TECHNISCHE UNIVERSITÄT WIEN

Institut für Computertechnik
Institute of Computer Technology

A MASTER THESIS ON

Fault-tolerant Design Techniques for Nano-Mechanical Circuits

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Diplom-Ingenieur
(Equivalent to Master of Science)

in

Embedded Systems (066 504)

by

Dominik Rudolf

1225777

Supervisor(s):

Univ.Prof. Dipl.-Ing. Dr.techn. Axel Jantsch

M.sc. Ardavan Elahi

Vienna, Austria

December 2024

Abstract

The shift from conventional manufacturing and monitoring techniques to systems with decentralised, interconnected, and increasingly self-aware components has created a demand for edge-devices that are capable of operating under harsh environmental conditions, such as high temperatures or high radiation doses. Due to constrained resources and limited supply, these edge-devices also require efficient circuitry with low leakage current. Traditional complementary metal-oxide-semiconductor (CMOS) technology is not suitable for meeting these requirements.

The i-EDGE project aims to establish nanoelectromechanical (NEM) relays as substitutes for CMOS transistors to address the aforementioned challenges. However, at the current Technology Readiness Level (TRL) of the project, the yield of manufactured NEM circuits, as well as the reliability under operation, is significantly lower than in CMOS technology. To tackle this issue, it is necessary to explore fault-tolerant design techniques that can be applied to NEM circuits.

This thesis first derives fault models from the physical defects that can occur in NEM devices. Conventional fault-tolerant design techniques are then reviewed and assessed for their applicability to NEM circuits. The unique characteristics of NEM devices are leveraged to simplify or enhance established fault-tolerant circuit designs, as well as to propose new NEM circuit designs. A simulation environment is developed to simulate test circuits with fault-tolerant design techniques applied. The resulting data is used to explore the design space and to evaluate the achievable fault tolerance.

Kurzfassung

Der Wandel von konventionellen Produktions- und Überwachungssystemen hin zu Systemen mit dezentralisierten, vernetzten und zunehmend intelligenten Geräten sowie Edge-Computing hat eine Nachfrage nach Komponenten geschaffen, die in der Lage sind, unter extremen Umweltbedingungen wie hohen Temperaturen oder hohen Strahlungsdosen zu arbeiten. Aufgrund begrenzter Hardware-Ressourcen und eingeschränkter Energieversorgung erfordern diese Edge-Geräte zudem häufig effiziente Schaltungen mit minimalem Leckstrom. Traditionelle Schaltungen auf Basis von CMOS sind daher oft nicht geeignet, diese Anforderungen zu erfüllen.

Das i-EDGE-Projekt hat zum Ziel, NEM-Relais als Ersatz für CMOS-Transistoren zu etablieren, um die oben genannten Herausforderungen zu bewältigen. Zum aktuellen Stand des TRLs dieses Projekts liegen sowohl der Ertrag der fabrizierten NEM-Schaltungen als auch die Zuverlässigkeit im Betrieb deutlich unter dem Niveau der CMOS-Technologie. Um dieses Problem zu lösen, ist es erforderlich, fehlertolerante Designtechniken auf ihre Eignung für NEM-Schaltungen zu prüfen.

Diese Masterarbeit leitet zunächst Fehlermodelle aus physischen Defekten ab, die in NEM-Relais auftreten können. Konventionelle fehlertolerante Designtechniken werden anschließend hinsichtlich ihrer Anwendbarkeit auf NEM-Schaltungen bewertet. Die einzigartige Funktionsweise von NEM-Relais wird genutzt, um etablierte fehlertolerante Schaltungsdesigns zu vereinfachen oder zu verbessern sowie um neue NEM-Schaltungsdesigns vorzuschlagen. Eine Simulationsumgebung wird entwickelt, um die fehlertoleranten Designtechniken mithilfe von Testschaltungen zu simulieren. Anhand der resultierenden Daten werden verschiedene Designs bewertet und die erreichbare Fehlertoleranz evaluiert.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Copyright Statement

I, Dominik Rudolf, hereby declare that this thesis is my own original work and, to the best of my knowledge and belief, it does not:

- Breach copyright or other intellectual property rights of a third party.
- Contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
- Contain material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.
- Contain substantial portions of third party copyright material, including but not limited to charts, diagrams, graphs, photographs or maps, or in instances where it does, I have obtained permission to use such material and allow it to be made accessible worldwide via the Internet.

Signature: _____

Vienna, Austria, December 2024 Dominik Rudolf

Acknowledgment

This thesis marks the final step of a long and eventful journey that I decided to go on many years ago. Like most ventures that seem overwhelming at first, this one wasn't accomplished alone. They become manageable only through the helping hands that prepare one's luggage both before and during the journey, are made more pleasant by the companionship of fellow travellers met along the way, and often involve unexpected losses that allow us to see the pathway we walk with sharper eyes. Similarly, this chapter of my personal journey, embodied in an academic career and culminating in this thesis, would not have been imaginable without such people. To them, I would like to express my heartfelt thanks in the following lines.

First and foremost, I would like to thank my colleagues at i-EDGE, who enabled me to carry out my thesis as part of this significant project. Not only were they always willing to answer technical questions of all kinds in great detail, but they also gave me the opportunity to travel to various parts of the world. I have always enjoyed the personal meetings we had. Looking back, I must say that participating in a multinational project has been an invaluable experience that I would not want to miss. I hope that with this thesis, I have been able to make a small contribution to its progress.

I owe special thanks to Prof. Dr. Axel Jantsch, who gave me the opportunity to work on this thesis and guided me throughout the process in a highly professional manner. His far-reaching perspective always enabled me to grasp what was essential in this work. Without his corrective hand, the scope of this thesis would have become far too extensive, making it nearly impossible to finish. Additionally, it is hard to put into words the immense patience my co-supervisor, M.Sc. Ardavan Elahi, showed with me and this thesis over several months. His expertise and encouraging words enabled me to overcome every obstacle, no matter how large, and even under significant time pressure, he never failed to provide me with the best possible support until the very end. Without the assistance of these two individuals, this work would not have reached its completed form. I therefore owe them my deepest gratitude.

This would not be a proper acknowledgment but an affront if I did not mention the colleagues who ended up sharing an office with me. Above all, I would like to thank Philipp, with whom I had countless conversations which led to fruitful thoughts, ideas, and impulses for my work. Thanks also

goes to my Dutch colleagues Jan and Jasper, particularly for introducing me to the Dutch language to an extent sufficient for basic communication (Ik hou van Nederlandse vrouwen). Furthermore, I would like to thank Sofia, who always helped me with organizational matters and transformed my presentations into something reasonable by giving countless suggestions for improvement. I further extend my gratitude to all the other colleagues and staff at ICT for always being helpful and for various distractions that helped reduce the effective working hours to a bearable level.

I would also like to thank all my former fellow students who became my companions and friends during my studies, whose names are too numerous to mention. Without them, the challenging academic path I pursued would have been not only significantly more arduous but also lacking many memorable experiences. At this point, I would also like to thank my friends outside the academic sphere: Dominik, Martin, and Daniel, who have accompanied me since childhood and adolescence; René and Daniel, who reliably provided the necessary motivation during every training session to keep my muscles in a hypertrophic rather than atrophic state throughout my studies; and Patrick, who has passed away but always brought the right energy to our group and knew how to pleasantly challenge and surprise me with technical discussions during his lifetime.

Finally, I extend my deepest thanks to my parents, Gerhard and Monika, who made this journey possible through their continuous support, and to my deceased sister Sarah, who was a pillar of strength throughout my whole life. Without you, this journey would not have reached its conclusion.

Danksagung

Die vorliegende Arbeit stellt den Schlussstein einer langen und ereignisreichen Reise dar, die ich vor etlichen Jahren beschloss, anzutreten. Wie bei solchen, anfangs unmöglich scheinenden Unternehmungen üblich, bewältigt man diese nicht im Alleingang. Sie werden gangbar gemacht durch eine Vielzahl unterstützender Hände, die einem vor und während der Reise das Weggepäck bereiten, werden erleichtert durch die Begleitung von Weggefährten, die man auf selbigem Wege trifft, und beherbergen oft auch unerwartete Verluste, die dem Weg erst eine schärfere Zeichnung verleihen. So ist auch jener Abschnitt meines persönlichen Lebensweges, der sich in Form einer akademischen Laufbahn ausgestaltete und mit dieser Arbeit seinen Abschluss findet, nicht ohne solche Menschen denkbar. Ihnen sei in den folgenden Zeilen mein Dank ausgesprochen.

Zuallererst möchte ich den Kollegen von i-EDGE danken, die es mir ermöglicht haben, meine Abschlussarbeit im Zuge dieses bemerkenswerten Projekts durchzuführen. Nicht nur waren sie stets bemüht, mir technische Fragestellungen aller Art in ausführlicher Weise zu beantworten, sie eröffneten mir auch die Möglichkeit, verschiedene Orte der Welt zu bereisen. Die persönlichen Zusammenkünfte in den Meetings habe ich stets genossen. Rückblickend möchte ich festhalten, dass die Mitarbeit an einem multinationalen Projekt eine wertvolle Erfahrung für mich darstellt, die ich keinesfalls missen möchte. Ich hoffe, dass ich mit dieser Arbeit einen kleinen Beitrag zu dessen Fortschritt beisteuern konnte.

Besondere Beachtung verdient in dieser Danksagung Prof. Dr. Axel Jantsch, der mir diese Abschlussarbeit ermöglichte und mich in weiterer Folge, im wahrsten Sinne des Wortes, professionell betreute. Sein richtender Blick half mir stets, das Wesentliche dieser Arbeit zu erfassen. Ohne seine korrigierende Hand wäre dieses, letztlich umfangreiche Werk, zudem vollends ausgeufert und hätte seinen Abschluss nur schwerlich gefunden. Zum Weiteren ist es nur schwer in Worte zu fassen, wie viel Geduld mein Mitbetreuer, M.sc. Ardavan Elahi, über mehrere Monate hinweg mit mir und dieser Arbeit aufgebracht hat. Seine Expertise und motivierenden Worte befähigten mich, jede noch so große Hürde zu meistern, und selbst unter großem Zeitdruck versäumte er es nicht, mich bis zuletzt bestmöglich zu unterstützen. Ohne die Hilfe jener beiden Personen würde diese Arbeit nicht in ihrer

abgeschlossenen Form vorliegen. Ihnen gebührt deshalb mein ausdrücklicher Dank.

Dies wäre keine Danksagung, sondern ein Affront, wenn nicht auch jene Kollegen erwähnt würden, die das Schicksal mit mir in einem Institutsbüro zusammengeführt hat. Allen voran möchte ich deshalb Philipp danken, mit dem ich eine nicht unerhebliche Anzahl an Konversationen führte, aus denen fruchtbare Gedanken, Ideen und Anstöße für meine Arbeit erwuchsen. Dieser Dank gilt auch meinen niederländischen Kollegen Jan und Jasper, insbesondere aber dafür, mir die niederländische Sprache in einem Ausmaß näher gebracht zu haben, das zur Verständigung vollkommen ausreicht (Ik hou van Nederlandse vrouwen). Des Weiteren möchte ich mich bei Sofia bedanken, die mir stets bei organisatorischen Fragen weiterhalf und durch unzählige Verbesserungsvorschläge auch meine Präsentationen in eine zumutbare Form brachte. Außerdem danke ich allen übrigen Kollegen und Mitarbeitern des ICT für ihre Hilfsbereitschaft und allerlei sonstigen Ablenkungen, um die effektiven Arbeitsstunden auf ein erträgliches Maß zu reduzieren.

Ich danke zudem allen meinen ehemaligen Kommilitonen, die mir während meiner Studienzeit als Weggefährten und Freunde zuteilwurden und deren Namen zu zahlreich sind, um sie alle zu nennen. Ohne sie wäre die akademische Laufbahn, die ich einschlug und die von vielen Herausforderungen durchzogen war, nicht nur um ein Vielfaches mühseliger, sondern auch um das ein oder andere Gelage ärmer gewesen. An dieser Stelle möchte ich auch meinen Freunden außerhalb des universitären Bereichs Dank aussprechen: Dominik, Martin und Daniel, die mich schon seit Kindheits- und Jugendtagen begleiten, René und Daniel, die zuverlässig in jedem Training den nötigen Ansporn lieferten, um meine Muskulatur während meiner gesamten Studienzeit in den hypertrophen anstatt in den atrophen Zustand zu versetzen, sowie dem verstorbenen Patrick, der stets den nötigen Schwung in die Runde brachte und es verstand, mich mit fachspezifischen Gesprächen zu überraschen und angenehm herauszufordern.

Abschließend danke ich ganz besonders meinen Eltern Gerhard und Monika, die mir durch Unterstützung jeglicher Art diesen Lebensweg erst ermöglicht haben, sowie meiner verstorbenen Schwester Sarah, die mir zu allen Lebenszeiten eine Stütze war. Ohne euch hätte diese Reise keinen Abschluss gefunden.

Contents

List of Tables

List of Figures

Acronyms

3-T 3-terminal. xv, 2, 3, 10, 12, 35, 36, 37

4-T 4-terminal. xv, 3, 35, 36, 37, 38

7-T 7-terminal. xv, 3, 4

ALU arithmetic logic unit. 29

CFRA Circuit Failure Rate. xvi, xvii, 48, 49, 54, 55, 57, 59, 60, 65, 66, 67, 68, 71, 73, 74, 75, 77, 79, 80, 81, 82

CFRE Circuit Fault Resilience. xvi, xvii, 49, 54, 56, 57, 58, 61, 62, 63, 64, 67, 69, 70, 71, 72, 74, 76, 77, 78

CMOS complementary metal-oxide-semiconductor. iii, iv, xi, 1, 2, 7, 10, 13, 14, 16, 18, 20, 21, 22, 24, 26, 28, 30, 32, 34, 35

CRC Cyclic Redundancy Check. 31

FD Fault Detection. 14, 15, 16

FIL Fault Injection Logic. 9, 45

IC integrated circuit. 1

IoT Internet of Things. 1

NCG nano-crystalline graphite. 8

NEM nanoelectromechanical. iii, iv, xi, xvi, xvii, 1, 2, 3, 4, 7, 9, 10, 13, 16, 17, 21, 22, 25, 26, 28, 30, 32, 35, 36, 37, 38, 39, 40, 46, 59, 61, 63, 67, 70, 83, 84

NMR N-Modular Redundancy. 17, 26, 27

PGM Probabilistic Gate Model. 43, 44

QD Quadded Design. xv, 22, 23, 24, 26, 74, 77

QL Quadded Logic. xv, 22, 23, 24

SCM Stochastic Computation Model. xvi, 43, 44, 45, 46, 47

SET Single Event Transient. 7

SEU Single Event Upset. 7

SOI silicon-on-insulator. 2

TD Tripled Design. xv, 21, 22, 23, 25, 26, 74, 77, 80

TMR Triple Modular Redundancy. xi, xv, xvi, xvii, 16, 17, 18, 23, 25, 26, 35, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 77, 80, 83, 84

TRL Technology Readiness Level. iii, iv

VIV Vulnerable Input Vector. 49

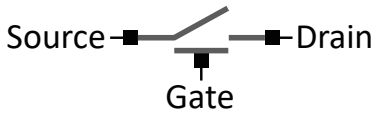
Chapter 1

Introduction

The continuous advancements in integrated circuit (IC) technology and artificial intelligence have lead to a paradigm shift in the development and application of technological devices, especially within the industrial sector. Conventional manufacturing practices are gradually being replaced by automated processes that rely on smart, self-monitoring devices with high interconnectivity. To describe this evolution, the term Internet of Things (IoT) was coined, which refers to a system that integrates sensors, actuators, controllers, cloud applications, and data storage within a communication network. In this network, components that monitor or directly interact with physical processes often have to deal with harsh environmental conditions, such as high temperatures or high radiation doses. Additionally, real-time requirements often need to be considered, which makes it reasonable to design these components as edge devices. These edge devices have limited resources and power supply, so there is a need for light-weight algorithms and technology with low leakage current.

The use of complementary metal-oxide-semiconductor (CMOS) technology in extreme environment is usually not practicable, as CMOS suffers from inherent limitations. High temperatures reduce carrier mobility, leading to reduced transistor switching speed and overall degraded circuit performance. Additionally, increased leakage currents cause power consumption to rise and a shift of the threshold voltage in MOSFETS may affect the switching characteristics of the transistor. Moreover, radiation effects can cause transient or permanent errors in the circuit, such as bit flips, latch-ups etc. Due to these circumstances, several other technological alternatives have been proposed. However, alternative technologies often require complex manufacturing processes in specialist foundries and cannot be integrated into mainstream semiconductor factoring.

The i-EDGE project introduces nanoelectromechanical (NEM) relays as a promising solution. NEM relays can act as a substitute for complementary CMOS transistors when designing components for edge applications. It stays operational at temperatures up to $300\,^{\circ}\mathrm{C}$ [14], has high radiation hard-



(a)

Gate	Source	Drain
0	0	Z
0	1	1
1	0	0
1	1	Z

(b)

Figure 1.1: 3-T relay: (a) schematic, (b) truth table.

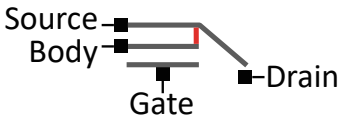
ness [15] and operates with zero standby power [16–18]. The NEM technology consists of different device primitives used for modelling logic circuits and non-volatile memory. The following section elaborates on the design and the functional behaviour of these NEM devices.

1.1 NEM Devices in i-EDGE

The i-EDGE project introduces NEM switching technology to substitute or complement conventional CMOS technology. The basic set of NEM devices for modelling logic circuits and non-volatile memory consists of three different device types. These NEM relays are fabricated on a wafer using a standard silicon-on-insulator (SOI) CMOS foundry process. This wafer is then placed on top of a prefabricated metal interconnect stack, which can be shared with CMOS circuitry located on the bottom side. Thus, the fabrication of NEM circuits can be integrated into established mainstream foundries while maintaining compatibility with CMOS technology [19–21]. The modelling and fabrication of these devices are done by project partners and are not part of this thesis.

1.1.1 3-Terminal Relay

The 3-terminal (3-T) device is a simple relay with a gate, source, and drain terminal (Fig. 1.1a). Its geometry features two parallel beams that are connected via a mechanical coupler and act as a cantilever [22, 23]. The relay can be actuated by applying a potential difference between the source and the gate. The voltage required to pull the beams in is called pull-in voltage. When the beams are actuated, an electrical contact between the source and the drain is established. The beams pull out and return to their original position when the voltage is removed. The functionality of the relay is depicted in the truth table shown in Fig. 1.1b. Digital logic gates in complementary CMOS technology are modelled using PMOS and NMOS transistors, which constitute the pull-up and pull-down networks. Since the pull-in voltage can be applied to either the gate or the source terminal to actuate the 3-T device, the switching characteristics of NMOS and PMOS transistors can be emulated. This allows for the modelling of logic gates using only 3-T devices.



(a)

Gate	Body/Beam	Source	Drain
0	0	0/1	Z
1	0	0/1	0/1
0	1	0/1	0/1
1	1	0/1	Z

(b)

Figure 1.2: 4-T relay: (a) schematic, (b) truth table.

1.1.2 4-Terminal Relay

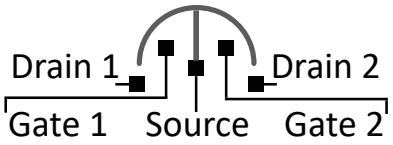
The 4-terminal (4-T) device is geometrically equivalent to the 3-T device [23]. However, in the 4-T device, the coupler connecting the two beams is electrically insulating (Fig. 1.2a). Thus, the data signal, applied at the source, is separated from the actuation voltage (Fig. 1.2b). To actuate the relay, a potential difference exceeding the pull-in voltage has to be established between the gate and the body. Therefore, this device extends the functionality of the 3-T device and its utilization allows for designs with lower device count. Applications include multiplexer and demultiplexer designs, as well as writing and reading circuitry for non-volatile memory, when used in conjunction with the other NEM relays.

1.1.3 7-Terminal Relay

The 7-terminal (7-T) device is a non-volatile relay that can perform rotational movements and exerts non-volatile behaviour [24,25]. It has two gate terminals, two drain terminals, and one source terminal, which is connected to a semicircular beam (Fig. 1.3a). In its neutral position, the relay is actuated by applying a voltage to one of the gate terminals. When the applied voltage exceeds the actuation threshold, the potential difference between the driven gate and the beam is large enough to overcome the stiffness of the hinge to which the beam is anchored. The beam then rotates either clockwise or anti-clockwise, depending on which gate is driven, and establishes electrical contact with one of the drains. When the actuation voltage is removed or both gates have the same potential difference to the beam, it remains in its position, due to adhesive forces. To move the beam to the opposite direction, the other gate has to be driven, to overcome the adhesion force between the beam and the drain. The functionality of the relay is depicted in the truth table shown in Fig. 1.3b. The non-volatile behaviour of this relay allows it to store information, making it suitable for modelling non-volatile memory [26].

1.2 Research Questions and Expected Results

Due to the novelty of this technology, the manufacturing of NEM devices is not mature enough to fabricate circuits with sufficient yield. Moreover, physical defects in NEM devices may occur during



(a)

Gate 1	Gate 2	Source	Drain 1	Drain 2
0	0	0	Z	Z
0	1	0	Z	0
0	1	1	1	Z
1	0	0	0	Z
1	0	1	Z	1
1	1	1	Z	Z
0	0	X	Z	Z
1	1	X	Z	Z

(b)

Figure 1.3: 7-T relay: (a) schematic, (b) truth table.

operation and affect their functionality, worsening the circuit's reliability. Therefore, it is necessary to make the circuit fault tolerant during the design phase by applying and evaluating various design techniques. Consequently, this thesis addresses the following research questions:

- What are suitable fault models for NEM circuits?
- Which established fault-tolerant circuit techniques can be utilized on NEM-based circuits?
- Can NEM devices be utilized to enhance existing fault-tolerant design techniques or to propose new designs?
- What fault tolerance can be obtained?

In the first step, a fault model is formulated. The physical defects that potentially occur in NEM devices are discussed. Based on this analysis, fault models are defined at different abstraction levels. The purpose of a fault model is to provide an abstract representation of how physical defects in NEM relays impact the circuit at the specified abstraction levels. Thus, it is expected that the fault models depict the real physical behaviour of defective components with sufficient accuracy and provide the necessary information on how to inject the corresponding faults into the modelled circuits during simulation.

In the next step, conventional fault-tolerant design techniques are reviewed and assessed for their applicability to NEM circuits. Fault-tolerant design techniques usually exploit some form of redundancy in the domains of hardware, time or information. Different concepts of design techniques are reviewed and elaborated upon. The chosen design techniques are expected to be suitable for NEM technology and will be applied to experimental circuits later on. Further, it is examined whether specific design techniques can be simplified or enhanced through the use of NEM devices. Additionally, new designs leveraging the unique functionalities of NEM devices are proposed.

To simulate the designs, a simulation framework has been developed. The framework can be expected to simulate NEM circuit models and inject faults according to the defined fault models during

simulation. The simulation setup can be modified via various parameters. Furthermore, the simulation process, as well as the evaluation process, was automated as much as possible.

Lastly, the outputs of the simulations are analysed. The results are expected to show that the implemented design techniques are effective against the identified faults. The design space is explored concerning fault tolerance and device count. The achievable fault tolerance for the specified fault types is determined, and trade-offs are discussed using different metrics.

Chapter 2

Physical Defects and Fault Models

This chapter elaborates on potential defects in NEM relays and further defines appropriate fault models. In the domain of hardware, a fault refers to the abnormal behaviour of a component or device and can further be classified into three types: transient, intermittent, and permanent faults. Transient faults usually persist temporarily and disappear without any intervention. In CMOS technology, transient faults are generally divided into Single Event Transients (SETs) and Single Event Upsets (SEUs). A SET refers to a sudden spike in a voltage signal within a circuit, which can be caused by the strike of a high-energy particle striking. A SEU, on the other hand, describes a change to an erroneous state in a memory cell. Intermittent faults occur and disappear sporadically. Permanent faults are consistent and require actions such as masking or repair to be removed. A fault can manifest as an error, which describes a deviation in a value or state within a system. If an error is not corrected, it may lead to system failure, meaning that the system is no longer capable of providing its intended service.

The purpose of a fault model is to provide an abstract view of the functional behaviour of a faulty device or component, making it possible to model and analyse a circuit's behaviour in the presence of defects. A fault model ideally covers numerous defects that would be impractical to model individually. Furthermore, a fault model can be defined at various levels of abstraction. Hardware faults are usually caused by physical defects that depend on the technology used. Therefore, a technology-specific investigation of the relevant components and devices is necessary.

2.1 Physical Defects in NEM Relays

Since NEM relays are essentially physical switches, most defects that occur are mechanical in nature. These physical defects can appear either during manufacture, lowering the yield, or during operation, decreasing the circuit's reliability. During manufacture, any terminal of a NEM relay can be expected to short to any other or collapse down to the substrate. This collapse usually cannot be recovered

from. Steady improvements in the fabrication process are expected to eventually lead to more mature manufacturing, narrowing this problem down to a satisfactory degree.

Under normal operation, regular use of the devices will wear down the conductive contact material at the tip of the beam, leading to increased resistance up to a point where sufficient electrical conductivity is no longer ensured. Additionally, oxidation might accelerate the increase of resistance. On the other hand, stiction might prevent the beam from returning to its neutral position, causing the relay to remain permanently closed. It is anticipated that further improvements in the fabrication process will mitigate this problems, making the relays more durable. Coating the contact surface with nanocrystalline graphite (NCG) has been tested as a solution and yielded promising results [22]. However, it must still be assumed that these defects may occur after a certain amount of switching cycles , so fault models need to take them into consideration. Lastly, overdriving the device causes the beam to make contact with the gate, shorting it to both the source and the drain. The beam is flexible enough to avoid breaking, but it cannot recover due to the surface adhesion being too large.

All the described defects can be classified as permanent faults. Transient faults, on the other hand, might occur when a device gets stuck and eventually resolves itself after some time without any intervention. Since only a small number of individual devices have been manufactured so far, there is no statistical data available to cover a device's fault probability over time or fault distribution across the chip area. Still, it is expected that devices will start to fail due to the on-resistance becoming too high after a certain number of switching cycles. At this point, a small number of devices might begin to fail, followed by an increasingly larger portion, meaning that the fault probability over time likely follows an exponential function. However, in this thesis, the focus is set on static permanent faults, as this approach is sufficient to investigate faults regardless of their temporal and spatial distribution. Aside from that, most fault-tolerant design techniques that address permanent faults are also applicable to correcting transient faults.

2.2 Gate Level

First, the faulty behaviour is examined from a gate level perspective. At this level of abstraction, the experimental circuits are modelled in Verilog using a structural description of primitive gates, such as NAND, NOR, AND, OR, etc. In the case of a fault, it is assumed that the functional behaviour of the corresponding gate is affected, potentially manifesting as a stuck input or output node, or as a flipped output value. This means that the affected node is either permanently set to zero, set to one, or its value is flipped. The advantage of this fault model is its low complexity and its ability to cover a variety of defects. The terms used to describe this fault model are:

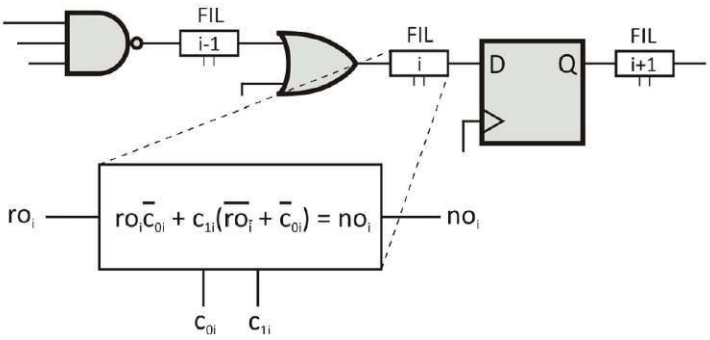


Figure 2.1: Fault Injection Logic [2].

- **stuck-at-0**: The affected node is permanently set to zero.
- **stuck-at-1**: The affected node is permanently set to one.
- **flipping error**: The value of the affected node is flipped.

To assess the circuit's fault tolerance, these faults need to be injected during the simulation according to the fault model. This is done using a Fault Injection Logic (FIL) [2]. The FIL is a Verilog module that is designed to inject faults of the desired type into the circuit. Each gate has a FIL module placed at its output, whereas the index *i* denotes the FIL module associated with the *i*-th gate (Figure 2.1). The input ro_i of the FIL module is connected to the output of the *i*-th gate, while its output no_i is connected to the node where the gate's output was previously connected. To determine the output of the FIL, the Boolean equation

$$no_i = ro_i\overline{c_{0i}} + c_{1i}(\overline{ro_i} + \overline{c_{0i}}) \tag{2.1}$$

is used. The result of this equation is primarily dependent on the control inputs c_{0i} and c_{1i}. Thus, the fault type can be selected by setting c_{0i} and c_{1i} accordingly. For example, a stuck-at-0 fault corresponds to c_{0i} set to 1 and c_{1i} set to 0, whereas a stuck-at-1 fault corresponds to c_{0i} set to 0 and c_{1i} set to 1. When both c_{0i} and c_{1i} are set to 1, the gate output is flipped, while setting c_{0i} and c_{1i} to 0 leaves the gate output unchanged (Table 2.1). The control inputs of the FIL are set in the simulation environment later to inject the different fault types.

Although the defined gate level fault models are widely used in fault simulations, they have their limitations due to their reduced complexity. Defects in NEM relays can impact the behaviour of the gates in various ways. That means that the occurrence of a defect does not always result in a flipped or stuck gate output. To obtain a more accurate understanding of a gate's behaviour under the influence of defects in one or more of its NEM relays, it is necessary to investigate the device level, which lies one abstraction layer deeper.

Table 2.1: Truth table of Equation (2.1).

c_{1i}	c_{0i}	no_i
0	0	roi
0	1	0
1	0	1
1	1	$\overline{ro_i}$

2.3 Device Level

At this abstraction level, the faulty behaviour of the NEM devices is examined to define a fault model. With this approach, a more accurate view on the behaviour of the faulty circuit can be obtained. Based on the discussion of potential physical defects above, it can be concluded that many faults during manufacture or operation lead to terminals being shorted, causing the NEM relay to behave like a closed switch. On the other hand, wear during operation will cause a NEM relay to behave as if it were permanently open. Thus, a simple fault model with the following cases is assumed:

- **stuck-on**: The relay is permanently closed.
- **stuck-off**: The relay is permanently open.

At device level, the NEM relays are modelled as Verilog modules in behavioural description. The fault mechanism, which implements the defined fault model, is integrated into the modules and can be activated via a port. The logic gates are then modelled using these modules of the NEM relays, whereas the experimental circuits are modelled using a structural description of the different logic gates. When designing the logic gates, special care must be taken to ensure the correct computation of the gate's output when one of the aforementioned faults occurs in one of its NEM relays.

To better understand how the function of a gate can be affected by the presence of different faults, a simple NAND gate implemented with 3-T devices will serve as an example (Figure 2.2a). In this gate, the PMOS transistors, which constitute the pull-up network in a traditional CMOS implementation, are replaced by the 3-T devices D1 and D2. The source terminals of D1 and D2 are connected to V_{dd}, while their drain terminals are connected to the output node. Conversely, the NMOS transistors, which constitute the pull-down network in a CMOS implementation, are substituted with the 3-T devices D3 and D4. The drain terminal of D3 is connected to the output node, and its source terminal is connected to the drain terminal of D4, with the source terminal of D4 connected to ground. In a fault free case where the inputs A and B are both set to one, D1 and D2 are open, while D3 and D4 are closed, pulling the output node to ground (Figure 2.2b). The following analysis shows how the gate's output is affected when faults occur in this switching state:

- *A single stuck-on fault occurs:* If a stuck-on fault occurs in D3 or D4, the output of the gate

remains unchanged (but may be incorrect for other inputs). However, if the fault occurs in D2 (or D1), the gate shorts from V_{dd} to ground (Figure 2.2c). This situation can also happen when a stuck-at-0 fault appears at the gate of D2 (or D1). Typically, this leads to a significant increase in power consumption due to continuous current flow, as well as increased heat dissipation, which may damage the devices. The output voltage is then indeterminate and can fluctuate between V_{dd} and ground. If the on-resistance of D1 and D2 is lower than that of D3 and D4, the output may be closer to V_{dd}. On the other hand, if D3 and D4 have a lower on-resistance, the output may be closer to ground. In a Verilog simulation, the value x is assigned to the output node, representing an unknown logic value. Experimental circuits are assessed later using digital behavioural simulations, which means that this x value could be either 1 or 0.

- *A single stuck-off fault occurs:* If a stuck-off fault occurs in D1 or D2, the output of the gate remains unchanged (but may be incorrect for other inputs). However, if the fault occurs in D3 or D4, the output node will be left in a floating state (Figure 2.2d). This can also be caused by a stuck-at-0 fault at the gate of D3 or D4. In a real implementation, this results in the output node being at high impedance, meaning that the node cannot discharge and thus retains the voltage level it had before. The device can hold its voltage level for several hours before discharging. In Verilog, the high-impedance state is denoted by the value z. To accurately depict this behaviour, the logic gates were modelled to retain the logic level of the previous clock cycle when a z value occurs at the output node.

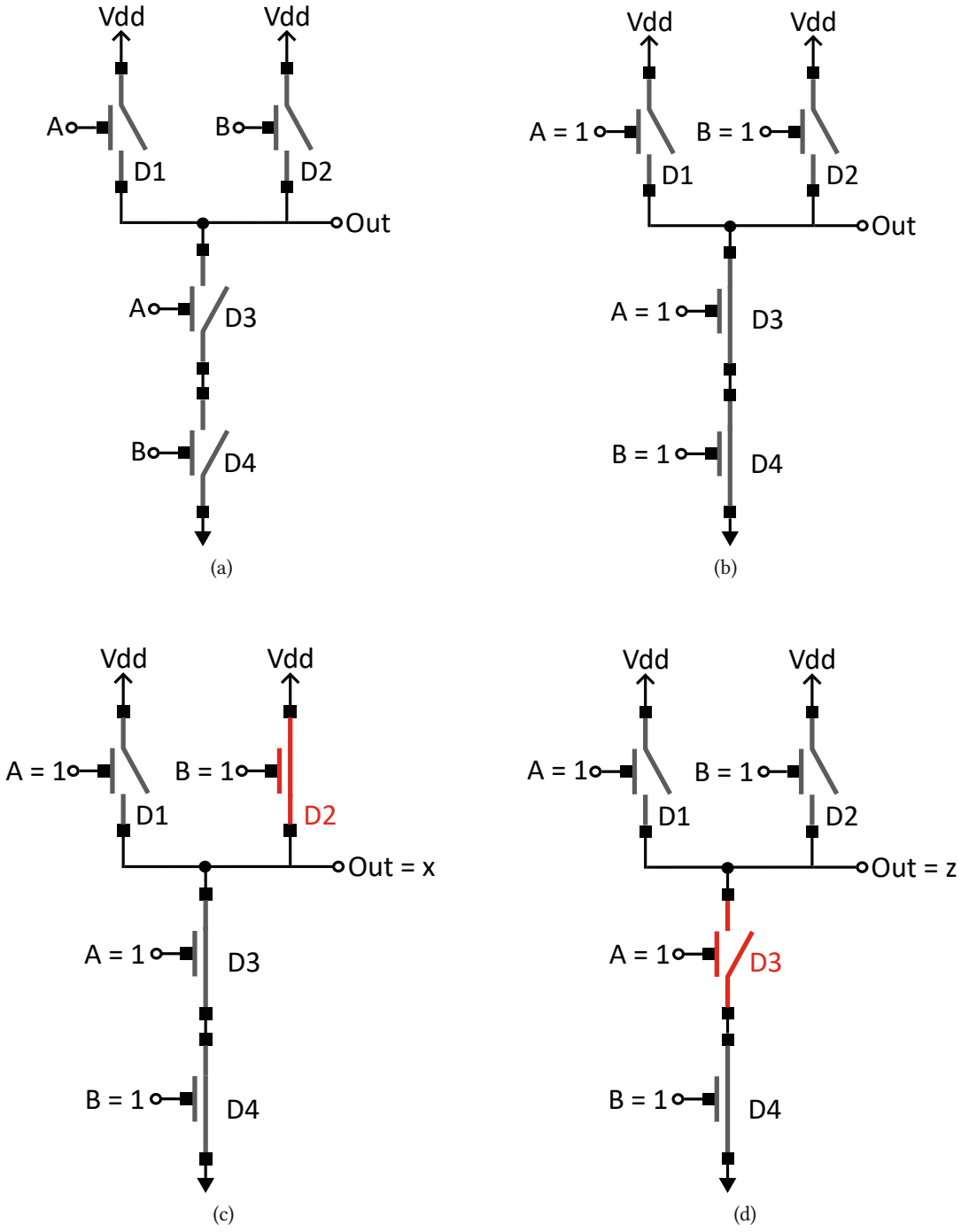


Figure 2.2: NAND gate: (a) implementation with 3-T devices, (b) fault free case (c) single stuck-on fault, (d) single stuck-off fault.

Chapter 3

Fault-tolerant Design Techniques in CMOS

This chapter aims to review various established fault-tolerant design techniques frequently used in CMOS circuits. These techniques usually introduce redundancy in the form of hardware, time, or information to provide additional resources when a fault occurs. The benefits and drawbacks of these design techniques are discussed in detail. Furthermore, their suitability for NEM circuits is assessed to determine which design techniques are appropriate for simulation in the subsequent steps.

3.1 Hardware Redundancy

Fault tolerance in the domain of hardware is one of the simplest and most widely used design techniques to increase the reliability of a circuit. The basic concept is to replicate hardware to provide backup resources when a component or device fails. Hardware redundancy can be implemented at different levels, including the entire system, individual components and modules, and even CMOS transistors or NEM relays at the nanometer scale. Moreover, it can be further divided into active, passive, and hybrid hardware redundancy.

Active hardware redundancy involves fault detection, fault location, and fault recovery to maintain the circuit's correct functionality. Fault detection determines whether a fault has occurred, while fault location identifies the specific location of the fault. In the fault recovery process, the system is reconfigured to remove the faulty component. During this procedure, the fault may propagate, putting the system into an erroneous state until the faulty component is removed. In passive hardware redundancy, fault propagation is prevented by masking the fault, meaning that the concept of fault masking does not require any further interaction to tolerate the fault. In hybrid hardware redundancy, characteris-

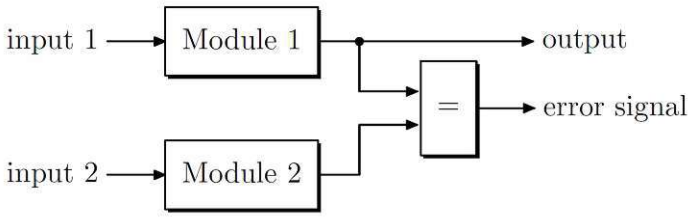


Figure 3.1: Duplication with comparison [3].

tics of both active and passive hardware redundancy are combined. Fault masking is used to prevent faults from manifesting in errors, while fault detection, location, and recovery is used to remove and replace the faulty components. The following sections elaborate on the different forms of hardware redundancy [3, 11].

3.1.1 Active Hardware Redundancy

Active hardware redundancy techniques employ a fault detection mechanism to determine if a fault has occurred. When a fault is detected, the faulty component or device is located. Finally, the required actions are performed by an operator or the system itself to remove or replace the faulty component. Active hardware redundancy techniques do not apply fault masking, meaning the system must be able to tolerate erroneous results for a certain amount of time. Thus, this design technique is not suitable for highly reliable systems that rely on permanently correct outputs. The following examples demonstrate the core principles of active hardware redundancy.

3.1.1.1 Duplication with Comparison

The simplest method of active hardware redundancy is *duplication with comparison*. In this technique, a module is duplicated, so the computation is performed by two identical modules (Figure 3.1). The outputs of these modules are then compared by a comparator. If the outputs do not match, the comparator generates an error signal, indicating that one of the modules has failed. After an error signal has been observed, the system has to be reconfigured, as this technique cannot mask or correct faults [3].

3.1.1.2 Standby Sparing

Another approach in active hardware redundancy is called *standby sparing*. In this design technique, the module is replicated into n identical modules (Figure 3.2). Out of these modules, only one is active, while the others serve as spares. A spare module can be seen as a backup module that is activated when the active module fails. The outputs of the modules are fed into a Fault Detection (FD) logic as well as into a n to 1 switch. The switch is responsible for directing the output of the active module to the

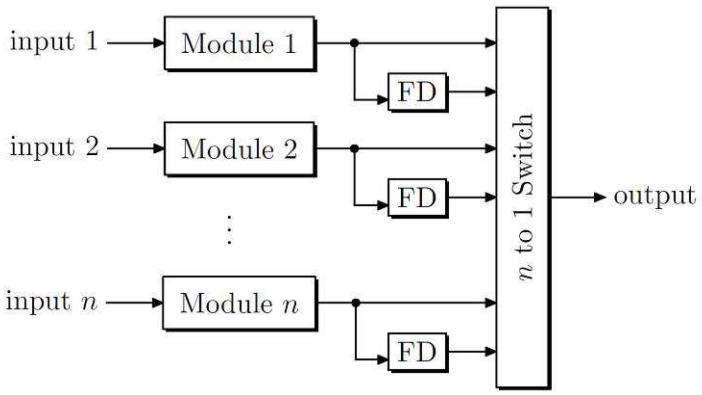


Figure 3.2: Standby sparing [3].

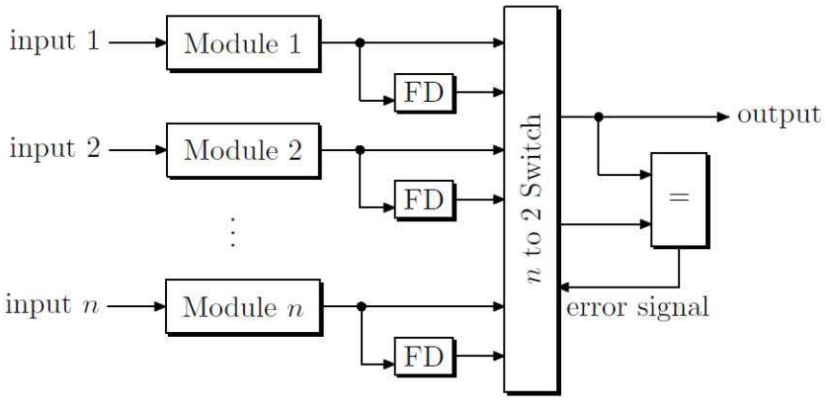


Figure 3.3: Pair-and-a-spare [3].

system's output. When the active module fails, the FD signals the switch to select the output from one of the spare modules. The spares can be held in either hot or cold standby. In hot standby sparing, the spare modules are on and work in parallel with the active module. When the active module fails, the spare modules can take over with minimal reconfiguration time. In cold standby all spare modules are powered off to reduce power consumption. Therefore, they need to be turned on when selected as active module, leading to higher reconfiguration time [3].

3.1.1.3 Pair-and-a-spare

The *pair-and-a-spare* technique is a combination of duplication with comparison and standby sparing (Figure 3.3). However, in this technique two modules are active at the same time. The outputs of all modules are fed into a n to 2 switch that directs the outputs of the two active modules into a comparator, with one of these outputs also being directed to the system's output. If a mismatch in the comparison occurs, the comparator generates an error signal. The switch then uses information from the FD logic to determine which module has failed and selects a spare module to replace the failed one. Thus, it is ensured that there are always two fault-free modules working [3].

3.1.1.4 Applicability to NEM Circuits

Since duplication with comparison is only capable of detecting faults but not correcting or masking them in the affected modules, it is not considered as an appropriate choice to address yield and reliability issues in NEM technology. The technique can be used to detect faulty modules, but there is no mechanism to determine which module has failed. Furthermore, the circuit must be reconfigured when a faulty module has been detected, which is impractical when faults occur during manufacture or while the device is in operation. Additionally, the high fault probability in NEM technology may cause both modules or even the comparator to fail.

In both standby sparing and pair-and-a-spare configurations, the system can reconfigure itself without requiring any intervention. Both systems are capable of tolerating $n - 1$ faults, assuming they are comprised of n modules. Therefore, there must be enough replicated modules to achieve the desired fault tolerance. This can lead to a significant increase in hardware cost. Moreover, the additional hardware required for FD and switching logic has to be considered. It is also important to note that the circuitry implementing the FD and switching logic is susceptible to faults as well, so it is crucial to ensure that this part of the circuit functions correctly, otherwise, the active redundancy system may fail.

In general, active hardware redundancy techniques are applied at an abstraction level where modules and functional units serve as the basic building blocks of a circuit. However, at the current stage of the i-EDGE project, it is not yet possible to reliably manufacture complex modules in NEM technology. Due to this limitation and the reasons mentioned above, the active hardware redundancy approach is not pursued further in this thesis. Nevertheless, active redundancy techniques may become more attractive in a later stage of the project when manufacturing processes have matured sufficiently to fabricate NEM modules with higher reliability.

3.1.2 Passive Hardware Redundancy

Passive hardware redundancy aims to mask faults, ensuring that the affected component continues to function correctly when a fault occurs. Thus, neither fault detection nor intervention by the system or the operator is needed to tolerate faults. Because of its low design complexity and versatility, it is a widely used approach to increase a system's fault tolerance. There are a multitude of passive hardware redundancy techniques, with the most prominent one being *Triple Modular Redundancy (TMR)*. Therefore, the first part of the following section is dedicated to TMR, followed by a discussion of other design techniques.

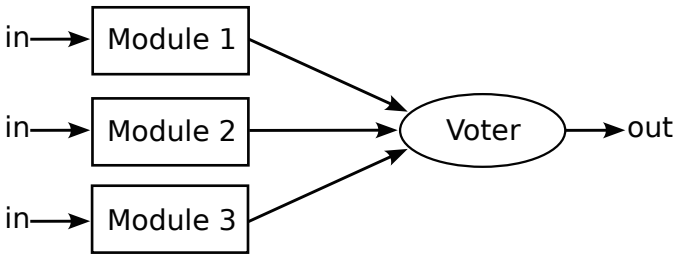


Figure 3.4: TMR system.

3.1.2.1 Triple Modular Redundancy

In TMR, a module is replicated three times to have three identical modules working in parallel. The output of each module is fed into a voting circuit, which performs majority voting on the inputs and provides the result as the output. It is evident that the voting system can tolerate one faulty module, as the voting process will mask the fault. In this structure, the voter constitutes a single point of failure, as it is critical that the voting circuit operates fault-free to provide the correct result. Selective hardening is not possible with NEM technology, so the critical parts of the circuit cannot be made more reliable by modifying the manufacturing process. Several design techniques have been proposed to improve the reliability of the voting process, with *R-fold modular redundancy* and *cascaded TMR* being two of them [27]. In R-fold modular redundancy, the module is replicated not just three times, but R times, allowing correct voting even when more than one module fails. Often, the term *N-Modular Redundancy (NMR)* is used in literature to refer to this technique. In cascaded TMR, the TMR concept is repeated, meaning that the entire TMR structure is replicated, and an additional voter is introduced that performs the voting process for the triplicated structures. However, these design techniques significantly raise hardware costs and do not address the issue of the voter being a single point of failure. Thus, several voter designs have been proposed to improve the reliability of the voter itself. A selection of different designs is presented below.

Standard Voter The standard majority voter can be implemented using four NAND gates (Figure 3.5). From a gate level perspective, this voter may fail if the output of the NAND gate in the second stage becomes stuck or flips. Additionally, the voting process fails if a fault in one of the NAND gates in the first stage causes one or more of the nodes N1, N2, or N3 to become stuck at 0, while the remaining nodes are set to 1 (which is assumed to be the correct value). On the other hand, the NAND gate in the last stage will still produce the correct output if one or more of the nodes N1, N2, or N3 are stuck at 1, while the other nodes are correctly set to 0. At the device level, a stuck-off fault can cause a gate's output to be stuck at either 0 or 1, leading to similar cases as those at the gate level. A stuck-on fault may cause a gate's output to remain at an undefined logic level. When an undefined logic level is fed

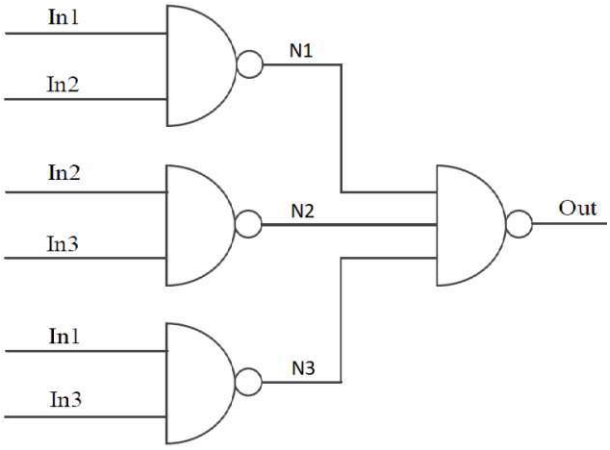


Figure 3.5: Standard TMR voter [4].

into a NAND gate, the output can only be determined when one of its other inputs is 0, as in this case, the output of the gate will always be 1. If an undefined logic level occurs at the output of the voter, it is considered a failure. In the simulations, this voter is labelled as *standard_voter*.

Voter with XORs, Multiplexer & Priority Encoder This voter design features two XOR gates, a priority encoder, and a multiplexer (Figure 3.6). The inputs A and C are compared to input B for inequality using XOR gates. The outputs of the XOR gates are fed into a priority encoder, which is designed to give priority to the input with the logic level 0 by implementing the Boolean equation $O = I_1 \wedge \overline{I_2}$. The multiplexer then selects either A or C as the output, depending on which input is determined to be fault-free based on the comparison. If all inputs are equal, or if both A and C are not equal to B, then A is selected as the output. This voter can handle a single external fault at the inputs, as well as a single internal fault at node S_1, S_2 or O. This means that the voter remains operational when S_1, S_2 or O becomes stuck at 1 or stuck at 0, assuming all inputs are correct. If both internal and external faults occur, the voter may still function in specific cases. For example, when the inputs A and C are equal, and input B is faulty, the voter will still perform correctly, even in the presence of an internal fault. In this case, it will also provide the correct output when an undefined logic level appears at node O due to stuck-on faults [5]. In the simulations, this voter is labelled as *priority_enc_voter*.

Voter with Multiplexer & XOR This voter is designed using an XOR gate and a multiplexer (Figure 3.7). The inputs A and B are compared for inequality by the XOR gate, and its output is used as select line for the multiplexer. This means that if inputs A and B are equal, the multiplexer selects input B; otherwise, input C is selected as the output. Similar to the previous design, this voter can handle a single external, as well as a single internal fault. If both internal and external faults occur, the voter remains functional only in specific cases. When inputs B and C are correct, and input A is faulty, the

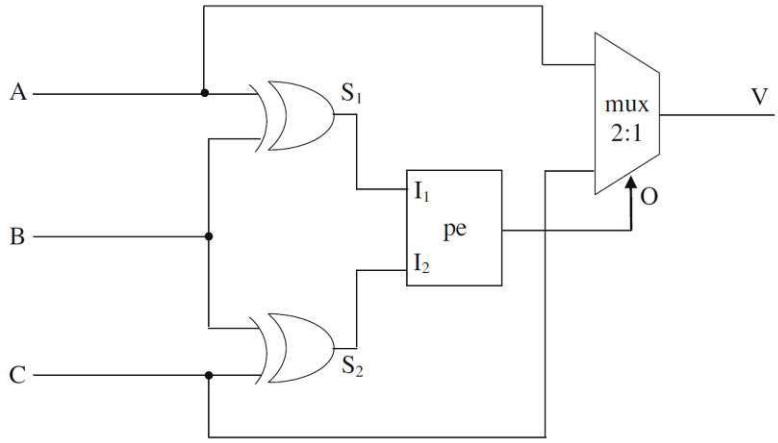


Figure 3.6: Voter with XORs, priority encoder, and a multiplexer [5].

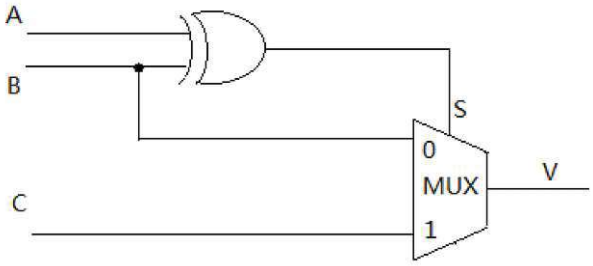


Figure 3.7: Voter with multiplexer and XOR [6].

voter can also handle an internal fault at node S, including undefined logic levels [6]. In the simulations, this voter is labelled as *mux_xor_voter*.

Voter with OR & Complex Gate This voter design consists of an OR gate and a complex gate, which implements the Boolean function $V = (M \wedge Z) \vee (X \wedge Y) \vee (Y \wedge Z)$ with $M = X \vee Y$ (Figure 3.8). In addition to handling a single internal fault or a single external fault, this design is capable of tolerating more scenarios where both occur simultaneously, compared to the previous designs. The downside of this voter is that its behaviour largely relies on the correct functioning of the complex gate, which can be compromised by stuck-on and stuck-off faults [7]. Because the logic function of this voter uses only AND and OR operations, it is labelled as *and_or_voter* in the simulations.

Voter with Triple Transistor Redundancy In this design, gate level redundancy and device level redundancy are combined to improve the fault tolerance of the standard voter. The modification involves replicating each NAND gate in the first level three times. The 3-input NAND gate in the second level is then redesigned using tripled transistor design (Figure 3.9), where each transistor is replaced by a structure consisting of three transistors (Section 3.1.2.2). The outputs of the triplicated gates in the first level provide the inputs to the tripled transistor structures. The redundant transistor structure

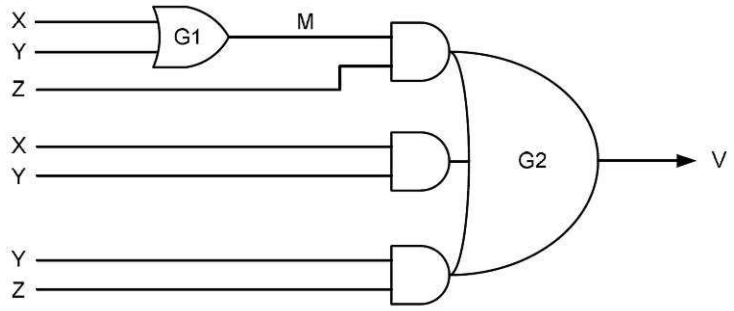


Figure 3.8: Voter with OR gate and complex gate [7].

of the NAND gate in the second level allows to tolerate faulty outputs coming from the gates in the first level . Moreover, it makes the gate more resilient against stuck-on and stuck-off faults [4]. In the simulations, this voter is labelled as *tripled_voter*.

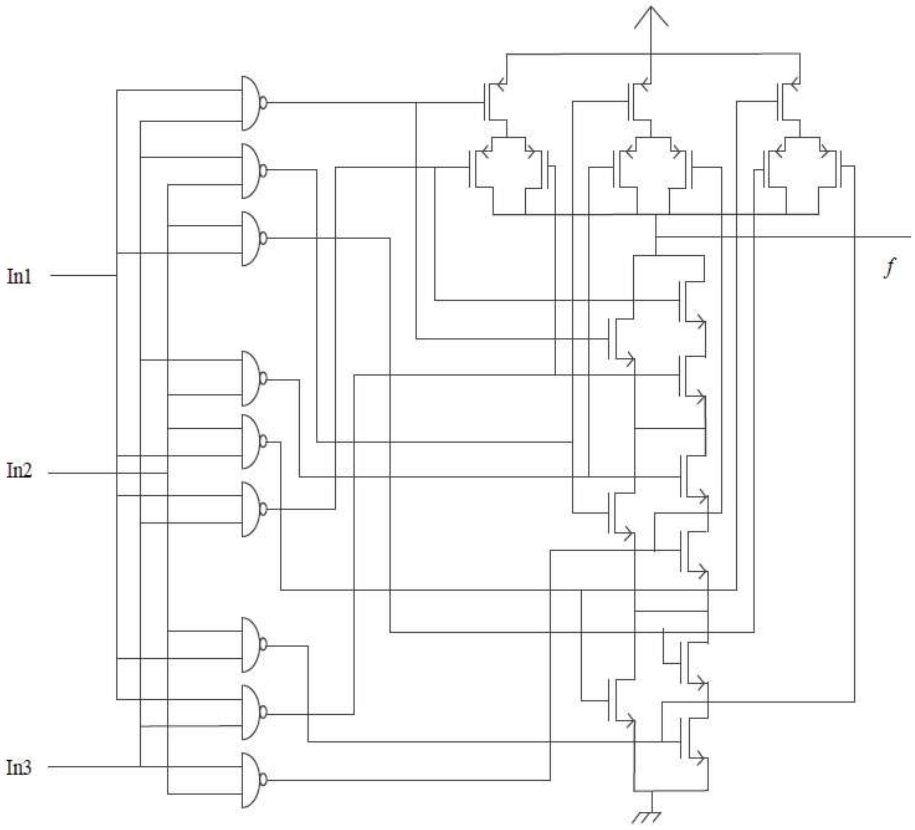


Figure 3.9: Voter with triple transistor redundancy [4].

3.1.2.2 Tripled Design

In this design technique, fault tolerance is achieved by introducing redundant transistors, which serve as backup devices [8, 28]. Each transistor is replaced by a structure of three transistors, which can be arranged in two different configurations. In the first version, a transistor is placed in series with

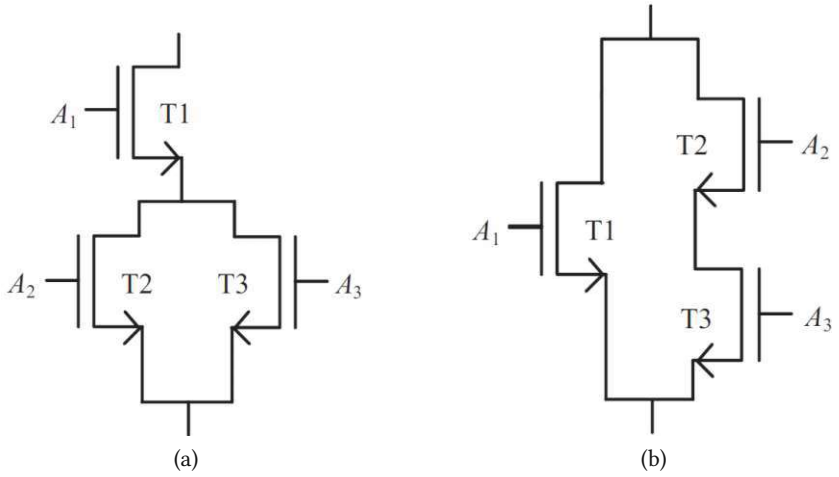


Figure 3.10: Tripled design: (a) version 1, (b) version 2 [8].

two parallel transistors (Figure 3.10a). In the second version, a transistor is placed in parallel with two transistors that are connected in series (Figure 3.10b). In a fault-free case, the gate inputs A_1, A_2, and A_3 are equal, causing the entire structure to behave like a single transistor. In the case of one or more faults, the following scenarios can occur:

- *Version 1:* This configuration can tolerate a single stuck-on fault in T1, T2, or T3 (or a stuck-at-1 fault at either A_1, A_2, or A_3). However, it can only tolerate a stuck-off fault in T2 or T3. If a stuck-off fault occurs in T1 (or A_1 gets stuck-at-0), the path remains open, resulting in incorrect behaviour of the circuit. Dual stuck-on faults can also be tolerated when they occur in T2 and T3, whereas the circuit always fails for dual stuck-off faults.
- *Version 2:* This configuration can tolerate a single stuck-off fault in T1, T2, or T3 (or a stuck-at-0 fault at either A_1, A_2, or A_3). A single stuck-on fault can only be tolerated when it occurs in T2 or T3. A stuck-on fault in T1 (or a stuck-at-1 fault at A_1) will cause a short circuit. Dual stuck-off faults are tolerated only when they occur in T2 and T3, whereas dual stuck-on faults always result in a short circuit.

From this analysis, it can be concluded that the first version is more resilient to stuck-on (or stuck-at-1) faults, whereas the second version is more resilient to stuck-off (or stuck-at-0) faults. Thus, the most suitable version depends on which type of fault is more likely to occur. This design technique can also be applied to NEM circuits by substituting the transistors in the structures with their NEM counterparts. In CMOS, this technique is usually referred to as triple-transistor redundancy, while in this work the term *Tripled Design (TD)* is used. In the simulations, the two design structures are labelled as *TD_v1* and *TD_v2*.

3.1.2.3 Quadded Design

This design technique makes use of the concept of replacing each transistor with a redundant structure to improve fault tolerance, similar to TD [29–32]. In this structure, four transistors are arranged in one of two possible configurations. In the first configuration, two pairs of transistors connected in series are arranged in parallel (Figure 3.11a). In the second configuration, two pairs of parallel transistors are connected in series (Figure 3.11b). In a fault-free case, the gate inputs A_1, A_2, A_3, and A_4 are equal, so the behaviour of the entire structure is the same as that of a single transistor. In the presence of faults, the following scenarios can occur:

- *Version 1:* This configuration can tolerate both single stuck-on and stuck-off faults in T1, T2, T3, or T4 (or stuck-at-1 and stuck-at-0 faults at either A_1, A_2, A_3, or A_4). Dual stuck-off faults are tolerated as long as they don't occur in T1 & T3, T1 & T4, T3 & T2, or T2 & T4. Dual stuck-on faults are tolerated as long as they don't occur in T1 & T2 or T3 & T4.
- *Version 2:* This configuration can tolerate both single stuck-on and stuck-off faults in T1, T2, T3, or T4 (or stuck-at-1 and stuck-at-0 faults at either A_1, A_2, A_3, or A_4). Dual stuck-off faults are tolerated as long as they don't occur in T1 & T3 or T2 & T4. Dual stuck-on faults are tolerated as long as they don't occur in T1 & T4, T1 & T2, T3 & T2, or T3 & T4.

This analysis shows that the first version is more resilient against stuck-on faults (or stuck-at-1 faults), whereas the second version is more resilient against stuck-off faults (or stuck-at-0 faults). The term *Quadded Design (QD)* is used to refer to this design technique, and similar to TD, the transistors are replaced with their NEM counterparts. Furthermore, the concept of redundant transistor structures can also be generalized by introducing N^2 transistor structures, where $N \geq 2$ [33]. Each N^2 structure consists of N serially connected blocks, where one block is comprised of N parallel transistors. It is evident that the number of tolerated faults can be increased by the use of redundant structures with a larger N. However, since QD already provides a significant improvement in fault tolerance, and N^2 structures with $N > 2$ come at the expense of a high hardware overhead, only QD was utilized. In the simulations, the two versions of QD are labelled as *QD_v1* and *QD_v2*.

3.1.2.4 Quadded Logic + Quadded Design

This design technique combines both Quadded Logic (QL) and QD to leverage the strengths of both approaches. The circuit is implemented in QL, with only the last level being realized in QD (Figure 3.12). In QL, each gate in the circuit is replicated four times. Additionally, the replicated gates have twice as many inputs as the original gates. The four outputs of the replicated gates are grouped into two pairs,

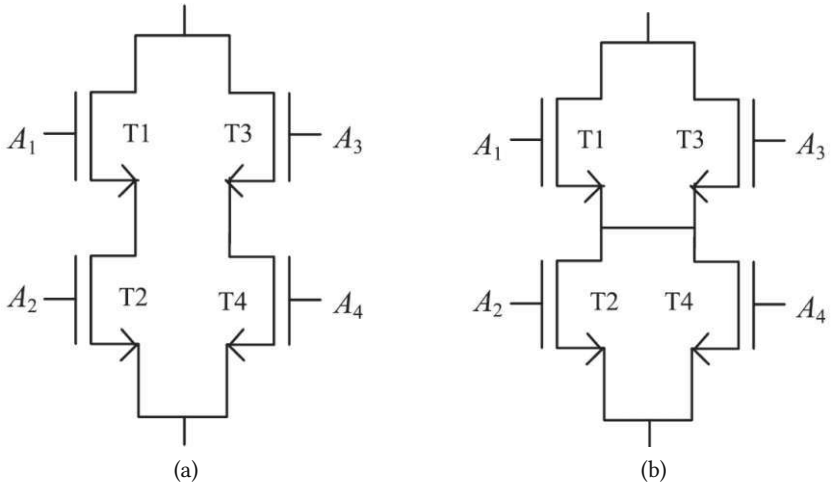


Figure 3.11: Quadded design: (a) version 1, (b) version 2 [8].

with each pair providing inputs to two gates at the next level. The outputs can be paired in three possible ways: (1,2) and (3,4); (1,3) and (2,4); and (1,4) and (2,3). These patterns determine which outputs provide the inputs to each gate at the next level. If the outputs are grouped into the pairs (1,2) and (3,4), the output pair (1,2) will be connected to gate 1 and 2 at the next level, while the output pair (3,4) will be connected to gate 3 and 4 at the next level. It is important that the interconnect pattern at the inputs of a group of gates at one level must be different from the interconnect pattern at the outputs of that group. To understand how faults are corrected in this design, the terms critical error and sub-critical error must be defined. A critical error at one of the gate's inputs always results in an incorrect output, regardless of the other inputs. An example of a critical error is an input of a NAND gate wrongly set to 0, which leads to the output always being 1. A sub-critical error at one of the gate's inputs does not always result in an incorrect output, as it depends on the other inputs. For example, if an input of a NAND gate is wrongly set to 1, the gate's output still depends on the the other inputs. A single sub-critical error is absorbed after passing one stage of gates, while a critical error is corrected after passing two stages. However, critical errors occurring in the last two stages, as well as sub-critical errors occurring in the last stage cannot be corrected. Combining the QL approach with QD addresses this issue by implementing the last stage in QD, which enables it to tolerate both faults coming from the previous stage, as well as faults occurring at the last stage [9, 34]. In the simulations, this design technique is labelled as *QL+QD*.

3.1.2.5 Triple Modular Redundancy + Tripled Design

This approach combines TMR at the gate level with TD at the device level. Similar to TMR, redundancy is achieved by replicating the gates three times. However, in this design, the gates at the last level of the circuit are not replicated but instead implemented in TD (Figure 3.13). The outputs of the triplicated

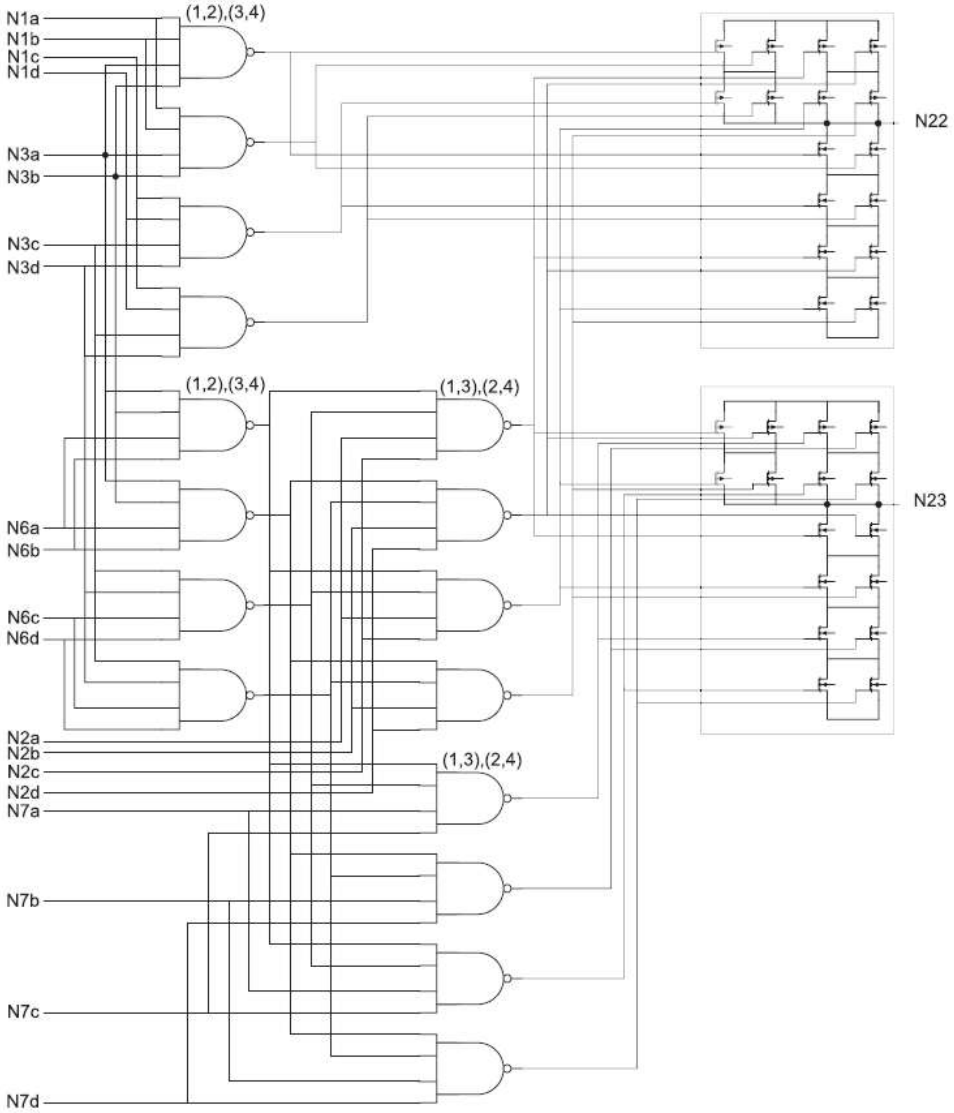


Figure 3.12: C17 circuit (Section 6.1.1) implemented in QL + QD [9].

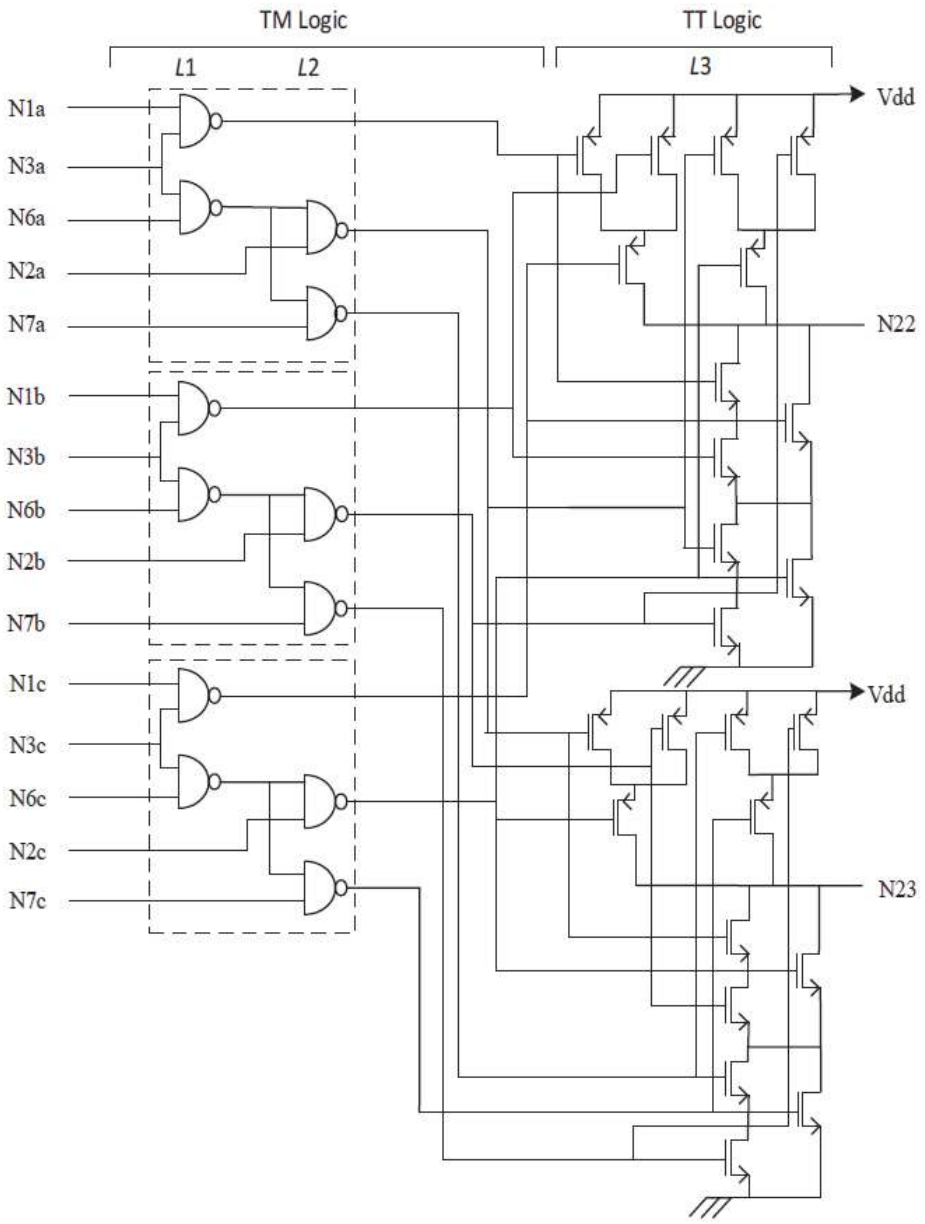


Figure 3.13: C17 circuit (Section 6.1.1) implemented in TMR + TD [10].

gates serve as inputs to the TD structure at the last level. The application of TD to the gates at the last level not only improves the fault tolerance of these gates but also implements a voting mechanism for the triplicated gates in the preceding stages, as TD structures can also tolerate faulty inputs. Therefore, voting circuits, which are susceptible to faults, are no longer required [10]. In the simulations, this design technique is labelled as *TMR+TD*.

3.1.2.6 Applicability to NEM Circuits

Passive hardware redundancy offers a wide range of design techniques for enhancing the fault tolerance of circuits. These techniques are capable of effectively masking faults that originate from defective

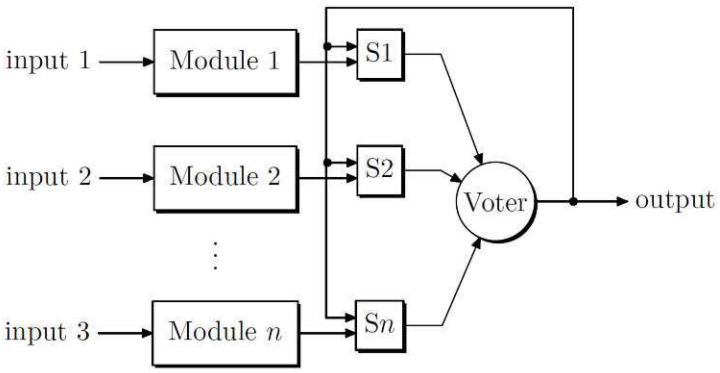


Figure 3.14: Self-purging redundancy [3].

NEM devices, whether they occur during manufacture or operation. This low level approach enables fault-tolerant circuits without requiring sophisticated additional circuitry or external intervention by an operator. Additionally, their versatility allows for various design choices, such as choosing the module size in TMR, selecting a specific design variation in TD or QD depending on the fault type, or determining a general trade-off between device overhead and device fault probabilities. Lastly, passive hardware redundancy techniques can not only easily be adapted to NEM circuits by replacing the conventional CMOS transistors with their NEM counterparts, but also enable designs that are only possible in NEM technology, such as NEM voting circuits in TMR (Chapter 4). Therefore, passive hardware redundancy is considered a viable option for increasing the yield and the reliability of NEM circuits.

3.1.3 Hybrid Hardware Redundancy

Hybrid hardware redundancy is the most complex form of hardware redundancy, as it combines elements of both passive and active hardware redundancy. Passive hardware redundancy provides the ability to mask faults, preventing them from propagating through the system, while active hardware redundancy employs fault detection, fault location, and fault recovery to reconfigure the circuit. The following sections provide two examples to elaborate on the concept.

3.1.3.1 Self-Purging Redundancy

The concept of *self-purging redundancy* is based on NMR, meaning that a module is replicated n times, and a voter determines the correct output through majority voting. However, the output of a replicated module is not fed directly into the voter, but into a switch that compares the output of the voter to the output of the connected module (Figure 3.14). If the comparison fails, the switch opens, removing the connected module from the voting process. The voter is designed to be capable of adapting the number of its inputs. A system with n replicated modules can tolerate $n - 2$ faulty modules [3].

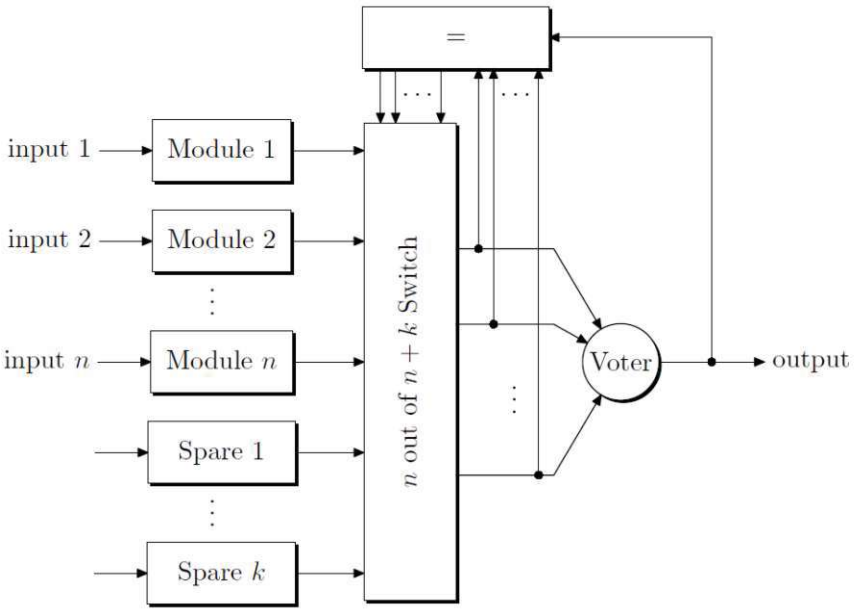


Figure 3.15: N-modular redundancy with spares [3].

3.1.3.2 N-Modular Redundancy with Spares

N-modular redundancy with spares is similar to self-purging redundancy in the sense that it utilizes NMR as its base technique. However, in addition to the n replicated modules, k spare modules are introduced (Figure 3.15). The outputs of the $n + k$ modules are connected to a switch. A comparator compares the output of the n active modules to the output of the voter and feeds the results of the comparisons into the switch. If a module's output fails the comparison, the module is considered faulty, and the switch replaces it with a spare module. When there are no spare modules left, the system can operate in one of two possible ways. In the first option, the comparator is turned off, and the system continues working like a NMR system. In this case, the system can tolerate $\lfloor n/2 \rfloor + k$ faulty modules. In the second option, the comparator continues working and the system operates like one with self-purging redundancy. In this case, the system can handle $k + n - 2$ faults [3].

3.1.3.3 Other Hybrid Architectures

More sophisticated hybrid architectures combine not only different hardware redundancy techniques but also incorporate redundancy in the domains of information or time. Designs that use both hardware duplication and information redundancy, in the form of parity check codes, to perform fault diagnosis and correction have been proposed [35]. Other architectures aim to modify or extend N-modular redundancy with spares by utilizing information redundancy for error detection, time redundancy for correcting transient errors, and hardware redundancy for correcting permanent errors [36, 37]. To achieve this, finite state machines are implemented and integrated into the design to recompute the

result or change the hardware configuration in the case of a fault.

3.1.3.4 Applicability to NEM Circuits

Hybrid hardware redundancy is the most sophisticated and complex form of redundancy, as it combines different approaches. Thus, it is often applied when there is a need for highly reliable systems. Regarding NEM circuits, a conclusion similar to that of active hardware redundancy techniques can be drawn. Hybrid hardware redundancy is normally applied at system level, using modules of high or moderate complexity as redundant units. This means that this type of redundancy may become a viable choice for NEM systems that operate in harsh environments with strict reliability constraints in the future. However, due to its complexity and hardware costs, it is currently not regarded as a viable approach.

3.2 Time Redundancy

Fault-tolerant designs that utilize time redundancy aim to achieve fault tolerance by using additional time instead of additional hardware, minimizing hardware costs. Therefore, this approach is best suited for applications without strict timing constraints. The basic idea is to detect faults by repeating the same computation two or three times and comparing them. If the comparison fails, it is assumed that a fault has occurred.

In its simplest form, time redundancy can be used to detect transient faults, by recomputing the results with unmodified operands. However, it is obvious that permanent faults cannot be detected by repeating the computation with the same operands, as the results will remain unchanged. Thus, time redundancy can also be utilized to determine whether a fault is transient or permanent. To do this, it is necessary to first employ another detection technique to detect the presence of errors (such as error-detecting codes etc.). Once an error is identified, the system can then employ recomputation and compare the results to determine whether the fault that caused the error is permanent or transient. If the results differ from each other and the error has disappeared, the fault is considered transient; if the results are the same and the error persists, the fault is considered permanent [11].

In order to detect permanent faults, it is necessary to modify the operands before performing the additional computations. The basic process of this concept is shown in Figure 3.16. In the first computation, the unmodified data is processed to determine the result, which is then stored in a register. Before the second computation, the data is encoded by an encoding function. After processing the encoded data, the result is decoded and stored in another register. The two results are then compared to determine if a fault has occurred. To encode the data, various encoding schemes with different advantages

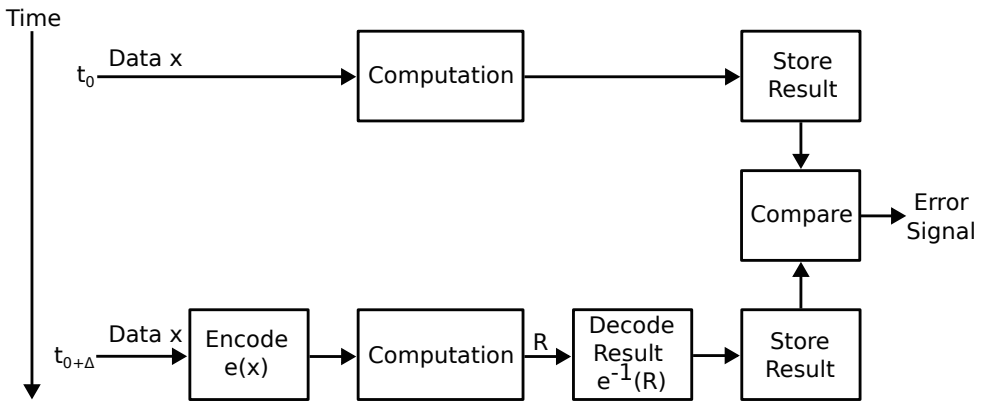


Figure 3.16: Concept to detect permanent faults using time redundancy [11].

and disadvantages exist. In the following, some of them are briefly discussed [11].

3.2.1 Alternating Logic

The *alternating logic* approach can be applied to both digital communication systems and digital circuits. Its principle is to first compute the result for the unmodified data and then perform the same computation for the inverted data. The results of both computations are then compared. If the result of the second computation is not the complement of the first computation, then a fault has occurred. To make this concept applicable to digital circuits, the circuit's function must meet the requirement of being self-dual. A function is regarded as self-dual when the output for the input vector x is the complemented output for the input vector $\overline{x}$. Therefore, a circuit's function is self-dual if $f(x_1, x_2, .., x_n) = \overline{f}(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$. If the function $f(x_1, x_2, .., x_n)$ implemented by the circuit is not self-dual, it can be transformed into a self-dual function by using $f_{sd} = x_{n+1}f + \overline{x}_{n+1}f_d$ with $f_d = \overline{f}(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$. The variable x_{n+1} is a control variable that determines whether f or f_d is used as the output, ensuring that f_{sd} produces complemented outputs for complemented inputs [3].

3.2.2 Recomputing with Shifted Operands

The method *recomputing with shifted operands* was initially designed to detect errors in arithmetic logic units (ALUs), which are assumed to be organized in bit slices. The left shift operation is used as the encoding function, while the right shift operation is used as the decoding function. In the first computation with unmodified operands, bit slice i, which is assumed to be faulty, causes the output to be incorrect at bit i. In the second computation, the operands are shifted, so the bit slice i now performs its computation on the $(i-1)$th bit. The result is then shifted back and compared to the previous one. A fault has occurred when the results differ in either the ith bit, the $i-1$th bit, or both [3].

3.2.3 Recomputing with Swapped Operands

Recomputing with swapped operands is similar to the previous technique; however, in this method, the operands are divided into upper and lower halves. After computing the result for the unmodified operands, both halves of the operands are swapped, and the computation is repeated. By comparing both results, it is possible to determine which bit slice is faulty [3].

3.2.4 Applicability to NEM Circuits

Redundancy in the domain of time is typically employed in systems that are not time-critical. For this reason, time redundancy may be impractical for NEM circuits that operate in edge devices, which often have timing constraints. This is particularly critical given that circuits in NEM technology are slower than circuits in CMOS. Furthermore, this approach is better suited for identifying and eliminating transient faults. Although the detection of permanent faults is also possible, the methods to achieve this are often complicated or tailored to specific types of circuits. Extra circuitry is required to implement the necessary functions, which diminishes the main advantage of hardware-efficiency. Lastly, to not only detect but also correct faults, the computations must be repeated two times or more so that the correct output can be determined through voting, which is inefficient. Thus, time redundancy is not considered a viable option for making NEM circuits fault-tolerant.

3.3 Information Redundancy

To achieve fault detection or correction using information redundancy, additional information is added to the data. This is usually done through the use of codes. Each code has its own set of rules that define how the codewords are formed. A binary codeword consists of the digits 0 and 1 and is mapped to a specific data word. The process of converting a specific data word into a code word according to the rules of the code is called encoding. The decoding process, on the other hand, retrieves the original data from the codeword. This process also enables the detection or correction of errors that have occurred. Thus, codes can be classified into error detecting codes and error correcting codes. Information redundancy encompasses a wide variety of techniques, and the selection of a code largely depends on the application. The following examples have been selected to provide a basic understanding of the concept.

3.3.1 Parity Codes

A simple and well-established error detecting code in the field of information redundancy is the *parity code*. In this code, a codeword is formed by appending a single bit to the data word so that the resulting

codeword has an even (or odd) number of ones. Thus, the appended bit can be used to determine whether the number of ones in the codeword is even or odd. If a bit in the codeword gets flipped due to an error, the number of ones changes from even to odd (or from odd to even), allowing the error to be detected. However, multiple bit errors remain undetected when the number of ones stays even (or odd). Furthermore, it is not possible to locate the position of the faulty bit and, therefore, correct it. This type of code is often used in memory. A parity generator creates the parity bit for a data word before writing the codeword to memory, while a parity checker calculates the parity of the codeword and compares it to the stored parity bit when reading from memory [11].

3.3.2 Linear Codes

A large number of error-correcting codes can be classified as *linear codes*. The methods of linear algebra can be applied to these types of codes, making computations on them more efficient. A data word is usually encoded into a codeword by multiplying it with a generator matrix. The decoding is performed by multiplying a parity check matrix with the codeword. The resulting vector is referred to as the syndrome. The syndrome is then used to determine whether an error has occurred. When the syndrome equals zero, no error has occurred. If the syndrome is non-zero, the position of the error corresponds to the position of the column in the parity check matrix that matches the syndrome. If the syndrome does not match any of the columns in the parity check matrix, more than one bit error has occurred. A prominent example of linear code is the Hamming Code, which can correct single errors and detect up to two errors [3].

3.3.3 Cyclic Codes

The *cyclic code* is an error correcting code that belongs to the class of linear codes. A cyclic code has the property that a circular shift of a codeword results in another valid codeword. Data words and codewords are typically represented as polynomials. Therefore, encoding and decoding algorithms are based on polynomial arithmetic operations, such as polynomial multiplication and division. A codeword is obtained by multiplying the data word's polynomial with the generator polynomial. In the decoding process, the polynomial representing the codeword is divided by the generator polynomial. The remainder from this division is called the syndrome polynomial. If the syndrome polynomial is not equal to zero, an error has occurred. Well known examples of cyclic codes that are frequently used in storage devices and communication systems include Cyclic Redundancy Check (CRC) codes and the Reed-Solomon code [3].

3.3.4 Arithmetic Codes

Circuits that perform arithmetic operations can be checked using *arithmetic codes*. The operands are encoded before any computations are performed on them. The result is then decoded to determine whether an error has occurred. Various techniques can be used for encoding. In AN codes, encoding is done by multiplying the data word N by a constant A. Conversely, decoding is accomplished by dividing the encoded result by the constant A. When the remainder is zero, no error has occurred. In residue codes, a residue is calculated by dividing the data by an integer and appending the remainder to the data word. The decoding is done by simply removing the residue [3].

3.3.5 Applicability to NEM Circuits

The brief overview of information redundancy methods shows that this type of redundancy is applied when entire functional units or systems need to be protected against errors, either by detecting or correcting them. This is achieved by encoding and decoding the data before manipulating it. The choice of the code largely depends on the application at hand. Often, information redundancy techniques are implemented in communication systems and storage devices. The fact that these methods are tailored to complex functional blocks or systems with specific functionalities renders this approach insignificant for improving the yield and reliability of smaller NEM circuits or even single NEM devices. Moreover, many codes only offer error detection without providing correction, which makes additional steps necessary to tolerate the error. Lastly, implementing encoding and decoding circuitry increases hardware overhead. Therefore, information redundancy is disregarded as a viable means of addressing the yield and reliability problems at the current stage of the project.

3.4 Summary

To conclude this chapter, an overview of all the conventional design techniques discussed is presented in Table 3.1. As elaborated above, only the design techniques of passive hardware redundancy were deemed appropriate and, therefore, implemented. An overview of all implemented design techniques is provided in Table 3.2. This table also includes the proposed designs in NEM technology, which are presented in the next chapter.

Table 3.1: Overview of the conventional fault-tolerant design techniques discussed.

Hardware Redundancy				**Time Redundancy**	**Information Redundancy**
Active Hardware Redundancy	**Passive Hardware Redundancy**		**Hybrid Hardware Redundancy**		
Duplication w. Comparison	TMR	Standard Voter	Self-Purging-Redundancy	Alternating Logic	Parity Codes
Standby Sparing		Voter w. XORs, MUX & Priority Encoder	N-Modular Redundancy w. Spares	Recomp. w. Shifted Operands	Linear Codes
Pair-and-a-Spare		Voter w. MUX & XOR		Recomp. w. Swapped Operands	Cyclic Codes
		Voter w. OR & Complex Gate			Arithmetic Codes
		Voter w. Triple Transistor Redundancy			
		Tripled Design			
		Quadded Design			
		Quadded Logic + Quadded Design			
		TMR + Tripled Design			

Table 3.2: Overview of the implemented design techniques.

Design Technique		**Abbreviation**	**Section**
TMR	Standard Voter	standard_voter	3.1.2.1
	Voter w. XORs, MUX & Priority Encoder	priority_enc_voter	3.1.2.1
	Voter w. MUX & XOR	mux_xor_voter	3.1.2.1
	Voter w. OR & Complex Gate	and_or_voter	3.1.2.1
	Voter w. Triple Transistor Redundancy	tripled_voter	3.1.2.1
	NEM Voter Version 1	nem_voter_v1	4.2.1
	NEM Voter Version 1 in Tripled Design v1	nem_voter_TD_v1	4.2.2
	NEM Voter Version 1 in Tripled Design v2	nem_voter_TD_v2	4.2.2
	NEM Voter Version 2	nem_voter_v2	4.2.3
	Tripled Design v1/v2	TD_v1/TD_v2	3.1.2.2
	Quadded Design v1/v2	QD_v1/QD_v2	3.1.2.3
	Quadded Logic + Quadded Design	QL+QD	3.1.2.4
	TMR + Tripled Design	TMR+TD	3.1.2.5

Chapter 4

Fault-tolerant Design Techniques in NEM Technology

4.1 Conventional TMR Voters with NEM Relays

The NEM devices are designed to replace CMOS transistors, allowing the voter circuits presented in Section 3.1.2.1 to be implemented using conventional gates consisting of 3-T devices. However, by utilizing the special functionality of the 4-T device, various gates can be implemented more efficiently, requiring fewer devices. A simple 2-to-1 multiplexer can serve as an example. This device can be implemented using two 4-T devices (Figure 4.1a), while the conventional design consists of three NAND gates and an inverter, requiring fourteen 3-T devices in total. Furthermore, the multiplexer can be used to design other gates with less devices, such as AND, OR, and XOR (Figures 4.1b to 4.1d). Therefore, the conventional voters were implemented using gates with 4-T devices whenever possible.

4.2 Proposed NEM Voters

The special functionality of the NEM devices allows not only for the design of gates with fewer devices but also for the creation of new voter designs. In the following, various NEM voter circuits that have been designed are presented.

4.2.1 NEM Voter Version 1

The first version of the designed NEM voters consists of a 3-T device, a 4-T device, and an inverter comprised of 3-T device (Figure 4.2). This minimalistic design features a total of four NEM devices, making it the voter with the lowest device count among all introduced voters. Its operating principle is based on the fact that either the 4-T device or the 3-T device is closed while the other device remains

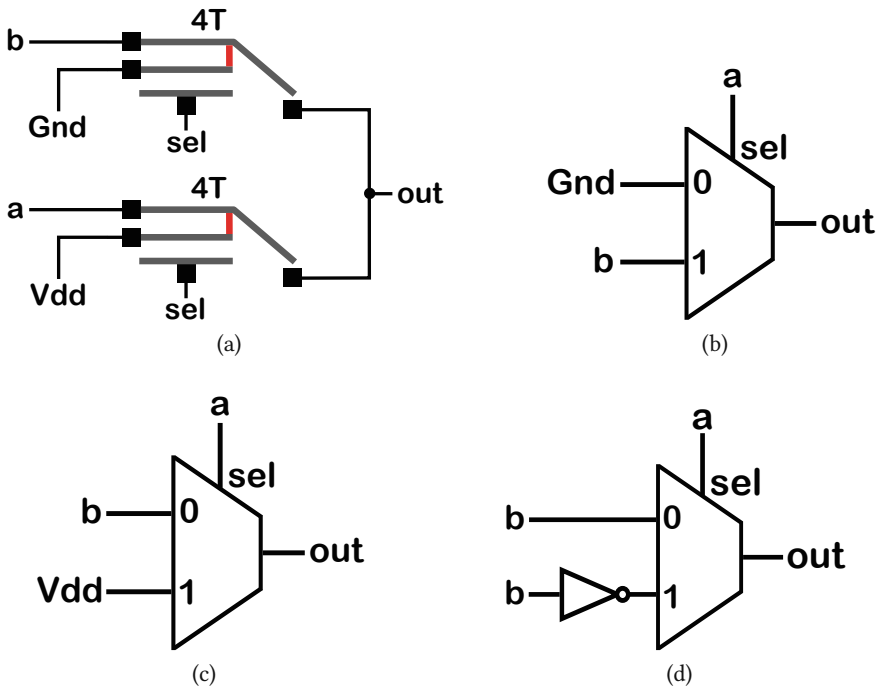


Figure 4.1: (a) 2-to-1 multiplexer, (b) AND gate with multiplexer (c) OR gate with multiplexer, (d) XOR gate with multiplexer.

open. If input b and input c are equal, the 3-T device is closed, and input b is passed to the output, while the 4-T device remains open. If input b and input c are not equal, the 4-T device is closed and input a is passed to the output, while the 3-T device remains open. This voter will fail in the following scenarios:

- *stuck-off faults:* If either the 3-T device or the 4-T device becomes stuck-off while the other device is also open, the output of the voter will be at high impedance, retaining the previous voltage level. This means the voter may provide an incorrect output value. Furthermore, a stuck-off fault in one of the inverter's devices may cause its output to be wrong, potentially leading to incorrect actuation of the 3-T device. This could result in both the 4-T and 3-T devices being either closed or open simultaneously.
- *stuck-on-faults:* If either the 3-T device or the 4-T device becomes stuck-on while the other device is also closed, the circuit may short when input a and input b are at different logic levels. A stuck-on fault in one of the inverter's devices may cause a short circuit in the inverter and its output to be at an undefined logic level. In this case, it cannot be determined whether the 3-T device is actuated or not, resulting in an undefined voter output, which is considered a failure.

The analysis shows that the correct functioning of this voter design is highly dependent on the fault-free behaviour of all its devices. Its low device count comes with the cost of a low intrinsic fault tolerance. In the simulations, this voter is labelled as *NEM_voter_v1*.

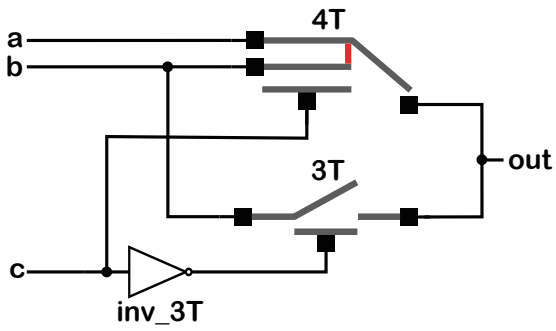


Figure 4.2: NEM voter version 1.

4.2.2 NEM Voter Version 1 in Tripled Design

To improve the fault tolerance of the NEM voter, the design was modified using the tripled design technique. This design technique improves resilience against stuck-off and stuck-on faults by introducing devices that serve as backups when one device fails. In this approach, the 3-T device, the 4-T device, and the devices comprising the inverter are replaced by a tripled structure, consisting of three devices of the same type (Section 3.1.2.2). These tripled device structures exist in two different versions. Both versions have been used to improve the voter design. The first version is more resilient against stuck-on faults (Figure 4.3), while the second version is more resilient against stuck-off faults (Figure 4.4). The application of this design technique increases the total device count of the voter to twelve, which is acceptable. However, this voter still has the limitation of being unable to handle undefined logic levels at the input. If either input b or input c is not known to be 0 or 1, it cannot be determined whether the devices in the tripled structure will switch, leaving the output of the voter undefined. In the simulations, the voter in tripled design version 1 is labelled as *NEM_voter_TD_v1*, while the voter in tripled design version 2 is labelled as *NEM_voter_TD_v2*.

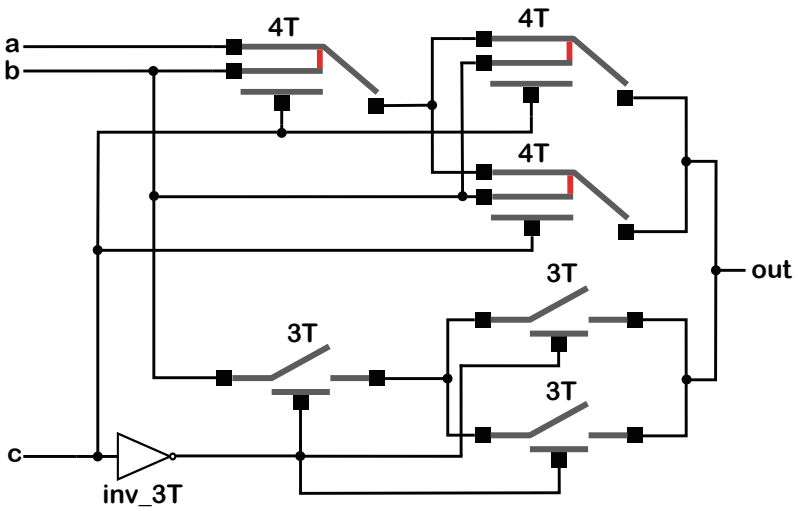


Figure 4.3: NEM voter in tripled design v1.

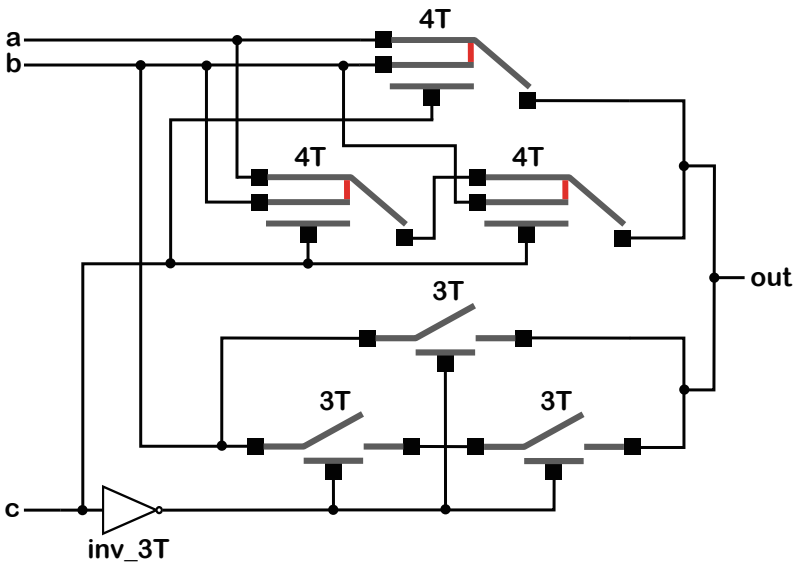


Figure 4.4: NEM voter in tripled design v2.

4.2.3 NEM Voter Version 2

To overcome the limitations of the NEM voter in tripled design, another version has been designed. This voter design is comprised of three XOR gates and three 4-T devices, which makes fifteen NEM devices in total. The XOR gates have been implemented using NEM devices, as shown in Figure 4.1. This voter operates on the principle that the 4-T devices at the output of the XOR gates conduct only when the inputs of the XOR gates are equal. When one of these 4-T devices is actuated, it passes one of the XOR gate's inputs to the output. Due to this behaviour, the voter is not only able to tolerate one faulty input but also to tolerate one input at an undefined logic level. Assuming that input a and input b are both 1, while input c is at an undefined logic level, the first XOR gate will have an output of 0, causing the 4-T relay at its output to close, thereby passing input a to the output node. The second XOR gate will only actuate the second 4-T relay and pass input b to the output when input c equals input b (which is 1). Similarly, the third XOR gate and the 4-T device will pass input c to the output only when input c is equal to input a (which is 1). Therefore, an unknown input that is at a different logic level than the other two inputs is isolated from the output. In the simulations, this voter is labelled as *NEM_voter_v2*.

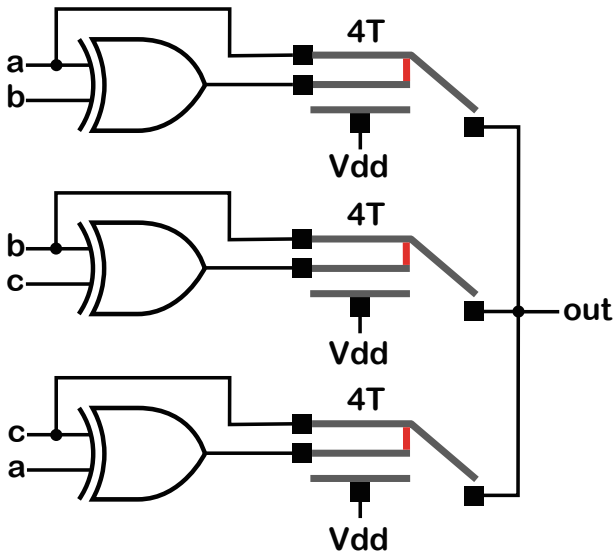


Figure 4.5: NEM voter version 2.

Chapter 5

Simulation Framework

To investigate the benefits and drawbacks of the selected fault-tolerant design techniques, various circuits were modelled, simulated, and evaluated using different metrics. A simulation framework was developed in Verilog and C++ for performing the simulations and evaluations. The structure and the operating principle of the framework is shown in Figure 5.1.

The testbench constitutes the core of the framework and is entirely implemented in Verilog. To simulate a circuit, an instance of the test circuit module is created first. The testbench then drives the test circuit with different inputs, which are read from an input file that contains all the input vectors to be applied. Additionally, faults can be injected during simulation by reading fault files that contain fault vectors. These fault vectors specify which devices or gates in the circuit are faulty. The resulting outputs of the test circuit are finally written to an output file. These output files are then further processed by programs that evaluate the simulation results.

The Verilog testbench was compiled and simulated using the *XCELIUM* package of Cadence, as well as *Icarus Verilog* [38]. The waveforms resulting from the simulations were analysed using *GTKWave* [39]. The input file and the fault files were automatically generated by programs implemented in C++. These programs accept various parameters, allowing the input file and the fault files to be created based on the specific requirements of the test circuit and the simulation. The programs that perform the evaluations were also implemented in C++ and aim to automate the evaluation process as much as possible.

5.1 Simulation Methods

The simulation framework was designed to support digital simulations for static permanent faults. Two different simulation methods have been implemented. Both methods follow different concepts, and different metrics are used to evaluate the results of each method. The following sections describe how these methods are implemented within the framework.

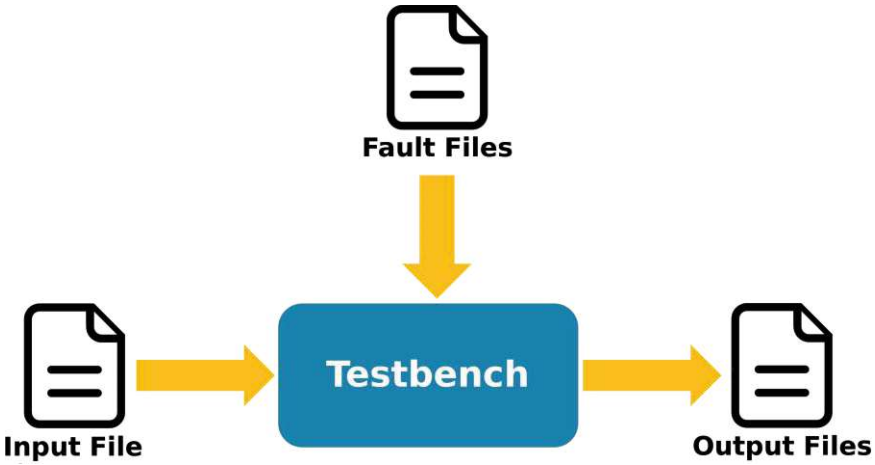


Figure 5.1: Structure of the simulation framework.

5.1.1 Simulation of N Fault Patterns

The first method is referred to as *simulation of N fault patterns*. In this method, the input file contains all possible input vectors for the test circuit. If the test circuit has n inputs, the input file will contain $m = 2^n$ lines, where each line represents a vector with n binary digits. Each of the n digits corresponds to one input of the circuit. The fault files contain N fault vectors, which are used to determine the faulty gates (or devices) in the circuit. A 1 in a fault vector indicates a faulty gate (or device), while a 0 indicates a non-faulty gate (or device). Consequently, a fault file for a circuit with k gates (or devices) will contain vectors consisting of k binary digits, where each digit corresponds to one gate (or device) in the circuit. Each fault file contains only fault vectors with a specified number of faulty gates (or devices), meaning that each fault vector in a fault file contains the same number of 1s. The positions of the 1s in each vector are determined by random permutations. However, the program that generates the fault file ensures that each fault vector is unique. If the number of possible permutations is less than N, the program stops generating new vectors once all possible permutations have been created and written to the fault file. The number of fault vectors N, as well as the number of faulty gates (or devices), can be specified during file creation by passing them as parameters to the program. In the simulations conducted, N was increased until the results no longer changed significantly, which occurred at $N = 50,000$.

When the simulation starts, the simulator reads the fault vectors from the fault file line by line and activates them one after the other by inserting the faults into the gates (or devices) accordingly. For each fault vector, the circuit is simulated by applying the input vectors from the input file sequentially. The output of the circuit is written to the output file for every input vector applied. Once the circuit has been simulated for all input vectors in the input file, the next fault vector is activated, and the process is repeated.

- *Example:* Given a test circuit that has $n = 3$ inputs and $k = 180$ devices. The circuit shall be simulated for 2 and 3 faulty devices, using $N = 10,000$ fault vectors for each case. In this scenario, the input file will contain $m = 2^3$ lines of input vectors, with each input vector consisting of $n = 3$ binary digits. There will be two fault files: one contains fault vectors with 2 faults, the other contains fault vectors with 3 faults. Each fault file will contain $N = 10,000$ fault vectors, with each fault vector consisting of $k = 180$ binary digits. The number of times the circuit is simulated can be calculated by multiplying the number of input vectors by the number of fault vectors and the number of fault files: $n_{total\_sim} = 2^3 \cdot 10,000 \cdot 2 = 160,000$.

The benefit of this simulation method is that it allows for a detailed analysis of the test circuit in the presence of faults. The behaviour of the circuit when specific gates or devices fail can easily be investigated using the corresponding fault vectors. Furthermore, it can be analysed which input vectors cause the circuit to fail and whether there are cases in which the circuit still works for all inputs. The drawback of this method is that the simulation time can become very high for large or even moderately sized circuits. Moreover, the number of possible permutations for a certain number of faults can become extremely high for larger circuits, potentially making the simulation results inaccurate when the number of fault vectors N is chosen too low. To address these issues, another simulation method was explored.

5.1.2 Simulation of Stochastic Computation Models

This simulation method utilizes *Stochastic Computation Models (SCMs)* to compute the reliability of a circuit. Since computations on SCMs are based on signal probabilities, this method supports simulations in which gates and devices exhibit probabilistic fault characteristics. The mathematical foundations of SCMs are based on Probabilistic Gate Models (PGMs) and stochastic computation.

A PGM represents the behaviour of a logic gate using signal probabilities. The probability of an input or output signal of a gate is the probability that the signal is at logic level 1. To determine how the input probability of a gate transforms into its output probability, the logical function of the gate must be considered. Assuming the inputs are independent, the Boolean operation of a logic gate can be mapped to an arithmetic operation of signal probabilities, as shown in Table 5.1, where $a = P(A = 1)$, $b = P(B = 1)$ and $c = P(C = 1)$. The output probability of an unreliable gate with an output named Z (where Z does not refer to high impedance) can generally be represented by the equation

$$\begin{aligned} Z =& P(\text{output "1"} \mid \text{gate faulty}) \cdot P(\text{gate faulty}) + \\ & P(\text{output "1"} \mid \text{gate not faulty}) \cdot P(\text{gate not faulty}). \end{aligned} \tag{5.1}$$

Table 5.1: Mapping of Boolean operations to arithmetic operations of signal probabilities [1].

Boolean Operation		Arithmetic Operation
NOT	$B = \overline{A}$	$b = 1 - a$
AND	$C = AB$	$c = a \cdot b$
OR	$C = A + B$	$c = a + b - a \cdot b$
XOR	$C = A\overline{B} + \overline{A}B$	$c = a \cdot (1 - b) + (1 - a) \cdot b$

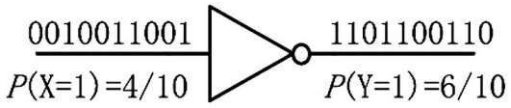


Figure 5.2: Inverter with stochastic encoding [1].

The idea of stochastic computing is to encode signal probabilities into binary bit streams serially in the time domain. This means that a specific probability is indicated by the mean number of 1s in a bit stream. An example of an inverter is shown in Figure 5.2. Here, the probability $P(X = 1) = 4/10$ is encoded into the input bit stream of the inverter. Since Boolean operations can be mapped to arithmetic operations, the inverter performs a probabilistic computation according to Table 5.1. The result of this computation is then encoded into the output bit stream of the inverter. The same concept of mapping Boolean operations to arithmetic operations while using stochastic logic can be applied to other gates (Figures 5.3a and 5.3b) [1, 40].

An SCM can now be designed by combining the concepts of PGMs and stochastic computing. Given that the output probability of any logic gate can be calculated using Equation (5.1), a gate affected by a fault that flips its output can be implemented using the stochastic logic of an XOR gate (see Table 5.1):

$$Z_{flip} = XOR_{sto}(p, \epsilon) = p(1 - \epsilon) + (1 - p)\epsilon, \tag{5.2}$$

where p is the fault-free output probability and ϵ is the error probability. Thus, an SCM for flipping errors can be modelled for any gate by connecting one input of the stochastic XOR to the gate's output, while the other input represents the gate's error rate (Figure 5.3c). Additionally, a stuck-at-1 fault can be implemented by replacing the XOR gate with an OR gate (Figure 5.3d):

$$Z_{SA1} = OR_{sto}(p, \epsilon) = p + \epsilon - p \cdot \epsilon = \epsilon + p \cdot (1 - \epsilon). \tag{5.3}$$

A stuck-at-0 fault can be implemented by using a NOT and an AND gate (Figure 5.3e):

$$Z_{SA0} = AND_{sto}(p, \bar{\epsilon}) = p \cdot (1 - \epsilon). \tag{5.4}$$

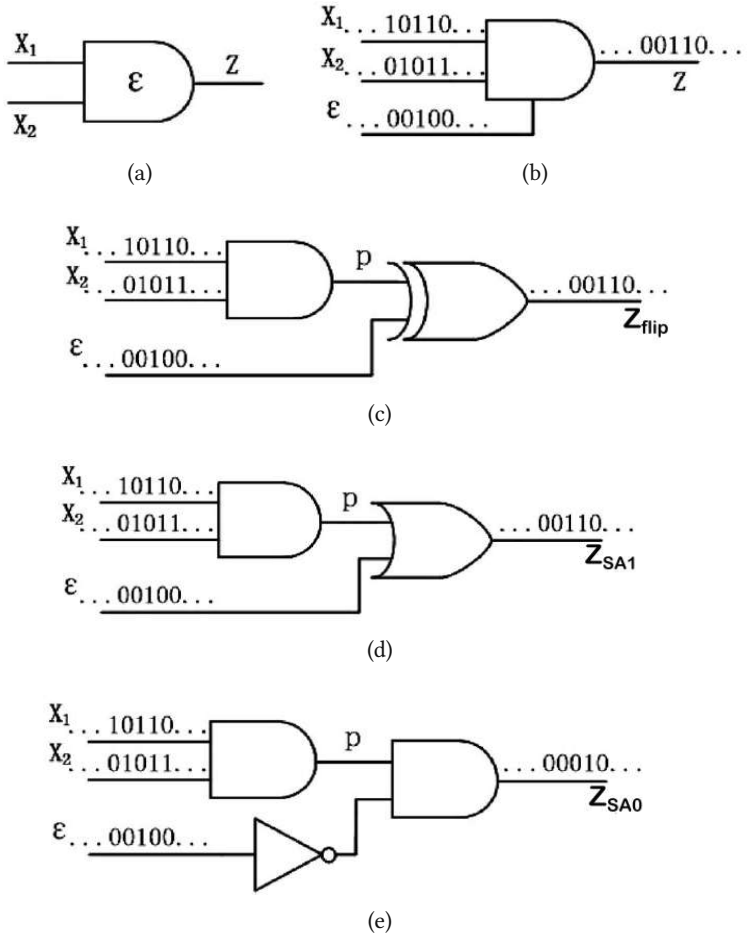


Figure 5.3: AND gate: (a) unreliable gate, (b) stochastic logic implementation, (c) SCM for flipping error, (d) SCM for stuck-at-1 fault, (e) SCM for stuck-at-0 fault [1].

However, to avoid designing a circuit for each fault type separately, the FIL module (Section 2.2) was used to model the SCMs. The FIL module replaces the XOR, OR, NOT and AND gates in Figures 5.3c to 5.3e, as it can emulate the behaviour of these gates by setting the control inputs accordingly [1, 40].

SCMs can also be constructed at the device level. Since the switching state of a transistor depends on the applied gate voltage, flipping errors as well as stuck-on and stuck-off faults, can be modelled by modifying the gate input of the device. In case of a flipping error, a stochastic XOR is used to flip the transistor's gate input when an error occurs, thereby changing the transistor's switching state

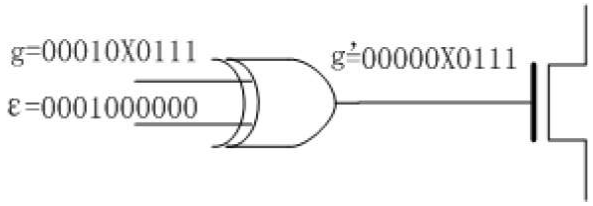


Figure 5.4: Stochastic transistor model for flipping error [12].

(Figure 5.4). In the case of a stuck-on or stuck-off fault, the XOR is replaced by an OR gate or an AND gate combined with a NOT gate, respectively. In the Verilog designs of the NEM relays, the gate inputs were mapped to the operations of the relays. The switching states of the relays were then mapped to the pull-up and pull-down networks of the logic gates, which, in turn, were be mapped to the gate's output [12].

The reliability of a circuit can be assessed by designing and simulating a specific stochastic architecture that utilizes SCMs. An example of this architecture for the C17 circuit (Section 6.1.1) is shown in Figure 5.5. Generally, the following steps are performed to construct the design and evaluate the circuit's reliability:

1. The circuit is constructed using the SCMs of the gates. In Figure 5.5, sub-circuit 1 represents the stochastic circuit for flipping errors, while sub-circuit 2 serves the reliable circuit.
2. The input bit streams and the error bit streams are generated. The input bit streams encode the input probability, while the error bit streams encode the error probabilities of the gates.
3. The bit streams are propagated from the inputs to the outputs in both sub-circuit 1 and sub-circuit 2.
4. The reliability of each output is determined from the output bit streams using XOR gates with one inverted input. The joint reliability can be obtained by feeding the output of the XOR gates into an AND gate. Conversely, the failure rate of the circuit can be obtained by using XOR gates with no inverted inputs and an OR gate instead of and AND gate [1, 40].

When using this method, the input file in the simulation framework contains the input bit streams, while the fault files contain the error bit streams. The bit streams are stored line by line in the files. If the test circuit has n inputs, each line of the input file will contain n binary digits, where each digit corresponds to one input of the circuit. The fault files will contain k binary digits per line for a circuit with k gates (or devices), with each digit corresponding to one gate (or device). When a sequence length of L is used for the bit streams, both the input file and the fault files will contain L lines. Similar to the first simulation method, a 1 in the error bit stream indicates a faulty gate (or device), while a 0 indicates a non-faulty gate (or device). Furthermore, each fault file contains error bit streams that encode a specific gate (or device) error probability. The positions of the 1s in the input bit stream and the error bit streams are determined through random permutations. The input probability, error probability, and sequence length L can be specified during file creation by passing them as parameters to the program. The selected sequence length L determines both the resolution error of the results and the quantization error of the encoded signal probabilities. For a sequence length of L, the resolution is given by $1/L$. For instance, if the sequence length is $L = 10$, the resolution is 0.1, meaning that probabilities lower

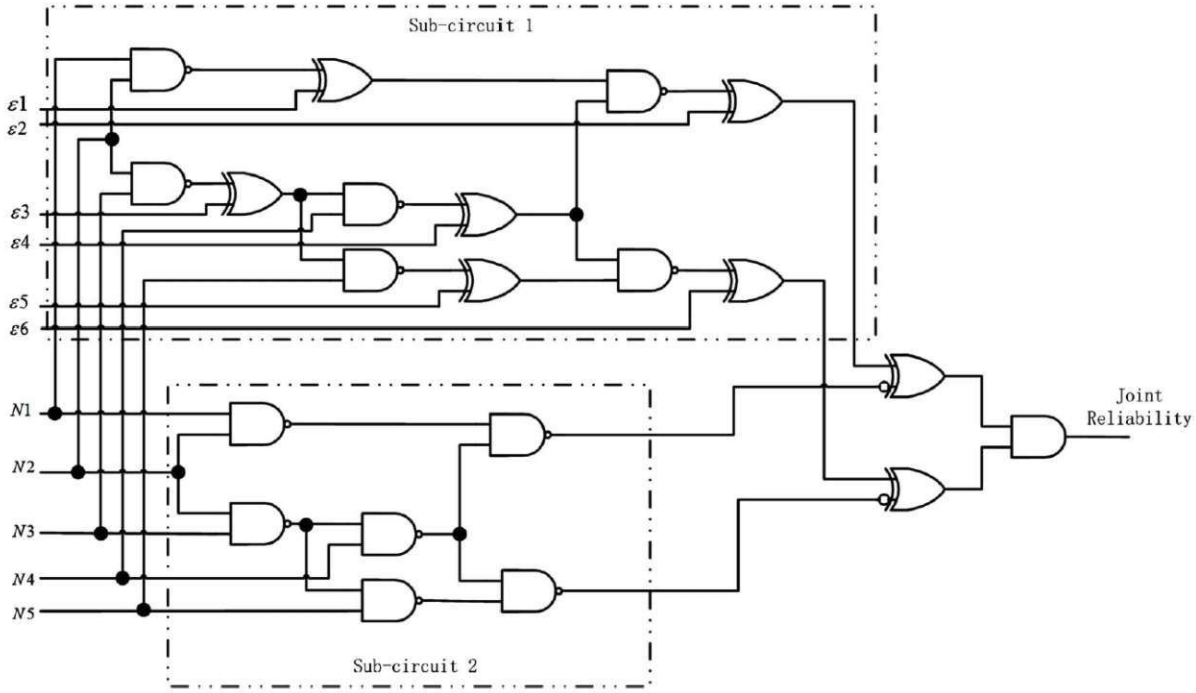


Figure 5.5: Architecture utilizing SCMs to assess the reliability of the C17 circuit [1].

than 0.1 cannot be represented. In the simulations conducted, $L = 10,000$ was selected, resulting in a resolution of 0.0001.

- *Example:* Given a test circuit that has $n = 3$ inputs and $k = 180$ devices. The circuit shall have an input probability of 50% and shall be simulated for device fault probabilities of 0.01% and 1%, using a sequence length of $L = 10,000$. In this scenario, the input file will contain $L = 10,000$ lines of binary patterns with $n = 3$ digits. Since the input probability is 50%, the input bit streams will contain $5,000$ digits of 1. There will be two fault files: one contains error bit streams that encode a signal probability of 0.01%, the other contains error bit streams that encode a signal probability of 1%. This means that the first fault file contains bit streams with a total of one 1, while the second fault file contains bit streams with a total of 100 digits of 1. Each fault file will contain $L = 10,000$ lines of binary patterns, each consisting of $k = 180$ digits. The number of times the circuit is simulated is given by the sequence length, which is $10,000$.

The benefit of this simulation method is that it requires fewer simulation runs to evaluate a circuit's reliability. Additionally, fault probabilities can be assigned to gates and devices rather than assuming a fixed number of faulty components. However, due to the stochastic nature of this approach, specific scenarios with certain input vectors and fault patterns cannot be investigated. Therefore, this approach can only yield the reliability (or failure rate) of a circuit.

5.2 Evaluation Metrics

The output files generated by the simulation are automatically evaluated. For the evaluation, different metrics are used. The choice of the metric depends on the specific simulation method employed.

5.2.1 Circuit Failure Rate

The first metric used for evaluation is the *Circuit Failure Rate (CFRA)* [31]. This metric serves as a general measure of the circuit's robustness against potential faults in gates or devices. It quantifies the probability of a circuit failing when a certain number of gates or devices have failed or when faults occur in them with a specific probability, indicating that a lower CFRA is preferable. In general, the CFRA can be calculated by

$$CFRA = \frac{k}{2^{n+m} - 2^n} \tag{5.5}$$

where k is the number of faulty combinations of gates (or devices) for which the circuit's output is incorrect, n is the number of circuit inputs, and m is the number of gates (or devices) in the circuit. In the denominator of this equation, 2^n is the number of non-faulty combinations, which is subtracted from the total number of faulty and non-faulty combinations, calculated by 2^{n+m}.

When simulations are performed using the first simulation method, an output file is created for each fault file. As described in Section 5.1.1, each fault file contains fault vectors with a specified number of faulty gates (or devices). The number of fault vectors per file is either equal to the given parameter N or the total number of possible faulty combinations, whichever is smaller. The CFRA is therefore calculated for each fault file, and Equation (5.5) becomes

$$CFRA = \frac{k}{n_{sim}} \tag{5.6}$$

where n_{sim} is the total number of simulations, calculated as $n_{sim} = 2^n \cdot \min(N, n_{comb}(x))$. Here, N represents the desired number of fault vectors in a fault file, and $n_{comb}(x)$ represents the total number of possible faulty combinations for x faults.

To calculate the CFRA based on a given fault probability per gate (or device) instead of using a fixed number of faulty gates (or devices), the evaluation must be conducted as follows when utilizing the first simulation method: For a circuit with N gates (or devices), the probability of exactly x faulty gates (or devices) is given by the binomial distribution

$$P(\text{x faults}) = \binom{N}{x} \cdot p^x \cdot (1-p)^{N-x} \tag{5.7}$$

where p is the fault probability of a single gate (or device). The failure probability of a circuit, given that exactly x faults occur, is given by

$$P_{fail} = P(\text{x faults}) \cdot P_{fail|x} \tag{5.8}$$

where $P_{fail|x}$ denotes the failure probability of the circuit when exactly x gates (or devices) have failed. To calculate the total failure probability of the circuit (and thus the CFRA), the sum over all possible fault cases is calculated:

$$P_{fail\_total} = \sum_{k=1}^{N} \binom{N}{k} \cdot p^k \cdot (1-p)^{N-k} \cdot P_{fail|k}. \tag{5.9}$$

It is evident that this approach requires significant computational effort, as the failure probabilities of the circuit for all possible fault cases $P_{fail|k}$ must be determined through simulation. For circuits with a large or even moderate number of gates (or devices), this becomes infeasible. Furthermore, the binomial coefficient $\binom{N}{k}$ can become extremely large, making the calculation of Equation (5.7) impractical and also necessitating a limit on the number of simulated fault vectors. Therefore, the second simulation method (Section 5.1.2) is more suitable for calculating the CFRA, as the circuit's reliability (or failure rate) can be decoded from the output bit stream. If the circuit's reliability R is given, the CFRA can be calculated as

$$CFRA = 1 - R. \tag{5.10}$$

5.2.2 Circuit Fault Resilience

The second metric used for evaluation is the *Circuit Fault Resilience (CFRE)*. This metric is introduced because the CFRA does not provide any information about the number of failed input vectors for a combination of faulty gates (or devices). Therefore, the term Vulnerable Input Vector (VIV) is introduced [41]. An input vector is defined as a VIV for a specific combination of faulty gates (or devices) if it causes the fault to propagate to the circuit's output. The CFRE is defined as

$$CFRE = \frac{k_c}{n_{sim}} \tag{5.11}$$

where k_c is the number of faulty combinations of gates (or devices) for which the circuit produces correct outputs across all input vectors, and n_{sim} is the total number of simulations. Thus, the CFRE quantifies the probability that a circuit will maintain correct functionality across all input vectors when a certain number of gates (or devices) have failed. Therefore, a higher CFRE is desirable.

5.2.3 Gate/Device Count

Another important factor to consider when evaluating the performance of fault-tolerant design techniques is the increase in the total number of gates or devices required when these techniques are applied. Therefore, the *gate and device count* is defined as third metric.

Chapter 6

Simulation Results

This chapter begins by detailing the test circuits used in simulations to evaluate the selected fault-tolerant design techniques. It then presents an analysis of the simulation results at both the gate and device levels, exploring the design space and discussing the associated trade-offs. Due to the large amount of data generated by the simulations, only the results for the full adder are evaluated in the following sections. The results for the C17 and 74283 fast adder can be found in [42].

The procedure used to obtain the results is shown in Figure 6.1. It can be divided into two parts: the design space, and the simulation framework. The workflow begins in the design space by selecting a test circuit and a fault-tolerant design technique. Next, the test circuit is implemented with the chosen fault-tolerant design technique applied in Verilog. The resulting fault-tolerant test circuit is then simulated within the simulation framework. Before running the simulation, the abstraction level, fault type, and the simulation method must be selected. The Verilog testbench is configured based on the chosen options. When the simulations are finished, the results are automatically evaluated by the evaluation programs.

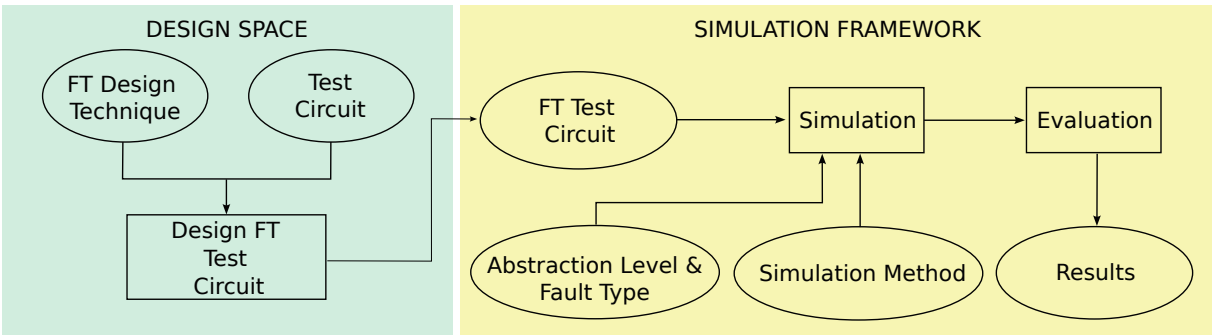


Figure 6.1: Workflow from design to evaluation.

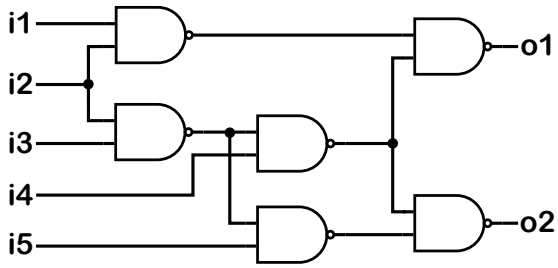


Figure 6.2: C17 circuit.

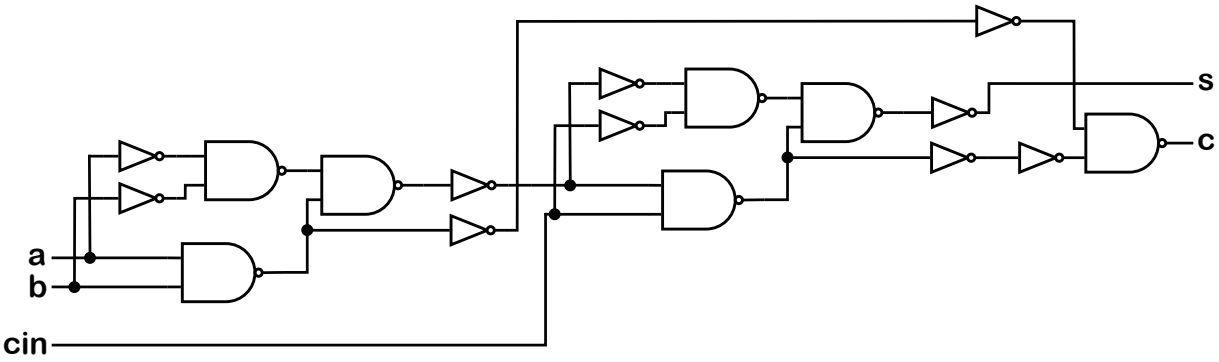


Figure 6.3: Full adder circuit.

6.1 Test Circuits

To evaluate the performance of the selected fault-tolerant design techniques, three circuit designs of different complexity were chosen as test candidates. Additionally, programs have been implemented to automatically create the fault-tolerant design for a test circuit by modifying its Verilog netlist. Moreover, the programs implement the necessary mechanisms for selecting the fault type and activating faults within the circuit, as described in Section 2.2 and Section 2.3.

6.1.1 C17

The first test circuit chosen is the ISCAS 85 circuit C17, which is commonly used in benchmarks for fault-tolerant design techniques [43]. This circuit has five inputs and two outputs and consists of six NAND gates (Figure 6.2).

6.1.2 Full Adder

The second test circuit chosen is a full adder, comprised of NAND gates and inverters (Figure 6.3). This design was selected because full adders are frequently used in various applications. It consists of a total of 17 gates.

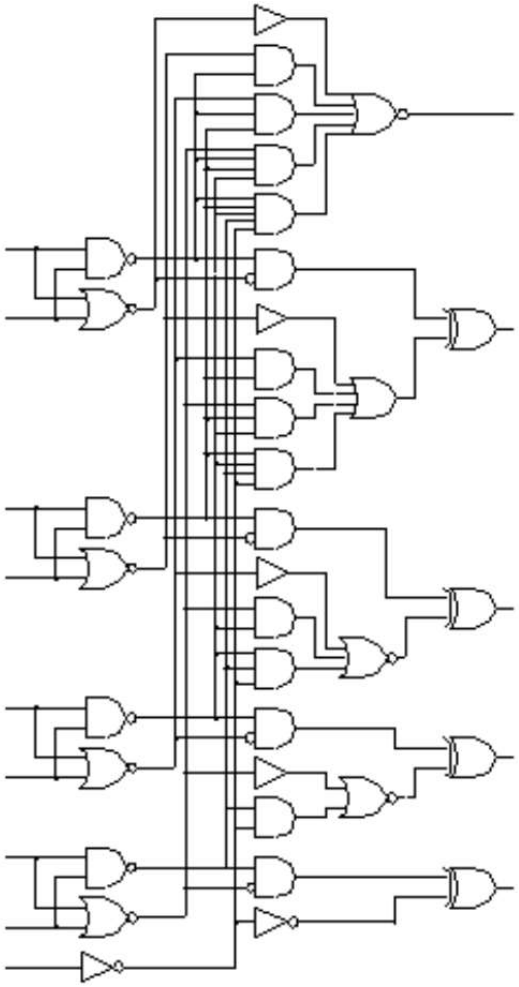


Figure 6.4: 74283 fast adder circuit [13].

6.1.3 74283 Fast Adder Circuit

The third test circuit chosen is the 74283 fast adder circuit of the 74X-Series (Figure 6.4) [13]. It has a total of 40 gates, making it the most complex test circuit used.

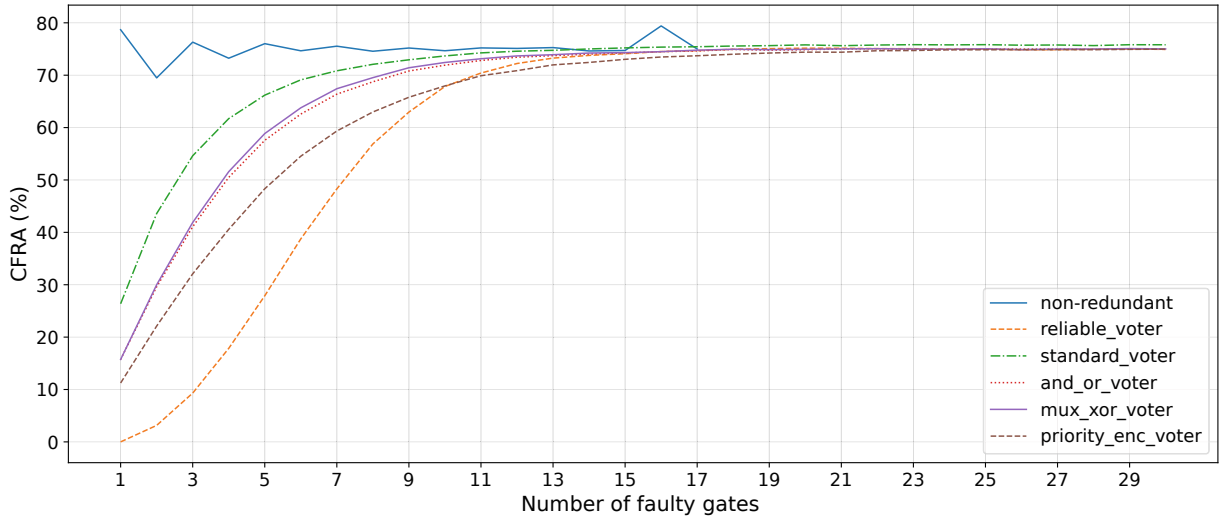
6.2 Gate Level Simulations

The first simulations were conducted at the gate level. Since design techniques that utilize redundancy at the device level cannot be tested from a gate level perspective - because this would require injecting faults into devices instead of gates - this section focuses on investigating TMR with different voter designs. In the following simulation results, TMR was applied to the test circuits at the gate level. This means that each gate in the circuit was triplicated, and their outputs were fed into a voter circuit. In addition to the TMR designs with the voter circuits presented above, a design with a reliable voter has also been implemented. The reliable voter is not affected by faults and is labelled as *reliable_voter*.

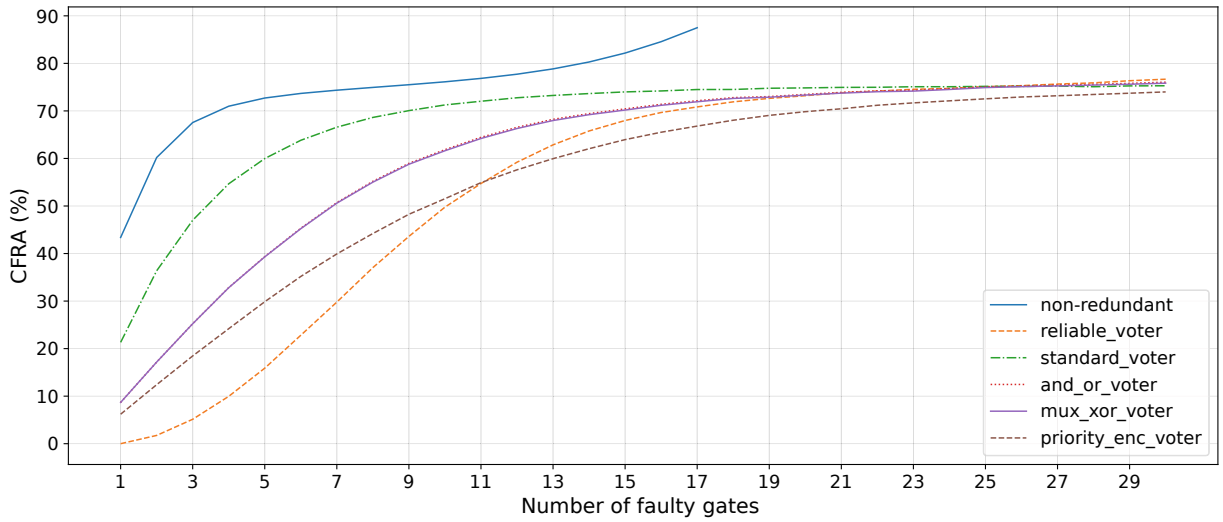
However, this voter cannot be implemented because voter circuits are always prone to faults. Therefore, this design serves solely as a benchmark for evaluating other voter designs.

The CFRAs for the full adder circuit with different TMR voter designs, plotted against the number of faulty gates for flipping errors, stuck-at-0 faults, and stuck-at-1 faults, are shown in Figure 6.5. When examining the lines representing the non-redundant test circuit, it can be observed that they end at 17 faulty gates. This is because the non-redundant circuit contains 17 gates in total, meaning that all gates have failed at this point. Further, it can be observed that the application of TMR significantly reduces the CFRA compared to the non-redundant circuit. The extent of the decrease depends on the number of faulty gates as well as the design of the voter used. The *reliable_voter* can tolerate one faulty gate of any fault type while maintaining a CFRA of 0%. It can also be observed that, as the number of faulty gates increases, the CFRA of the *reliable_voter* becomes worse than that of the other voters. The reason for this is that faults cannot occur in the *reliable_voter*, meaning they will instead occur in the gates of the circuit itself, while in the other designs, faults can also occur in the voting circuit (and may be tolerated). Thus, for a specific number of faulty gates in the circuit, the design with the reliable voter may produce more faulty gate outputs, making it worse than the other designs at some point. The best performing voter, next to the *reliable_voter*, is the *priority_enc_voter*. Assuming the same number of faulty gate occurs, using TMR with this voter design to address flipping errors can improve the CFRA from $75\% - 79\%$ in the non-redundant circuit to $11\% - 74\%$. For stuck-at-0 faults, the CFRA can be improved from $43\% - 88\%$ to $6\% - 67\%$. For stuck-at-1 faults, the *standard_voter* performs slightly better, as expected in Section 3.1.2.1, and the CFRA can be improved from $35\% - 88\%$ to $5\% - 60\%$. For a larger number of faulty gates, all voter designs will approach a CFRA of approximately $70\% - 80\%$ across all fault types.

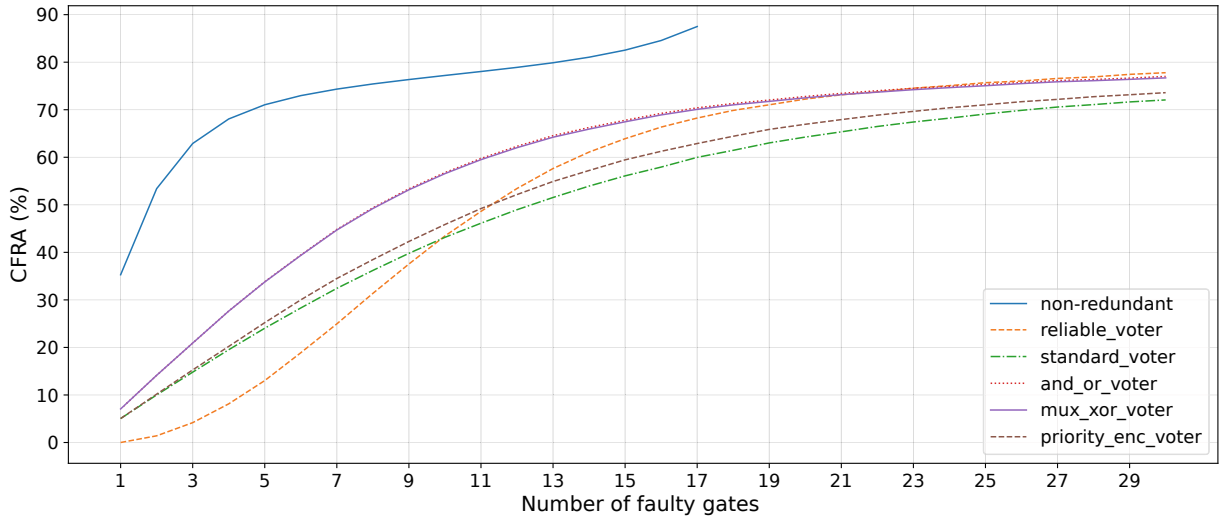
The CFREs for the full adder circuit are shown in Figure 6.6, using the same simulation setup as before. It can be observed that the CFRE is 0% for the non-redundant circuit in almost all scenarios, meaning it cannot tolerate any faults while maintaining correct functionality across all input vectors. The only voter design that can tolerate a single faulty gate and achieve a CFRE of 100% across all fault types and circuits is the *reliable_voter*. The next best-performing voter design is the *priority_enc_voter*. This is the only voter design that can maintain a CFRE of more than 80% when one fault of any type occurs. Among all the other voter designs, only the *standard_voter* performs similarly for stuck-at-1 faults. The performance of the *mux_xor_voter* and the *and_or_voter* is nearly identical, with the *mux_xor_voter* performing slightly better at higher fault counts.



(a)

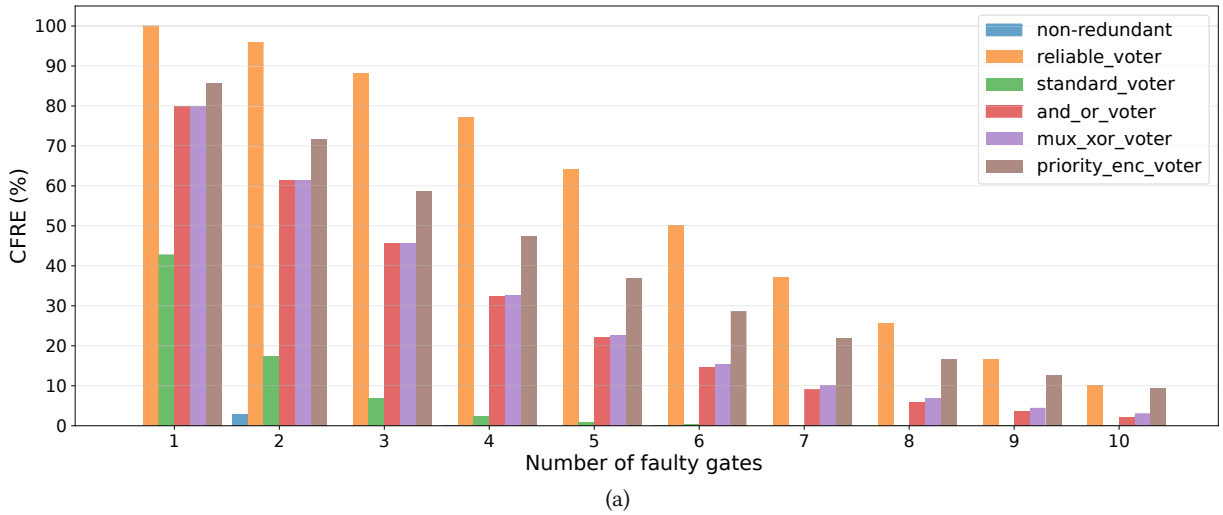


(b)

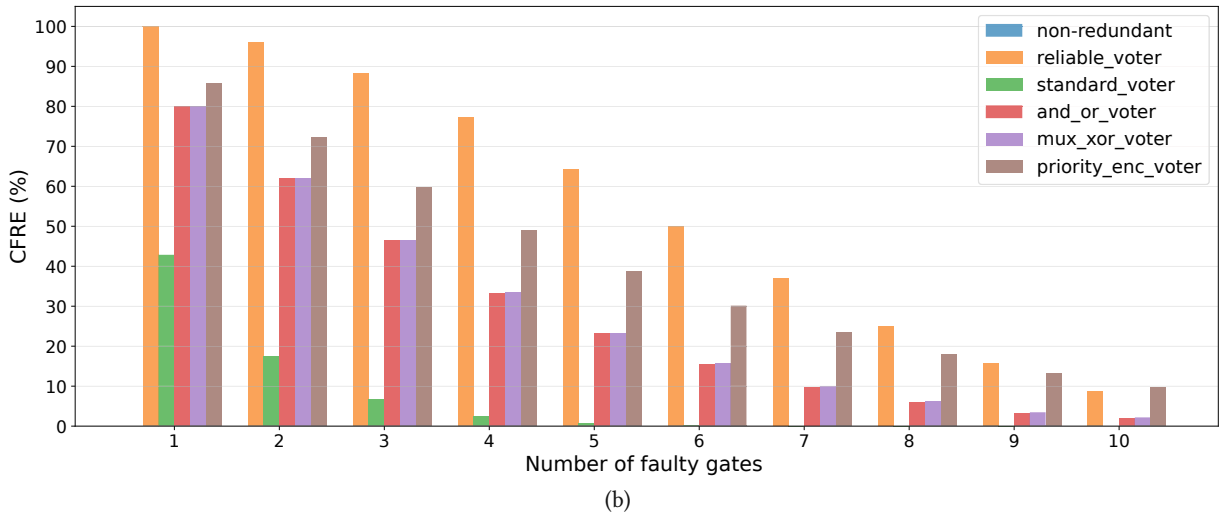


(c)

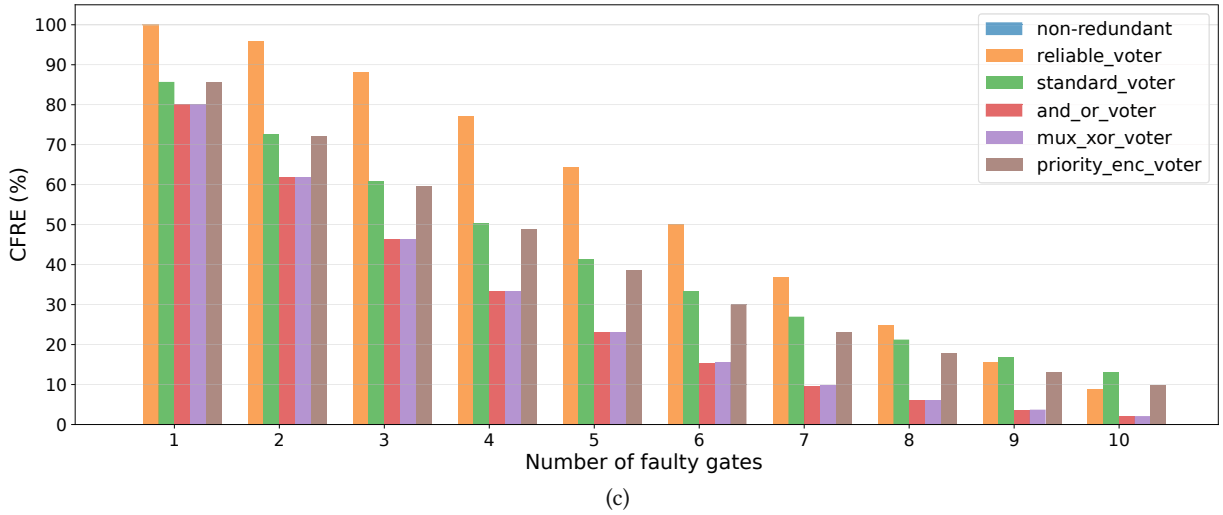
Figure 6.5: CFRA for the full adder with TMR and different voter designs: (a) flipping error, (b) stuck-at-0, (c) stuck-at-1.



(a)



(b)



(c)

Figure 6.6: CFRE for the full adder with TMR and different voter designs: (a) flipping error, (b) stuck-at-0, (c) stuck-at-1.

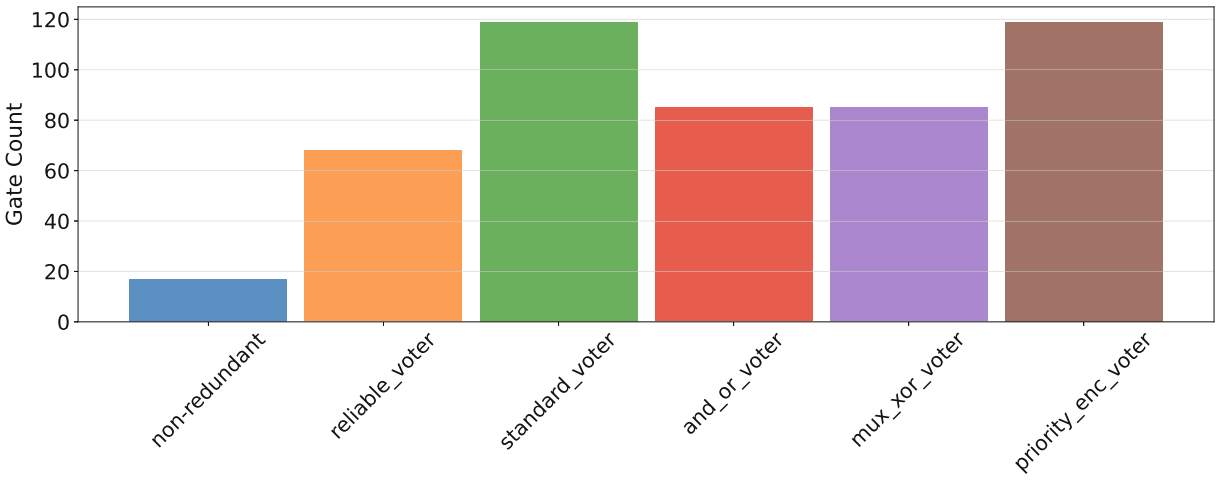


Figure 6.7: Gate count of the full adder with TMR and different voter designs.

The gate counts of the full adder with TMR are shown in Figure 6.7 for different voter designs. It is evident that the voter designs with higher fault tolerance also have higher gate counts. Since a certain number of faulty gates can have a greater impact on a circuit with fewer gates than on one with more gates, it is useful to relate the CFRE to the number of additional gates introduced by the redundant circuit. Thus, the increase in CFRE per additional gate is calculated as

$$CFRE_{inc,k} = \frac{\Delta CFRE_k}{n} \tag{6.1}$$

where $\Delta CFRE_k = CFRE_{red,k} - CFRE_{non\_red,k}$ represents the difference in CFRE between the redundant and non-redundant circuit for k faulty gates, and n is the number of additional gates introduced by the redundant circuit. Figure 6.8 shows the results for one and for five faulty gates. It can be seen that, for one faulty gate, the *priority_enc_voter* is less efficient than the *mux_xor_voter* and the *and_or_voter* across all test circuits, even though it has a higher overall CFRE. With five faulty gates, the efficiency of the *priority_enc_voter* is slightly higher. This is expected, as the CFRE of the other voters decreases more rapidly with an increasing number of faulty gates. Consequently, the efficiency of the *priority_enc_voter* improves as the number of faulty gates increases.

From a gate level perspective, the choice of the voter design therefore depends on the number of expected faults and the required CFRE or CFRA. When the required CFRE or CFRA can be met, the *mux_xor_voter* and the *and_or_voter* provide more efficient designs at low numbers of faulty gates, as illustrated in Figure 6.8. The reason for their higher efficiency is that their performance difference compared to the *priority_enc_voter* at lower numbers of faulty gates is marginal, while requiring fewer gates. However, assuming a fixed number of faulty gates, the *priority_enc_voter* can tolerate a higher number of faults than the other voter designs, meaning its performance degrades less rapidly as the

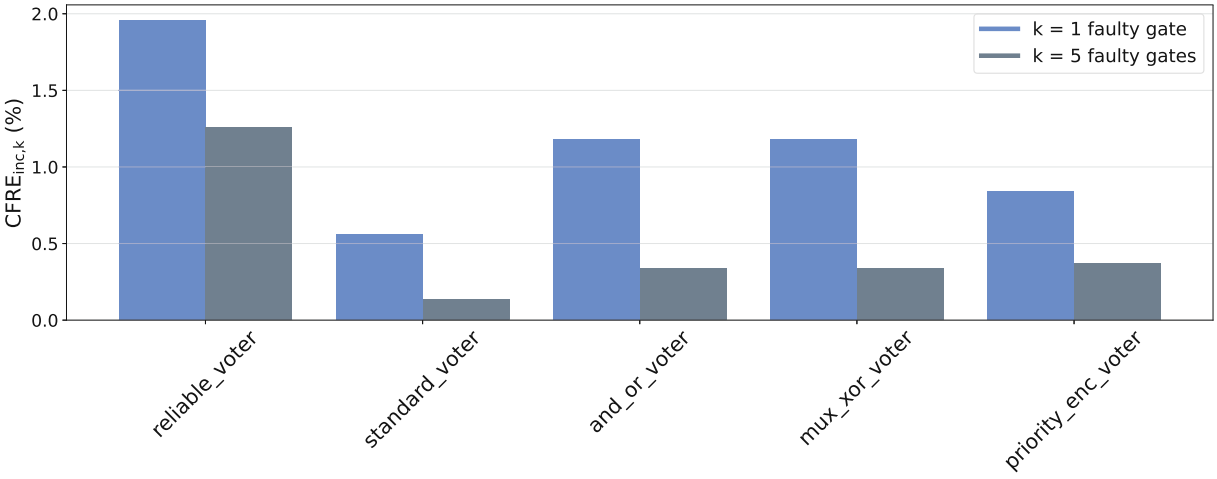


Figure 6.8: Increase in CFRE per additional gate for the full adder with TMR averaged over all fault types.

number of faulty gates increases, making it more efficient at higher fault counts. Considering all fault types, the *standard_voter* is the least efficient design, performing worse than other designs while having the same gate count as the *priority_enc_voter*. However, it performs slightly better on stuck-at-1 faults, so it may be a viable choice when this fault type is most likely to occur.

6.3 Device Level Simulations

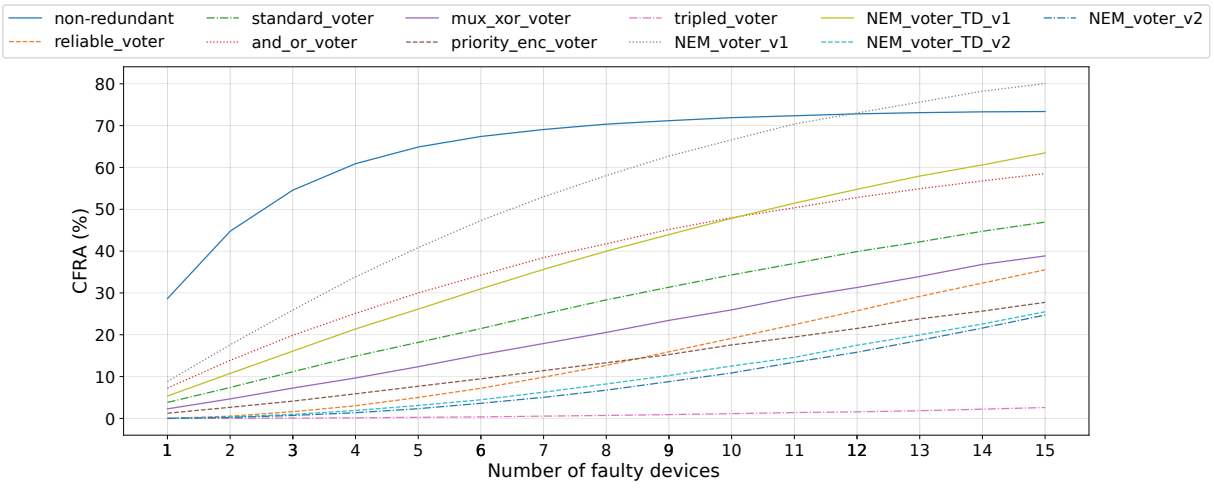
This section elaborates on the simulations conducted at the device level. Device level simulations allow for a more detailed analysis of the test circuits and the fault-tolerant design techniques applied. Furthermore, a more realistic view of the hardware cost is provided, as the gate count does not account for the number of NEM devices used. First, conventional TMR voter designs are evaluated and compared with the novel NEM voter designs. The second section examines the design techniques that utilize device level redundancy and compares them to the TMR designs.

6.3.1 TMR

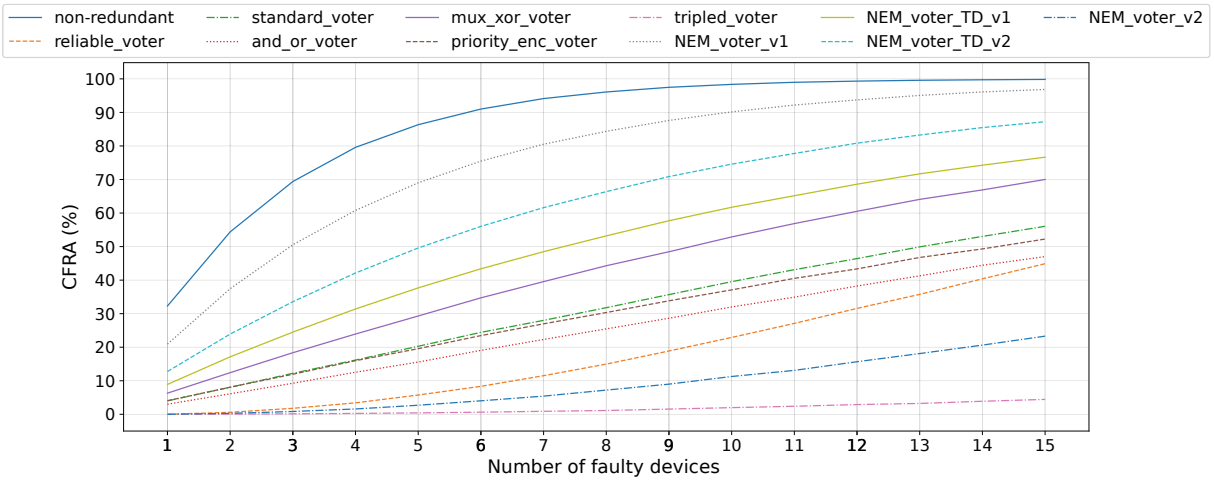
The TMR approach, along with the conventional voter designs used for gate level simulations in the previous section, is now analysed at the device level for defects. Additionally, the novel NEM voter designs can now be modelled, simulated, and compared to conventional voter designs in terms of their performance. As previously mentioned, TMR can be applied to modules of varying sizes and complexities. To explore how module size impacts the fault tolerance of the TMR design, two extreme cases were considered. In the first case, TMR was implemented at the lowest possible level of abstraction, the gate level. In the second case, TMR was applied at the highest possible level of abstraction, the circuit level.

6.3.1.1 Application at Gate Level

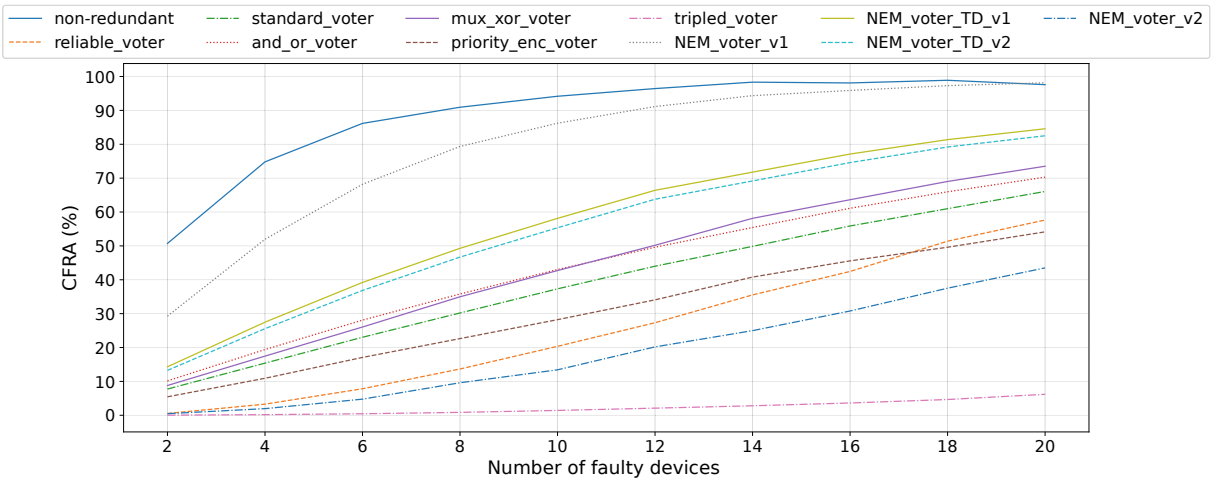
At this level of abstraction, each gate in the test circuit is replicated three times, and the outputs of the replicated gates are fed into a voter. The CFRAs for the full adder circuit with TMR applied at gate level and different voter designs are illustrated in Figure 6.9 for different numbers of faulty devices. The graphs cover three fault scenarios: stuck-off faults, stuck-on faults, and mixed faults, consisting of a distribution of 50% stuck-off and 50% stuck-on faults. It can be observed that all TMR voter designs can reduce the CFRA compared to the non-redundant circuit, given a specific number of device failures. The *tripled_voter* shows the best performance across all fault types. For stuck-off faults, applying TMR with this voter design can improve the CFRA from $29\% - 73\%$ in the non-redundant circuit to $0\% - 3\%$. For stuck-on faults the CFRA can be improved from $32\% - 100\%$ to $0\% - 4\%$, and for mixed fault types the CFRA can be improved from $51\% - 98\%$ to $0\% - 6\%$. The next best voter is the *NEM_voter_v2*, which performs similarly to the tripled voter at low numbers of faults but performs increasingly worse as the number of faults increases.



(a)



(b)



(c)

Figure 6.9: CFRA for the full adder with TMR applied at gate level and different voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

The CFREs for the full adder circuit with TMR and conventional voter designs are shown in Figure 6.10. The non-redundant circuit can maintain correct functionality across all input vectors with only a 10% probability in the presence of a single stuck-off fault. If more faults or other types of faults occur, its CFRE approximates or drops to 0%. The *tripled_voter* exhibits the best CFRE due to its highly redundant design, meaning that it can maintain a CFRE of more than 80% even with higher numbers of faults, regardless of the fault type. Especially in a mixed fault scenario, where stuck-off and stuck-on faults can occur simultaneously, the *tripled_voter* is the only viable choice at high numbers of faulty devices. The results for the novel NEM voter designs are shown in Figure 6.11. It is evident that the *NEM_voter_v2* achieves the highest CFRE across all fault types and also outperforms most conventional voters. As expected, the *NEM_voter_TD_v1* performs better for stuck-on faults, whereas the *NEM_voter_TD_v2* performs better for stuck-off faults when compared to each other.

To evaluate the efficiency of the voters, their device counts need to be considered. Figure 6.12 illustrates the device counts of full adder using TMR and different voter designs. While achieving the best performance in terms of fault tolerance, the *tripled_voter* also clearly has the highest device count. To quantify the efficiency of the different designs, the increase in CFRE per additional device is calculated, similar to the approach in the previous chapter. Figure 6.13 shows the results for two and six faulty devices. It can be concluded that the *tripled_voter*, despite having the best performance, is less efficient than the other voter designs due to its high device overhead. The *mux_xor_voter* performs relatively poorly in terms of CFRE; however, its low device count makes it slightly more efficient than the other voters when dealing with a small number of faulty devices. Similar observations can be made about the *NEM_voter_v1*, which does not outperform the other voters in terms of performance but offers the lowest device count. Therefore, its design is more suitable for optimizing device count or area, at the cost of lower fault tolerance. In general, the *NEM_voter_v2* can be considered the most efficient design (aside the *reliable_voter*), particularly when dealing with a higher number of faulty devices.

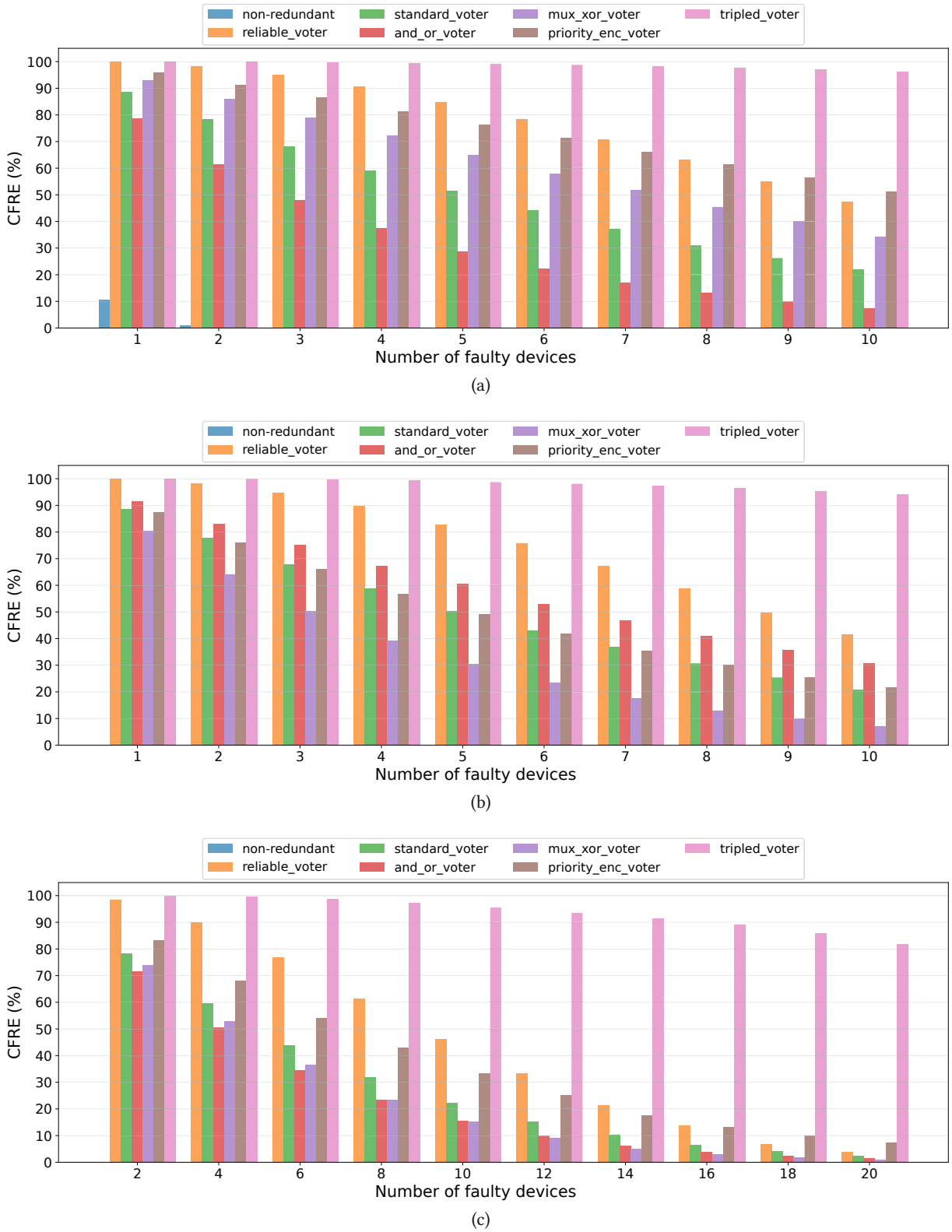
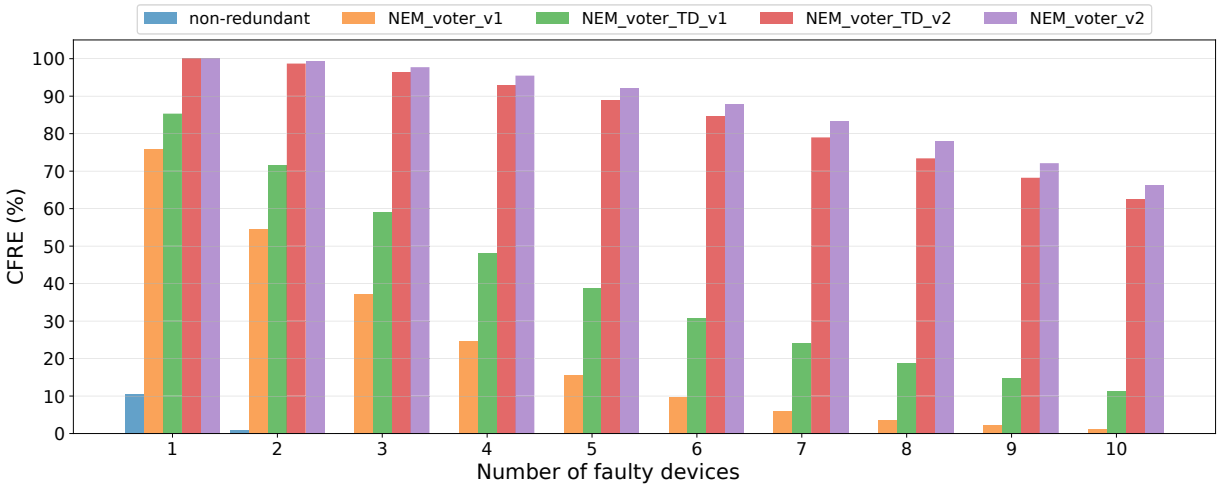
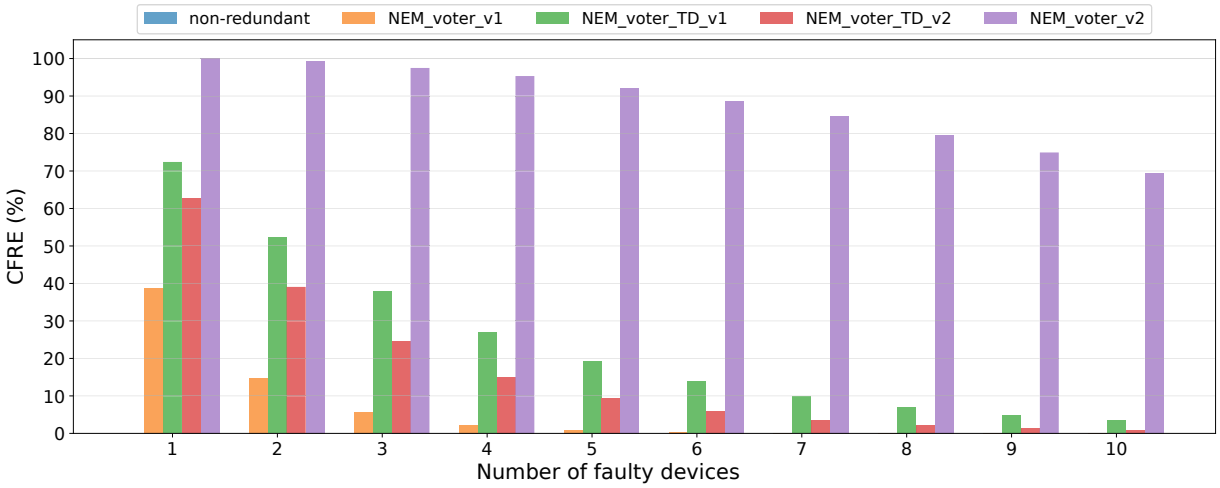


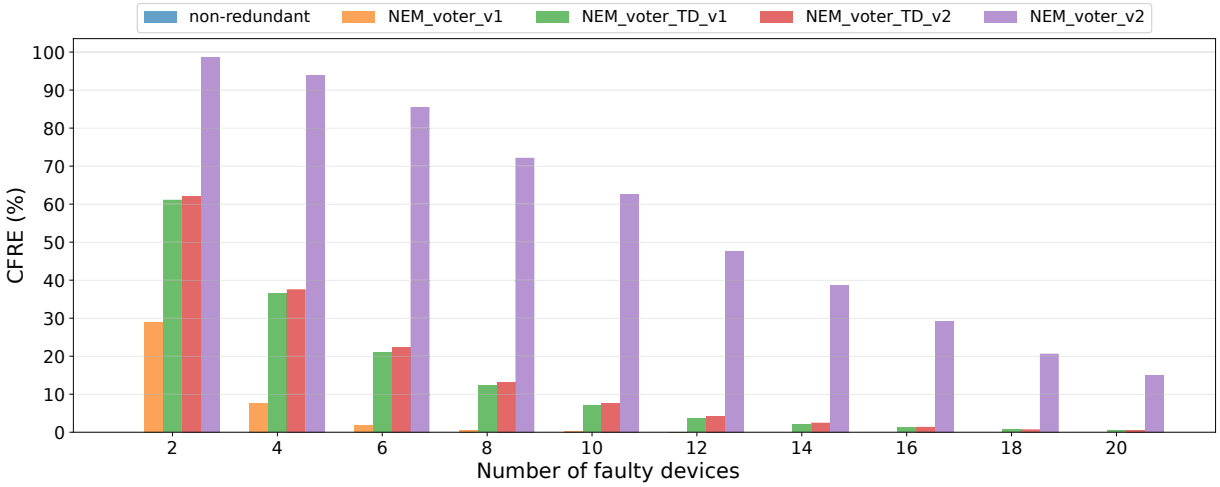
Figure 6.10: CFRE for the full adder with TMR applied at gate level and conventional voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).



(a)



(b)



(c)

Figure 6.11: CFRE for the full adder with TMR applied at gate level and NEM voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

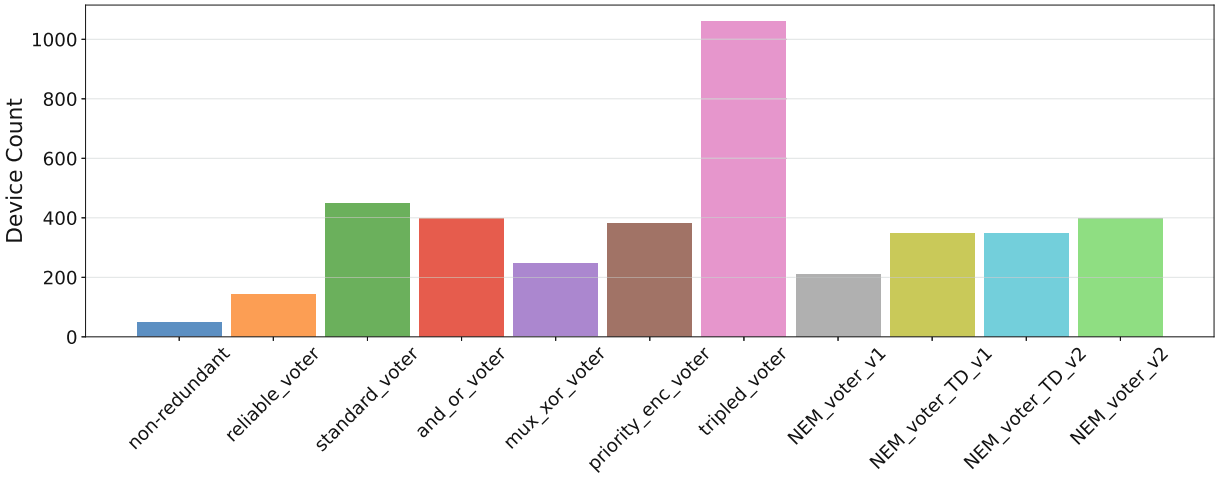


Figure 6.12: Device counts of the full adder with TMR applied at gate level and different voter designs.

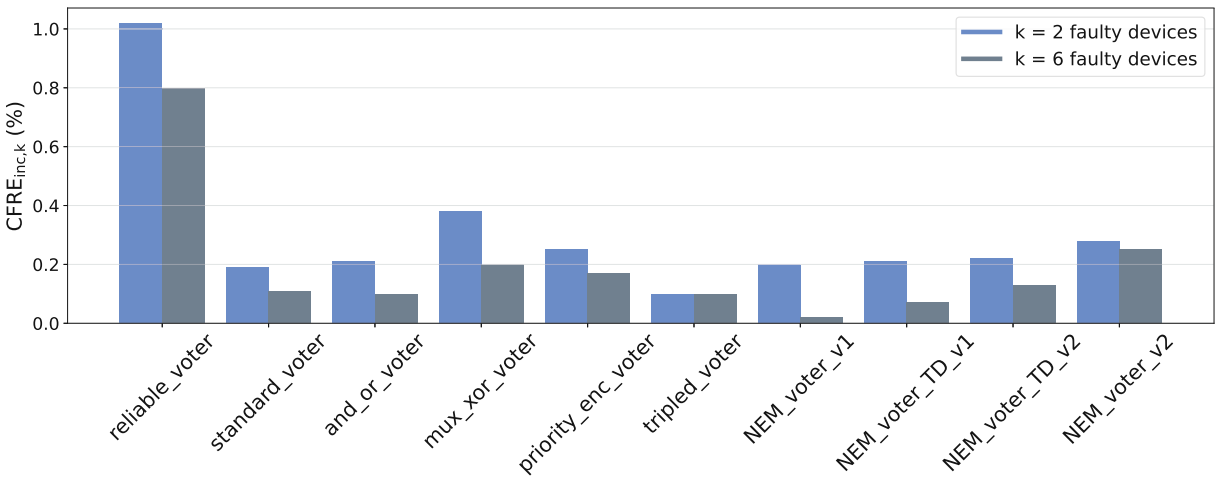


Figure 6.13: Increase in CFRE per additional device for the full adder with TMR applied at gate level averaged over all fault types.

When examining the graphs in Figure 6.9 and Figure 6.10, it becomes evident that the *reliable_voter* performs worse than some other voters. The reason for this peculiarity is that these simulations assume a constant number of faulty devices, as specified in the first simulation method (Section 5.1.1). A specific number of faulty devices may impact a design with fewer devices more significantly than a design with a greater number of devices. As shown in Figure 6.12, the design with the *reliable_voter* has the lowest device count among all TMR designs. To account for the fact that a circuit with a greater number of devices may experience more faults than one with a smaller device count, the device fault probability is considered instead of a fixed number of faults, using the second simulation method (Section 5.1.2). Figure 6.14 illustrates the CFRAs for the full adder circuit, considering scenarios with low device fault probabilities ($0.01\% - 0.1\%$) and high device fault probabilities ($1\% - 2\%$). As expected, the *reliable_voter* now performs best in this simulation setup. Moreover, not all voter designs succeed in decreasing the CFRA compared to the non-redundant circuit. In the scenarios with low device fault probabilities, the designs with the *tripled_voter* and the *NEM_voter_v2* manage to decrease the CFRA to almost 0% for all fault types. In the scenarios with high fault probabilities, the CFRA can be reduced from $8\% - 15\%$ to $1\% - 4\%$ for stuck-off faults, from $18\% - 27\%$ to $2\% - 6\%$ for stuck-on faults, and from $21\% - 38\%$ to $6\% - 21\%$ for mixed faults with these two voter designs. Further, the difference in performance of all voter designs is smaller at low fault rates and becomes larger as the device fault probability increases. It can be observed that the lines for the *tripled_voter* and *NEM_voter_v2* have a smaller slope than the others at low device fault probabilities, meaning that their CFRAs increase more slowly than those of the other voters.

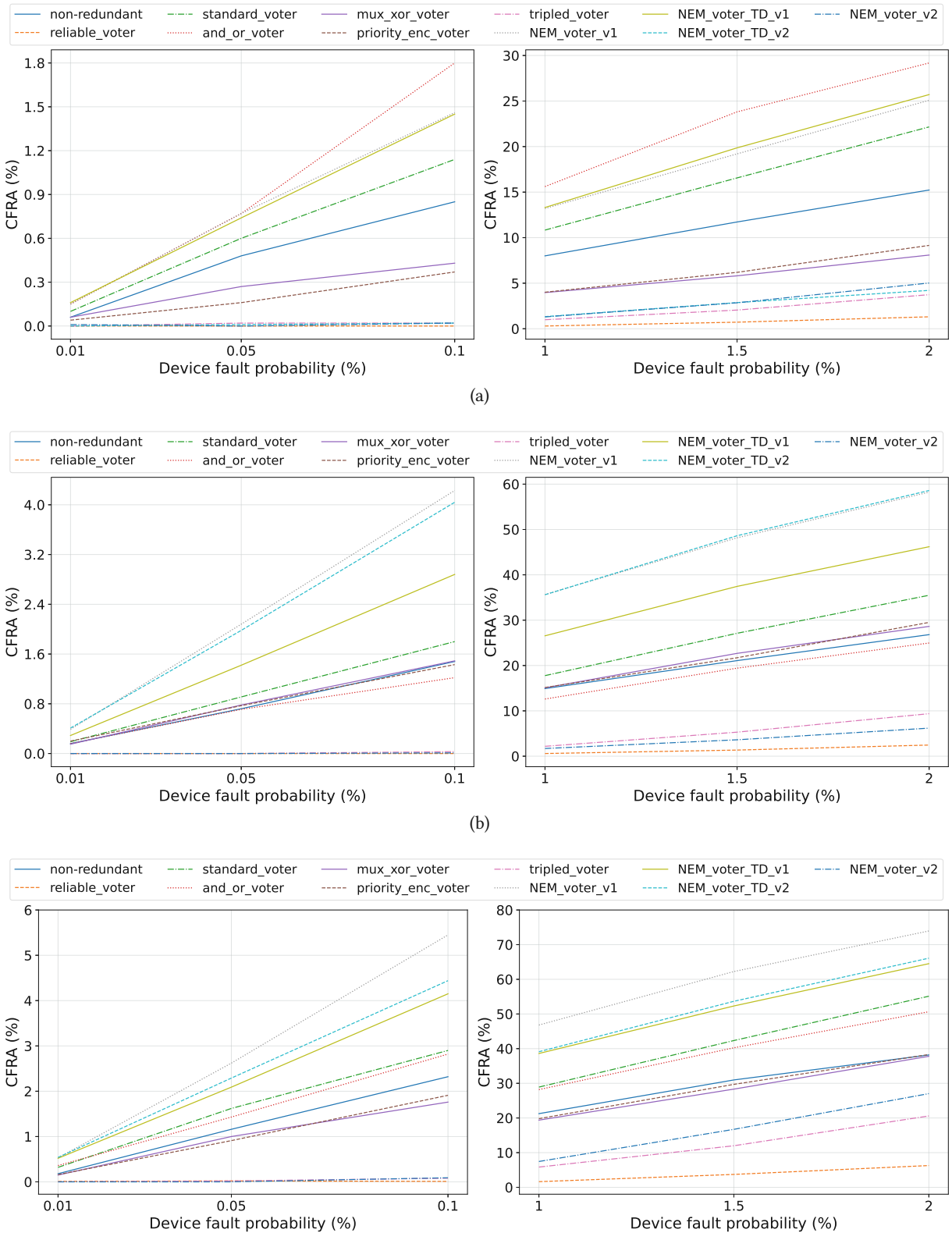
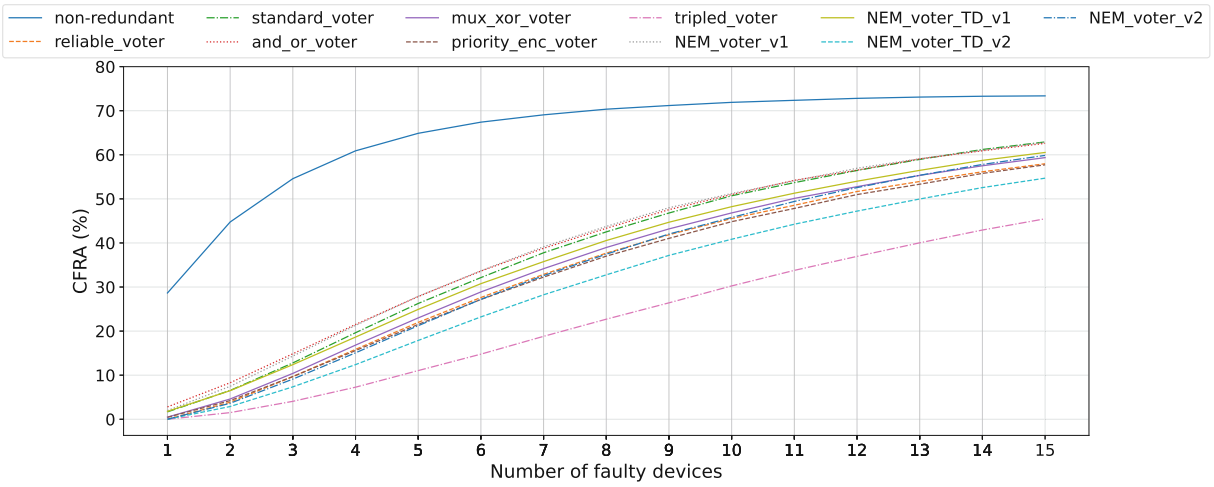


Figure 6.14: CFRA for the full adder with TMR applied at gate level for different device fault probabilities: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

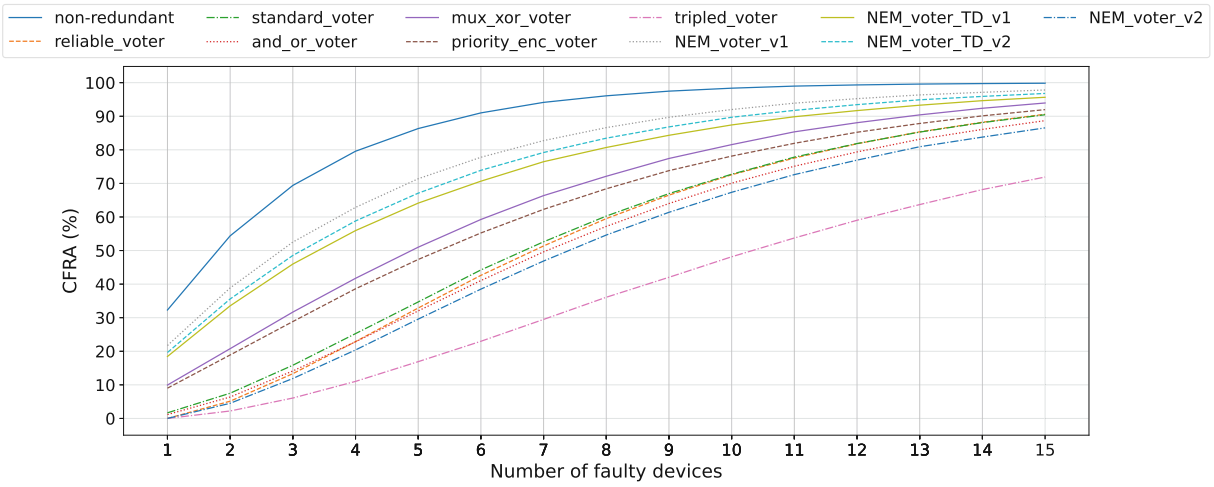
6.3.1.2 Application at Circuit Level

At this abstraction level, the entire circuit is replicated three times, and their outputs are fed into a voter. The CFRAs for the full adder circuit with TMR applied at circuit level are shown in Figure 6.15 for different numbers of faulty devices. The first observation that can be made is that the performance differences between the voter circuits are smaller compared to the approach in which TMR was applied at the gate level. This is because it is much less likely for a voter to be affected by a fault, as this approach only requires only $3n$ voters, where n is the number of outputs of the test circuit. Consequently, the choice of the voter design becomes less critical compared to the previous approach. The performance difference between the voters is comparable to that of TMR applied at gate level, meaning that the *tripled_voter* also performs best here. In general, it can be concluded that in this simulation setup, the application of TMR at circuit level performs worse than its application at gate level. A lower or approximately equivalent CFRA can only be achieved by some voter designs at one or two faulty devices. The results also indicate that the CFRA for TMR applied at the circuit level becomes worse compared to its application at gate level as the number of faults increases. This is because a higher fault count is more likely to affect all three circuit duplicates, resulting in incorrect outputs being fed into the voter. An exception is the *NEM_voter_v1*, which performs better for stuck-off faults and approximately the same for stuck-on faults and mixed faults, even at higher numbers of faulty devices. However, its performance remains worse than that of the other voters.

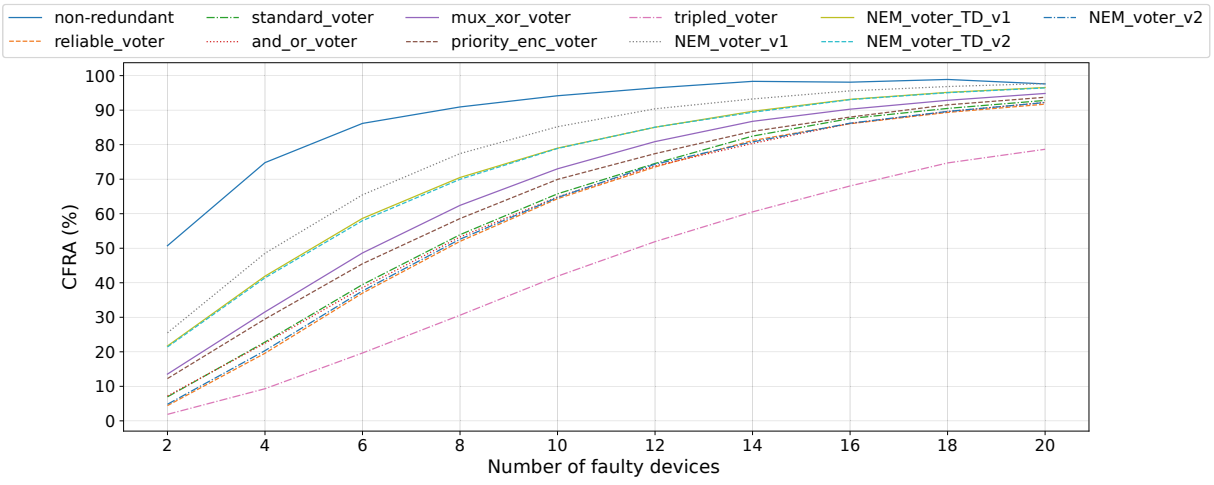
The CFREs for the full adder circuit with TMR and conventional voter designs are shown in Figure 6.16. Again, a higher CFRE is achieved only with some voters when one or two devices are faulty. The CFREs for the novel NEM voters are illustrated in Figure 6.17. The *NEM_voter_v1* performs better than its TMR implementation at gate level but cannot outperform the other voters. Furthermore, the results for stuck-on faults confirm that the *NEM_voter_v2* is significantly more resilient to unknown logic levels at its inputs caused by these faults.



(a)

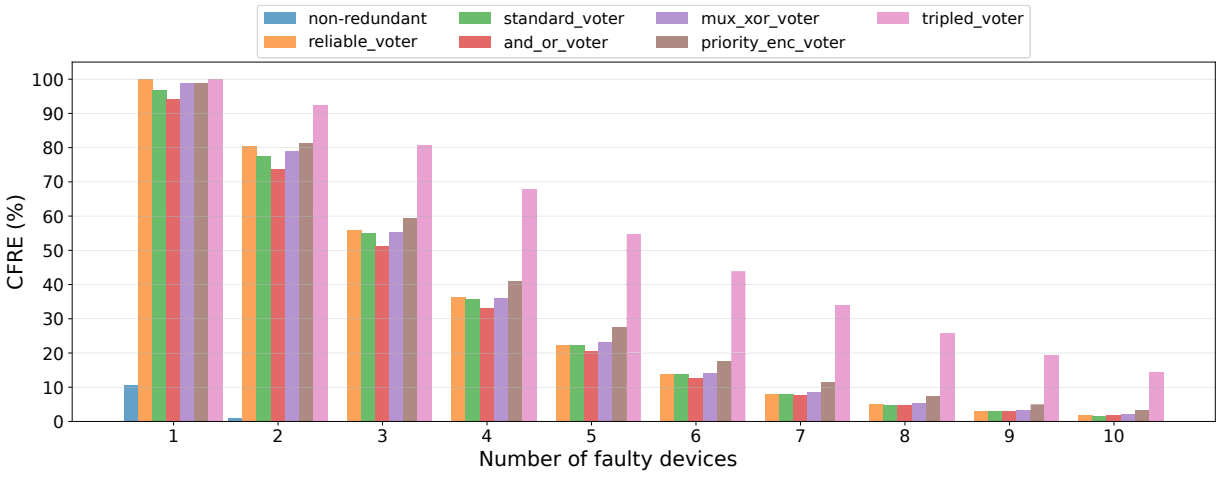


(b)

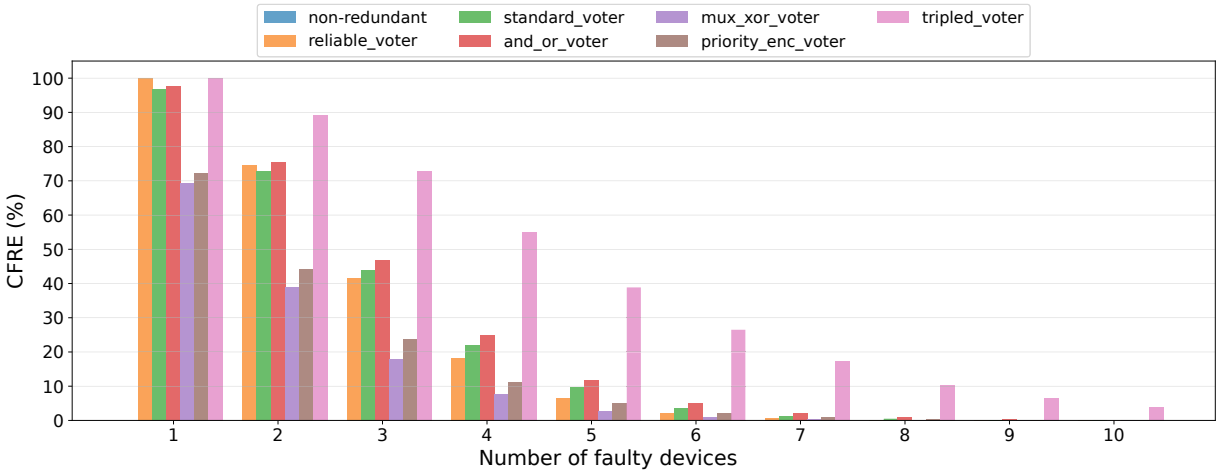


(c)

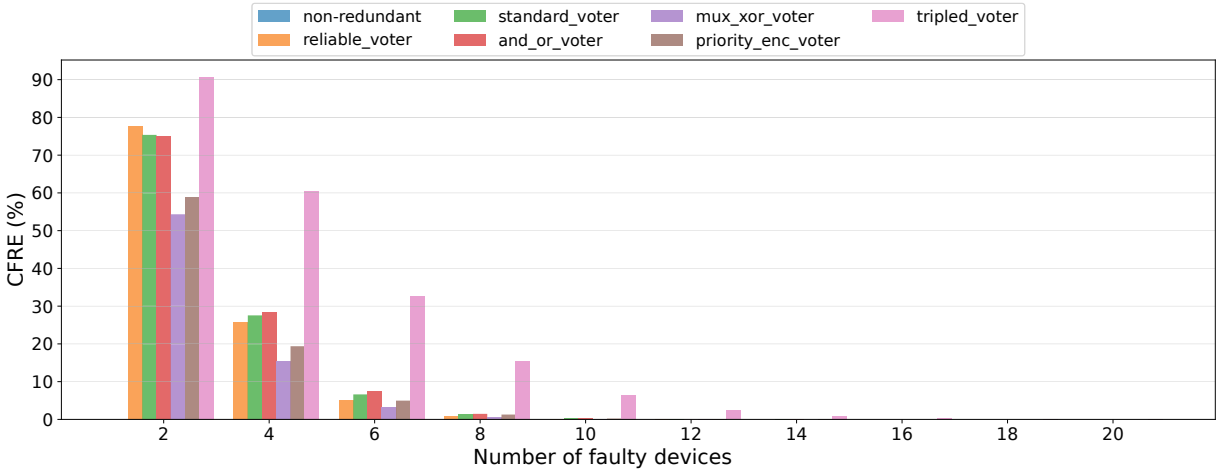
Figure 6.15: CFRA for the full adder with TMR applied at circuit level and conventional voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults.



(a)

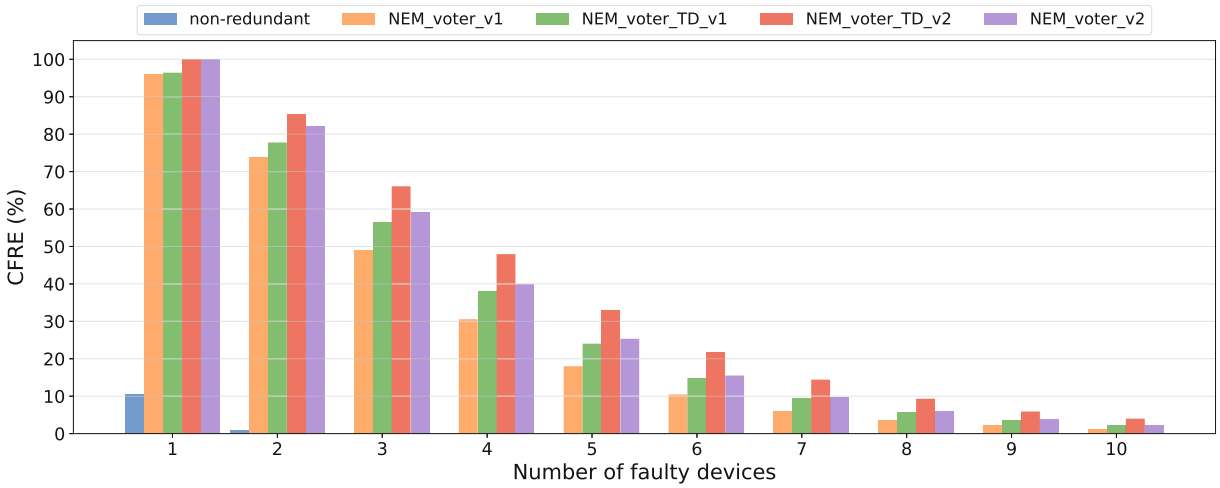


(b)

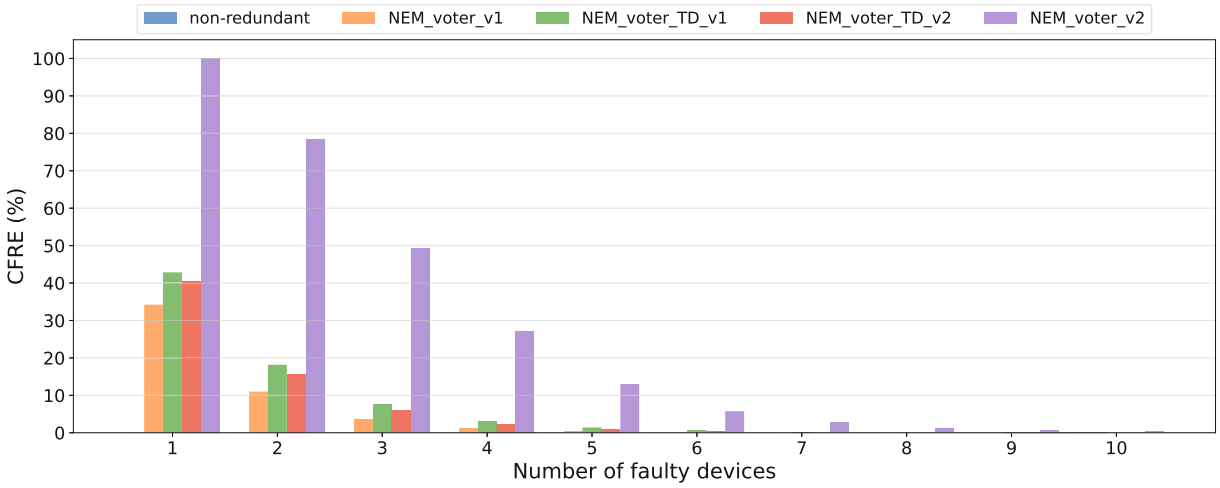


(c)

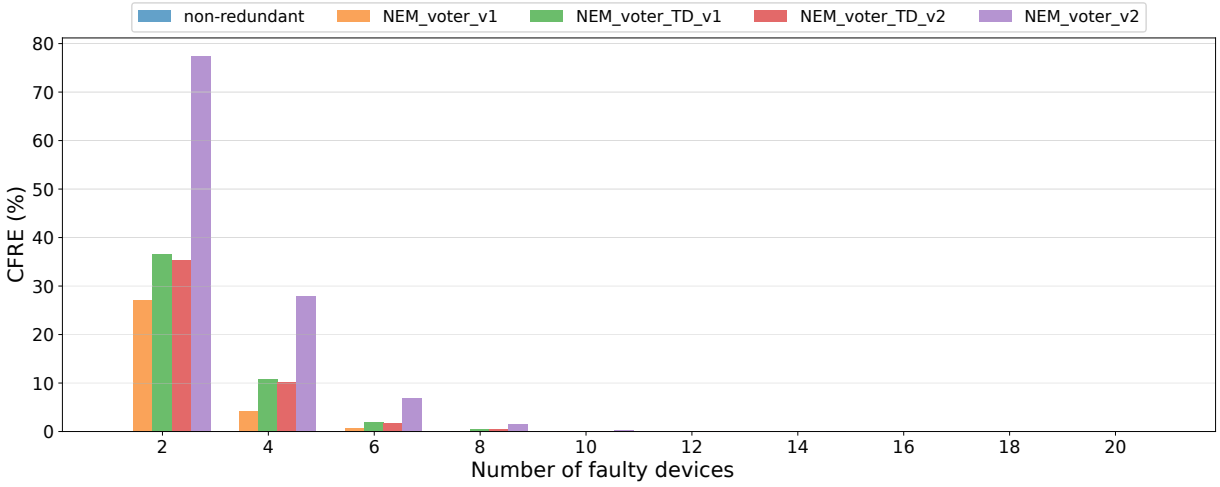
Figure 6.16: CFRE for the full adder with TMR applied at circuit level and conventional voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults.



(a)



(b)



(c)

Figure 6.17: CFRE for the full adder with TMR applied at circuit level and NEM voter designs: (a) stuck-off, (b) stuck-on, (c) mixed faults.

The device counts of the test circuits with TMR and different voter designs are shown in Figure 6.18. Naturally, the device counts are significantly lower than in the TMR implementations applied at gate level. A review of the increase in CFRE per device in Figure 6.19 reveals that, among all test circuits, only the *NEM_voter_v1* can be considered more efficient than its TMR implementation at the gate level when the number of faulty devices is low. When comparing the performance of the voters relative to each other, the *NEM_voter_v2* again shows high efficiency for both low and high numbers of faults. For low numbers of faults, the *mux_xor_voter* is more efficient, as it has a lower device count and only a slightly lower CFRE. However, as Figure 6.16 shows, this is mostly confined to stuck-off faults, while the *NEM_voter_v2* offers good performance across all fault types. Furthermore, all voter designs become less efficient as the number of faulty devices increases.

As discussed in the previous subsection, it is reasonable to consider the device fault probability in simulations to account for the increased likelihood of faults in circuits with higher device counts. Figure 6.20 shows the CFRAs for the full adder circuit with TMR applied at circuit level, under scenarios with low and high device fault probabilities. For stuck-off faults, all voter designs significantly reduce the CFRA compared to the non-redundant circuit. Furthermore, for this fault type, the results for most voter designs are better or approximately equal to those of the TMR implementation at gate level. Considering the voter that performs best in most fault scenarios, which is the *tripled_voter*, the CFRA can be reduced to 0% at low device fault probabilities, and from $8\% - 15\%$ in the non-redundant circuit to $2\% - 8\%$ at high device fault probabilities. For stuck-on faults, not all voter designs are capable of reducing the CFRA. The *tripled_voter* can reduce it to approximately 0% at low device fault probabilities, and from $15\% - 27\%$ to $6\% - 18\%$ at high device fault probabilities. For mixed fault types, the CFRA can again be reduced to approximately 0% at low device fault probabilities, and from $21\% - 38\%$ to $13\% - 37\%$ at high device fault probabilities. The *NEM_voter_v2* achieves similar performance with a lower device count, as shown in Figure 6.18. For most voter designs, it can be concluded that in this simulation setup, the CFRA decreases when TMR is applied at circuit level. The reason for this is that a TMR implementation with a smaller module size requires a larger number of voters, resulting in a higher total device count. Consequently, the circuit will experience a greater number of faults when a constant device fault probability is assumed, and the reliability of the circuit with TMR applied at the gate level becomes largely dependent on the design of the voter. Therefore, only voters with high intrinsic fault tolerance, such as the *tripled_voter*, the *NEM_voter_v2*, and, particularly, the *reliable_voter*, perform better when TMR is applied at gate level. However, the high device overhead associated with TMR applied at gate level may still make the approach at the circuit level more practical, regardless of the voting circuit.

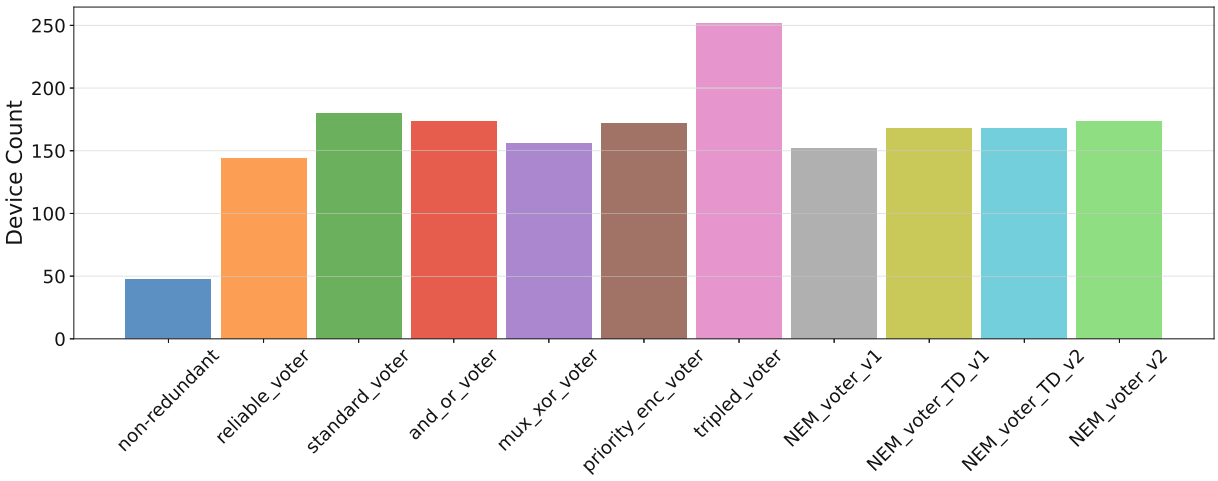


Figure 6.18: Device counts of the full adder with TMR applied at circuit level and different voter designs.

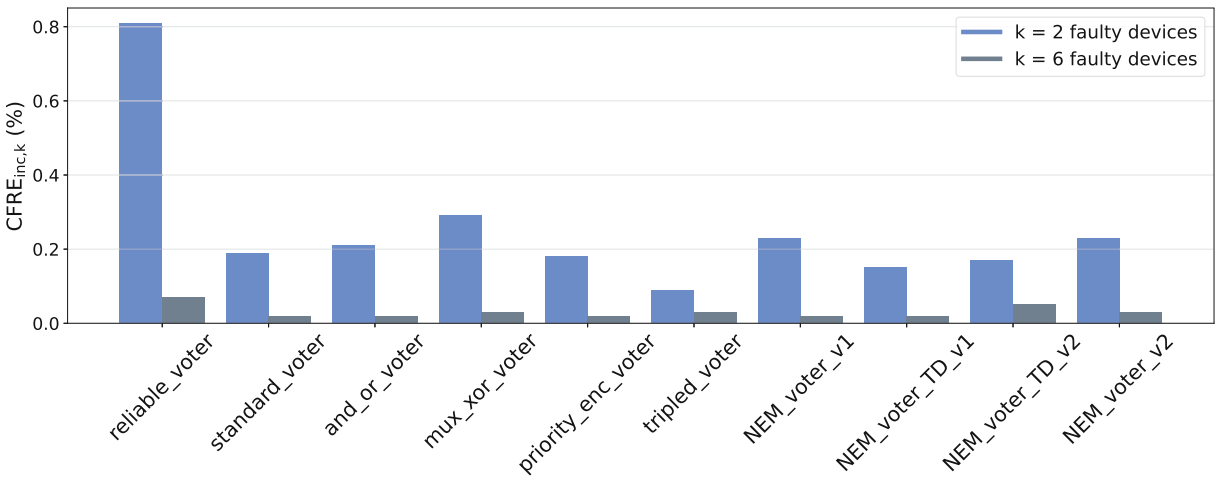


Figure 6.19: Increase in CFRE per additional device for the full adder with TMR applied at circuit level averaged over all fault types.

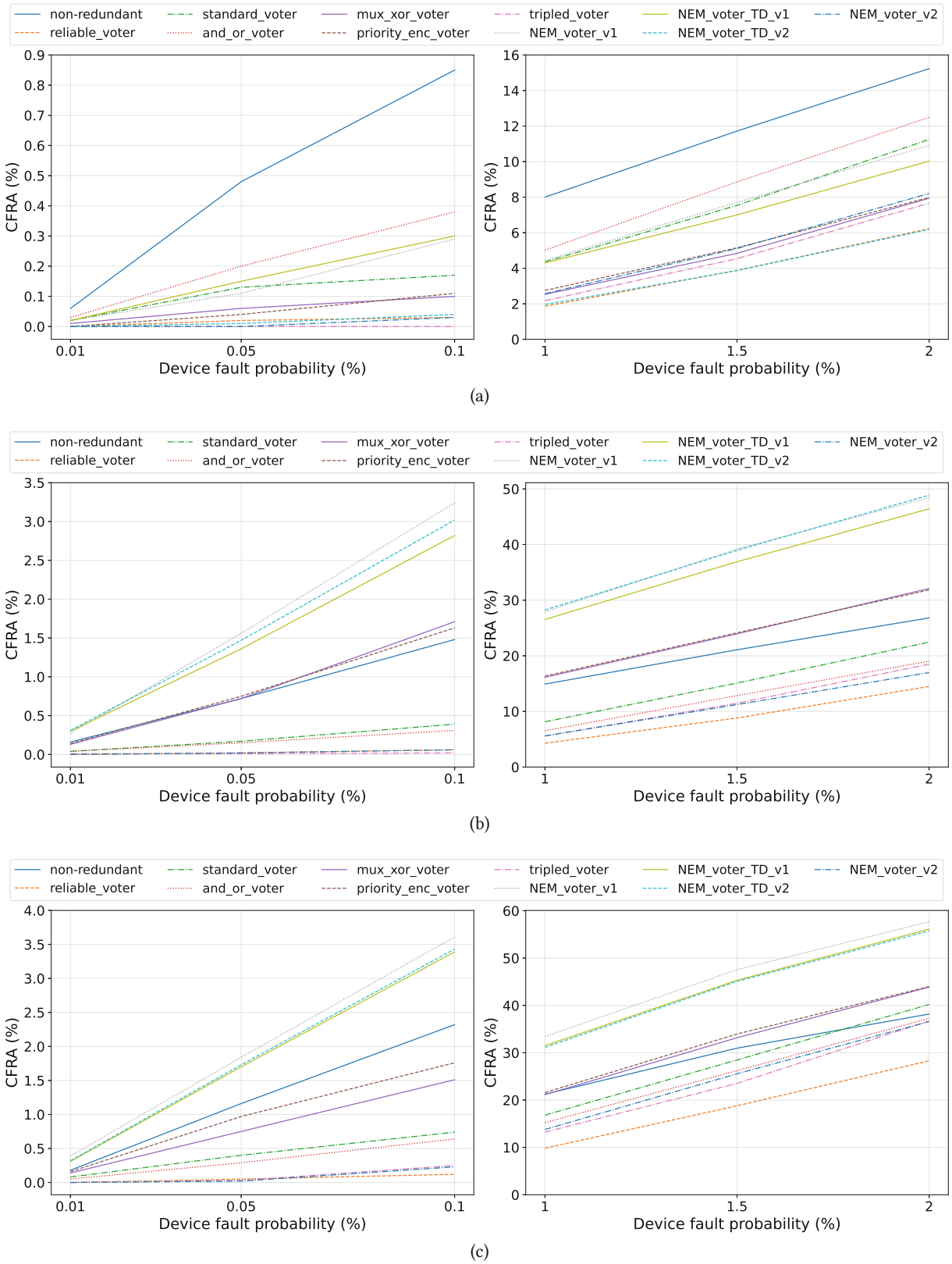
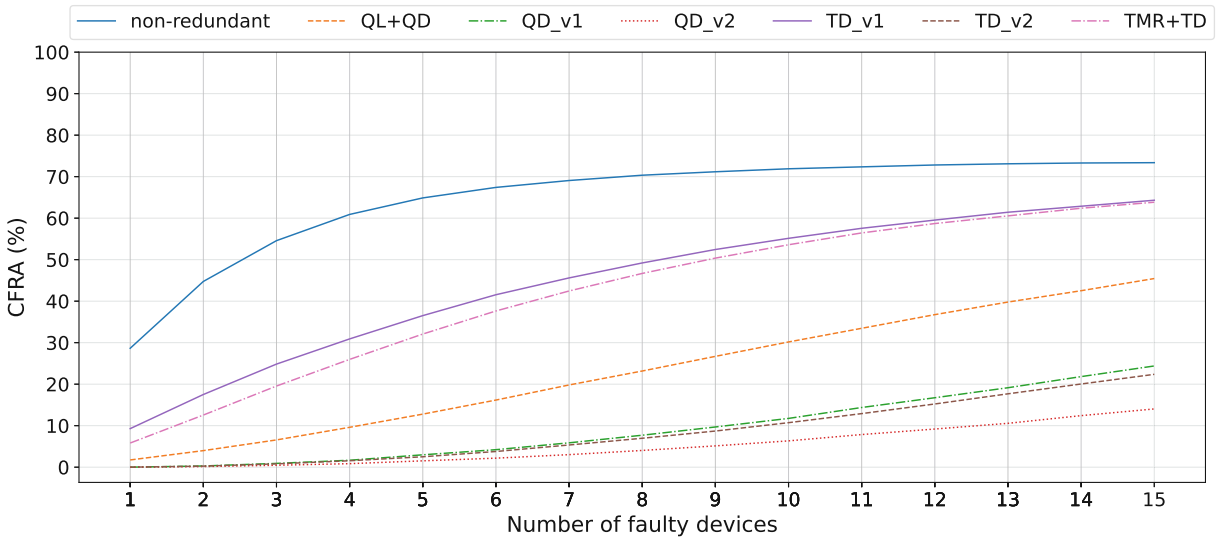


Figure 6.20: CFRA for the full adder with TMR applied at circuit level for different device fault probabilities: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

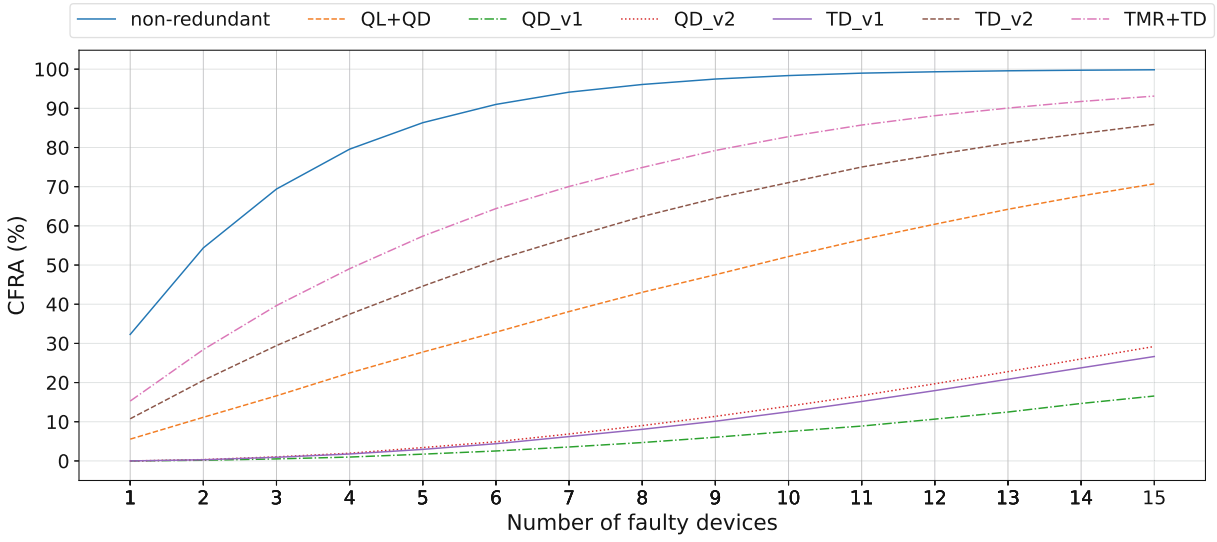
6.3.2 Device Level Design Techniques

In the final subsection, the remaining device level techniques are evaluated and compared to the TMR approach. Figure 6.21 shows the CFRAs for the full adder circuit for different numbers of faulty devices. It is evident that all design techniques can significantly reduce the CFRA compared to the non-redundant circuit. The results also confirm that *QD_v2* and *TD_v2* are more resilient to stuck-off faults, while *QD_v1* and *TD_v1* are more resilient to stuck-on faults. However, due to the higher degree of redundancy in QD, the performance difference between the two versions is smaller than in TD. Generally, QD exhibits the best performance across all fault types. For stuck-off faults, the CFRA can be reduced from $29\% - 73\%$ to $0\% - 14\%$, for stuck-on faults from $32\% - 100\%$ to $0\% - 17\%$, and for mixed faults from $51\% - 98\%$ to $0\% - 16\%$.

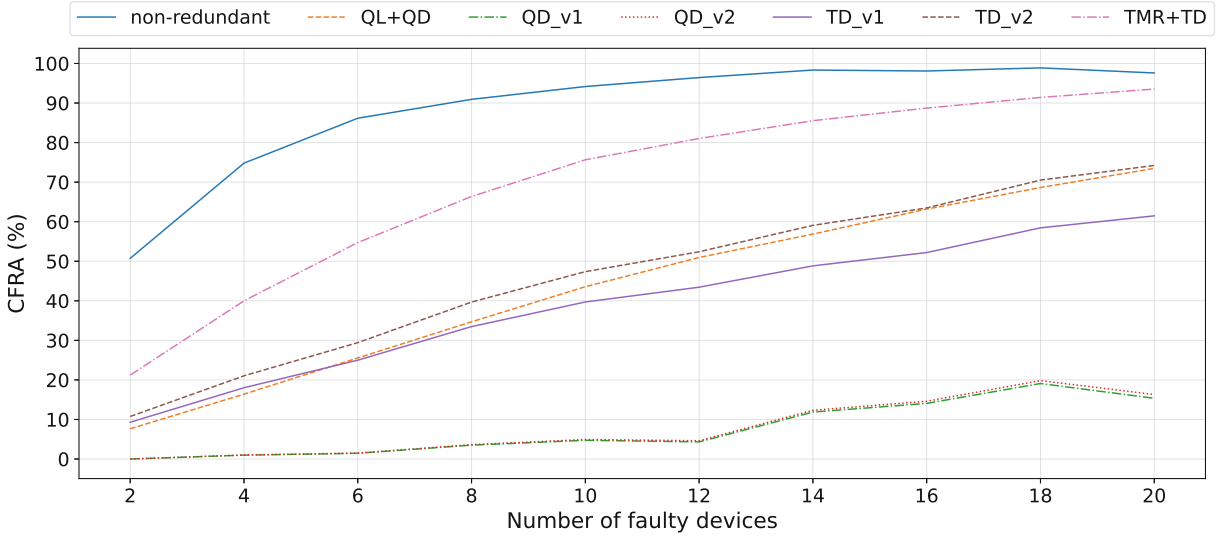
The CFREs for the full adder circuit are depicted in Figure 6.22. In the scenarios with stuck-off faults and stuck-on faults, both versions of QD can maintain a CFRE close to 100% at lower numbers of faulty devices and more than 60%, or even higher, at larger numbers of faulty devices, depending on the fault type and the version used. It also can be observed that for TD, the version suited to the corresponding fault type performs similar as QD, while the other version performs rather poorly. This indicates that in TD, the performance of the selected version is much more affected by the type of fault that occurs than that of QD, which can also be seen in the mixed fault type scenario, where both versions of TD perform poorly.



(a)



(b)



(c)

Figure 6.21: CFRA for the full adder with device level design techniques: (a) stuck-off, (b) stuck-on, (c) mixed faults.

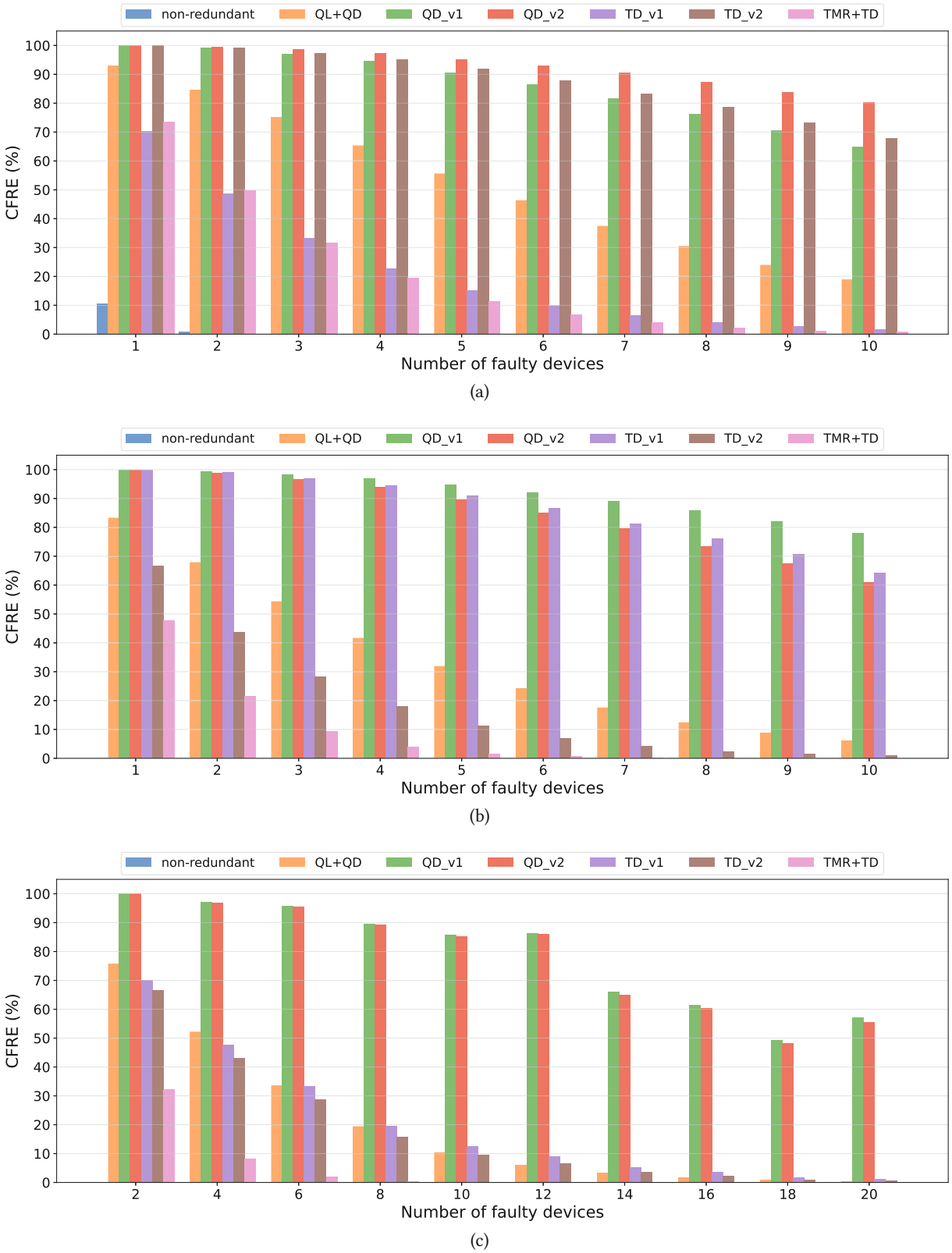


Figure 6.22: CFRE for the full adder with device level design techniques: (a) stuck-off, (b) stuck-on, (c) mixed faults.

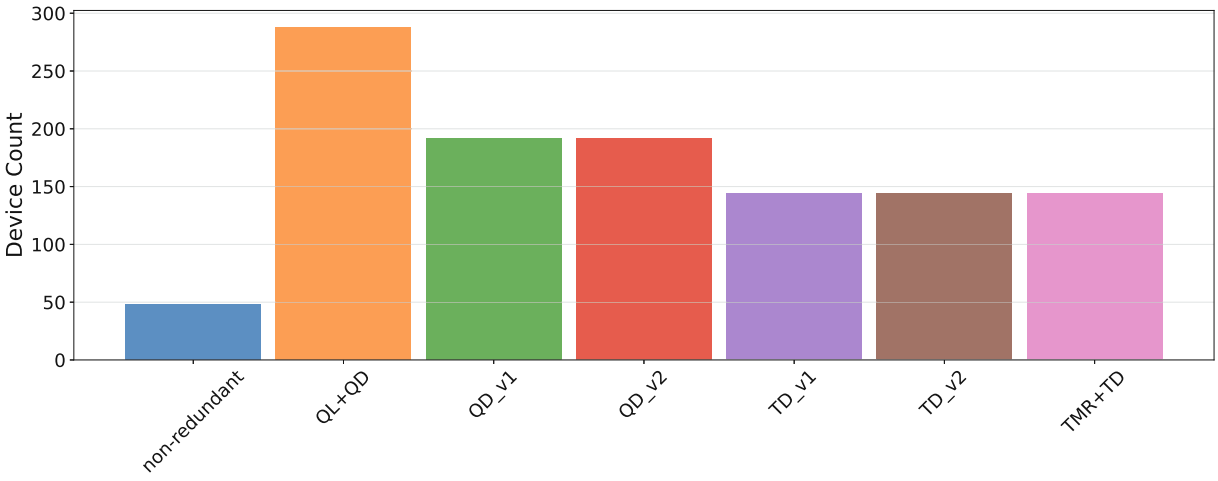


Figure 6.23: Device counts of the full adder with device level design techniques.

The device counts of the circuit designs are shown in Figure 6.23. It can be observed that the investigated designs generally have a lower device count compared to most TMR approaches. The increase in CFRE per additional device can be seen in Figure 6.24. The device level design techniques are generally more efficient than TMR techniques. At low numbers of faulty devices, TD is the most efficient technique, whereas QD becomes more efficient as the number of faulty devices increases.

When considering device fault probabilities in the simulations, the results indicate that, for certain design techniques, the additional redundancy introduced does not always lead to a lower CFRA (Figure 6.25). At low device fault probabilities, both versions of QD can reduce the CFRA to 0% for both stuck-off faults and stuck-on faults. Out of the two versions of TD, the one that is more resilient to the respective fault type can achieve a comparable CFRA, while the other version performs similarly to the non-redundant circuit. At higher device fault probabilities, QD and the respective version of TD can reduce the CFRA from $4\% - 15\%$ to $0\% - 0.7\%$ for stuck-off faults and from $18\% - 27\%$ to $0.3\% - 1.2\%$ for stuck-on faults. In the mixed fault type scenario, QD can reduce the CFRA to 0% at lower device fault probabilities and from $21\% - 38\%$ to $0.7\% - 2.7\%$ at higher device fault probabilities. Further, both versions of TD perform better than the non-redundant circuit for mixed fault types, but worse than QD. The performance of the other techniques largely depends on the fault type. Both *QL+QD* and *TMR+TD* outperform the non-redundant circuit for stuck-off faults at any device fault probability, but perform slightly or significantly worse for stuck-on or mixed faults.

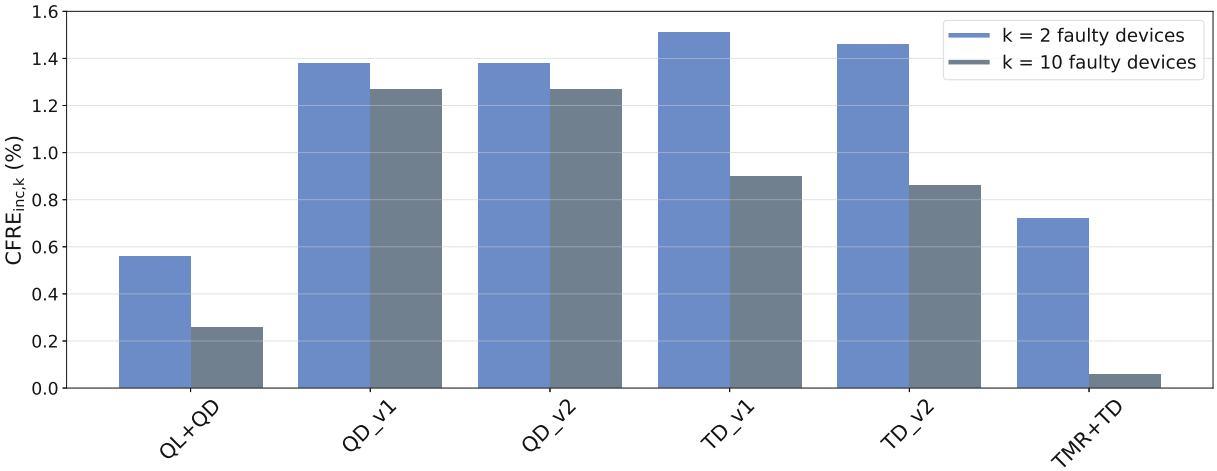


Figure 6.24: Increase in CFRE per additional device for the full adder with device level design techniques averaged over all fault types.

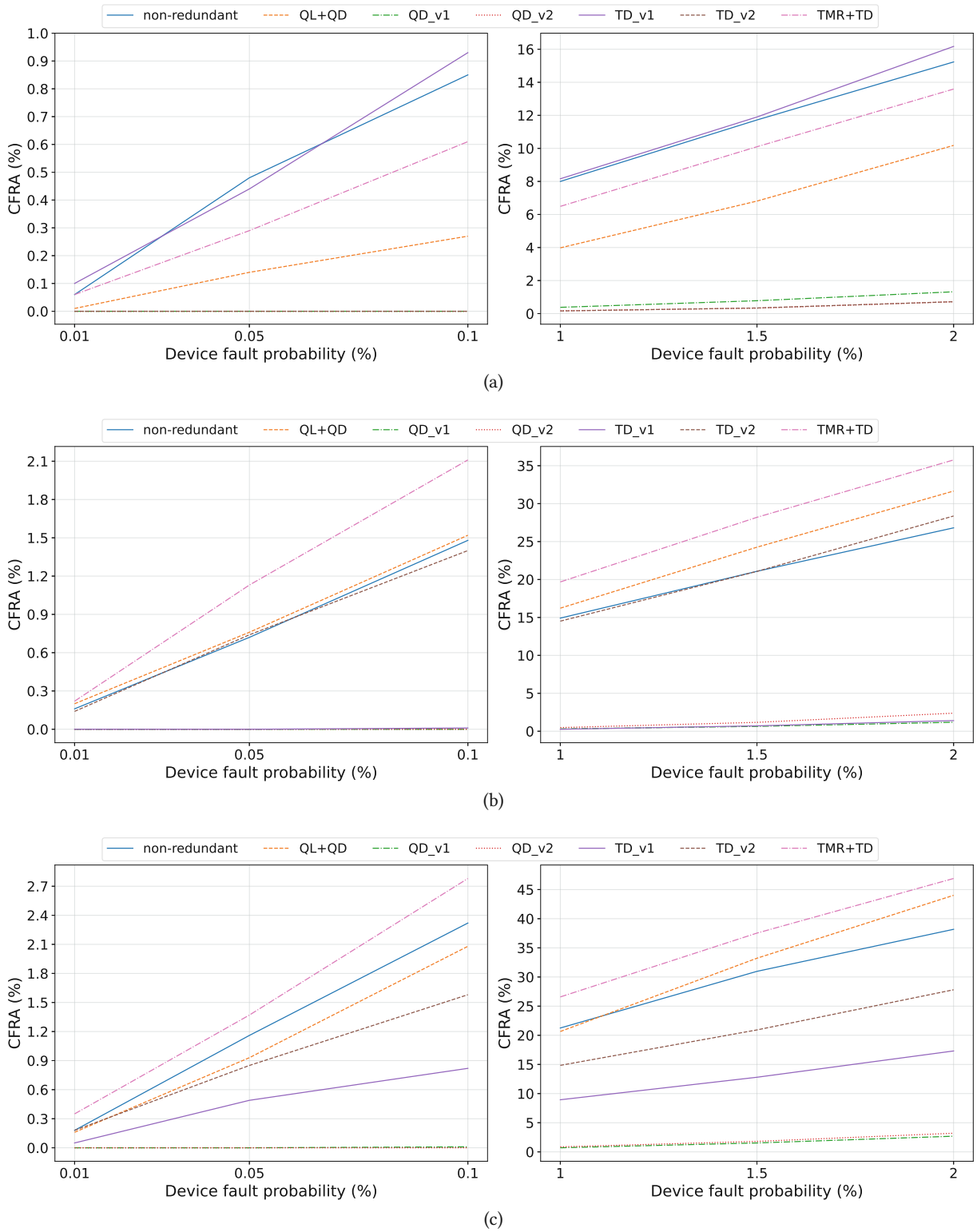


Figure 6.25: CFRA for the full adder with device level design techniques for different device fault probabilities: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

Finally, to explore the trade-offs between fault tolerance at a specific device fault probability and the device count in the discussed design techniques, their CFRA is plotted against device count. To keep the graphs clear, only the best-performing TMR designs have been selected for plotting and comparison (the suffix 'CL' indicates that TMR is applied at circuit level). Figure 6.26 shows the results for the full adder with 0.1% device fault probability. This specific representation now allows for identifying Pareto optimal points. In the case of stuck-off faults, there is only one Pareto optimal point (excluding the non-redundant circuit), which is *TD_v2*. This design technique has the lowest device count and, at the same time, can already reduce the CFRA to 0%. The same holds for stuck-on faults, where *TD_v1* is the only Pareto optimal point. In the mixed fault scenario, four different Pareto points can be identified. The first point is *TD_v1*, which has the lowest device count but the highest CFRA among the four points. The second and third points are *QD_v1* and *QD_v2*, both of which have a CFRA of 0% but the highest device count. The fourth point is the *NEM_voter_v2_CL*, which has a CFRA and device count that fall between those of the other three Pareto points. This means that the choice of design technique depends on the parameter that needs to be optimized. If the primary goal is to minimize the CFRA, regardless of the device count, then *QD_v1* or *QD_v2* are the appropriate choices. If minimizing device count or area optimization is most important, and a higher CFRA is acceptable, then *TD_v1* would be optimal. If a balance between CFRA and device count is preferred, TMR applied at circuit level with the *NEM_voter_v2* would be the most suitable choice. If the most likely fault type is known beforehand, choosing the appropriate version of TD might also reduce the CFRA sufficiently, while keeping the device count low. For a higher device fault probability, the results show similar characteristics. Figure 6.27 illustrates the results for the full adder with 2% device fault probability. For stuck-off faults, there is again only one Pareto optimal point that can be identified (excluding the non-redundant circuit), which is *TD_v2*. For stuck-on faults, there are now two Pareto points, which are *TD_v1* and *QD_v1*. The same holds for the mixed fault scenario, with the two Pareto points *TD_v1* and *QD_v1*. Unlike in the earlier scenario with low device fault probability, the *NEM_voter_v2_CL* is no longer a Pareto point. This is because the higher device fault probability significantly increases its CFRA, making it no longer a viable option compared to the other two design techniques.

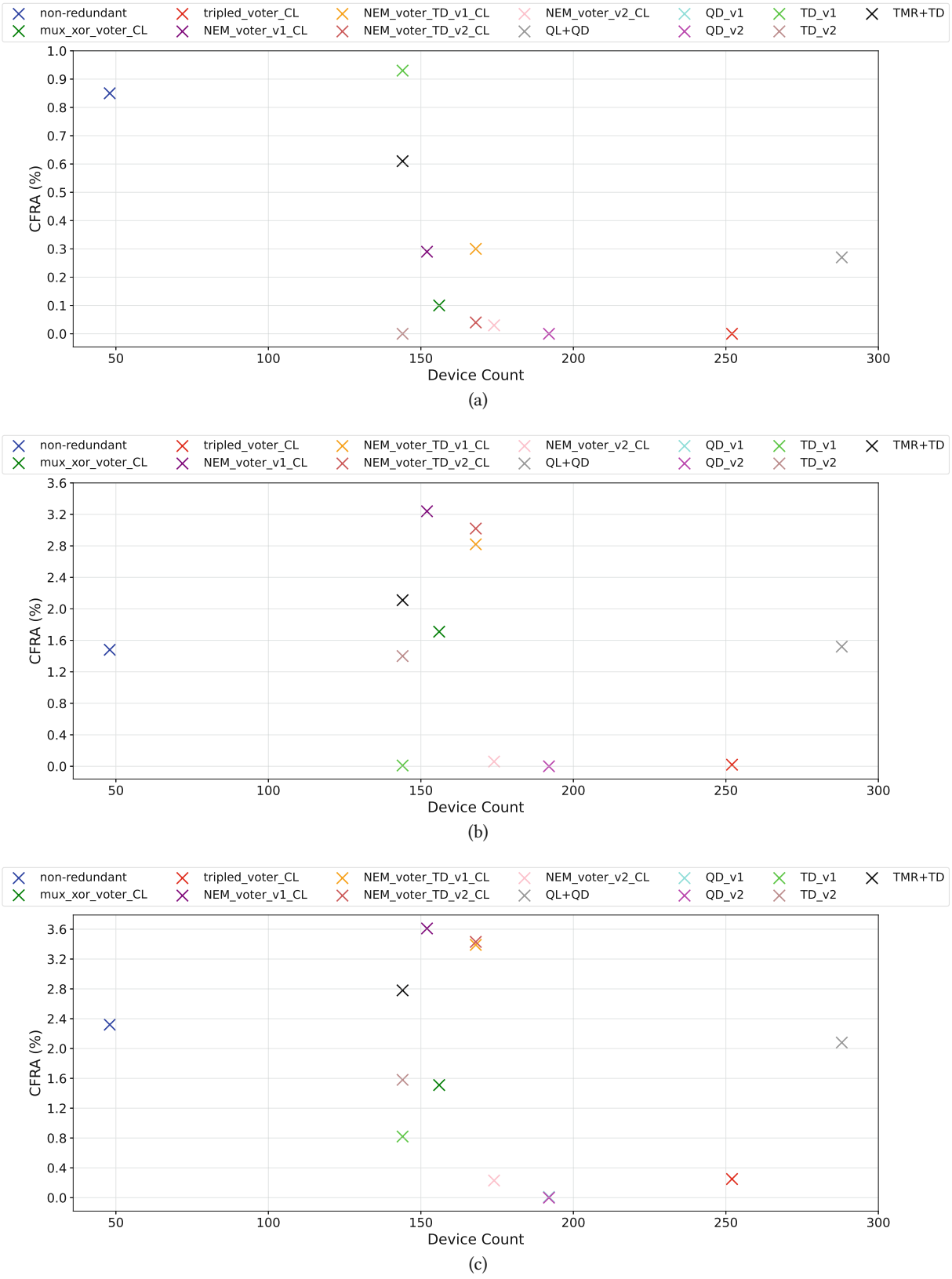


Figure 6.26: CFRA and device count for the full adder with various fault-tolerant design techniques and 0.1% device fault probability: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

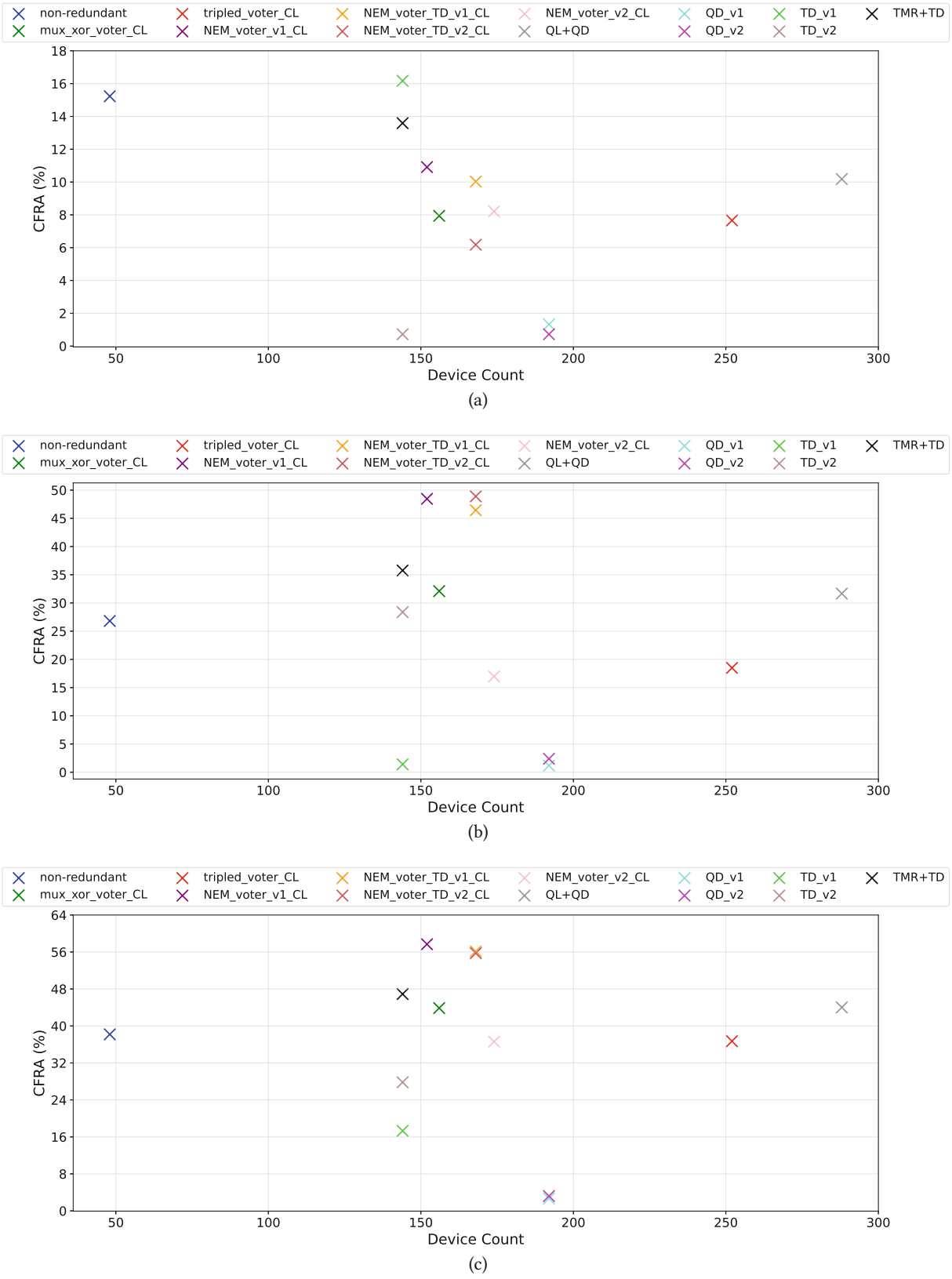


Figure 6.27: CFRA and device count for the full adder with various fault-tolerant design techniques and 2% device fault probability: (a) stuck-off, (b) stuck-on, (c) mixed faults (50%/50%).

Chapter 7

Conclusion

In this thesis, physical defects that could potentially occur in NEM devices were discussed. Their impact on the circuit was analysed at different abstraction levels, and fault models were derived to describe the behaviour of the affected components. At the gate level, *stuck-at-0*, *stuck-at-1*, and *flipping errors* have been identified as suitable fault models. At the device level, *stuck-on* and *stuck-off* models have been defined to represent faulty behaviour of the NEM devices.

Established fault-tolerant design techniques in the domains of hardware, time, and information were reviewed and evaluated for their applicability to NEM circuits. Among the design techniques discussed, passive hardware redundancy techniques were identified as suitable for improving the yield and reliability of NEM circuits. It has been shown that conventional TMR voting circuits can be implemented with fewer devices by using NEM technology. Further, new NEM voter designs have been proposed.

A simulation framework was developed to enable the simulation of NEM test circuits, with the capability to inject faults based on the defined fault models. The simulations have shown that conventional fault-tolerant design techniques are effective in improving the reliability of NEM circuits with high failure rates. The achievable fault tolerance depends on the type of fault, the number of faults (or the device fault probability), and the design technique used.

From a gate level perspective, test circuits with TMR applied to the circuit's gates have been simulated. Various conventional voter designs have been investigated for specific numbers of faulty gates in the circuit. The voter using the *priority encoder* has been identified as the best-performing design among all voters, as well as the most efficient in terms of hardware overhead for larger numbers of faulty gates.

From a device level perspective, test circuits with TMR applied to both the circuit's gates and the entire circuit were simulated, as well as the other device level design techniques. It has been shown

that applying TMR to gates introduces significant hardware overhead, leading to a higher occurrence of faults in the circuit when a constant device fault probability is assumed. This makes the application of TMR to the entire circuit a more viable choice for most voter designs. The voter in *tripled transistor design* has proven to be the best-performing voter in terms of fault tolerance, but its high device count makes it less efficient than other voters in terms of hardware overhead. The *NEM Voter Version 2* has proven to be the most efficient voter, as it offers only slightly lower performance but significantly reduces the device count compared to the voter in *tripled transistor design*.

When considering all the design techniques investigated, *Quadded Design* has proven to be the most fault-tolerant, as both versions reduce the circuit's failure rate to nearly zero across all fault types when constant device fault probabilities are assumed. When one fault type is more likely to occur, *Tripled Design* can achieve comparable performance with a lower device count by selecting the version best suited to the fault type. This means that the device count or area can be further optimized when it is known in advance which fault type is most likely to occur. Furthermore, the choice of the most appropriate design technique also depends on whether fault tolerance or device count needs to be optimized. In this respect, it has been shown that TMR, when implemented with the proposed *NEM Voter Version 2*, can be a viable option in scenarios with low device fault probability, offering a balance between fault tolerance and device count.

Lastly, it is important to note that this thesis did not investigate the delay or power consumption of the proposed design techniques. However, these factors may need to be considered when evaluating trade-offs, offering opportunities for future research on this topic.

Bibliography

[1] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, "A stochastic computational approach for accurate and efficient reliability evaluation," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1336–1350, 2014.

[2] A. Simevski, R. Kraemer, and M. Krstic, "Automated integration of fault injection into the asic design flow," in *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013, pp. 255–260.

[3] E. Dubrova, *Fault-tolerant design*. New York, NY [u.a.]: Springer, 2013.

[4] A. Mukherjee and A. S. Dhar, "Defect tolerant majority voter design using triple transistor redundancy," in *2019 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, 2019, pp. 63–68.

[5] R. Kshirsagar and R. Patrikar, "Design of a novel fault-tolerant voter circuit for tmr implementation to improve reliability in digital circuits," *Microelectronics Reliability*, vol. 49, no. 12, pp. 1573–1577, 2009, special Section on Electrostatic Discharge Reliability. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271409003114

[6] T. Ban and L. A. de Barros Naviner, "A simple fault-tolerant digital voter circuit in tmr nanoarchitectures," in *Proceedings of the 8th IEEE International NEWCAS Conference 2010*, 2010, pp. 269–272.

[7] P. Balasubramanian, K. Prasad, and N. E. Mastorakis, "A fault tolerance improved majority voter for TMR system architectures," *CoRR*, vol. abs/1605.03771, 2016. [Online]. Available: http://arxiv.org/abs/1605.03771

[8] A. Mukherjee and A. Dhar, "Triple transistor based fault tolerance for resource constrained applications," *Microelectronics Journal*, vol. 68, pp. 1–6, 10 2017.

[9] J. Han, E. Leung, L. Liu, and F. Lombardi, "A fault-tolerant technique using quadded logic and quadded transistors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 8, pp. 1562–1566, 2015.

[10] A. Mukherjee and A. S. Dhar, "Triple transistor based triple modular redundancy with embedded voter circuit," *Microelectronics Journal*, vol. 87, pp. 101–109, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269218305561

[11] B. W. Johnson, *Design and analysis of fault-tolerant digital systems*, reprint. with corr. ed., ser. Addison-Wesley series in electrical and computer engineering. Reading, Mass. [u.a.]: Addison-Wesley, 1989.

[12] H. Chen, J. Han, and F. Lombardi, "A transistor-level stochastic approach for evaluating the reliability of digital nanometric cmos circuits," in *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2011, pp. 60–67.

[13] http://web.eecs.umich.edu/ jhayes/iscas.restore/benchmark.html. Last Accessed: October 24, 2024.

[14] T.-H. Lee, S. Bhunia, and M. Mehregany, "Electromechanical computing at 500°c with silicon carbide," *Science*, vol. 329, no. 5997, pp. 1316–1318, 2010. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1192511

[15] H. R. Shea, "Effects of radiation on MEMS," in *Reliability, Packaging, Testing, and Characterization of MEMS/MOEMS and Nanodevices X*, S. Garcia-Blanco and R. Ramesham, Eds., vol. 7928, International Society for Optics and Photonics. SPIE, 2011, p. 79280E. [Online]. Available: https://doi.org/10.1117/12.876968

[16] T. Qin, S. Bleiker, S. Rana, F. Niklaus, and D. Pamunuwa, "Performance analysis of nanoelectromechanical relay-based field-programmable gate arrays," *IEEE Access*, vol. 6, pp. 15 997–16 009, Apr. 2018.

[17] H. Kam, V. Pott, R. Nathanael, J. Jeon, E. Alon, and T.-J. K. Liu, "Design and reliability of a micro-relay technology for zero-standby-power digital logic applications," in *2009 IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4.

[18] A. Peschot, C. Qian, and T.-J. Liu, "Nanoelectromechanical switches for low-power digital computing," *Micromachines*, vol. 6, pp. 1046–1065, 08 2015.

[19] W. Y. Choi and Y. J. Kim, "Three-dimensional integration of complementary metal-oxide-semiconductor-nanoelectromechanical hybrid reconfigurable circuits," *IEEE Electron Device Letters*, vol. 36, no. 9, pp. 887–889, 2015.

[20] M. Lapisa, G. Stemme, and F. Niklaus, "Wafer-level heterogeneous integration for moems, mems, and nems," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 17, no. 3, pp. 629–644, 2011.

[21] S. Chong, B. Lee, K. B. Parizi, J. Provine, S. Mitra, R. T. Howe, and H.-S. P. Wong, "Integration of nanoelectromechanical (nem) relays with silicon cmos with functional cmos-nem circuit," in *2011 International Electron Devices Meeting*, 2011, pp. 30.5.1–30.5.4.

[22] S. Rana, J. D. Reynolds, T. Y. Ling, M. S. Shamsudin, S. H. Pu, H. M. Chong, and D. Pamunuwa, "Nano-crystalline graphite for reliability improvement in mem relay contacts," *Carbon*, vol. 133, pp. 193–199, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0008622318302549

[23] J. D. Reynolds, S. Rana, E. Worsey, Q. Tang, M. K. Kulsreshath, H. M. H. Chong, and D. Pamunuwa, "Single-contact, four-terminal microelectromechanical relay for efficient digital logic," *Advanced Electronic Materials*, vol. 9, no. 1, p. 2200584, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aelm.202200584

[24] S. Rana, J. Mouro, S. Bleiker, J. Reynolds, H. Chong, F. Niklaus, and D. Pamunuwa, "Nanoelectromechanical relay without pull-in instability for high-temperature non-volatile memory," *Nature Communications*, vol. 11, 03 2020.

[25] D. Pamunuwa, E. Worsey, J. D. Reynolds, D. Seward, H. M. H. Chong, and S. Rana, "Theory, design, and characterization of nanoelectromechanical relays for stiction-based non-volatile memory," *Journal of Microelectromechanical Systems*, vol. 31, no. 2, pp. 283–291, 2022.

[26] E. Worsey, M. K. Kulsreshath, Q. Tang, and D. Pamunuwa, "Fully microelectromechanical non-volatile memory cell," in *2023 IEEE 36th International Conference on Micro Electro Mechanical Systems (MEMS)*, 2023, pp. 507–510.

[27] K. Nikolic, A. Sadek, and M. Forshaw, "Fault-tolerant techniques for nanocomputers," *NANOTECHNOLOGY Nanotechnology*, vol. 13, pp. 357–362, 06 2002.

[28] A. Mukherjee and A. S. Dhar, "New triple-transistor based defect-tolerant systems for reliable digital architectures," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 1917–1920.

[29] M. R. Rohanipoor, B. Ghavami, and M. Raji, "Design of fault tolerant digital integrated circuits based on quadded transistor logic," in *2016 Eighth International Conference on Information and Knowledge Technology (IKT)*, 2016, pp. 188–192.

[30] D. Manimekalai and P. Dixit, "Analysis of reliability for fault tolerant design in nano cmos logic circuit," *Experimental and Theoretical NANOTECHNOLOGY*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:231686652

[31] A. H. El-Maleh, B. M. Al-Hashimi, and A. Melouki, "Transistor-level based defect tolerance for reliable nanoelectronics," in *2008 IEEE/ACS International Conference on Computer Systems and Applications*, 2008, pp. 53–60.

[32] R. McWilliam, P. Schiefer, A. Purvis, and S. Khan, "Stuck-at fault resilience using redundant transistor logic gates," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2019, pp. 595–601.

[33] A. H. El-Maleh, B. M. Al-Hashimi, A. Melouki, and F. Khan, "Defect-tolerant n2-transistor structure for reliable nanoelectronic designs," *IET Comput. Digit. Tech.*, vol. 3, pp. 570–580, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:17450855

[34] A. Mukherjee and A. Dhar, "Fault tolerant architecture design using quad-gate-transistor redundancy," *IET Circuits, Devices & Systems*, vol. 9, 05 2015.

[35] S. Almukhaizim and Y. Makris, "Fault tolerant design of combinational and sequential logic based on a parity check code," in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 563–570.

[36] D. Tran, A. Virazel, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, and H.-J. Wunderlich, "A hybrid fault tolerant architecture for robustness improvement of digital circuits," in *2011 Asian Test Symposium*, 2011, pp. 136–141.

[37] D. Tran, A. Virazel, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovich, and H.-J. Wunderlich, "A new hybrid fault-tolerant architecture for digital cmos circuits and systems," *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 30, pp. 401–413, 08 2014.

[38] S. Williams. (2024) Icarus verilog. Last Accessed: October 24, 2024. [Online]. Available: http://iverilog.icarus.com

[39] T. Bybell. (2024) Gtkwave: A waveform viewer. Last Accessed: October 24, 2024. [Online]. Available: http://gtkwave.sourceforge.net

[40] H. Chen and J. Han, "Stochastic computational models for accurate reliability evaluation of logic circuits," in *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, ser. GLSVLSI '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 61–66. [Online]. Available: https://doi.org/10.1145/1785481.1785497

[41] T. Arifeen, A. Hassan, and J. A. Lee, "A fault tolerant voter for approximate triple modular redundancy," *Electronics*, vol. 8, p. 332, 03 2019.

[42] D. Rudolf, "thesis-ft-nem-design," 2024, accessed: 2024-12-21. [Online]. Available: https://github.com/DominikRudolf/thesis-ft-nem-design.git

[43] https://sportlab.usc.edu/ msabrishami/benchmarks.html. Last Accessed: October 24, 2024.