upto 20% defect rates which is higher than recently reported repair techniques. The proposed techniques are also efficient in utilisation of spare units. Repair is performed using a tagging mechanism for bad line exclusion. The theoretical equations that predict their repair capability including an estimate of their repair costs are derived. The impact of defect distribution on their repair capability is also investigated.

The chapter is organised as follows: Section 4.1 provides background on repair techniques and also outlines the targeted hybrid nano/CMOS architecture. Section 4.2 presents a recent repair technique called Repair Most and investigates its suitability for LUT-based logic implementation since it will be used as a reference point for the evaluation of the new repair techniques. In Sections 4.3 and 4.4, the design and implementation of the proposed techniques are presented. The experimental results for each technique are reported in its corresponding section. The yield improvement obtained by the proposed repair techniques when used to implement large ISCAS'85 benchmark circuits is highlighted in Section 4.5. The repair costs incurred by the proposed techniques are estimated in Section 4.6 and Section 4.7 outlines the impact of defect distribution on their repair capabilities. Section 4.8 concludes the chapter.

All the work presented in this chapter is the sole contribution of the author.

4.1 Introduction

Chemically assembled nanodevices are believed to be the current complement and the future possible alternative to CMOS-based computing [Brown and Blanton, 2007, Copen Goldstein and Budiu, 2001, Butts et al., 2002, Mishra and Goldstein, 2003]. Due to the fabrication regularity imposed by the chemical-assembly process, molecular circuits are restricted to regular and periodic crossbar structures, as outlined in Chapter 1, Section 1.1. Most of the earlier works in nano/CMOS design have targeted the crossbar architecture [Zhang et al., 2006, Strukov and Likharev, 2005a,b]. This reconfigurable architecture, however, suffers from one major drawback which is its unreliability due to the high physical defect rates in nanoscale fabrics (Chapter 1, Section 1.2.2). While the exact manufacturing defect rate is not yet pinpointed, it is believed to exceed 10% [Stan et al., 2003]. These defects are introduced during manufacturing rather than during operation [Hogg and Snider, 2007]. Hence, rather than attempting to eliminate all the defects completely during manufacturing, defect tolerance is needed to make such nanofabrics commercially viable [Stan et al., 2003, Jacorne et al., 2004, Tahoori, 2005b, DeHon and Naeimi, 2005, Tahoori, 2006b, Mishra and Goldstein, 2003]. Manufacturing yield can be increased by devising new post-fabrication reconfiguration-based repair techniques to allow the chip to work in spite of its defects.

To date, one of the most popular approaches to improving yield is by using redundant elements to replace faulty memory elements. In [Li and Zhang, 2009], the authors proposed a hybrid defect tolerance strategy that is based on coarse-grained redundant repair (i.e. replacing defective LUTs with redundant non-defective LUTs). In this chapter, the proposed repair techniques implement defect tolerance at the wire level (i.e. fine-grained redundant repair) rather than the LUT level. This helps target higher defect rates since it provides the maximum resolution of repair. The most common types of wire-based redundancy include row-based redundancy and column-based redundancy where entire redundant rows or columns are used to replace the faulty cells. The effectiveness of the replacement technique depends on the type and the amount of redundancy and also on the redundancy analysis algorithm [Strukov and Likharev, 2005a, Huang et al., 2003]. This chapter focuses on nanoscale crossbar architecture implementing logic circuits as look-up tables (Chapter 1, Section 1.1.1). The main type of defects to be tackled are junction defects including stuck-open and stuck-short defects (Chapter 1, Section 1.2.2). To achieve acceptable levels of manufacturing yield for nano/CMOS architecture, efficient repair techniques need to be implemented [Mishra and Goldstein, 2003]. There is a large body of literature available for efficient CMOS memory repair techniques, recent examples include [Lu and Hsu, 2006, Chang et al., 2008]. The aim of this chapter is to use the available literature to find efficient techniques that can exclude defective rows and columns in LUTs to obtain the best manufacturing yields using minimum spares in both dimensions. The proposed design flow is shown in Fig. 4.1 where defect tolerance is achieved by designing around defective resources but without the need to be

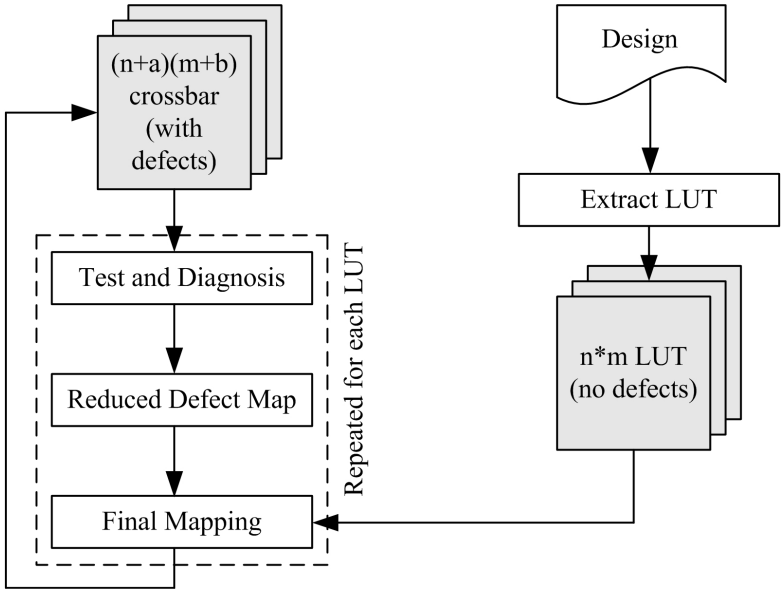


FIGURE 4.1: Proposed defect-unaware design flow

aware of the exact locations of the unusable bits i.e. defect unaware flow. The repair techniques to be proposed in this chapter do not require defect mapping before repair because the size of the defect map for the entire nanofabric can be prohibitively large (10^{12} devices/cm^2 [Tahoori, 2006a, Ziegler and Stan, 2003, Tahoori, 2005b]). A reduced defect map is created by counting the number of defects per rows and columns and then one of the proposed repair techniques (Tagged Replacement or Modified Tagged Replacement) is applied. The reconfigurable aspect of crossbars is exploited in the testing and diagnosis phase of the proposed design flow.

Hybrid nano/CMOS architecture has shown promise in bridging the gap between CMOS and emerging nanotechnologies [Ziegler and Stan, 2003, Jeffery et al., 2004, DeHon et al., 2005, Sun and Zhang, 2007]. As mentioned in Chapter 2, Section 2.3, a number of promising architectures have been proposed for the hybrid nano/CMOS design paradigm. In order to advance computational nanocircuits, new designs must be pursued. One such promising computational design is the LUT-based nano/CMOS architecture where Boolean logic functions are implemented as LUTs. Fig. 4.2 illustrates this architecture where unreliable but highly dense nanodevices are used to provide data storage and computation while CMOS devices are utilised for interfacing and for highly critical circuit operations.

The simulation procedure used to calculate the manufacturing yield exhibited by the proposed repair techniques is based on Monte Carlo simulation procedure. Simulations are performed on 5000 randomly-generated symmetric LUTs where the probability of 0 and 1 are equal. The LUTs under consideration are of sizes ranging from $2^3 \times 3$ (3 inputs, 3 outputs) to $2^8 \times 8$ (8 inputs, 8 outputs). It is assumed that a nanodevice is subject to both stuck-open and stuck-short defects with equal probabilities and all the defects are

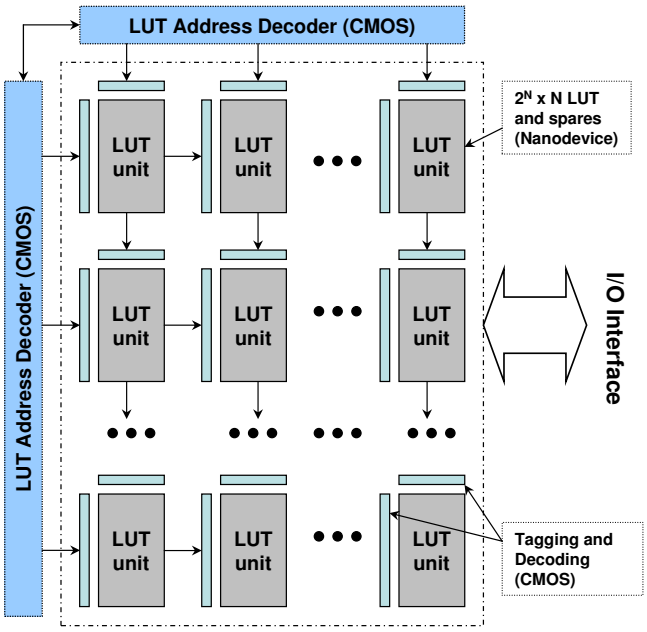


FIGURE 4.2: Implementation of LUT based Boolean logic approach using nano/CMOS architecture

randomly and uniformly distributed across the fabric causing the corresponding bits to change their values (i.e. $1 \rightarrow 0$ or $0 \rightarrow 1$).

To calculate the yield ratio of a certain repair technique after injecting d defects, the following procedure is used:

1. Set the number of iterations to be performed, I, to 5000 and the number of successful simulations, K, to 0.
2. Generate a random $m \times n$ LUT.
3. Allocate a spare rows and b spare columns for repair.
4. Randomly inject d defects in the $(n + a) \times (m + b)$ matrix.
5. Apply the repair algorithm.
6. If the $m \times n$ LUT is successfully instantiated in the $(n + a) \times (m + b)$ matrix, increment K by 1.
7. Decrement I by 1 and if I is not 0 goto step 2.
8. $Yield = K/5000$

The failure rate is calculated as follows:

$$P_{failure} = 1 - Yield \tag{4.1}$$

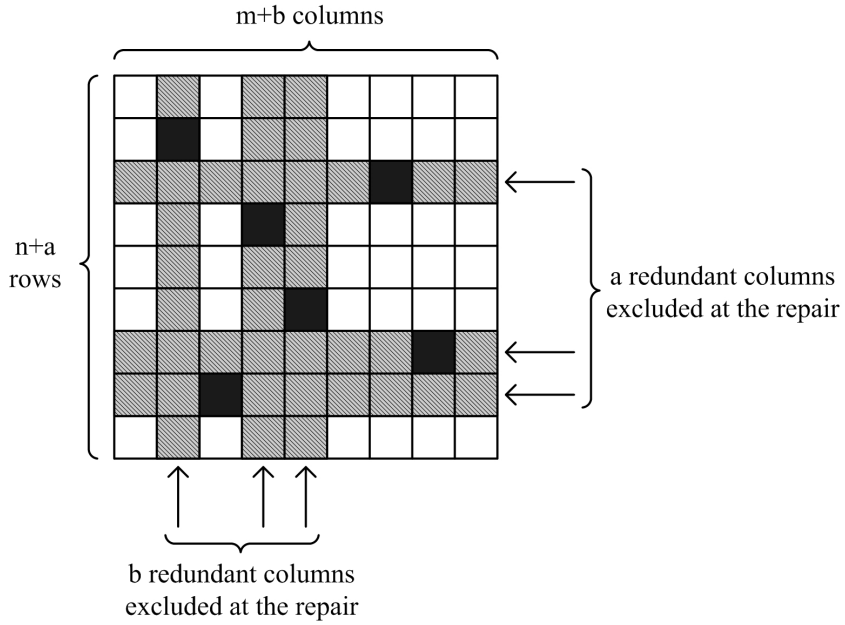


FIGURE 4.3: Illustration of Repair Most technique

4.2 Repair Most Technique

Repair Most is a repair technique that was proposed as an efficient approach to tolerate physical defects in memory designs [Lu and Hsu, 2001, Strukov and Likharev, 2005a]. In this section, this technique is applied to LUT-based nano/CMOS architecture. The aim is to analyse the capability of the Repair Most in identifying a defect-free instance of a LUT of size $m \times n$ within a defective fabric of size $(n + a) \times (m + b)$ where a and b are the number of spare rows and spare columns respectively as illustrated in Fig. 4.3. The rows and columns on which defective bits exist are deleted. The repair process comprises of two phases: In the first phase of the technique, if the number of defects per row (or column) exceeds the threshold r_{th} (or c_{th}), the row (or column) will be excluded from the LUT and replaced with a spare row (or column). r_{th} (or c_{th}) are predefined parameters that represent the maximum number of errors that are allowed to be present in a row (or column) for it to be not excluded. In the second phase, the remaining defective cells are repaired by replacing the defective columns (or rows) with the spare columns (or rows) as shown in Fig. 4.4(b). Redundancy (or spares) is the percentage of extra rows/columns that are allocated for repair. As an example in Fig. 4.4(a), in case of a $2^N \times N$ LUT, 25% redundancy implies an extra 0.25×2^N rows and $0.25 \times N$ columns respectively. Hence the total size of the defective LUT implementation for 25% redundancy becomes 1.25×2^N rows and $1.25 \times N$ columns.

In the Repair Most technique, defective lines are excluded in both dimensions arbitrarily. Since in a $2^N \times N$ LUT the number of rows 2^N and columns N are not equal, the impact of the order of line exclusion on the instantiation probability needs to be examined. Fig. 4.5(a) shows the failure rate obtained from excluding rows first then columns for

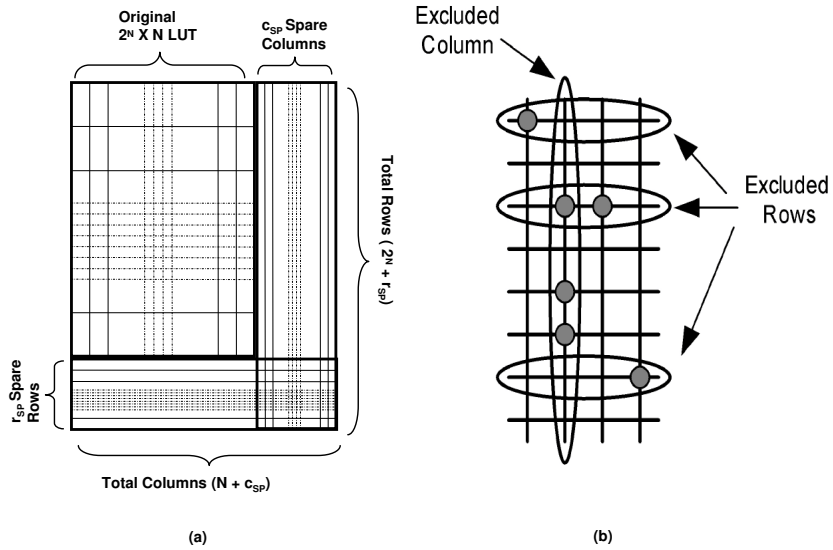


FIGURE 4.4: Repair Most technique (a) Implementation for LUT based nano/CMOS architecture (b) Repair mechanism by setting value of $c_{th} = 2$ and $r_{th} = 0$. The columns (rows) with number of defects $> c_{th}$ (r_{th}) are excluded

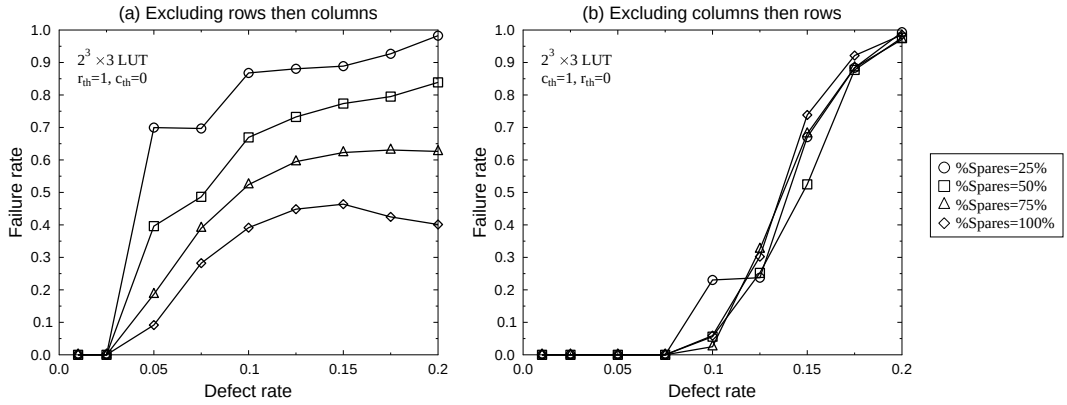


FIGURE 4.5: Repair Most - The order of excluding rows and columns significantly affects the yield

a small $2^3 \times 3$ LUT given different percentages of spares and $r_{th} = 1$ and $c_{th} = 0$. As can be observed, as the percentage of allocated spare rows and columns for repair is increased, the failure rate decreases; however this implementation of the technique can only achieve 40% failure rate at 10% of injected defects. The failure rate obtained from excluding columns first then rows is shown in Fig. 4.5(b). As can be seen, the yield is significantly improved given the small amount of spares needed to achieve small failure rates. As an example, 50% of spare rows and columns is sufficient to achieve less than 5% of failure rate in the presence of upto 10% of defect rates. The exclusion of columns before rows allows the removal of a larger number of defects at the initial repair phase.

4.2.1 Theoretical Analysis of Failure Rate (Repair Most Technique)

Next, the failure probability $P_{failure}$ of instantiating LUTs on defective nanofabrics using the Repair Most technique given the bit defect probability P is theoretically determined. If the number of defects in a column exceeds the threshold c_{th}, the line is considered defective and has to be replaced with a spare column. Therefore, the probability P_{col} of a column being defective is:

$$P_{col} = \sum_{k=c_{th}+1}^{r+r_{sp}} \binom{r+r_{sp}}{k} P^k (1-P)^{r+r_{sp}-k} \tag{4.2}$$

where r and r_{sp} are the number of rows and spare rows in the LUT respectively. In order to get a defect-free LUT, the row threshold r_{th} is set to 0 so that all the remaining defects in the fabric are removed. A row is excluded if it contains one or more defects and this is given by the following equation:

$$P_{row} = 1 - \Big(1 - P(1 - P_{col})\Big)^{(c+c_{sp})(1-P_{col})} \tag{4.3}$$

where c and c_{sp} are the number of columns and spare columns in the LUT respectively. The probability of failure to instantiate a LUT on the fabric given the number of spare rows and columns is computed using the following equation:

$$P_{failure} = \sum_{k=c_{sp}+1}^{c+c_{sp}} \binom{c+c_{sp}}{k} P_{col}^k (1-P_{col})^{c+c_{sp}-k} +$$

$$\sum_{k=r_{sp}+1}^{r+r_{sp}} \binom{r+r_{sp}}{k} P_{row}^k (1-P_{row})^{r+r_{sp}-k} \times \Big(1 - \sum_{k=c_{sp}+1}^{c+c_{sp}} \binom{c+c_{sp}}{k} P_{col}^k (1-P_{col})^{c+c_{sp}-k}\Big) \tag{4.4}$$

The yield ratio is calculated as follows:

$$Yield = 1 - P_{failure} \tag{4.5}$$

The failure rates obtained theoretically and experimentally are plotted in Fig. 4.6. As can be seen, there is a very close match between the two graphs which validates the derived theoretical equations.

The implementation of the Repair Most technique is based on predefined parameters c_{th} and r_{th}. The parameter r_{th} is always set to 0 because the second phase of repair dictates removing the remaining defects in the fabric after column exclusion. However,

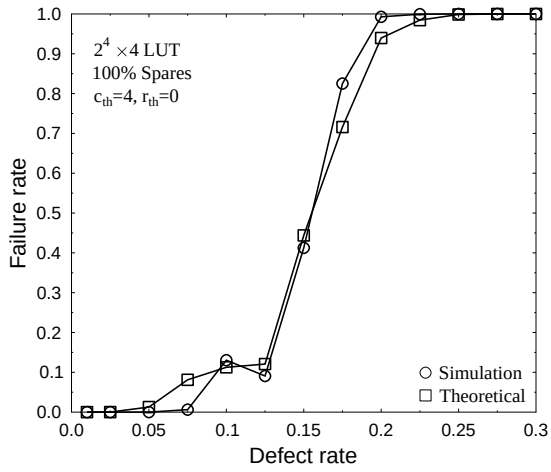


FIGURE 4.6: Repair Most - Failure rate obtained both theoretically and experimentally

one of the limitations of the Repair Most technique is that the choice of the value of c_{th} can significantly affect the efficiency of this technique. Fig. 4.7 illustrates this effect. As can be observed, for each defect rate there is a value for the parameter c_{th} to obtain the minimum failure rate. For a LUT size of $2^4 \times 4$ and a defect rate of 10% for instance, $c_{th} = 3$ gives the smallest possible failure rate of instantiation and at 15% of defect rate, $c_{th} = 5$ gives the smallest failure rate. c_{th} also varies with the size of LUT, hence for each LUT size and each defect rate there exist an optimum c_{th} value that guarantees the best repair rate.

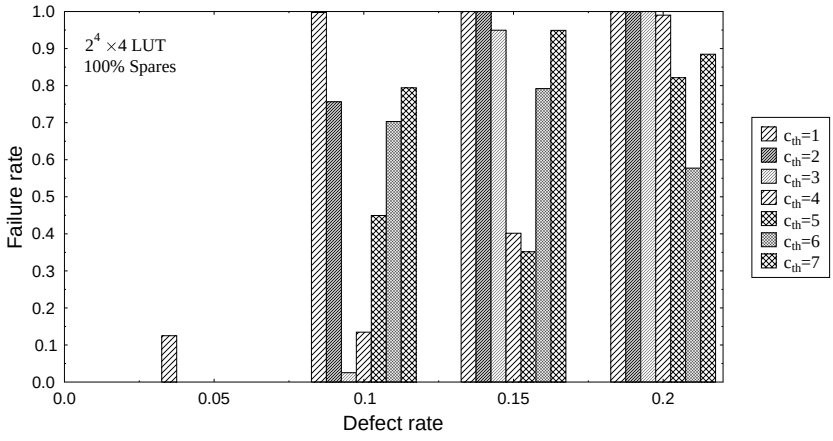


FIGURE 4.7: Repair Most - Effect of varying c_{th} on failure rate

Table 4.1 demonstrates that as the number of defects increases, more spares are needed to keep the level of defect tolerance lower than 5%. In the case of $2^3 \times 3$ LUTs, only 75% of spare rows and columns are needed to tolerate upto 20% of defect rates. It should

be observed that as the LUT size increases, the percentage of spares also increases to achieve low failure rates. As an example, for $2^6 \times 6$ LUTs, the Repair Most technique requires a huge amount of spare lines equal to 1100% of the original size of the LUT which significantly decreases the useful bit density offered by nanodevices to an unacceptable level. It can be concluded that in terms of area overhead, the Repair Most technique efficiently tolerates upto 10% of defect rates. It can also be observed from Table 4.1 that c_{th} has small values for small LUTs and in the presence of small defect rates, however, this parameter increases dramatically for defect rates higher than 10% which makes it more difficult to obtain through simulation.

Defect rate	$2^3 \times 3$ LUT		$2^4 \times 4$ LUT		$2^6 \times 6$ LUT	
	%Spares	c_{th}	%Spares	c_{th}	%Spares	c_{th}
1%	25%	1	25%	1	25%	1
5%	25%	1	50%	2	75%	6
10%	50%	1	100%	3	200%	18
15%	50%	3	175%	6	400%	44
20%	75%	4	300%	11	1100%	140

TABLE 4.1: Repair Most - The required amount of spares and c_{th} to achieve failure rates less than 5%

4.2.2 Effect of Don't Cares

The effect of injecting Don't Cares on the efficiency of the Repair Most technique is examined. Fig. 4.8 shows the failure rate obtained before and after injecting 50% of Don't Cares into a $2^4 \times 4$ LUT. As can be seen, the presence of DCCs can significantly improve the tolerance against physical defects. This is illustrated by the fact that the range of defect rate up to which the failure rate is nearly 0% is doubled from 7.5% before injecting DCCs to 15% after injecting them for the same percentage of spare lines.

Theoretically, the existence of DCCs (P_{DCC}) in LUTs can significantly reduce the bit failure rate as outlined in the following equation:

$$P' = P \times (1 - P_{DCC}) \tag{4.6}$$

The probability of a column to be defective P_{col} is given by the following equation:

$$P'_{col} = \sum_{\substack{k=0 \\ k'=0 \\ k+k'>c_{th}}}^{c_{th}} \binom{r}{k} P'^{k}(1-P')^{r-k} \times \binom{r_{sp}}{k'} P^{k'}(1-P)^{r_{sp}-k'} \tag{4.7}$$

The probability of a row being defective given P'_{col} is:

$$P'_{row} = 1 - (1 - P')^{(c+c_{sp})(1-P'_{col})} \tag{4.8}$$

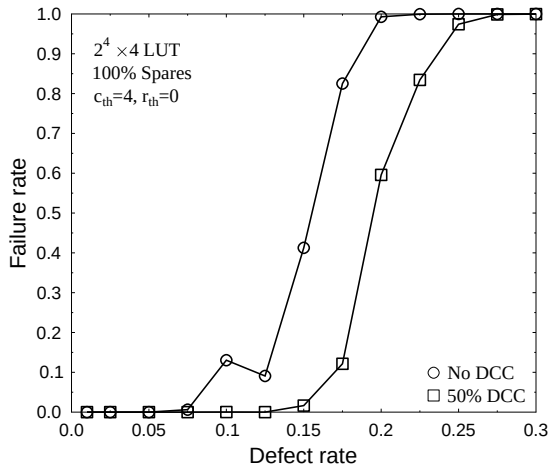


FIGURE 4.8: Repair Most - Impact of Don't Cares on failure rate

The total failure rate in the presence of DCCs is given by equation 4.4 and the yield is calculated using equation 4.5 after replacing P_{row} and P_{col} with P'_{row} and P'_{col} respectively. In order to validate these equations, simulation-based results were compared with the theoretical failure rate as shown in Fig. 4.9. Both graphs prove that the presence of DCCs extended the range of defect rates within which the failure rate is equal to 0%.

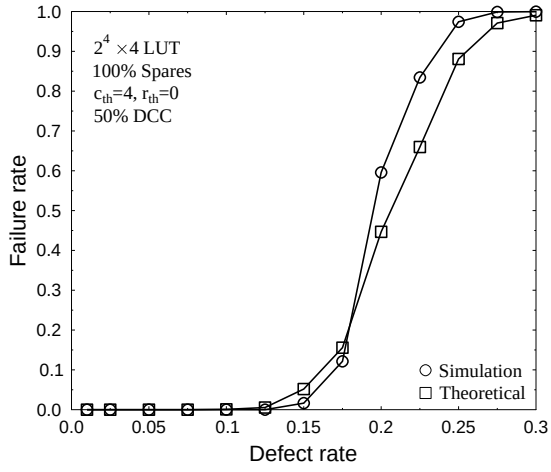


FIGURE 4.9: Repair Most - Failure rate obtained both theoretically and experimentally in the presence of Don't Cares

4.3 Proposed Tagged Replacement Technique

From the analysis carried out in Section 4.2, it can be concluded that the Repair Most technique suffers from two major drawbacks. The first one is its inefficiency in terms of area in tolerating physical defects. This technique requires unacceptable amounts of spare resources to tolerate defect densities higher than 10% (see Table 4.1) which significantly affects the useful bit density offered by nanodevices especially in the case of large LUTs such as $2^6 \times 6$ LUTs. The second drawback is that the repair rate exhibited by this approach is completely dependant on c_{th} parameter. It was demonstrated that a wrong choice of the value of c_{th} may result in a complete failure in LUT instantiation even if the appropriate amount of spare lines is allocated (see Fig. 4.7). This increases the complexity of the design process as this parameter varies with the rate of defects and also with the size of the LUT. Therefore, in this section, a new repair technique that have been specifically developed for LUT-based Boolean logic architecture is proposed which is independent of the size of the LUT and the defect rate of the fabric.

The general repair concept is derived from a memory repair technique that has been used in CMOS rather than nanodevice based systems [Lu and Hsu, 2006]. The original technique involves using blocks of spare memory units for replacing defective areas of CMOS memory. It also involves using three sets of register banks to store the row and column addresses as well as the address of current row and column bank. The architecture proposed in [Lu and Hsu, 2006] is not applicable by itself to LUT-based architecture targeted in this chapter because an individual LUT size is much smaller as compared to a highly dense memory design, hence the replacement of blocks of memory unit is not required. Moreover, the original architecture if applied to LUT-based architecture will impose a significant CMOS area overhead which will nullify the gain in device density achieved by using nanoscale components. Therefore, the following mechanism that makes the algorithm reported in [Lu and Hsu, 2006] more suitable to LUT-based nano/CMOS architecture is proposed. The modified algorithm involves replacing rows and columns instead of blocks of defective units. A tagging mechanism to isolate defective rows and columns is also included. Each row/column is associated with a CMOS tag that holds one bit of information. A '1' or '0' tag value specifies whether or not a row/column is selected in the final LUT after repair. This technique is referred to as the Tagged Replacement technique. The yields exhibited by this technique will be compared with the yields of the Repair Most technique to highlight the improvement in the efficiency of repair.

Fig. 4.10 shows the implementation of Tagged Replacement technique. For a LUT of size $2^N \times N$, c_{sp} spare columns and r_{sp} spare rows are allocated. In the simulation plots presented in this chapter, 100% redundancy was chosen (i.e. $c_{sp} = N$ and $r_{sp} = 2^N$). The implementation algorithm for the Tagged Replacement technique is outlined in Algorithm 1. Initially the tags for the original 2^N rows and N columns in LUT are set

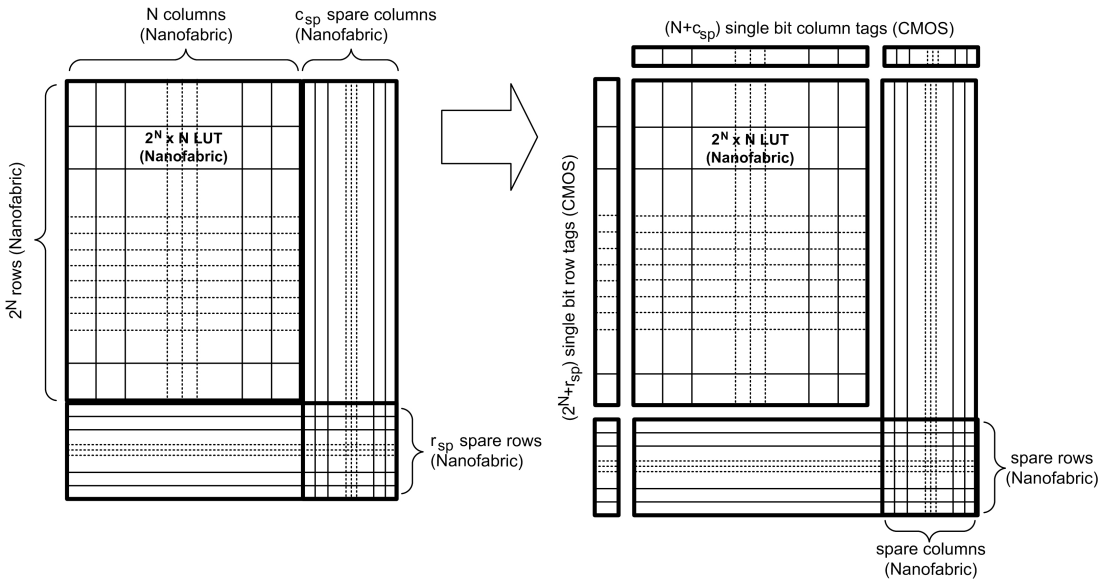


FIGURE 4.10: Tagged Replacement Technique: Implementation of a $2^N \times N$ LUT using 1-bit CMOS tags

to 1 and tags for the spare rows and spare columns (r_{sp} and c_{sp} respectively) are set to 0. Then starting with the original defective LUT, the columns are first scanned and subsequently replaced if less defective spare columns are found. Similarly, the procedure is repeated for the spare rows. Unlike the Repair Most technique, there is no need to specify the values of row threshold r_{th} and column threshold c_{th}. This technique uses a tagging method to tag rows and columns that are least defective. After the repair process, the tags will hold '1' for the least defective rows and columns and '0' for the excluded ones. The CMOS area overhead of this technique is $(2^N + r_{sp})$ single bit row tags and $(N + c_{sp})$ single bit column tags (Fig. 4.10).

Algorithm 1 Tagged Replacement Implementation Algorithm

1: Initialise LUT size, spare rows r_{sp}, spare columns c_{sp}
2: Initialise all LUT tages
Scan Column-wise
3: **for all** i $(< N)$ **do**
4: **for all** j $(< c_{sp})$ **do**
5: **if** $totalDefects(c_{sp}(j)) < totalDefects(column(i))$ **then**
6: $Tag(c_{sp}(j) = 1)$, $Tag(column(i)) = 0$
7: **end if**
8: **end for**
9: **end for**
Repeat scan Row-wise

A comparison between the efficiency of the Tagged Replacement technique with Repair Most is demonstrated in Fig. 4.11 assuming a $2^4 \times 4$ LUT, 100% redundancy and 0% DCCs. As can be observed, while the Repair Most technique could only handle defect rates less than 10%, the Tagged Replacement technique can target defect rates upto 15%. As noted in Table 4.1, the Repair Most can tolerate such a high defect density

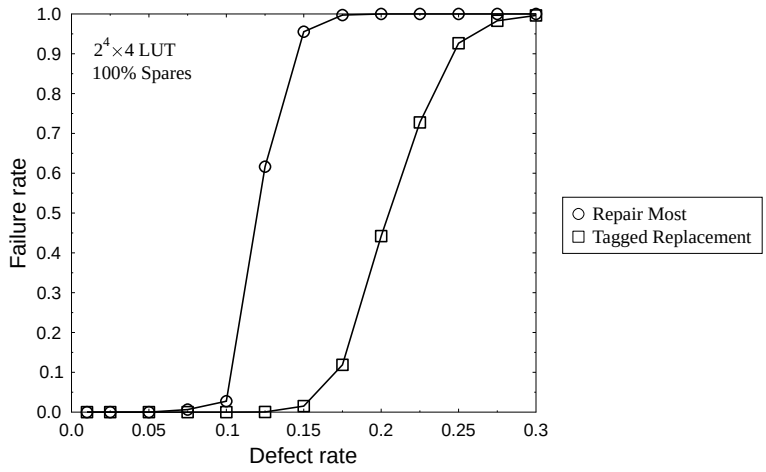


FIGURE 4.11: Comparison between Repair Most and Tagged Replacement techniques

(i.e. 15%) only if 175% of spare lines are allocated and c_{th} is set to 6.

Fig. 4.12 shows the plot of failure rate Vs defect rate using the Tagged Repair technique for different LUT sizes with percentage redundancy varying between 25% and 100%. As can be seen in Fig. 4.12(a), at 25% redundancy, the Tagged Repair technique exhibits higher defect tolerance in the case of smaller sized LUTs (such as $2^3 \times 3$ LUT) than larger LUTs (such as $2^6 \times 6$ LUT). This is because for a particular defect rate, the likelihood of finding a non-defective row/column to replace a defective one decreases since the number of defects in the LUT increase as the LUT size increases. Bigger LUTs require allocating more redundant resources to repair the increasing number of physical defects. Hence synthesis of larger circuits into smaller LUTs will result in improved defect tolerance for the targeted nano/CMOS architecture. Similarly, the failure rate Vs defect rate plots for 50% and 100% redundancy were simulated and shown in Fig. 4.12(b)-(c). The defect tolerance is significantly improved as the percentage of spares is increased. The Tagged Replacement technique exhibits an outstanding performance by tolerating upto 20% of defects with only 100% spares in the case of $2^3 \times 3$ LUT as compared to the 11% defect rate tolerated in the presence of 25% of spares. It can also be observed that the Tagged Replacement technique only requires 100% of spare lines to tolerate 10% of defects for $2^6 \times 6$ LUTs whereas the Repair Most technique requires 200% of spares to tolerate such an amount of defects (see Table 4.1). Table 4.2 includes the percentage of spares needed by the Tagged Replacement technique to achieve failure rates less than 5% for each LUT size. A comparison between Tables 4.1 and 4.2 demonstrates that the proposed Tagged Replacement technique requires much smaller redundancy than the Repair Most technique to achieve good repair rates. For instance, Repair Most requires 1100% of spares to repair $2^6 \times 6$ LUTs at 20% defect rate, however the Tagged Replacement technique requires only 300% of spares.

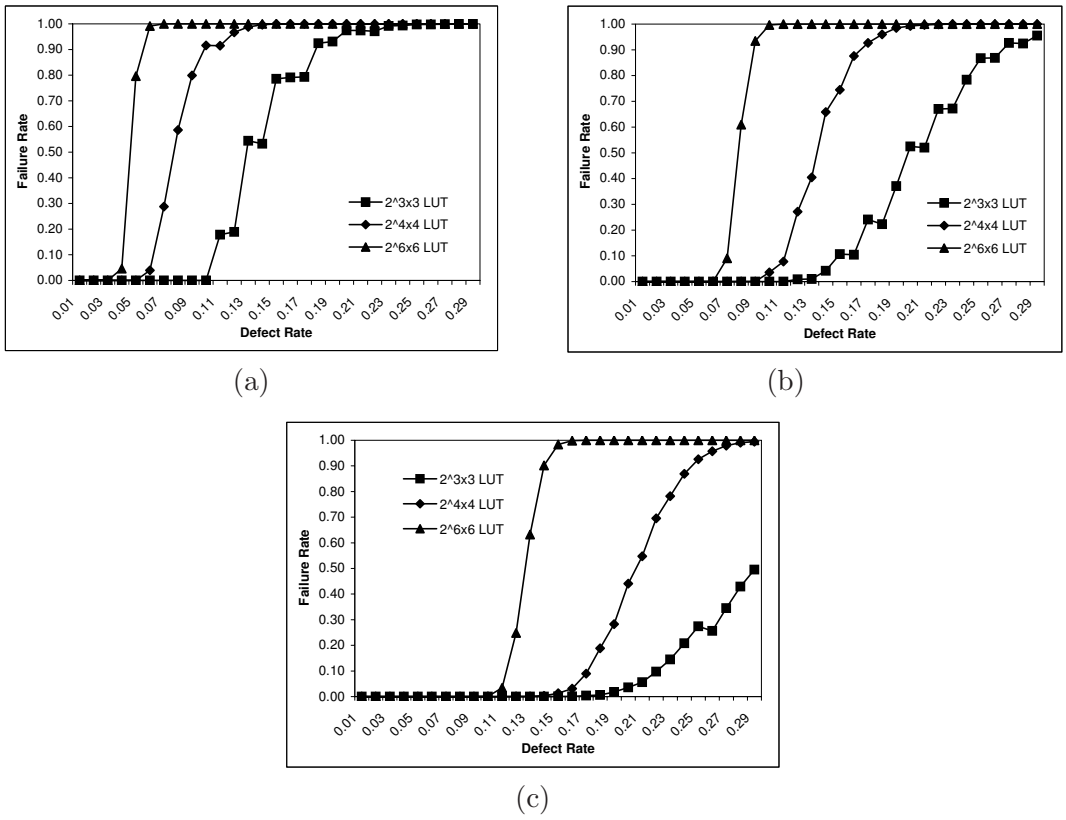


FIGURE 4.12: Plot of Failure rate Vs Defect rate using Tagged Repair technique for different LUT sizes with (a) 25% redundancy (b) 50% redundancy and (c) 100% redundancy in rows and columns

Defect rate	$2^3 \times 3$ LUT	$2^4 \times 4$ LUT	$2^6 \times 6$ LUT
1%	25%	25%	25%
5%	25%	25%	50%
10%	50%	50%	100%
15%	50%	100%	175%
20%	100%	150%	300%

TABLE 4.2: Tagged Replacement technique - Redundancy required to achieve failure rates less than 5%

4.3.1 Theoretical Analysis of Failure Rate (Tagged Replacement Technique)

The aim of the proposed Tagged Replacement technique is to identify a defect-free instance of a LUT of size $2^N \times N$ within a defective fabric given a certain amount of spare columns c_{sp} and spare rows r_{sp}. Hence, a theoretical estimation of the circuit failure rate of this technique reduces to the calculation of the probability of the non-existence of a subset of defect-free resources $(2^N \times N)$ within the partially-usable fabric $\big((2^N + r_{sp}) \times (N + c_{sp})\big)$.

$P_{col}(L)$ is the probability of a column of size $(r + L)$ being defective (i.e. in which the total number of defective bits exceeds the number of spare rows L). It is given by the following equation:

$$P_{col}(L) = \sum_{k=L+1}^{r+L} \binom{r+L}{k} P^k (1-P)^{r+L-k} \tag{4.9}$$

where r and c are the number of rows (2^N) and columns (N) of the LUT respectively, P is the defect rate of the fabric and $L = r_{sp}$ for the Tagged Replacement technique.

To successfully create an instance of the LUT on the fabric, columns should not only have less than L (where $L = r_{sp}$) defective bits but also there should be at least r defect-free bits in at least c columns that are aligned. This is illustrated in Fig. 4.13 where the size of the LUT is 4×3. Although the number of defective bits in column 3 are less than L, it was excluded because its defect-free bits are not aligned with the defect-free bits in the other columns. Hence, the probability of a column being excluded is equal to the sum of probabilities of being defective and not defective but not aligned with the other non-defective columns.

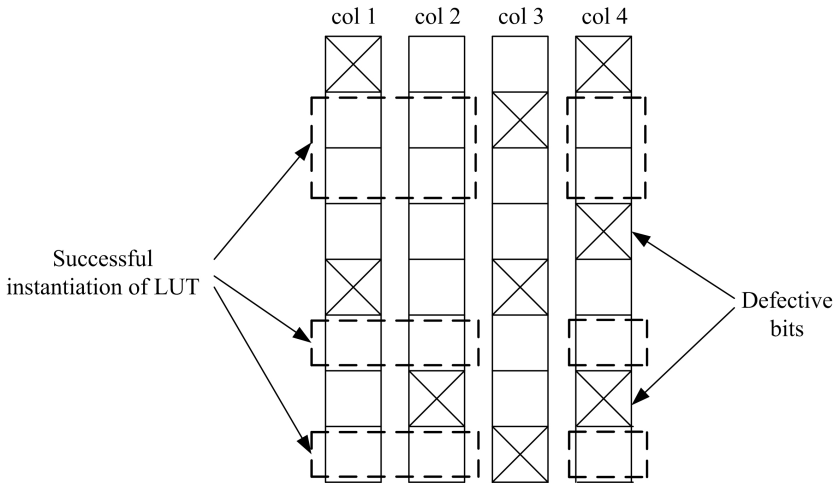


FIGURE 4.13: Tagged Replacement - Defect-free bits in columns should be aligned to ensure successful instantiation of LUTs

The total probability that two non-defective columns $col1$ with k defective bits $(0 \leq k \leq L)$ and $col2$ with k' defective bits $(0 \leq k' \leq L)$ are aligned where $k' \leq k$, is as follows:

$$\begin{aligned} P(col1, col2, L) &= \binom{r+L}{k} P^k (1-P)^{r+L-k} \\ &\times \left[\sum_{n=0}^{k'} \binom{k}{n} \binom{r+L-k}{k'-n} \times P^{k'} (1-P)^{r+L-k'}; k+k'-n \leq L \right] \end{aligned} \tag{4.10}$$

The probability that $col1$ and $col2$ are not aligned with each other is given by the

following equation:

$$\begin{aligned} P'(col1, col2, L) &= \binom{r+L}{k} P^k (1-P)^{r+L-k} \\ &\times \left[\sum_{n=0}^{k'} \binom{k}{n} \binom{r+L-k}{k'-n} \times P^{k'} (1-P)^{r+L-k'}; k+k'-n > L \right] \end{aligned} \tag{4.11}$$

Using equations 4.9, 4.10 and 4.11, the probability that n columns out of $(c + c_{sp})$ are not defective and aligned can be computed using the following equation:

$$\begin{aligned} P_{inst}(n, L) &= \sum_{a=1}^{n} \left[\left(\prod_{b=1}^{n} \sum_{k=0}^{L} \sum_{k'=0}^{L} P(a, b, L); a \neq b \right) \right. \\ &\times \left. \prod_{d=1}^{c+c_{sp}-a} \left(\sum_{k=0}^{L} \sum_{k'=0}^{L} P'(a, d, L) + P_{col}(L) \right) \right] \end{aligned} \tag{4.12}$$

Hence, the probability of successfully finding enough resources to create an instance of a given LUT using the Tagged Replacement technique where the number of spare rows is $L = r_{sp}$:

$$P_{succ} = \sum_{x=c}^{c+c_{sp}} \binom{c+c_{sp}}{x} P_{inst}(x, r_{sp}) \tag{4.13}$$

and therefore, the overall failure rate is:

$$P_{failure} = 1 - P_{succ} \tag{4.14}$$

Fig. 4.14 illustrates the failure rate obtained both theoretically, based on equation 4.14 and experimentally based on the simulations. As can be seen, the two graphs are almost identical validating the derived theoretical equations.

Next, the impact of injecting DCCs on the efficiency of the Tagged Replacement technique is examined. Theoretically, the existence of DCCs (P_{DCC}) in LUTs will significantly reduce the bit failure rate as outlined in equation 4.6. The total failure probability of Tagged Replacement technique in the presence of DCCs is obtained by replacing the value of bit failure rate P with P' in equations 4.9, 4.10 and 4.11. Fig. 4.15 shows the obtained failure rate before and after injecting 50% of Don't Cares into a $2^4 \times 4$ LUT. As can be seen from the graph, the existence of Don't Cares have an insignificant effect on the instantiation probability of this technique. DCCs slightly reduce the failure rate at defect rates higher than 17.5% to values that are considered prohibitively high.

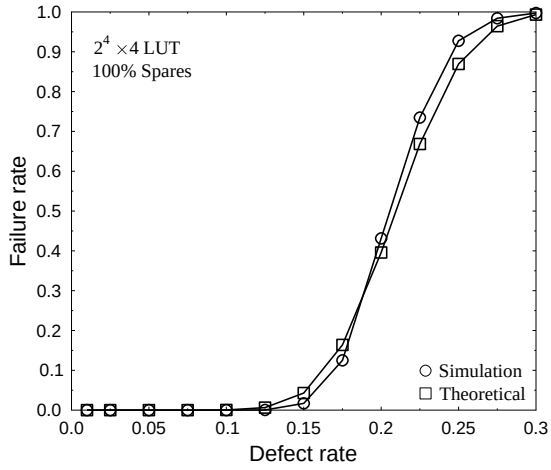


FIGURE 4.14: Tagged Replacement - Failure rate obtained both by theory and simulation for a $2^4 \times 4$ LUT and 100% redundancy

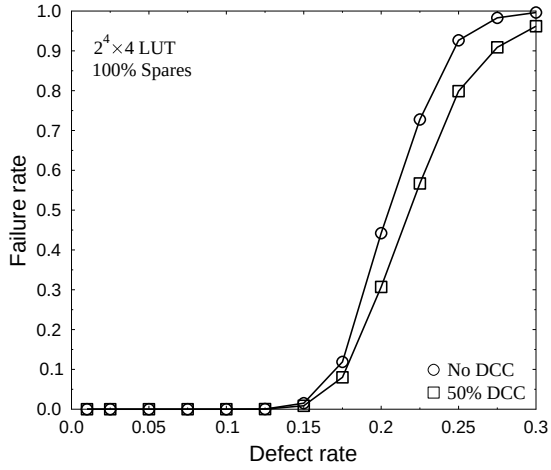


FIGURE 4.15: Tagged Replacement - Impact of Don't Cares on failure rate

4.4 Proposed Modified Tagged Replacement Technique

So far, the repair algorithms of both the Repair Most and the proposed Tagged Replacement techniques result in a significant defect-free portion of the LUT to be excluded as illustrated in the example shown in Fig. 4.16(a). For instance, although column 7 in Fig. 4.16(a) includes only two defective bits and the remaining bits are defect-free, the entire column is considered defective and hence excluded. Such an inefficient use of memory resources and defective line exclusion result in a significant decrease in the repair rate and an increase in the number of required spare rows and columns. Therefore, to enhance the line exclusion algorithms of both the Repair Most and the Tagged Replace-

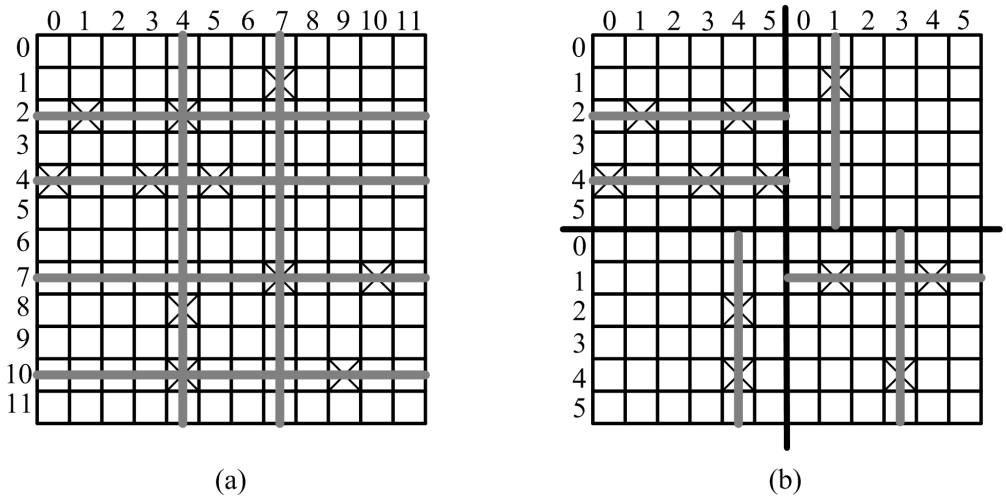


FIGURE 4.16: Illustration of the principle of the Modified Tagged Replacement technique

ment techniques and better allocate spare resources, a new repair algorithm based on the principle illustrated in Fig. 4.16(b) is proposed. The divided memory array, shown in Fig. 4.16(b), allows to reduce the amount of wasted resources (i.e. less defect-free bits are removed) and also increase the resolution of repair. For example, the number of excluded defect-free bits in column 7 are reduced from 10 in Fig. 4.16(a) to only 5 in Fig. 4.16(b). As will be shown in Figs. 4.19, 4.20 and 4.21, such a reconfiguration of defective memory arrays will enhance the repair rate and reduce the amount of spare lines needed to achieve acceptable yields.

The proposed Modified Tagged Replacement technique is a modified implementation of the previously proposed Tagged Replacement technique, shown in Fig. 4.10. This technique allows increasing the resolution of repair by dividing large LUTs of sizes $2^N \times N$ column-wise to 2^α sub-LUTs where $\alpha < N$. Fig. 4.17 illustrates the implementation of a $2^N \times N$ LUT using the Modified Tagged Replacement technique for $\alpha = 1$. Instead of replacing entire columns as in the Tagged Repair technique, the columns are split in two equal sections (because α is set to 1) before applying tagging and replacement, to optimise the use of spare units as compared to the Tagged Repair technique. To further enhance the instantiation probability of large LUTs onto defective fabrics, the resolution of repair needs to be enhanced to cope with the increased number of defects. This is achieved by choosing bigger values for the parameter α, as illustrated in Fig. 4.18 for three different values.

The implementation algorithm used for the Modified Tagged Repair technique is similar to the Tagged Repair technique but it has the following distinctive feature. In the algorithm of the Modified Tagged Repair technique, the column-wise scan is done in multiple stages depending on the value of α, unlike the Tagged Repair algorithm where column-wise scan is done in a single stage. The row-wise scan in the Modified Tagged Repair algorithm is done in a single stage like in the Tagged Repair algorithm. The

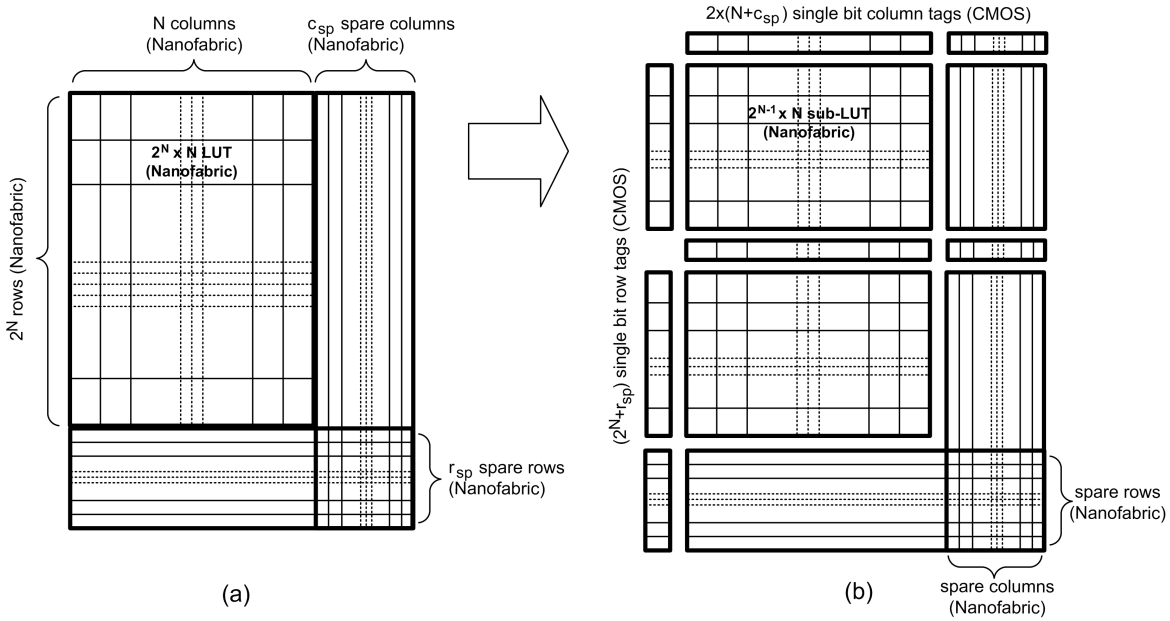


FIGURE 4.17: Modified Tagged Replacement Technique (a) Allocated $(2^N + r_{sp}) \times (N + c_{sp})$ nanofabric for the implementation of the $2^N \times N$ LUT (b) Implementation of the LUT using Modified Tagged Replacement technique in the case of $\alpha = 1$

reason for this is as follows: since a $2^N \times N$ LUT will always have an even number of elements 2^N in each column, hence it is easy to split each column by a constant of 2^α into sub-columns of size $2^{N-\alpha}$. A similar technique to split and tag rows cannot be used since, in a LUT of size $2^N \times N$, the number of rows N is very small as compared to the number of columns 2^N and dividing the LUT row-wise will only increase the complexity of the proposed technique without a significant improvement in the yield. The downside of using this technique is that it has a higher area overhead since the number of the required column tags is 2^α times that of the Tagged Replacement technique. This will cause an increase in CMOS area overhead of the tagging circuitry. When compared to the Tagged Replacement technique, this technique requires $2^\alpha \times (N + c_{sp})$ single bit column tags. Considering a single bit SRAM cell requires 6 transistors [Bellaouar and Elmasry, 1995], the overall CMOS area overhead in terms of transistor count can be calculated accordingly. The number of row tags will be equal to the Tagged Replacement technique.

Fig. 4.19 compares the repair capability of the Modified Tagged Replacement technique (for $\alpha = 1$) with the Tagged Replacement and Repair Most techniques for LUTs of size $2^4 \times 4$ with 100% redundancy ($c_{sp} = 4$ spare columns and $r_{sp} = 2^4$ spare rows). As can be seen, the Modified Tagged Replacement technique targets the highest defect rate followed by the Tagged Replacement and Repair Most respectively. For example, when the defect rate is 15%, the Modified Tagged Repair technique gives a failure rate of 0%, and the Tagged Replacement technique gives a failure rate of 2%, whereas, the Repair Most technique completely fails.

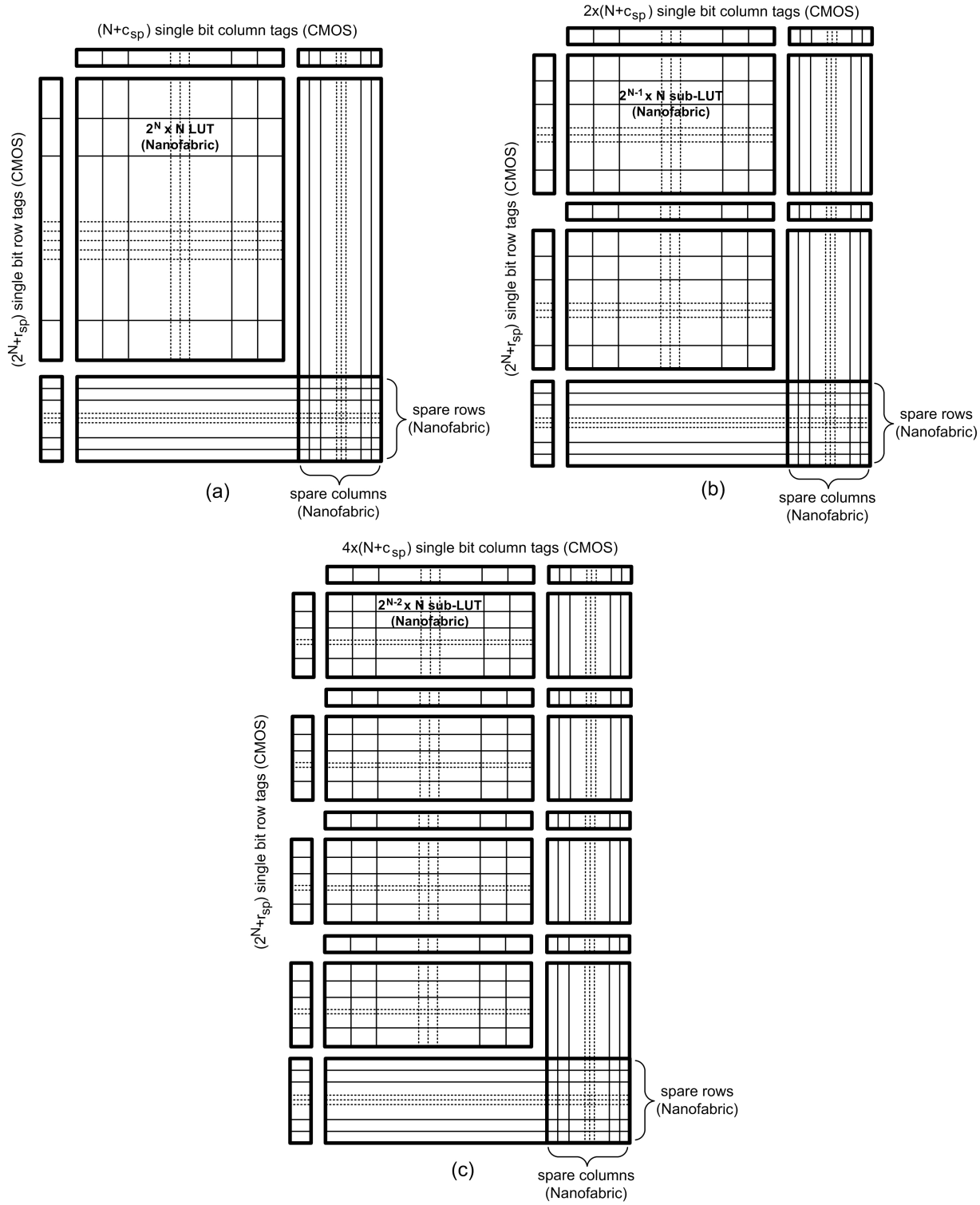


FIGURE 4.18: Modified Tagged Replacement Technique (a) $\alpha = 0$ (Tagged Replacement technique) (b) $\alpha = 1$ (c) $\alpha = 2$

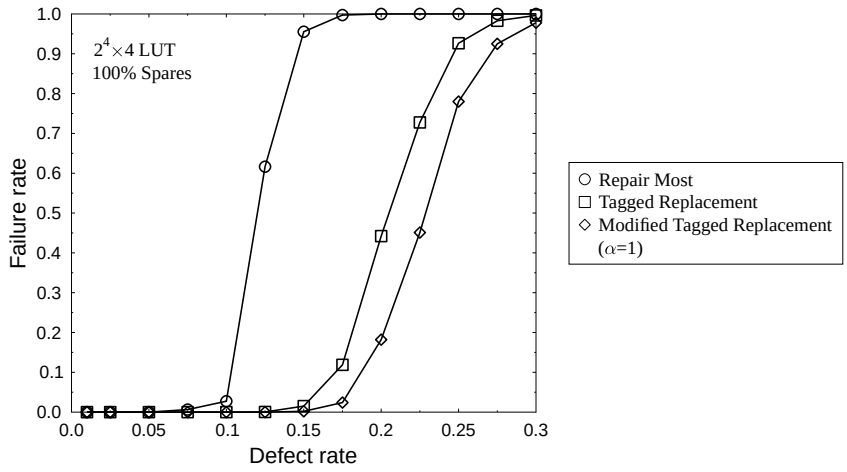


FIGURE 4.19: Failure rate comparative study for $2^4 \times 4$ LUT with 100% redundancy

Fig. 4.20 shows the plot of the Modified Tagged Repair technique ($\alpha = 1$) for different LUT sizes with varying redundancy. Similarly to the Tagged Replacement technique, by increasing the amount of redundant resources, the targeted defect rate of the Modified Tagged Repair technique is significantly increased. For instance, the targeted defect rate for $2^4 \times 4$ LUTs increases from 7% at 25% redundancy (Fig. 4.20(a)) to 11% at 50% redundancy (Fig. 4.20(b)) and to 17% at 100% redundancy (Fig. 4.20(c)). Furthermore, by comparing Figs. 4.12 and 4.20, it can be observed that the Modified Tagged Replacement technique further improves the defect tolerance of the LUT-based architecture as compared to the Tagged Replacement techniques for all LUT sizes and redundancy overheads. As an example, the results of the Tagged Repair technique for $2^3 \times 3$ LUTs and 100% redundancy (Fig. 4.12(c)) are compared with the Modified Tagged Repair technique (Fig. 4.20(c)). While the Tagged Repair technique can achieve 0% failure rate at defect rates of upto 17%, the Modified Tagged Repair technique can target defect rate upto 20%. This improvement in repair capability ($1 - P_{failure}$) is due to the more optimised usage (by splitting the columns in two before applying repair) of the redundant spare units.

The improvement in yield that can be achieved by the Modified Tagged Replacement technique by dividing LUTs into smaller sub-LUTs is demonstrated in Fig. 4.21. The instantiation probability is significantly enhanced when dividing the $2^4 \times 4$ LUTs into smaller 2^2 sub-LUTs (i.e. when $\alpha = 2$). This is illustrated by a failure rate of 0% for defect rates upto 20% which clearly outperforms the results obtained by the Modified Tagged Replacement technique when $\alpha = 1$ and the Tagged Replacement technique ($\alpha = 0$).

According to Fig. 4.20(c), although 100% redundancy was allocated, the Modified Tagged Replacement technique can only tolerate upto 12% in the case of $2^6 \times 6$ LUTs. Imple-

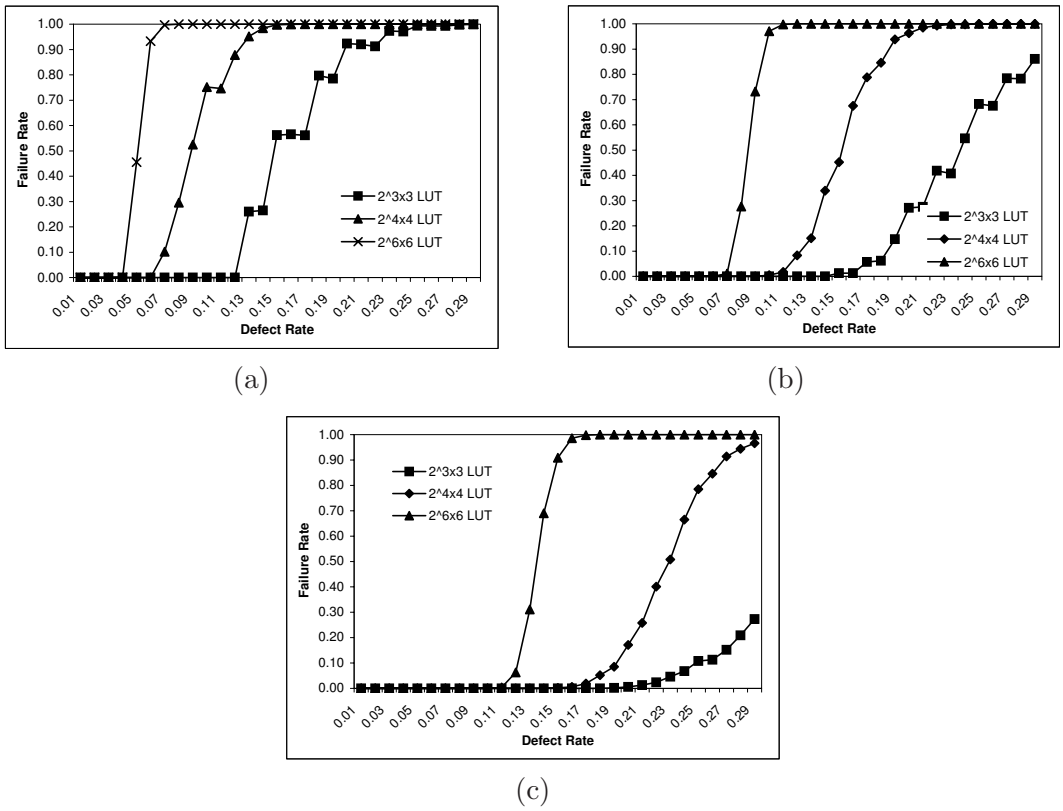


FIGURE 4.20: Plot of Failure rate Vs Defect rate using Modified Tagged Repair technique ($\alpha = 1$) for different LUT sizes with (a) 25% redundancy (b) 50% redundancy and (c) 100% redundancy in rows and columns

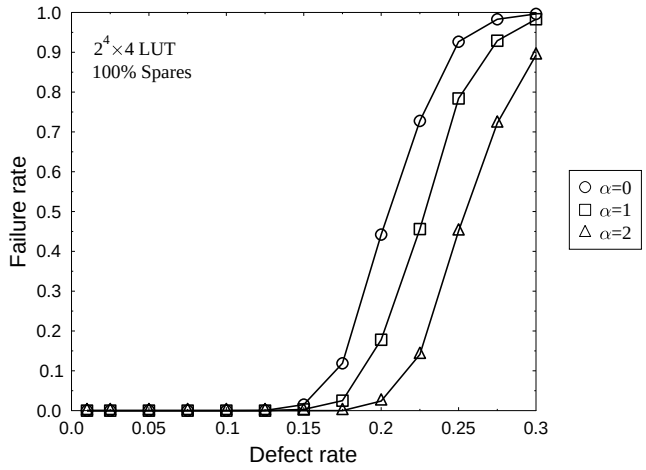


FIGURE 4.21: Modified Tagged Replacement - Failure rate vs. Defect rate for different values of α

menting large LUTs using the Modified Tagged Replacement technique at a reduced area overhead requires increasing the value of the parameter α so that the LUTs are divided into much smaller sub-LUTs. Table 4.3 includes the failure rates exhibited by

the Modified Tagged Replacement technique for large LUT sizes (upto $2^{10} \times 10$) and different α values. This technique achieves higher instantiation probabilities than the previously proposed repair techniques. For instance, for $2^8 \times 8$ LUTs, the targeted defect rate is increased from 7.5% for $\alpha = 1$ to 12.5% for $\alpha = 5$. Therefore, it can be observed that a successful instantiation of a large LUTs can be achieved without the need to allocate a large amount of redundancy to repair high defect rates. Instead, this can be obtained by dividing the LUTs into much smaller sub-LUTs i.e. bigger α values. For instance, in Table 4.2, the Tagged Replacement technique requires 175% redundancy in order to tolerate 15% defect rate for a $2^6 \times 6$ LUT, whereas in Table 4.3, this can be achieved with only 100% redundancy and $\alpha = 4$.

$2^6 \times 6$ LUT							
α	5%	7.5%	10%	12.5%	15%	17.5%	20%
1	0.000	0.000	0.000	0.174	0.910	1.000	1.000
2	0.000	0.000	0.000	0.015	0.584	0.988	1.000
3	0.000	0.000	0.000	0.000	0.094	0.728	0.991
4	0.000	0.000	0.000	0.000	0.000	0.004	0.402
$2^8 \times 8$ LUT							
α	5%	7.5%	10%	12.5%	15%	17.5%	20%
1	0.000	0.000	0.943	1.000	1.000	1.000	1.000
2	0.000	0.000	0.635	1.000	1.000	1.000	1.000
3	0.000	0.000	0.072	1.000	1.000	1.000	1.000
4	0.000	0.000	0.000	0.951	1.000	1.000	1.000
5	0.000	0.000	0.000	0.021	0.999	1.000	1.000
$2^{10} \times 10$ LUT							
α	5%	7.5%	10%	12.5%	15%	17.5%	20%
1	0.000	0.151	1.000	1.000	1.000	1.000	1.000
2	0.000	0.007	1.000	1.000	1.000	1.000	1.000
3	0.000	0.000	1.000	1.000	1.000	1.000	1.000
4	0.000	0.000	1.000	1.000	1.000	1.000	1.000
5	0.000	0.000	1.000	1.000	1.000	1.000	1.000
6	0.000	0.000	0.147	1.000	1.000	1.000	1.000
7	0.000	0.000	0.000	1.000	1.000	1.000	1.000

TABLE 4.3: Modified Tagged Replacement technique- Failure rate obtained for different LUT sizes and different values of α

4.4.1 Theoretical Analysis of Failure Rate (Modified Tagged Replacement Technique)

In the Modified Tagged Replacement technique, a successful instantiation of a LUT on the defective fabric is achieved by successfully instantiating each of the sub-LUTs of the divided LUT given the amount of allocated spare columns c_{sp} and spare rows r_{sp}. For example, for $\alpha = 1$, a successful instantiation is obtained when each half of the

LUT is successfully implemented. Equation 4.15 represents the total probability P_{succ} of instantiating a $2^N \times N$ LUT with r_{sp} spare rows and c_{sp} spare columns for $\alpha = 1$ where P_{inst} is given by Equation 4.12. The variable i represents the number of spare rows used by the Modified Tagged Replacement technique to repair the defective rows in the first half of the LUT, whereas the rest of spare rows (i.e. $r_{sp} - i$) are used in the repair of the second half of the LUT. Hence P_{succ} can be computed as follows:

$$P_{succ} = \sum_{i=0}^{r_{sp}} \left[\left(\sum_{x=c}^{c+c_{sp}} \binom{c+c_{sp}}{x} P_{inst}(x,i) \right) \times \left(\sum_{x'=c}^{c+c_{sp}} \binom{c+c_{sp}}{x'} P_{inst}(x', r_{sp}-i) \right) \right] \quad (4.15)$$

Next, the effect of injecting Don't Cares into the LUTs on the efficiency of the proposed Modified Tagged Replacement technique is examined. Fig. 4.22 shows the failure rate exhibited by the Modified Tagged Repair technique (for $\alpha = 1$) when 50% of the entries in the LUT are assumed as DCCs. The presence of DCCs can improve the tolerance against physical defects. At 17.5% defect rate for instance, 0% DCCs result in 3% failure rate, while 50% DCCs result in 0% failure rate. Hence, the presence of DCCs can be used to either enhance the targeted defect rate or to reduce the cost of repair.

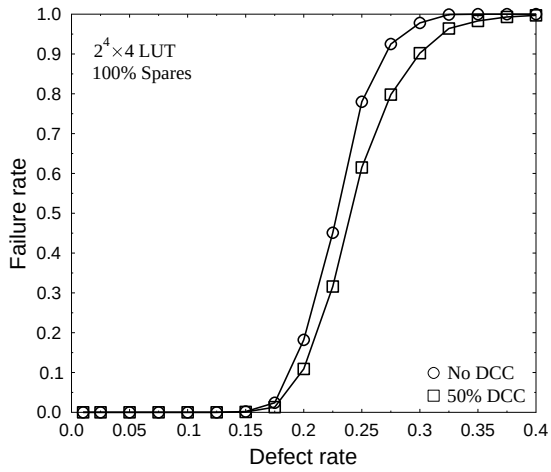


FIGURE 4.22: Modified Tagged Replacement ($\alpha = 1$) - Effect of Don't Cares on failure rate

4.5 Yield Analysis of Large Circuits

To assess the repair capability of the proposed techniques for larger circuits, an analysis was performed on the ISCAS'85 benchmark circuits using the Modified Tagged Repair technique. The ISCAS'85 benchmark circuits were first synthesized into smaller LUTs

	$2^2 \times 2$ LUTs	$2^3 \times 3$ LUTs	$2^4 \times 4$ LUTs	$2^5 \times 5$ LUTs	$2^6 \times 6$ LUTs
c432	3	1	2	1	5
c499	0	0	2	2	8
c880	1	4	2	4	5
c1335	5	0	4	5	5
c1908	2	2	2	4	10
c2670	2	4	5	3	9
c3540	6	2	10	13	22
c5315	8	8	12	8	25
c6288	20	4	14	9	48
c7552	6	18	16	20	30

TABLE 4.4: ISCAS'85 benchmark circuits synthesised into smaller $2^N \times N$ LUTs

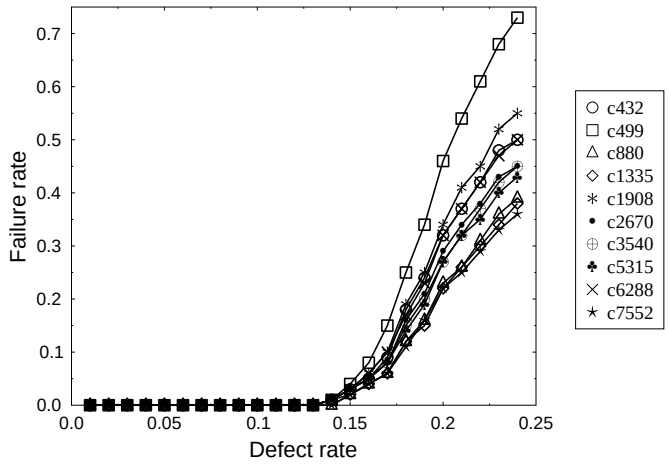


FIGURE 4.23: Failure probability of synthesised ISCAS'85 benchmark circuits using Modified Tagged Replacement technique ($\alpha = 1$)

sizes (between $2^2 \times 2$ to $2^6 \times 6$). Table 4.4 shows the size and the number of the LUTs of each of the ISCAS'85 circuits. Because the size of the LUTs is less than or equal to $2^6 \times 6$, the parameter α of the Modified Tagged Repair technique was set to 1 to obtain acceptable yield values (Fig. 4.20). The failure rate results obtained from the simulations are shown in Fig. 4.23. However, because the majority of the synthesised circuits contain a higher proportion of $2^5 \times 5$ and $2^6 \times 6$ LUTs (Table 4.4), the targeted defect rate of the various benchmark circuits is around 13%-14% with little variation as shown in Fig. 4.23. This can be addressed by further synthesising the circuits into LUTs of smaller sizes.

4.6 Estimation of Repair Cost

The area overheads incurred by the new repair techniques are estimated. This includes both redundant nanodevice resources and CMOS overhead. As explained in Sections 4.2, 4.3 and 4.4, the proposed Tagged Repair and Modified Tagged Repair techniques use considerably less redundancy to tolerate higher defect rates as compared to the Repair Most technique. Table 4.5 compares the repair cost of the proposed techniques and the Repair Most technique in terms of targeted defect rate. As an example, in case of $2^3 \times 3$ LUTs, the Modified Tagged Replacement technique can target upto 10% defect rate with only 25% spares, while Repair Most is not capable of targeting upto 10% of defects with even 100% spares. The targeted defect rate values and the amount of spare units given in this table have been rounded off to the nearest 0.5%.

%Spares	LUT size	Repair Most	Tagged Replacement	Modified Tagged Replacement ($\alpha = 1$)
25%	$2^3 \times 3$	5.0%	7.5%	10.0%
	$2^4 \times 4$	1.0%	5.0%	5.0%
	$2^6 \times 6$	1.0%	1.0%	2.5%
50%	$2^3 \times 3$	7.5%	12.5%	12.5%
	$2^4 \times 4$	5.0%	7.5%	10.0%
	$2^6 \times 6$	1.0%	5.0%	5.0%
75%	$2^3 \times 3$	7.5%	15%	17.5%
	$2^4 \times 4$	5%	12.5%	12.5%
	$2^6 \times 6$	5%	7.5%	7.5%
100%	$2^3 \times 3$	7.5%	17.5%	20.0%
	$2^4 \times 4$	7.5%	12.5%	15.0%
	$2^6 \times 6$	5.0%	10.0%	10.0%

TABLE 4.5: Comparative repair cost of the proposed techniques with the Repair Most technique in terms of targeted defect rate

The CMOS area overhead of a $2^N(rows) \times N(columns)$ LUT implemented using the Tagged Repair technique (illustrated in Fig. 4.10) with r_{sp} spare rows and c_{sp} spare columns is $2^N + r_{sp}$ single bit row tags and $N + c_{sp}$ single bit column tags. When compared to the Tagged Repair technique, the Modified Tagged Repair technique requires a total of $2^\alpha \times (N + c_{sp})$ single bit CMOS column tags. Here the multiplication factor 2^α accounts for the splitting of the LUT column-wise into equally-sized sub-LUTs to obtain better repair capability. The number of row tags will be equal to the Tagged Repair technique. For example, in case of a $2^4 \times 4$ LUT with 100% redundancy, the CMOS area overhead for Tagged Repair technique will be a total of $2 \times (2^4 + 4) = 40$ single bit tags. However, for the Modified Tagged Repair technique, the CMOS area overhead will be $2 \times 2^4 + 2^\alpha \times (4 + 4) = 32 + 8 \times 2^\alpha$ tags. In case $\alpha = 1$, this will give a total of 48 tags for a single LUT. Considering a single bit SRAM cell requires 6 transistors [Bellaouar

and Elmasry, 1995], the overall CMOS area overhead in terms of transistor count can be calculated as follows:

$$N_{trans} = 6 \times \Big((2^N + r_{sp}) + 2^{\alpha}(N + c_{sp}) \Big) \tag{4.16}$$

In case of a $2^4 \times 4$ LUT with 100% redundancy, the Modified Tagged Repair technique with $\alpha = 1$ will incur $6 \times (48 - 40) = 48$ extra transistors/LUT as compared to the Tagged Repair technique.

4.7 Effect of Defect Distribution Model

So far only uniformly distributed defects were assumed where the probability of a bit being defective is the same throughout the fabric as illustrated in Fig. 4.24(a). However, a more effective way to deal with large percentages of defects would be to first identify and then target the defect model inherent in a particular nanotechnology. Therefore, other defect distributions that are expected to occur in nanoscale fabrics should be addressed. Clustered defects is another model that is projected in nanoscale fabrics as outlined in [Biswas et al., Tahoori, 2005b]. According to [Tahoori, 2005b], defects in practice tend to cluster together and their distribution is not uniform. In such case, the defect probabilities in the regions near defect clusters are higher than other regions and as the distance between a given switch and the centre of the cluster increases, its defect probability decreases. Hence, the defect probability is inversely proportional to the distance from the existing defect clusters. The authors suggested equation 4.17 to calculate the defect probability P_i at location i where N_d is the number of defects, d_{ij} is the distance between switch i and j and α is the clustering parameter.

$$P_i \propto \sum_{j=1}^{N_d} \left(\frac{1}{d_{ij}}\right)^{\alpha} \tag{4.17}$$

In [Biswas et al.], clustered defects are correlated using Gaussian distribution whose probability density function is defined as:

$$P(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.18}$$

which has a mean μ and variance σ^2. In the simulations carried out in this section, Gaussian distribution function was used to emulate clustered defect distribution where $\mu = 0$ and $\sigma = 1.5$, as shown in Fig. 4.24(b).

Another defect distribution model that is projected in nanoscale fabrics is Row/Column defect distribution model which is caused by broken nanowires (Chapter 1, Section 1.2.2).

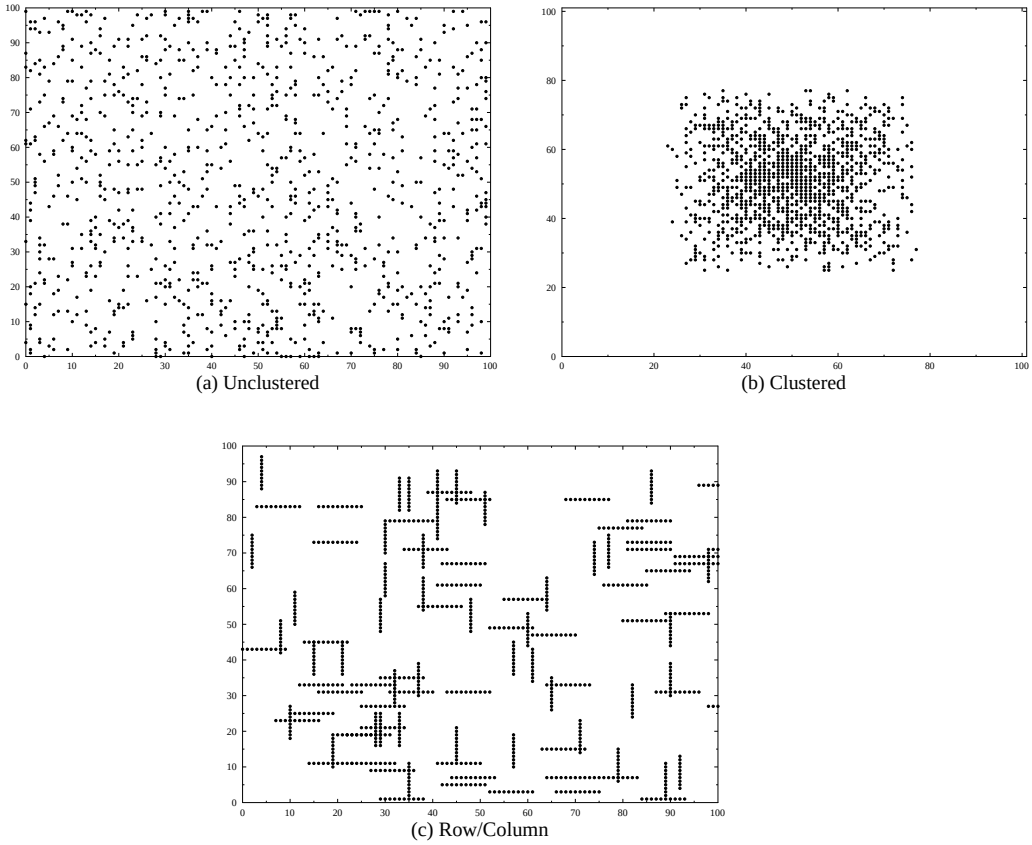


FIGURE 4.24: Defect distribution in nanofabrics: (a) random defect distribution (b) clustered defect distribution using Gaussian distribution function (c) row/column defect distribution. Defect rate is assumed to be 10% in a 100×100 crossbar

Fig. 4.24(c) illustrates this defect model where defects are concentrated entirely along particular rows or columns.

Fig. 4.25 demonstrates the impact of defect distribution on the repair capability of the Modified Tagged Repair technique. With Gaussian and Row/Column defect distributions, the proposed technique can target defect rates upto 22% and 27% respectively. An interesting observation from Fig. 4.25 is the rate of change of failure rate. While the failure rate for random defects shows a smooth curve, there is an abrupt increase in failure rate for the Gaussian defect distribution. This abrupt change in failure rate (for defect rates higher than 22%) can be attributed to the large number of defects that lie within the cluster and requires more redundant resources than the allocated ones to be successfully tolerated. The failure rate for Row/Column defect distribution increases much more slowly. It can be concluded that clustered and Row/Column defects are easier to repair than the uniformly distributed defects because these distributions require less redundant lines to repair them.

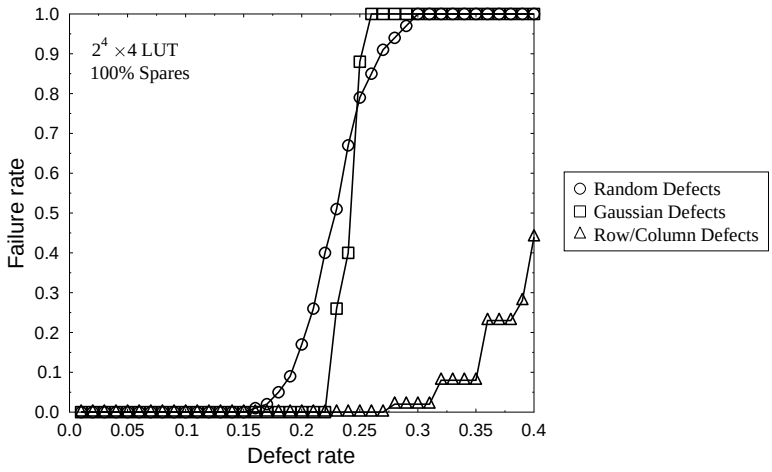


FIGURE 4.25: Effect of defect distribution on defect tolerance for a $2^4 \times 4$ LUT with 100% redundancy using Modified Tagged Repair technique ($\alpha = 1$) under (a) random defect distribution (b) Gaussian/clustered defect distribution (c) Row/Column defect distribution

4.8 Conclusion

In this chapter, two new and effective Tagged Repair techniques were proposed for emerging nano/CMOS computational architecture implementing logic circuits as LUTs. It was shown that these redundancy-based repair techniques, while simple to implement, also have low redundancy requirements and are able to provide higher levels of defect tolerance as compared to the previously proposed Repair Most technique. While Repair Most could handle defect rates of only upto 10% in the case of $2^3 \times 3$ LUTs, the proposed Tagged Replacement techniques exhibited much higher efficiency and were shown to handle defect rates higher than 20% for the same LUT size and upto 14% defect rate for synthesised ISCAS'85 benchmark circuits. It was also shown that the proposed repair techniques perform best when the circuits are synthesized into smaller sized LUTs. The repair capability of the proposed techniques has been achieved at a reasonable area overhead for the tagging mechanism. The effect of DCCs on the repair rate was also examined and it was demonstrated that DCCs can be used to enhance the efficiency and/or reduce the cost of repair. Finally, the reliability of the proposed techniques in repairing clustered and row/column defect distributions was investigated. It was shown that the proposed repair techniques can tolerate higher clustered and row/column defect rates than the uniformly distributed defects.

Chapter 5

Defect Tolerance Design Flow for Nanoscale PLA Architecture

In Chapter 4, the proposed repair techniques address the high defect rates in hybrid nano/CMOS computational architecture implementing logic circuits as LUTs. This chapter presents two new defect tolerance (DT) techniques incorporated in a design flow capable of tolerating the physical defects in crossbar architecture implementing logic circuits as PLAs. These two techniques tolerate nanowire and junction defects described in Chapter 1, Section 1.2.2. Defect tolerance is achieved at a reduced post-fabrication test overhead by limiting defect diagnosis to the nanowire level rather than the crosspoint level used in previously reported DT techniques [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004, DeHon and Naeimi, 2005]. Junction defects are tolerated without locating them by adding redundancy in both AND and OR planes of the PLA. After running extensive tests over a suite of benchmark PLAs, the proposed design flow proved its capability of achieving the targeted yields regardless of the size, geometry and density of active junctions of the benchmark PLAs.

The chapter is organised as follows: Section 5.1 introduces the problem addressed in this chapter and outlines the defect model adopted in the simulations. Section 5.2 presents the principle of the relevant techniques that were recently proposed. The limitations of these techniques are discussed in Section 5.3 and an overview of the proposed DT design flow is presented. Section 5.4 gives a motivational example that highlights the difference between the proposed design flow and the previously reported techniques. The first technique of the proposed design flow targets nanowire defects and it is outlined in Section 5.5. Section 5.6 lists the recently reported PLA implementations of logic circuits that will be used in the evaluation of the second technique of the proposed design flow. This technique targets junction defects and it is outlined in Section 5.7. The impact of nanowire and junction defect rates on the manufacturing yield is discussed in Section 5.8. The chapter is concluded in Section 5.9.

All the work presented in this chapter is the sole contribution of the author.

5.1 Introduction

Amongst the different circuit implementations proposed for the nanoscale crossbar architecture is the programmable logic array based implementation, outlined in Chapter 1, Section 1.1.2. This approach that make use of the semiconducting properties of nanoscale devices for circuit implementation can be set to evaluate any logical formula expressed as a combination of AND and OR operations. In this chapter, a defect tolerance design flow addressing the projected high defect rates in nanoscale PLAs is developed. Various defect tolerance techniques targeting nanoscale PLAs have recently been proposed [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004, Brown and Blanton, 2007, Wang and Chakrabarty, 2007]. Because these techniques are based on defect avoidance methods which are in turn based on defect diagnosis and mapping at the lowest level of granularity i.e. junction level, adopting them in the fabrication process will cause a major scalability problem given the high density of nanoscale fabrics (10^{12} devices/cm^2 [Tahoori, 2006a, Ziegler and Stan, 2003, Tahoori, 2005b]). In this chapter, defect tolerance is addressed at the nanowire level rather than the junction level and hence reducing the test and diagnosis overhead. The proposed design flow tolerates the physical defects by adding redundancy in both AND and OR planes of the PLA. The central question addressed in this chapter is: given a defect rate and a certain PLA size (i.e. the number of input, product and output wires), how much redundancy should be allocated in both AND and OR planes to achieve a certain predefined yield. Determining the required redundancy to achieve this yield is a probabilistic problem rather than a deterministic problem whose answer is presented in this chapter.

The main type of defects taken into consideration are the physical defects that are introduced during the manufacturing phase rather than during operational phase. This includes stuck-open and stuck-short junction defects and broken nanowire defects (Chapter 1, Section 1.2.2). According to [DeHon and Naeimi, 2005, Naeimi and DeHon, 2004, Chen et al., 2003a, Rao et al., 2007, Mishra and Goldstein, 2003, Joshi and Al-Assadi, 2007], closed-junction defects are expected to be much less likely than open-junction defects which are considered to be the dominant defects. Fig. 5.1 is an illustration of the crossbar defect model considered in this chapter. In [Naeimi and DeHon, 2004, DeHon and Naeimi, 2005, Mishra and Goldstein, 2003, Sun and Zhang, 2007], closed-junction defects are treated as wire defects rather than junction defects because a closed-junction will result in the entire horizontal and vertical nanowires being unusable. In [Huang et al., 2004], the proposed defect-aware graph-based technique uses closed-junction defects for circuit implementation in certain occasions if there is an edge between the corresponding horizontal and vertical nodes in the circuit's graph. The defect tolerance technique proposed in this chapter is not based on junction defect mapping, and be-

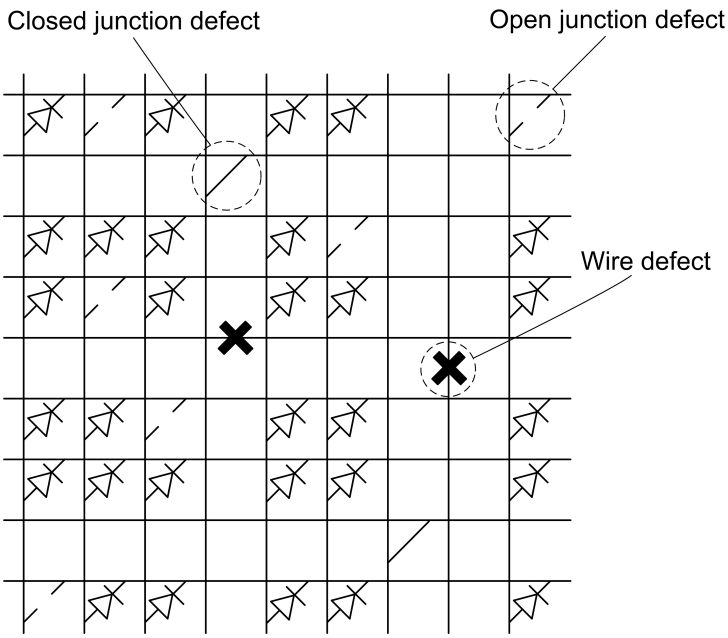


FIGURE 5.1: An illustration of the possible physical defects in nanoscale PLAs

cause of the low occurrence of closed-junction defects, both row and column making the closed-junction defect are considered as defective wires and hence should be excluded from the crossbar, as illustrated in Fig. 5.1.

For the analysis that follows, it is assumed that all physical defects are uniformly distributed across the nanoscale fabric i.e. without any spatial correlation among defect locations. This assumption is consistent for both open-junction defects and defective wires resulting from both broken nanowires and closed-junction defects [Naeimi and DeHon, 2004, DeHon and Naeimi, 2005].

To measure the yields obtained by the proposed two DT techniques, a set of benchmark PLAs with different sizes, geometries and active junction densities are used in the simulations. The simulation procedure used to calculate yield is outlined below. The proposed techniques are applied to the $N \times (I + O)$ PLAs where N, I and O are the number of product, input and output wires respectively, and a certain junction/nanowire defect rate d is randomly injected into the resulting PLAs.

1. Set the number of iterations to be performed, I, to 5000 and the number of successful simulations, K, to 0.

2. Generate a defect-tolerant $N' \times (I' + O')$ PLA resulting from applying the DT technique to the $N \times (I + O)$ PLA given the defect rate d ($N' \geq N$, $I' \geq I$ and $O' \geq O$).

3. Randomly inject junction/nanowire defects into the $N' \times (I' + O')$ PLA.

4. If the outputs of the defect-tolerant $N' \times (I' + O')$ PLA and the outputs of the original $N \times (I + O)$ PLA are logically equivalent, increment K by 1.

5. Decrement I by 1 and if I is not 0 goto step 3.

6. $Yield = K/5000$

5.2 Previous Work

The literature review provided in Chapter 2 provides an overview of the previously proposed architectures and techniques for different nanoscale technologies. In this section, a summary of the recently reported defect tolerance techniques targeting nanoscale PLA architecture is given.

In [Hogg and Snider, 2007], the authors examined the ability to map adder circuits onto crossbar architecture in the presence of open-junction defects. The proposed technique in [Hogg and Snider, 2007] is based on an allocation algorithm that uses graphs with annotated edges and nodes to represent both the circuit to be implemented and the physical crossbars. Graphs for the crossbars are constructed by representing a wire with a node and the junction is represented by an edge between the two nodes representing the wires that define the junction. Defective crossbars have edges only for usable junctions whereas the defective junctions are represented by removing their edges from the graph. Similarly, circuit graphs are constructed by representing the wires and junctions in the circuit with nodes and edges respectively. Allocation is accomplished by first creating the graphs representing the circuit to be mapped and the crossbars and then a monomorphism between the circuit graph and the crossbar graph is searched. Graph monomorphism is defined as the embedding of a small graph into a larger one by specifying the correspondence between the nodes of the small graph and a subset of nodes in the larger graph so that the small graph forms a subgraph of the larger one.

Similarly in [Al-Yamani et al., 2007] a greedy algorithm is proposed that is based on bipartite graph representation of the crossbar. The graph's construction is as outlined in [Hogg and Snider, 2007]. The problem addressed in [Al-Yamani et al., 2007] reduces to the maximum balanced bipartite subgraph problem. The solution to this problem aims to find the largest possible defect-free $k \times k$ subgraph out of the defective $n \times n$ graph representing the crossbar where $k \leq n$. In [Naeimi and DeHon, 2004], another graph-based defect tolerance technique was proposed to tackle crosspoint defects in nanoscale PLAs. The allocation algorithm searches for a match between the OR terms that are compatible with the nanowire's defect pattern. A match is found when the OR term's inputs are a subset of the nanowire's nondefective junctions. The graph is constructed by determining which OR terms are compatible with each nanowire's defect pattern. The nodes on the right side of the graph are the OR terms and the nodes on the left

are the nanowires. An edge between an OR term and a nanowire indicates that the OR term is compatible with the nanowire's defect pattern. A successful instantiation is achieved when all OR terms are assigned to the nanowires.

In [Brown and Blanton, 2007] and [Wang and Chakrabarty, 2007], built-in-self-test techniques for nanofabrics implementing logic circuits as PLAs were proposed. The architecture adopted in these two papers is similar to Field-Programmable Gate Arrays where the nanofabric is decomposed into nanoblocks that can be programmed after fabrication to implement logic functions as PLAs. The nanoblocks are interconnected through switchblocks that are used to route signals between adjacent nanoblocks. The proposed built-in-self-test procedures in [Brown and Blanton, 2007] and [Wang and Chakrabarty, 2007] configure the nanoblocks into test pattern generators, blocks under test and output response analysers. A set of configurations is used to obtain 100% fault coverage for junction defects such as stuck-open and stuck-short faults. The circuit to be implemented is then mapped onto the nanofabric by avoiding the located defective nanoblocks.

Similarly in [Garcia and Orailoglu, 2008], a fault tolerance scheme for nanoscale PLAs which is based on checkpointing is presented. Checkpoints are safe points during the operation of a system. Upon detection of an error, the system rolls back to the most recent checkpoint and continue execution. To generate a new checkpoint, all PLA blocks in the system have to be thoroughly tested. The system is divided into a group of N PLA blocks, two of which are designated as surrogates. During operation, while one of the $(N-2)$ non-surrogate blocks is in a testing state, one of the surrogates will be configured to implement its normal functionality. To diagnose crosspoint faults, test vectors are applied to the block under test .

5.3 Proposed DT Design Flow

In all the previously reported techniques [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004, DeHon and Naeimi, 2005], briefly outlined in the previous section, the manufacturing yield is not only determined by the efficiency of the defect tolerance technique but also by the accuracy of the techniques identifying the defect maps. In [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004] the graphs representing the crossbars are constructed based on the defect maps, and any false negative in the map will certainly result in an erroneous output and hence a decline in yield. Therefore to obtain the required yield, chips need to be thoroughly tested and thus causes a delay overhead in the manufacturing phase. Furthermore defect mapping must be applied on a per-chip basis where each PLA in the chip will be diagnosed on its own and will have its unique defect pattern as illustrated in Fig. 5.2. Given the significant device density offered by nanoscale devices (10^{12} devices/cm^2 [Tahoori, 2006a, Ziegler and Stan, 2003, Tahoori, 2005b]), a modest die will contain millions of

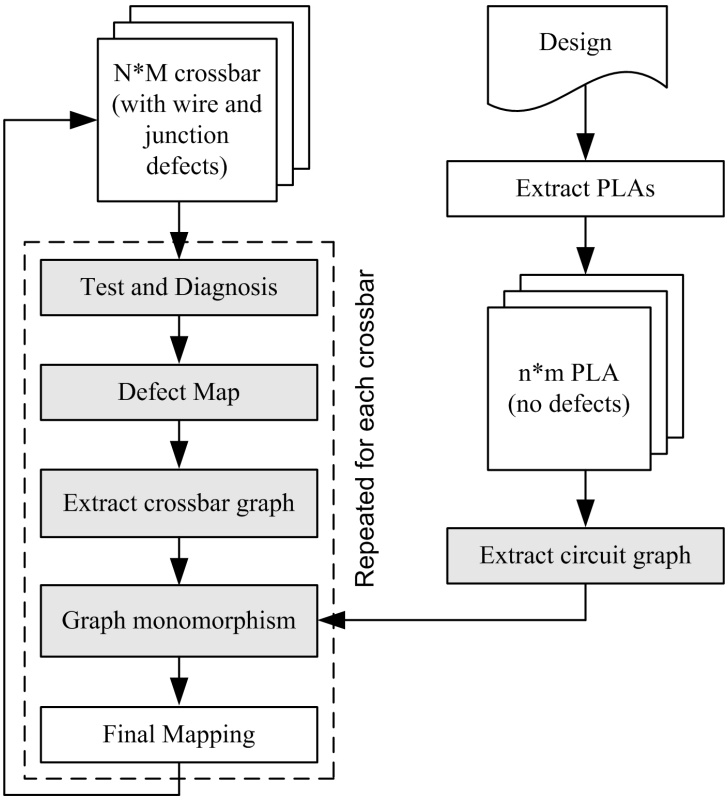


FIGURE 5.2: Defect-aware design flow [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004]

nanoscale PLAs [Naeimi and DeHon, 2004]. Consequently, applying such defect tolerance techniques will result in a significant amount of time needed to map around physical defects.

The built-in-self-test techniques proposed in [Brown and Blanton, 2007, Garcia and Orailoglu, 2008, Wang and Chakrabarty, 2007] use sets of fault detection configurations for diagnosing the block-under-test. Such techniques will suffer from long testing delays because they test each nanoblock in the nanofabric using all the possible configurations to build an accurate defect map for it. To enhance the diagnosis resolution, the configuration set increases, and as the size of the configuration set increases testing becomes a major bottleneck in the manufacturing process. In [DeHon and Naeimi, 2005], the authors suggested using an on-chip microprocessor responsible for defect mapping.

The allocation algorithm proposed in [Hogg and Snider, 2007] which is based on graph monomorphism to find a possible embedding of the circuit's graph into the crossbar's graph represents another bottleneck in the fabrication process. Graph monomorphism computation time depends on many factors such as the circuit's size and the defect rate in the fabric. The topological properties of the graphs also affect the computation time needed to find a monomorphism between the two graphs. Similarly in [Naeimi and DeHon, 2004], the algorithm proposed checks if every OR term in the circuit to be implemented matches any of the defective wires in the crossbar before mapping around

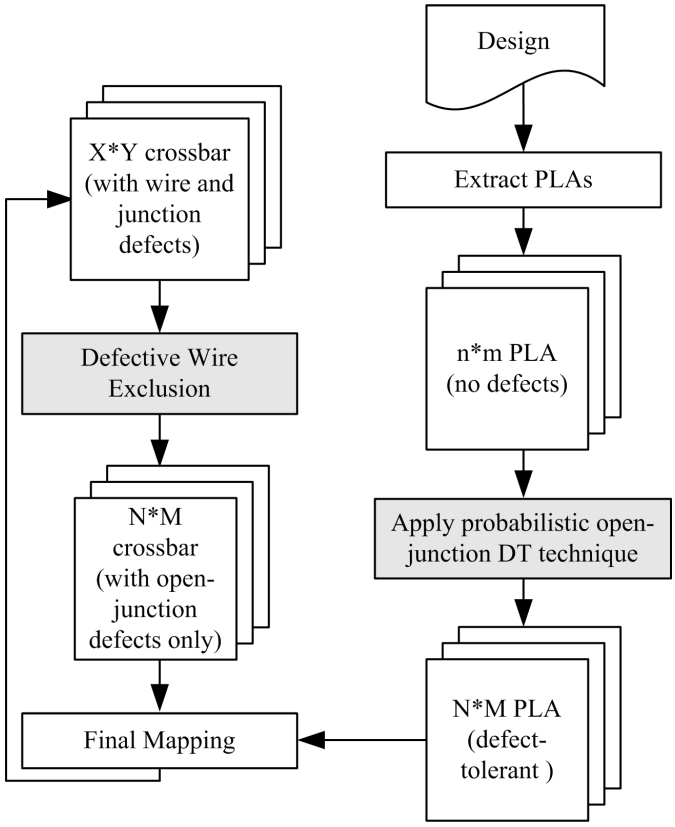


FIGURE 5.3: Proposed design flow

the defective junctions. According to the authors in [Naeimi and DeHon, 2004], assuming that the number of inputs N, number of OR functions F and number of nanowires W are equal, the proposed algorithm takes N^2 program and test operations. Therefore, as the circuit size increases, the computation time to map the circuit onto the defective crossbar increases exponentially.

In this chapter, a probabilistic defect tolerance design flow is proposed to allow the successful implementation of logic circuits onto defective crossbars. To avoid the significant time overhead in diagnosing the chip and mapping the logic around the defective components, such as the work in [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004], the proposed design flow takes advantage of the density and reconfigurability of nanoelectronic circuits by raising the granularity of defect diagnosis to the row/column nanowire level. Defect diagnosis is only performed to detect defective rows and columns in nanoscale PLAs. Defects in crosspoints are masked by logically compensating for the terms altered by the physical defects as will be outlined in Section 5.4. In a PLA with n input lines, p product lines and m output lines, there are $\big((n+m)\times p\big)$ crosspoints to diagnose at the crosspoint level. However, at the row/column nanowire level, there are only $(n+m+p)$ rows and columns to examine. Therefore, by decreasing the diagnostic accuracy from crosspoint to row/column nanowire level, the testing overhead is significantly reduced especially as the size of the PLA increases.

Fig. 5.3 illustrates the proposed design flow where the need to obtain a defect map for the physical crossbar is completely eliminated leading to the elimination of the need to implement any graph monomorphism algorithm [Hogg and Snider, 2007, Al-Yamani et al., 2007, Naeimi and DeHon, 2004, DeHon and Naeimi, 2005]. In the proposed design flow, testing overhead is significantly reduced as compared to the design flow shown in Fig. 5.2. In Fig. 5.2, testing should cover both nanowire and open/closed junction defects. However, in the proposed design flow in Fig. 5.3, testing is only required to locate the defective nanowires and to exclude them. It is worth mentioning that testing for nanowire defects, caused by broken nanowires or closed-junction defects, is simpler to implement and requires less time overhead as compared to testing for open-junction defects. This is because the diagnosis granularity required to locate such defects is limited to the row/column nanowire level rather than the junction level. Defective nanowires caused by broken nanowires or closed-junction defects are located by testing the nanowires individually in the PLA. Spare nanowires are used to compensate for the defective ones, as will be shown in Section 5.5.1. However, testing for open-junction defects, which is needed in the design flow in Fig. 5.2 to build the defect map, requires programming each junction individually and applying a number of test vectors to decide if a given junction is an open-junction or not. In the case of a 20×20 PLA, for instance, only 40 wires are tested for broken wires and closed-junction defects. However, testing for open-junction defects requires testing $20 \times 20 = 400$ junctions. Therefore, the proposed design flow will not only enhance the manufacturing yield but also simplify the process of programming the PLA by reducing testing time.

In the proposed probabilistic design flow, programming a defective crossbar is done through two steps. The open-junction defect tolerance technique (Section 5.7) is applied only once to the extracted PLAs (i.e. performed off-line) and the obtained defect-tolerant $N \times M$ PLAs are used to directly program the crossbars. The amount of redundant resources needed to achieve the predefined yield is mathematically computed based on the dimensions of the PLA and the probability of a crosspoint being open-junction P_{oj}. In the second step, before programming the crossbar, it is first examined and if any defective nanowires are detected, they will be excluded and replaced with spare defect-free nanowires (i.e. on-line repair) (Section 5.5). The number of allocated redundant wires for replacement in the AND and OR planes are calculated based on the targeted yield, the dimensions of the PLA and the probability of a wire being defective P_{wire}.

5.4 Motivational Example

Fig. 5.4 illustrates how the graph-based DT technique, proposed in [Hogg and Snider, 2007], maps logic circuits onto defective crossbars. Nanowire and junction defects need to be located and then the circuit is mapped around these defects (i.e. a *Defect Avoidance* technique). A successful instantiation is achieved when a matching is found between the

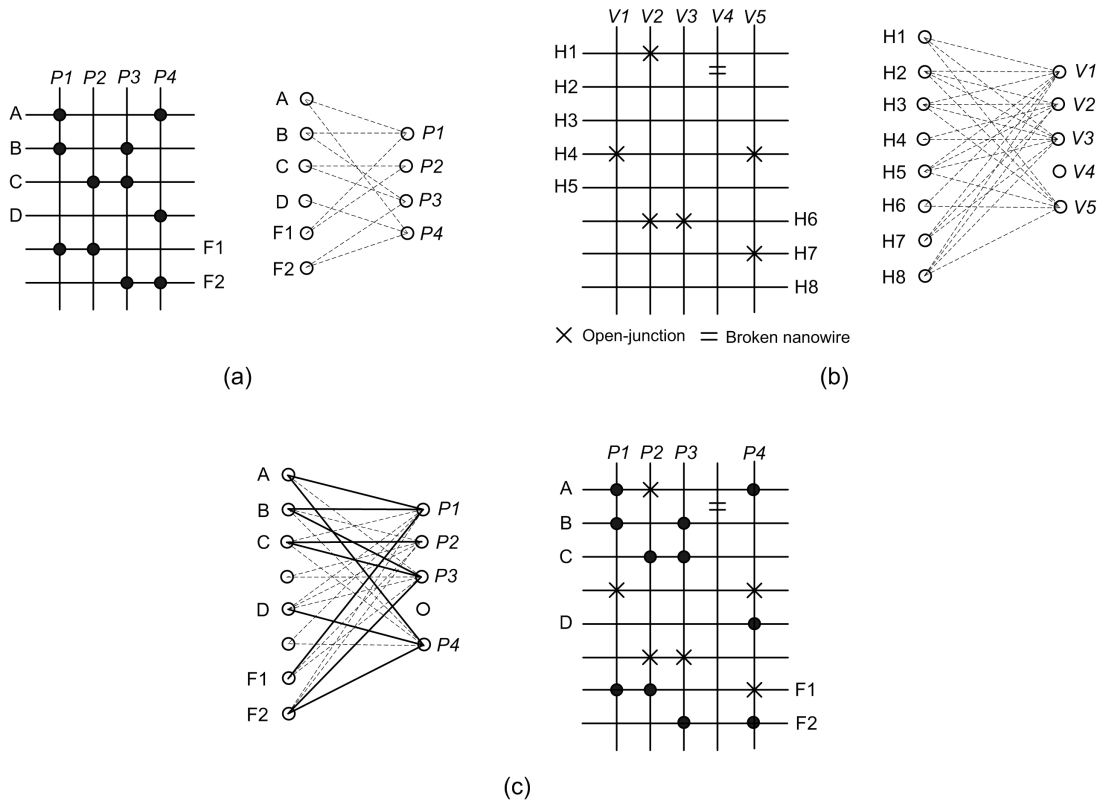


FIGURE 5.4: Defect-aware DT technique [Hogg and Snider, 2007] (a) Creating circuit graph where nanowires and active junctions are represented by nodes and edges respectively (b) Creating crossbar graph where defective junctions and broken nanowires are represented by removing their edges (c) Graph monomorphism is searched between circuit and crossbar graphs for successful implementation

circuit graph and the crossbars graph, as shown in Fig. 5.4(c).

In the proposed design flow (Fig. 5.3), defective crossbars are set to successfully implement logic circuits by allocating redundant input, product and output wires to tolerate defective nanowires and junctions. In Fig. 5.5(c), if the redundant input nanowire D is not used, the function $F2$ will be equal to $F2 = BC + A$ which will result in the wrong output. Similarly, in the absence of the redundant product nanowire $P4$, the output $F2$ will become $F2 = BC$. However, by introducing the redundant nanowires D and $P4$ and in the presence of the defects shown in Fig. 5.5(b), the output is $F2 = BC + AD$ which is equal to the expected output. This example illustrates how errors in logic functions, incurred by physical defects, are tolerated using redundant input and product nanowires and this will be used in the DT techniques comprising the proposed design flow and which will be described in Sections 5.5.1 and 5.7.2. This type of DT techniques are called defect masking techniques. The advantage of these techniques, as compared to the graph-based technique shown in Fig. 5.4, is that no defect mapping or graph matching is required to successfully map logic circuits and thus significantly reducing the testing phase during manufacturing which is the main objective of this chapter.

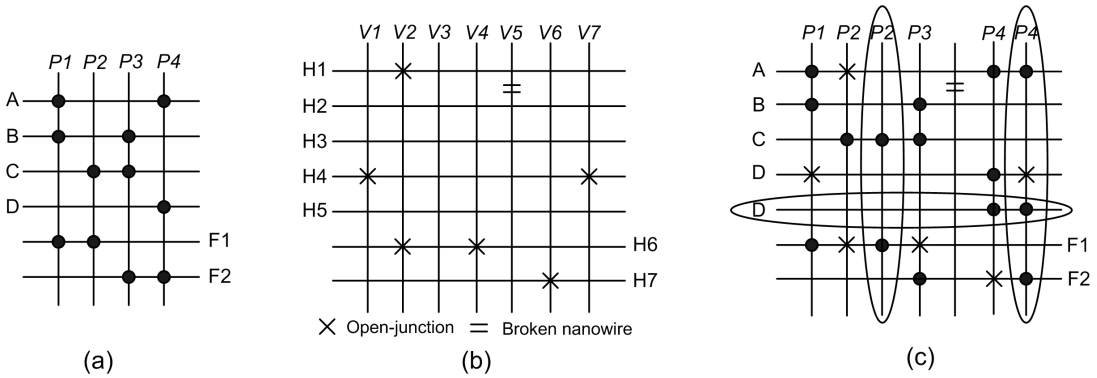


FIGURE 5.5: Proposed DT technique (a) Original circuit (b) Defective crossbar (c) Mapping circuit onto defective crossbar where redundant nanowires are used to compensate for the missing variables and product terms caused by defective junctions and broken nanowires

5.5 Nanowire Defects

According to [Wehn et al., 1988, Wu et al., 1999], testing procedures and defect tolerance techniques targeting CMOS-based PLAs only tackle junction defects. In [Wehn et al., 1988], for example, defects involving input and output lines are deemed irreparable and a chip with such a defect must be rejected. Because of the high defect probability of nanowires, such a decision to discard a chip with defective nanowires will lead to a prohibitively low manufacturing yield. Therefore it is necessary to devise an efficient DT technique to tolerate the highly defective nanowires projected in future nanoscale PLAs. In this section, a probabilistic DT technique to address this problem is proposed.

5.5.1 Proposed Probabilistic DT Technique for Nanowire Defects

Nanowire defects are tolerated by allocating sufficient spare wires for each nanoscale PLA in the fabric. Defective nanowires including product, input and output wires in a $N \times (I + O)$ PLA are replaced with spare defect-free wires as illustrated in Fig. 5.6. Because any given PLA requires a minimum number of input, output and product wires for a successful instantiation on the defective fabric, a probabilistic technique is proposed in this section to estimate the number of spare product wires N_{sp}, spare input wires I_{sp} and spare output wires O_{sp} to ensure sufficient usable nanowires to meet the logic requirements (i.e. I input wires, N product wires and O product wires).

The product repair failure probability P_{prod} is the probability that the number of defect-free wires within product wires and the allocated redundant nanowires N_{sp} is less than the required number of product lines N. Theoretically, given a certain number of spare product wires N_{sp}, P_{prod} is equal to the probability of having more than N_{sp} defective

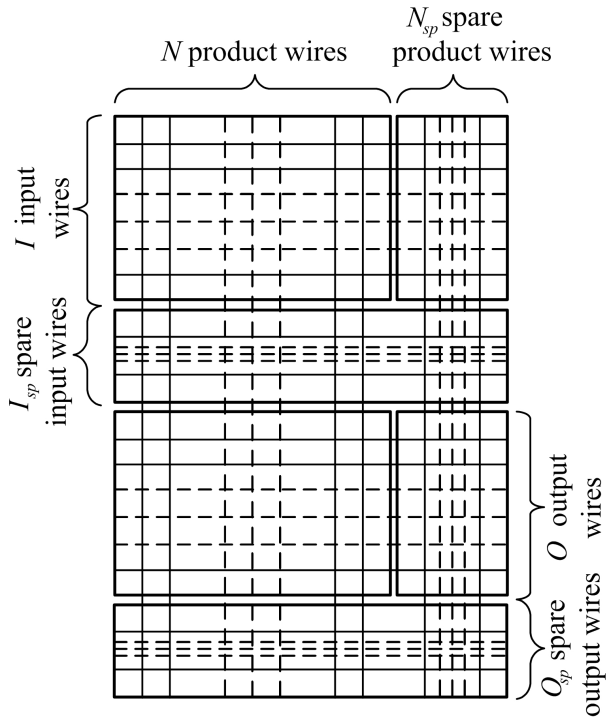


FIGURE 5.6: Implementation of a $N \times (I + O)$ PLA with N_{sp} spare product wires, I_{sp} spare input wires and O_{sp} spare output wires

wires within $N + N_{sp}$ wires and this is given by the following binomial equation:

$$P_{prod} = \sum_{k=N_{sp}+1}^{N+N_{sp}} \binom{N + N_{sp}}{k} P_{wire}^{k} (1 - P_{wire})^{N+N_{sp}-k} \tag{5.1}$$

where P_{wire} is the probability that a nanowire is defective and N is the original number of product wires of the PLA. Similarly, in the AND-plane, given the number of input wires I in the PLA and the number of spare input wire I_{sp}, the input wire repair failure probability P_{ip} is given below:

$$P_{ip} = \sum_{k=I_{sp}+1}^{I+I_{sp}} \binom{I + I_{sp}}{k} P_{wire}^{k} (1 - P_{wire})^{I+I_{sp}-k} \tag{5.2}$$

In the OR-plane, given the number of output wires O and the number of spare output wires O_{sp}, the probability P_{op} is:

$$P_{op} = \sum_{k=O_{sp}+1}^{O+O_{sp}} \binom{O + O_{sp}}{k} P_{wire}^{k} (1 - P_{wire})^{O+O_{sp}-k} \tag{5.3}$$

Algorithm 2 Nanowire Defect Repair Algorithm

1: Initialise N, I, O, Y_{wire}, P_{wire}, N_{max}, I_{max}, O_{max}, $area$
2: Initialise $N_{sp} = 0$, $I_{sp} = 0$, $O_{sp} = 0$ and $area = 0$
3: **for all** i such that $1 \leq i \leq N_{max}$ **do**
4: **for all** j such that $1 \leq j \leq I_{max}$ **do**
5: **for all** k such that $1 \leq k \leq O_{max}$ **do**
6: Calculate $P_{prod\_failure}$ // Equation 5.1
7: Calculate $P_{ip\_failure}$ // Equation 5.2
8: Calculate $P_{op\_failure}$ // Equation 5.3
9: Calculate $P_{failure}$ // Equation 5.4
10: **if** $P_{failure} \leq 1 - Y_{wire}$ **then**
11: **if** $area = 0$ **then**
12: $area = (N + i) \times (I + j + O + k)$
13: $N_{sp} = i$ and $I_{sp} = j$ and $O_{sp} = k$
14: **else**
15: **if** $(N + i) \times (I + j + O + k) < area$ **then**
16: $area = (N + i) \times (I + j + O + k)$
17: $N_{sp} = i$ and $I_{sp} = j$ and $O_{sp} = k$
18: **end if**
19: **end if**
20: **end if**
21: **end for**
22: **end for**
23: **end for**
24: **if** $area = 0$ **then**
25: **return** error // No feasible solution found
26: **end if**

The final failure probability of the proposed repair technique is:

$$\begin{aligned} P_{failure} &= P_{prod}P_{ip}(1 - P_{op}) + P_{prod}(1 - P_{ip})P_{op} + P_{prod}(1 - P_{ip})(1 - P_{op}) \\ &\quad + (1 - P_{prod})P_{ip}P_{op} + (1 - P_{prod})P_{ip}(1 - P_{op}) + (1 - P_{prod})(1 - P_{ip})P_{op} \\ &\quad + P_{prod}P_{ip}P_{op} \\ P_{failure} &= P_{prod} + P_{ip} + P_{op} - P_{prod}P_{ip} - P_{prod}P_{op} - P_{ip}P_{op} + P_{prod}P_{ip}P_{op} \end{aligned} \tag{5.4}$$

Algorithm 2 describes how the proposed technique allocates the least number of spare wires needed to achieve a predefined yield Y_{wire}. The aim is to find the smallest possible crossbar of size $(N + N_{sp}) \times (I + I_{sp} + O + O_{sp})$ that is able to instantiate a PLA of size $N \times (I+O)$ at a wire defect rate P_{wire}. The technique starts with one redundant product wire, one redundant input wire and one redundant output wire and loops through all possible combinations of N_{sp}, I_{sp} and O_{sp} where $N_{sp} \leq N_{max}$, $I_{sp} \leq I_{max}$ and $O_{sp} \leq O_{max}$ (lines 3, 4 and 5 in Algorithm 2). In every iteration, it calculates the yield using equations 5.1, 5.2, 5.3 and 5.4 (lines 6, 7, 8 and 9 in Algorithm 2). If the yield obtained is smaller than the predefined value (line 10 in Algorithm 2), N_{sp}, I_{sp} and O_{sp} are increased until the required yield is obtained.

5.5.2 Experimental Results (Nanowire DT Technique)

Table 5.1 shows the yields obtained for different PLA sizes after applying the proposed algorithm in the presence of various defect rates (5%-20%) and various targeted yields (90%-99%). The yield is computed using the simulation procedure outlined in Section 5.1. Broken nanowires and closed-junction defects were randomly injected into the PLAs after allocating the required redundant wires N_{sp}, I_{sp} and O_{sp} to achieve the targeted yield Y_{wire}. In every iteration, the defective PLA was repaired by excluding the defective wires and replacing them with the allocated redundant wires. As can be seen, for all PLA sizes and all defect rates, the proposed algorithm successfully achieves the targeted yields. This is demonstrated by values higher than the targeted yields. The table also summarises the relative area overhead required to achieve 90%, 95% and 99% yield ratios as a function of the targeted yield, defect rate and PLA size. The relative area is the ratio of PLA size after allocating the spare wires for repair to the actual size of the PLA before repair. As can be observed, the relative area ratio decreases as the PLA size increases. Therefore, the proposed nanowire defect tolerance technique is considered more efficient when repairing bigger PLA circuits because smaller proportional area overhead is required to achieve the targeted yield. Moreover, the area overhead is proportionally related to the defect rate and the targeted yield with more redundancy needed to tolerate higher defect rates and to achieve higher manufacturing yields. It is worth highlighting the efficiency of the proposed algorithm in tolerating high defect rates at a reduced area overhead. This is illustrated by an average relative area of only 2.5 (i.e. the redundant area is equal to 150% of the actual size of PLA) at a defect rate as high as 20% and a targeted yield of 99%.

5.6 N-bit Adder Circuit Implementation

Before introducing the open-junction DT technique, the different methods of implementing logic circuits using nanoscale PLA architecture are outlined. N-bit adder circuit was chosen as a benchmark circuit to demonstrate the impact of circuit implementation methods on yield and area overhead. In this section, three recently reported methods of mapping N-bit adders onto crossbar fabrics are explored.

In [Hogg and Snider, 2007], the authors suggested two different methods of implementing N-bit adders using nanoscale PLAs. The first one is shown in Fig. 5.8(a) which is a multiple-stage (MS) N-bit adder built using a chain of 1-bit full adders. Fig. 5.8(b) illustrates the implementation of the multiple-stage 3-bit adder shown in Fig. 5.7 producing output bits $S_0 \dots S_3$ representing the sum of the two 3-bit numbers $A_0 \dots A_2$ and $B_0 \dots B_2$. The least significant bit of the sum S_0, for instance, is computed as $S_0 = A_0\bar{B}_0 + \bar{A}_0B_0$ using the bottom wire of the crossbar and the first and second columns from the left. As can be seen, this implementation uses several levels of logic,

$N \times (I + O)$ PLA	5% defects		10% defects		15% defects		20% defects	
	Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area
Targeted Yield=90%								
$50 \times (10 + 10)$	90.83%	1.32	90.24%	1.56	90.44%	1.87	90.04%	2.21
$75 \times (15 + 15)$	91.58%	1.29	90.19%	1.50	90.14%	1.77	90.33%	2.08
$100 \times (20 + 20)$	90.56%	1.25	91.25%	1.48	90.55%	1.72	90.28%	2.00
Targeted Yield=95%								
$50 \times (10 + 10)$	95.63%	1.36	95.15%	1.70	95.21%	1.98	95.11%	2.37
$75 \times (15 + 15)$	95.84%	1.32	95.06%	1.59	95.02%	1.87	95.46%	2.21
$100 \times (20 + 20)$	95.88%	1.30	95.17%	1.54	95.15%	1.79	95.79%	2.12
Targeted Yield=99%								
$50 \times (10 + 10)$	99.15%	1.51	99.06%	1.89	99.08%	2.27	99.04%	2.73
$75 \times (15 + 15)$	99.25%	1.44	99.31%	1.74	99.14%	2.10	99.02%	2.47
$100 \times (20 + 20)$	99.21%	1.38	99.07%	1.65	99.03%	1.96	99.14%	2.33

TABLE 5.1: Targeted vs. actual yields obtained using the proposed probabilistic nanowire DT technique

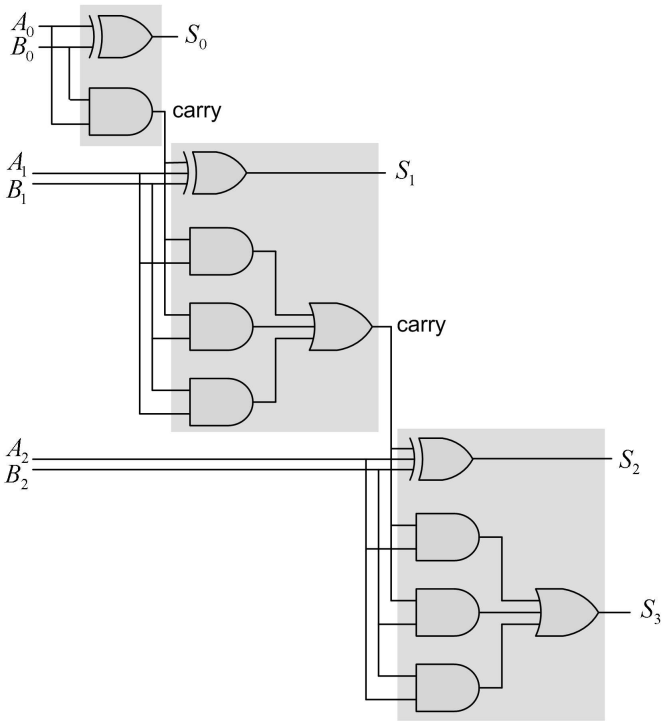


FIGURE 5.7: Schematic of a 3-bit adder

some of the intermediate output signals must be fed back to some of the inputs. The circuit uses 12 input wires ($A_0A_1A_2$, $\bar{A}_0\bar{A}_1\bar{A}_2$, $B_0B_1B_2$ and $\bar{B}_0\bar{B}1\bar{B}_2$) and 4 output wires ($S_0S_1S_2S_3$) in addition to the feedback signals giving a total of 30 rows. Forming the required logical operations on these input rows uses 25 columns. Therefore, this PLA implementation of the 3-bit adder requires a minimum crossbar area of $30 \times 25 = 750$ junctions.

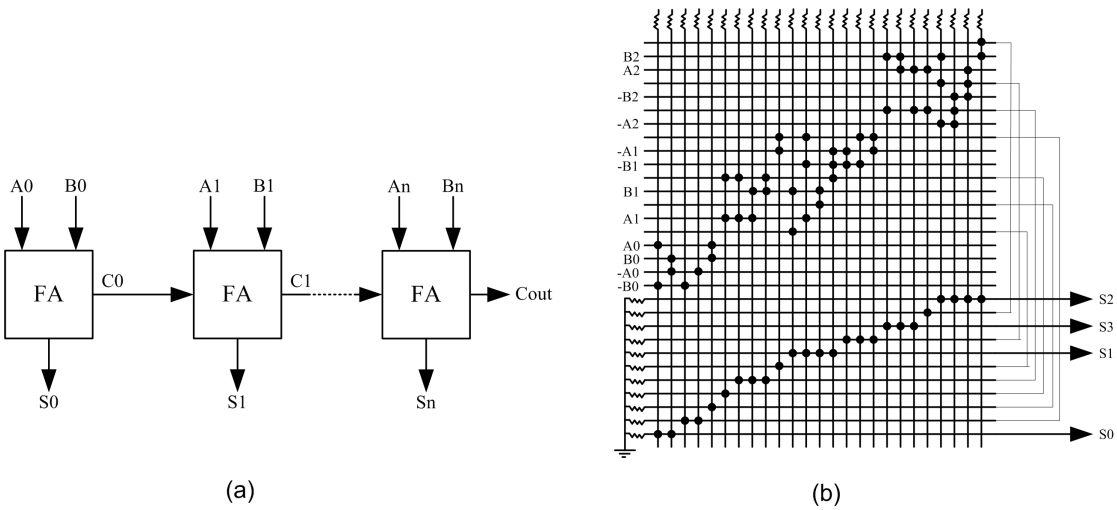


FIGURE 5.8: (a) A multiple-stage (MS) N-bit adder implemented as a ripple-carry logic chain of 1-bit adders. (b) PLA implementation of a multiple-stage 3-bit adder

The second method proposed in [Hogg and Snider, 2007] is an implementation of single-stage (SS) N-bit adders which is shown in Fig. 5.9(a). Fig. 5.9(b) shows the PLA

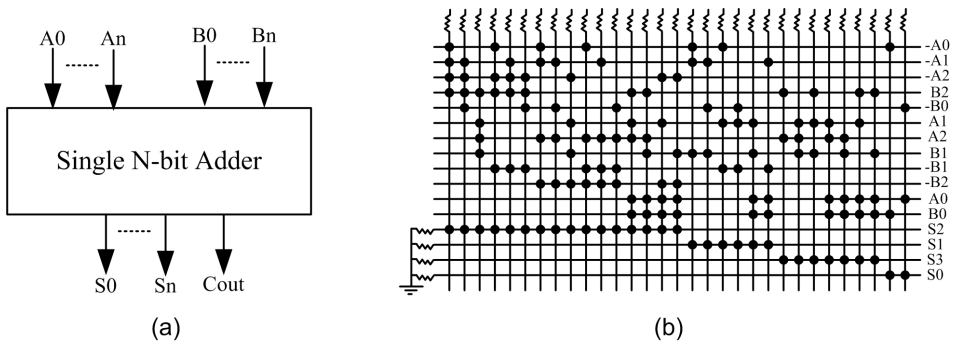


FIGURE 5.9: (a) A single-stage (SS) N-bit adder. (b) PLA implementation of a single-stage 3-bit adder

implementation of a 3-bit single-stage adder. This implementation requires only 16 rows and 31 columns and hence a minimum crossbar area of $16 \times 31 = 496$. As can be observed, the need for intermediate signals is eliminated and hence no feedback wires. This approach also requires only enough rows to handle the input and output wires i.e. 16 rows. In Fig. 5.8(b), the output signal S_1 for instance is computed using 4 intermediate signals. In Fig. 5.9(b), on the other hand, S_1 is computed straightforward using the following logical expression $S_1 = \bar{A}_0 A_1 \bar{B}_1 + \bar{A}_0 \bar{A}_1 B_1 + A_1 \bar{B}_0 \bar{B}_1 + \bar{A}_1 \bar{B}_0 B1 + A_0 \bar{A}_1 B_0 \bar{B}_1 + A_0 A_1 B_0 B_1$. This logical implementation of signal S_1 uses a significant number of AND and OR functions and this explains why the number of active crosspoints in this implementation is significantly bigger than the number in the previous one.

According to [Ziegler and Stan, 2003], there are two scenarios to be considered in terms of circuit mapping in self-assembled nanoscale fabrics. One scenario allows the dimensions of each crossbar array to be different and controlled at fabrication time. This will allow the implementation of the two suggested mappings of the 3-bit adder (Fig. 5.8(b), Fig. 5.9(b)). The second scenario involves a self-assembly process that can produce only uniformly-sized crossbar arrays or nano-blocks. The circuit to be mapped is first partitioned into equally-sized portions and then each portion is implemented by a single nano-block. This implementation is named multiple-stage multiple-block (MSB) technique. Fig. 5.10 shows an implementation of a 3-bit adder using 3 equally-sized nano-blocks. This approach requires less area (only 181 junctions) and utilises fewer active junctions with respect to the previously proposed approaches which are shown in Fig. 5.8(b) and Fig. 5.9(b). Nano-block 1 implements the first half adder and the second and third 1-bit full adders are implemented in the second and third nano-blocks respectively. Because PLA crosspoints are unable to perform inversion operation [Wang and Chakrabarty, 2007], complemented inputs are required and the complement of each output variable needs to be computed. In Fig. 5.10, nano-block 2 for instance is required to compute C_1 and also $\bar{C}_1$.

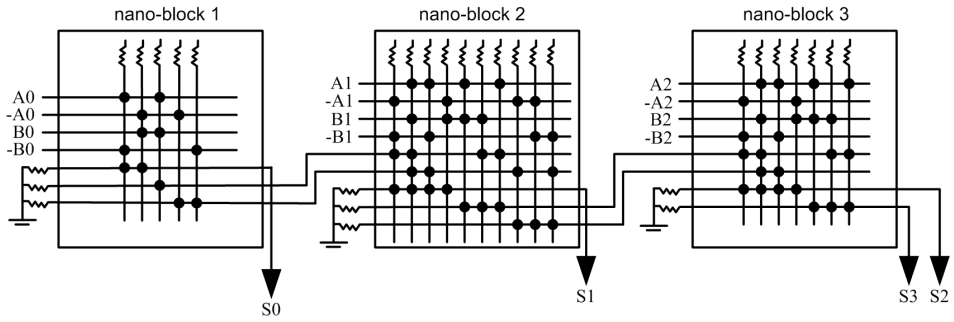


FIGURE 5.10: A multiple-stage 3-bit adder implemented using three equally-sized nano-blocks (MSB)

Circuit	Area	5% defects	10% defects	15% defects	20% defects
1-bit full adder	56	30.60%	4.58%	0.53%	0.03%
3-bit adder MS	750	1.57%	0.01%	0%	0%
3-bit adder SS	496	0.02%	0%	0%	0%
3-bit adder MSB	181	2.88%	0.02%	0%	0%

TABLE 5.2: Manufacturing yields when no DT technique is applied to nanoscale PLAs

5.7 Open-Junction Defects

In this section, the impact of open junction defects on the manufacturing yield of the adder circuits described in the previous section is first examined when no DT technique is applied to nanoscale PLAs. The yield is given by equation 5.5 where J is the number of active junctions in the PLA and P_{oj} is the probability of a junction being open-junction. The obtained yields are tabulated in Table 5.2. As can be seen, the presence of defect rates as low as 5% gives a yield of 30% for the 1-bit adder PLA and as the PLA size increases, the yield decreases to values close to 0%. The yield also drops rapidly as the defect rate increases. These results dictate the necessity for an efficient DT technique to tolerate the high rates of open-junction defects projected in future nanoscale PLAs.

$$Yield = 1 - \sum_{k=1}^{J} \binom{J}{k} P_{oj}^{k} (1 - P_{oj})^{(J-k)} \tag{5.5}$$

5.7.1 Open-Junction Defect Masking in Nanoscale PLA

This section focuses on the physical defects that lead to open-junction defects. It was reported in [Naeimi and DeHon, 2004, Rao et al., 2007] that open-junctions will be the dominant defect in future nanoscale PLA fabrics. Table 5.3 shows the effect of open-junction defects on the logical output of function f. When the fault occurs in the AND-plane, the corresponding variable is dropped from the product term and results in the wrong logical formula. Similarly, when the fault occurs in the OR-plane, the

Location	Expected function	Fault effect in logic	Actual function
AND-plane	$f = ab + cd$	missing a variable (a) in a product term (ab)	$f' = b + cd$
OR-plane	$f = ab + cd$	missing a product term (ab) in a logic function (f)	$f' = cd$

TABLE 5.3: Stuck-open defect manifestation in nanoscale PLA

corresponding product term is dropped from the expected output.

The probabilistic DT technique to be proposed in the next section is based on a fault masking scheme that introduces redundancy within the logic function to be implemented. In the following equation $\widehat{f_{AND}}$ is logically equivalent to f but is able to mask any single open-junction defect within the AND-plane.

$$f = ab \equiv aabb = \widehat{f_{AND}}$$

Similarly, any open-junction defect in the OR-plane can be masked with a redundant form $\widehat{f_{OR}}$ given by the following equation:

$$f = a + b \equiv a + a + b + b = \widehat{f_{OR}}$$

In the case of a sum-of-product two-level Boolean logic function where open-junction defects in both AND and OR planes are considered, AND and OR functions are insufficient to mask defects in both planes as illustrated by the following example:

$$\begin{aligned} f = ab + cd \quad &\Rightarrow \quad \widehat{f_{AND}} = abab + cdcd \\ &\Rightarrow \quad \widehat{f_{OR}} = ab + cd + ab + cd \end{aligned}$$

As can be observed, both $\widehat{f_{AND}}$ and $\widehat{f_{OR}}$ functions can only mask open-junction defects in one plane of the PLA, yet is susceptible to defects in the other plane. For example, if variable a is dropped from $\widehat{f_{OR}}$ due to an open-junction defect in the AND-plane, $\widehat{f_{OR}}$ will result in an erroneous function $f = b + cd$. Similarly, if the product $abab$ is dropped from $\widehat{f_{AND}}$ due to an open-junction defect in the OR-plane, the output function will be $f = cdcd$. Therefore, in order to mask open-junction defects in both AND and OR planes of a two-level PLA logic, the following function $\widehat{f}$ needs to be implemented:

$$f = ab + cd \equiv \widehat{f} = abab + abab + cdcd + cdcd$$

Fig. 5.11 illustrates the fault masking scheme in the nanoscale PLA architecture. Fig. 5.11(a) shows the original logic function $f = ab + cd$ and Fig 5.11(b) shows the implementation of $\widehat{f}$ which necessitates a doubling in the number of variable wires and product term wires. To tolerate higher defect rates, the number of redundant variables and product terms

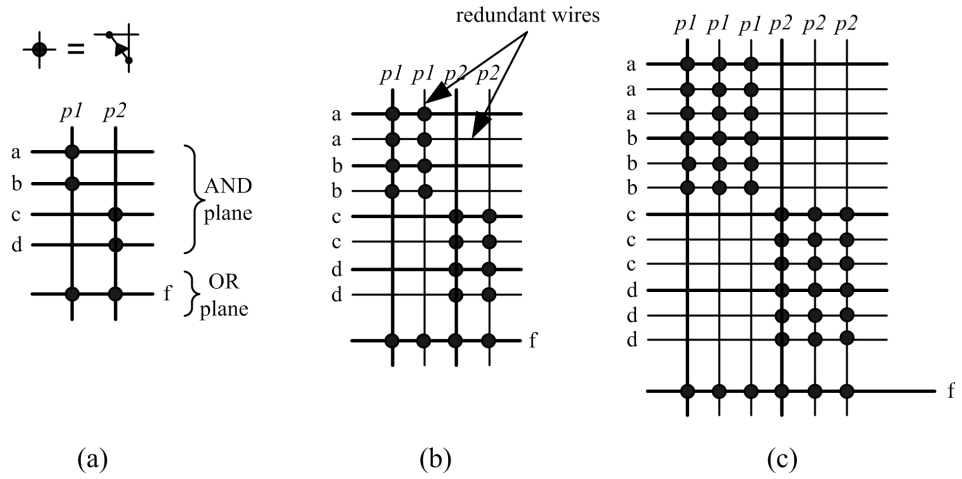


FIGURE 5.11: Different levels of fault masking of function $f = ab + cd$ in nanoscale PLA architecture

can be further increased as shown in Fig. 5.11(c) which implements the logic function $\widehat{f} = ababab + ababab + ababab + cdcdcd + cdcdcd + cdcdcd$.

5.7.2 Proposed Probabilistic DT Technique for Open-Junctions

The second probabilistic DT technique to be proposed in this section addresses tolerating the dominant open-junction defects in future nanoscale PLAs. The aim is to define the required number of variable and product term redundant wires in both AND and OR planes of a given PLA to achieve a predefined yield Y_{cross}.

For a given targeted yield, the failure rate $P_{failure}$ is given by the following equation:

$$P_{failure} = 1 - Y_{cross} \tag{5.6}$$

$P_{failure}$ is equal to the sum of failure rates of all output wires. For a Boolean function with O output wires, the proposed DT technique must satisfy the following condition:

$$\sum_{k=1}^{O} P_{ORwire(k)} \leq P_{failure} \tag{5.7}$$

where $P_{ORwire(k)}$ is the failure rate of output wire k.

$\widehat{P}_{ORwire(k)}$ represents the upper limit of $P_{ORwire(k)}$ after injecting redundant wires (i.e. $P_{ORwire(k)} \leq \widehat{P}_{ORwire(k)}$) and is obtained by multiplying $P_{failure}$ by the ratio of the number of product terms ($PJ(k)$) in output wire k to the total number of product

terms in all output wires.

$$\widehat{P}_{ORwire(k)} = \frac{PJ(k)}{\sum_{x=1}^{O} PJ(x)} \times P_{failure} \tag{5.8}$$

Equ. 5.8 allows output wires with small number of active junctions to have a low $\widehat{P}_{ORwire}$ and output wires with bigger number of active junctions to have a higher $\widehat{P}_{ORwire}$. Such a distribution of failure rate $P_{failure}$ across the output wires reduces the amount of redundant wires as output wires with small number of active junctions have a lower probability to be defective than output wires with bigger number of active junctions.

The failure rate of output wire k with m product terms is equal to the probability that at least one of the m product term junctions is erroneous and is calculated using the following binomial equation:

$$P_{ORwire(k)} = \sum_{x=1}^{m} \binom{m}{x} P_{ORcross(k)}^{x} (1 - P_{ORcross(k)})^{(m-x)} \tag{5.9}$$

where $P_{ORcross(k)}$ is the probability that any of the product terms m in output wire k is dropped due to an open-junction defect in the OR-plane or its logical value is erroneous because of a defect in the AND-plane. This is given by the following equation:

$$P_{ORcross(k)} = P_{oj}^{(\alpha+1)} + \sum_{x=1}^{\alpha+1} \binom{\alpha+1}{x} P'^{x} (1 - P')^{(\alpha+1-x)} \tag{5.10}$$

where P_{oj} is the probability of a crosspoint being open-junction, α is the number of redundant product nanowires allocated to an active junction within output wire k. A product term crosspoint becomes defective if all of its junctions are open junctions in the OR-plane or because of an open junction in one of the product wires in the AND-plane that alters the logical expression of the product term itself which is given by equation 5.11 as follows:

$$P' = \sum_{x=1}^{m} \binom{m}{x} \left((1 - P_{oj}) P_{ANDwire(k')}\right)^{x} \left(1 - (1 - P_{oj}) P_{ANDwire(k')}\right)^{(m-x)} \tag{5.11}$$

$P_{ANDwire(k')}$ is the probability that product nanowire k' ($k' \leq m$) in the AND-plane is defective. A product wire is considered defective if at least one of its active junctions with all its allocated redundant wires are open junctions. Equation 5.12 calculates $P_{ANDwire(k')}$.

$$P_{ANDwire(k')} = \sum_{x=1}^{n} \binom{n}{x} P_{oj}^{(\alpha'+1)\times x} (1 - P_{oj}^{(\alpha'+1)})^{(n-x)} \tag{5.12}$$

where n is the number of variable terms in nanowire k' and α' is the number of allocated redundant wires.

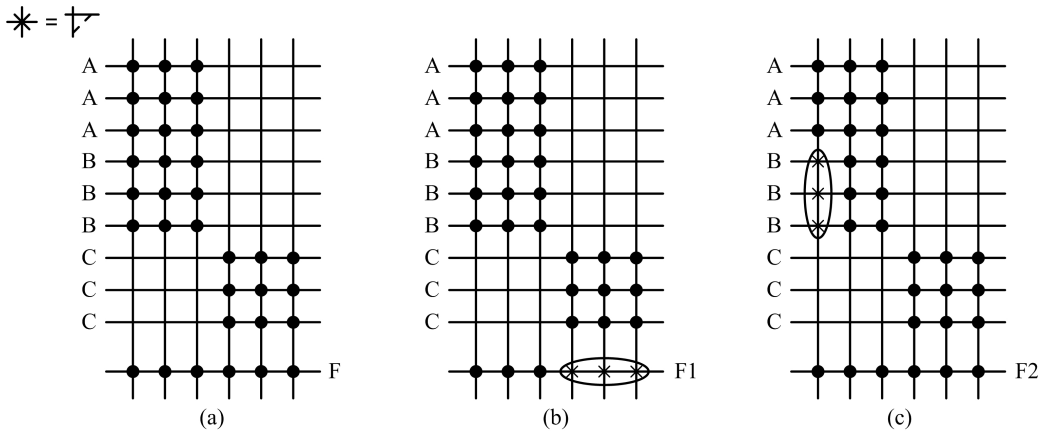


FIGURE 5.12: Open junction defects leading to erroneous output of logic functions implemented in nanoscale PLAs. (a) Original function $F = AB + C$ (b) Wrong logic function $F1 = AB$ (c) Wrong logic function $F2 = A + C$

Fig. 5.12 above illustrates how open junction defects lead to an erroneous output. In Fig. 5.12(b), the product term C disappears from $F1$ because its crosspoint and its two redundant crosspoints are all open junctions. Such defects result in a logic function $F1 = AB$ which is different from the expected $F = AB + C$. Another scenario where the original function F can be altered due to open junctions is shown in Fig. 5.12(c). When the crosspoint representing one input variable and all its redundant crosspoints on the same product wire are open junctions, this leads to the disappearance of that input variable from the logical formula of F. The output becomes $F2 = AAA + AAABBB + AAABBB + CCC + CCC + CCC$ and this is equivalent to $F2 = A + C$ which is different from the original logic function F.

Below is an algorithmic description of the proposed probabilistic technique to achieve the predefined targeted yield Y_{cross}. The technique tolerates a given defect rate P_{oj} in a nanoscale fabric implementing logic circuits as PLAs with I number of inputs and O number of outputs. Other predefined values are α_{max} and α'_{max} which represent the upper limit of redundant wires to be allocated to both product and output wires. The required numbers of redundant wires in both planes (α and α') needed to achieve a yield higher or equal to Y_{cross} are stored in a two-dimensional array $red[I][O]$.

Algorithm 3 scans through all the active crosspoints in the OR-plane and the AND-plane and allocates the minimum number of redundant input and product nanowires required to achieve a junction failure probability $P_{ORcross(j)}$ smaller or equal to the upper limit $\widehat{P}_{ORcross(i)}$. If such condition is satisfied for each junction in all output nanowires, the final yield will definitely be higher or equal to the required yield Y_{cross}. It is worth noting that the above probabilistic approach can only work if the following conditions are satisfied. The first is that open junction defects are randomly distributed across the fabric and that there is no correlation between them. And the second condition is that all nanowire defects are tolerated using the technique proposed in Section 5.5 where any defective nanowire is replaced with a defect-free one.

Algorithm 3 Redundancy Allocation Algorithm For Open-Junction Defects

Initialise I, O, Y_{cross}, P_{oj}, α_{max}, α'_{max}
Initialise the 2D redundancy array $red[I][O] = \{0, 0\}$
Calculate $P_{failure}$ // Equation 5.6
{S}can output wires in OR-plane
for all i such that $1 \leq i \leq O$ **do**
  Calculate $\widehat{P}_{ORwire(i)}$ // Equation 5.8
  Calculate $\widehat{P}_{ORcross(i)}$ // solve Equation 5.9 so that $P_{ORwire(i)} \leq \widehat{P}_{ORwire(i)}$
  {S}can product wires in AND-plane
  for all j such that $1 \leq j \leq m$ **do**
    for all α such that $1 \leq \alpha \leq \alpha_{max}$ **do**
      for all α' such that $1 \leq \alpha' \leq \alpha'_{max}$ **do**
        Get n for product wire j
        Calculate $P_{ANDwire(j)}$ // Equation 5.12
        Calculate $P_{ORcross(j)}$ // Equations 5.10, 5.11
        if $P_{ORcross(j)} \leq \widehat{P}_{ORcross(i)}$ and $red[i][j] = \{0, 0\}$ **then**
          $red[i][j] \leftarrow \{\alpha, \alpha'\}$
        end if
      end for
    end for
    if $red[i][j] = \{0, 0\}$ **then**
      return error // No feasible solution found
    end if
  end for
end for

5.7.3 Yield Optimisation

Algorithm 3 outlined above results in different product and output wires with different levels of redundancy depending on the number of active junctions m and n in each wire. Although the yield obtained from this probabilistic approach is bigger or equal to the predefined yield Y, the yield can be further enhanced by programming more junctions to tolerate higher number of defective crosspoints without changing the logical formulas of output wires. As illustrated in Fig. 5.13, the junctions shown as squares can be programmed into active crosspoints without affecting the outputs O_1 and O_2. In the AND-plane, product wire P_2 requires more redundant wires than product wire P_1, and instead of using only one redundant wire for input wire A, the remaining two redundant wires can be programmed to enhance the yield. Similarly, in the OR-plane, output wire O_2 requires more redundant product wires than O_1 which can exploit all these redundant product wires to enhance yield without affecting the logical value of O_1.

5.7.4 Experimental Results (Open-Junction DT Technique)

This section evaluates the yield improvement that can be achieved by the probabilistic DT technique outlined in Algorithm 3. The different PLA implementations of the N-bit

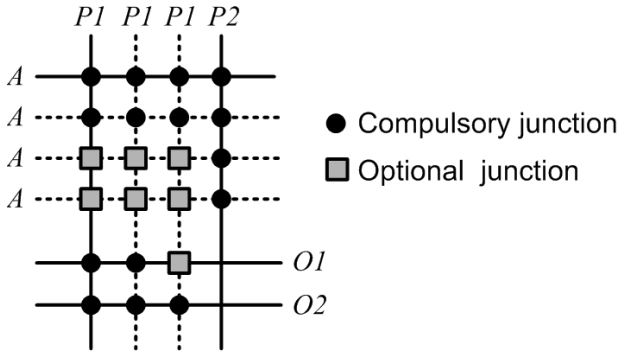


FIGURE 5.13: Yield optimisation by programming more junctions without affecting the logical formulas of output wires

adder circuits presented in Section 5.6 were used as benchmark circuits to measure the yield Y_{cross} and to highlight the impact of PLA size and the density of active-junctions on area overhead. The simulation procedure outlined in Section 5.1 was used to calculate the yield Y_{cross}. The simulations were implemented in C++ where arrays of binary bits were used to represent the PLAs. Open-junction defects were randomly injected into the PLAs by changing the value of bits corresponding to defective junctions. Three targeted yield values (90%, 95% and 99%) and four defect rates (5%, 10%, 15% and 20%) were selected for illustration purposes. The results obtained are shown in Table 5.4. Generally, the proposed probabilistic DT technique improves the yield of the adder circuits as compared to the yields tabulated in Table 5.2. The values obtained are higher than the targeted yields regardless of the PLA size, the density of active-junctions in the PLA and the defect rate. Table 5.4 also includes the area overhead expressed as relative area ratio required to achieve the targeted yields. As expected, higher yields can be achieved using the proposed DT technique at the expense of higher area overheads. The area overhead also increases as the defect rate increases. This is due to the increasing number of defective junctions in the fabric that need to be compensated. In equations 5.10, 5.11 and 5.12, when the defect probability P_{oj} is increased both $P_{ANDwire}$ and $P_{ORcross}$ increase and hence require more spare wires α and α' to obtain the required yield.

Next, the impact of the type of PLA implementation (i.e. SS, MS or MSB) on area overhead is investigated. In [Hogg and Snider, 2007], the proposed graph-based technique incurs less area overhead when using multiple-stage MS rather than single-stage SS implementation of the adder circuits. This is due to the dramatic increase of active-junctions in SS PLA. Similarly, in Table 5.4, SS adder implementation incurs the highest area overhead when mapped to defective PLAs. This is due to the large number of functioning diodes which are needed by this type of implementation. Theoretically, in equations 5.9 and 5.12, it can be observed that if either the number of product terms m in the output wires or the number of variable terms n in the product wires increase, the required number of redundant row and column wires to achieve a given yield will increase. It is worth noting that although the SS adder incurs higher relative area

ratios, the total crossbar area is smaller than the area overhead incurred by the MS adder. For instance, for a targeted yield of 95% and a defect rate of 20%, the crossbar area of MS adder is $750 \times 20.83 = 15622$, whereas the crossbar area of SS adder is only $496 \times 23.75 = 11780$ junctions. Moreover, by comparing the area overheads incurred by the proposed probabilistic technique and the graph-based technique in [Hogg and Snider, 2007], it can be observed that the probabilistic technique is more efficient in tolerating the open-junction defects in the SS adder implementation. As an example, the proposed technique incurs a relative area ratio equal to 19 for a 90% targeted yield and a defect rate of 20% whereas the graph-based technique requires a much higher ratio. However, for the MS adder implementation, the proposed probabilistic technique is considered less effective due to the bigger area overheads incurred by this technique as compared to the graph-based technique. From Table 5.4, it can be observed that the multiple-stage multiple-nanoblock MSB implementation of the 3-bit adder achieves the targeted yield at the lowest possible area overhead. For instance, for a targeted yield of 95% and a defect rate of 20%, the crossbar area after applying the proposed DT technique is $181 \times 19.57 = 3542$ junctions which is significantly smaller than the crossbar area of both SS and MS implementation methods. To achieve the targeted yield Y_{cross}, the proposed technique is applied to each single nano-block to obtain a yield of $Y_{cross}^{\frac{1}{3}}$. In general, if the circuit to be implemented is divided into N equally-sized nano-blocks, the targeted yield for each nano-block should be $Y_{cross}^{\frac{1}{N}}$.

Table 5.5 shows more results for other benchmark circuits, details of which can be found in Section A.3, Appendix A. The proposed probabilistic technique for open-junction defects successfully achieves the targeted yields regardless of the size of the PLAs at different area costs. C17 is a small circuit with few active-junctions. Mapping C17 onto defective crossbars requires the least area overhead as compared to the other bigger benchmark circuits. Moreover, although the 3-bit multiplier requires the largest PLA implementation area, the technique incurs less area overhead to successfully map it than the ALU4 circuit. This is because of the difference in the density of active-junctions in their PLA implementations. Therefore, the area overhead incurred by the proposed probabilistic algorithm depends not only on the size of the PLA but also on the number of active-junctions in it.

5.8 Choice of Defect Rates P_{wire} and P_{oj}

In both proposed techniques, outlined in Sections 5.5.1 and 5.7.2 targeting nanowire and open-junction defects respectively, the amount of redundant resources needed to achieve the predefined yields Y_{wire} (Algorithm 2) and Y_{cross} (Algorithm 3) depend not only on the PLA size and the type of PLA implementation but also on the defect rates P_{wire} and P_{oj}. Therefore, an accurate choice of these two parameters will significantly affect the efficiency of the proposed techniques and hence the design flow shown in Fig. 5.2 in

Targeted Yield=90%									
Adder Circuits	Area	5% defects		10% defects		15% defects		20% defects	
		Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area
1-bit	56	95.19%	5.00	92.65%	6.43	95.14%	9.75	91.72%	13.71
3-bit MS	750	93.30%	4.53	94.32%	8.70	90.42%	12.41	91.02%	15.60
3-bit SS	496	93.73%	6.77	93.72%	9.75	95.04%	16.00	92.50%	19.00
3-bit MSB	181	94.43%	4.70	95.12%	8.73	93.91%	13.11	91.60%	18.02
Targeted Yield=95%									
1-bit	56	97.96%	5.00	97.29%	9.75	95.87%	11.60	96.70%	16.00
3-bit MS	750	97.57%	6.80	95.94%	8.70	97.03%	14.13	97.52%	20.83
3-bit SS	496	95.43%	7.50	95.60%	14.15	95.38%	18.00	95.97%	23.75
3-bit MSB	181	97.26%	6.63	96.27%	9.51	97.34%	14.78	96.36%	19.57
Targeted Yield=99%									
1-bit	56	99.29%	7.50	99.33%	13.00	99.11%	16.00	99.28%	23.75
3-bit MS	750	99.56%	8.70	99.49%	14.13	99.34%	20.83	99.32%	28.80
3-bit SS	496	99.33%	9.75	99.13%	16.00	99.22%	23.75	99.01%	36.00
3-bit MSB	181	99.51%	9.07	99.54%	14.78	99.32%	21.11	99.30%	28.44

TABLE 5.4: Targeted vs. actual yields obtained using the proposed probabilistic open-junction DT technique

Circuit	Area	5% defects		10% defects		15% defects		20% defects	
		Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area	Actual Yield	Relative Area
Targeted Yield=90%									
C17	56	93.52%	3.50	90.93%	5.00	95.82%	9.75	90.07%	13.00
ALU4	2223	94.64%	9.31	92.29%	12.80	91.81%	16.57	92.32%	22.95
MUL3	3315	95.15%	7.59	92.22%	9.76	92.24%	13.59	93.39%	19.71
Targeted Yield=95%									
C17	56	97.90%	5.00	95.23%	7.50	95.73%	9.75	97.19%	16.00
ALU4	2223	97.60%	9.46	97.24%	15.49	95.20%	18.36	95.98%	26.54
MUL3	3315	98.29%	8.12	98.05%	13.10	95.82%	15.37	96.72%	22.06
Targeted Yield=99%									
C17	56	99.45%	7.50	99.46%	13.00	99.37%	16.00	99.28%	23.75
ALU4	2223	99.16%	11.45	99.22%	18.36	99.30%	26.54	99.05%	38.37
MUL3	3315	99.38%	10.16	99.21%	15.37	99.30%	22.06	99.27%	29.88

TABLE 5.5: Targeted vs. actual yields obtained using the proposed probabilistic open-junction DT technique for various benchmark PLAs

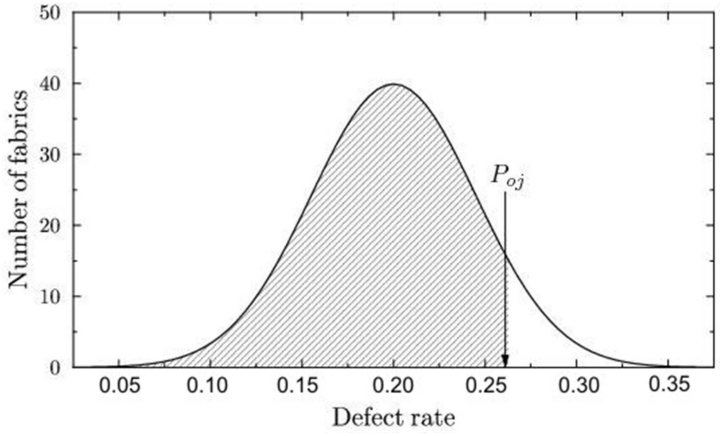


FIGURE 5.14: Illustration of open-junction defect rates distribution across fabricated nanoscale PLAs

achieving a high manufacturing yield Y which is given by the following equation:

$$Y = Y_{wire} \times Y_{cross} \tag{5.13}$$

Fig. 5.14 is an illustration of the distribution of open-junction defect rates in nanoscale fabrics. In this example, although most nanofabrics have a defect rate of around 20%, choosing $P_{oj} = 20\%$ will only allow tolerating defects in 50% of the fabricated fabrics. However, if P_{oj} is set to be higher than 20%, the manufacturing yield will certainly be higher. The shaded area in Fig. 5.14 represents the region where all open-junction defects are successfully tolerated by the proposed technique and the remaining area is the region where the allocated redundancy is not sufficient to tolerate such a high level of defect rates. Setting P_{oj} to cover most of the region in the graph will result in better yields, however, this will require allocating more redundant wires and hence a bigger area overhead. Therefore, the targeted manufacturing yield and the level of redundancy should be both taken into consideration to decide on the values of P_{wire} and P_{oj}.

5.9 Conclusion

A novel probabilistic design flow that improves the manufacturing yield in nanoscale PLAs was presented in this chapter. It comprises of two DT techniques that tackle nanowire and junction defects. Contrary to the previously proposed techniques, this design flow does not need crosspoint-by-crosspoint verification prior to the mapping of logic functions, therefore cutting on post-manufacturing costs. Defect diagnosis is performed at the row/column nanowire level rather than at the crosspoint level. This level of granularity helps reduce test and diagnosis time significantly.

When testing over a suite of benchmark PLAs, injecting defects in a Monte Carlo fash-

ion, the analysis proves that the two proposed techniques always achieve the targeted yields. The defect model adopted in the simulations includes all possible physical defects expected in future nanoscale PLAs including broken nanowires, closed-junction defects and the dominant open-junction defects. Various PLA implementations of logic circuits were taken into consideration. Simulation results demonstrated that the two proposed techniques successfully exhibit the required yields regardless of size, geometry and density of active junctions in the PLAs. It was also shown that the area overhead incurred by the proposed DT techniques depends on the defect rate of the fabric, the targeted yield, the size of the PLA and the density of active junctions in the PLA.

Chapter 6

Coding-based Fault Tolerance for Nano/CMOS Architecture

Chapters 3, 4 and 5 present defect tolerance techniques to address the problem of low manufacturing yield in nanometre CMOS and nanoscale crossbar architectures. In this chapter, three efficient fault tolerance techniques that improve both manufacturing yield and computational reliability of hybrid nano/CMOS architecture are proposed. Nanoscale LUT-based implementation of logic circuits is targeted in this chapter. The efficiency of the proposed techniques is compared with recently reported methods that use single coding schemes to tolerate the high fault rates in nanoscale fabrics. The proposed techniques are based on error correcting codes to tackle different fault rates. In the first technique, a combined two dimensional coding scheme is implemented using Hamming and Bose-Chaudhuri-Hocquenghem (BCH) codes to address fault rates greater than 5%. In the second technique, the level of fault tolerance is further enhanced by combining the coding schemes of the first technique with N-Modular Redundancy to target upto 10% of fault rates. In the third technique, Hamming coding is complemented with Bad Line Exclusion technique to tolerate fault rates upto 20%.

The rest of the chapter is organised as follows: an overview of the targeted hybrid nano/CMOS architecture is presented in Section 6.1, followed by Section 6.2 that outlines the simulation procedure used to calculate the reliability ratios obtained by the proposed techniques. Sections 6.3 and 6.4 provide background on recently proposed error correcting codes and redundancy-based fault tolerance techniques and examine their suitability in dealing with high fault rates. The details of the proposed three techniques are presented in Sections 6.5, 6.6, 6.7 and 6.8 where the simulation results and theoretical predictions are outlined. Section 6.9 estimates the area, energy and operational latency overheads incurred by CMOS components in the hybrid nano/CMOS architecture. Finally, Section 6.10 concludes the chapter.

All the work presented in this chapter is the sole contribution of the author.

6.1 Introduction

Although nanowires that are just few atoms in diameter are now reliably fabricated [Bachtold et al., 2001, Cui and Lieber, 2001, Huang et al., 2001], it is widely recognised that these nanoscale devices will exhibit fault densities much higher than state-of-the-art silicon technology. As mentioned in Chapter 1, Section 1.2.2, the immaturity of the self-assembly techniques that are used to assemble nanoscale devices into regular structures will make them exhibit high defect rates [Brown and Blanton, 2007, Wang and Chakrabarty, 2007, Bahar, 2006, Jacorne et al., 2004, Heath et al., 1998, Copen Goldstein and Budiu, 2001, Bourianoff, 2003]. Moreover, nanodevices are also likely to be much more susceptible to transient faults [Jacorne et al., 2004, Zhao et al., 2005, Nepal et al., 2006]. With feature sizes shrinking to the nanometre scale, clock frequencies reaching the multi GHz level and supply voltages declining to the subvoltage range, computation is becoming more vulnerable to the effects of various noise sources [Zhao et al., 2005, Cohen et al., 1999]. Such high fault rates pose a major challenge in implementing reliable computation of Boolean functions using these emerging nanotechnology devices. Therefore, to achieve acceptable levels of manufacturing yield and computational reliability, fault tolerance must be integrated into the design flow of nanoscale circuits.

A computational architecture that is based on hybrid nano/CMOS design (Chapter 2, Section 2.3) is proposed in this chapter. The idea behind this is to enhance yield and computational reliability by combining the reliable but low performance CMOS components with the high performance but unreliable nanoscale devices. Fig. 6.1 gives an overview of the targeted hybrid nano/CMOS architecture. The high density of nanodevices is exploited to store circuit's Boolean logic implemented as look-up tables, as outlined in Chapter 1, Section 1.1.1, whereas the CMOS components are utilised for global interconnect and highly critical functions. LUT implementation is an effective approach that provides low-level protection of individual Boolean functions [Shanbhag et al., 2008, Ziegler and Stan, 2003]. The LUT is protected by encoding its content using coding techniques that will increase the probability of successfully instantiating the LUTs onto defective nanofabrics and also enhance the circuit's resilience to transient faults. The corresponding decoders are implemented in CMOS and thus impose area and delay overheads compared to the dense nanoscale LUTs. It is worth noting that both CMOS components and high fault rates will reduce the net density offered by nanodevices.

In this chapter, physical and transient faults are integrally addressed i.e. the proposed fault tolerance techniques increase both manufacturing yield and computational reliability. A fault (or error) is caused by either a manufacturing defect or a transient fault. Unlike the static and dynamic reconfiguration techniques, outlined in Chapter 2, Section 2.2, the techniques that are proposed in this chapter do not require test and

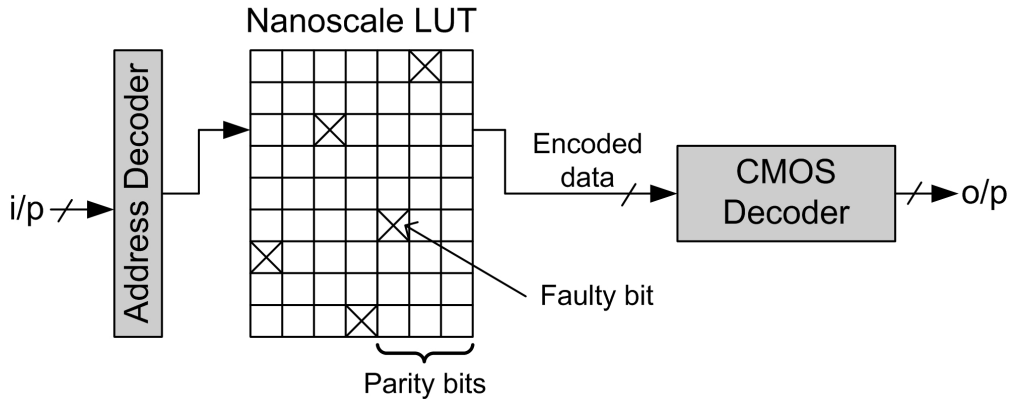


FIGURE 6.1: Targeted hybrid nano/CMOS architecture overview

diagnosis methods to identify the faulty nanodevices. Moreover, to be able to tolerate high fault rates, fault tolerance is addressed at the lowest level of granularity i.e. junction level. The aim of this chapter is to first investigate the suitability of the well-known error correcting codes (ECCs) in tolerating the high permanent and transient fault rates projected in future nano-circuits. Then, new hybrid fault tolerance techniques that are based on various coding schemes and capable of addressing higher fault rates than the preliminary coding techniques are devised. The realisation of high levels of fault tolerance will incur area, energy and operational latency overheads. Such overheads are analysed when investigating and evaluating the proposed techniques.

6.2 Failure Rate Calculation

To demonstrate the improvements in reliability that are achieved by the techniques to be presented in this chapter, experiments are conducted using 5000 randomly generated symmetric LUTs where the probability of 0 and 1 are equal. The LUTs taken into consideration are of sizes ranging from $2^3 \times 3$ (3 inputs, 3 outputs) to $2^6 \times 6$ (6 inputs, 6 outputs). Errors (faults) caused by physical defects such as stuck-open and stuck-short junction defects (Chapter 1, Section 1.2.2) are referred to as physical (hard) errors. Errors caused by transient faults are referred to as transient (soft) errors. Errors are randomly injected into the nanofabric causing the corresponding bits to change their values (i.e. $1 \rightarrow 0$ or $0 \rightarrow 1$). It is also assumed that both physical and transient errors are uniformly distributed across the fabric where both physical and transient errors are random and statistically independent. For comparison purposes the simple Hamming code [Singh et al., 2007] is used as a reference point for the evaluation of the proposed techniques.

To calculate the reliability of a certain ECC technique resulting from injecting m errors into the LUTs, the following simulation procedure is used:

1. Set the number of iterations to be performed, I, to 5000 and the number of successful simulations, K, to 0.
2. Generate a random 0-1 LUT.
3. Encode every row of the LUT using the encoder of the ECC technique.
4. Randomly inject m errors in the encoded LUT.
5. Decode every row of the encoded LUT using the decoder of the ECC technique.
6. If the decoded LUT and the original LUT are the same, increment K by 1.
7. Decrement I by 1 and if I is not 0 goto step 2.
8. $Reliability = K/5000$

The failure rate is calculated as follows:

$$P_{failure} = 1 - Reliability \tag{6.1}$$

6.3 Error Correcting Codes (Related Work)

In semiconductor memories, error correcting codes are mainly reserved for the suppression of transient faults rather than physical defects [Koren and Singh, 1990]. Recently, coding techniques have been proposed as a promising approach to improve the reliability and yield of hybrid nano/CMOS systems. In [Nepal et al., 2006, Jeffery et al., 2004], ECCs were used for the suppression of soft errors rather than physical defects i.e. maintaining the level of fault tolerance rather than enhancing defect tolerance. In [Jeffery et al., 2004], the authors proposed a hybrid fault tolerance technique based on Hamming code and reconfiguration. In [Nepal et al., 2006], an implementation of ECCs based on the theory of Markov random fields (MRF) was proposed to combat transient faults thus increasing computational reliability of hybrid systems. In [Sun and Zhang, 2007], two nanoelectronic memory fault-tolerant system design approaches based on Bose-Chaudhuri-Hocquenghem (BCH) codes were reported. Previously, single ECCs such as Hamming and BCH have been used in the context of reliable memory designs [Jeffery et al., 2004, Strukov and Likharev, 2005a]. In [Strukov and Likharev, 2005a], the authors explored combining error correction codes with various repair techniques to combat the high defect rates in hybrid nano/CMOS fabrics with particular focus on memory architectures. The previous works in [Jeffery et al., 2004, Sun and Zhang, 2007] have only addressed fault tolerance in memory architectures. ECC-based techniques can also be applied to memory-based implementation of logic circuits (i.e. look-up tables) which includes Don't Care Conditions (DCCs). The presence of DCCs in Boolean logic functions presents a strong case to apply these techniques to circuits

implemented as look-up tables on hybrid nano/CMOS fabrics. As will be demonstrated in this chapter, the existence of DCCs can be exploited in this type of architecture since it helps in masking of erroneous bits which is not possible in memory design.

In this chapter, because defect tolerance and transient fault tolerance are integrally addressed, ECCs are initially involved in the repair process of LUTs to be able to instantiate the circuit onto the highly defective nanofabric and then, the adopted code should also combat soft errors during the operation lifetime of the circuit to maintain its computational reliability. The techniques to be proposed in this chapter are based on Hamming and BCH codes. While the exact manufacturing defect rate and transient error rate are not yet pinpointed, it is believed that they will easily exceed 10% [Wang and Chakrabarty, 2007, Bourianoff, 2003]. The authors in [Singh et al., 2007] assume small fault rates (less than 10%) in nanoscale fabrics and small LUT sizes with 50% of LUT entries set as DCCs. To demonstrate the effectiveness of the techniques to be proposed in this chapter, they are compared with the techniques proposed in [Singh et al., 2007, Jeffery et al., 2004] in terms of tolerating higher fault rate scenarios in bigger LUT sizes.

To generate a codeword using an (n, k, t) error-correcting technique, the k information bits are encoded producing m parity bits giving a codeword length $n = k + m$. The maximum number of error bits that can be corrected is given by $t = (d_{min} - 1)/2$ where d_{min} is the Hamming distance of the codewords. The Hamming distance d_{min} is the number of disagreements between two valid codewords of the same code.

6.3.1 Hamming Code

Hamming is a single-error-correcting and double-error-detecting (SED-DEC) code i.e. the code is capable of correcting one error and detecting two errors in a codeword. A typical Hamming code is $(2^m - 1, 2^m - m - 1)$, in other words, for $(2^m - m - 1)$ data bits, m parity bits need to be added for full protection [HAMMING, 1950]. More details on the encoding and decoding algorithms of this code can be found in Section B.2, Appendix B.

Defect tolerance in hybrid nano/CMOS architecture using SEC Hamming code has been addressed in [Singh et al., 2007]. The authors analysed the effectiveness of Hamming code for defect rates in the range of 5-10%. In this section, the effectiveness of Hamming code is examined in the presence of higher error rates (upto 20%). Fig. 6.2 shows the variation of failure rate, calculated using the simulation procedure outlined in Section 6.2, with respect to several percentages of injected error rates and for various LUT sizes ranging from $2^3 \times 3$ to $2^6 \times 6$. As can be observed, for $2^3 \times 3$ LUTs and fault rates upto 2.5%, the Hamming code is capable of detecting and correcting all errors, however, as the percentage of injected errors increases above 5%, the failure rate increases exponentially.

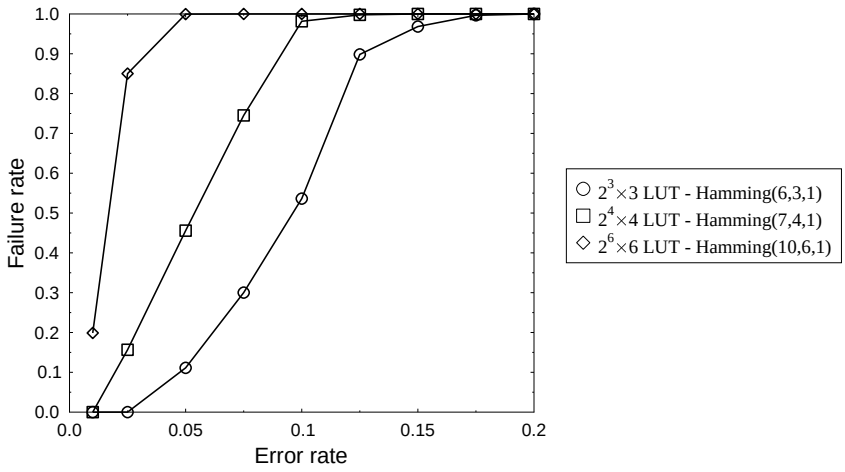


FIGURE 6.2: Hamming - Failure rate vs. Error rate for various LUT sizes

Fig. 6.2 also shows the impact of varying the LUT size on the performance of Hamming code. As can be seen, as the LUT size increases, the failure rate increases indicating the weakness of the Hamming technique in coping with bigger circuits. For instance, for error rates as small as 1%, the Hamming code perfectly detects and corrects all faults for LUTs of sizes smaller than $2^4 \times 4$ as reported in [Singh et al., 2007]. However, it can be seen that even for small $2^3 \times 3$ LUTs and error rate greater than 5%, more than 10% of circuits fail. It can also be observed that as the LUT size increases, the level of fault tolerance falls more rapidly indicating the inefficiency of this scheme.

To validate the simulation results, the mathematical equations that predict the variation of failure rate with respect to the injected fault rate are derived. Theoretically, the probability of a row of length l having m faulty bits is given by the following binomial equation:

$$P(m) = \binom{l}{m} P^m (1-P)^{l-m} \tag{6.2}$$

where P is the error rate of the nanofabric. The probability that the Hamming decoder fails to correctly decode an erroneous codeword is equal to the probability of having more than one error per row. Using equation 6.2, this is given by the following equation:

$$P_{row} = \sum_{k=2}^{r+r_{par}} \binom{r+r_{par}}{k} P^k (1-P)^{r+r_{par}-k} \tag{6.3}$$

where r and r_{par} are the number of bits and number of parity bits in a row respectively. The failure rate of the Hamming code to decode a LUT with c columns is equal to the probability that at least one row is erroneous and it can be computed as:

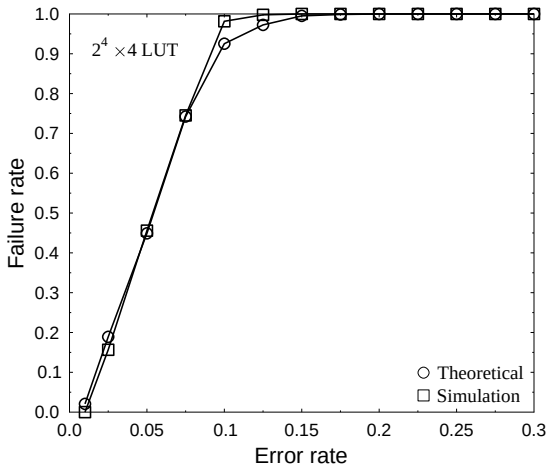


FIGURE 6.3: Hamming - Failure rate obtained theoretically and experimentally

$$P_{failure} = \sum_{k=1}^{c} \binom{c}{k} P_{row}^{k} (1 - P_{row})^{c-k} \tag{6.4}$$

In the case of $2^4 \times 4$ LUTs, equations 6.3 and 6.4 can be rewritten as:

$$P_{row} = \sum_{k=2}^{4+3} \binom{4+3}{k} P^{k} (1 - P)^{4+3-k}$$

$$P_{failure} = \sum_{k=1}^{16} \binom{16}{k} P_{row}^{k} (1 - P_{row})^{16-k}$$

Fig. 6.3 shows the failure rate obtained both theoretically (using equations 6.3 and 6.4) and experimentally based on the simulation procedure outlined in Section 6.2. As can be seen, the two graphs are almost identical, validating the derived theoretical equations.

If an entry in a LUT is a DCC, the output can be either 0 or 1. Next, the impact of the existence of a certain percentage of DCCs in a LUT on the efficiency of the Hamming code is investigated. The same simulation procedure outlined in Section 6.2 is used. However, the simulations were conducted after randomly introducing don't cares into the LUTs. The variations in the failure rate under different percentages of DCCs and error rates are shown in Fig 6.4. As can be observed, the efficiency of the Hamming technique is enhanced as the number of injected don't cares is increased. For instance, there is an improvement of 15% in reliability when the DCC set increases from 0% to 50% of the size of the LUT at a fault rate equal to 5%. However, in the case of higher fault rates, the amount of don't cares has little or no effect on the failure rate.

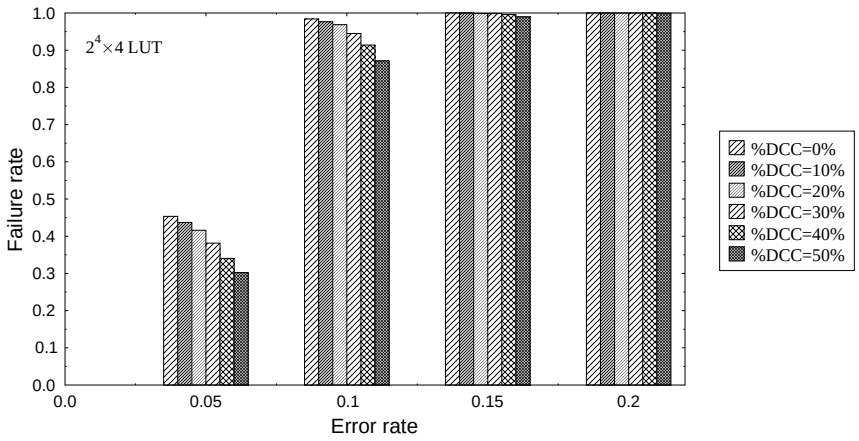


FIGURE 6.4: Hamming - Effect of don't cares on failure rate

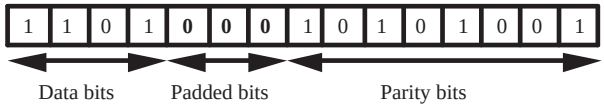


FIGURE 6.5: BCH - Padding

6.3.2 Bose-Chaudhuri-Hocquenghem (BCH) Code

In this section, fault tolerance techniques that are based on BCH coding are examined and their efficiency in dealing with error rates higher than Hamming code is evaluated. BCH is a multilevel and variable length ECC used to correct multiple random errors in a codeword [Biswas et al., 2007]. A t-error-correcting (n, k, t) BCH code adds $(n - k)$ parity bits to the information word to correct t errors. More details on the encoding and decoding algorithms of this code can be found in Section B.3, Appendix B.

Padding

Figs. 6.2 clearly demonstrates that single error correcting codes are not capable of dealing with fault rates higher than 5% in the case of large LUTs. This section presents a fault tolerance technique that uses a stronger BCH code and evaluates its efficiency in dealing with higher fault rates. The proposed BCH code is BCH(15,7,2) which adds 8 parity bits in order to detect and correct 2 errors in the codeword. The required word length is 7 bits, whereas the size of each entry in the LUT is only 4-bits in the case of $2^4 \times 4$ LUTs. Therefore, data bits need to be padded with the necessary bits so that it is equal to the required data word size, as shown in Fig. 6.5.

The second graph (marked $\diamond$) in Fig. 6.6 represents the failure rate obtained by BCH(15,7,2) coding technique. As can be seen, the level of fault tolerance obtained by this technique is very similar to single error correcting Hamming code. BCH(15,7,2) performs slightly

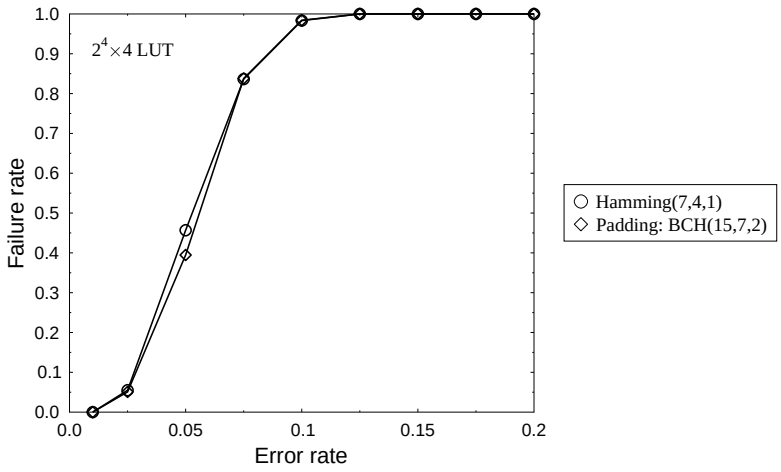


FIGURE 6.6: BCH vs. Hamming

better at relatively small error rates with a small improvement at an error rate of 5% over the Hamming technique. The reason behind this is that even though the BCH(15,7,2) code can tolerate more errors, applying this technique increases the number of errors in the LUTs. The padded bits and the redundant bits added to the data word doubles the number of errors in each entry of the LUT. The codeword is 15-bit long when using the 2-bit error-correction BCH which is twice longer than the 7-bit long codeword for the single error correcting Hamming code. Hence, strong ECCs have the capability of tolerating more errors but at the cost of adding more parity bits to the codeword, which in turn makes them vulnerable to higher error numbers and thus a rapid drop in their efficiency especially as the error rate increases.

Column Coding - BCH

Instead of coding row entries in LUTs, stronger BCH codes are used to encode columns. BCH(31,16,3) for example can detect and correct 3 errors per column, but at the cost of adding 15 parity bits. The results obtained from the simulations are shown in Fig 6.7. For low error rates, BCH exhibits a better performance than Hamming. For example, at an error rate of 5%, there is a 70% improvement in failure rate over Hamming. However, when the error rate exceeds 10%, this coding scheme completely fails. Therefore, it can be concluded that applying error correction techniques to tolerate the high fault rates in future nanoscale fabrics will result in a significant decrease in circuit's reliability.

6.4 N-Modular Redundancy (Related Work)

Before introducing the new hybrid FT techniques, the limitations of a redundancy-based FT technique that has recently been proposed in [Nikolic et al., 2002, Thaker et al.,

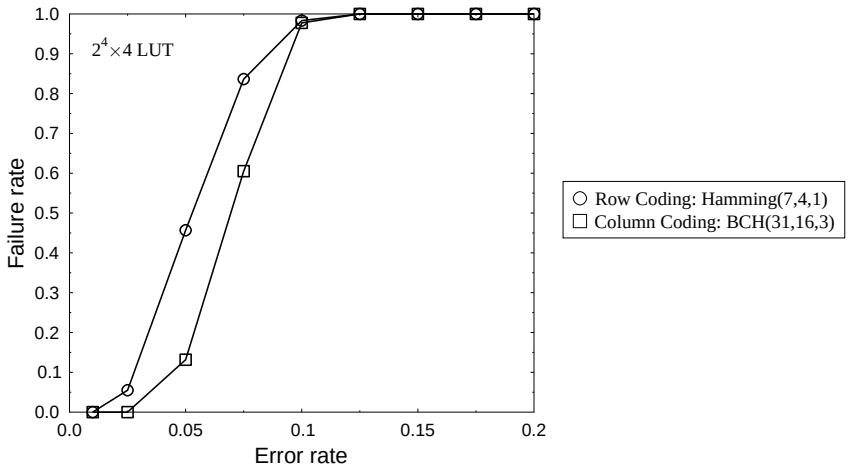


FIGURE 6.7: Row Coding Hamming vs. Column Coding BCH

2005] is outlined. N-Modular Redundancy (NMR) methodology requires replicating N-copies of the same LUT where N is an odd number. The high bit density offered by nanoscale devices is exploited to allow replicating LUTs at a reduced area cost. The m-bit outputs from the same entry in all N LUTs are compared by an N-bit majority voter. Triple Modular Redundancy (TMR), shown in Fig. 6.8, is an instance of NMR which requires the triplication of LUTs (Chapter 3, Section 3.4.1). The overall reliability of this methodology is limited by that of the voter marked V in Fig. 6.8. Therefore, enhancing the reliability of the arbiter by using a fault-free CMOS majority voter is compulsory to achieve a highly-reliable system.

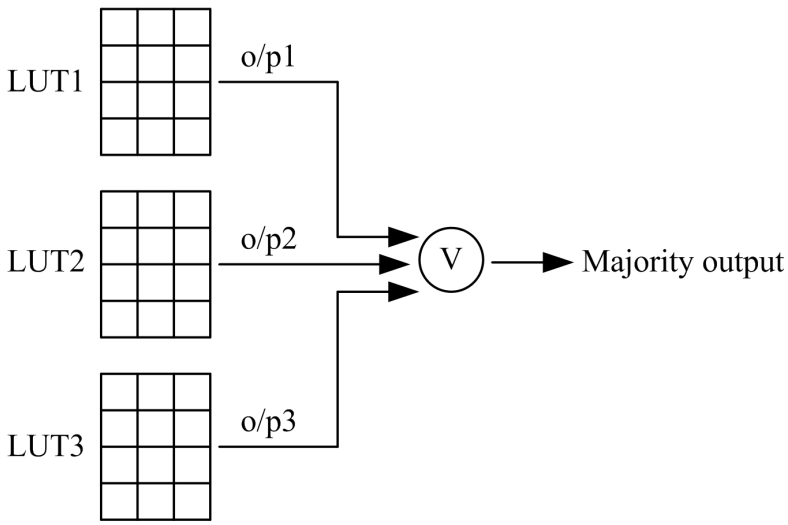


FIGURE 6.8: Triple Modular Redundancy (TMR)

The efficiency of TMR in terms of tolerating the high error rates projected in future

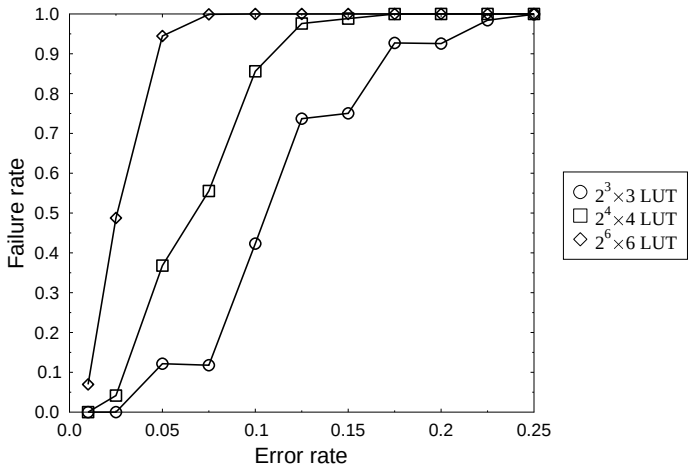


FIGURE 6.9: TMR - Failure rate vs. Error rate for different LUT sizes

nanoscale LUTs is investigated. The failure rates obtained after applying the TMR technique to different LUT sizes are shown in Fig. 6.9. As can be seen, the failure rate increases proportionately with the size of the LUTs. TMR exhibits adequate tolerance against fault rates lower than 5% for small LUTs. However, this technique completely fails to detect and correct fault rates higher than 5% in $2^6 \times 6$ LUTs. It can also be observed that as the percentage of injected faults is increased, the failure rate increases more rapidly in the case of bigger LUT sizes. Comparing the graphs in Figs. 6.2 and 6.9 indicates that TMR has slightly better performance than the Hamming code. For instance, for LUT size of $2^6 \times 6$ and error rate of 1%, the Hamming technique achieves a failure rate of 20%, however, TMR achieves a failure rate of only 7%. And even for the smallest $2^3 \times 3$ LUTs, TMR outperforms Hamming in terms of reliability. The Hamming code completely fails at an error rate of 15%, whereas TMR is still capable of tolerating 27% of the total number of LUTs used in the simulations. Despite the improvement in reliability that is exhibited by TMR, both TMR and the single error correcting code techniques, outlined in Sections 6.3.1 and 6.3.2, are considered inefficient in achieving acceptable levels of reliability. Therefore, new hybrid techniques that are capable of tolerating higher fault rates need to be devised.

6.5 Proposed Hybrid Techniques

So far, two different categories of fault tolerance techniques that integrally address both defect tolerance and transient fault tolerance for hybrid nano/CMOS architecture have been evaluated. The first technique is based on error correcting codes (Hamming and BCH) and the second technique is based on redundancy (NMR). Both techniques exhibit adequate reliabilities in the presence of small fault densities and small LUT sizes.

Hybrid Techniques	Combined Techniques	Targeted Fault Rate
2D Coding (Section 6.6)	Hamming Systematic BCH	7.5%
NMR-ECC (Section 6.7)	Hamming/BCH N-Modular Redundancy	10%
Bad Line Exclusion-ECC (Section 6.8)	Hamming Bad Line Exclusion	20%

TABLE 6.1: Summary of the proposed techniques

Moreover, the efficiency of these techniques quickly degrades in the presence of higher fault rates.

Due to the high defect and transient fault rates and their possible large temporal and spatial variations, different physical memory portions may have largely different number of faulty cells and hence demand different error correction capabilities [Sun and Zhang, 2007]. Therefore, instead of using single error correction schemes, hybrid techniques that combine two different coding schemes or strong ECCs with redundancy based techniques are devised to address higher fault rates in nanoscale LUTs. Table 6.1 summarises the hybrid techniques to be proposed in the following sections.

6.6 Combined Two-Dimensional Coding: Technique 1

6.6.1 Hamming - BCH

To reduce the failure rate at fault rates exceeding 5%, a two-dimensional coding technique that is based on Hamming and BCH is implemented (also known as product code [Mizuochi et al., 2004, Fu and Ampadu, 2008]). The reason behind combining the two coding schemes together is to enhance the error correcting capability of both techniques by applying both coding schemes together. The idea is to encode both rows and columns in LUTs as shown in Fig. 6.10(a). In [Fu and Ampadu, 2008], the authors proposed a Hamming-based two-dimensional coding scheme to tolerate the occurrence of random and burst errors in on-chip interconnects. The single error correction Hamming code is used for both row and column encoding. In the two-dimensional coding technique proposed in this section, the SEC Hamming code is used to encode data bits in each row of the LUT, and then a stronger BCH code is used to encode each column. The choice of BCH for column encoding is due to its ability to tolerate more errors in a codeword and given the size of columns which is 2^N, this choice seems appropriate. In the case of $2^4 \times 4$ LUT for instance, BCH(31,16,3) is used to encode columns. BCH(31,16,3) can detect and correct up to 3 errors per column at the cost of adding 15 parity bits.

Retrieving data from the encoded LUT comprises of two decoding steps. In the first step, columns are first decoded using the BCH decoder. This step will allow the detection and correction of the biggest portion of errors because of the capability of the BCH decoder

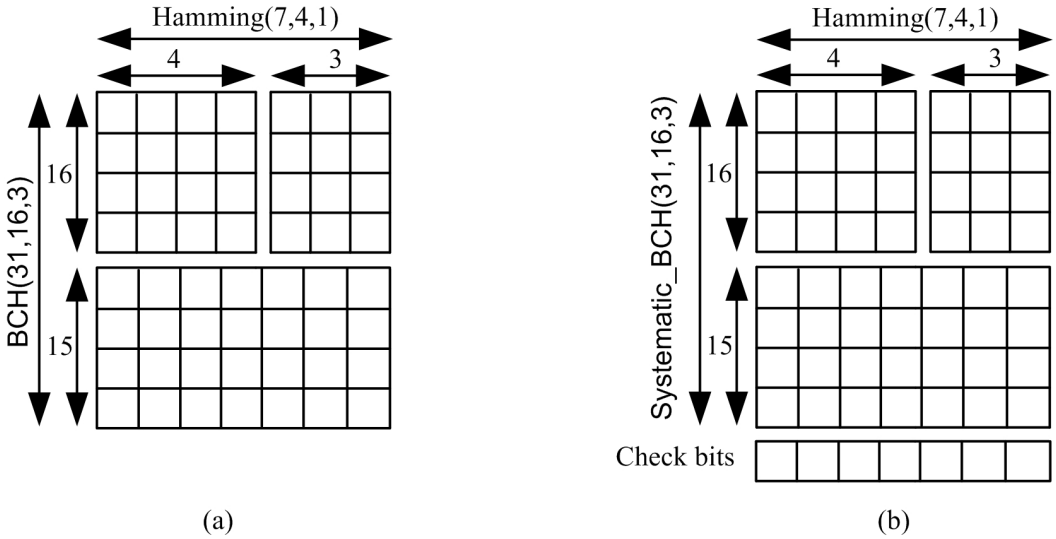


FIGURE 6.10: 2D Coding (a) Hamming & BCH (b) Hamming & Systematic BCH with check bits - $2^4 \times 4$ LUT

to correct more errors in the codeword than the Hamming decoder. Then, in the second decoding step, the Hamming decoder is used to remove the remaining faults.

The failure rates obtained by Hamming(7,4,1), BCH(31,16,3) and 2D coding techniques are plotted in Fig. 6.11. As can be seen, for error rates smaller than 15%, the proposed 2D coding technique (without check bits) exhibits better fault tolerance than both Hamming and BCH. As an example, when the percentage of injected errors is 5%, 2D coding perfectly detects and corrects all the injected faults, whereas Hamming code achieves a failure rate of approximately 45%. However, as the error rate increases beyond 15%, this technique completely fails. It is worth noting here that this improvement in reliability is achieved at the cost of a higher number of parity bits which will result in additional area and energy overhead which will be discussed in Section 6.9.

6.6.2 Hamming - Systematic BCH with Check Bits

The fault tolerance capability of the proposed 2D coding technique can be further improved by using systematic BCH along with check bits as illustrated in Fig.6.10(b). In systematic block codes, data bits remain unchanged in the codeword and the parity bits are attached to the end of the data bit sequence [Naeimi and DeHon, 2007]. In this technique, the fact that the number of 1's in any wrong decoded word will most probably be different from the number of 1's in the expected correct word is exploited. Check bits are used to store the number of 1's of each column after all row entries of LUT are encoded using Hamming and before columns are encoded using the systematic BCH code. In the case of $2^4 \times 4$ LUT for instance, systematic BCH(31,16,3) is used for column encoding. Systematic BCH(31,16,3) is capable of correcting 3 errors so if the number of faults per column exceeds the error correction capability limit of systematic BCH, the

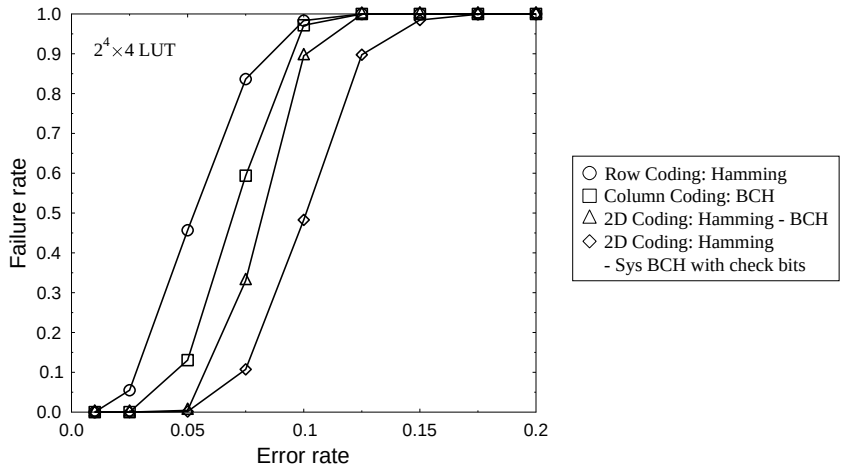


FIGURE 6.11: Failure rate comparison between 1D and 2D coding techniques

BCH decoder will generate the wrong output and hence cause the entire technique to fail. Therefore, to avoid failure, the check bits are always compared with the number of 1's of the output of the BCH decoder, if they are not equal, the codeword remains unchanged and all the errors in the first 16 bits of the corresponding column are left to the second iteration of decoding to be corrected by the Hamming decoder. The flow chart in Fig. 6.12 illustrates the decoding process to retrieve row M from a LUT of size $2^N \times N$.

Fig. 6.11 also shows the enhancement achieved in fault tolerance by incorporating the check bits into the 2D coding technique. 2D coding with check bits achieves significantly lower failure rates for error rates greater than 5% and upto 10% as compared to the basic 2D coding technique. At an error rate of 10%, 2D coding with check bits achieves a failure rate of 49%, whereas the former technique achieves a failure rate of 78% resulting in an improvement of 37% in fault tolerance.

It is intuitively justifiable that, by counting the number of 1's before encoding and checking the number of 1's after decoding, this will improve the fault tolerance efficiency of the proposed 2D coding technique. However, in support of this approach, the number of 1's needs to be stored in a highly-reliable memory (i.e. store at most approximately $\lceil \log_2(NUM_{rows}) \times NUM_{columns} \rceil$ bits in a CMOS memory) and this will incur an extra area and delay overheads besides the Hamming and BCH decoders that need to be evaluated based on practical IC designs as will be explained in Section 6.9.

Next the effect of varying the LUT size on circuit failure probability is examined. As can be seen from Fig. 6.13, the failure rate increases rapidly in bigger LUTs. For an error density of 10%, the failure probability of successfully detecting and correcting errors in faulty LUTs increases from 5% in the case of $2^3 \times 3$ LUTs to complete failure for $2^6 \times 6$

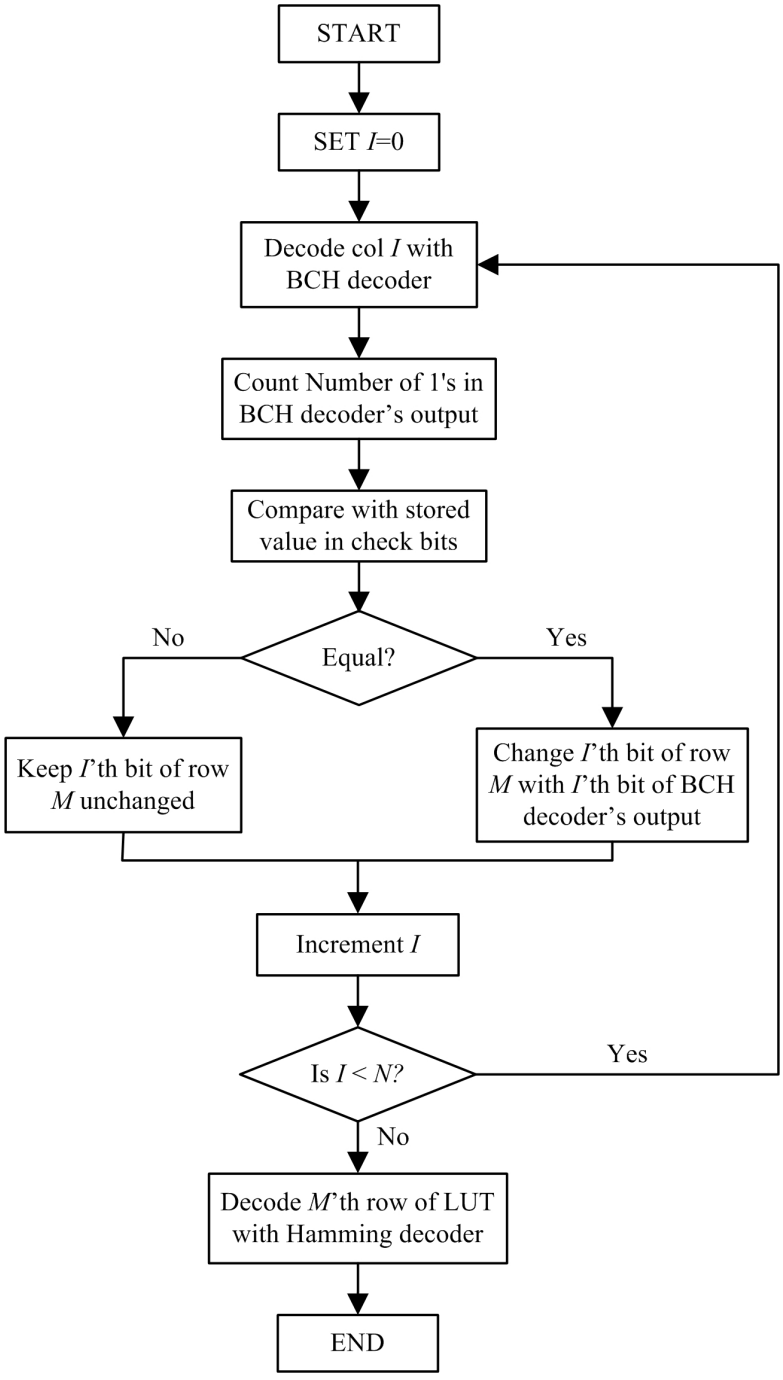


FIGURE 6.12: Flow chart to illustrate the multiple-step decoding process of the proposed 2D coding technique

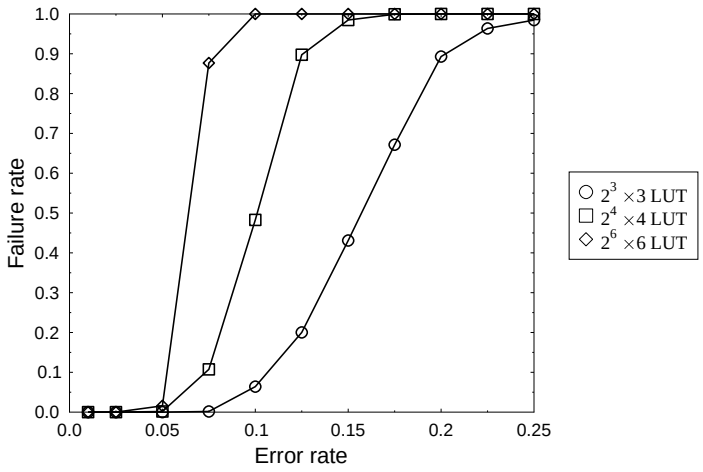


FIGURE 6.13: 2D Coding - Effect of varying LUT size on failure rate

LUTs. Comparing the graphs in Figs. 6.13 and 6.2, it can be observed that the combined 2D coding technique outperforms single dimensional coding in terms of fault tolerance despite its insufficiency in coping with error rates higher than 10% and bigger LUT sizes. In the case of $2^6 \times 6$ LUTs, for instance, the proposed 2D coding technique achieves a failure rate of nearly 0% at 5% error rate whereas the Hamming code completely fails in Fig. 6.2.

To validate the simulation results, the mathematical equations that predict the performance of the proposed 2D coding technique with respect to the injected fault rate are developed. Theoretically, assuming the same error probability P for each bit in the LUT, the probability of having m defective bits in a column of length $(c + c_{par})$ follows the binomial distribution given in equation 6.2. Therefore, the probability that the BCH decoder fails to correct a column because the number of faults exceeds its correction capability bch_err is given by:

$$P_{col} = \sum_{k=bch\_err+1}^{c+c_{par}} \binom{c + c_{par}}{k} P^k (1-P)^{c+c_{par}-k} \tag{6.5}$$

where c and c_{par} are the number of bits and parity bits in a column respectively.

The BCH correction of columns reduces the probability of a bit being erroneous by a factor of P_{col}. Therefore, after BCH correction of columns, the remaining faults which are randomly distributed over the rows will have a new error probability P_{new} which is given by the following equation:

$$P_{new} = P \times P_{col} \tag{6.6}$$

Using this new error probability, the failure rate for each row after Hamming decoding is obtained using equation 6.3, as follows:

$$P_{row} = \sum_{k=2}^{r+r_{par}} \binom{r + r_{par}}{k} P_{new}^{k} (1 - P_{new})^{r+r_{par}-k} \tag{6.7}$$

Hence, the final failure probability of the proposed combined 2D coding technique is given by the following equation:

$$P_{failure} = 1 - (1 - P_{row})^{r} \tag{6.8}$$

For the example used in Fig. 6.10, equations 6.5, 6.7 and 6.8 are rewritten as follows:

$$P_{col} = \sum_{k=3+1}^{16+15} \binom{16 + 15}{k} P^{k} (1 - P)^{16+15-k}$$

$$P_{row} = \sum_{k=2}^{4+3} \binom{4 + 3}{k} P_{new}^{k} (1 - P_{new})^{4+3-k}$$

$$P_{failure} = 1 - (1 - P_{row})^{16}$$

Fig. 6.14 shows the failure rate obtained both theoretically, using equation. 6.8, and experimentally based on simulations in the case of $2^4 \times 4$ LUT. As can be seen, there is an excellent correlation between experimental and theoretical results which validates the derived equations.

Next, the impact of DCCs on the 2D coding technique is examined. Fig. 6.15 shows the results obtained before and after injecting 50% of don't cares into $2^4 \times 4$ LUTs. As can be observed, the optimum improvement is recorded at 10% error rate where the failure rate is reduced from nearly 50% to 37%. In order to theoretically calculate the circuit failure rate, given a certain percentage of DCCs in the LUT, the failure probability of the BCH decoder and the new error rate after decoding which are given by equations 6.5 and 6.6 need to be calculated. After column decoding, the probability that the Hamming decoder fails to correctly detect and correct all errors does not only depend on the number of faults per each row, but also on the number of erroneous bits in the output of the decoder. Therefore, the probability that a given number of bits are erroneous in the output of the decoder, denoted as $P_{(n)bit}$, assuming a number of errors in the codeword has to be estimated where n is smaller or equal to the number of bits of the decoded word. For a $2^4 \times 4$ LUT, the values of $P_{(n)bit}$ are shown in Table 6.2.

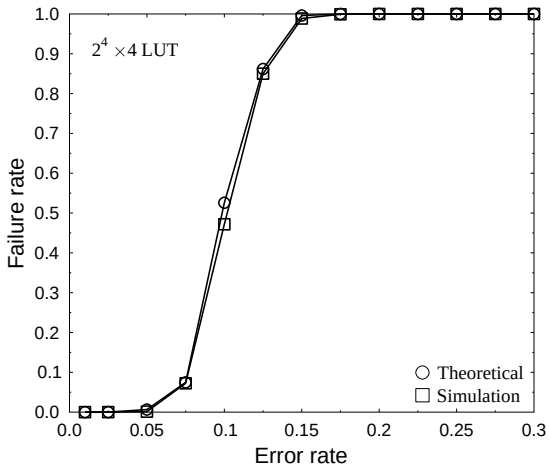


FIGURE 6.14: 2D Coding - Failure rate obtained both theoretically and experimentally

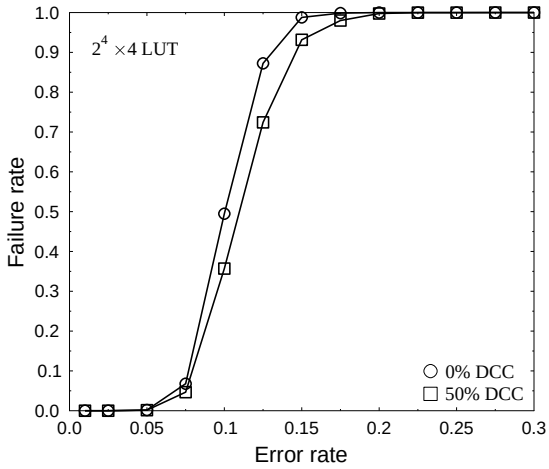


FIGURE 6.15: 2D Coding - Effect of Don't Cares on Failure rate

The failure rate of correctly decoding a row using the Hamming decoder is obtained using the following equation:

$$P'_{row} = \sum_{k=2}^{r+r_{par}} \left[\binom{r+r_{par}}{k} P_{new}^{k} (1-P_{new})^{r+r_{par}-k} \times \sum_{n=1}^{r} \left[P_{(n)bit} \sum_{m=1}^{n} \binom{n}{m} (1-P_{DCC})^{m} P_{DCC}^{n-m} \right] \right] \tag{6.9}$$

The total failure probability is computed as follows:

$$P_{failure} = 1 - (1 - P'_{row})^{r} \tag{6.10}$$

$P_{(n)bit}$	number of errors in a codeword					
	2	3	4	5	6	7
P_{1bit}	0.4306	0.1952	0.3639	0.1415	0	0
P_{2bit}	0.4287	0.4274	0.4335	0.4271	0	0
P_{3bit}	0.1407	0.3774	0.2026	0.4314	0	0
P_{4bit}	0	0	0	0	1	1

TABLE 6.2: Hamming decoder - Distribution of erroneous bits in the output word - $2^4 \times 4$ LUT

Fig. 6.16 presents the failure rate obtained both theoretically, based on the derived equations, and experimentally based on simulations in the presence of 50% of DCCs. As can be seen, the two graphs perfectly match each other which validates the derived theoretical equations.

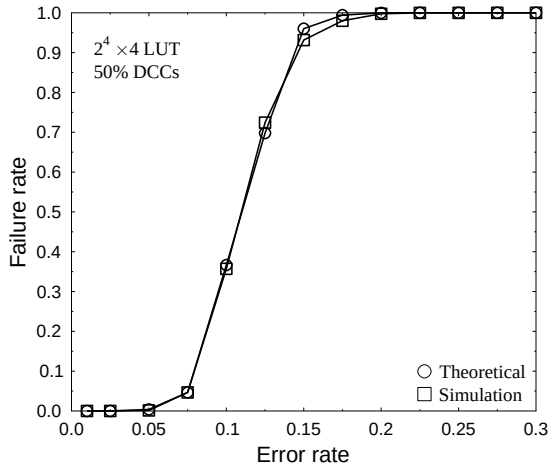


FIGURE 6.16: 2D Coding - Failure rate obtained theoretically and experimentally in the presence of 50% DCCs

6.7 N-Modular Redundancy with ECC: Technique 2

The fault tolerance techniques investigated so far exhibit excellent reliability values against small percentages of errors in the case of small LUTs. However, these techniques fail to maintain their level of fault tolerance in the presence of massive numbers of errors and for bigger LUT sizes. For fault rates higher than 15%, the efficiency of these techniques significantly degrades to unacceptable levels (see Fig. 6.11). Therefore, it was concluded that fault tolerance techniques that are solely based on coding schemes are incapable of tolerating higher error rates. To be able to tolerate higher error rates, coding schemes need to be complemented with other fault tolerance techniques. N-Modular Redundancy is examined in this section.

NMR-ECC hybrid technique comprises of two phases: in the first phase, NMR voter reduces the number of errors in the codeword as illustrated in Fig. 6.18(c). Then in the second phase, the remaining errors in the codeword are corrected by the ECC decoder. Theoretically, the probability of one bit being faulty P_{bit} after NMR correction and before the decoding phase is given by the following equation:

$$P_{bit} = \sum_{k=N/2+0.5}^{N} \binom{N}{k} P^k (1-P)^{N-k} \tag{6.11}$$

where P is the probability of a bit being faulty and N is the number of replications of the original LUT (in case of TMR technique, $N = 3$). After NMR correction, the remaining errors in the codeword will have an error probability of P_{bit} given by equation 6.11. Therefore, given the error correction capability of the coding scheme ECC_err used in the proposed hybrid technique, the probability that the output of the decoder is wrong is equal to the probability of having more than ECC_err faulty bits in the codeword which is given by equation 6.12 below.

$$P_{cod} = \sum_{k=ECC\_err+1}^{c+c_{par}} \binom{c+c_{par}}{k} P_{bit}^k (1-P_{bit})^{c+c_{par}-k} \tag{6.12}$$

Finally, the failure rate of NMR-ECC is calculated as follows:

$$P_{failure} = 1 - (1 - P_{cod})^r \tag{6.13}$$

where r is the number of rows in the LUT.

For the hybrid TMR-Hamming technique to be presented in Section 6.7.1 and LUTs of size $2^4 \times 4$, equations 6.11, 6.12 and 6.13 are rewritten as follows:

$$P_{bit} = \sum_{k=3/2+0.5}^{3} \binom{3}{k} P^k (1-P)^{3-k}$$

$$P_{cod} = \sum_{k=2}^{4+3} \binom{4+3}{k} P_{bit}^k (1-P_{bit})^{4+3-k}$$

$$P_{failure} = 1 - (1 - P_{cod})^{16}$$

The failure rates obtained theoretically using equations 6.11, 6.12 and 6.13 and experimentally based on simulations are shown in Fig. 6.17. As can be seen, the two graphs are nearly identical which validates the derived mathematical equations.

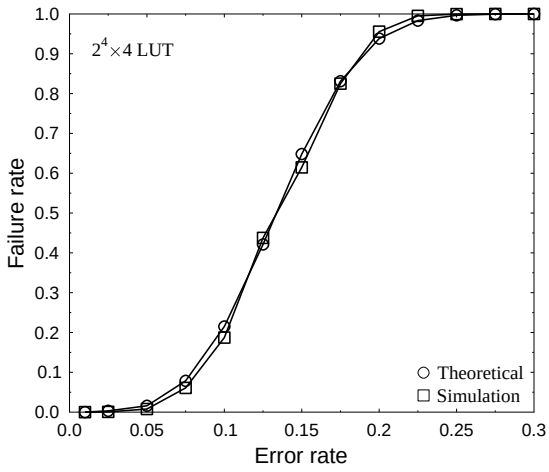


FIGURE 6.17: TMR & Hamming - Failure rate obtained theoretically and experimentally

6.7.1 NMR with Hamming

In this section, the improvement that can be achieved by combining the single error correcting Hamming code with N-Modular Redundancy technique is investigated. Fig. 6.18(b) illustrates an example where the Hamming decoder fails to correct a codeword with two errors, however, in the case of hybrid TMR and Hamming technique shown in Fig. 6.18(c), the TMR voter helps reduce the number of errors in the codeword to only 1 error which can be detected and corrected by the hamming decoder.

To demonstrate the effectiveness of NMR in terms of reducing the failure rate, a statistical analysis on the distribution of errors in LUTs for three different techniques: Hamming, TMR-Hamming and QMR-Hamming was performed. Quintuple-Modular-Redundancy (QMR) technique requires replicating the LUT five times on the nanofabric instead of only three copies in TMR. The same simulation procedure outlined in Section 6.2 was used with the same number of iterations. In every iteration, the maximum number of injected errors per row in the randomly-generated $2^4 \times 4$ LUTs was calculated. Table 6.3 shows the results obtained for different error rates. As can be seen, the distribution of injected errors determines the fault tolerance capability of the Hamming code. As an example, for a 10% fault rate, only 1.92% of the LUTs have a maximum of 1 error-bit per row which can be perfectly corrected by the Hamming decoder, whereas the remaining 98.08% of LUTs have more than 1 error-bit per row which causes the failure of the decoder to give the correct output. And as the number of the uniformly distributed errors increases, the maximum number of errors per row increases and hence an increase in the failure rate because the Hamming code is only capable of correcting a maximum of one error per row. However, for the TMR-Hamming hybrid technique, the distribution of errors is improved. Most of the errors to be detected and corrected by

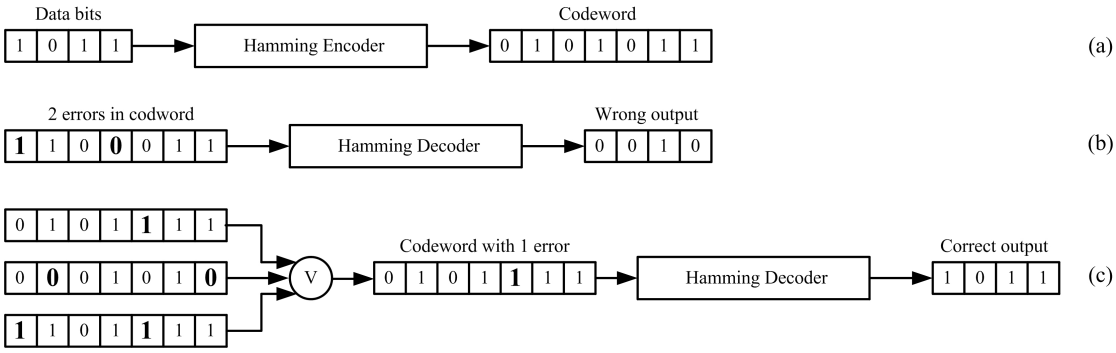


FIGURE 6.18: Combining TMR with Hamming helps reduce the number of errors per codeword before decoding

the Hamming decoder are within the tolerance capability of the Hamming code. This improvement is due to the TMR technique which reduces the number of errors per codeword and then the remaining errors are detected and corrected by the decoder. This can be clearly observed by the significant improvement in the failure rate with respect to the Hamming code at an error rate of 10% where the failure rate is reduced from 98.08% to only 19.26%. However, as the error rate is increased beyond 15%, the TMR voter becomes insufficient to tolerate such high number of errors. The third category in Table 6.3 shows a further improvement in the number of errors to be corrected by the Hamming decoder exhibited by the QMR technique. As can be observed, the proposed hybrid QMR-Hamming technique achieves a failure rate of only 2.08% at a fault rate of 10%. However, this unprecedented performance requires a big area and delay overheads due to the QMR voter and the replication of LUTs on the nanofabric. The QMR arbiter is more complex than TMR arbiter and also represents a reliability bottleneck, thus it should be implemented using highly-reliable CMOS gates. Moreover, the LUT has to be instantiated five times on the nanofabric which causes a further area overhead. Therefore, it can be concluded that to tolerate higher fault rates (i.e. higher that 10%) with a minimum area overhead, a stronger ECC should investigated in combination with TMR. Section 6.7.2 highlights the reliability improvement exhibited by the hybrid TMR-BCH technique.

Next the impact of DCCs on the hybrid TMR-Hamming technique is examined. Fig.6.19 shows the failure rate before and after injecting don't cares into the $2^4 \times 4$ LUTs. As can be observed, the existence of DCCs has little effect on the actual fault tolerance of this technique.

6.7.2 TMR with BCH

To reduce the replication overhead, a hybrid fault tolerance technique that combines TMR and BCH(31,16,3) is proposed. BCH(31,16,3) code which is capable of correcting three errors per codeword is used to encode the 16-bit long columns instead of rows

——	Hamming						
Fault rate	1%	5%	10%	15%	20%	25%	30%
Failure rate	0%	46.58%	98.08%	100%	100%	100%	100%
Fault Distribution							
0	0%	0%	0%	0%	0%	0%	0%
1	100%	53.42%	1.92%	0%	0%	0%	0%
2	0%	44.18%	68.62%	29.8%	3.18%	0.04%	0%
3	0%	2.34%	26.94%	57.7%	58.32%	27.62%	7.44%
4	0%	0%	2.38%	11.46%	33.32%	55.24%	59.82%
5	0%	0%	0.14%	0.01%	4.86%	15.30%	29.00%
6	0%	0%	0%	0.04%	0.32%	1.7%	3.64%
7	0%	0%	0%	0%	0%	0.1%	0.1%
——	TMR & Hamming						
Error rate	1%	5%	10%	15%	20%	25%	30%
Failure rate	0%	1.06%	19.26%	59.44%	96.06%	99.94%	100%
Fault Distribution							
0	97.34%	47.1%	2.24%	0%	0%	0%	0%
1	2.66%	51.84%	78.48%	40.56%	3.94%	0.06%	0%
2	0%	1.06%	18.18%	53.18%	60.7%	22.32%	2.78%
3	0%	0%	1.08%	5.94%	31.14%	59.46%	50.12%
4	0%	0%	0.02%	0.30%	4.08%	16.38%	39.38%
5	0%	0%	0%	0.02%	0.14%	1.72%	7.02%
6	0%	0%	0%	0%	0%	0.06%	0.62%
7	0%	0%	0%	0%	0%	0%	0.08%
——	QMR & Hamming						
Error rate	1%	5%	10%	15%	20%	25%	30%
Failure rate	0%	0%	2.08%	14.44%	60.18%	96.36%	99.9%
Fault Distribution							
0	99.92%	91.76%	38.72%	4.96%	0.04%	0%	0%
1	0.08%	8.24%	59.2%	80.6%	39.78%	3.64%	0.1%
2	0%	0%	2.08%	13.84%	53.2%	60.52%	22.78%
3	0%	0%	0%	0.56%	6.6%	31.32%	58.06%
4	0%	0%	0%	0.04%	0.38%	4.24%	17.24%
5	0%	0%	0%	0%	0%	0.26%	1.7%
6	0%	0%	0%	0%	0%	0.02%	0.12%
7	0%	0%	0%	0%	0%	0%	0%

TABLE 6.3: NMR & ECC - Distribution of errors in $2^4 \times 4$ LUTs

in LUTs. A comparison between the failure rates of the different hybrid NMR-ECC techniques for $2^4 \times 4$ LUTs is shown in Fig. 6.20. At fault rates lower than 15%, hybrid TMR-BCH(31,16,3) is capable of realising a much better fault tolerance than TMR-Hamming(7,4,1). For example, at an error rate of 15%, TMR-BCH achieves a failure rate of 31% which is half of the failure rate obtained by the TMR-Hamming approach which is equal to 60%. The size of the LUT after encoding using BCH(31,16,3) becomes $(2^4 + 15) \times 4$ which is 10% bigger than the size of the LUT after encoding using the Hamming code which is $(4 + 3) \times 2^4$. Such a small area overhead in the size of LUTs can be accepted given the significant improvement in the failure rate.

Fig. 6.20 also compares the efficiency of the proposed hybrid NMR-ECC techniques with the Hamming code under different error rate scenarios. As can be seen, combining

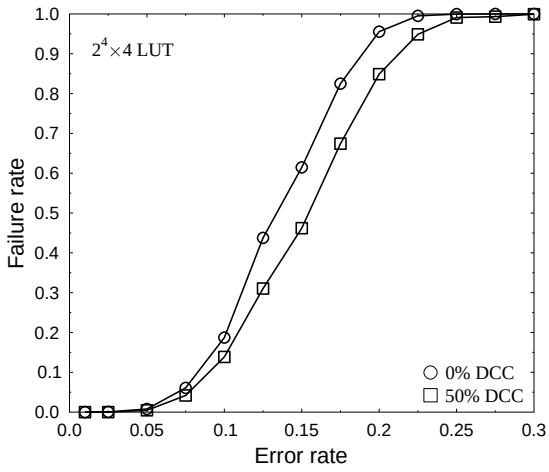


FIGURE 6.19: TMR & Hamming - Impact of don't cares on circuit failure rate

redundancy-based techniques such as TMR or QMR with ECCs helps to significantly improve the reliability obtained by these coding schemes. However, such performance is obtained at the cost of more area and delay overheads. The suitability of a given fault tolerance technique is not only based on the reliability metric but also on other design factors. A trade-off between the required reliability level and the various design metrics can be chosen by the chip designer. The high density of nanodevices can be exploited to create more copies of the original circuit for the NMR technique and also to store the increasing number of parity bits of the stronger coding techniques. The decoder's area and delay overheads need to be measured as stronger ECCs require more CMOS components on the chip (Section 6.9).

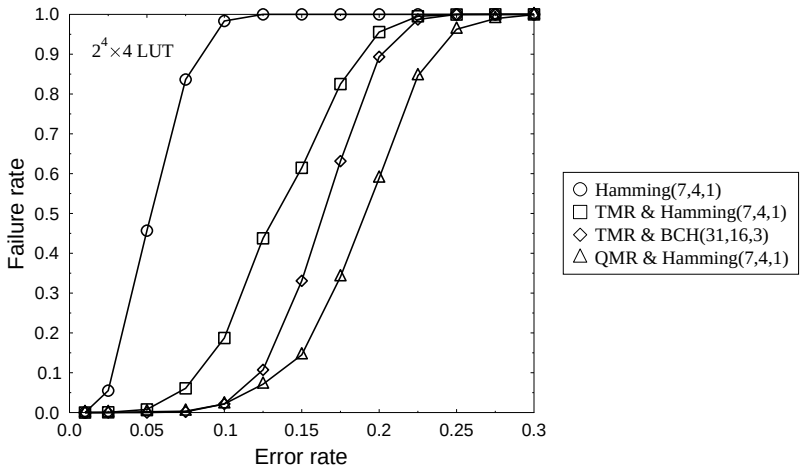


FIGURE 6.20: Comparison between the failure probabilities of the investigated fault tolerant techniques

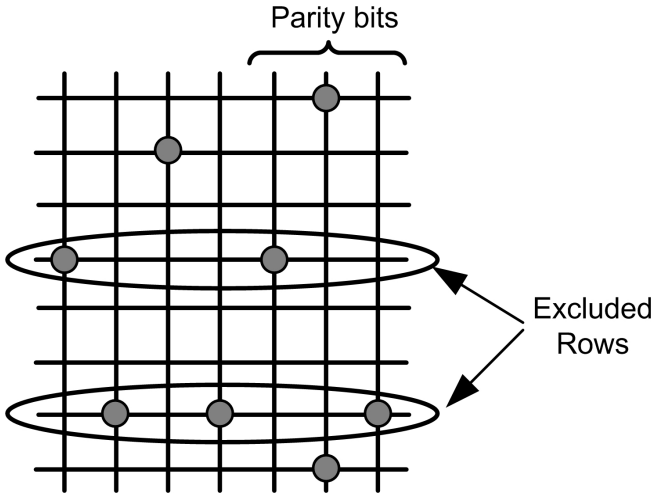


FIGURE 6.21: Hamming with Bad Line Exclusion

6.8 Bad Line Exclusion with ECC: Technique 3

In semiconductor memories, ECCs are usually preserved for the suppression of transient faults to enhance computational reliability [Strukov and Likharev, 2005a]. However, the high defect densities expected in future nano-circuits dictates involving coding techniques at the initial repair of the physical defects to minimise the required amount of redundant resources. As can be seen from Fig. 6.11, ECCs alone are not able to address the issue of both high defect rates induced during manufacturing and transient faults that occur during computation time. Fig. 6.20 also proves that coding techniques in combination with redundancy-based techniques such as NMR are incapable of tolerating error rates higher than 10%. Therefore, it is imperative to use ECCs in conjunction with other techniques in order to detect and correct larger portions of physical defects as well as transient faults (up to 20%).

To deal with higher fault rates, error correcting codes such as Hamming are combined with Bad Line Exclusion technique [Strukov and Likharev, 2005a]. This technique requires allocating enough redundant rows for each LUT to be repaired. The use of redundant wires to tolerate physical defects was presented in [Strukov and Likharev, 2005a, Lehtonen et al., 2007]. As will be shown, the amount of spare rows depends on two main factors: the defect rate of the fabric and the size of the LUT. The proposed Bad Line Exclusion-ECC technique consists of two phases: a must-repair phase and final-repair phase. In the must-repair phase, line exclusion is applied in one dimension only where the defective rows are excluded and replaced with spare ones if the number of physical defects per row exceeds the correction capability of the Hamming code. Therefore, defect counters for the faulty rows are required during the initial testing process of the nanofabric. It is worth mentioning that the configuration process is performed only once during the manufacturing phase (i.e. off-line configuration) and hence the area of the configuration logic is not added to the CMOS area overhead of this technique. In

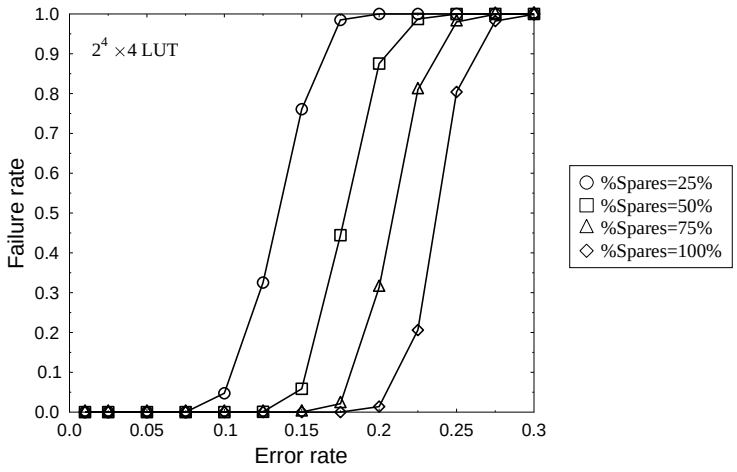


FIGURE 6.22: Hamming & Bad Line Exclusion - Failure rate obtained for different percentages of spare rows

the final-repair phase, the Hamming decoder detects and corrects the remaining errors as illustrated in Fig. 6.21. During the initial analysis of the fabric, the bad rows are detected and their physical address is used to create a special table to map the continuous logical address to the actual physical location of defect-free rows. Such a mapping table has to be stored in a highly-reliable memory implemented in CMOS. The physical implementation of this logical-to-physical mapping table is beyond the scope of this work and is not included in the area overhead estimation of this technique in Section 6.9.

Fig. 6.22 shows the variations in the failure rate exhibited by the Bad Line Exclusion-Hamming technique with respect to the error rate and the percentage of spare rows allocated for repair. As can be seen, this technique is capable of tolerating an unprecedented percentage of errors when compared to 1D coding, 2D coding and hybrid NMR-ECC techniques (see Figs. 6.11 and 6.20). This is demonstrated by a failure rate of nearly 0% for upto 20% of injected errors into the randomly generated $2^4 \times 4$ LUTs. To further the reliability analysis, the impact of varying the number of spare rows on the failure rate is examined for different LUT sizes. Fig. 6.23 demonstrates that as the error rate increases, more spares are needed to keep the level of fault tolerance close to 0%. In case of $2^4 \times 4$ LUTs, for instance, only 25% of spare rows are needed i.e. 4 more rows, to completely tolerate fault rates of upto 10%. And as more errors are injected into the LUTs, more spares should be allocated and hence decreasing the useful bit density of the fabric. It can also be observed that as the LUT size increases, the percentage of spares also increases to achieve low failure rates. As an example, $2^6 \times 6$ LUTs require twice their original size to tolerate 20% error rate. However, such high redundancy can be minimised by adopting a more powerful ECC such as BCH instead of Hamming.

Next, the theoretical equations that predict the reliability of the proposed technique

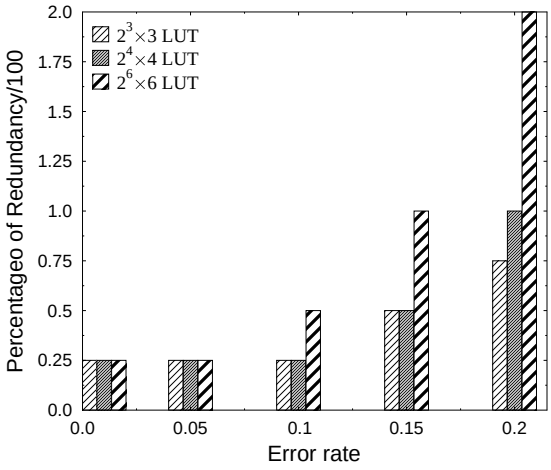


FIGURE 6.23: Percentage of Redundancy needed to achieve 0% failure rate for different LUT sizes

with respect to the injected fault rate and the amount of spare rows r_{sp} are derived. To obtain the probability of failure, the probability P_{row} of a row of length $(c + c_{par})$ being excluded is first calculated. P_{row} is equal to the probability of having more than one bad bit in a row and it is given by the following equation:

$$P_{row} = \sum_{k=2}^{c+c_{par}} \binom{c + c_{par}}{k} P^k (1-P)^{c+c_{par}-k} \tag{6.14}$$

where P is the probability of a bit being defective. Therefore, the probability of failure to instantiate a LUT on the fabric given the original number of rows r and the upper limit of spare rows r_{sp} can be computed as follows:

$$P_{failure} = \sum_{k=r_{sp}+1}^{r+r_{sp}} \binom{r + r_{sp}}{k} P_{row}^k (1-P_{row})^{r+r_{sp}-k} \tag{6.15}$$

For $2^4 \times 4$ LUTs and 25% spare rows, equations 6.14 and 6.15 can be rewritten as follows:

$$P_{row} = \sum_{k=2}^{4+3} \binom{4+3}{k} P^k (1-P)^{4+3-k}$$

$$P_{failure} = \sum_{k=4+1}^{16+4} \binom{16+4}{k} P_{row}^k (1-P_{row})^{16+4-k}$$

While the authors in [Singh et al., 2007] assumed 50% DCCs in their simulations, the

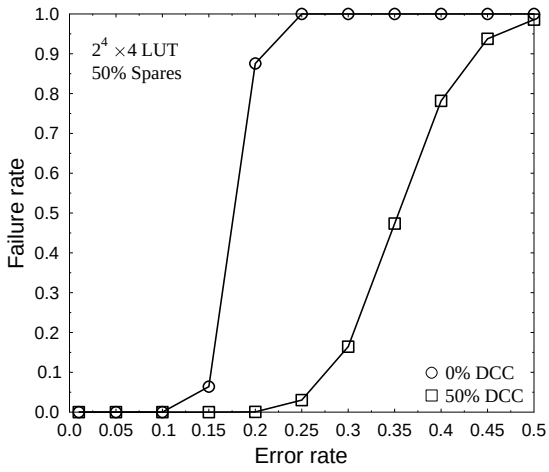


FIGURE 6.24: Hamming & Bad Line Exclusion - Variation of failure rate in the presence of Dont Care entries. Inclusion of 50% DCCs improves fault tolerance by almost two times

technique proposed in this section has exhibited a remarkable improvement in fault tolerance even with 0% DCCs in the LUT implementation. However, it can be seen from Fig. 6.24 that the fault tolerance of this technique is significantly improved when some of the entries in the LUTs are assumed as DCCs as compared to the results shown in Fig. 6.22. The targeted error rate is doubled from 10% to 20% due to DCCs injection into the LUTs.

The existence of DCCs (P_{DCC}) in LUTs significantly reduces the bit failure rate as outlined in the following equation:

$$P' = P \times (1 - P_{DCC}) \tag{6.16}$$

The new probability of a row being excluded after injecting don't cares is obtained by replacing the error rate P with the new error rate P' in equation 6.14 as follows:

$$P'_{row} = \sum_{k=2}^{c+c_{par}} \binom{c + c_{par}}{k} P'^{k} (1 - P')^{c+c_{par}-k} \tag{6.17}$$

Fig. 6.25 shows the failure rate obtained both theoretically using equations 6.17 and 6.15 and experimentally based on simulations. As can be observed, there is an identical match between the two graphs indicating the correctness of the derived mathematical equations.

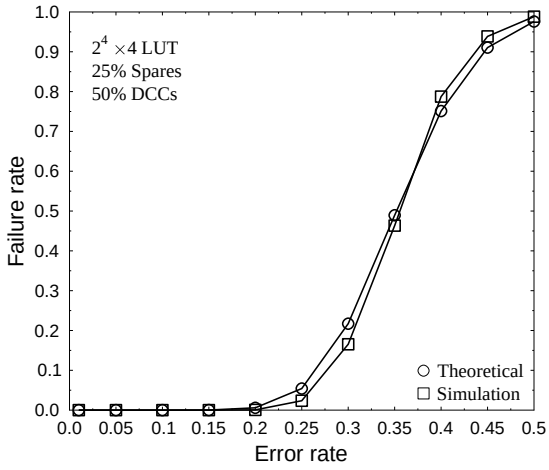


FIGURE 6.25: Hamming & Bad Line Exclusion - Failure rate obtained both theoretically and experimentally in the presence of 25% spares and 50% DCCs

6.9 Overheads of the Proposed Fault Tolerance Techniques

The realisation of fault tolerance in nano/CMOS nanoelectronic architecture will incur area, energy and operational latency overheads in CMOS domain [Bahar et al., 2007]. Such overheads must be taken into account when investigating and evaluating hybrid nano/CMOS fault tolerant architectures. The highly reliable decoders are implemented in CMOS and therefore incur an increase in the area and energy consumption compared to the denser and low energy nanoscale LUTs. Additional clock cycles are also lost in decoding and correcting codewords which causes latency overhead. Coding row and column entries in LUTs requires additional area overhead to store the parity bits.

To obtain an estimate of the area, latency and energy overheads of the proposed fault tolerant techniques, the corresponding decoders were designed in VHDL using the BCH Codec Synthesis tool developed in [Jamro, 1997] (refer to Section B.4, Appendix B for further details regarding this tool) . The VHDL design code of BCH(7,4,1) decoder was used to measure the area, latency and energy overheads of Hamming(7,4,1) decoder because the two coding schemes are equivalent. The decoders were thoroughly tested through simulation using the appropriate test benches. To minimise their silicon area, both Hamming and BCH decoders are serial i.e. they receive 1-bit input and generate a 1-bit output per clock cycle, therefore, the decoding latency is proportional to the codeword length. Table 6.4 presents the area, latency and energy overheads of the Hamming and BCH decoders designed using $0.12\mu m$ CMOS standard cell library. The table also shows the results obtained for the TMR and QMR arbiters. The values were obtained after synthesising the different VHDL codes using Synopsys Design Compiler tool with assumption of a $25MHz$ clock signal. It can be observed that the area overhead

——	Area (μm^2)	Power (μW)		Latency (μs)	Energy (μJ)
		Dynamic Power	Leakage Power		
Hamming(7,4,1) Decoder	905.72	43.12		0.52	22.42
		36.41	6.71		
BCH(31,16,3) Decoder	9131.86	315.54		2.76	870.88
		249.34	66.20		
3-bit TMR Voter	20.17	45.35		0.28	12.67
		45.10	0.25		
5-bit QMR Voter	141.20	78.79		0.28	22.05
		77.70	1.09		

TABLE 6.4: Area, Delay and Energy overheads of CMOS Components Assuming a $0.12\mu m$ CMOS technology, $2^4 \times 4$ LUT and a Clock Frequency of $25MHz$

of the Hamming decoder is $906\mu m^2$ and decoding one 7-bit long codeword requires 13 clock cycles and an energy overhead of approximately $9pJ/MHz$. The BCH decoder incurs higher overheads due to its high complexity: an area overhead of $9132\mu m^2$, a latency of 69 clock cycles and an energy overhead of $286pJ/MHz$. It is worth noting that the decoding circuitry's area overhead can be minimised by using the time multiplexing strategy where one decoder is shared by multiple LUTs as outlined in [Singh et al., 2007] (see Fig. 6.26).

Based on the postlayout results of $0.12\mu m$ CMOS technology, the decoders and NMR voters implementation metrics, including silicon area, computation latency and energy consumption can be projected at future $32nm$ CMOS technology based on a simple scaling rule presented in the International Technology Roadmap for Semiconductors (ITRS) [Sun and Zhang, 2007, ITRS, 2005] where the silicon area will be scaled down by approximately 16, the logic datapath propagation delay will scale down by approximately 10 and the energy consumption will scale down by approximately 7, as shown in Table 6.5.

	Area (μm^2)	Latency (μs)	Energy (μJ)
Hamming(7,4,1) Decoder	56.61	0.05	3.21
BCH(31,16,3) Decoder	570.74	0.28	124.41
3-bit TMR Voter	1.26	0.03	1.81
5-bit QMR Voter	8.83	0.03	3.15

TABLE 6.5: Expected Area, Delay and Energy Overheads of CMOS Components at Future $32nm$ CMOS Technology

Area/useful bit ratio

In all the three proposed techniques in this chapter, the significant area overhead due to the CMOS components as well as the redundant parity bits, spare rows and the replication of the original LUT will reduce the useful bit density offered by nanodevices. Therefore, a design parameter called area per useful bit ratio is used to compare the

efficiency of the various techniques in terms of area overhead. Area per useful bit ratio (a) reflects the area necessary to achieve a certain useful bit capacity and is obtained by dividing the total area of the fabric by the number of useful bits in the LUT.

$$a = \frac{Total\ area\ of\ fabric}{Number\ of\ useful\ bits\ in\ LUT} \tag{6.18}$$

The total area of the fabric comprises of the area of nanodevices and that of CMOS subsystems. A model presented in [DeHon et al., 2005] was adopted to estimate the area of nanoscale memory. Each bank in the memory is composed of a set of crossed nanoscale wires supported by a set of interface microscale wires. For a nano-circuit of 2^n inputs and m outputs, the area can be estimated using the formula presented in [Singh et al., 2007]:

$$A = \left(W_{litho}(n + \log_2 m) + W_{nano}2^n\right) \times \left(W_{litho}n + mW_{nano}2^n\right) \tag{6.19}$$

The main parameters in the model are the number of rows 2^n and columns m. The area of nanoscale memory is dominated by the address lines which are microwires. $W_{litho} = 105nm$ is the wire pitch of the lithographic address wires and $W_{nano} = 10nm$ is the pitch of the nanoscale wires. As an example, for a $2^4 \times 4$ LUT, the area of the LUT before encoding is $0.84\mu m^2$.

	Total Area (μm^2)	Area/Useful bit ($\mu m^2/bit$)
Hamming alone [Singh et al., 2007]	907	14
2D Coding [Section 6.6]	10040	156.88
TMR & Hamming [Section 6.7.1]	929.55	14.52
Hamming & Bad Line Excl. [Section 6.8]	908	14

TABLE 6.6: Area/Useful bit of the proposed techniques

Table 6.6 compares the area/useful bit ratio of the proposed techniques. It is worth noting that in all these techniques, the CMOS components dominate the total area of the fabric as compared to the area of nanoscale LUTs which is estimated to be only 0.84 μm^2 in the case of $2^4 \times 4$ LUTs. While it was proven that Hamming in combination with Bad Line Exclusion achieves much better failure rates as compared to error correcting schemes such as Hamming or BCH, this improvement in failure rate is achieved with little or no increase in area overhead when compared with the simplest correction technique proposed in [Singh et al., 2007]. This is illustrated by an area/useful bit ratio equal to the value achieved by the Hamming code. It can also be observed that the two dimensional coding technique achieves a bit density more than ten times larger than the other techniques. This is due to the significant area overhead of the BCH decoder. TMR in combination with Hamming also achieves a small ratio despite the triplication in the area of the LUT.

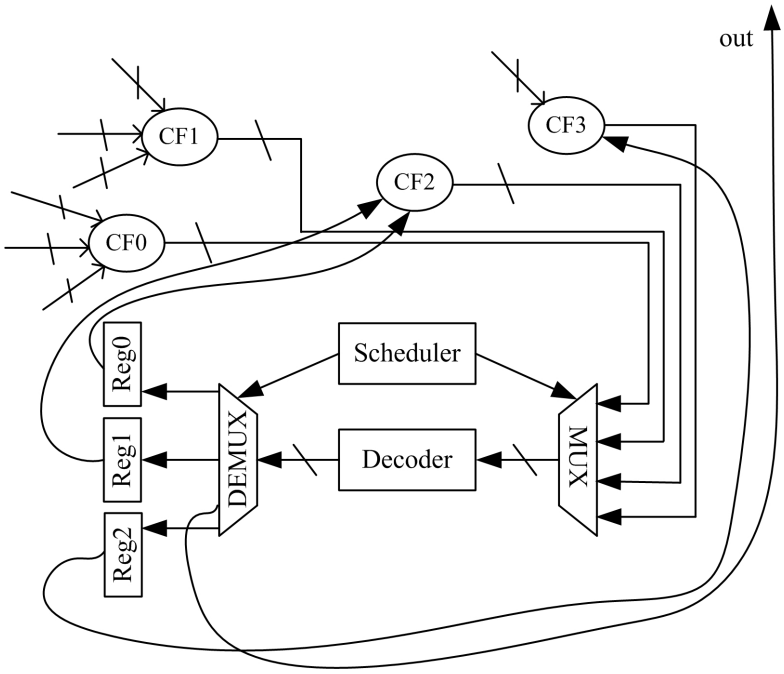


FIGURE 6.26: Time Multiplexing Strategy

While significant area improvement can be achieved over current CMOS for high density fabrics using the hybrid nano/CMOS architecture [Sun and Zhang, 2007], an improvement in the useful bit density can be achieved by sharing the decoders by multiple LUTs using time multiplexing strategy as outlined in [Singh et al., 2007]. This is illustrated by the example shown in Fig. 6.26 where the combinational circuit is divided into four multi-input multi-output logic blocks i.e. LUTs. The appropriate coding technique is applied to each block to produce functions $CF0 - CF3$. The output function is computed by first decoding the output of $CF0$ and storing the output in a register. The output of $CF1$ is also computed and stored in another register. The inputs to $CF2$ are then available, so its output is decoded and stored in a third register. This output is then used to compute the output of $CF3$ which is decoded to produce the final output. Another way to minimise the CMOS area overhead is to synthesise logic circuits into smaller LUTs because the size of the decoder increases proportionally with the size of the LUT. Moreover, as demonstrated in the previous sections, using smaller LUTs allows achieving higher levels of fault tolerance at the cost of low area overhead.

6.10 Conclusion

This chapter addresses fault tolerance in a hybrid nano/CMOS architecture implementing Boolean logic functions as LUTs. The targeted faults include both physical defects and transient faults. The aim is to increase manufacturing yield by increasing the probability of successful instantiation of LUTs on nanofabrics and also to enhance computational reliability against transient faults that occur during circuit's operation lifetime.

Experimental studies proved that single error correcting codes such as Hamming and BCH are insufficient in tolerating high fault rates especially as the LUT size increases. Redundancy-based techniques such as N-Modular Redundancy was also investigated and proved its incapability of tolerating higher fault rates.

Three hybrid fault tolerance techniques were presented in this chapter that are capable of enhancing both yield and computational reliability in the presence of high error rates. In the first technique, both rows and columns of LUTs are encoded using Hamming and BCH codes respectively to target higher number of faults. This technique significantly improves fault tolerance with respect to single error correction schemes for error rates upto 7.5%. Another hybrid technique that was investigated is NMR with ECCs. This technique exhibits better resilience to higher error rates but at the cost of replicating the LUT multiple times. In the third technique, Hamming is complemented with Bad Line Exclusion. This technique results in remarkable improvement of failure rate against a substantial fraction of bad nanodevices (upto 20%). This is achieved at the cost of minimal increase in area overhead compared with Hamming, yet with much higher efficiency in tolerating errors. Based on the conducted studies, this technique is very effective for LUT-based Boolean logic architectures. The impact of LUT size and the number of don't cares on the efficiency of the proposed techniques was also investigated. It was concluded that the presence of DCCs in LUTs can significantly improve the efficiency of the proposed techniques namely the hybrid Hamming-Bad Line Exclusion technique. The theoretical equations that predict the efficiency of the proposed techniques for a given fault rate and LUT size were also derived. Finally, the impact of these techniques in terms of area, latency and energy overheads was investigated and showed that improved fault tolerance can be achieved using the proposed techniques with little overheads as compared to previous coding techniques.

Chapter 7

Conclusions and Future Work

The continuous scaling down of Silicon CMOS beyond the 100nm feature sizes has lead to a remarkable increase in manufacturing defect rates in nanometre CMOS design. In order to enhance the computational efficiency of integrated circuits beyond CMOS capabilities, several technologies have been proposed including chemically assembled nanoscale devices. These devices are expected to achieve extremely high device densities yielding computational fabrics with many billions of components. However, the imprecision and lack of determinism in the self-assembly and self-alignment techniques used in their fabrication are expected to lead to large defect and fault densities that are significantly higher than those in current lithography-based CMOS technology. Such high levels of physical defects and operation-time fault rates pose a significant challenge to current design methodologies and tools. Although there has been numerous research efforts to develop new defect tolerance and avoidance techniques; up-to-now, there are no definite architectures or design methodologies for designing circuits and systems using nanoscale devices.

Various techniques have been proposed to tackle the physical and transient faults in nanometre CMOS and nanoscale devices. One approach is to use classical fault tolerance techniques such as Triple Modular Redundancy, NAND Multiplexing and Quadded Logic techniques. These techniques exploit the large device densities offered by these technologies to tolerate the high fault rates by allocating redundant resources at different levels of design hierarchy. However, they are regarded as inflexible and rigid to the high and time varying fault rates in nanometre CMOS and nanodevice-based circuits. Static and dynamic reconfiguration is another approach that provides a powerful method for enhancing yield and computational reliability by designing around faulty resources in the fabric. However, designing complex systems using reconfiguration poses a major scalability challenge as defect mapping and configuration are performed on a per-chip basis. The overall aim of this research is to devise efficient defect and fault tolerance techniques that are capable of tackling the projected high defect and fault rates in nanometre CMOS and nanoscale device technologies. To meet this aim, the techniques

proposed in this thesis address defect and fault tolerance at the lowest level of abstraction to provide robust tolerance. To overcome the drawbacks of reconfiguration-based techniques, defect mapping and diagnosis are either not required by some of the proposed techniques or limited to higher levels of granularity in other proposed techniques. Section 7.1 presents a summary of the research contributions made by this thesis and Section 7.2 outlines a number of worthy future research directions.

7.1 Thesis Contributions

Production yield is a major problem facing nanometre CMOS due to its high permanent defect rates including stuck-at and bridging defects. To enhance yield, a defect tolerance technique that is based on adding redundancy at the transistor level is presented in Chapter 3. Each transistor in the circuit is replaced by an N^2-transistor structure ($N \geq 2$) that guarantees defect tolerance of all $N-1$ defects. Two particular cases of this technique for $N = 2$ and $N = 3$ are evaluated in terms of tolerating stuck-at and bridging defects. The proposed technique is compared with some previously proposed techniques that add redundancy at the gate-level such as Quadded Logic and TMR techniques. Experiments are conducted using ISCAS'85 and ISCAS'89 benchmark circuits. Experimental results show that production yield can be significantly improved by using the N^2-transistor structure to combat permanent defects. The proposed technique demonstrated its capability of tolerating higher defect rates than the other techniques and at a reduced area overhead.

In Chapter 4, two new repair techniques, called Tagged Replacement and Modified Tagged Replacement, that address the issue of high manufacturing defect rates in emerging nanoscale technologies are presented. The targeted fabric is based on hybrid nano/CMOS architecture implementing logic circuits as LUTs. A recently proposed repair technique called Repair Most that was implemented in the context of highly dense nano/CMOS memory architecture is used as a reference point for the evaluation of the proposed techniques. Experimental results show that the proposed techniques, while simple to implement, also have low redundancy requirements and are able to provide higher levels of defect tolerance than the Repair Most technique. While Repair Most could handle only upto 10% defect rates for small $2^3 \times 3$ LUTs, the proposed techniques exhibit a much higher efficiency and are shown to handle upto 20% defect rates. The effect of DCCs on the repair rate of the proposed techniques was also examined and it was proved that DCCs can be exploited to enhance the efficiency and/or reduce the cost of repair. The impact of defect distribution, such as clustered and row/column distributions, on the repair techniques is also evaluated. It was shown that the proposed repair techniques can tolerate higher clustered and row/column defect rates than the uniformly distributed defects.

A probabilistic design flow that improves manufacturing yield in nanoscale PLAs is presented in Chapter 5. It comprises of two defect tolerance techniques that tackle broken nanowires and defective crosspoints. Most of the recently proposed graph-base techniques rely on defect mapping at the crosspoint level which requires prohibitively large defect maps and excessive computation time for successful circuit implementation. To outperform these techniques, defect diagnosis in the proposed design flow is limited to the nanowire level rather than the crosspoint level. This level of granularity helps cutting on post-manufacturing costs by significantly reducing testing time. To validate the proposed design flow, experiments are conducted using a suite of benchmark PLAs. Various PLA implementation methods of logic circuits are taken into consideration. Experimental results show that the proposed techniques successfully achieve the required yield regardless of size, geometry and density of active junctions in the PLAs. The impact of defect rate, targeted yield, size of PLAs and density of active junctions on the area overhead of the proposed design flow is also evaluated.

Chapter 6 focuses on designing efficient techniques capable of achieving high levels of fault tolerance against physical defects and transient errors in hybrid nano/CMOS fabrics implementing logic circuits as LUTs. The aim is to enhance manufacturing yield by increasing the probability of successful instantiation of LUTs onto nanoscale fabrics and also to enhance computational reliability against transient faults that occur during circuit's operation lifetime. Experimental studies proved that single coding schemes such as Hamming and BCH and redundancy-based techniques such as N-Modular Redundancy are insufficient to tolerate the high fault rates projected in future nanoscale fabrics. Therefore, three hybrid fault tolerance techniques that combine error correcting codes with other techniques to increase their fault tolerance capability are proposed. One such approach is the two-dimensional coding technique where both rows and columns of LUTs are encoded using Hamming and BCH codes respectively. This technique significantly improves fault tolerance with respect to single error correction schemes for error rates upto 7.5%. In the second technique, error correcting codes are combined with N-Modular Redundancy. This technique exhibits better resilience to higher error rates (upto 10%) but at the cost of replicating the LUT multiple times. In the third technique, Hamming is complemented with Bad Line Exclusion. Experimental results show that this technique results in remarkable improvement in fault tolerance against a substantial fraction of faulty nanodevices (upto 20%). It was proven that the presence of DCCs in LUTs can significantly improve the efficiency of the proposed techniques namely the Hamming-Bad Line Exclusion technique. Finally, the impact of the proposed techniques in terms of area, latency and energy overheads was examined.

The contributions presented in this thesis provide novel and highly-efficient defect and fault tolerance techniques for nanometre CMOS and nanoscale devices. The conclusions drawn in this thesis are supported by extensive experimental results and mathematical equations that predict the efficiency of the proposed techniques. It is hoped that the de-

fect and fault tolerance techniques proposed in this thesis will make useful contributions towards the development of future nano-electronics.

7.2 Future Work

As demonstrated in the thesis, the efficiency of the proposed defect and fault tolerance techniques depend mainly on the rate and distribution of physical defects and transient faults in nanoscale fabrics. Accurate predictions of how defects and transient faults happen and their effect on the functionality of nanoscale devices will allow to optimise these techniques to further improve their performance and achieve better manufacturing yield and computational reliability at reduced overheads. To achieve this goal, accurate Spice models that study the electrical behaviour and performance of nanoscale devices in the presence of manufacturing defects and transient faults need to be developed. Existing models (such as the ones proposed in [Kazmierski et al., 2010, Rahman et al., Deng and Wong, 2007] for ideal Carbon Nanotubes) don't consider these defects and faults and performing simulations using these models will lead to poor predictions of the electrical behaviour and performance of nanoscale devices. Hence, the development of Spice models that consider both manufacturing defects and transient faults and provide an accurate and quantitative description of these defects and faults is essential to optimise the yield and reliability of nanodevice-based circuits.

Appendix A

Tools and Benchmark Circuits

This appendix provides brief descriptions of the benchmark circuits and tools used in the experiments referred throughout the thesis.

A.1 Software Tools

- **ModelSim (Mentor Graphics)** [Mentor-Graphics] is a Verilog/VHDL simulation tool.
- **Design Compiler (Synopsys)** [Synopsys, a] is a behavioural to gate-level synthesis tool.
- **Synplify Pro (Synopsys)** [Synopsys, b] is a synthesis tool used in FPGA designs.
- **IT++** [IT++] is a C++ library of mathematical, signal processing and communication classes and functions.

A.2 ISCAS Benchmark Circuits

The ISCAS'85 benchmark circuits [ISCAS-Circuits] are purely combinational designs with the number of gates, inputs and outputs listed in Table A.1.

circuit	Circuit Function	Total Gates	Inputs	Outputs
c432	Priority Decoder	160	36	7
c499	ECAT	202	41	32
c880	ALU and Control	383	60	26
c1355	ECAT	546	41	32
c1908	ECAT	880	33	25
c2670	ALU and Control	1193	233	140
c3540	ALU and Control	1669	50	22
c5315	ALU and Selector	2307	178	123
c6288	16-bit Multiplier	2406	32	32
c7552	ALU and Control	3512	207	108

TABLE A.1: ISCAS'85 benchmark suite

The ISCAS'89 benchmark circuits [ISCAS-Circuits] are sequential designs with the number of gates, inputs, outputs and flip-flops listed in Table A.2.

Circuit	Total Gates	Inputs	Outputs	Flip-Flops
s526	141	3	6	21
s641	107	35	24	19
s713	139	35	23	19
s820	256	18	19	5
s832	262	18	19	5
s838	288	34	1	32
s953	311	16	23	29
s1196	388	14	14	18
s1238	428	14	14	18
s1423	490	17	5	74
s5378	1004	35	49	179
s9234	2027	19	22	228
s13207	2573	31	121	669
s15850	3448	14	87	597
s35932	12204	35	320	1728
s38417	8709	28	106	1636
s38584	11448	12	278	1452

TABLE A.2: ISCAS'89 benchmark suite

A.3 Other Benchmark Circuits

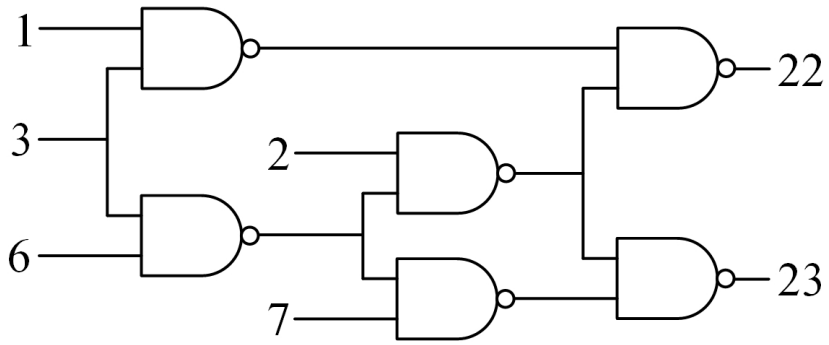


FIGURE A.1: ISCAS c17 - Original schematic

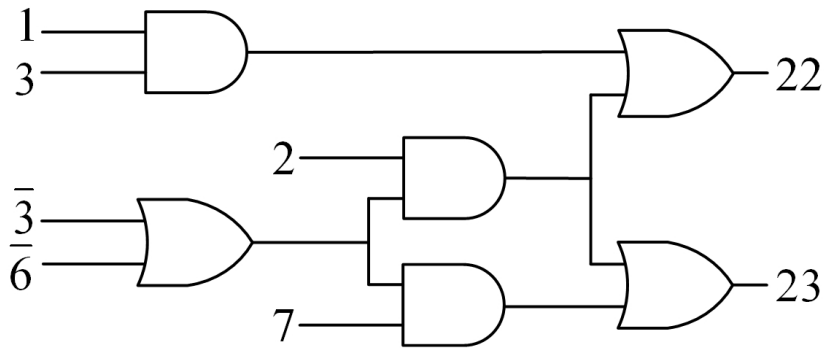


FIGURE A.2: ISCAS c17 - Equivalent schematic

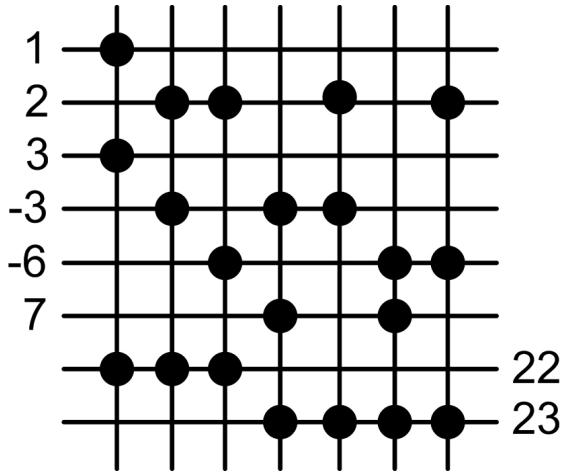


FIGURE A.3: ISCAS c17 - Single-stage PLA implementation

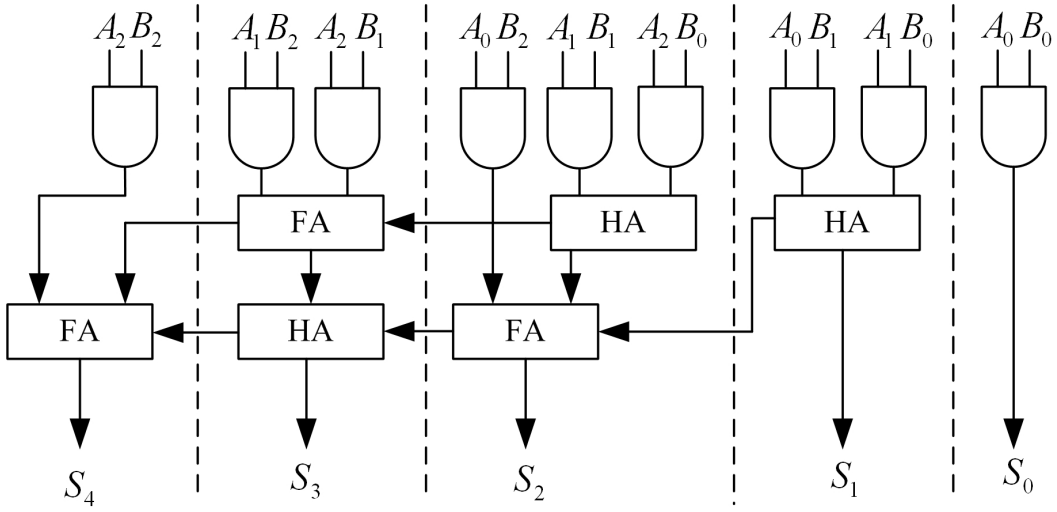


FIGURE A.4: Schematic of a 3-bit multiplier circuit

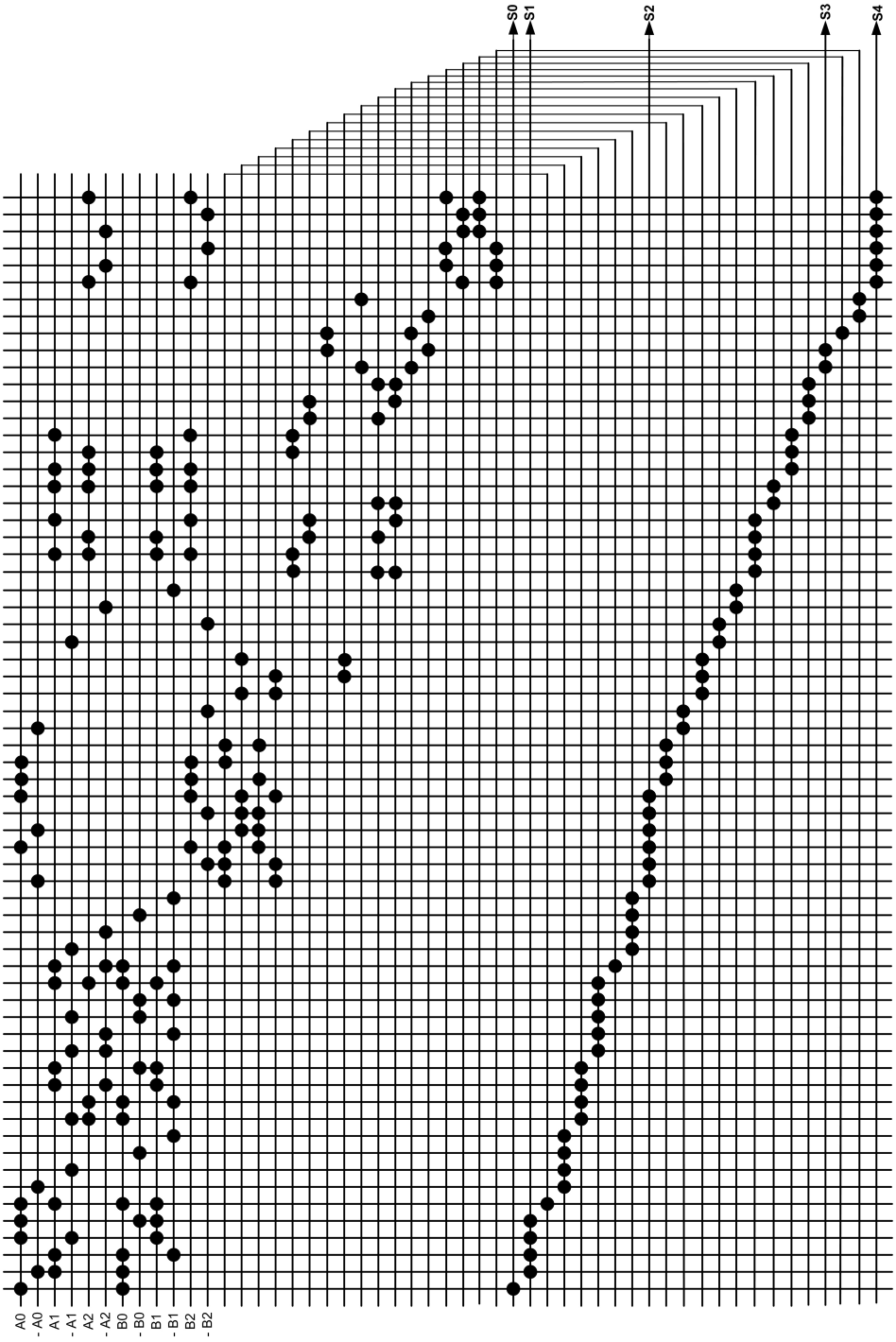
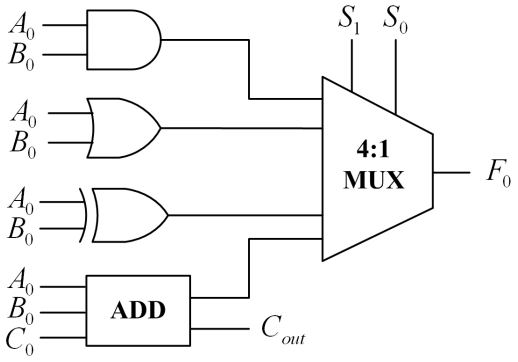


FIGURE A.5: Multiple-stage PLA implementation of a 3-bit multiplier



(a)

Control signals S_1	Control signals S_0	Operation
0	0	A and B
0	1	A or B
1	0	A xor B
1	1	A add B

(b)

FIGURE A.6: 1-bit ALU: (a) circuit schematic (b) output depends on control signals S_0 and S_1

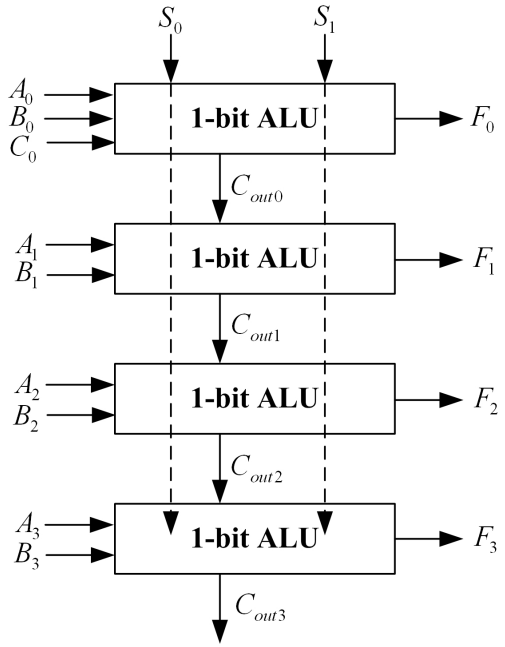


FIGURE A.7: Schematic of a 4-bit ALU circuit

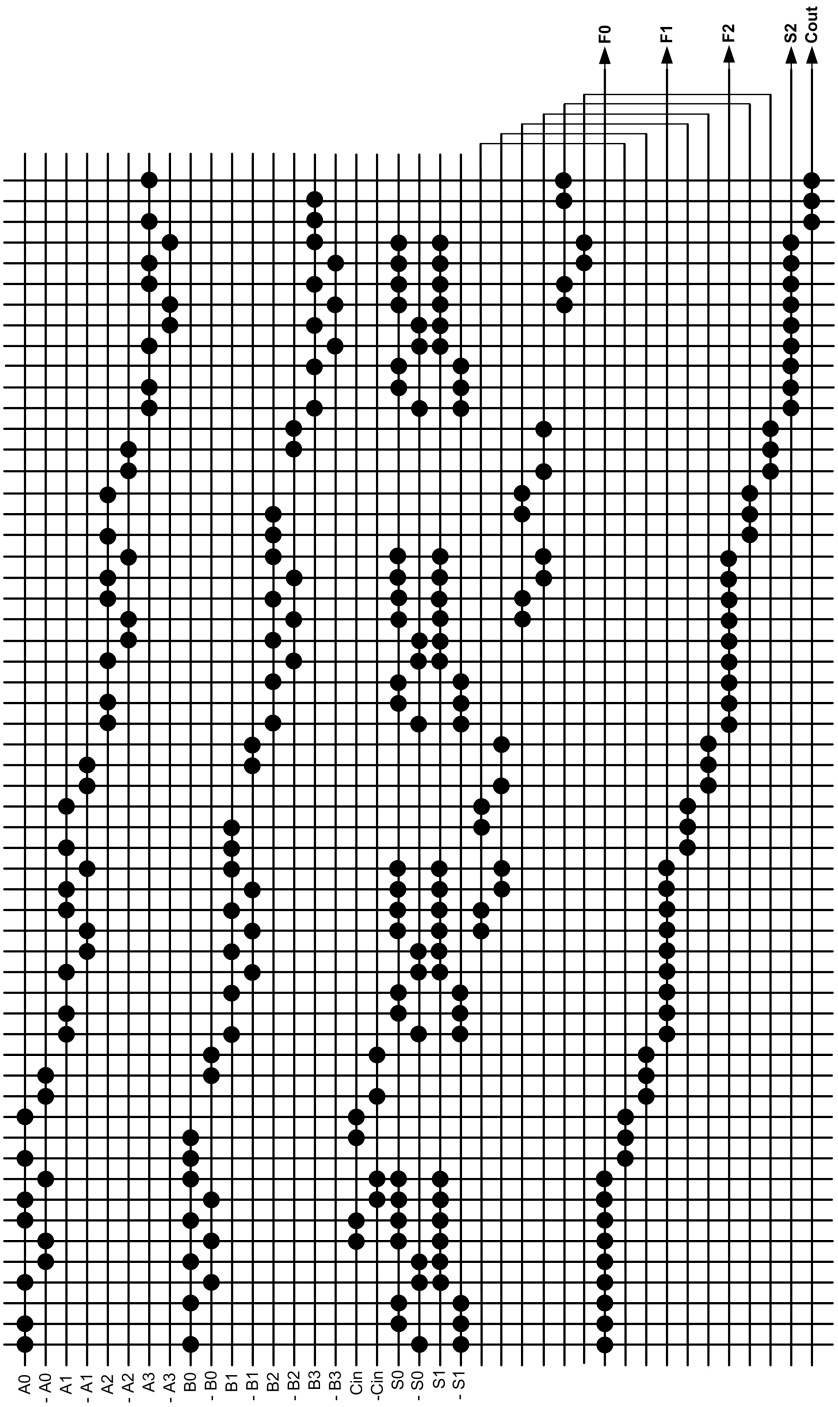


FIGURE A.8: Multiple-stage PLA implementation of a 4-bit ALU

Appendix B

Error Correcting Codes

This appendix provides an overview of the error correcting schemes used in Chapter 6 and the implementation of their decoding units using the BCH Codec Synthesis tool developed in [Jamro, 1997].

B.1 Error Correcting Codes

An error correcting code is an algorithm for expressing a sequence of numbers such that any errors which are introduced can be detected and corrected (within certain limitations) based on the remaining non-erroneous numbers. Error correcting codes are widely used in telecommunications where they are used to ensure reliable delivery of digital data over unreliable communication channels that are subject to channel noise. Error correcting codes have also been widely used in memory architectures to protect data from corruption due to transient faults.

To generate a codeword using an (n, k, t) error correcting code, the k information bits are encoded producing m parity bits giving a codeword length $n = k + m$. Any error correcting code can be used for error detection. A code with a minimum Hamming distance d_{min} can detect upto $d_{min} - 1$ errors in a codeword. For error detection and correction, the maximum number of error bits that can be corrected is $t = (d_{min} - 1)/2$. The Hamming distance d_{min} is the number of disagreements between two valid codewords of the same code.

Block Codes

Block codes operate on a block of bits. They are specified by (n, k). The code takes k information bits and adds $(n - k)$ parity bits computed by the code generator matrix to make a larger block.

Systematic Block Codes

Error detection and correction schemes can be either systematic or non-systematic. In a systematic block code, information bits remain unchanged in the codeword and the parity bits are attached either to the front or the back of the information sequence.

B.2 Hamming Code

Hamming code is considered the simplest block code. It is a single-error-correcting and double-error-detecting (SED-DED) code i.e. the code can detect up to two simultaneous bit errors and correct single-bit errors. A Hamming code is generally specified as $(2^n - 1, 2^n - n - 1)$. In other words, for full protection, n parity bits need to be added to the $2^n - n - 1$ data bits to obtain a $2^n - 1$ codeword.

The encoding and decoding algorithms of Hamming codes are based on multiplication of matrices. To generate a codeword matrix c, the information vector d is multiplied by the generator matrix G of the Hamming code as follows:

$$c = d \times G$$

The generator matrix G is based on the parity matrix H of the code and it decides how the information sequences are mapped to the valid codewords. Parity matrix H is a $(2^n - 1, n)$ matrix that contains all the possible combinations of the n parity bits without the all-zero combination. Each code is uniquely specified by its generator matrix G or parity matrix H.

Decoding of Hamming block codes is simple. A row vector called a syndrome s is computed by multiplying the codeword matrix by the transposed of the parity matrix H as follows:

$$s = H^T \times c$$

The size of the syndrome is equal to $n - k$ bits. If the syndrome is an all-zero matrix, then no error has occured in the codeword. However, if the syndrome is not an all-zero matrix, to detect and correct the error, its location is obtained by computing the ordinal number of the syndrome matrix. Correction is performed by converting the value of the erroneous bit.

B.3 Bose-Chaudhuri-Hocquenghem (BCH) Code

Bose-Chaudhuri-Hocquenghem (BCH) codes is a large class of error correction codes that are a generalisation of the Hamming codes for multiple-error correction. A BCH code is a polynomial code that operate over Galois fields (or finite fields) with a particularly chosen generator polynomial. A t-error-correcting (n, k, t) BCH code adds $(n-k)$ parity bits to the information word to correct t errors.

A binary Galois Field GF(2^m), where m is an integer, is a field with 2^m elements that can be represented by polynomials whose co-efficients are elements of the field $GF(2)$ i.e. 0 and 1. The block length of a BCH code operating over the Galois Field GF(2^m) is $n = 2^m - 1$ and the error correction capability of the code is bounded by: $t < (2^m - 1)/2$.

In (n, k, t) BCH codes, a codeword contains two parts: a remainder part for checking and the infomation part. To generate the complete codeword, the k information symbols are formed into the information polynomial $i(x) = i_0 + i_1 x + ... + i_{k-1} x^{(k-1)}$ where $i_j \in GF(2)$. Then, the codeword polynomial $c(x) = c_0 + c_1 x + ... + c_{n-1} x^{(n-1)}$ is formed by dividing the information polynomial by a generator polynomial $g(x)$ to get the remainder part $r(x)$ as follows:

$$r(x) = x^{(n-k)} i(x) (mod\ g(x))$$

The encoded codeword is:

$$c(x) = x^{(n-k)} i(x) + r(x)$$

The encoding scheme can be summerised in the following steps:

- Choose the degree m and construct Galois Field GF(2^m).
- Obtain a generator polynomial $g(x)$.
- Determine remainder $r(x)$.
- Left shift information bits by number of bits assigned for remainder.
- Append remainder to information to get a complete codeword.

Although BCH encoding is very simple and only involves Galois Field polylnomial multiplication and division, BCH decoding is much more complex and computation intense. To determine the error locations in a codeword $r(x)$, the decoding algorithm consists of the following steps:

- Calculate the syndrome set $S = S_1, S_2, ..., S_{2t}$ from the codeword $r(x)$ to be decoded.

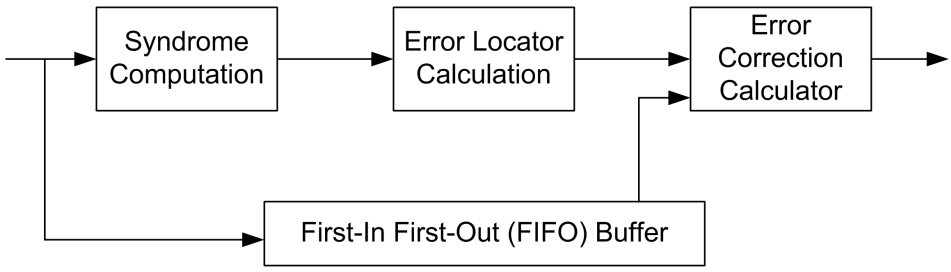


FIGURE B.1: Binary BCH decoder structure

- Determine the error-location polynomial $\sigma(x)$ from the syndrome values using the Berlekamp-Massey algorithm [Hanzo et al., 2002].
- The error locations are found by finding the inverses of the roots of $\sigma(x)$.
- Once the locations of the errors are known, the codeword $r(x)$ can be corrected to obtain the correct codeword.

Fig. B.1 illustrates the structure of binary BCH decoders which consist of three computational blocks and one first-in first-out (FIFO) buffer [Sun and Zhang, 2007, Blazek et al., 1988, Wei and Wei, 1993].

B.4 BCH Codec Synthesis Tool

In [Jamro, 1997], the author developed a BCH Codec Synthesis (BCS) tool that automatically generates VHDL description files for BCH codes. The BCS system uses a number of different files as illustrated in Fig. B.2. The BCS system consists of a C program (bch.exe) and VHDL template files (.vht files). The C program accepts the design parameters of the BCH code (bch.in) such as the code's block length and its error correcting ability that are entered by the user. Using these parameters and the VHDL templates, the tool determines the design of the encoder (enc.vhd) and decoder (dec.vhd) of the BCH code. The template files contain low level description codes that are common in all BCH codes. The C program reads these templates and copies them into the VHDL files. The C program also inserts the VHDL codes which are difficult to determine for every BCH code such as the generator polynomial for the encoders or the syndrome calculator, Berlekamp-Massey unit and the error locator unit in the decoders that are calculated using sophisticated C functions.

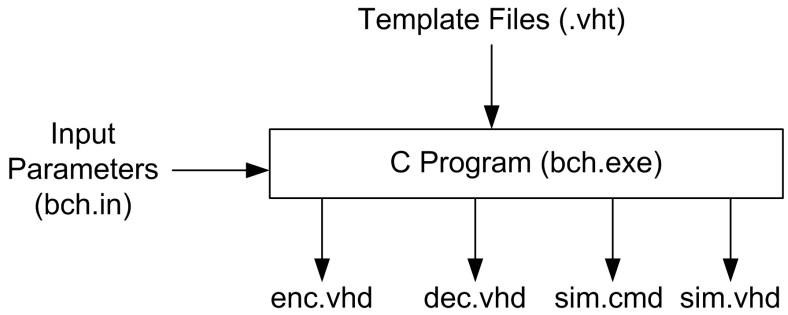


FIGURE B.2: File structure of the BCH Codec Synthesis tool

The BCS tool also generates an additional circuit in order to thoroughly simulate the design files (sim.vhd). Random input and error values are generated by the C program and exported into simulation command files (sim.cmd). Then, the simulation circuit is simulated. During simulation, input data is automatically compared with encoded-corrupted-decoded data, and if they differ, a signal is asserted to indicate that the generated codes are faulty.

Bibliography

M. Abramovici, M. Breuer, and A. Friedman. *Digital Systems Testing and Testable Tesigns*. Computer Science Press, 1990.

A.A. Al-Yamani, S. Ramsundar, and D.K. Pradhan. A Defect Tolerance Scheme for Nanotechnology Circuits. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(11):2402–2409, Nov. 2007. ISSN 1549-8328. doi: 10.1109/TCSI.2007.907875.

M. Ashouei, A.D. Singh, and A. Chatterjee. Reconfiguring CMOS as Pseudo N/PMOS for Defect Tolerance in Nano-Scale CMOS. *VLSI Design, 2008. VLSID 2008. 21st International Conference on*, pages 27–32, Jan. 2008. ISSN 1063-9667. doi: 10.1109/VLSI.2008.104.

Adrian Bachtold, Peter Hadley, Takeshi Nakanishi, and Cees Dekker. Logic Circuits with Carbon Nanotube Transistors. *Science*, 294:1317 – 1320, 2001.

R. Iris Bahar. Trends and Future Directions in Nano Structure Based Computing and Fabrication. *Computer Design, 2006. ICCD 2006. International Conference on*, pages 522–527, Oct. 2006. ISSN 1063-6404. doi: 10.1109/ICCD.2006.4380865.

R. Iris Bahar, Dan Hammerstrom, Justin Harlow, William H. Joyner Jr., Clifford Lau, Diana Marculescu, Alex Orailoglu, and Massoud Pedram. Architectures for Silicon Nanoelectronics and Beyond. *Computer*, 40(1):25–33, Jan. 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.7.

Abdellatif Bellaouar and Mohamed Elmasry. *Low-Power Digital VLSI Design: Circuits and Systems*. Springer Publication, 1995.

D. Bhaduri, S. K. Shukla, P. Graham, and H. Quinn. Transient-Error Tolerant Configuration of Nanofabrics using a Reliability driven Probabilistic Approach. *International Conference on Bio-nano-Informatics fusion*, July, 2005.

S. Biswas, Gang Wang, T.S. Metodi, R. Kastner, and F.T. Chong. Combining static and dynamic defect-tolerance techniques for nanoscale memory systems. *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 773–778, Nov. 2007. ISSN 1092-3152. doi: 10.1109/ICCAD.2007.4397359.

Susmit Biswas, Tzvetan Metodi, Frederic T. Chong, Ryan Kastner, and Tim Sherwood. Efficient Storage of Defect Maps for Nanoscale Memory.

Z. Blazek, W.D. Little, and V.K. Bhargava. Design of a (127,99) four error correcting BCH codec. pages 94 –100, 11-12 1988. doi: 10.1109/WESCAN.1988.27674.

George Bourianoff. The Future of Nanocomputing. *Computer*, 36(8):44–53, 2003. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2003.1220581.

M.A. Breuer, S.K. Gupta, and T.M. Mak. Defect and error tolerance in the presence of massive numbers of defects. *Design Test of Computers, IEEE*, 21(3):216 – 227, may-june 2004. ISSN 0740-7475. doi: 10.1109/MDT.2004.8.

Jason G. Brown and R. D. Blanton. A Built-in Self-test and Diagnosis Strategy for Chemically Assembled Electronic Nanotechnology. *J. Electron. Test.*, 23(2-3):131–144, 2007. ISSN 0923-8174. doi: http://dx.doi.org/10.1007/s10836-006-0552-x.

M. Butts, A. DeHon, and S.C. Goldstein. Molecular electronics: devices, systems and tools for gigagate, gigabit chips. *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 433–440, Nov. 2002. ISSN 1092-3152. doi: 10.1109/ICCAD.2002.1167569.

R.S. Chakraborty, S. Paul, and S. Bhunia. Analysis and Robust Design of Diode-Resistor Based Nanoscale Crossbar PLA Circuits. pages 441 –446, 4-8 2008. ISSN 1063-9667. doi: 10.1109/VLSI.2008.44.

Da-Ming Chang, Jin-Fu Li, and Yu-Jen Huang. A Built-In Redundancy-Analysis Scheme for Random Access Memories with Two-Level Redundancy. *J. Electron. Test.*, 24(1-3): 181–192, 2008. ISSN 0923-8174. doi: http://dx.doi.org/10.1007/s10836-007-5032-4.

Yong Chen, Gun-Young Jung, Douglas A A Ohlberg, Xuema Li, Duncan R Stewart, Jan O Jeppesen, Kent A Nielsen, J Fraser Stoddart, and R Stanley Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14(4):462–468, 2003a. URL http://stacks.iop.org/0957-4484/14/462.

Yong Chen, Gun-Young Jung, Douglas A A Ohlberg, Xuema Li, Duncan R Stewart, Jan O Jeppesen, Kent A Nielsen, J Fraser Stoddart, and R Stanley Williams. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14(4):462, 2003b. URL http://stacks.iop.org/0957-4484/14/i=4/a=311.

N. Cohen, T.S. Sriram, N. Leland, D. Moyer, S. Butler, and R. Flatley. Soft error considerations for deep-submicron CMOS circuit applications. pages 315 –318, 1999.

Louis Comtet. *Advanced combinatorics: the art of finite and infinite expansions; rev. version*. Reidel, Dordrecht, 1974. Trans. of : Analyse combinatoire. Paris : Presses Univ. de France, 1970.

S. Copen Goldstein and M. Budiu. NanoFabrics: spatial computing using molecular electronics. *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, pages 178–189, 2001. doi: 10.1109/ISCA.2001.937446.

S. Cotofana, A. Schmid, Y. Leblebici, A. Ionescu, O. Soffke, P. Zipf, M. Glesner, and A. Rubio. CONAN - a design exploration framework for reliable nano-electronics architectures. *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE International Conference on*, pages 260–267, July 2005. ISSN 1063-6862. doi: 10.1109/ASAP.2005.26.

Yi Cui and Charles M. Lieber. Functional Nanoscale Electronics Devices Assembled Using Silicon Nanowire Building Blocks. *Science*, 291:851 – 853, 2001.

M. Dalpasso, M. Favalli, P. Olivo, and B. Ricco. Fault simulation of parametric bridging faults in CMOS IC's. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 12(9):1403 –1410, sep 1993. ISSN 0278-0070. doi: 10.1109/43.240087.

A. DeHon. Array-based architecture for FET-based, nanoscale electronics. *Nanotechnology, IEEE Transactions on*, 2(1):23–32, Mar 2003. ISSN 1536-125X. doi: 10.1109/TNANO.2003.808508.

A. DeHon. Nanowire-Based Programmable Architectures. *ACM Journal of Emerging Technologies in Computing Systems*, 1(2):109–162, July 2005.

A. DeHon and H. Naeimi. Seven strategies for tolerating highly defective fabrication. *Design & Test of Computers, IEEE*, 22(4):306–315, July-Aug. 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.94.

A. DeHon, S.C. Goldstein, P.J. Kuekes, and P. Lincoln. Nonphotolithographic nanoscale memory density prospects. *Nanotechnology, IEEE Transactions on*, 4(2):215–228, March 2005. ISSN 1536-125X. doi: 10.1109/TNANO.2004.837849.

Jie Deng and H.-S.P. Wong. A Compact SPICE Model for Carbon-Nanotube Field-Effect Transistors Including Nonidealities and Its Application-Part I: Model of the Intrinsic Channel Region. *Electron Devices, IEEE Transactions on*, 54(12):3186 –3194, dec. 2007. ISSN 0018-9383. doi: 10.1109/TED.2007.909030.

Lisa J K Durbeck and Nicholas J Macias. The Cell Matrix: an architecture for nanocomputing. *Nanotechnology*, 12(3):217–230, 2001. URL http://stacks.iop.org/0957-4484/12/217.

Xinyue Fan, W. Moore, C. Hora, M. Konijnenburg, and G. Gronthoud. A gate-level method for transistor-level bridging fault diagnosis. pages 6 pp. –271, april 2006. doi: 10.1109/VTS.2006.6.

F. Joel Ferguson and John P. Shen. Extraction and simulation of realistic CMOS faults using inductive fault analysis. *Proceedings of the International Test Conference, IEEE*, pages 475–484, 1988.

Bo Fu and Paul Ampadu. An Energy-Efficient Multiwire Error Control Scheme for Reliable On-Chip Interconnects Using Hamming Product Codes. *VLSI Design*, 2008: 1–14, 2008.

Saturnino Garcia and Alex Orailoglu. Online test and fault-tolerance for nanoelectronic programmable logic arrays. *Nanoscale Architectures, IEEE International Symposium on*, 0:8–15, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/NANOARCH.2008.4585786.

S. Goldstein, M. Budiu, M. Mishra, and G. Venkataramani. Reconfigurable computing and electronic nanotechnology. pages 132 – 142, 24-26 2003. ISSN 1063-6862.

R. W. HAMMING. ERROR DETECTING AND ERROR CORRECTING CODES. *BELL SYSTEM TECHNICAL JOURNAL*, 29(2):147–160, 1950. ISSN 0005-8580.

I. Hamzaoglu and J.H. Patel. Test set compaction algorithms for combinational circuits. *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, pages 283–289, Nov 1998.

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

Jie Han and Pieter Jonker. From Massively Parallel Image Processors to Fault-Tolerant Nanocomputers. pages 2–7, 2004. doi: http://dx.doi.org/10.1109/ICPR.2004.402.

Jie Han, J. Gao, P. Jonker, Yan Qi, and J.A.B. Fortes. Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *Design & Test of Computers, IEEE*, 22(4): 328–339, July-Aug. 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.97.

L. Hanzo, T.H. Liew, and B.L. Yeap. *Turbo Coding, Turbo Equalisation and Space-Time Coding*. John Wiley & Sons, August 2002. URL http://eprints.ecs.soton.ac.uk/8252/.

Chen He and M.F. Jacome. RAS-NANO: A Reliability-Aware Synthesis Framework for Reconfigurable Nanofabrics. 1:1 –6, 6-10 2006. doi: 10.1109/DATE.2006.244020.

Chen He and M.F. Jacome. Defect-Aware High-Level Synthesis Targeted at Reconfigurable Nanofabrics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(5):817–833, May 2007. ISSN 0278-0070. doi: 10.1109/TCAD.2006.884401.

Chen He, M.F. Jacome, and G. de Veciana. Scalable defect mapping and configuration of memory-based nanofabrics. *High-Level Design Validation and Test Workshop, 2005.*

Tenth IEEE International, pages 11–18, Nov.-2 Dec. 2005a. ISSN 1552-6674. doi: 10.1109/HLDVT.2005.1568807.

Chen He, M.F. Jacome, and G. de Veciana. A reconfiguration-based defect-tolerant design paradigm for nanotechnologies. *Design & Test of Computers, IEEE*, 22(4): 316–326, July-Aug. 2005b. ISSN 0740-7475. doi: 10.1109/MDT.2005.76.

James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, 1998.

Tad Hogg and Greg Snider. Defect-tolerant Logic with Nanoscale Crossbar Circuits. *J. Electron. Test.*, 23(2-3):117–129, 2007. ISSN 0923-8174. doi: http://dx.doi.org/10.1007/s10836-006-0547-7.

Chih-Tsun Huang, Chi-Feng Wu, Jin-Fu Li, and Cheng-Wen Wu. Built-in redundancy analysis for memory yield improvement. *Reliability, IEEE Transactions on*, 52(4): 386–399, Dec. 2003. ISSN 0018-9529. doi: 10.1109/TR.2003.821925.

Jing Huang, Mehdi B. Tahoori, and Fabrizio Lombardi. On the Defect Tolerance of Nano-Scale Two-Dimensional Crossbars. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium d on*, 0:96–104, 2004. ISSN 1550-5774.

Jing Huang, Mariam Momenzadeh, and Fabrizio Lombardi. On the Tolerance to Manufacturing Defects in Molecular QCA Tiles for Processing-by-wire. *J. Electron. Test.*, 23(2-3):163–174, 2007. ISSN 0923-8174. doi: http://dx.doi.org/10.1007/s10836-006-0548-6.

Yu Huang, Xiangfeng Duan, Yi Cui, Lincoln J. Lauhon, Kyoung-Ha Kim, and Charles M. Lieber. Logic Gates and Computation from Assembled Nanowire Building Blocks. *Science*, 294:1313 – 1317, 2001.

ISCAS-Circuits. Collaborative Benchmarking and Experimental Algorithmics Lab. North Carolina State University. http://www.cbl.ncsu.edu/benchmarks/Benchmarks-upto-1996.html.

IT++. http://itpp.sourceforge.net/current/.

ITRS. International Technology Roadmap for Semiconductors. Technical report, 2005.

ITRS. International Technology Roadmap for Semiconductors. 2001.

M. Jacorne, Chen He, G. de Veciana, and S. Bijansky. Defect tolerant probabilistic design paradigm for nanotechnologies. *Design Automation Conference, 2004. Proceedings. 41st*, pages 596–601, 2004. ISSN 0738-100X.

Ernest Jamro. *THE DESIGN OF A VHDL BASED SYNTHESIS TOOL FOR BCH CODECS*. PhD thesis, The University of Huddersfield, 1997.

C.M. Jeffery, A. Basagalar, and R.J.O. Figueiredo. Dynamic sparing and error correction techniques for fault tolerance in nanoscale memory structures. *Nanotechnology, 2004. 4th IEEE Conference on*, pages 168–170, Aug. 2004. doi: 10.1109/NANO.2004.1392285.

Paul A. Jensen. Quadded NOR Logic. *Reliability, IEEE Transactions on*, R-12(3):22 –31, sept. 1963. ISSN 0018-9529. doi: 10.1109/TR.1963.5218213.

M.V. Joshi and W.K. Al-Assadi. Nanofabric PLA Architecture with Double Variable Redundancy. pages 32 –36, 20-22 2007. doi: 10.1109/TPSD.2007.4380347.

Tom Kazmierski, Dafeng Zhou, Bashir Al-Hashimi, and Peter Ashburn. Numerically efficient modelling of CNT transistors with ballistic and non ballistic effects for circuit simulation. *IEEE Transactions on Nanotechnology*, 9(1):99–107, January 2010. URL http://eprints.ecs.soton.ac.uk/20863/.

Israel Koren and Adit D. Singh. Fault Tolerance in VLSI Circuits. *Computer*, 23(7): 73–83, 1990. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.56854.

R. Kothe, H.T. Vierhaus, T. Coym, W. Vermeiren, and B. Straube. Embedded Self Repair by Transistor and Gate Level Reconfiguration. *Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE*, pages 208–213, 0-0 2006. doi: 10.1109/DDECS.2006.1649613.

Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. *VLSI Design*, 2007:1–13, 2007.

C.S. Lent, P.D. Tougaw, and W. Porod. Quantum cellular automata: the physics of computing with arrays of quantum dot molecules. pages 5 –13, 17-20 1994. doi: 10.1109/PHYCMP.1994.363705.

Shu Li and Tong Zhang. Exploratory study on circuit and architecture design of very high density diode-switch phase change memories. pages 424–429, 2009. doi: http://dx.doi.org/10.1109/ISQED.2009.4810332.

Shyue-Kung Lu and Chih Hsien Hsu. Built-In self-repair for divided word line memory. *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, 4:13–16 vol. 4, May 2001. doi: 10.1109/ISCAS.2001.922156.

Shyue-Kung Lu and Chih-Hsien Hsu. Fault tolerance techniques for high capacity RAM. *Reliability, IEEE Transactions on*, 55(2):293–306, June 2006. ISSN 0018-9529. doi: 10.1109/TR.2006.874912.

X. Ma, D.B. Strukov, J.H. Lee, and K.K. Likharev. Afterlife for silicon: CMOL circuit architectures. pages 175 – 178 vol. 1, 11-15 2005. doi: 10.1109/NANO.2005.1500722.

X. Ma, J. Huang, F. Chiminazzo, D. Rossi, C. Metra, and F. Lombardi. Resistive Crossbar Switching Networks for Inherently Fault Tolerant Nano LUTs. pages 21 –24, 29-30 2008. doi: 10.1109/NDCS.2008.21.

W. Maly. Realistic fault modeling for VLSI testing. pages 173–180, 1987. doi: http://doi.acm.org/10.1145/37888.37914.

Mentor-Graphics. ModelSim. http://www.mentor.com/products/fv/modelsim/.

M. Mishra and S.C. Goldstein. Defect tolerance at the end of the roadmap. *Test Conference, 2003. Proceedings. ITC 2003. International*, 1:1201–1210, 30-Oct. 2, 2003. ISSN 1089-3539.

T. Mizuochi, Y. Miyata, T. Kobayashi, K. Ouchi, K. Kuno, K. Kubo, K. Shimizu, H. Tagami, H. Yoshida, H. Fujita, M. Akita, and K. Motoshima. Forward error correction based on block turbo code with 3-bit soft decision for 10-Gb/s optical communication systems. *Selected Topics in Quantum Electronics, IEEE Journal of*, 10 (2):376–386, March-April 2004. ISSN 1077-260X. doi: 10.1109/JSTQE.2004.827846.

H. Naeimi and A. DeHon. A greedy algorithm for tolerating defective crosspoints in nanoPLA design. *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 49–56, Dec. 2004. doi: 10.1109/FPT.2004.1393250.

Helia Naeimi and Andre DeHon. Fault tolerant nano-memory with fault secure encoder and decoder. *Nano-Net '07: Proceedings of the 2nd international conference on Nano-Networks*, pages 1–7, 2007.

T. Nakura, Y. Tatemura, G. Fey, M. Ikeda, S. Komatsu, and K. Asada. SAT-based ATPG testing of inter- and intra-gate bridging faults. pages 643 –646, 23-27 2009. doi: 10.1109/ECCTD.2009.5275065.

Nantero. http://www.nantero.com.

K. Nepal, R.I. Bahar, J. Mundy, W.R. Patterson, and A. Zaslavsky. MRF Reinforcer: A Probabilistic Element for Space Redundancy in Nanoscale Circuits. *Micro, IEEE*, 26(5):19–27, Sept.-Oct. 2006. ISSN 0272-1732. doi: 10.1109/MM.2006.96.

K Nikolic, A Sadek, and M Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotechnology*, 13(3):357–362, 2002. URL http://stacks.iop.org/0957-4484/13/357.

Somnath Paul, Rajat Subhra. Chakraborty, and Swarup Bhunia. Defect-Aware Configurable Computing in Nanoscale Crossbar for Improved Yield. *IEEE International On-Line Testing Symposium/On-Line Testing Symposium, IEEE International*, 0:29–36, 2007. doi: http://doi.ieeecomputersociety.org/10.1109/IOLTS.2007.25.

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

Ilia Polian, Sandip Kundu, Jean-Marc Galliere, Piet Engelke, Michel Renovell, and Bernd Becker. Resistive Bridge Fault Model Evolution from Conventional to Ultra Deep Submicron Technologies. *VLSI Test Symposium, IEEE*, 0:343–348, 2005. ISSN 1093-0167. doi: http://doi.ieeecomputersociety.org/10.1109/VTS.2005.72.

Reza M. P. Rad and Mohammad Tehranipoor. A new hybrid FPGA with nanoscale clusters and CMOS routing. pages 727–730, 2006a. doi: http://doi.acm.org/10.1145/1146909.1147094.

R.M.P. Rad and M. Tehranipoor. A Reconfiguration-based Defect Tolerance Method for Nanoscale Devices. pages 107 –118, 4-6 2006b. ISSN 1550-5774. doi: 10.1109/DFT.2006.10.

A. Rahman, J.Wang, J. Guo, S. Hasan, Y. Liu, A. Matsudaira, S. S. Ahmed, S. Datta, and M. Lundstrom. Fettoy 2.0. URL https://www.nanohub.org/resources/220.

W. Rao, A. Orailoglu, and R. Karri. Logic Level Fault Tolerance Approaches Targeting Nanoelectronics PLAs. *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pages 1–5, April 2007. doi: 10.1109/DATE.2007.364401.

Wenjing Rao, A. Orailoglu, and R. Karri. Nanofabric topologies and reconfiguration algorithms to support dynamically adaptive fault tolerance. *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, pages 6 pp.–, April-4 May 2006. doi: 10.1109/VTS.2006.50.

Naresh R. Shanbhag, Subhasish Mitra, Gustavode de Veciana, Michael Orshansky, Radu Marculescu, Jaijeet Roychowdhury, Douglas Jones, and Jan M. Rabaey. The Search for Alternative Computational Paradigms. *IEEE Des. Test*, 25(4):334–343, 2008. ISSN 0740-7475. doi: http://dx.doi.org/10.1109/MDT.2008.113.

A. Singh, H.A. Zeineddine, A. Aziz, S. Vishwanath, and M. Orshansky. A heterogeneous CMOS-CNT architecture utilizing novel coding of boolean functions. *Nanoscale Architectures, 2007. NANOSARCH 2007. IEEE International Symposium on*, pages 15–20, Oct. 2007. doi: 10.1109/NANOARCH.2007.4400852.

Naran Sirisantana, B.C. Paul, and Kaushik Roy. Enhancing yield at the end of the technology roadmap. *Design & Test of Computers, IEEE*, 21(6):563–571, Nov.-Dec. 2004. ISSN 0740-7475. doi: 10.1109/MDT.2004.86.

G. S. Snider and R. S. Williams. Nano/CMOS architectures using a field-programmable nanowire interconnect. *Nanotechnology*, 18(3):035204 (11pp), 2007. URL http://stacks.iop.org/0957-4484/18/035204.

S. Spagocci and T. Fountain. Fault Rates in Nanochip Devices. *The Electrochemical Society Inc.*, 98-19:582–593, 1999.

M.R. Stan, P.D. Franzon, S.C. Goldstein, J.C. Lach, and M.M. Ziegler. Molecular electronics: from devices and interconnect to circuits and architecture. *Proceedings of the IEEE*, 91(11):1940–1957, Nov 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2003.818327.

Dmitri B Strukov and Konstantin K Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotechnology*, 16(1):137–148, 2005a. URL http://stacks.iop.org/0957-4484/16/137.

Dmitri B. Strukov and Konstantin K. Likharev. CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16(6):888–900, 2005b. doi: http://dx.doi.org/10.1088/0957-4484/16/6/045. URL http://dx.doi.org/10.1088/0957-4484/16/6/045.

Dmitri B. Strukov and Konstantin K. Likharev. A reconfigurable architecture for hybrid CMOS/Nanodevice circuits. pages 131–140, 2006. doi: http://doi.acm.org/10.1145/1117201.1117221.

Fei Sun and Tong Zhang. Defect and Transient Fault-Tolerant System Design for Hybrid CMOS/Nanodevice Digital Memories. *Nanotechnology, IEEE Transactions on*, 6(3):341–351, May 2007. ISSN 1536-125X. doi: 10.1109/TNANO.2007.893572.

Synopsys. Design Compiler. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DCUltra.aspx, a.

Synopsys. Synplify Pro. http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPro.aspx, b.

M.B. Tahoori. A mapping algorithm for defect-tolerance of reconfigurable nano-architectures. pages 668 – 672, 6-10 2005a. doi: 10.1109/ICCAD.2005.1560150.

M.B. Tahoori. Defects, yield, and design in sublithographic nano-electronics. *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, pages 3–11, Oct. 2005b. doi: 10.1109/DFTVS.2005.28.

Mehdi B. Tahoori. Application-independent defect tolerance of reconfigurable nanoarchitectures. *J. Emerg. Technol. Comput. Syst.*, 2(3):197–218, 2006a. ISSN 1550-4832. doi: http://doi.acm.org/10.1145/1167943.1167945.

Mehdi B. Tahoori. Application-independent defect-tolerant crossbar nano-architectures. pages 730–734, 2006b. doi: http://doi.acm.org/10.1145/1233501.1233652.

Mohammad Tehranipoor. Guest Editorial, J. Electronic Testing: Theory and Applications. volume 23, pages 115–116, 2007.

D.D. Thaker, R. Amirtharajah, F. Impens, I.L. Chuang, and F.T. Chong. Recursive TMR: scaling fault tolerance in the nanoscale era. *Design & Test of Computers, IEEE*, 22(4):298–305, July-Aug. 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.93.

J. G. Tryon. Quadded Logic, Redundancy Techniques for Computing Systems. *Spartan Books*, pages 205–228, 1962.

J. von Neumann. Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components. *Automata Studies*, pages 43–98, 1956.

Zhanglei Wang and Krishnendu Chakrabarty. Built-in Self-test and Defect Tolerance in Molecular Electronics-based Nanofabrics. *J. Electron. Test.*, 23(2-3):145–161, 2007. ISSN 0923-8174. doi: http://dx.doi.org/10.1007/s10836-006-0550-z.

N. Wehn, M. Glesner, P. Mann, K. Caesar, and A. Roth. Design of a defect-tolerant and fully testable PLA. *Circuits and Systems, 1988., IEEE International Symposium on*, pages 213–216 vol.1, Jun 1988. doi: 10.1109/ISCAS.1988.14905.

S.-W. Wei and C.-H. Wei. A high-speed real-time binary BCH decoder. *Circuits and Systems for Video Technology, IEEE Transactions on*, 3(2):138 –147, apr 1993. ISSN 1051-8215. doi: 10.1109/76.212719.

Chi Feng Wu, Chua Chin Wang, Rain Test Hwang, and Chia Hsiung Kao. Dynamic NOR-NOR PLA Design with IDDQ testability. *International Journal of Electronics*, 86:79–85, 1999.

A.J. Yu and G.G. Lemieux. FPGA defect tolerance: impact of granularity. pages 189 –196, 11-14 2005. doi: 10.1109/FPT.2005.1568545.

W. Zhang, Niraj K. Jha, and Li Shang. NATURE: a hybrid nanotube/CMOS dynamically reconfigurable architecture. pages 711–716, 2006. doi: http://doi.acm.org/10.1145/1146909.1147091.

Wei Zhang, Li Shang, and N.K. Jha. NanoMap: An Integrated Design Optimization Flow for a Hybrid Nanotube/CMOS Dynamically Reconfigurable Architecture. *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 300–305, June 2007. ISSN 0738-100X.

C. Zhao, S. Dey, and X. Bai. Soft-spot analysis: targeting compound noise effects in nanometer circuits. *Design & Test of Computers, IEEE*, 22(4):362–375, July-Aug. 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.95.

Yexin Zheng and Chao Huang. Defect-aware logic mapping for nanowire-based programmable logic arrays via satisfiability. *Design, Automation and Test in Europe, DATE'09*, pages 1279–1283, 2009.

Matthew M. Ziegler and Mircea R. Stan. A Case for CMOS/nano co-design. pages 348–352, 2002a. doi: http://doi.acm.org/10.1145/774572.774624.

M.M. Ziegler and M.R. Stan. CMOS/nano co-design for crossbar-based molecular electronic systems. *Nanotechnology, IEEE Transactions on*, 2(4):217–230, Dec. 2003. ISSN 1536-125X. doi: 10.1109/TNANO.2003.820804.

M.M. Ziegler and M.R. Stan. Design and analysis of crossbar circuits for molecular nanoelectronics. pages 323 – 327, 2002b. doi: 10.1109/NANO.2002.1032256.