Robust Design Techniques for Emerging Technologies of Computing

A Dissertation Presented

by

Masoud Zamani

to
The Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

for the Degree of

Doctor of Philosophy

in

Computer Engineering

Northeastern University
Boston, Massachusetts

April 2013
To My Parents
Robust Design Techniques for Emerging Technologies of Computing

by

Masoud Zamani

ABSTRACT OF DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Engineering in the Graduate School of Engineering Northeastern University, April 2013
Abstract

Conventional CMOS technology used in implementation of computational circuits faces major challenges in continues downscaling. Therefore, researchers explore alternative emerging implementation technologies and alternative computational approaches to overcome these challenges. However, in nanoscale regimes due to atomic scale of devices and poor control in nanofabrication, reliability is a major challenge. In this thesis, we will study reliability issues in crossbar nano-architectures, as an example of alternative implementation, as well as reversible logic, as an example of alternative computational technology.

We study two approaches, namely logic mapping and architectural techniques, to incorporate variation and defect tolerance in crossbar nano-architectures. In the logic mapping approach, different configurations of a logic function on a crossbar nano-architecture are explored to find a reliable configuration which results in better variation and defect tolerance. Simulation results, on a set of benchmarks circuits, show that the proposed method achieve variation tolerance more than 98% of the cases, while in 100% of the cases all defects are eliminated in the mapping. We also use asynchronous design methodologies to propose a self-time crossbar nano-architecture, which allows us to eliminate global clock-like signals (by replacing with local handshake signals) to reduce circuit vulnerability to delay variation. Compared to synchronous counterparts, with around 50% overhead the proposed architecture provides 100% tolerance to delay variations.

In terms of reversible circuits, we study online and offline testing of these circuits as well as fault masking and diagnosis. In order to provide online testing, we use a parity generation methodology to detect faults. This method provides 100% coverage for single fault and more than 99% coverage for multiple faults with 25% fault rate. Furthermore, a cyclic test generation methodology is used to provide test patterns with small amount of test information, applicable for in the field testing. We also propose a set of majority voting gates which can be used in Triple Modular Redundancy (TMR) circuits to enable
both fault masking and fault diagnosis. Simulation results show that the average overhead of the proposed majority voters are less than 38% while they provide fault masking and diagnosis.
I wish to express my sincere gratitude to my advisor Professor Tahoori for all his supports during my PhD study. I would like to thank him for all his helpful guidances, stimulating discussions, significant comments, and continuous follow ups. I have learned a lot of technical and non-technical skills from Professor Tahoori, and I owe him my special thanks.

My sincere thanks also goes to Professor Schirner and Professor Fei for participating in my thesis committee and their constructive comments to improve the quality of this thesis.

I would also like to thank Professor Salehi, Professor Farhat, and Mrs. Crisley for all their helps during my study at Northeastern University.

Last but not the least, a very special thanks to my parents, Alireza and Azimeh, and my grandmother Rokhsareh for all their supports and never-ending love during my entire life. Another special thanks to my beloved friend Hanieh for all her supports and encouragements. I also would like to thanks my brothers Davoud, Ehsan, and Ali and my sister-in-law Talieh for always being there for me.

Masoud Zamani

Northeastern University

April 2013
Contents

Abstract iv

Acknowledgments vii

List of Figures x

List of Tables xiii

Chapter 1 Introduction 1
  1.1 Background and Motivation ........................................ 1
  1.2 Reliability Concerns in Emerging Technologies and Computations ........ 2
  1.3 Target Emerging Technology and Computational Approach ................... 3
  1.4 Organization ................................................................ 4

Chapter 2 Emerging Technologies for Computing 5
  2.1 Crossbar Nano-Architectures ......................................... 5
  2.1.1 Nano-PLA ...................................................... 9
  2.1.2 FET-Based Crossbar ............................................ 10
  2.2 Reversible Computation ............................................. 11

Chapter 3 Robust Design for Crossbar Nano-Architectures 15
  3.1 Previous Work ....................................................... 15
  3.1.1 Background ..................................................... 15
  3.1.2 Defect Tolerant Techniques ................................... 18
Contents

3.1.3 Variation Tolerant Techniques ........................................... 20

3.2 Proposed Approaches ......................................................... 21
  3.2.1 Architectural Approach .................................................. 22
  3.2.2 Post Synthesis Transformations ...................................... 33
  3.2.3 Post Synthesis Methodologies and Algorithms .................. 36

Chapter 4 Reliable Reversible Circuit Design ..................................... 52

  4.1 Previous Work ................................................................. 52
    4.1.1 Background ............................................................. 52
    4.1.2 Test Generation ....................................................... 56
    4.1.3 Fault Masking .......................................................... 57
    4.1.4 Online Testing ......................................................... 57
    4.1.5 Design for Testability ............................................... 58

  4.2 Proposed Approaches ........................................................ 59
    4.2.1 Online Testing of Missing/Repeated Gate Faults ............... 59
    4.2.2 Fault Masking in Reversible Circuits ............................ 66
    4.2.3 Test Generation for Reversible Circuits ....................... 76

Chapter 5 Summary and Conclusions ................................................. 84
# List of Figures

2.1 Crossbar Array, as a feasible structure for Nanoelectronics ........................................... 6
2.2 Suspension method to configure crosspoints [13] ................................................................. 6
2.3 Structure consists of p-FET, n-FET, and switch crossbar arrays ......................................... 7
2.4 Realizations of crosspoint as n-FET, p-FET and switch ....................................................... 8
2.5 Cascaded Diode-based crossbars with global control signal .................................................. 9
2.6 Circuit equivalent of a logic block of nano-PLA followed by restoration (inversion) unit [14] ........................................................................................................................................ 10
2.7 FET-based crossbar: (a) a FET-Based crossbar realizing NOR logic (b) cascaded blocks ............................................................................................................................................. 11
2.8 Universal reversible gates: (a) NOT (b) Feynman (c) Toffoli (d) Fredkin .............................. 13
3.1 A portion of diode-based crossbar: (a) the schematic of the circuit (b) the equivalent RC model of the circuit ........................................................................................................................................ 15
3.2 A portion of FET-based crossbar: (a) the schematic of the circuit (b) the equivalent RC model of the circuit ........................................................................................................................................ 17
3.3 Defective crossbar: (a) possible different defects, (b) defect effects on the diode-based crossbar, and (c) defect effects on the FET-based crossbar .............................................. 18
3.4 A set of functions and the corresponding graph representation ............................................... 19
3.5 A crossbar and the corresponding graph representation .......................................................... 20
3.6 Structure of a block of self-timed nano-PLA .......................................................................... 22
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>Structure of logic computation unit: (a) Crossbar structure to implement AND logic (b) Crossbar structure to implement OR logic</td>
</tr>
<tr>
<td>3.8</td>
<td>A part of control implementation: (a) implementation of AND-Plane inputs (b) implementation of AND-Plane reset (c) implementation of $I_{Ack}$ generator</td>
</tr>
<tr>
<td>3.9</td>
<td>Two cascaded blocks of self-timed nano-PLA on 2D crossbars for logic and routing units, and lithography based fabricated FET devices</td>
</tr>
<tr>
<td>3.10</td>
<td>Mapping of two logic functions ($O_3$ and $O_4$) into two logic blocks of a nano-PLA a) The configurations of the logic blocks before applying any logic transformation b) The configurations of the logic blocks after applying a set of intra and inter block logic transformations</td>
</tr>
<tr>
<td>3.11</td>
<td>Improvements of solving ILP formulations on delay minimization ($2X$ of delay at each time has been shown), variation tolerance (VT), and Defect Tolerance (DT) with respect to different time limits for a crossbar with 144 crosspoints</td>
</tr>
<tr>
<td>3.12</td>
<td>Improvements of running SA on delay minimization ($2X$ of delay at each time has been shown), variation tolerance (VT), and Defect Tolerance (DT) over the time for a crossbar with 144 crosspoints</td>
</tr>
<tr>
<td>3.13</td>
<td>Saturation time of solving ILP formulations for different objectives with respect to crossbar size</td>
</tr>
<tr>
<td>4.1</td>
<td>Illustration of logical fault models for reversible circuits: (a) the fault-free circuit (b) the SMGF on the second gate (c) the RGF on the second gate (d) the MMGF of the last two consecutive gates (e) the first-order PMGF of the first control input of the first gate (f) Appearance fault on the second gate</td>
</tr>
<tr>
<td>4.2</td>
<td>General scheme of online missing/repeated gate detection: a) A fault-free circuit which produces $D$ on the first primary output, b) Gate missing fault on gate $R_2$ which results in $\neg D$ on the first primary output, and c) Repeated gate fault on gate $R_2$ which results in $\neg D$ on the first primary output</td>
</tr>
<tr>
<td>4.3</td>
<td>The schematic of the proposed reversible Logic Gate (LG)</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.4 Configuration of LG to provide fan-out branch .......................... 61
4.5 The schematic of the reversible D-Collector Gate (DCG) ............... 62
4.6 Detection scheme for a circuit consists of two D-paths (FG denotes LG which is used as fan-out) ............................................................. 63
4.7 Proposed reversible majority voter, with triplicated voted output ...... 66
4.8 Minimal Triplicated Voter (MTV) ............................................. 67
4.9 Robust Triplicated Voter (RTV) ............................................. 68
4.10 RTV in a TMR system: Address lines and data outputs determine the fault location (values in fault-free case/values in the presence of a single fault on $DI_2$/values in the presence of two faults on $DI_1$ and $C_1$) .......... 71
4.11 Module collecting the diagnosis information of $i$ RTVs .................. 72
4.12 Cascaded DCs to collect diagnosis information of 4 RTVs ............. 72
4.13 An implementation of a DC for 4 RTVs .................................... 73
4.15 The schematic of a reversible circuit (Circuit 3-17 [15]) ................. 77
4.16 An example of Ping-Pong test vector: (a) the truth table and the fault coverage of each input pattern (b) a Ping-Pong sequence to detect all faults ($V^1 = 000$, $V^2 = 111$); $T = (000, 2, 101)$ ................................. 77
4.17 The fault coverage of the four Ping-Pong tests with respect to Multiple Missing Gate Faults ..................................................... 81
4.18 The fault coverage of the four Ping-Pong tests with respect to Multiple Single Missing Gate Faults ........................................ 82
4.19 The fault coverage of the four Ping-Pong tests with respect to Multiple Stuck-at 0 Faults ..................................................... 82
4.20 The fault coverage of the four Ping-Pong tests with respect to Multiple Stuck-at 1 Faults ..................................................... 83
List of Tables

3.1 Comparison of different methods at various abstraction levels (Level) in achieving Variation Tolerance (VT) and Defect Tolerance (DT) ........................................ 21
3.2 Dual-rail coding ................................................................................... 25
3.3 Simulation results on a set of benchmarks, Total number of FET devices ($FET$), total number of Diode devices ($D$), average number of activated inputs ($I$), products ($P$), and outputs ($O$) per block, number of stages ($S$), variation tolerance ($VT$), the improvements, and overheads ................................................. 31
3.4 Variation Tolerance (VT) and Defect Tolerance (DT) achieved by un-aware mapping (random mapping), SA and ILP ................................................. 45
3.5 Critical path delay (P-Delay) and average crossbar delay (C-delay) achieved by un-aware mapping (random mapping), SA and ILP ........................................ 46
3.6 Average unused rows, columns, and crosspoints with respect to crossbar size ............................................................ 50
3.7 Percentage of successful reliable mapping of blocks (Block Yield (B-Y)) and the circuits (Circuit Yield (C-Y)) achieved by Simulated Annealing (SA) and the proposed method ................................................................. 51
4.1 The truth table of the proposed reversible Logic Gate (LG) .................. 61
4.2 The truth table of the reversible $D$-Collector Gate (DCG) .................... 62
4.3 Experimental results on a set of benchmarks; The number of garbage outputs and delay .............................................................................. 64
4.4 Experimental results on a set of benchmarks; The number of gates and the number of input nets ................................. 65
4.5 Online detection coverage achieved by the proposed method for different fault rates .................................................. 65
4.6 Input-output relation of the MTV ........................................... 67
4.7 Input-output relation of the RTV ........................................... 69
4.8 Fault Analysis for MTV and RTV: Percentage of Maskable errors (M), Recoverable errors (R), and Unrecoverable errors (U) ........................................... 69
4.9 Offline fault diagnosis based on address outputs of RTV ............... 71
4.10 Area comparison of the two proposed reversible voters. # G: Number of GATES, # IN: total number of INPUTS ...................... 74
4.11 Area overhead of the two proposed reversible voters .................. 75
4.12 Ping-Pong test sessions generated for 100% coverage of SMGF (P-SMGF), SSA0 (P-SSA0), SSA1 (P-SSA1), and all these three fault models (P-Merge). The number of numbers inside of brackets shows the number of test sessions, and each number show the number of cycles for the corresponding test session. m indicates the number of gates and n indicates the number of inputs of the corresponding circuit. ........................................... 80
Chapter 1

Introduction

1.1 Background and Motivation

Conventional *Complementary Metal Oxide Semiconductor* (CMOS) technology faces major challenges in the continuation of the Moore’s law in down-scaling of the device feature size. Inherent physical issues such as ultra-thin gate oxides, short channel effects, doping fluctuations across the chip, and diffraction effects of sub-wavelength lithography cause significant complications in successive device feature size reduction. Therefore, researchers are investigating *alternative implementations* as well as *alternative computational approaches* to overcome such limitations. In the alternative implementations new materials and devices are exploited in emerging fabrication process to implement a circuit. Examples of alternative implementations include: *Memristor* (resistor with memory; a passive element which provides relation between flux and charge) [16], *Spintronics* (Spin Electronics; using spin effects to create electronic device elements) [17], *CNTFET* (Carbon NanoTube Field Effect Transistor; FET transistor exploiting carbon nanotube as transistor channel) [18], *GFET* (Graphene Field Effect Transistor; using graphene instead of silicon in FET devices) [19], *Ion-Trap Technology* (technology to implement devices by trapping ion) [20]. However, using regular structures such as crossbar nano-architectures are easier to fabricate in self-assembly nano fabrication. A crossbar nano-architecture is a two-dimensional grid structure with configurable switches and devices at the crosspoints [21], which can be built
up from Memristor, CNTFET, or crossed nanowires and nanotubes. On the other hand, a set of alternative computational approaches have been proposed to resolve computational limitations of conventional approaches. Examples of alternative computational approaches include: Quantum Computing (computational approach based on quantum mechanical phenomena) [22], Probabilistic Automaton (a mathematical framework for the specification and analysis of probabilistic systems) [23], Reversible Logic (bijective reversible computational process) [24].

1.2 Reliability Concerns in Emerging Technologies and Computations

In nanoscale regimes because of atomic device sizes and poor control on new nanofabrication processes (such as self-assembly nanofabrication) the challenges in terms of reliability of the circuits is increasing. Some of the major reliability issues are extreme parametric variations, high defect rate at manufacturing, and high failure rate during lifetime operation. In the nanoscale regime, due to atomic-scale of device parameters, a small deviation in the device parameters can significantly change device characteristics [25]. Furthermore, poor control in nanofabrication process increases the process variation [26]. Process variation results in unpredictability of the fabricated circuit by adversely affecting circuit delay, power and even functionality. On the other hand, high defect rate is predicted to be a major challenge in digital systems in terms of low manufacturing yield. This is more likely and significant for circuit design in leading-edge technology nodes, due to inherent unreliability and operational principles of devices in such technologies [27]. Furthermore, due to faults during runtime operation of the circuit as well as undetected defects during manufacturing test, device aging poses challenges in terms of limiting the reliability of the circuits.
CHAPTER 1. INTRODUCTION

1.3 Target Emerging Technology and Computational Approach

Crossbar nano-architecture covers wide range of alternative implementation technologies, because a device at a crosspoint of a crossbar nano-architecture can be a memristor, a CNTFET, or the crossed nanowires and nanotubes. On the other hand, since quantum computers are built up from elementary unitary matrices which are inherently reversible, therefore focus on reversible logics can cover both reversible circuits and quantum computers. Therefore, the scope of this research is reliability of the crossbar nano-architecture as an example of alternative implementation technologies and reversible logic as an example of alternative computational approaches.

Using Carbon Nano Tubes (CNTs) and semiconductor Nano Wires (NWs) in bottom-up self-assembly techniques is a promising alternative for conventional CMOS technology. Fundamental electronic devices such as diode, Field Effect Transistors (FET), and memory elements have been assembled from well-defined nanoscale building blocks (CNTs and SiNWs) [28, 29, 30, 31]. In crossbar nano-architectures, CNTs and SiNWs are aligned in one direction orthogonally superimposed with another set of CNTs or SiNWs to construct architectures similar to Programmable Logic Array (PLA) [28, 30].

Studies on reversible logic were initiated in 1985, based on a thermodynamic theory stating every lost or duplicated bit information causes energy loss of $kT \ln 2$, where $k$ is the Boltzmann’s constant and $T$ is the temperature [32]. Therefore, reversible computation, which avoids the information loss, was introduced as a computational logic approach to reduce or even eliminate the power consumed by the computation. A circuit without fan-out branches and feedback loops is called reversible if each input pattern to the circuit can be determined from the output pattern and vice versa [33]. Further research on reversible logic is motivated based on the fact that reversible logic is an inherent design requirement of quantum computation [33, 34, 35].
1.4 Organization

Chapter 2 provides preliminaries about emerging technologies in terms of alternative implementations as well as alternative computational approaches. Various emerging technologies and computational approaches are briefly reviewed. Crossbar nano-architectures are introduced in more detail in terms of devices and nano-architectures. In addition, preliminaries about reversible computation and basic idea behind reversible logic as well as reversible gates and circuits are introduced.

Robust design for crossbar nano-architectures are explained in Chapter 3. In this chapter previous work as well as delay/defect model are introduced. Then, our approaches in the field of robust design for crossbar nano-architectures are introduced. A self-timed nano-architecture is proposed to eliminate the need for global clock-like signals in the circuit. Simulation results show that the proposed architecture provides 100% robustness against delay variation. Furthermore, a set of logic transformations and algorithms are proposed to provide defect and variation tolerant configuration, during logic mapping into crossbar nano-architectures. These algorithms and transformations in more than 99% of the cases provide defect and variation tolerant configurations, which enable realization of high yield.

Chapter 4 discusses reliable circuit design in reversible logic. Fault models in these circuits are introduced followed by review of previous work on test generation, fault masking, online testing, and design for testability of reversible circuits. Our approach for online testing of reversible circuits is introduced in this chapter, which enables 100% coverage for single faults. A fault masking and diagnosis methodology is introduced to develop reversible majority voters with less than 38% area overhead. In addition, a test generation methodology which reduces required test information to be used in the field testing is also introduced.

Finally, Chapter 5 concludes the proposal. In this section briefly our achievements in reliable design for emerging technologies (robust design for crossbar nano-architectures as well as reliable reversible circuit design) are reviewed.
Chapter 2

Emerging Technologies for Computing

In the recent decades, various alternative implementation technologies based on new materials and devices as well as emerging fabrication processes have been proposed. On the other hand, alternative computational approaches have been investigated to overcome inherent limitations of the conventional computation approaches. In the following, the preliminaries on crossbar nano-architectures as an alternative implementation technology and reversible logic as an alternative computational approach (which are the scope of this research) are reviewed.

2.1 Crossbar Nano-Architectures

Exploiting Carbon Nano Tubes (CNTs) and semiconductor Nano Wires (NWs) fabrication based on bottom-up self-assembly offers the possibility of significantly denser circuits than current lithography-based manufacturing [28, 36, 37, 21]. In this technology, materials are created by chemical assembly, using methods such as Langmuir-Blodgett films, flow-based alignment, random assembly, and catalyzed growth [28].

Selectively doped semiconducting CNTs and NWs are orthogonally superimposed to construct fundamental electronic devices such as diode and FET [28, 38, 21, 30, 29]. Pro-
grammable interconnect, logic cores, and memory blocks are implemented by means of configurable junctions in two-dimensional crossed horizontal and vertical arrays of CNTs or NWs [14, 21, 39, 40, 41]. This two-dimensional grid structure with configurable switches and devices at the crosspoints (junctions) is known as crossbar. Cascaded crossbar blocks construct a crossbar array. Many proposed nano-architectures for nanoscale electronics have focused on this structure (crossbar) due to its simplicity for self-assembly based nanofabrication [21]. Figure 2.1 shows the general structure of a crossbar with activated and deactivated crosspoints as well as pull-up and pull-down voltage sources. The activated crosspoints refer to crosspoints which act as electrical devices (configured during logic mapping). Depending on the type of the doping of wires, a crosspoint shows different device behaviors (FET, diode, or resistor).

Figure 2.2: Suspension method to configure crosspoints [13]
One option for configuration of a crossbar is suspension. In this method, upper lines are suspended by periodic inorganic or organic supports, as shown in Figure 2.2. A crosspoint can be configured to one of two stable states (known as activated and deactivated states). Configuration is done by applying an appropriate voltage on crossed lines to set the crosspoint to the activated state by creating attractive van der Waals (vdW) energy between the lower line and the suspended upper line. A reverse voltage should be applied to release the energy between the two lines in order to set the deactivated state on the crosspoint [13].

Figure 2.3: Structure consists of p-FET, n-FET, and switch crossbar arrays

Based on materials used in crossbar lines, crosspoints can be configured as resistive interconnect, diode, and FET, as shown in Figure 2.3. If both upper (suspended) and lower lines are NW, then the crosspoint acts as a configurable switch. In order to construct configurable FET, lower line should be doped based on type of FET. Then, doped line is covered by oxide to prevent direct contact. Upper line is NW which is suspended by support and can be configured to activated or deactivated state. For activated state, the electrical field of upper wire can then be used to "gate" the other wire-locally evacuating a region of the doped SiNW of carriers to prevent conduction [42]. If both upper and lower lines are doped, one as p-type and the other as n-type, without using oxide as a cover the crosspoint acts as a diode. The device realization of various crossbars have been shown in Figure 2.4.
Using crossbar structure, various architectures have been proposed. The hybrid combination of CMOS and nanowire-based devices are used to propose CMOL, an architecture similar to Field Programmable Gate Array (FPGA) [43]. In this architecture, nano crossbars are used as storage elements, routing blocks, and wired-OR logic blocks. However, the most difficult functions (e.g. inversion, signal restoration, and demultiplexing) are moved into the CMOS layer. Field-programmable nanowire interconnect (FPNI) is an architecture which exploits the nano-crossbars as routing blocks, while the entire logic is implemented on CMOS layer [41].

FET-based crossbars are used in a complementary / symmetry architecture [44]. Each basic unit of this architecture consists of three types of crossbars: 1) p-FET crossbar, 2) n-FET crossbar, and 3) switching crossbar. This architecture can be used to implement AND-OR-INVERT functions similar to CMOS technology.

Various diode-based programmable crossbar architectures have been proposed [28, 14, 45]. Each basic unit of these architectures consists of a programmable diode crossbar to implement a logic function. These basic units are connected together to form a larger mesh of configurable elements. Using two-dimensional homogeneous nano structures, NanoFabric have been proposed [45]. The basic logic unit of this architecture is called nanoBlock. A nanoBlock is a diode-based crossbar used to implement diode-resistor logic. Dehon and et al. have proposed an architecture similar to Programmable Logic Array (PLA), called
nano-PLA [28], which uses diode-based crossbar to implement wired-OR logic.

Using diodes to implement logic functions necessitates restoration and inversion on data signals provided by logic blocks. In NanoFabric architecture, *Negative Differential Resistor* (NDR) is used to signal restoration and latching. Nano-PLA uses a set of nano-wire based FET devices to provide inversion and restoration on the signals between logic blocks. Furthermore, since diode is a passive element, its inputs/outputs must be precharged/discharged before each computation phase. Therefore, the operation of these architectures has a hysteric behavior between computation and reset phases. In these architectures, a set of global signals are used to control the timing of computation (evaluation) and reset phases. Figure 2.5 shows general structure of these architectures consisting of a set of cascaded blocks and the global signals to control timing of the blocks.

![Figure 2.5: Cascaded Diode-based crossbars with global control signal](image)

### 2.1.1 Nano-PLA

Dehon and et al. have proposed a nano-architecture similar to *Programmable Logic Array* (PLA), called nano-PLA [14]. In this nano-architecture, diode-based crossbar is used to implement wired-OR logic. Cascaded logic blocks of nano-PLA provide NOR-OR logic using selective inversion scheme at the inputs of each block. Nano-PLA uses a set of nano-wire based FET devices to provide inversion and restoration on the signals between logic blocks.

The correct operation of nano-PLA depends on the timing of its global signals as well as the computation delay (propagation delay) of logic blocks. If the computation delay of
a logic block exceeds a threshold value, then outputs will be ready while the succeeding block is in the reset phase and the data will not be captured. Figure 2.6 shows a block of nano-PLA consisting of a diode-based logic block with the FET-based restoration unit [28]. Here, the logic block acts as a wired-OR logic. In this figure, the restoration unit produces the inverted value of A and B. This circuit produces $\neg A + \neg B$ on $O_1$ as well as $\neg B$ on $O_2$. The operation of this block consists of two phases: a pre-charge phase followed by an evaluation phase [28]. These two phases are non-overlapping on a high value. During the pre-charge phase (high pre-charge signal), the output signals ($O_1$ and $O_2$) are discharged to low (strong 0). In this phase, inverted inputs ($\neg A$ and $\neg B$) are set to low (0). After return to low on the pre-charge signal and rise to high on the evaluation signal, the evaluation phase starts. During the evaluation phase, inverted inputs are evaluated based on input values, and the outputs of the block are determined.

![Figure 2.6: Circuit equivalent of a logic block of nano-PLA followed by restoration (inversion) unit [14]](image)

### 2.1.2 FET-Based Crossbar

FET-based crossbar can be used for logic realization in PLA-like crossbar nano-architectures [42, 28, 30, 46]. In these nano-architectures, a PLA block consists of FET-based devices to im-
implement logical NAND (NOR) functions. Please note that, in order to have a programmable structure, the FET devices must be *depletion mode* transistors [42, 28]. Cascaded blocks of such crossbars provide a set of NAND-NAND or NOR-NOR blocks similar to PLA structure. Figure 2.7.a shows a structure in realizing a logical NOR block. The circuit has a dynamic behavior in two phases. In the first phase, the precharge signal (Pre) is high. Therefore, in this phase the pull-down network discharges the outputs. Then, evaluation starts by switching the precharge signal to low. During the evaluation phase, the pull-down network is disconnected while the pull-up network can be connected (if all inputs are low). Unlike diode-based crossbars, the use of FET-based devices eliminates the need for voltage restoration on data signals. The blocks of FET-based crossbars can be cascaded in different structures. Figure 2.7.b shows a portion of FET-based crossbar in a cascaded nano-architecture.

![FET-based crossbar](image)

Figure 2.7: FET-based crossbar: (a) a FET-Based crossbar realizing NOR logic (b) cascaded blocks

### 2.2 Reversible Computation

Studies on reversible logic were initiated in 1980 [24], based on a thermodynamic theory. This theory states reversibility as the necessary condition for zero-energy computation [47].
Any logical irreversible computation is associated with a physical irreversible realization which requires a constant technology independent heat generation, per computation cycle [47]. Traditional digital circuits erase bit information every time they perform logic operations. These logic operations are therefore called irreversible. Every lost or duplicated bit information causes \( kT \ln 2 \) energy dissipation, where \( k \) is the Boltzmann’s constant and \( T \) is the temperature [32]. Therefore, reversible computation, which avoids the information loss, was introduced as a promising computational logic approach to reduce or even eliminate the power consumed by the computation. A reversible function is a function which supports the computation in the system in the both backward and forward directions (input to output and output to input directions). It must be noted, although, at current technology, energy losses due to irreversibility are negligible with respect to the overall power dissipation, but this may change for emerging non-CMOS technologies, such as quantum computing [33, 48].

Further research on reversible logic is motivated based on the fact that reversible logic is an inherent design requirement of quantum computation (known to solve NP hard problems in polynomial time) [33]. The operation of quantum logic gates must be based on elementary unitary matrix operations which are inherently reversible. Furthermore, reversible computing has application in many modern computational problems including cryptography, computer graphics, digital signal processing, and optical computing [49, 35].

In the reversible logic, data is bijectively transformed without losing any of the original information. Therefore, reversible logic is a Boolean function which provides a one to one mapping between inputs and outputs [33]. In other words, each input entry in truth table of a reversible module corresponds to a distinct output entry, which maps each input assignment to a unique output assignment [50]. A bijective logic function \( f : X \rightarrow Y \) from the set of inputs \( X = \{x_1, x_2, ..., x_n\} \) to the set of outputs \( Y = \{y_1, y_2, ..., y_m\} \) is a reversible function iff [51]:

1. The number of inputs of the function is equal to the number of its outputs (i.e. \( n = m \)).
2. That function is a one-to-one relation between the input permutations and the output permutations. In other words, each distinct input permutation is mapped to a unique output permutation, and vice versa.

A reversible circuit is realized from a reversible function, using a set of reversible gates. In a reversible circuit, each sub-circuit must be reversible circuit, too. Therefore, having fan-out on any internal wire of a reversible circuit is not allowed (fan-out on wire results in irreversibility of the corresponding sub-circuit). In addition, reversible circuits do not contain any state holder, feedback, and other conventional sequential elements [52].

However, reversible modules which duplicate an input in the output are used as the fan-out branches in a reversible circuit. Special techniques have also been proposed to implement sequential circuits satisfying reversibility characteristic of the circuit [52].

Reversible gates usually have more inputs/outputs compared to irreversible gates, due to the reversibility constraints. A gate with \( n \) inputs and \( n \) outputs is called \( n \times n \) reversible gate. Inputs of reversible gates are divided to two groups: Target inputs (\( T \)) and Control inputs (\( C \)). Outputs are also categorized as Garbage outputs (\( G \)) and Function outputs (\( F \)). Control inputs control logic operation of function outputs on target inputs. Garbage outputs are unused outputs which are added due to reversibility.

![Universal reversible gates](image)

Figure 2.8: Universal reversible gates: (a) NOT (b) Feynman (c) Toffoli (d) Fredkin

A set of universal reversible gates have been introduced. Commonly studied reversible
gates include: NOT, Feynman Gate, Toffoli Gate, and Fredkin Gate [53, 54, 55, 56, 57]. Figure 2.8 shows the schematic of these reversible gates. The NOT, Feynman Gate, and Toffoli Gate are also known as *K-Controlled-NOT (K-CNOT)*, where $K$ denotes number of control inputs. In a K-CNOT gate $F = T \oplus \prod_{i=1}^{K} C_i$ and for each $i$, $G_i = C_i$. In a Fredkin Gate for each $i$, $G_i = C_i$. If $\prod_{i=1}^{n} C_i$ (all $C_i$ are 1), then $F_1 = T_2$, and $F_2 = T_1$, otherwise $F_1 = T_1$, and $F_2 = T_2$. 
Chapter 3

Robust Design for Crossbar

Nano-Architectures

3.1 Previous Work

3.1.1 Background

Delay Model for Diode-Based Crossbar

Figure 3.1: A portion of diode-based crossbar: (a) the schematic of the circuit (b) the equivalent RC model of the circuit

A portion of a diode-based crossbar and its equivalent RC model are shown in Figure 3.1.
The delay of the circuit is determined during the evaluation phase. Since the output of the circuit is initialized to low during the precharge phase, its fall time is almost zero. However, the rise time of the output is defined as time delay between a rise on evaluation signal (fall on $\neg$Eval) to a rise on output signal. During the evaluation phase, precharge signal (Pre) is high ($\neg$Pre is low), and hence, transistors $T_3$ and $T_4$ are off (very high resistance), while transistors $T_1$ and $T_2$ are on. Using Elmore’s model, the delay of the portion of the circuit shown in Figure 3.1, can be calculate as the followings [58]:

$$
\tau_{Out} = (R_{wire} + R_{onT_1}) \cdot C_{T_1} \\
+ (R_{wire} + R_{onT_1} + R_{onT_2}) \cdot C_{T_2} \\
+ (R_{wire} + R_{onT_1} + R_{onT_2} + R_{D_1}) \cdot C_{D_1} \\
+ (R_{wire} + R_{onT_1} + R_{onT_2} + R_{D_1} + R_{Load}) \cdot C_{Load}
$$

Where, $C_{T_1}$ and $R_{onT_1}$ are the equivalent capacitance and resistance of $T_1$, respectively. $R_{D_1}$ and $C_{D_1}$ are the resistance and the capacitance of diode $D_1$ when it is switching. $R_{Load}$ is the equivalent resistance of the load. $C_{Load}$ is the capacitance of the load as well as the parasitic capacitance of the connected nanowires. Since, transistors $T_3$ and $T_4$ are off, they act as high impedance. Therefore, $R_{offT_3}$ and $R_{offT_4}$ do not contribute in the delay equation. Since all diodes are in parallel, in the worst case delay analysis, only one of the diodes is on to drive the output. For each individual diode and corresponding restoration unit, the same delay equation is valid in which the worst case delay is equal to the maximum delay.

**Delay Model for FET-Based Crossbar**

The schematic and the equivalent RC model of a column of a FET-based crossbar have been shown in Figure 3.2. During evaluation phase of this crossbar, Pre signal is high ($T_4$ is off). To have a rise transition on the output, all of the series transistors in the pull-up
network must be on. In this figure, $C_{T_i}$ denotes the equivalent capacitance of each FET device. Using the Elmore’s model, the delay of this circuit is expressed as the followings:

$$
\tau_{Out} = (R_{\text{Wire}} + R_{\text{on}T_0} + R_{\text{on}T_1}) \cdot C_{T_1} \\
+ (R_{\text{Wire}} + R_{\text{on}T_0} + R_{\text{on}T_1} + R_{\text{on}T_2}) \cdot C_{T_2} \\
+ (R_{\text{Wire}} + R_{\text{on}T_0} + R_{\text{on}T_1} + R_{\text{on}T_2} + R_{\text{on}T_3}) \cdot C_{T_3} \\
+ (R_{\text{Wire}} + R_{\text{on}T_0} + R_{\text{on}T_1} + R_{\text{on}T_2} + R_{\text{on}T_3} + R_{\text{Load}}) \cdot C_{\text{Load}}
$$

**Defect Model**

High defect rate is another challenge for any emerging technology. The effect of defective wires and crosspoints can be expressed as the followings.

- **Unprogrammable crosspoints:** A crosspoint loses its programmability because of insufficient electrons at the junction area (due to poor doping), and poor contacts [46].
• Breaks in NWs: Increase in the length of NW and axial stress may result in broken NWs [59]. Break in NW results in defective crosspoint which are followed by that wire.

Defective crosspoints have the following behaviors:

• Stuck-short: Device at the crosspoint (diode or FET) is permanently on (conducting), independent of the input which drives that device.

• Stuck-open: Device at the crosspoint (diode or FET) is permanently off (non-conducting), independent of the input which drives that device. This defect can be translated as no active device at the crosspoint.

Figure 3.3 shows possible defects and their effects on diode-based and FET-based crossbar nano-architectures. As seen in this figure, all defects (nanowire break, and crosspoint stuck open/short) can be mapped to crosspoint defects [46, 59].

3.1.2 Defect Tolerant Techniques

An application-independent defect tolerant design flow has been proposed [60]. In this flow, higher level of the design process are unaware about defect locations, while defect consideration is done at logic mapping level, by using a set of recursive and greedy algorithms.
Defect tolerant techniques for crossbar nano-architectures take advantage of abundance of resources to introduce redundancy in the logic mapping [61]. The logic mapping problem and the problem of finding the largest square sub-crossbar with no defects have been shown to be NP-hard [62]. Therefore, the proposed defect tolerant logic mapping techniques include heuristic algorithms during mapping to avoid defective crosspoints in logic mapping. For example, defect-aware mapping is modeled as a bipartite graph problem [59].

\[
\begin{align*}
O_1 &= I_1 + I_2 + I_3 \\
O_2 &= I_2 + I_3 \\
O_3 &= I_1 + I_3 + I_4
\end{align*}
\]

Figure 3.4: A set of functions and the corresponding graph representation

Problem of defect-free mapping is translated to search monomorphism in a graph [63]. In this approach, the Boolean function is converted to a graph. The nodes of this graph represent the inputs and outputs of the Boolean function. The edges of the graph represent input/output relations. Figure 3.4 shows a function and the corresponding graph representation. On the other hand, the crossbar is represented by another graph. In this graph, inputs and outputs of the crossbar are represented by nodes. There is an edge between an input and an output node if and only if the crosspoint constructed by the corresponding horizontal and vertical wires is defect-free. Figure 3.5 shows a crossbar and the corresponding graph representation. Using these graphs the problem of defect tolerant mapping is translated as searching for a graph monomorphism in the crossbar graph. This problem is finding a sub-graph in crossbar graph such a way that it matches the Boolean function graph, which is a classical NP-complete problem [64]. Such a method based on searching monomorphism in graphs is not efficient for large circuits.

A SAT-based defect-aware logic mapping framework has been introduced [27]. In this
approach, the mapping problem is converted to a SAT problem. Solving the SAT problem, the method indicates whether there can be a defect-free mapping on the crossbar or not. Using ILP formulations, a logic mapping method is introduced in [65]. These formulations provide a defect-free mapping (if possible) for logic function on crossbar nano-architecture. The order of complexity of the variables and the constraints of these formulations are $O(n^2m^2)$, where $n$ and $m$ are the number of rows and columns of the crossbar, respectively.

### 3.1.3 Variation Tolerant Techniques

The performance of the nanowire and nanotube based FET devices under process variation is analyzed in [66, 66]. It is shown that nanowire and nanotube based FETs are less sensitive to process variations compared to CMOS and FinFET counterparts. However, in this comparison it is assumed that all these three technologies follow the same distribution in parameter variations, without consideration of self-assembly fabrication. On the other hand, although it is shown that nanowire and nanotube based FETs are less sensitive to process variations, but still for the $3\sigma = 30\%$ of the normal distribution on process parameters (which is shown as reasonable distribution), the normal distribution of delay follows a normal distribution with $3\sigma = 30\%$.

Considering process variation, a method to reduce the net range of path delays is introduced [58]. This method, which is a post-fabrication mapping methodology, matches the fanout of logical nets with physical transistor threshold voltages to tolerate the effect of variation on physical threshold voltages of devices. This method, which is based on a greedy
algorithm, tries to match a high-fanout product term with a low $R_{off}FET$ NAND-term.

*Simulated Annealing (SA)* algorithm is used for variation and defect tolerant mapping on a crossbar [67]. In this method, mapping on vertical and horizontal lines are swapped to change variation cost of a crossbar. On-the-fly mapping without any pre-characterization has been introduced in [68]. This method enables built-in self-mapping capability.

Table 3.1 summarizes different methods in achieving defect and variation tolerance.

Table 3.1: Comparison of different methods at various abstraction levels (Level) in achieving Variation Tolerance (VT) and Defect Tolerance (DT)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>VT</th>
<th>DT</th>
<th>Level</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Independent + Recursive &amp; Greedy</td>
<td>No</td>
<td>Yes</td>
<td>Multi</td>
<td>[60]</td>
</tr>
<tr>
<td>Logic Duplication</td>
<td>No</td>
<td>Yes</td>
<td>Logic</td>
<td>[61]</td>
</tr>
<tr>
<td>Greedy</td>
<td>Yes</td>
<td>Yes</td>
<td>Mapping</td>
<td>[58]</td>
</tr>
<tr>
<td>Searching Monomorphism in Graph</td>
<td>No</td>
<td>Yes</td>
<td>Mapping</td>
<td>[63]</td>
</tr>
<tr>
<td>SAT-based</td>
<td>No</td>
<td>Yes</td>
<td>Mapping</td>
<td>[27]</td>
</tr>
<tr>
<td>Greedy</td>
<td>No</td>
<td>Yes</td>
<td>Mapping</td>
<td>[59]</td>
</tr>
<tr>
<td>SA</td>
<td>Yes</td>
<td>Yes</td>
<td>Mapping</td>
<td>[67]</td>
</tr>
<tr>
<td>ILP-based</td>
<td>No</td>
<td>Yes</td>
<td>Mapping</td>
<td>[65]</td>
</tr>
</tbody>
</table>

3.2 Proposed Approaches

In terms of crossbar nano-architectures, we study two approaches, namely logic mapping and architectural techniques, to incorporate variation and defect tolerance in crossbar nano-architectures [1, 2, 3, 4, 5, 6, 7]. In the logic mapping approach, different configurations of a logic function on a crossbar nano-architecture are explored to find a reliable configuration which results in better variation and defect tolerance. We also use asynchronous design methodologies to propose a self-time crossbar nano-architecture, which allows us to eliminate global clock-like signals (by replacing with local handshake signals) to reduce circuit vulnerability to delay variation. In the follows, a sub-set of these approaches will be introduced.
3.2.1 Architectural Approach

Using nano-PLA for logic implementation in nano-architectures suffers from a set of challenges. One issue is nondeterministic values on buffered signals during pre-charge phase. In addition, this architecture is highly vulnerable to variations on switching delay of diodes at logic block. It is due to the fact that all valid outputs of a logic block must be ready before the evaluation phase of the successor blocks. On the other hand, the logic mappings on this architecture assumes triggers on evaluation and pre-charge signals occur simultaneously for all of the crossbars, which is not easy to control. Also, per-charge and evaluation signals must be routed to entire of the fabric, which requires micro wiring in fabrication. Therefore, it seems using a self-timed method, which locally controls timing, is a promising method to deal with these challenges in this emerging technology.

![Figure 3.6: Structure of a block of self-timed nano-PLA](image)

We propose self-timed nano-PLA architecture which consists of multiple cascaded blocks [1]. In each block logical computations on data signals are done in diode-based PLA-like logic units. Figure 3.6 shows the structure of a block in this architecture. Each block
consists of three major parts:

1. **Restoration/Control unit.** This unit is constructed from non-programmable FET-based devices. Pre-charging and evaluation phases are controlled by this unit. It also restores signals generated by the diode-based block at the previous stage.

2. **AND-Plane.** A programmable diode-based crossbar is used to implement logical AND operation on the inputs (i.e. to generate the product terms).

3. **OR-Plane.** Another programmable diode-based crossbar which is fed by the outputs of AND-Plane is used to produce OR function on the product terms (sum of products).

Figure 3.7 shows the general structure of AND-Plane and OR-Plane.

![Hantex](image.png)

**Figure 3.7:** Structure of logic computation unit: (a) Crossbar structure to implement AND logic (b) Crossbar structure to implement OR logic

**Sequence of Operations**

Since diode devices are passive elements, therefore before each logic computation the diodes in the logic block must be initialized to appropriate states. In the AND-Plane before each computation, the horizontal and vertical lines (inputs and product terms) must
be initialized to $V_{DD}$. On the other hand, all lines at the OR-Plane must be initialized to $GND$. After this initialization, computation can be done by the logic block. These initializations and computations (pre-charges and evaluations) are done in three major steps, which determine the sequence of operations of a logic block. These three steps are as the followings:

1. AND-Plane pre-charging: In this phase the inputs of AND-Plane must be $V_{DD}$. In addition to this, vertical lines of the AND-Plane must be connected to $V_{DD}$ through upper FETs. Furthermore, the AND-Plane must be isolated from the subsequent OR-Plane through FETs between these two planes.

2. AND-Plane evaluation & OR-Plane pre-charging: In this phase, still AND-Plane and OR-Plane are isolated from each other. But the inputs of the AND-Plane are calculated corresponding to the inputs of restoration/control unit. In addition to this, vertical lines of the AND-Plane must be isolated from $V_{DD}$ by turning off the corresponding FETs. Since in this phase the AND-Plane is isolated from the OR-Plane, during the evaluation of the AND-Plane, the OR-Plane can be pre-charged. Pre-charging of the OR-Plane is done by connecting its inputs (product lines) and its outputs to $GND$.

3. OR-Plane evaluation: In this phase, the outputs of the OR-Plane are calculated. Therefore, the connection between the AND-Plane and the OR-Plane is established through the FETs between these two planes. However, the connection between the OR-Plane and $GND$ must be stopped by applying appropriate values to the gates of the corresponding FETs.

In order to distinguish valid and invalid values on data signals, we have used dual-rail coding to represent data signals. In this coding scheme, each data bit is coded on two signals ($D_0D_1$). A valid data is represented by $D_0D_1 = 10$ or $D_0D_1 = 01$. If both these signals are 0 or both of them are 1, then the data is empty or invalid, respectively.
This coding scheme is represented in Table 3.2. Invalid and empty data are separated to distinguish pre-charge value from faulty ones.

<table>
<thead>
<tr>
<th>$D_1$</th>
<th>$D_0$</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Empty data</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Logical 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Logical 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Invalid data</td>
</tr>
</tbody>
</table>

Furthermore, there is a signal (acknowledgment signal) between each pair of subsequent blocks. This signal acknowledges the completion of the computation on data for the successor block. Using the acknowledgment signal as well as dual-rail coding, the three major operation steps of a self-timed nano-PLA logic block are divided to 6 sub-phases. The order of these sub-phases is as the followings:

1. Reset AND-Plane; The control unit waits for empty inputs to reset the inputs of the AND-Plane. However, to reset these signals the acknowledge signal between this block and the successor block (called $O_{Ack}$) must be 0. It means that the block must ensure that the successor block has used its previous data. Therefore, this block waits for the completion of the computation in the successor block. In this phase, $C_A = 0$ and all inputs of the AND-Plane are connected to $V_{DD}$ through restoration unit. In addition to this, $C_i = 1$ to isolate the AND-Plane from the OR-Plane.

2. Reset OR-Plane; Since $O_{Ack} = 0$, the block resets the OR-Plane by forcing $C_O$ to 1.

3. Acknowledge previous block; After resetting the AND-Plane, the control unit acknowledges the previous block by placing 1 on the acknowledge signal (called $I_{Ack}$) which is between these two blocks. This signal informs the previous block that this block can accept new valid data.

4. Wait for valid inputs; After capturing 1 on $I_{Ack}$ by the previous block, it can generate valid data. Control unit waits for these valid data to generate valid inputs for the
AND-Plane. After all inputs are valid, the control unit changes $C_A$ to 1 and calculates the inputs of the AND-Plane.

5. OR-Plane computations; The block waits for 1 on $O_{Ack}$ as well as computation completion of the AND-Plane to start the computation of OR-Plane. In this phase, $C_A = 1$, $C_I = 0$ and $C_O = 0$.

6. Acknowledge the previous block; After completion of the computation, the control unit acknowledges the previous block by changing $I_{Ack}$ to 0. Now, the previous block can generate empty data to start a new round of computation.

Implementation

Implementation of logic function on self-timed nano-PLA can be divided to implementation of two units: 1) the logic unit, and 2) the control and restoration unit. Control and restoration unit has a fixed application-independent implementation, which must be done during fabrication. However, the implementation of the logic unit is flexible and depends on the logic function, which is reconfigurable.

Logic Unit Implementation

Dual-rail mapping on logic blocks can be achieved by using conventional PLA synthesis tools. However, these tools are developed for single-rail logics. Therefore, following steps are used for dual-rail conversion.

- Logic decomposition; Logic functions which are described in *Berkeley Logic Interchange Format* (blif) format [69], must be converted to custom-sized PLA blocks. PLAMAP function of RASP synthesis tool [70] can be used to convert logic function to custom-sized cascaded PLA blocks.

- Truth table generation; Each PLA block produced by PLAMAP is in blif format for single-rail logic. Therefore, these files must be converted to truth table to be modified and simplified by logic optimization tool.
• Dual-rail conversion; In the truth tables, a dual output is added for each output. For each entry in which the output is 1, its dual output is 0. For the remaining entries, the dual output is 1.

• Logic minimization; The truth tables for dual-rail functions are not optimized. Therefore, a conventional logic optimization tool is used to optimize the truth tables. We use ESPRESSO [71] logic optimization tool to optimize the dual-rail truth tables.

• Dual-rail mapping matrices; Configuration of a logical block is determined by a matrix called Mapping Matrix (MM). This matrix is defined as follow:

\[
MM_{i,j} = \begin{cases} 
1, & \text{if diode at position } i, j \text{ is activated} \\
0, & \text{otherwise} 
\end{cases}
\]

Each logic block is determined by two MMs: AMM and OMM. AMM determines the configuration of the AND-Plane and OMM determines the configuration of the OR-Plane. The rows of an AMM are the inputs of logic block, while rows of an OMM are the outputs of that logic block. For both of these matrices, columns are the product terms.

Outputs of optimized truth tables are dual-rail outputs, while the inputs are still single-rail. These optimized tables must be converted to AMMs and OMMs. In this conversion, a dual input is added for each input. An input may have three values in product terms: 1, 0, and \(d\) (don’t care). If an input is 1 for a product term, there will be a 1 at the row of that input and the column of that product term in the AMM entry, while the entry at the row of the dual of that input and the column of that product term is 0. The situation is similar for 0 entries. If the input is \(d\), both of the entries will be 0. Since, outputs are dual-rail, therefore the OMMs is directly generated from the truth tables without any modifications.

**Restoration/Control Unit Implementation**

The implementation of each component of restoration and control unit is done as follows:
• AND-Plane input reset and calculation unit: An input is reset to 1, if the both corresponding data signals are 0 (empty), and $O_{Ack}$ is 0. The pull-up network of Figure 3.8.a shows the implementation of this condition. The pull-down networks of these circuits implement input calculation of the AND-Plane. During the reset phase, both signals which are the inputs to the AND-Plane ($I_0$ and $I_1$) are set to 1. If the inputs to the block ($D_0$ and $D_1$) are changed from empty to valid, the inputs to the AND-Plane are calculated. Assuming $D_0D_1$ changes from empty (00) to 10, then pull down network of $I_1$ will be connected and $I_1$ changes to 0. However, since $D_1$ is 0, therefore $I_0$ remains at 1.

• AND-Plane reset; AND-Plane is reset if the inputs are empty and $O_{Ack} = 0$, otherwise it will be in the computation mode. Figure 3.8.b shows the implementation of AND-Plane reset unit.

• OR-Plane reset; OR-Plane is reset if $O_{Ack} = 0$, otherwise it must be at the computation mode ($C_O = \neg O_{Ack}$). In this state, AND-plane can not accept new data.

• AND-Plane/OR-Plane isolation; AND-Plane and OR-Plane must be isolated when at least one of them is in the reset mode (i.e. if $C_A = 0$ or $C_O = 1$), otherwise these two planes must not be isolated. Since, $C_O = \neg O_{Ack}$, therefore, the isolation control is $C_I = \neg(C_A.O_{Ack})$. 

Figure 3.8: A part of control implementation: (a) implementation of AND-Plane inputs (b) implementation of AND-Plane reset (c) implementation of $I_{Ack}$ generator
• Input acknowledgment signal; $I_{ack}$ is 1 if the AND-Plane is reset ($C_A = 0$). When the outputs of the OR-Plane are valid (computation is completed) and AND-Plane is not in reset mode, then $I_{ack}$ returns to 0. The implementation of $I_{ack}$ control unit has been shown in Figure 3.8.c.

Figure 3.9: Two cascaded blocks of self-timed nano-PLA on 2D crossbars for logic and routing units, and lithography based fabricated FET devices

Discussion

Using lithography based fabrication to manufacture FET-based devices, logic and routing units can fit in high density two dimensional (2D) regular crossbar structure (fabricated by self-assembly). Two cascaded blocks of self-timed nano-PLA in 2D structure are shown in Figure 3.9. As it is seen in the figure, a set of signals control FET devices are placed between logic units and pull-up and pull-down networks ($C_A$, $C_I$, and $C_O$). Since these signals are carried by nano-wires, they can drive a limited number of FET devices. To drive more FET devices, these nano-wire can be replaced by micro-wires or alternatively, each signal can be carried by multiple nano-wires (each nano-wire controls a subset of FET devices).

It must be noted, self-timed nano-PLA is designed based on dynamic logic, similar to previous work [14, 28]. Although using dynamic logic has drawbacks in terms of noise immunity and robustness, however, it is necessitated by the use of diode-based devices.
in logic implementation. Since diode devices are passive elements, they must be reset periodically before each computation.

**Simulation Results**

A subset of MCNC benchmarks are synthesized using PLAMAP function of RASP logic synthesis toolset [72]. It converts a logic circuit to custom-sized multi-stage PLAs. In defining the size of PLAs, we assumed crossbar array of $16 \times 16$ crossbars (16 inputs and 16 outputs) for both nano-PLA and self-timed nano-PLA. Considering this assumption, the benchmark circuits are mapped on nano-PLA and self-timed nano-PLA as follows.

Since nano-PLA contains only OR-Planes (some of the inputs can be inverted to produce NOR logic of the previous block), AND-Plane and OR-Plane of each PLA produced by PLAMAP must be separated to be mapped on two blocks. The product terms of an AND-Plane are the inputs to the OR-Plane implemented on the successor block. The AND-Plane of the PLA blocks must be converted to NOR logic. This can be done by inverting the inputs. It means that if an input is used as 1 in a product term, it must be converted to 0 in that product term. Then, these two planes are optimized by using ESPRESSO (logic minimization tool).

It must be noted, if an input and its complement are used in a product term, then that input must be mapped on two input lines of the corresponding block. One of these lines generates the buffered input to the block, and the other line generates inverted input. This is due to the fact that nano-PLA supports one of the buffered or inverted inputs at each line. Since, we have assumed all logic blocks have 16 rows (16 inputs), therefore 8 rows of the input lines of a block are reserved for the buffered inputs and the other 8 rows are reserved for the inverted inputs. Therefore, in PLAMAP the number of inputs for each PLA must be set to 8. In other word, the number of inputs of PLAs must be set to the half of inputs of a block to support both inverted and buffered inputs. However, the number of product terms and outputs are set to 16.

Mapping matrices of logic blocks are generated by the introduced dual rail mapping
method. Since in self-timed nano-PLA, inputs and outputs are dual rail, in PLAMAP the input and output sizes must be set to 8. But, the number of product terms is set to 16.

It must be noted, after mapping the PLA blocks on these architectures, only a subset of lines are used to map inputs or outputs. For example, in Table 3.3 \((I, O, P)\) shows the average number of activated inputs, product terms, and outputs for each benchmark mapping onto self-timed nano-PLA. In this table, \((I, O)\) shows the average number of inputs and outputs used in the mappings onto nano-PLA.

The simulation results (shown in Table 3.3) compares the proposed architecture with original nano-PLA \([28]\) in terms of area and variation tolerance.

Table 3.3: Simulation results on a set of benchmarks. Total number of FET devices \((FET)\), total number of Diode devices \((D)\), average number of activated inputs \((I)\), products \((P)\), and outputs \((O)\) per block, number of stages \((S)\), variation tolerance \((VT)\), the improvements, and overheads

<table>
<thead>
<tr>
<th>Circuit</th>
<th>FET</th>
<th>D</th>
<th>((I,O))</th>
<th>S</th>
<th>VT</th>
<th>((I,P,O))</th>
<th>S</th>
<th>VT</th>
<th>Improvement</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>18432</td>
<td>8837</td>
<td>((8,6))</td>
<td>156</td>
<td>38%</td>
<td>18720</td>
<td>11659</td>
<td>((14,10,4))</td>
<td>78</td>
<td>-1%</td>
</tr>
<tr>
<td>b9</td>
<td>3328</td>
<td>1388</td>
<td>((7,5))</td>
<td>26</td>
<td>46%</td>
<td>3120</td>
<td>1745</td>
<td>((14,8,5))</td>
<td>13</td>
<td>6%</td>
</tr>
<tr>
<td>C1355</td>
<td>6912</td>
<td>3543</td>
<td>((8,8))</td>
<td>54</td>
<td>46%</td>
<td>7200</td>
<td>4714</td>
<td>((12,12,6))</td>
<td>30</td>
<td>-4%</td>
</tr>
<tr>
<td>duke2</td>
<td>15872</td>
<td>4795</td>
<td>((7,4))</td>
<td>128</td>
<td>25%</td>
<td>15900</td>
<td>6917</td>
<td>((15,7,3))</td>
<td>64</td>
<td>3%</td>
</tr>
<tr>
<td>exp5</td>
<td>10890</td>
<td>5295</td>
<td>((9,8))</td>
<td>60</td>
<td>43%</td>
<td>9840</td>
<td>7915</td>
<td>((13,13,4))</td>
<td>41</td>
<td>-28%</td>
</tr>
<tr>
<td>k2</td>
<td>32256</td>
<td>10210</td>
<td>((9,5))</td>
<td>252</td>
<td>37%</td>
<td>28080</td>
<td>15380</td>
<td>((15,7,7))</td>
<td>117</td>
<td>13%</td>
</tr>
<tr>
<td>rd84</td>
<td>7424</td>
<td>849</td>
<td>((3,2))</td>
<td>58</td>
<td>26%</td>
<td>6960</td>
<td>4154</td>
<td>((15,12,3))</td>
<td>29</td>
<td>6%</td>
</tr>
<tr>
<td>t481</td>
<td>20992</td>
<td>5808</td>
<td>((7,4))</td>
<td>164</td>
<td>18%</td>
<td>18480</td>
<td>10482</td>
<td>((15,8,4))</td>
<td>77</td>
<td>12%</td>
</tr>
<tr>
<td>term1</td>
<td>3120</td>
<td>2148</td>
<td>((8,5))</td>
<td>40</td>
<td>41%</td>
<td>3040</td>
<td>2850</td>
<td>((14,9,5))</td>
<td>21</td>
<td>1%</td>
</tr>
<tr>
<td>termm</td>
<td>5888</td>
<td>2355</td>
<td>((6,5))</td>
<td>46</td>
<td>43%</td>
<td>5520</td>
<td>3228</td>
<td>((14,9,5))</td>
<td>23</td>
<td>6%</td>
</tr>
<tr>
<td>vda</td>
<td>12544</td>
<td>5376</td>
<td>((9,6))</td>
<td>98</td>
<td>37%</td>
<td>12000</td>
<td>7669</td>
<td>((15,9,6))</td>
<td>50</td>
<td>4%</td>
</tr>
<tr>
<td>x4</td>
<td>13824</td>
<td>4100</td>
<td>((8,4))</td>
<td>108</td>
<td>39%</td>
<td>10880</td>
<td>5288</td>
<td>((15,7,3))</td>
<td>42</td>
<td>27%</td>
</tr>
<tr>
<td>Average</td>
<td>14421</td>
<td>4359</td>
<td>((8,8))</td>
<td>99</td>
<td>37%</td>
<td>11700</td>
<td>6848</td>
<td>((15,10,6))</td>
<td>49</td>
<td>18%</td>
</tr>
</tbody>
</table>

We have considered same-sized blocks (crossbars) for both architectures. Therefore, the number of stages and the number of FET devices indicates the difference of these two architectures in terms of area. The number of stages depends on capability of the architectures in providing AND-Plane and OR-Plane, as well as initial settings for logic synthesis, based on the limitations of the architectures.

As can be seen in Table 3.3, the number of stages for mapping on self-timed nano-PLA is considerably lower than the number of stages required for mapping on original nano-PLA. It is due to the fact that, in the mapping on nano-PLA each of the AND-Plane and the OR-Plane must be mapped on two separate nano-PLA blocks, while for implementation on
self-timed nano-PLA, both planes are mapped on the same block. However, the number of stages of nano-PLAs is not exactly twice as the number of stages of self-time nano-PLA, because the setting for their synthesis is not the same. The improvement on the number of stages achieved by self-timed nano-PLA has been shown in the table.

The number of FET devices depends on the size of logic blocks (the number of inputs and outputs), the number of FET devices per block, and the number of stages. The number of FET devices per block in self-timed nano-PLA is more than that in nano-PLA. However, due to fewer stages in self-time nano-PLA, the total number of required FET devices for entire circuit mapped onto self-timed nano-PLA is 18% (in average) less than that for nano-PLA. The required FET devices and the improvements achieved by self-timed nano-PLA have been shown in the table. For the column for improvements on FET, the negative percentages indicate where self-timed nano-PLA leads to more FETs compared to nano-PLA.

This table also shows the total number of activated diodes in each architecture. Since, in self-time nano-PLA the dual of outputs must be computed, the number of activated diodes in self-timed architecture is more than that for nano-PLA architecture. However, since the size of crossbars are same for both architectures, this does not translate to actual area overhead. This means better crosspoint utilization in self-timed nano-PLA. Nevertheless, it must be noted that more crosspoint utilization (more activated diodes) results in higher power consumption.

Extensive Monte Carlo simulations for delay variations have been done to estimate variation immunity of the benchmark circuits implemented on the architectures. For these simulations, the average delay of the blocks of nano-PLA in 1000 simulation runs is used to determine the periods of pre-charge and evaluation signals. Then, if the delay of a block exceeds the period of evaluation signal, that block fails in the timing constraint of nano-PLA. The average number of the cases where nano-PLA can tolerate variation over the total number of cases, variation tolerance, is shown in $VT$ column. For all of these simulations, our propose self-timed architecture can achieve 100% variation tolerance compared to only
37% in the original nano-PLA architecture.

### 3.2.2 Post Synthesis Transformations

The mapping of a logic function into a crossbar array (aka configuration), i.e. which crosspoints are used (activated) and which are deactivated, has a considerable impact on the delay as well as fault-free functionality (i.e. not using defective devices) of the mapped circuit. Here, we present a set of logic transformations, which preserves the logic functionality, while changing the way crossbar resources are used for the implementation of that function [4, 6, 7]. Some of these transformations are local (intra block), meaning that they preserve the functionality of the portion of the logic function mapped into a block, while modifying the configuration of that block. On the other hand, global transformations (inter block) may modify the portion of the logic function mapped into different logic blocks while preserving the entire logic functionality of the circuit.

#### Intra Block Transformations

**Swapping**

In the swapping transformation, the configurations of two rows or two columns of a crossbar are swapped \((A + B = B + A)\). This transformation is used as the basic operation in variety of algorithms (e.g. [67, 65, 73, 74, 75, 76]) . Swapping results in cascading changes, i.e. it introduces constraints outside of the logic block, by forcing the switching crossbars, both at the input and the output of this logic block, to route in a specific order. Since swapping changes the order of the inputs, such routing order imposed by swapping may be in contradiction with delay improvements achieved by routing algorithms.

**Input Duplication**

The crossbar array is a regular architecture, and the crossbar itself is a complete structure (there is a crosspoint at every intersection, in contrast to FPGA). Therefore, during logic mapping there are always unused input rows. These unused input rows can be used to
duplicate some of the inputs [76]. Input duplication preserve the functionality, because $A + B = A + A + B$. Similar to swapping, duplication results in cascading changes (to the preceding switch blocks). Any duplication on input lines must be done by switching crossbars. This increases the number of used paths in switching crossbar, adding to the routing complexity. In other words, the duplication method introduces extra constraints on routing in the form of additional paths.

**Output Decomposition**

In wired-OR logic, an output can be decomposed into two or more parts, each mapped into a separate vertical line. However, all these lines must be wired to construct the output. This transformation does not change the functionality, because $(A + B + C + D) = (A + B) + (C + D)$ [6]. Unlike the other two intra block transformations, all changes done by output decomposition are non-cascading, i.e. all modifications introduced by decomposition are done inside the logic block. Since all decomposed vertical lines are wired at the logic block, output decomposition does not introduce any extra routing constraints. Therefore, unlike the other two intra block transformations, output decomposition can be done after any optimizations applied at routing crossbars.

**Inter Block Transformations**

In an inter block transformation, the configuration of a block is modified to be distributed between two blocks. In this transformation, the entire logic function mapped into a block, or a portion of that is decomposed into two sets of functions each mapped into a separate crossbar. In the follows we propose two inter block transformations: 1) function decomposition, and 2) block decomposition.

**Function Decomposition**

A function decomposition transformation decomposes the function of some of the outputs into sub-functions, where the sub-functions together form the original function of the decom-
posed outputs. If $O_i = i_1 + i_2 + \ldots + i_m$ is an output of a block, then it can be decomposed into two sub-functions $O_1^i$ and $O_2^i$, where $O_1^i = i_1 + i_2 + \ldots + i_t$ and $O_2^i = O_1^i + i_{t+1} + \ldots + i_m$ (for any $t < m$). Since the blocks of nano-PLA produce OR function, function decomposition does not affect the functionality of $O_i (O_i = O_2^i)$.

The function decomposition transformation reduces the complexity of logic mapping into a block (the block which produces $O_1^i$), by reducing the complexity of $O_i$ to $O_1^i$. However, it adds to the complexity of the block which produces $O_2^i$, in terms of an extra output and a set of extra inputs. It also introduces extra complexity to routing crossbar to route extra lines for the decomposed outputs.

**Block Decomposition**

In block decomposition transformation, a subset of the output functions generated by a block $B_i$ is mapped into a new block $B_j$. So if $B_i$ originally implements $n$ outputs, it will implement $m$ ($m < n$) outputs and the remaining $n - m$ outputs are implemented by another block.

Block decomposition introduces area overhead in terms of an extra logic block and the corresponding routing block. However, since block decomposition reduces the number of the outputs of each block, it reduces the mapping complexity of that block.

Figure 3.10.a shows the configurations of two logic blocks of a nano-PLA architecture. For the sake of simplicity, the restoration units as well as routing crossbars are not shown in this figure. These configurations correspond to the mapping of two logic functions ($O_3 = I_1 + I_3$ and $O_4 = I_1 + I_2 + I_3 + I_4$). Using a set of intra and inter block logic transformations, the configurations are modified in Figure 3.10.b. The intra block transformations include the swapping of $I_1$ and $I_2$ (block 1), the duplication of $I_3$ over two rows (block 1), and the output decomposition of $O_4$ (block 2). Using the function decomposition transformation $O_2$ is decomposed into two sub-functions, one of them ($O_2^1 = I_2 + I_3$) is implemented on block 1, and another sub-function ($O_4 = O_2^1 + I_4$) is implemented on block 2.
3.2.3 Post Synthesis Methodologies and Algorithms

The proposed reliable mapping algorithms categorized to two groups: 1) local mapping (intra block mapping algorithms) and 2) global mapping (inter block mapping algorithms). The local mapping algorithms use intra block transformations to provide a reliable mapping for each nano-PLA block. If the local mapping fails in providing reliable mapping for a block, then that block is decomposed into two nano-PLA blocks using inter block transformations (global mapping).

Definitions

A logic function and a crossbar can be represented by a set of matrices which are defined in the follows.

- The binary Function Matrix (FM) indicates the logic function to be mapped into a crossbar. Rows and columns of an FM are inputs and outputs of the crossbar, respectively. The entries of FM are defined as below:

\[
FM[i][j] = \begin{cases} 
1, & \text{if input } i \text{ used in function at column } j \\
0, & \text{otherwise}
\end{cases}
\]
• **Mapping Matrix (MM)**: Configuration of a crossbar is determined by a matrix called *Mapping Matrix (MM)*. This matrix is defined as follows:

\[
MM[i][j] = \begin{cases} 
1, & \text{if crosspoint at position } i, j \text{ is activated} \\
0, & \text{otherwise}
\end{cases}
\]

• **Variation Matrix (VM)**: Deviations on switching delay of crosspoints in a crossbar are determined by a matrix called *Variation Matrix (VM)*. In this matrix, delay variation of a defect-free crosspoint is denoted by its delay value (an arbitrary value). The entries of this matrix are extracted by using delay test [67, 68]. Defective crosspoints are denoted by infinite value in corresponding elements in VM.

\[
VM[i][j] = \begin{cases} 
\text{Delay variation of defect-free crosspoint at position } i, j \\
\infty, & \text{defective crosspoint}
\end{cases}
\]

• **Defect Matrix (DM)**: Defective crosspoints in a crossbar are determined by a matrix called *Defect Matrix (DM)*. This matrix is defined as follows:

\[
DM[i][j] = \begin{cases} 
1, & \text{if crosspoint at position } i, j \text{ is Defective} \\
0, & \text{otherwise}
\end{cases}
\]

MM is determined by logic mapping of FM. Delay variation on outputs of a crossbar as well as the correctness of the logic function mapped into the crossbar (with respect to defective crosspoints) depend on MM, which defines the configuration of the crossbar.

### ILP Formulations

Using swapping transformation, we also have introduced *Integer Linear Programming* (ILP) formulations which result in efficient MM for the crossbar in terms of both variation and defect tolerance [5]. The descriptions of the notations used in the proposed ILP formulations are as follows (the number of columns and rows of an MM are \(m\) and \(n\), respectively):

• Swapping of columns:
∀ \( i, j \) \( 1 \leq i, j \leq m \)
\[
C_{i,j} = \begin{cases} 
1, & \text{if column } i \text{ of } FM \text{ is placed at column } j \text{ in } MM \\
0, & \text{otherwise}
\end{cases}
\]

- Swapping of Rows:
∀ \( i, j \) \( 1 \leq i, j \leq n \)
\[
R_{i,j} = \begin{cases} 
1, & \text{if row } i \text{ of } FM \text{ is placed at row } j \text{ in } MM \\
0, & \text{otherwise}
\end{cases}
\]

- Movement of an entry:
∀ \( i, j, s, t \) \( 1 \leq i, s \leq n, 1 \leq j, t \leq m \)
\[
E_{s,t,i,j} = \begin{cases} 
1, & \text{if entry } s, t \text{ of } FM \text{ is placed at position } i, j \text{ in } MM \\
0, & \text{otherwise}
\end{cases}
\]

- Delay on each output:
∀ \( j \) \( 1 \leq j \leq m \)
\[
\tau_{Oj} = \text{Delay of output at vertical line } j
\]

**Objective Function**

The goal is to minimize the maximum delay on outputs of the crossbar. Therefore, the objective function is defined as

\[
\text{Minimize: } \tau_{MO}
\] (3.1)

**Constraints**

Introduced constraints are as follows:

- Objective-function computation: The objective is minimizing delay of all outputs. Therefore, the objective function can be computed by the following constraint:
∀ \( j \) \( 1 \leq j \leq m \)
\[
\tau_{MO} \geq \tau_{Oj}
\] (3.2)
• Delay of the output at column $j$ of a FET-based crossbar: The delay at each output $j$ ($1 \leq j \leq m$) of a FET-based crossbar is formulated as follows:

\[
\forall i, s, t \quad 1 \leq i, s \leq n, \quad 1 \leq t \leq m
\]

\[
\tau_{Oj} = \tau_{Load}^j + \sum_{i=1}^{n} (VM[i][j]) \sum_{s=1}^{n} \sum_{t=1}^{m} (FM[s][t]E_{s,t,i,j})
\]  

(3.3)

• Delay of the output at column $j$ of a diode-based crossbar: The delay at each output $j$ ($1 \leq j \leq m$) of a diode-based crossbar is formulated as follows:

\[
\forall i, s, t \quad 1 \leq i, s \leq n, \quad 1 \leq t \leq m
\]

\[
\tau_{Oj} \geq \tau_{Load}^j + VM[i][j] \sum_{s=1}^{n} \sum_{t=1}^{m} (FM[s][t]E_{s,t,i,j})
\]  

(3.4)

• Exclusivity row movement constraint: The exclusivity row movement constraint states that, each row of the FM can be placed to only one of the rows of the MM, represented by the following constraint:

\[
\sum_{j=1}^{n} R_{i,j} = 1 \quad \forall i \quad 1 \leq i \leq n
\]  

(3.5)

• Exclusivity row placement constraint: The exclusivity row placement constraint states that, each row of the MM can contain only one of the rows of the FM. This constraint is represented as follows:

\[
\sum_{i=1}^{n} R_{i,j} = 1 \quad \forall j \quad 1 \leq j \leq n
\]  

(3.6)

• Exclusivity column movement constraint: Similar to the exclusivity row movement constraint, each column of the FM can be moved to only one of the columns of the MM, represented by the following constraint:
\[ \sum_{j=1}^{m} C_{i,j} = 1 \quad \forall \ i \quad 1 \leq i \leq m \tag{3.7} \]

- Exclusivity column placement constraint: The exclusivity column placement constraint states that, each column of the MM can contain only one of the columns of the FM. This constraint is represented as follows:

\[ \sum_{i=1}^{m} C_{i,j} = 1 \quad \forall \ j \quad 1 \leq j \leq m \tag{3.8} \]

- Position of an entry: Exclusivity constraints make sure the functionality of FM is not changed during generation of the MM. However, in order to calculate delay at the outputs, the exact position of each entry of the FM in the MM must be known. Entry[s][t] of FM is placed at position [i][j] in MM if and only if \( R_{s,i} = 1 \) and \( C_{t,j} = 1 \). Therefore, \( E_{s,t,i,j} \) can be defined as follows:

\[
\forall \ i, j, s, t \quad 1 \leq i, \ s \leq n, \ 1 \leq j, \ t \leq m
\]

\[ E_{s,t,i,j} = R_{s,i} \cdot C_{t,j} \tag{3.9} \]

- In equation 3.9, ”AND” operation on two variables leads to quadratic integer programming. Therefore, we use the following three equations to linearize equation 3.9:

\[
\forall \ i, j, s, t \quad 1 \leq i, \ s \leq n, \ 1 \leq j, \ t \leq m
\]

\[ E_{s,t,i,j} \geq R_{s,i} + C_{t,j} - 1 \tag{3.10} \]

\[ E_{s,t,i,j} \leq R_{s,i} \tag{3.11} \]

\[ E_{s,t,i,j} \leq C_{t,j} \tag{3.12} \]

Note that \( E_{s,t,i,j} \) must be defined as binary variable.
Modification on the ILP-Formulations for Defect Avoidance

The objective in the above ILP formulations is to minimize delay on the outputs of a crossbar (or multi-stage crossbar array) to achieve a variation tolerant mapping. Here, we introduce modifications to these ILP formulations to avoid using defective crosspoints during logic mapping. Since, delay of defective crosspoints are determined by $\tau_{\infty}$ (in practice, the value for this “infinity”, $\tau_{\infty}$, could be $\tau_{\infty} > (\sum_{j=1}^{m} \sum_{i=1}^{n} (VM[i][j]) + \tau_{\text{Load}}^{j}) \times (n + 2)$). Therefore, in the defect-free mapping, delay at the outputs must be less than $\tau_{\infty}$. Then by introducing another constraint on $\tau_{MO}$, MM will be defect-free mapping (if feasible).

$$\tau_{MO} < (\sum_{j=1}^{m} \sum_{i=1}^{n} (VM[i][j]) + \tau_{\text{Load}}^{j}) \times (n + 2)$$ (3.13)

This approach avoids defective crosspoints in the mapping while it simultaneously reduces the delay variation.

Complexity Analysis

It is easy to verify that the variables which represent the elements location (E) dominate the total number of variables. The number of variables representing the locations of elements is $O(n^2 m^2)$, and therefore, this is the order of the total number of variables in the basic ILP formulations. It can be verified that the constraints 3.10, 3.11, and 3.12 introduce the maximum number of constraints, since it models ”AND” operation between every pair of row and column placements. Therefore, there are $n^2 \times m^2$ of such constraints, which is the dominant order of the constraints in the basic ILP formulations.

Simulation Results

PLAMAP function of RASP synthesis tool [72] is used to synthesis a set of MCNC benchmark circuits on custom size PLA blocks. The crossbar size is 1.5X the size of the PLA blocks [27]. In other words, the number of inputs (outputs) of the crossbars are 1.5X the number of inputs (outputs) of the PLA blocks. The PLA blocks are converted
Figure 3.11: Improvements of solving ILP formulations on delay minimization (2X of delay at each time has been shown), variation tolerance (VT), and Defect Tolerance (DT) with respect to different time limits for a crossbar with 144 crosspoints to Function Matrices (FMs) to be mapped on the crossbar nano-architectures. Since the size of the crossbars is chosen to be larger than the size of the PLA blocks, the size of the MMs are larger than the size of the FMs. However, in the proposed ILP formulations, the size of these two types of matrices are supposed to be equal (equations 3.5-3.8). Therefore, to be consistent with the formulations extra rows and columns are added to the FMs, where all of the entries of the added rows and columns are 0. For each FM, 100 different Variation Matrices (VMs) and 100 different Defect Matrices (DMs) are generated. VMs are generated by using Gaussian distribution \(\mu = 50\) and \(3\sigma = 30\). On the other hand, DMs are generated by uniform random distribution with defect density of 40%. A code implemented in C generates the corresponding ILP formulations for each FM with respect to corresponding VM and DM. Furthermore, in order to evaluate the efficiency of using the proposed ILP formulations, it has been compared to SA [67] and un-aware mapping. The un-aware mapping is the traditional logic mapping which does not consider defects or variations in the mapping algorithm. This mapping is done by converting outputs of the RASP synthesis tool to the blocks compatible with crossbar nano-architectures.

We have used IBM ILOG CPLEX Optimizer 12.1.0 to solve the ILP formulations. We adjusted the CPLEX default parameter settings for the MIP EMPHASIS parameter to FEA-
SIBLITY OVER OPTIMALITY, the IMPLIED Bound, CLIQUE, and GOMORY FRACTIONAL CUTS parameters to Aggressive, and the RINS Heuristic parameter to Every 10 iteration. Both of the ILP solver and the SA were executed on a 2Ghz quad core computer with 16GB Memory.

![Figure 3.12: Improvements of running SA on delay minimization (2X of delay at each time has been shown), variation tolerance (VT), and Defect Tolerance (DT) over the time for a crossbar with 144 crosspoints](image)

**Runtime and Scalability**

The computation complexity of ILP solving is related to the size of ILP formulations, specially, the number of variables and constraints. On the other hand, the number of variables and constraints is related to the number of crosspoints in each crossbar. However, simulation results show that the improvements achieved by solving the ILP formulations are saturated over the time. In other words, if solving a set of ILP equations takes $T$ seconds, there is $T_1$ ($T_1 < T$), in which continuation of solving the ILP equations in the period of $T_1$ to $T$ does not lead considerable improvements on reducing the effect of delay variation. The time $T_1$ is called saturation time, which depends on the number of crosspoints in the crossbar. For example, improvements on delay, defect tolerance, and variation tolerance achieved by ILP solver over the time on a set of ILP formulations for a crossbar of size $12 \times 12$ (144 crosspoints) have been shown in Fig. 3.11. On the other hand, running SA
over the time has been shown in Fig. 3.12. In these figures, 2X of delay in each time frame has been shown to show the improvements, clearly. As seen in these figures, the improvement achieved by SA is saturated earlier than ILP. Furthermore, Fig. 3.13 shows saturation point of ILP for crossbars of different sizes.

We use saturation time of solving ILP formulations to set as the time limit of CPLEX ILP solver. This time limit is also used to tune initial temperature and cooling rate of SA. In other words, initial temperature and cooling rate of SA are set such a way that it terminates optimization on the same time as ILP-solver stops.

![Figure 3.13: Saturation time of solving ILP formulations for different objectives with respect to crossbar size](image)

Simulations results for a set of MCNC benchmark circuits synthesized to crossbar size of $12 \times 12$ (the size of PLA is $8 \times 8$) have been shown in Table 3.4 and Table 3.5. For the results of these tables, the time limit for both SA and ILP is set to 3 seconds for each crossbar. Therefore, if a crossbar array consists of $n$ blocks, then total runtime on that crossbar array will be $3n$ seconds. Table 3.4 compares the efficiency of ILP in terms of variation and defect tolerance with respect to SA and un-aware mappings. Variation tolerance is defined as the number of cases in which mapping results in delay in each block less than the threshold delay. In Table 3.4, the columns named $VT$ show variation tolerance achieved by un-aware mapping, SA and ILP, when the threshold delay is set to 80 (which
Table 3.4: Variation Tolerance (VT) and Defect Tolerance (DT) achieved by un-aware mapping (random mapping), SA and ILP.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VT</td>
<td>DT</td>
<td>VT</td>
</tr>
<tr>
<td>alu4</td>
<td>3%</td>
<td>20%</td>
<td>84%</td>
</tr>
<tr>
<td>b9</td>
<td>8%</td>
<td>28%</td>
<td>89%</td>
</tr>
<tr>
<td>C1355</td>
<td>2%</td>
<td>12%</td>
<td>53%</td>
</tr>
<tr>
<td>duke2</td>
<td>3%</td>
<td>22%</td>
<td>82%</td>
</tr>
<tr>
<td>rd84</td>
<td>2%</td>
<td>21%</td>
<td>83%</td>
</tr>
<tr>
<td>term1</td>
<td>2%</td>
<td>19%</td>
<td>76%</td>
</tr>
<tr>
<td>Average</td>
<td>3.3%</td>
<td>20.3%</td>
<td>77.8%</td>
</tr>
</tbody>
</table>

As can be seen in the table, using the ILP formulations results in 95% variation tolerant mapping, compared to SA which achieves 77% variation tolerance. In this table, defect tolerance is calculated as the number of cases in which a defect-free mapping for FMs was found over the total number of FMs. In Table 3.4, defect tolerance achieved by un-aware mapping, SA and ILP is shown under the column called DT. As seen in this table, using the proposed ILP-based approach outperforms SA in both variation and defect tolerance. ILP formulations have been introduced in [65], which can provide defect tolerant mapping (if possible). However, since the order of complexity of variables and constraints of those formulations are at the same order of our proposed formulations, solving both ILP formulations has almost the same time complexity. The efficiency of these two ILP formulations in defect tolerance mapping is the same. However, the previous work does not consider variation effect as well as delay optimizations.

Table 3.5 compares different mapping algorithms in terms of delay for diode-based crossbar nano-architecture. Since the crossbar blocks are cascaded, the critical path delay in these nano-architectures is defined as the number of stages multiply by the period of pre-charge. If maximum delay among all blocks, after a mapping is less than the threshold delays, then the value is used as the period of pre-charge signal. Therefore, secondary objective of variation tolerant mapping is reducing maximum delay of all blocks. P-Delay in table 3.5 shows critical path delay achieved by ILP, SA, and un-aware mappings. Furthermore, this table shows average delay of each crossbar (C-Delay) for each of the mapping.
Table 3.5: Critical path delay (P-Delay) and average crossbar delay (C-delay) achieved by un-aware mapping (random mapping), SA and ILP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P-Delay</td>
<td>C-Delay</td>
<td>P-Delay</td>
<td>C-Delay</td>
</tr>
<tr>
<td>alu4</td>
<td>176</td>
<td>37907</td>
<td>38</td>
<td>24444</td>
</tr>
<tr>
<td>b9</td>
<td>30</td>
<td>5567</td>
<td>33</td>
<td>3261</td>
</tr>
<tr>
<td>C1355</td>
<td>134</td>
<td>16065</td>
<td>37</td>
<td>13362</td>
</tr>
<tr>
<td>duke2</td>
<td>130</td>
<td>28429</td>
<td>34</td>
<td>17728</td>
</tr>
<tr>
<td>rd84</td>
<td>106</td>
<td>35926</td>
<td>43</td>
<td>24242</td>
</tr>
<tr>
<td>term1</td>
<td>52</td>
<td>9210</td>
<td>39</td>
<td>7050</td>
</tr>
<tr>
<td>Average</td>
<td>104</td>
<td>22184</td>
<td>37.3</td>
<td>15014</td>
</tr>
</tbody>
</table>

methods. Using the ILP method in the logic mapping results in 59\% delay improvement over the un-aware mapping. It also has 39\% improvement compared to SA.

Intra/Inter Block Mapping Algorithm

The algorithm consists of two phases: 1- local mapping, 2- global mapping. The local mapping uses intra block transformations, and in case it fails on a block, global mapping runs on the failed blocks, which uses inter block transformations.

Local Mapping

A heuristic algorithm is developed to provide reliable mappings for nano-PLA blocks [4]. The algorithm applies the intra block transformations on each block. The proposed intra block mapping algorithm converts a configuration (called $MM_1$) to a reliable configuration (called $MM_2$). In the beginning, it applies a set of random swapping transformations on the $MM_1$ to avoid local optima. In each step, the algorithm maps a non-zero entry of the $MM_1$ ($MM_1[i][j]$) into the $MM_2$ ($MM_2[t][k]$), then it changes the $MM_1[i][j]$ to zero. Therefore, the algorithm terminates successfully, when all entries of the $MM_1$ are 0. After mapping the $MM_1[i][j]$ into the $MM_2[t][k]$, the row $t$ and the column $k$ of the $MM_2$ are assigned to the input $i$ and the output $j$, respectively. Using the swapping transformation, the algorithm first tries to map a $MM_1[i][j]$ into any $MM_2[t][k]$, where the row $t$ (column $k$) has been assigned to the input $i$ (the output $j$). If using such a row (column) is not
possible (due to reliability constraints), the algorithm assigns a new row (column) for that input (output), using input duplication (output decomposition) transformation. If the local mapping algorithm fails, then the last updated $MM_1$ and $MM_2$ are saved to be used in the global mapping (the inter block mapping algorithm); otherwise, only the $MM_2$ is saved.

The algorithm repeats the following steps until all non-zero entries of the $MM_1$ change to zero.

1. Assign a weight to each entry of the $MM_1$; If the entry $MM_1[i][j]$ is 0, then its weight is 0. If a $MM_1[i][j] = 1$, then its weight is the sum of the weights of the row $i$ and the column $j$. The number of 1s on each row (column) determines its weight.

2. Select an entry of the $MM_1$; A non-zero entry of the $MM_1$ is randomly selected. The probability of selecting an entry is proportional to its weight.

3. Map the selected entry into the $MM_2$; All defect-free crosspoints of the $MM_2$, which have delay less than the threshold delay, can be used in the mapping of the $MM_1[i][j]$. The rows (columns) of the $MM_2$, which have been assigned to the input $i$ (output $j$), have the highest priority in the mapping of the $MM_1[i][j]$. If the entries on such rows (columns) are defective or they have delay more than the threshold delay, then a new row (column) is assigned to the input $i$ (output $j$), randomly.

4. Remove the $MM_1[i][j]$; After mapping an entry, it will be 0 in the $MM_1$.

If the number of rows and column of an MM is $m$ and $n$, respectively, then runtime complexity of step 1 is $O(nm)$. Because, the weight assignment runs on each entry of the matrix. On the other hand, in step 3 the entries on each row are searched. Therefore the runtime complexity of step 3 is $O(nm)$. However, since all steps run for every non-zero entry of MM, therefore, in the worst case there are $nm$ non-zero entries. As a result, the runtime complexity of the local mapping for the worst case is $O(n^2m^2)$.

If the local mapping (intra block mapping algorithm) fails in providing reliable mapping for a block, then the global mapping is applied to that block. Two global mapping algorithms are developed. The first algorithm (function decomposition algorithm) uses function
decomposition transformation to map the remaining entries of the failed block into one of the existing blocks (without adding area overhead in terms of using an extra block). However, if in this way a reliable mapping is not found, the second algorithm (block decomposition algorithm) is executed to decompose the block by exploiting an extra block.

**Global Mapping**

The function decomposition algorithm is executed on all of the $MM_1$ matrices which have at least one non-zero entry. The algorithm tries to map the unmapped entries of each $MM_1$ into the existing $MM_2$ matrices. For example, if the function of an output is $i_1 + i_2 + i_3 + i_4$, but a part of its functionality (e.g. $i_1 + i_2$) is already mapped on an output ($O_1$). The algorithm tries to map $O_1 + i_3 + i_4$ into one of the existing $MM_2$ matrices.

At first, all of the unmapped or partially mapped outputs are inserted to a list (called *unmapped list*). An output is added to this list if it has at least one non-zero entry in a $MM_1$. Furthermore, the input set of each $MM_2$ matrix is determined. This set includes all of the inputs which are used in the configuration of the corresponding block. Finally, for each output in the unmapped list a set (called *function input set*) is determined. The function input set of an output includes all of the required inputs in the mapping of that output. An input $i$ is added to the function input set of an output $j$ if $MM_1[i][j] = 1$.

The algorithm selects the outputs from the unmapped list, one by one. It tries to map the selected output into one of the existing $MM_2$ matrices. However, a $MM_2$ matrix can be used as a candidate for mapping an output if: 1) the "function input set" of that output is a subset of the input set of the $MM_2$ matrix, and 2) the matrix has at least one unused row and one unused column (if needed). The first condition eliminates the extra complexity in the local mapping of the candidate matrix, in terms of adding extra inputs. The unused column is required to map the output (if needed), while the unused row is used to include the already mapped part of the function. If using the local mapping algorithm the output can be mapped into the candidate $MM_2$, the output is removed from the unmapped list. The algorithm repeats this procedure until all entries of the unmapped list are removed.
If the crossbar array consists of \( k \) cascaded crossbars, the runtime complexity of finding unmapped list will be \( O(knm) \). On the other hand, the runtime complexity of finding the input set is \( O(knm) \). Since the algorithm repeats for each entry of unmapped list, and in each iteration it searches among the crossbars, the complexity of this algorithm is \( O(k^2nm) \).

If the function decomposition algorithm cannot successfully find a reliable mapping for a block, a block decomposition transformation is used. The block decomposition partitions the block to two separate blocks. However, the efficiency of the block decomposition transformation depends on the partitioning algorithm. Since increasing the number of unused rows and columns in a block increases the flexibility of the local mapping, the objective of the block decomposition transformation is to maximize the number of unused rows and columns in both the blocks.

In order to maximize the number of unused columns, half of the outputs are mapped into one block and the rest are mapped into another block. Therefore, in each block at least half of the columns will be unused. On the other hand, each shared input between the blocks reduces the number of unused rows by two (one from each block), while each unshared input reduces the number of unused rows only by one. Therefore, the total number of unused rows in the blocks increases by reducing the number of the shared inputs between the blocks. An input is shared between two blocks, if at least one of the outputs in each block is a function of that input. In order to reduce the number of the shared inputs between the blocks, partitioning algorithm selects the outputs with the maximum number of shared inputs to be mapped into the same block. These outputs must also have the maximum number of unshared inputs with the outputs of another block. Therefore, the partitioning is done based on the joint differential input dependency of the outputs. Joint differential input dependency of two outputs is defined as the difference between the number of shared and unshared inputs of those outputs. Furthermore, joint differential input dependency of an output and a set of outputs is defined as the difference between the number of shared and unshared inputs of that output with all of the outputs of the set. The outputs with highest joint differential input dependency are mapped into the same block.
Finding joint differential input dependency of the outputs can be identified by a bipartite graph. The nodes in each side of the graph correspond to the outputs. For each common input there is an edge between output nodes. Therefore, the runtime complexity of joint differential input dependency of the outputs is $O(n^2m)$. Since, it repeats for each block, therefore the complexity of the block decomposition algorithm is $O(kn^2m)$.

The runtime complexity of the local mapping is $O(n^2m^2)$, which must be repeated for each block, therefore its total complexity is $O(kn^2m^2)$. For the failed blocks, the function decomposition algorithm with the complexity of $O(k^2nm)$ or the block decomposition algorithm with the complexity of $O(kn^2m)$ is executed. If, $k$ is the dominant factor (the number of block is more than size of the blocks), then the runtime complexity of total algorithm will be $O(k^2nm)$, otherwise it will be $O(kn^2m^2)$.

**Simulation Results**

We have synthesized a set of MCNC benchmarks by RASP PLA synthesis tool [72] with different sizes for the PLA. The synthesized circuits are mapped into the crossbar array. Table 3.6 shows the average number of unused rows, columns and crosspoints with respect to different sizes of crossbars. As can be seen in this table, more than 50% of rows and columns as well as more than 85% of crosspoints are unused for typical crossbar mappings.

<table>
<thead>
<tr>
<th>Crossbar size</th>
<th>Unused rows</th>
<th>Unused columns</th>
<th>Unused crosspoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 × 4</td>
<td>57%</td>
<td>61%</td>
<td>87%</td>
</tr>
<tr>
<td>6 × 6</td>
<td>60%</td>
<td>62%</td>
<td>91%</td>
</tr>
<tr>
<td>8 × 8</td>
<td>59%</td>
<td>59%</td>
<td>92%</td>
</tr>
<tr>
<td>10 × 10</td>
<td>59%</td>
<td>57%</td>
<td>92%</td>
</tr>
<tr>
<td>12 × 12</td>
<td>60%</td>
<td>53%</td>
<td>92%</td>
</tr>
<tr>
<td>14 × 14</td>
<td>63%</td>
<td>55%</td>
<td>93%</td>
</tr>
<tr>
<td>16 × 16</td>
<td>63%</td>
<td>50%</td>
<td>94%</td>
</tr>
</tbody>
</table>

In order to evaluate efficiency of the introduced operations, the proposed algorithm has been compared to *Simulated Annealing* (SA) [67] (exploits swapping operation in logic mapping). Both algorithms are executed on a set of MCNC benchmark circuits. These circuits synthesized to custom sized PLA blocks, by using PLAMAP function of RASP
synthesis tool [72]. The size of crossbar arrays were considered as $16 \times 16$ (16 input 16 and 16 outputs). Each circuit is mapped on 1000 crossbar arrays. VMs of crossbars of each crossbar array were generated using Gaussian random distribution ($\mu = 50, \sigma = 15$), while DMs were generated by uniform random function (defect rate = 20%).

Table 3.7: Percentage of successful reliable mapping of blocks (Block Yield (B-Y)) and the circuits (Circuit Yield (C-Y)) achieved by Simulated Annealing (SA) and the proposed method

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SA [67]</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B-Y</td>
<td>C-Y</td>
</tr>
<tr>
<td>alu4</td>
<td>58.6%</td>
<td>0.0%</td>
</tr>
<tr>
<td>apex4</td>
<td>95.2%</td>
<td>0.0%</td>
</tr>
<tr>
<td>b9</td>
<td>97.9%</td>
<td>75.7%</td>
</tr>
<tr>
<td>C880</td>
<td>75.31%</td>
<td>0.0%</td>
</tr>
<tr>
<td>C1355</td>
<td>46.42%</td>
<td>0.0%</td>
</tr>
<tr>
<td>C3540</td>
<td>69.6%</td>
<td>0.0%</td>
</tr>
<tr>
<td>duke2</td>
<td>88.2%</td>
<td>0.0%</td>
</tr>
<tr>
<td>ex5p</td>
<td>15.1%</td>
<td>0.0%</td>
</tr>
<tr>
<td>k2</td>
<td>91.8%</td>
<td>0.0%</td>
</tr>
<tr>
<td>rd84</td>
<td>25.8%</td>
<td>0.0%</td>
</tr>
<tr>
<td>t481</td>
<td>89.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Average</td>
<td>68.9%</td>
<td>6.9%</td>
</tr>
</tbody>
</table>

Two kinds of yield are calculated: 1) Block Yield (B-Y), and 2) Circuit Yield (C-Y). Block yield (B-Y) is defined as the number of reliable mapping of blocks of each circuit to the total number of blocks of that circuit in 1000 mappings. Circuit Yield (C-Y) for the mapping of a circuit is defined as the number of cases were all of the blocks of that circuit are reliable mapped. Timing constraint (threshold delay) of nano-PLA is set to 65 ($\mu + \sigma$).

The simulation results for each of the algorithms has been shown in Table 3.7. As seen in the table, the proposed algorithm can achieve more than 99.9% block and circuit yields, while SA achieves 68.9% block yield and 6.9% circuit yield. It must be noted although for some cases the block yield is high (more than 97%) but the yield for reliable mapping of circuit is very low. This is due to the fact that if at least one of the blocks of a circuit cannot be mapped reliable, then the mapping of the circuit will not be reliable. Area overhead of the proposed method compared to SA is 12.5%, in average. In fact area overhead of SA is zero, because it does not require crossbars more than what is determined by synthesis tool.
Chapter 4

Reliable Reversible Circuit Design

4.1 Previous Work

4.1.1 Background

The requirements of a technology to be used in implementation of any kind of quantum computers (including reversible circuits) have been discussed in [77, 78]. These requirements include: robust representation of quantum information, performing a universal set of unitary transformations, preparation of accurate initial states, and measuring the output results.

One of the appropriate candidates to fulfill such requirements is trapped-ion technology [77, 78, 79]. In this technology, the logical values are represented by the certain spin and vibrational modes of electrically charged atoms (ions). By analyzing this technology, a set of appropriate logical fault models for reversible circuits have been proposed [78]. These fault models are briefly reviewed in the follows:

- Single Missing Gate Fault (SMGF): In this fault, a single gate is completely removed from the circuit [78]. An SMGF results in one-to-one connection of inputs and outputs of the missed gate. In other words, if a K-CNOT gate is missed, then it operates as $F = T$ and $G_i = C_i \ (\forall i, 1 \leq i \leq K)$. A fault-free circuit is shown in Figure 4.1.a. This circuit with a SMGF on the second gate (gate inside of the dotted rectangle) has been shown in Figure 4.1.b. As seen in the figure, missing gate changes functionality...
of at least one of the primary outputs. This is due to the fact that when the control input of this gate is logical 1, then its function output will be $\neg d$ (inverted value of $d$), where $d$ is the target input. However, for the same control input, the function output will be $d$ in case of SMGF on this gate. Therefore, this gate produces two different values for the function output in faulty and fault-free cases. Since the rest of the circuit (the sub-circuit in the right-hand side) is not changed and also it is a reversible circuit, the response of the circuit for these two different values will be different at primary outputs.

Figure 4.1: Illustration of logical fault models for reversible circuits: (a) the fault-free circuit (b) the SMGF on the second gate (c) the RGF on the second gate (d) the MMGF of the last two consecutive gates (e) the first-order PMGF of the first control input of the first gate (f) Appearance fault on the second gate

Therefore, it can be concluded SMGF is always testable. A test vector can detect the SMGF of a gate if the test vector justifies the logical values of 1 at all of the control inputs of that gate. When all control inputs of a gate are 1, then the response of that gate will be different for input vectors in the faulty and the fault-free cases. Therefore, the fault will be activated. Since any sub-circuit of a reversible circuit is also a reversible circuit, the activated fault is detected at primary outputs by producing
different values at primary outputs. It has been shown that an upper bound on the
number of test vectors which can detect all of SMGFs in a circuit (complete test set)
is equal to \( \lceil N/2 \rceil \), where \( N \) is the number of gates in the circuit [80]. However, this
is an roughly upper bound, while in most of the cases, all SMGFs in a circuit can be
detected with fewer number of test vectors. For example, the test vector \( abcd = 1101 \)
can detect all three possible SMGFs in Figure. 4.1.a.

- **Repeated Gate Fault (RGF):** In this fault, a gate is replaced by several instances of
  the same gate, with cascaded corresponding connections [78]. Figure. 4.1.c shows the
case in which second gate is repeated. As seen, in a RPG the position of the repeated
gate is preserved and also the connections are the same for all of the instances. In
other words, the garbage output \( G_i \) of the gate at stage \( m \) is connected to the control
input \( C_i \) of the gate at stage \( m + 1 \). Also the function output of the gate at stage \( m \)
is used as the target input of the gate at stage \( m + 1 \).

If a gate is replaced by even number of instances of the same gate, then fault effect is
identical to SMGF of that gate [78]. This is due to the fact that when all control inputs
of the repeated gate instance in the most left side are logical 1, all other instances will
have logical 1 on all of their control inputs (the control inputs of a gate are garbage
outputs of the gate instance in its left hand side). Therefore, each pair of instances
will produce \( d \) on the function output of that pair (\( d \) is the target input for the left
hand side gate). The first instance of the pair produces \( \neg d \), while the second gate
produces \( \neg (\neg d) = d \) on function output. Therefore, each pair of repeated instances
will preserve the logical value of target input, similar to SMGF.

Therefore, all RGFs with even number of repeated gate instances are testable by test
vectors produced to detect SMGFs. In other words, a complete test set for SMGF
can detect all of the RGF with even number of instances. However, replacement of a
gate by odd number of instances does not have any effect on the functionality of the
circuit. Therefore, RGFs which are resulted by replacement of odd number of gates
are not testable. Those RGFs result in redundancy in the circuit without affecting
CHAPTER 4. RELIABLE REVERSIBLE CIRCUIT DESIGN

the functionality of the circuit.

- **Multiple Missing Gate Fault** (MMGF): In this model, it is assumed that several consecutive gates are completely removed from the circuit (Figure 4.1.d). In contrast to traditional definition of multiple faults, MMGF is not necessarily the same as multiple SMGF. In this model, all of the missed gates must be consecutive gates, while multiple SMGF implies that several distinct arbitrary gates are missed which are not necessarily consecutive.

For example, in Figure 4.1, if the first and the last gates are missed, it is not considered as MMGF. In a circuit with \(N\) gates, the number of possible MMGFs is \(N(N + 1)/2\), whereas the number of possible multiple SMGFs in the same circuit is \(2^N - N - 1\) [78].

- **Partial Missing Gate Fault** (PMGF): This fault, also known as a disappearance fault, assumes a subset of control inputs of a gate is removed. This fault turns a \(K\)-CNOT gate into \(K'\)-CNOT gate, where \(K' < K\). If \(m\) of the \(K\) control inputs of a \(K\)-CNOT is removed, then the gate will be transferred to \((K - m)\)-CNOT gate, where \(m\) is called the order of PMGF [80]. Figure 4.1.e shows one of the possible PMGFs on the first gate. This fault is first order PMGF.

It has been shown that \(K\) test vectors can detect all of the first order PMGF on a \(K - CNOT\) gate. In each of these test vectors one of the control inputs is 0 and the rest of the control inputs must be 1. Therefore, if a control input is removed while the logical value of that input is 0, then in fault free case that gate will produce \(d\) on its function output (\(d\) is the target input of the gate). However, in the faulty case that control input (with value of logical 0) is removed, while the other control inputs are logical 1. Therefore, that gate will produce \(\neg d\) on its function output. Since the function output of the faulty gate is not the same as function output of the fault-free gate, that fault can be detected at primary outputs. Furthermore, these \(K\) test vectors can detect all of PMGFs on that gate. However, \(K + 1\) test vectors are required to detect SMGF on a gates as well as all possible PMGFs on that gate [80].
• *Appearance Fault (AF):* this fault assumes $m$ extra control lines are added to the gate, which transforms a $K$-CNOT gate to a $(K + m)$-CNOT gate. Appearance of a control line to the first gate of Figure 4.1.a has been shown in Figure 4.1.f.

### 4.1.2 Test Generation

Reversibility leads to full controllability and observability in the circuit, which significantly simplifies testing compared to irreversible circuits [81]. Due to full controllability and full observability, any single stuck-at fault can be detected in reversible circuits. In other words, for any reversible circuit, there is a complete test set which detects all single stuck-at faults. Furthermore, it is shown that any test set that detects all single stuck-at faults in a reversible circuit also detects all multiple stuck-at faults [81]. However, finding a minimum complete test set to detect all stuck-at faults is NP-hard [82]. Therefore, various heuristic algorithms to generate complete test set for single/multiple stuck-at faults have been introduced [83, 84].

A randomized algorithm with polynomial time complexity to generate a complete test set for stuck-at faults is proposed in [85]. A polynomial time, test pattern generation algorithm for detecting single and multiple input bridging faults is presented in [86]. Testing of the intra-level single bridging fault model (i.e. any single pair of lines, both lying at the same level of the circuit) is investigated in [87]. An *Integer Linear Programming* (ILP) based test-set generation algorithm has been proposed to generate complete test set which can detect all single/multiple stuck-at faults in a reversible circuit [88]. This approach leads to almost 50% reduction in the number of required test vectors compared to conventional Automatic Test Pattern Generation (ATPG). A test pattern generation method to detect all MMGFs has been introduced in [89]. However, it requires a large number of test vectors. These test patterns can also detect all SMGFs. A randomized ATPG algorithm to detect crosspoint faults has been proposed in [80].
4.1.3 Fault Masking

Failure analysis of Triplicated Modular Redundancy (TMR) in conventional irreversible logic has been widely explored in [90]. Various distributions of TMR and voters in a circuit have been also presented. Logic synthesis on a set of reversible gates called Majority Based Reversible Logic Gate (MBRLG) has been discussed in [91]. An MBRLG is a reversible gate with odd number of inputs such that at least one of the gate outputs is a majority Boolean function of all inputs. A voting technique for reversible logic has been introduced based on majority multiplexing [92]. A reversible majority gate called $MAJ$ was proposed [92]. In this module one of the output bits is the majority of the input bits. However, this gate suffers from single point of failure problem. To resolve this problem, authors have used this gate similar to the Von-Neumann multiplexing method, which results in area overhead more than $6X$ (due to six copies of the module and its reverse as well as fan-out branch gates). Therefore, they have used three copy of this module between two stages of TMR modules, which increases the area overhead.

4.1.4 Online Testing

Variety of parity generating techniques have been proposed in [93, 94]. The general idea behind these techniques is generating a parity before (reference parity) and after (checking parity) computation. Each computational stage in these techniques consist of two cascaded blocks. The logical operation of the stage is implemented by the first block which generates functional outputs as well as a parity bit of the functional operation. The second block also generates another parity for those outputs. Any fault resulting in incorrect parity can be detected by comparing these two parity bits. However, in addition to high area overhead of this method, it cannot detect faults on the inputs of the first block.

A $4 \times 4$ reversible gate named Online Testable Gate (OTG) is proposed in [95]. This gate is combined with Feynman Gate (FG) [56] to design online testable reversible circuits. In this method, functionality is implemented by FG, and outputs of the FG are passed through an OTG. However, combined gates are realized as a NAND gate. A specific output of FG is
compared with a specific output of the OTG. In fault free case, these two outputs must be complement of each other. This method can detect any fault on output of FG and OTG, but if a fault occurs in the input of FG, it will not be detected. The area overhead of this method is approximately twice of the original circuit.

Parity preserving approach to design of reversible logic circuits has been investigated in [96]. In this paper, a class of reversible logic gates has been introduced in which the parity of the outputs matches that of the inputs. An error detection and correction using multiple parity prediction technique based on Low Density Parity Check (LDPC) code has been introduced in [97]. A $4 \times 4$ reversible gate for implementing hamming coding and detection circuits is proposed in [98]. This gate is modified to incorporate parity preserving property for achieving fault tolerance for the hamming error correcting code and detection circuits. Similarly, another set of gates are introduced in [99, 100] with parity preserving property. However, these techniques are ad-hoc and are not easy to extend to all reversible circuits. In other words, applying these techniques to a reversible circuit, introduces complexity to synthesis process of reversible circuits which is still a challenge. Besides, these techniques require the incorporation of parity checkers for every primitive cell in the circuit which involves large area overhead.

4.1.5 Design for Testability

A Design-For-Testability (DFT) method to convert reversible circuit synthesized by Toffoli gates into a fully testable circuit for single intra-level bridging fault has been introduced in [101]. The considered fault model in this method is classical AND/OR-bridging fault model. A DFT method to detect SMGFs in a reversible circuit is introduced in [102], which effectively simplifies testing for SMGFs. This method results in up to 60% gate overhead. A duplication approach is introduced to detect all SMGFs, RGFs, and PMGFs in a reversible circuit. However, such technique requires $N$ test vectors ($N$ is the number of inputs of the circuit) to detect all SMGFs, RGFs, and PMGFs. This approach leads to more than 100% area overhead in the circuit.
4.2 Proposed Approaches

In terms of reversible circuits, we study online and offline testing of these circuits as well as fault masking and diagnosis [8, 9, 10, 11, 12]. In order to provide online testing, we use a parity generation methodology to detect faults. Furthermore, a cyclic test generation methodology is used to provide test patterns with small amount of test information, applicable for in the field testing. We also propose a set of majority voting gates which can be used in Triple Modular Redundancy (TMR) circuits to enable both fault masking and fault diagnosis. However, in the follows a sub-set of these approaches will be reviewed.

4.2.1 Online Testing of Missing/Repeated Gate Faults

We have proposed a technique to detect missing/repeated gate faults in reversible circuits, during runtime operation of circuits [10]. In this technique a set of reversible gates are proposed to generate the inverted value of one of the inputs on one of the garbage outputs. The inverted line in cascaded gates is compared to the expected value to detect any missing or repeated gate fault.

The basic idea behind the proposed gates is keeping information of the number of gates in order to detect missing/repeated gate faults. This can be done by creating inversion of one of the control inputs on one of the garbage outputs. This garbage output can be used as a control input to the successor gate. In the cascaded gates, again the inversion on the same signal is done. In other words, this approach holds the parity information based on the level (stage) of each gate. Therefore, if one of the gates is missed or a gate is repeated, by checking the garbage output at primary outputs the fault can be detected, since the parity information will be distorted.

Figure 4.2 shows the general structure of this method. Figure 4.2.a shows the fault-free case where the expected value of the first garbage output at the primary output is $D$. Missing and repeated gate faults are shown in Figure 4.2.b and Figure 4.2.c, respectively. In both of these two cases, the expected value on the first garbage output at the primary output is $\neg D$. A simple checker can be used at primary outputs to check $D$ and raise
appropriate error signal.

![Diagram](image)

Figure 4.2: General scheme of online missing/repeated gate detection: a) A fault-free circuit which produces $D$ on the first primary output, b) Gate missing fault on gate $R_2$ which results in $\neg D$ on the first primary output, and c) Repeated gate fault on gate $R_2$ which results in $\neg D$ on the first primary output.

The proposed gates must be universal gates to be used in any (reversible) logic implementation. Furthermore, since fan-out branches are not allowed in reversible circuits, these gates must be able to provide multiple copies of a signal for reversible implementation of fan-out branches. In the follows, the proposed reversible gates will be introduced.

**Proposed Reversible Logic Gate (LG)**

The schematic of the proposed reversible *Logic Gate* (LG) is shown in Figure 4.3. Inputs $I_1$ and $I_2$ are the target inputs (computation is done on these inputs), $C$ is the control input (to control gate operation), and $D$ is the detection line (to propagate information of the number of gates). Outputs $F_1$ and $F_2$ are the function outputs, $G$ is the garbage output, and $\neg D$ is the propagation line for the detection input.

The truth table of this gate is presented in Table 4.1.

![Truth Table](image)

Figure 4.3: The schematic of the proposed reversible *Logic Gate* (LG)

In this gate, if $C = 0$, $I_1 \cdot I_2$ (logical AND) is computed on $F_1$ while $I_1 + I_2$ (logical OR) is computed on $F_2$. On the other hand, if $C = 1$, logical NAND and NOR are computed on
$F_1$ and $F_2$, respectively. The garbage output produces $I_1$ to be used in the next stage (if needed). However, the detection line is the line which is used to keep the information of the number of the gates in a path. It produces the inverted value of $D$ regardless of the value of the control input.

Table 4.1: The truth table of the proposed reversible Logic Gate (LG)

<table>
<thead>
<tr>
<th>Target Inputs</th>
<th>Odd Stages</th>
<th>Even Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \ D = 0 \ 0$</td>
<td>$C \ D = 1 \ 0$</td>
<td>$C \ D = 0 \ 1$</td>
</tr>
<tr>
<td>$I_1 \ I_2$</td>
<td>$G \ \neg D \ F_1 \ F_2$</td>
<td>$G \ \neg D \ F_1 \ F_2$</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 0 0 1 0 1</td>
<td>0 1 1 0</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 0 0 1 1 1 0</td>
<td>1 1 1 0</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

Function ($F_1$) | AND | NAND | AND | NAND |
Function ($F_2$) | OR | NOR | OR | NOR |

Figure 4.4: Configuration of LG to provide fan-out branch

Fan-out branches also can be realized from LG. If in LG both the $I_1$ and $I_2$ are used as control inputs, while $C$ is used as target input, then the gate realizes fan-out of $C$. In this realization if both $I_1$ and $I_2$ are set to constant value of 0, then the gate duplicates $C$ on $F_1$ and $F_2$. Figure 4.4 shows such configuration.

Detection Scheme

A multi-stage circuit consists of a set of gates in each stage. The number of gates in each stage is defined as the depth of that stage. The maximum depth determines the total number of D-paths. Therefore, in a circuit there are $k$ paths which contain D-lines ($k$: the number of D-paths). Some of these paths contain an odd number of gates and the rest contains an even number of gates. All D inputs of all gates at stage 1 are connected to 0.
Therefore, the paths with an odd number of gates result in D at the end which must be equal to 1. On the other hand, the primary outputs of D-paths with an even number of gates are 0. If a gate or an odd number of gates in a D-path is missed, then the expected output of that D-path will be changed. This change can be determined by an “Alarm” signal produced by a gate called \textit{D-Collector Gate} (DCG). This gate is designed based on the fact that all D-inputs at level-0 are 0. It collects a D at an odd stage (call it \(D_{\text{Odd}}\)) and a D at an even stage (call it \(D_{\text{Even}}\)).

Figure 4.5: The schematic of the reversible \textit{D-Collector Gate} (DCG)

The schematic and truth table of DCG are shown in Figure 4.5 and Table 4.2, respectively. This gate combines a pair of D-paths, one from an odd path and another from an even path. After combining all pairs of D-paths, for the rest of D-paths (if the number of even and odd paths are not equal), the corresponding inputs can be replaced by constant 0 and 1, respectively. For example, if there are 4 D-paths containing an odd number of gates, and 5 D-paths containing an even number of gates. Then 4 pairs of paths (each pair contains one odd D-path and one even D-path) are combined by four DCGs. The remaining even D-path is combined by another DCG whose \(D_{\text{odd}}\) input is connected to 1. Each DCG produces two other outputs (in addition to Alarm signal). These outputs, called \(A_1\) and \(A_0\), indicate the address of the faulty path for diagnosis purposes.

Table 4.2: The truth table of the reversible \textit{D-Collector Gate} (DCG)

<table>
<thead>
<tr>
<th>Situation</th>
<th>D-Lines</th>
<th>(C = 1)</th>
<th>(C = 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both Fault-Free</td>
<td>(D_{\text{Odd}}) (D_{\text{Even}})</td>
<td>(A_{\text{Alarm}}) (A_1) (A_0)</td>
<td>(A_{\text{Alarm}}) (A_1) (A_0)</td>
</tr>
<tr>
<td>Fault on (D_{\text{Odd}})</td>
<td>0 0</td>
<td>1 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>Fault on (D_{\text{Even}})</td>
<td>1 1</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>Both Faulty</td>
<td>0 1</td>
<td>1 1 1</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>
Figure 4.6 shows the detection scheme on two paths. For the sake of simplicity, the rest of the inputs of the gates are not shown in the figure. Furthermore, to distinguish logic gates with the gate for fan-out and inversion, the logic gates used for fan-out and inversion are denoted by $FG$, and the extra inputs and outputs are removed on those gates. In this example, one of the D-paths (indicated by red dotted line) consists of 3 gates, while another D-path (indicated by green dashed line) consists of 4 gates. These two paths for D-lines are collected by a DCG. If one or both paths contain an odd number of faults, it is detected by the Alarm signal.

![Figure 4.6: Detection scheme for a circuit consists of two D-paths (FG denotes LG which is used as fan-out)](image)

**Discussion**

The proposed gates are designed to detect SMGF and RGF. However, they are able to detect MMFG as long as the number of missing gates is odd. On the other hand, this approach can detect multiple faults as long as the error is not masked in at least one of the paths collected by a DCG. The error is masked on a path $i$ if $|N^i_m - N^i_r|$ is even on this path, where $N^i_m$ and $N^i_r$ are the number of missing and repeating gate faults on path $i$, respectively.
Simulation Results

The proposed gates have been compared to previous work (Toffoli Gate [57] and Fredkin Gate [56]) in terms of area (number of gates and input nets), delay (maximum gate level), and number of garbage outputs (number of unused outputs). To do so, a set of MCNC benchmark circuits have been mapped to these gates. Table 4.4 and Table 4.3 compare our proposed method to Toffoli Gate and Fredkin Gate in mapping of different benchmark circuits.

Table 4.3: Experimental results on a set of benchmarks; The number of garbage outputs and delay

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Toffoli Gate [57]</th>
<th>Fredkin Gate [56]</th>
<th>Proposed Gates [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Garbage</td>
<td>Delay</td>
<td>Garbage</td>
</tr>
<tr>
<td>alu2</td>
<td>2560</td>
<td>19</td>
<td>2856</td>
</tr>
<tr>
<td>b9</td>
<td>1001</td>
<td>14</td>
<td>1224</td>
</tr>
<tr>
<td>majority</td>
<td>25</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>t481</td>
<td>14135</td>
<td>22</td>
<td>15109</td>
</tr>
<tr>
<td>term1</td>
<td>5628</td>
<td>19</td>
<td>6162</td>
</tr>
<tr>
<td>vda</td>
<td>3536</td>
<td>17</td>
<td>3814</td>
</tr>
<tr>
<td>x4</td>
<td>7010</td>
<td>18</td>
<td>8117</td>
</tr>
<tr>
<td>Average</td>
<td>4842</td>
<td>16.4</td>
<td>5331</td>
</tr>
</tbody>
</table>

In Table 4.3, the maximum number of gates in all the paths is selected for delay comparison. In other words, a path from the primary inputs to the primary outputs which contains the maximum number of gates is used in delay estimation. In this table, the number of garbage outputs is calculated by counting the number of unused outputs.

Toffoli and Fredkin gates consists of $3 \times 3$ reversible gates. However, our proposed gates are $4 \times 4$ logic gate (LG) and $3 \times 3$ d-collector gate (DCG). Therefore, area estimation of the circuits depends on the physical realization of the gates. However, in Table 4.4, the relative area of the circuits are reported. In order to have a relative area estimation, the total number of the gates as well as the total number of the input nets for each reversible mapping method are reported in the table. Since LG has the capability of producing two basic logic operation (e.g. AND and NAND) on a gate, therefore the total number of gates mapped on proposed gates is less than the other methods.
Table 4.4: Experimental results on a set of benchmarks; The number of gates and the number of input nets

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Toffoli Gate [57]</th>
<th>Fredkin Gate [56]</th>
<th>Proposed Gates [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gates</td>
<td>Inputs</td>
<td>Gates</td>
</tr>
<tr>
<td>alu2</td>
<td>1715</td>
<td>5145</td>
<td>3895</td>
</tr>
<tr>
<td>b9</td>
<td>655</td>
<td>1965</td>
<td>764</td>
</tr>
<tr>
<td>majority</td>
<td>17</td>
<td>51</td>
<td>22</td>
</tr>
<tr>
<td>t481</td>
<td>9436</td>
<td>28308</td>
<td>9918</td>
</tr>
<tr>
<td>term1</td>
<td>3761</td>
<td>11283</td>
<td>4021</td>
</tr>
<tr>
<td>vda</td>
<td>2398</td>
<td>7194</td>
<td>2521</td>
</tr>
<tr>
<td>x4</td>
<td>4688</td>
<td>14064</td>
<td>5238</td>
</tr>
<tr>
<td>Average</td>
<td>3238</td>
<td>9714</td>
<td>3768</td>
</tr>
</tbody>
</table>

Table 4.5 shows the coverage of the proposed gates for different fault rates for multiple fault detection analysis. For each fault rate, we have randomly injected faults to the circuit. The number of inserted faults is determined by the fault rate. For example, if there are 100 gates in the circuit and the fault rate is 10%, then we inject 10 faults to the circuit. As mentioned in the previous section, multiple faults are detected as long as the error is not masked in at least one D-path. It must be noted, since the probability of all paths containing even number of faults is very small, therefore the proposed method can detect multiple faults in most of the cases. The fault simulation of each circuit is repeated 100,000 times for each fault rate. The average number of detection (at least one D-path with odd number of faults) for these 100,000 simulations has been reported in Table 4.5. The second column of this table (# of D-paths) shows the number of D-paths in each circuit.

Table 4.5: Online detection coverage achieved by the proposed method for different fault rates

<table>
<thead>
<tr>
<th>Circuit</th>
<th>D-paths</th>
<th>Fault rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5%</td>
</tr>
<tr>
<td>alu2</td>
<td>213</td>
<td>99.97%</td>
</tr>
<tr>
<td>b9</td>
<td>96</td>
<td>99.78%</td>
</tr>
<tr>
<td>majority</td>
<td>5</td>
<td>100.00%</td>
</tr>
<tr>
<td>t481</td>
<td>1057</td>
<td>100.00%</td>
</tr>
<tr>
<td>term1</td>
<td>455</td>
<td>100.00%</td>
</tr>
<tr>
<td>vda</td>
<td>248</td>
<td>99.93%</td>
</tr>
<tr>
<td>x4</td>
<td>656</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
4.2.2 Fault Masking in Reversible Circuits

Reversible Majority Voters

Two different implementations for a majority voter with triplicated voted outputs have been presented [11]. These modules provide fault masking and fault diagnosis in the scope of *Triplicated Modular Redundancy* (TMR). The schematic of such voter is shown in Fig. 4.7. It consists of three data inputs ($I_1$, $I_2$, and $I_3$), and two control lines in the input of the module ($C_1$, and $C_2$). It generates three data outputs ($O_1$, $O_2$, and $O_3$), and two garbage outputs ($G_1$, and $G_2$). Voting is done on data inputs, and results are produced on data outputs.

![Majority Voter Diagram](image)

**Figure 4.7:** Proposed reversible majority voter, with triplicated voted output

The objective of majority voter implementation is producing all 0 or all 1 on three data outputs. In other words, two permutations out of eight possible permutations of these outputs are used in the implementation. On the other hand, reversibility impose one to one mapping between input and output permutations. Therefore, at least two garbage outputs are required to preserve reversibility.

Since in a reversible circuit fan-out on wires is not possible, the proposed voters provide three copies of the voted output to enable efficient implementation of distributed TMR architectures. In addition, the triplicated outputs can be implemented in such a way that eliminates the problem of single point of failure in the voter.
Area-Efficient Implementation

*Minimal Triplicated Voter (MTV)* is an implementation of the described voter which is shown in Fig. 4.8. The main objective in MTV implementation is using minimum number of stages. This implementation consists of 4 stages (i.e. 4 reversible gates). The relation between \( I_i \) (\( 1 \leq i \leq 3 \)) and \( O_i \) (\( 1 \leq i \leq 3 \)) is shown in Table 4.6.

Table 4.6: Input-output relation of the MTV

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_3 \ I_2 \ I_1 )</td>
<td>( O_3 \ O_2 \ O_1 )</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

*Analysis of MTV:* Considered fault models include missing-gate and repeated-gate faults. Bit-flip on signal lines (including the inputs of the voters) is also considered in our analysis which could be the effect of missing or repeated gate in previous stage. A single fault results in three types of errors as the followings:

1. *Maskable errors:* those who can be masked or result in error on garbage lines which do not affect functionality of the circuit.
2. Recoverable errors: if a fault results in bit-flip on one of the three copies of voted output, then it can be recovered by the voter at successor TMR stage. Therefore, the effect of this fault can be recovered if no other fault(s) occur in that successor voter.

3. Unrecoverable errors: any fault which changes more than one copy of the voted outputs, results in an error which is neither maskable nor recoverable.

Any single fault on inputs of the MTV is masked by this module (Maskable error). Furthermore, a single fault inside the MTV results in Maskable or Recoverable errors. However, it is not robust against missing gate and repeated gate faults which occur in the voter while there is a fault on the input lines of voter. In addition to this, the diagnosis information of this voter is not sufficient to indicate faulty input. In order to increase the robustness of the voter as well as increasing the diagnosis information of the voter another implementation of the voter is proposed.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{robotic_triplicated_voter.png}
\caption{Robust Triplicated Voter (RTV)}
\end{figure}

**Robust Implementation**

The proposed Robust Triplicated Voter (RTV) is shown in Fig. 4.9. The input-output relation of RTV is shown in Table 4.7. Implementation of the proposed voters follows the general rule of majority voting. The output is equal to 0 if more than one of the inputs (at least two of inputs) are 0 otherwise the output is 1. In MTV implementation, the output is calculated based on the value of inputs. Then three copies of the output
are generated. In contrast to MTV implementation, in RTV implementation each copy of voted value is produced independently, directly from the inputs. Since RTV produces three independent copies of the majority input bits, it reduces number of faulty cases which result in Unrecoverable error. Also, most of the single faults are maskable in RTV rather than being recoverable. This module provides full diagnosis for single faults. Note that, these capabilities are achieved at the cost of increased area and delay (in terms of circuit depth).

Table 4.7: Input-output relation of the RTV

<table>
<thead>
<tr>
<th>Inputs I1 I2 I3</th>
<th>Outputs O1 O2 O3 D1 D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

The effects of all types of faults on the proposed implementations have been evaluated and the percentage of resulting errors are presented in Table 4.8. Results of this table have been obtained from evaluations at three phases. In the first phase, we assumed inputs of the voters are fault-free, and a fault occurs inside of the voter. This fault results in missing or repeating one of the gates of the voter. In the second phase we assumed the voter is fault-free, but a fault in circuit resulted in a bit-flip on one of the inputs of the voter. In the third phase, we assume a fault has occurred inside the voter while another fault in the circuit results in a bit-flip on one of the voter inputs.

Table 4.8: Fault Analysis for MTV and RTV: Percentage of Maskable errors (M), Recoverable errors (R), and Unrecoverable errors (U)

<table>
<thead>
<tr>
<th></th>
<th>MTV</th>
<th></th>
<th></th>
<th>RTV</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M</td>
<td>R</td>
<td>U</td>
<td>M</td>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>(1) SMGF</td>
<td>50%</td>
<td>50%</td>
<td>-</td>
<td>87.5%</td>
<td>12.5%</td>
<td>-</td>
</tr>
<tr>
<td>(2) Single Bit-flip</td>
<td>100%</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Both (1) and (2)</td>
<td>34.4%</td>
<td>50%</td>
<td>15.6%</td>
<td>56.2%</td>
<td>42.1%</td>
<td>1.7%</td>
</tr>
</tbody>
</table>
CHAPTER 4. RELIABLE REVERSIBLE CIRCUIT DESIGN

Diagnosis Scheme

The proposed diagnosis scheme is a hierarchical scheme which provides different diagnosis resolution requiring different diagnosis steps. This scheme provides online diagnosis of permanent and transient faults, during runtime operation of circuit. The diagnosis resolution can vary from a stage of a TMR block (the sub-circuit between the two consecutive voters), to multiple blocks of TMR stages in distributed TMR architectures.

Single-Block Level Diagnosis

While MTV, as can be seen in Table 4.6, does not provide any diagnosis information, however RTV provides useful information on $D_1$ and $D_2$ which can be used to indicate faulty input line. Without loss of generality, here we consider bit-flips on inputs of the voter. This bit-flip may be due to a single fault in functional block (MGF or RGF), or multiple faults on the voter of previous TMR stage. The faulty input (indicating the block producing faulty values) is determined by garbage outputs of RTV. Table 4.9 shows diagnosis information of a TMR block by using garbage outputs resulting from the voter of that stage ($D_1$ and $D_2$). The resolution of diagnosis depends on the granularity of the triplicate sub-circuits. In other words, if each TMR stage consists of three copies of a sub-circuit with $k$ gates along with an RTV voter, the resolution of diagnosis is a block of $k + 8$ gates ($k$ gates of a copy of functional block and 8 gates of RTV).

Fig. 4.10 shows an example of a TMR system using an RTV block. Three different values are placed for each node corresponding to three cases: 1) Fault-free situation (left-most value). 2) Fault on $DI_2$ (middle value): fault is masked in this case and the address of the faulty input ($DI_2$) is produced on the address outputs of the RTV, i.e. $A_1A_0 = 10$. 3) There are two faults in the circuit (right-most value): one on the $DI_1$ (data input) and the other one on control input $C_1$. These faults affect one of the data outputs and the address of faulty data input is indicated by address outputs ($A_1A_0 = 01$ since $DI_1$ is faulty). Note that garbage outputs only show the location of the fault on data inputs which is used to identify faulty sub-circuits for the repair/replacement phase.
Table 4.9: Offline fault diagnosis based on address outputs of RTV

<table>
<thead>
<tr>
<th>$D_1$</th>
<th>$D_2$</th>
<th>Situation</th>
<th>Fault location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>$O_1 = O_2 = O_3$</td>
<td>no fault on data inputs / no fault on control inputs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_2 = O_3 \neq O_1$</td>
<td>no fault on data inputs / $C_2$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_3 \neq O_2$</td>
<td>no fault on data inputs / $C_1$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_2 \neq O_3$</td>
<td>no fault on data inputs / Both $C_2$ and $C_1$ are faulty.</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>$O_1 = O_2 = O_3$</td>
<td>$I_1$ is faulty / no fault on control inputs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_2 = O_3 \neq O_1$</td>
<td>$I_1$ is faulty / $C_2$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_3 \neq O_2$</td>
<td>$I_1$ is faulty / $C_1$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_2 \neq O_3$</td>
<td>$I_1$ is faulty / Both $C_2$ and $C_1$ are faulty.</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>$O_1 = O_2 = O_3$</td>
<td>$I_2$ is faulty / no fault on control inputs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_2 = O_3 \neq O_1$</td>
<td>$I_2$ is faulty / $C_2$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_3 \neq O_2$</td>
<td>$I_2$ is faulty / $C_1$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_2 \neq O_3$</td>
<td>$I_2$ is faulty / Both $C_2$ and $C_1$ are faulty.</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>$O_1 = O_2 = O_3$</td>
<td>$I_3$ is faulty / no fault on control inputs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_2 = O_3 \neq O_1$</td>
<td>$I_3$ is faulty / $C_2$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_3 \neq O_2$</td>
<td>$I_3$ is faulty / $C_1$ is faulty.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O_1 = O_2 \neq O_3$</td>
<td>$I_3$ is faulty / Both $C_2$ and $C_1$ are faulty.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.10: RTV in a TMR system: Address lines and data outputs determine the fault location (values in fault-free case/values in the presence of a single fault on $DI_2$/values in the presence of two faults on $DI_1$ and $C_1$)
Multiple-Blocks Level Diagnosis

Indicating the existence/location of a fault and initiating fault diagnosis are done using garbage/diagnosis outputs of the gates. Each RTV produces two diagnosis outputs. Therefore, using \( n \) RTVs in a distributed TMR circuit results in \( 2n \) diagnosis lines. In order to identify the existence/location of the faults these lines are monitored. Observing \( 2n \) lines iteratively imposes a very high diagnosis time. In order to alleviate this, a reversible circuit is introduced that collects these diagnosis lines to a single output, which indicates the existence of (masked) faults in the circuit. It also keeps the detailed information regarding faulty blocks.

![Diagram](image1)

**Figure 4.11**: Module collecting the diagnosis information of \( i \) RTVs

Fig. 4.11 shows the schematic of the circuit which collects diagnosis lines of a set of voters. This module is called *Diagnosis Collector (DC)*. The inputs of this circuit are diagnosis outputs of a set of voters. The function output of this circuit is \( D \) which is 1 when at least one of the input lines is 1. The other outputs are copies of the corresponding inputs. By monitoring only the output \( D \), the existence of fault represented by any RTV in the circuit is identified. Thereafter, the diagnosis procedure is initiated to identify individual faulty blocks using the information on other outputs of this gate (actual RTV outputs).

![Diagram](image2)

**Figure 4.12**: Cascaded DCs to collect diagnosis information of 4 RTVs
This module can also provide a tradeoff between diagnosis resolution and diagnosis time. Clusters of RTVs can be connected to a cascaded structure of DCs as shown in Fig. 4.12. This way diagnosis procedure can identify a faulty cluster of RTVs up to the desired resolution considering the required diagnosis time. However, keeping the information at the inputs of a DC module results in more control inputs as well as garbage outputs in DC implementation. An implementation of a DC over 4 RTVs is presented in Fig. 4.13. As seen in this figure, the implementation consists of a constant garbage output (the output which constantly is 1). This constant output can be used as an input for another DC module. Therefore, the number of garbage outputs does not increase (except one garbage output for all of the DC modules). However, for each DC module an extra control line is added to the circuit.

Figure 4.13: An implementation of a DC for 4 RTVs

Simulation Results

In order to evaluate the effectiveness of the proposed voters in terms of fault masking coverage, overhead, and diagnosability, fault simulations on reversible benchmark circuits were carried out. The simulation environment is a program written in C. This program gets the original reversible circuit as the input and creates the TMR circuit by triplication.
of individual gates and adding voters to the triplicated circuit. Faults, based on MGF and RGF models, are then injected in random places (individual gates) in the circuit (on both original gates as well as the voters) and the percentage of the cases the faults are not masked is identified through fault simulation.

Table 4.10: Area comparison of the two proposed reversible voters. # G: Number of GATES, # IN: total number of INPUTS

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Original</th>
<th>TMR with Voter</th>
<th>TMR with Voter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># G</td>
<td># In</td>
<td># G</td>
</tr>
<tr>
<td>RMRLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.28.2.78</td>
<td>19</td>
<td>38</td>
<td>65</td>
</tr>
<tr>
<td>hwb5.26</td>
<td>27</td>
<td>250</td>
<td>278</td>
</tr>
<tr>
<td>c.28.2.56</td>
<td>56</td>
<td>858</td>
<td>196</td>
</tr>
<tr>
<td>c.15.2.27</td>
<td>26</td>
<td>76</td>
<td>90</td>
</tr>
<tr>
<td>g.c.20.19</td>
<td>78</td>
<td>1175</td>
<td>93</td>
</tr>
<tr>
<td>SIS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR5</td>
<td>82</td>
<td>278</td>
<td>290</td>
</tr>
<tr>
<td>missex</td>
<td>102</td>
<td>335</td>
<td>362</td>
</tr>
<tr>
<td>rd53</td>
<td>166</td>
<td>544</td>
<td>590</td>
</tr>
<tr>
<td>o64</td>
<td>171</td>
<td>732</td>
<td>609</td>
</tr>
<tr>
<td>clip</td>
<td>1025</td>
<td>3298</td>
<td>3659</td>
</tr>
</tbody>
</table>

Variety of algorithms and techniques have been proposed in the literature for logic synthesis of reversible circuits. Based on these algorithms, different tools have been developed. However, we used RMRLS synthesis tool in order to prepare reversible benchmark circuits to estimate relative overhead. RMRLS library includes Toffoli, Fredkin, Peres and Reverse Peres basic reversible gates. Synthesized circuits are represented in netlist format. As shown in Table 4.10, the five largest benchmark circuits have been chosen from the reversible benchmarks that accompany the RMRLS tool. Moreover, in order to evaluate the proposed technique for larger reversible circuits, SIS synthesis tool was also utilized. The other five benchmark circuits in Table 4.10 are created by transforming the synthesized circuit from SIS to reversible counterparts. In this process, circuits are synthesized using basic logic gates, i.e. 2-input AND, NAND, OR, and NOR gates. Then the gates are replaced with their reversible equivalent using basic reversible gates (for example logical OR is implemented by 3 Toffoli gates). This general approach can be applied for reversible synthesis of any logic circuit by performing 1- synthesizing the circuit using basic (traditional)
logic gates 2- replacing basic gates with their reversible implementations 3- triplication all reversible gates 4- adding voters to the output of individual gates.

Table 4.11: Area overhead of the two proposed reversible voters

<table>
<thead>
<tr>
<th>Circuit</th>
<th># G</th>
<th># In</th>
<th>depth</th>
<th># G</th>
<th># In</th>
<th>depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMRLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.28.2.78</td>
<td>14.0%</td>
<td>14.9%</td>
<td>0</td>
<td>31.6%</td>
<td>35.1%</td>
<td>21%</td>
</tr>
<tr>
<td>hw5.26</td>
<td>14.8%</td>
<td>3.2%</td>
<td>0</td>
<td>42.3%</td>
<td>6.0%</td>
<td>15%</td>
</tr>
<tr>
<td>c.28.2.56</td>
<td>16.7%</td>
<td>2.0%</td>
<td>0</td>
<td>37.5%</td>
<td>5.2%</td>
<td>6%</td>
</tr>
<tr>
<td>c.15.2.27</td>
<td>15.4%</td>
<td>10.5%</td>
<td>50%</td>
<td>34.6%</td>
<td>25.9%</td>
<td>130%</td>
</tr>
<tr>
<td>g.c.20.19</td>
<td>18.8%</td>
<td>2.3%</td>
<td>21%</td>
<td>33.3%</td>
<td>7.9%</td>
<td>39%</td>
</tr>
<tr>
<td>SIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR5</td>
<td>17.9%</td>
<td>8.9%</td>
<td>100%</td>
<td>40.2%</td>
<td>24.7%</td>
<td>180%</td>
</tr>
<tr>
<td>misex1</td>
<td>18.3%</td>
<td>9.5%</td>
<td>40%</td>
<td>41.2%</td>
<td>26.2%</td>
<td>75%</td>
</tr>
<tr>
<td>rd53</td>
<td>18.5%</td>
<td>9.7%</td>
<td>44%</td>
<td>41.6%</td>
<td>26.6%</td>
<td>92%</td>
</tr>
<tr>
<td>o64</td>
<td>18.7%</td>
<td>7.5%</td>
<td>107%</td>
<td>42.1%</td>
<td>20.6%</td>
<td>193%</td>
</tr>
<tr>
<td>clip</td>
<td>19.0%</td>
<td>10.3%</td>
<td>24%</td>
<td>42.7%</td>
<td>28.0%</td>
<td>60%</td>
</tr>
</tbody>
</table>

Table 4.10 represents the area of the original circuits as well as the area for the TMR circuits using the proposed MTV and RTV reversible voters. Since there is no precise delay model for reversible circuits, their relative delays are compared based on number of stages (logic depth). In this comparison it is assumed that reversible gates have unity delays and also the delay of a circuit is proportional to its logic depth. Therefore, the performance overheads of the voters are shown in terms of reversible circuit depth (stages).

Table 4.11 represents the area overhead of the MTV and RTV implementations. The area overheads are about 17.2% and 37.8% with respect to the area of triplicated circuits. The voter area overhead is calculated as the area of voter over the area of triplicated circuit. The results show that the performance overhead of the circuits obtained directly from reversible benchmarks are lower than that of circuits obtained from SIS synthesized circuits. This supports the motivation for specific synthesis techniques for reversible circuits to reduce parameters such as performance overhead, number of garbage lines, etc.

In these experiments, there is a voter for every 7 gates in the circuit, to reduce voter area and performance overheads.

Figure 4.14 presents fault masking coverage of TMR circuits created using MTV and RTV as voters for different number of multiple injected faults. Each data point in this
graph is the average coverage among all simulated circuits. In our experiments, the number of fault injections for each data point per circuit was at least 12,500. In these experiments, MGF, RGF, and bit-flip were considered as the reversible fault models. A fixed number of faults are injected into random locations of the circuit and the fault masking coverage for the corresponding number of faults in the circuit is calculated. The coverage for single fault is around 94% for MTV and 100% for RTV, verifying that RTV solves the problem of single point of failure in TMR circuits. Increasing the number of faults decreases fault masking coverage for both voter implementations. However, RTV provides better fault masking coverage than MTV as the number of multiple faults increases. So, a simple and yet reasonable conclusion is that MTV is suitable only for the systems with low error rates when the area is a major concern.

![Fault masking coverage for MTV and RTV](image)

Figure 4.14: Fault masking coverage for MTV and RTV [11]

### 4.2.3 Test Generation for Reversible Circuits

We have proposed a so-called Ping-Pong test as a new testing technique for reversible circuits [12]. The objective of Ping-Pong test is reducing the information required for high (100%) fault coverage testing. In Ping-Pong testing, a test vector is applied to the circuit and the response of the circuit for that vector is used as the next test vector. The test vector set can be represented only by the initial vector and the length of the sequence, which can
drastically reduce test data volume. This compact test generation technique can be used for efficient BIST implementation for reversible circuits.

**Proposed Method: Ping-Pong Testing**

The objective of Ping-Pong test is to derive the minimum amount of information required for a test set to achieve full coverage for testing reversible circuits. Without loss of generality, we assume single SMGFs/RGFs as the target fault model.

![Figure 4.15: The schematic of a reversible circuit (Circuit 3_17 [15])](image)

![Figure 4.16: An example of Ping-Pong test vector: (a) the truth table and the fault coverage of each input pattern (b) a Ping-Pong sequence to detect all faults (\(V^1 = 000\), \(V^2 = 111\)); \(T = (000, 2, 101)\)](image)

Operation of a reversible circuit on an input pattern \(V^m\); \(V^m = \{i_1^m, i_2^m, ..., i_n^m\}\) is expressed as \(f(V^m)\); \(f(V^m) = \{o_1^m, o_2^m, ..., o_n^m\}\). In a Ping-Pong test, the subsequent test vector is the response of the circuit to the previous test vector, i.e. in a sequence of \(k\) test vectors \(\{V^1, V^2, ..., V^k\}\), \(V^j = f(V^{j-1})\); \(\forall j; 2 \leq j \leq k\). This can greatly reduce the amount of test data volume. If there is such test vector sequence, then instead of storing...
all of the vectors, testing can be identified by three values: i- the first test vector \((V^1)\), ii- the number of required vectors \((k)\), which indicates the number of test cycles, and iii- the expected output pattern for the last test vector in the sequence \(f(V^k)\). So, instead of storing \(2kn\) bits for test vectors and responses, only \(2n + \log k\) bits of information is enough. A Ping-Pong test is defined as \((V^1, k, f(V^k))\). It is obvious that all these \(k\) test vectors in a Ping-Pong test can be ordered in a unique sequence, i.e., there is no repetitions, which is implied from reversibility. Therefore, for each reversible circuit, a set of Ping-Pong test vectors is identifiable. Ideally one Ping-Pong test can be used to cover all faults. However, in order to reduce test time \((k)\) and improve coverage, various Ping-Pong test sessions might be needed \(\{V^1_i, k_i, f(V^1_i)\}, i = 1 \text{ to } N\).

An example of a sequence of input vectors which has the Ping-Pong property is shown in Figure 4.16. Ping-Pong test can be identified as \(T = (V^1, k, f(V^k))\), where, \(T\) is the required test information. This information can be translated as applying a sequence of \(k\) test vectors \((k\) cycles), starting from \(V^1\) ending to final output \(f(V^k)\). For example, \(T = (000, 2, 101)\) shows the Ping-Pong test of the circuit in Figure 4.15. Please note, the fault coverage of a sequence of test vectors is less than or equal to the fault coverage of each individual test vectors. In other words, if a fault is not detected by none of \(V^1, V^2, \ldots, \text{ and } V^k\) test vectors, then that fault is not detected by a sequence constructed by these vectors.

**Implementation of Test Sequence Generation**

The concept of Ping-Pong test is general and can be applied to various fault models. However, the purpose of Ping-Pong test sequence generation is to minimize test time (total number of cycles) and test information (the number of Ping-Pong test sessions) while achieving the required coverage for a given fault model. The fault models considered in this implementation include *Single Missing Gate Fault* (SMGF), *Single Stuck-at 0* (SSA0), and *Single Stuck-at 1* (SSA1). It must be mentioned, any test which detects SMGF, it also detects Repeated Gate Fault (RGF), too. The proposed algorithm is a heuristic greedy algorithm. Its operation on a circuit with \(m\) gates and \(n\) inputs consists of the following
steps:

1. Find coverage list of each gate: using back tracing for each gate the vectors which can detect the fault on that gate are identified. In this step, fault(s) on the gate is activated, then by using back tracing coverage list of the gate is identified. An input pattern can detect a SMGF on a gates if it justifies the logical values of 1 at all of the control inputs of that gate. Furthermore, in this implementation in order to reduce the complexity of search space, all inputs of the gate are forced to logical 1 to detect any SSA0 on the input lines of the gate. Similarly, SSA1 is reduced to all inputs of the gate (all 0 on the inputs of the gate activates the fault).

2. Assign a weight for each gate: The weight of a gate is defined as one over the number of vectors in its coverage list.

3. Construct list of not-detected faults: initially all of the gates are in the list of not-detected faults.

4. Select a gate: A gate is randomly selected and removed from the list of not-detected faults. The probability of selecting a gate is proportional its weight.

5. Select a vector from the coverage list of the selected gate: A vector from the coverage list of the selected gate is selected. However, among all vectors in that coverage list, the vector which belongs to the maximum number of coverage lists is selected. All detected faults by the selected vector are removed from the list of not-detected faults.

6. Construct a test session: A test session $S = (I, C, E)$ is created. $I$ is the selected vector, $C$ is 1 and $E$ is the response of the circuit for $I$. The following steps are repeated, until $C$ is less than $L$ (a threshold on the maximum cycles of a test session).

- Find response of the circuit to $E$ (call it $E'$).
- If $E'$ is not visited yet, then, remove all gates which are detected by $E'$ from the not-detected list.
7. Run fault simulation and update the list of not-detected faults.

8. If not-detected list is not empty, then repeat from step 4.

9. Merge test sessions: Two test sessions \( S_1 = (I_1, C_1, E_1) \) and \( S_2 = (I_2, C_2, E_2) \) can be merged if starting from \( E_1 \) after \( k \) cycles response of the circuit is \( I_2 \) or starting from \( E_2 \) after \( k \) cycles response of the circuit is \( I_1 \). However, \( C_1 + C_2 + k \) must be less than threshold on the number of cycles \( (L) \).

Table 4.12: Ping-Pong test sessions generated for 100% coverage of SMGF (P-SMGF), SSA0 (P-SSA0), SSA1 (P-SSA1), and all these three fault models (P-Merge). the number of numbers inside of brackets shows the number of test sessions, and each number show the number of cycles for the corresponding test session. \( m \) indicates the number of gates and \( n \) indicates the number of inputs of the corresponding circuit.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>( m )</th>
<th>( n )</th>
<th>P-SMGF</th>
<th>P-SSA0</th>
<th>P-SSA1</th>
<th>P-Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>ham3tc</td>
<td>5</td>
<td>3</td>
<td>{1,1}</td>
<td>{1,1}</td>
<td>{1}</td>
<td>{2,1,1}</td>
</tr>
<tr>
<td>3_17tc</td>
<td>6</td>
<td>3</td>
<td>{2}</td>
<td>{2}</td>
<td>{2}</td>
<td>{4,1}</td>
</tr>
<tr>
<td>mod5d1</td>
<td>8</td>
<td>5</td>
<td>{1}</td>
<td>{1,1,1}</td>
<td>{1}</td>
<td>{1,1,1}</td>
</tr>
<tr>
<td>2of5d2</td>
<td>12</td>
<td>7</td>
<td>{6}</td>
<td>{9}</td>
<td>{1}</td>
<td>{3,3,1}</td>
</tr>
<tr>
<td>mod5adders</td>
<td>17</td>
<td>6</td>
<td>{4,1}</td>
<td>{4,4}</td>
<td>{1,3}</td>
<td>{4,4,1}</td>
</tr>
<tr>
<td>5mod5tc</td>
<td>17</td>
<td>6</td>
<td>{1}</td>
<td>{1,1}</td>
<td>{1}</td>
<td>{1,1,1}</td>
</tr>
<tr>
<td>5mod5_10_71a</td>
<td>10</td>
<td>6</td>
<td>{1,1}</td>
<td>{5}</td>
<td>{1}</td>
<td></td>
</tr>
<tr>
<td>mspk_nth_primes4_11</td>
<td>11</td>
<td>4</td>
<td>{4}</td>
<td>{3,3}</td>
<td>{1,1,2}</td>
<td>{5}</td>
</tr>
<tr>
<td>mspk_4_49_14</td>
<td>14</td>
<td>4</td>
<td>{1,1}</td>
<td>{4}</td>
<td>{1}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>mspk_nth_primes4_14</td>
<td>14</td>
<td>4</td>
<td>{4}</td>
<td>{5}</td>
<td>{1}</td>
<td>{7}</td>
</tr>
</tbody>
</table>

Repeat steps 3-9 for \( L = 2 \) to \( L = m \), and find the best test session (s). Using this method, a set of test sessions to detect each individual fault model is generated. However, there are two methods to generate a test which can detect all three fault models. The first method is generating tests for individual fault models, then merging the generated test sessions. Using this method redundant test sessions will be generated which is the major disadvantage of this method. The second method is running algorithm on the three fault models sequentially. In this method, after generating the test sessions for a fault, the fault model is changed and the list of not-detected faults is reduced by fault simulation using the generated test sessions.
CHAPTER 4. RELIABLE REVERSIBLE CIRCUIT DESIGN

Simulation Results

In order to evaluate the efficiency of the proposed Ping-Pong testing method, it is verified on a set of reversible benchmark circuits [15]. Our implemented flow extracts the required information for Ping-Pong testing of each benchmark circuit. This information includes the required sequences in the forms of \((V^1, k, f(V^k))\), where \(V^1\) is the starting vector of the sequence, \(k\) indicates the number of cycles in the testing, and \(f(V^k)\) indicates the expected output for the sequence. In this evaluations four different Ping-Pong tests are generated as follows:

1. P-SMGF: Ping-Pong test which provides 100% coverage for Single Missing Gate Fault, and as a result for Single Repeated Gate Fault.
2. P-SSA0: Ping-Pong test which provides 100% coverage for Single Stuck-at 0.
3. P-SSA1: Ping-Pong test which provides 100% coverage for Single Stuck-at 1.
4. P-Merge: Ping-Pong test which provides 100% coverage for all above mentioned faults.

Table 4.12 shows the simulation results. The first three columns of this table show the circuit names, the number of gates in the circuit \((m)\), and the number of inputs \((n)\). Extracted Ping-Pong sessions are shown in the next four columns, in terms of the number
Figure 4.18: The fault coverage of the four Ping-Pong tests with respect to Multiple Single Missing Gate Faults

of test cycles per test session. In this columns the number of numbers inside of brackets shows the number of test sessions, and each number shows the number of cycles for the corresponding test session. For example, "\{2,1,1\}" means that the test consists of three test sessions, and the number of cycles for the test sessions are 2, 1, and 1.

Figure 4.19: The fault coverage of the four Ping-Pong tests with respect to Multiple Stuck-at-0 Faults

The sequences are simulated with respects to different fault models. In each simulation, the fault is injected to the circuit, and then the sequences are applied to the faulty circuit. If the output of the faulty circuit for at least one of the sequences deviates from
the expected output, then the fault is marked as detected. The fault models used in these simulations include: *Multiple Missing Gate Fault* (MMGF), *Multiple Single Missing Gate Fault* (MSMGF), *Multiple Stuck-at 0* (MSA0), and *Multiple Stuck-at 1* (MSA1). In the fault simulation, the circuits are simulated for $10^6$ random faulty cases. Figures 4.17 and 4.18 show fault coverage of the four Ping-Pong tests with respect to MMGF and MSMGF, respectively. Fault coverage of the four Ping-Pong tests with respect to MSA0 and MSA1 is shown in figures 4.19 and 4.20, respectively.

Figure 4.20: The fault coverage of the four Ping-Pong tests with respect to Multiple Stuck-at 1 Faults
Chapter 5

Summary and Conclusions

Various emerging implementation technologies (such as crossbar nano-architectures) as well as alternative computational approaches (such as reversible circuits) have been investigated as the potential methods to overcome the challenges raised in conventional CMOS technology in continuation of Moore’s law. However, in nanoscale regimes the atomic scale of devices as well as poor control in nanofabrication raise reliability issues in terms of manufacturing yield, predictability in the presence of extreme process variation, testing, and runtime reliability. Hence, in a viable emerging technology, these reliability issues must be addressed adequately. Therefore, in this thesis we proposed a set of methods to address the reliability concerns in crossbar nano-architectures, as an example of alternative implementation, as well as reversible logic, as an example of alternative computational technology.

In terms of crossbar nano-architectures, the proposed methods include an architectural approach by introducing the self-timed nano-architecture as well as a set of logic transformations and algorithms to provide defect and variation tolerant configuration [1, 2, 3, 4, 5, 6, 7]. The proposed self-time crossbar nano-architecture allows us to eliminate global clock-like signals (by replacing with local handshake signals) to reduce circuit vulnerability to delay variation. While, in the logic mapping approaches, different configurations of a logic function on a crossbar nano-architecture are explored to provide variation and defect tolerant mapping. The simulation results indicate that these techniques improve the reliability of the
circuits by providing robust functionality, which enables realization of high yield circuits.

In the field of reversible logic, we proposed a set of methods to enable online and offline testing of reversible circuits as well as fault masking and diagnosis [8, 9, 10, 11, 12]. In order to provide online testing, a parity generation and preserving methodology was introduced. We also proposed a set of reversible majority voting gates which can be used in Triple Modular Redundancy structure to enable both fault masking and fault diagnosis. Furthermore, Ping-Pong test generation methodology was proposed to provide test patterns with small amount of test information to detect reversible faults. The simulation results indicate that, the proposed techniques improve reliability of reversible circuits by increasing testability of these circuits with respect to various fault models.
Bibliography


