Repairing FPGA Configuration Memory Errors using Dynamic Partial Reconfiguration

Nguyen Tran Huu Nguyen

A thesis in fulfillment of the requirements for the degree of

Doctor of Philosophy

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

November 2017

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Nguyen**

First name: **Tran Huu Nguyen** Other name/s:

Abreviation for degree as given in the University calendar: **PhD**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: Repairing FPGA Configuration Memory Errors using Dynamic Partial Reconfiguration

Abstract

The configuration memory of SRAM-based *Field-Programmable Gate Arrays* (FPGAs) is susceptible to radiation-induced *Single Event Upsets* (SEUs). This has limited their adoption for space applications and led to intensive research to discover techniques for mitigating the radiation effects in such devices.

The reliability of FPGA user circuits is commonly improved by applying *Triple Modular Redundancy* (TMR), whereas configuration memory errors are corrected by reloading a golden bitstream for the design. Two approaches have emerged for doing so. The first, known as scrubbing, periodically refreshes the configuration memory of the entire device. The second makes use of dynamic partial reconfiguration to reload the configuration of an individual circuit module that has been found to be in error. This latter approach, which we refer to as *Module-based Error Recovery* (MER) holds promise for being more responsive and needing less energy than scrubbing, at the cost of greater implementation complexity.

The research work reported in this thesis aims to clarify the design, and improve the reliability of FPGA systems that employ TMR with MER. The research has involved studying and contributing to the development of several aspects of TMR-MER infrastructure, most notably, the design of reliable *Reconfiguration Control Networks* (RCNs) for conveying reconfiguration requests to a central reconfiguration controller, new reliability models for TMR-MER systems and improved scheduling techniques to check for faulty modules.

This thesis evaluates the impact of RCNs on system reliability and performance. Results show that a "hard RCN" is the most reliable despite having the highest network latency. As the order in which voters are checked for errors over the RCN has an impact on overall system reliability, this thesis then proposes a *Voter Scheduling Engine* (VSE) for dynamically prioritizing the TMR component to be checked next. This thesis proposes reliability models for TMR-MER systems suffering multiple SEUs and employing round-robin or *Variable-Rate Voter Checking* (VRVC) and proposes the use of a genetic algorithm to determine a static schedule for maximizing the system reliability. Simulation results indicate that the mean time to failure of TMR-MER systems employing VRVC is up to 400% greater than when the usual round robin is used to check components for errors.

The thesis concludes with directions for further study.

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Nguyen Tran Huu Nguyen
13 Nov, 2017

Copyright Statement

Nguyen Tran Huu Nguyen
13 Nov, 2017

Authenticity Statement

Nguyen Tran Huu Nguyen
13 Nov, 2017

Abstract

The configuration memory of SRAM-based *Field-Programmable Gate Arrays* (FPGAs) is susceptible to radiation-induced *Single Event Upsets* (SEUs). This has limited their adoption for space applications and led to intensive research to discover techniques for mitigating the radiation effects in such devices.

The reliability of FPGA user circuits is commonly improved by applying *Triple Modular Redundancy* (TMR), whereas configuration memory errors are corrected by reloading a golden bitstream for the design. Two approaches have emerged for doing so. The first, known as scrubbing, periodically refreshes the configuration memory of the entire device. The second makes use of dynamic partial reconfiguration to reload the configuration of an individual circuit module that has been found to be in error. This latter approach, which we refer to as *Module-based Error Recovery* (MER) holds promise for being more responsive and needing less energy than scrubbing, at the cost of greater implementation complexity.

The research work reported in this thesis aims to clarify the design, and improve the reliability of FPGA systems that employ TMR with MER. The research has involved studying and contributing to the development of several aspects of TMR-MER infrastructure, most notably, the design of reliable *Reconfiguration Control Networks* (RCNs) for conveying reconfiguration requests to a central reconfiguration controller, new reliability models for TMR-MER systems and improved scheduling techniques to check for faulty modules.

This thesis evaluates the impact of RCNs on system reliability and performance. Results show that a "hard RCN" is the most reliable despite having the highest network latency. As the order in which voters are checked for errors over the RCN has an impact on overall system reliability, this thesis then proposes a *Voter Scheduling Engine* (VSE) for dynamically prioritizing the TMR component to be checked next. This thesis proposes reliability models for TMR-MER systems suffering multiple SEUs and employing round-robin or *Variable-Rate Voter Checking* (VRVC) and proposes the use of a genetic algorithm to determine a static schedule for maximizing the system reliability. Simulation results indicate that the mean time to failure of TMR-MER systems employing VRVC is up to 400% greater than when the usual round robin is used to check components for errors.

The thesis concludes with directions for further study.

Acknowledgements

As with any great endeavour, this thesis would not have been possible to complete without the help and support of many others. First and foremost, I would like to express my sincere gratitude to my supervisor, Associate Professor Oliver Diessel. His supervision and expert knowledge have added considerably to my research work. He has also significantly changed my style of thinking in research, and in writing scientific papers. The inherently reader-responsible thinking of a South-East Asian guy has caused me trouble to produce clear, concise and writer-responsible papers. Therefore, with each draft I wrote, Oliver put great effort, not only into commenting on the strengths and the drawbacks, fixing tons of grammar errors and re-wording paragraphs that were not clear, but also in inspiring me in the direction of writing, and suggesting me to read relevant books such as "How to Write and Publish a Scientific Paper" by Robert Day to improve my writing skills. Moreover, I usually found a relief after many meetings we sat together. Perhaps, this was because during every meeting, Oliver's guidance and advice helped me see my research problem clearer. He also encouraged me to express thoughts in the context of an overall big picture and to emphasize WHY instead of HOW. With his support, I have also had many chances to be a demonstrator and a tutor of undergraduate courses. These helped me gain not only teaching experience, but also confidence in speaking in front of a group of students. After all, I deeply appreciate his kindness, patience and continuous guidance throughout my Ph.D. candidature.

I would like to express my gratitude to my co-supervisor, Dr. Ediz Cetin, who helped me discuss my research along with Oliver and reviewing my papers before I submitted them. I would also like to thank my colleagues: Dimitris Agiakatsikas, Zhuoran (Reimond) Zhao, Tong Wu, Lingkan (George) Gong, and Alexander Kroh. They have helped me in one way or another at different periods of time during my 3-year Ph.D. candidature. I would also like to thank my annual progress review panel, including Dr. Hui Annie Guo, Professor Andrew Dempster and Dr. Jorgen Peddersen, who gave me many suggestions on planning my thesis.

I would like to thank my wife, Nguyet Tran, for her love, sympathy and patience during this work. While I was away from home, she went through difficult times without me - the last 6 months of her pregnancy before giving birth to our daughter, raising our daughter, earning money for living and doing her Ph.D. as well, to name just a few. Despite these difficulties, she has been supportive and understanding during my Ph.D. candidature.

Both my wife and daughter have also been an inspiration for me to do better and be better. Of course, I would also like to thank my father-in-law (Nghia Tran), mother-in-law (Tuyet Nguyen), sister-in-law (Tuong Tran), and brother-in-law (Khanh Tran), who have always supported my wife in raising our daughter. This allowed me to worry less about my wife and daughter, and helped me focus on finishing my thesis.

Without a doubt, gratitude must also be expressed to my Mum (Lai Tran), Dad (Tri Nguyen) and my younger brother (Duc Nguyen). I am so grateful for the support and love they have continually expressed and shown me. My parents have also pushed me to do my best and pulled me along when I needed help. Even if they may not understand this thesis, they are proud of me as they have always been.

Finally, I would like to thank the University of New South Wales, Sydney for the Tuition Fee Scholarship (TFS) and the Research Stipend. Thanks also to the School of Computer Science and Engineering and the Australian Centre for Space Engineering Research (ACSER) for international conference travel funding.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC Analog to Digital Converter

ASIC Application–Specific Integrated Circuit

ASIP Application–Specific Instruction Set Processor

AXI Advanced eXtensible Interface

BAQ Block Adaptive Quantizer

BST Binary Search Tree

CAD Computer Aided Design

CB Conditional Block

CF Configuration Frame

CLB Configurable Logic Block

CMOS Complementary Metal Oxide Semiconductor

COTS Commercial, Off-The-Shelf

CRC Cyclic Redundancy Check

CUT Component Under Test

DCM Digital Clock Manager

DICE Dual Interlocked storage CEll

DMA Direct Memory Access

DMAC Direct Memory Access Controller

DPR Dynamic Partial Reconfiguration

DSP Digital Signal Processing

DUT Design Under Test

ECC Error Correcting Code

ESA European Space Agency

FF Flip-Flop

FIR Finite Impulse Response

FIFO First–In, First–Out

FMER Frame– and Module–based Error Recovery

FPGA Field Programmable Gate Array

FP European Union Framework Programme

FSM Finite State Machine

GA Genetic Algorithm

GCR Galactic Cosmic Ray

GEO Geostationary Earth Orbit

GPIO General Purpose Input/Output

GPS Global Positioning System

HEP High–Energy Physics

HWICAP AXI Hardware Internal Configuration Access Port

IC Integrated Circuit

ICAP Internal Configuration Access Port

INT INTerconnect column

IOB Input/Output Block

IP Intellectual Property

ITAR International Traffic in Arms Regulations

JPL Jet Propulsion Laboratory

JTAG Join Test Action Group

LEO Low Earth Orbit

LHC Large Hadron Collider

LUT Look–Up Table

MAC Multiply ACcumulator

MB MicroBlaze

MBS MicroBlaze System

MBU Multiple–Bit Upset

MCU Multiple–Cell Upset; Microcontroller Unit

MIU Multiple–Independent Upset

MTTD Mean Time To Detect(ion)

MTTF Mean Time To Fail(ure)

MTTR Mean Time To Recover(y)

NA Network Arbiter

NC Network Controller

NI Network Interface

NP Non–deterministic Polynomial time

NRE Non–Recurring Engineering

OBC On–Board Computer

OTP One–Time Programmable

PC Personal Computer

PCAP Processor Configuration Access Port

PCC Programmable Configuration Controller

PCIe PCI Express

PIP Programmable Interconnection Point

PLL Phase Lock Loop

PS Population Size

RAM Random Access Memory

RbC Readback Capture

RbV Readback Verify

RC Reconfiguration Controller

RCN Reconfiguration Control Network

Rdone Reconfiguration done

RHBD Radiation–Hardened By Design

RR Reconfiguration Request

RTVP Response Time Variability Problem

RUSH Rapid recovery from SEUs in Reconfigurable Hardware

R/W Read/Write

VKI Von Karman Institute

VRVC Variable–Rate Voter Checking

VSE Voter Scheduling Engine

SAA South Atlantic Anomaly

SAR Synthetic Aperture Radar

SD Standard Deviation

SECDED Single Error Correction, Double Error Detection

SEFI Single–Event Functional Interrupt

SEL Single Event Latch–up

SEM Soft Error Mitigation

SEP Single Event Phenomena

SET Single Event Transient

SEU Single Event Upset

SM Switch Matrix

SoC System–on–Chip

SOI Silicon–On–Insulator

SR Shift Register

SRAM Static Random Access Memory

TID Total Ionizing Dose

TMR Triple Modular Redundancy

TMR-MER Triple Modular Redundancy with Module-based configuration memory Error Recovery

TMR-Scrubbing Triple Modular Redundancy with configuration memory Scrubbing

UB Upper Bound Value

UNSW The University of New South Wales

Chapter 1

Introduction

1.1 Why SRAM-based FPGAs in Space?

Future satellite-based space missions are expected to acquire and process very high data rates from active and passive instruments [36]. Recent internal studies at NASA's Jet Propulsion Laboratory (JPL) estimate approximately 1–5 Terabytes per day of raw data (uncompressed) are expected, for example, from spectroscopy instruments [120]. Hence, there is the need to implement high performance, on-board processing systems that can handle such data rates and that, for example, are able to perform lossless data compression to reduce data volumes to those within the downlink capabilities of the spacecraft and existing ground stations. Apart from the increase in performance, the next generation of on-board processing is also required to be flexible and re-programmable in-orbit and during active service. This brings about challenging requirements for future on-board processing systems that cannot be met with space-qualified processors available today [50, 88, 160].

The implementation devices most suited to meeting such requirements are *Field Programmable Gate Arrays* (FPGAs). FPGAs offer a low *Non-Recurring Engineering* (NRE) cost alternative to *Application-Specific Integrated Circuit* (ASIC) technologies for custom hardware. Although FPGAs cannot provide the same level of performance as an ASIC, they offer at least an order of magnitude computational efficiency advantage over general-purpose processors. FPGAs also offer the flexibility to be reconfigured on-demand for in-field bug fixes, upgrades or entirely new applications.

FPGAs, in general, have demonstrated their benefits in a variety of space-based projects [106, 170]. More examples include Mars Exploration Rovers, which use Xilinx FPGAs for

motor control and landing pyrotechnics [138], and the Los Alamos National Laboratory satellite (CFEsat), which uses nine FPGAs as part of its high performance computing payload [137]. Another example is the Sentinel-2 spacecraft, a current mission of the *European Space Agency* (ESA), whose payloads are mostly FPGA based [53].

FPGAs are typically classified into three different types based on the technology used to store the configuration. These include anti-fuse, flash and SRAM-based FPGAs [173]. Until now the most commonly used implementation technology in space has been anti-fuse, which uses *One-Time Programmable* (OTP) fuses to permanently set the state of each FPGA configuration bit. The advantage of these devices is their relative immunity to radiation-induced effects and they are generally the most reliable type of FPGA to use in space applications [99]. However, the main drawback of anti-fuse FPGAs is that the configuration data cannot be changed once it is configured. This prevents the user from updating the device in-flight or from using them in reconfigurable computing applications.

The second most commonly used technology in space is based on flash memory technology. Recently, flash-memory based FPGAs, such as the ProASIC3 devices from Microsemi, have been considered for use in space-based instruments [102, 107]. Flash cells are generally immune to radiation-induced upsets and, thus the configuration memory of a flash FPGA is protected from upsets [98]. However, the use of flash FPGAs on long-term space-based missions is problematic due to their rather low immunity to *Total Ionizing Dose* (TID) effects and *Single Event Latch-ups* (SELs) [14, 103]. Although flash FPGAs can be reconfigured and offer good performance, the limitation of the number of times that they can be reconfigured renders them less desirable for use in long-term missions or in reconfigurable systems that are regularly reconfigured.

SRAM-based FPGAs use static memory cells to store the internal FPGA configuration. These static memory cells require power to store configuration state and must be programmed from external memory after the FPGA is powered up. There are two types of SRAM-based FPGAs, including radiation-hardened and *Commercial, Off-The-Shelf* (COTS) FPGAs. Compared to COTS FPGAs, radiation-hardened FPGAs are more commonly used in space because they provide protection against radiation-induced faults in the configuration and application memory. However, apart from the sale and use of these devices being restricted by the *International Traffic in Arms Regulations* (ITAR) rules [37], these devices are orders of magnitude more expensive than equivalently sized COTS FPGAs, consume more power to operate and usually lag a couple of process generations behind current COTS device technologies. Most importantly, because they are much smaller in size and cannot be reconfigured at run time, they are more limited and

less flexible than COTS devices [20].

COTS SRAM-based FPGAs offer additional flexibility over OTP FPGAs in both the development and operation of space instruments. In [126], Pingree describes the typical problems of one-time programmable FPGAs that were used in the command and telemetry interface of the key instrument on NASA's Juno spacecraft. To meet requirements, engineers had to design and configure the FPGA two years before launch. The FPGA configuration could not be modified, improved or corrected without significantly impacting on the project cost and schedule. Moreover, the 5-year trip to Jupiter also required calibration activities that had to be performed without altering the FPGA configuration. It is therefore impractical to use an OTP FPGA when the numerous reasons for potentially needing to update the FPGA configuration to better meet mission objectives are considered. This is not the case with SRAM-based FPGAs that can be updated at any stage of a mission.

Due to the benefits of COTS SRAM-based FPGAs, there is growing interest in using them for data-intensive processing applications, such as those that are prevalent in space-based systems, particularly for use in low cost, low orbit, micro- and nano-satellites, which are typified by far lower costs and shorter life-cycles than large-scale, higher orbit satellites designed for communications, scientific and defence applications. In addition to their ready availability, low cost and flexibility, these FPGAs contain abundant programmable logic resources and high-bandwidth on-chip memories that are suitable for complex, high-throughput applications, such as signal-processing. Modern SRAM-based FPGAs can also be reconfigured to allow different applications to be instantiated at different times and for specific mission objectives using the one device. Last but not least, reconfigurability can also be used for uploading new applications and for fixing bugs found in the existing design.

1.2 Scope and Objectives

1.2.1 Scope

Current state-of-the-art SRAM-based FPGAs, which can include on the order of a billion configuration memory bits, are being looked to as suitable candidates for hosting complex, high-performance, space-based and extra-terrestrial digital systems due to their low cost, low power consumption, run-time reconfigurability, and their impressive processing

performance [50, 88]. However, the designers of space-based and extra-terrestrial SRAM-based FPGA applications must consider the impact of ionizing radiation, i.e., high-energy charged particles and cosmic rays, on the device, primarily in the form of *Single Event Upsets* (SEUs) [26]. SEUs may alter the logic state of any static memory element, i.e., configuration latches, user flip-flops, internal block memory and other device-specific control registers. Since millions of configuration latches within an FPGA are programmed to implement the user functionality, an SEU in the configuration memory can adversely and dramatically affect the expected FPGA functionality. Therefore, in this thesis we mainly focus on techniques to mitigate configuration memory SEUs.

Apart from shielding, which may not be feasible in micro- and nano-satellites, the safe use of FPGAs in harsh radiation environments requires the implementation of robust SEU mitigation design techniques. Hardware redundancy, such as *Triple Modular Redundancy* (TMR), is one of the most commonly used techniques [146, 167]. TMR can mask any single design failure by voting on the result of three functionally equivalent replicas. The TMR technique can be applied at different levels of granularity. At the coarse end of the spectrum, it can be applied to the system as a whole, whereas at the fine end of the spectrum, it can be applied to each individual memory element of a system. More fine-grained application of TMR offers shorter error detection latencies together with higher area overheads due to the additional voters needed. However, TMR is unable to correct errors or eliminate erroneous values that have become trapped within a cyclic user circuit or within the configuration memory. Errors trapped in user circuitry can, though, be corrected by resetting the faulty module or by resynchronizing the module with its functionally equivalent siblings. To deal with configuration memory errors, TMR is usually combined with error recovery techniques, such as scrubbing [26,66], or *Module-based Error Recovery* (MER) [20]. We use the term TMR-Scrubbing to refer to an FPGA-based TMR system in which configuration memory errors due to SEUs are recovered by scrubbing, whereas TMR-MER is used to refer to FPGA-based TMR systems that rely on module-based error recovery to correct configuration memory errors due to SEUs.

Both TMR-Scrubbing and TMR-MER rely on *Dynamic Partial Reconfiguration* (DPR) to correct configuration memory errors. TMR-Scrubbing is typically initiated periodically and commonly involves reading back each configuration memory frame of the device, checking it for errors using in-built *Error-Correction Code* (ECC) or by comparing it to a golden reference, correcting any errors that are found and writing back any corrected frame (memory segment). In contrast, TMR-MER is commonly triggered when repeated errors are detected by the voter associated with a TMR component and involves rewriting the configuration memory for the specific module that has been found to be in error.

TMR-Scrubbing, which could be referred to as a *frame-based* recovery technique, is thus more fine-grained than TMR-MER in its corrective action, but involves reading or writing the entire configuration memory contents. On the other hand, TMR-MER is more coarse-grained than TMR-Scrubbing, in so far as the configuration memory contents of a complete module are rewritten; multiple configuration memory errors affecting the one frame/module can thus be corrected in a single action, and correction is typically faster with TMR-MER than with TMR-Scrubbing.

In the past couple of decades, more research has focused on the use of TMR-Scrubbing than on TMR-MER to improve the reliability of SRAM-based FPGA systems [146]. However, TMR-MER is being seen as offering certain advantages over TMR-Scrubbing. A significant drawback of TMR-scrubbing is that it results in unnecessary power consumption because it is invoked periodically even when no SEU has occurred [158]. Furthermore, the delay in correcting errors using TMR-scrubbing may be excessive: current state-of-the-art FPGAs, e.g., Xilinx UltraScale XCVU440, can include on the order of a billion configuration bits and the time required to read back the entire configuration memory during a scrub cycle can thus exceed 120 ms. This means that SEUs will be detected in the system after 60 ms on average, which could be too long for time- and safety-critical systems. TMR-MER aims to avoid these costs by reconfiguring just that portion of the device that is suspected of being in error and by providing low-latency error detection via the TMR voters [20]. TMR-MER, thus, aids both the system power consumption and reliability [3, 158], which are both desirable outcomes for space-based systems. In this thesis, we explore a new approach to further improve the reliability of TMR-MER systems. We thereby aim to further understand the relative merits of TMR-MER and TMR-Scrubbing.

1.2.2 Objectives

This thesis aims to provide solutions to further improve the reliability of TMR-MER systems. The thesis has three objectives.

Both TMR-Scrubbing and TMR-MER utilize a controller to operate, but TMR-MER also requires a means of relaying *Reconfiguration Requests* (RRs) from the voters in the system to a central *Reconfiguration Controller* (RC) [4]. A star-, bus-, or ring-based *Reconfiguration Control Network* (RCN) is often employed to perform this function [4]. Another alternative is to use the in-built FPGA configuration infrastructure to access distributed status registers containing the RRs [4]. The performance and reliability of the RCN are important for a few reasons. Firstly, the latency of the RCN has a direct impact

on the *Mean Time To Recover* (MTTR) from errors in the system, and the sooner the module is recovered, the lower the likelihood that the protection provided by TMR will fail. On the other hand, the RCN is often implemented as a non-redundant component in the system, whereby it introduces a single point of failure that can greatly compromise system reliability. Therefore, the first objective of this thesis is to focus our attention on the design of the RCN for high performance with low resource utilization so as to reduce its sensitivity to SEUs. We investigate possible network topologies for implementing an RCN and compare their area, performance, and upset vulnerability with a view to establishing the best solution for a given operating environment.

Irrespective of the RCN topology and technique employed, the voters, which trigger a module-based reconfiguration by raising a request, cannot be checked in parallel. They must be checked sequentially. Conventionally, the voters are checked in round-robin order [4, 20, 152, 163]. Round robin is appropriate when the system contains similarly sized TMR components, which are equally likely to suffer SEUs. However, when TMR components vary in size, such as in the RUSH (Rapid recovery from SEUs in Reconfigurable Hardware) payload [30], the order in which the voters of TMR components are checked has an inevitable impact on overall system reliability. Intuitively, larger components are more susceptible to configuration memory SEUs, and thus should be checked more frequently than smaller ones. Therefore, the second objective of this thesis is to propose an on-chip *Voter Scheduling Engine* (VSE) to help the RC dynamically adjust the order in which RRs from TMR voters are checked for module errors based on the likelihood of the next checked component being in error [113]. The approach was implemented based on the idea that the RRs from the more vulnerable components, i.e., those comprising a greater number of essential bits [89], are checked more frequently than the less vulnerable ones.

Consequently, the VSE work prompted us to investigate whether a static voter checking schedule could be found to enhance TMR-MER system reliability beyond that possible with the dynamic voter checking method. Therefore, the third objective of this thesis is to explain our approach to developing a static schedule for checking voters that maximizes the reliability of TMR-MER systems. It has also been noted that while TMR-MER is generally effective for mitigating SEUs affecting the configuration memory [4], it is not well suited to protecting systems against multiple coincident SEUs that affect multiple modules of a TMR component and that thereby defeat the protection afforded by redundancy. Therefore, to fulfil the third objective, we investigate the reliability of TMR-MER systems consisting of multiple triplicated components operating in harsh radiation environments, such as in geosynchronous orbit during solar flares and in high-energy physics laboratories like the Large Hadron Collider, where multiple coincident SEUs are more likely to occur

[123]. Our main interest in this objective is in determining the impact on overall system reliability of varying the order and rate at which the voters of TMR components are checked for RRs.

To achieve the objectives mentioned above, and to demonstrate their applicability in general, we have conducted experiments both on synthetic systems comprising a variety of TMR components, and on the RUSH micro-satellite payload, which contains 9 differently sized components. The RUSH payload is described in detail in the Appendix of this thesis.

1.3 Thesis Contributions

The work described in this thesis aims to improve the reliability of TMR-MER systems. The key contributions of this thesis are:

1.3.1 Reconfiguration Control Network

We compare four RCNs with respect to reliability, latency, scalability and power consumption. Fault injection experiments were conducted to evaluate the impact of each RCN on system reliability. We demonstrate that the hard network, which uses the *Internal Configuration Access Port* (ICAP) of the FPGA to read the voter state, achieves the highest reliability in a case study that is implemented on the RUSH (Rapid recovery from SEUs in Reconfigurable Hardware) payload [30]. We also show that the *Mean Time To Detect* (MTTD) configuration memory errors is greatest for the ICAP-based approach due to the relatively large latency involved in retrieving user state this way, but we demonstrate an effective optimization that significantly narrows the gap in MTTD between this hard approach and the soft RCNs. Finally, we assess the reliability of a real system employing module-based recovery relative to the same system using blind scrubbing. We have determined that scrub-based error recovery results in higher reliability unless the RCN is itself triplicated and repaired when its configuration becomes corrupted.

1.3.2 Voter Scheduling Engine

We propose and evaluate a *Voter Scheduling Engine* (VSE) that dynamically prioritizes and manages the voter checks in an FPGA-based TMR-MER system. The proposed VSE is based on the idea that the currently most vulnerable TMR component needs to be

checked next. Moreover, we incorporate the VSE into an ICAP-based RCN to readback configuration frames that contain the health status of the system's TMR components [4, 163].

Furthermore, we assess and compare the reliabilities of a TMR system with MER, whereby the TMR voter states are checked in a round-robin fashion, with that of the same system implementing VSE. We demonstrate that TMR systems that utilize the VSE to determine which component to check next are generally more reliable than those using a round-robin order for checking component voters. This is especially the case when the period between two successive checks is increased, e.g., when there is an increased number of TMR components to check, or when the check frequency is reduced for the purpose of saving energy. Results obtained using four different radiation conditions show that the failure probability of the TMR system incorporating VSE is up to 50% lower than that of the same system using round-robin voter checking during a simulated 30-day mission in *Geostationary Equatorial Orbit* (GEO) and during a simulated 10-year mission in *Low Earth Orbit* (LEO).

1.3.3 Variable-Rate Voter Checking

In the second objective, we developed methods for identifying the next component to check at run time based on the likelihood that the component has failed since the last check [113]. In contrast, the third objective is to report on an off-line approach to determining a fixed voter checking sequence that maximizes system reliability. Our contributions are:

- To derive reliability models of TMR-MER systems that comprise finitely many TMR components whose voters are checked in round-robin order and at a variable rate. We refer to such a schedule as *Variable-Rate Voter Checking* (VRVC). Previous work has primarily focused on the effects of SEUs on SRAM FPGA-based systems while our analysis considers the impact of multiple consecutive events, which is an important consideration in providing a more accurate analysis of the system reliability in high radiation environments.
- To propose a *Genetic Algorithm* (GA) for finding the optimal rate at which to check all components so as to maximize the *Mean Time To Failure* (MTTF) and the reliability of TMR-MER systems.
- To show that power consumed checking for errors can be reduced by reducing the checking frequency. In this case, VRVC is capable of ensuring a higher system

reliability than round robin or VSE.

- To demonstrate that MTTD is reduced by 44% and 30% on average when VRVC is used instead of round robin and VSE, respectively.

1.4 Publications

1.4.1 Journal Papers

- **N. T. H. Nguyen**, D. Agiakatsikas, Z. Zhao, T.Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration Control Networks for TMR Systems with Module-based Recovery," *Microprocessors and Microsystems*, 2017 (under review) [112].

- **N. T. H. Nguyen**, E. Cetin, O. Diessel, "Improving Reliability of FPGA-based Systems by Scheduling Checks for Configuration Memory Errors," *IEEE Transactions on Aerospace and Electronic Systems*, 2017 (under review) [114].

1.4.2 Conference Papers

- **N. T. H. Nguyen**, E. Cetin, and O. Diessel, "Scheduling Voter Checks to Detect Configuration Memory Errors in FPGA-based TMR Systems," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2017 [116].

- **N. T. H. Nguyen**, E. Cetin, and O. Diessel, "Scheduling considerations for voter checking in TMR-MER systems," in *IEEE International Symposium on Field Programmable Custom Computing Machines (FCCM)*, April 2017, pp. 30-30 [111].

- **N. T. H. Nguyen**, E. Cetin, and O. Diessel, "Dynamic scheduling of voter checks in FPGA-based TMR systems," in *International Conference on Field Programmable Technology (FPT)*, Dec 2016, pp. 169–172 [113].

- D. Agiakatsikas, **N. T. H. Nguyen**, Z. Zhao, T.Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration Control Networks for TMR Systems with Module-based Recovery," in *IEEE International Symposium on Field Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 88–91 [4].

- L. Gong, T. Wu, **N. T. H. Nguyen**, D. Agiakatsikas, Z. Zhao, E. Cetin, and O. Diessel, "A Programmable Configuration Controller for Fault-tolerant Applications,"

in *IEEE International Conference on Field Programmable Technology (FPT)*, Dec 2016, pp. 117-124 [58].

- L. Gong, A. Kroh, D. Agiakatsikas, **N. T. H. Nguyen**, E. Cetin, and O. Diessel, "Reliable SEU monitoring and recovery using a programmable configuration controller," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep 2017 [57].

1.4.3 Technical Reports

- **N. T. H. Nguyen**, E. Cetin, and O. Diessel, "Scheduling Considerations for Voter Checking in FPGA-based TMR Systems," *School of Computer Science and Engineering, UNSW Sydney*, Tech. Rep., 05 2017 [115].

The work described in this thesis is published in papers [111, 113, 116], papers under-review [112, 114], and a technical report [115], as well as partly in the published papers [4, 57, 58]. The contributions of this author to the papers [4, 57, 58] in which he is not the lead author is as follows. In [4], this author: (1) surveyed the RCN designs available in the literature, (2) helped propose optimized RCN architectures for experimental evaluation, helped implement the synthetic layout and RUSH layout described in the Appendix and obtain implementation results, including design logic and routing utilization, latency, power consumption estimates and numbers of essential bits that were reported from the implementation tool for experimental evaluations; (3) proposed the fault injection procedure and contributed to the proposal of the "RePin" approach for generating test vectors during fault injection experiments and assisted in implementing the fault injection campaign; and (4) analyzed the system reliability. Paper [4] is described in part in Section 3.3.

In [58], this author reviewed the literature on reconfiguration controllers for systems that support dynamic partial reconfiguration and implemented the firmware for various fault-tolerant methods such as TMR-Scrubbing, TMR-MER and Frame/Module-based Error Recovery (FMER) [3] running on the *Programmable Configurable Controller* (PCC). In [57], this author was only involved in reviewing the literature on reliable reconfiguration controllers for fault-tolerant systems. The literature reviews of both papers [57, 58] are included in Section 3.2 of this thesis.

1.5 Thesis Outline

This thesis is organized as follows. After reviewing the radiation effects on SRAM-based FPGAs and the sources of radiation, Chapter 2 details SRAM-based FPGA architectures and analyses radiation-induced effects on each memory type within these devices. Next, it provides a survey of the techniques used to mitigate these radiation-induced effects with the aim of improving FPGA-based system reliability and provides an overview of two techniques that have been widely used to mitigate configuration memory errors in the literature. This chapter also describes related work on novel techniques that have been proposed to further improve system reliability. Finally, practical and theoretical methods to validate the effectiveness of mitigation techniques for radiation-induced effects are given in this chapter.

Chapter 3 describes the architecture and operation of reliable TMR-MER systems including the design of suitable voters, a reliable RC and, most importantly, the study of RCNs available in the literature.

Chapter 4 describes our proposed approach for dynamically scheduling voter checks to enhance TMR-MER system reliability.

Chapter 5 presents a solution to the question that Chapter 4 raises as to whether it is possible to find a static voter checking schedule that maximizes system reliability.

The last chapter concludes the thesis and discusses directions for future work in enhancing system reliability estimations.

The thesis includes one Appendix, which describes an implementation of a TMR-MER system that has been deployed in a CubeSat, and that has been used as a case study in obtaining results for Chapters 3 – 5.

Chapter 2

Background and Related Work

This chapter provides background related to SRAM-based FPGAs that are used in radiation environments. The chapter begins with a general discussion on the possible effects of radiation on SRAM-based FPGAs (Section 2.1), which is followed by a description of common radiation sources that digital circuits are exposed to in the field (Section 2.2). Section 2.3 describes the heterogeneous architecture of modern SRAM-based FPGAs that include different memory types, e.g., configuration memory and user memory, and describes the consequences if these memories suffer errors caused by radiation-induced effects, primarily *Single Event Upsets* (SEUs). Section 2.4 provides an overview of the techniques used to mitigate against SEUs in each type of memory within an SRAM-based FPGA. For example, the use of redundancy, such as *Duplication with Compare* (DWC) and *Triple Modular Redundancy* (TMR), can be combined with error recovery techniques to mitigate SEUs. In Section 2.5, we focus on two upset mitigation techniques — TMR with configuration memory scrubbing (TMR-Scrubbing) and TMR with Module-based Error Recovery (TMR-MER), both of which are proven to be able to mitigate configuration memory errors in SRAM-based FPGAs with a view to improving overall system availability and reliability. Section 2.6 reviews the previous work on utilizing novel techniques to further improve system availability and reliability. Sections 2.7 and 2.8 provide both practical and theoretical methods that have been widely used to assess the effectiveness of SEU mitigation techniques in FPGAs. The last section summarizes the chapter.

2.1 Radiation Effects on SRAM-based FPGAs

Radiation is the emission or transmission of energy in the form of atomic or subatomic particles moving at high speeds (usually greater than 1% of the speed of light) [171]. Particularly in space, radiation is generated by particles emitted from a variety of sources both within and beyond our solar system. When high-energy particles such as electrons, heavy ions or protons travel through a semiconductor circuit, they may not only cause degradation, but may also cause negative effects on the operational circuit such as introducing malfunctions or even permanently damaging the semiconductor system [118, 119].

Modern *System-On-Chip* (SOC) *integrated circuits* (ICs) are increasingly using *Static Random Access Memory* (SRAM) to provide high-speed, on-chip memory such as for registers and cache. This is particularly so for conventional FPGAs, which also use SRAM to store circuit configuration (look-up table and routing settings) as well as for block RAM storage. For the past two decades, such SRAM-based FPGA devices have been investigated by many researchers seeking to improve the suitability of these devices in radiation environments, particular in space [51, 83, 146]. It has certainly been established that SRAM-based FPGAs are very sensitive to the radiation environments — particularly radiation-induced effects.

Figure 2.1 presents an overview of common radiation effects on digital circuits, including SRAM-based FPGAs. High-energy ionizing radiation has two types of effects that include a long-term and damaging effect known as *Total Ionizing Dose* (TID), and immediate effects known as *Single-Event Effects* (SEEs). When investigating the effects of radiation on SRAM-based FPGAs, both TID and SEEs must be considered.

TID is defined as the total amount of radiation dose that a device can tolerate before failing to meet the electrical parameters specified for the device. In *Complementary Metal Oxide Semiconductor* (CMOS) devices, TID generates electron-hole pairs within the gate oxide from the total ionizing energy deposited by photons or charged particles over time. TID is a cumulative effect that leads to the degradation of electrical parameters, such as decreasing the threshold voltage, increasing the leakage current and modifying the timing of MOS transistors [14, 122, 189]. For long-term missions, or for missions with extreme ionizing radiation environments, the accumulation of ionizing radiation ultimately causes the device to fail [49]. Many factors may affect the TID absorbed by an FPGA device such as orbit/location, length of mission, placement in the satellite, and the thickness of shielding around the satellite. FPGAs may be exposed to 1 – 5 krad(Si) per year for short missions in near Earth orbits, whereas these numbers are about 10 – 100 krad(Si) per

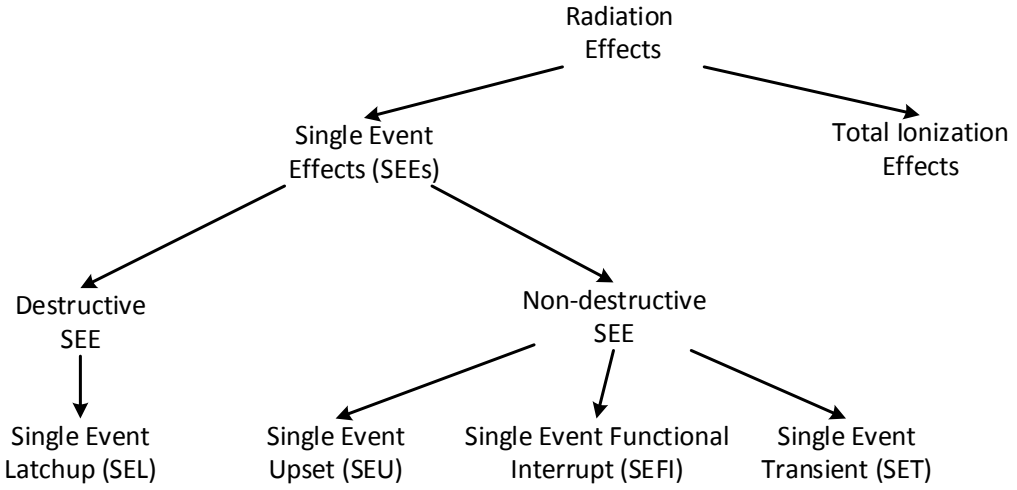


Figure 2.1: Common radiation effects on digital circuits [146]

week for missions to Jupiter [134]. FPGA devices vary in their TID limits. For example, COTS Xilinx Virtex 6 FPGAs have a TID limit of 380 krad(Si), whereas space-grade Xilinx Virtex-5QV devices can withstand up to 1,000 krad(Si) [134].

On the other hand, SEEs are electrical disturbances caused by the direct ionization of a silicon lattice by an energetic charged subatomic particle. Such ionization may lead to destructive (e.g., *Single Event Latchup* (SEL)) or non-destructive events, as can be seen in Figure 2.1. Non-destructive events, which are also called "soft errors", include non-stable events, such as *Single Event Transients* (SETs), or stable events, such as SEUs and *Single Event Functional Interrupts* (SEFIs). SEEs are random and happen according to a probability related to energy level, flux, and cell susceptibility. A brief summary of these SEEs is provided below [44, 69]:

- Single-event latch-ups (SELs): An SEL is an abnormal, high-current state in a device caused by the passage of a single energetic particle through sensitive regions of the device structure, resulting in the loss of device functionality. In many cases the current is high enough to cause permanent damage to the device. If the device is not permanently damaged, power cycling of the device (off and back on again) is necessary to restore normal operations. For example, a SEL may occur in a CMOS device when a single particle triggers shorting of power to ground via a parasitic p-n-p-n thyristor structure. Any device considered for use in high radiation environments must be tested for SELs [79].

- Single-event upsets (SEUs): An SEU is a state change (or bit-flip) of a single data bit or memory cell caused by ionizing particles [69]. In terrestrial applications, the two ionizing radiation sources of concern are alpha particles emitted from package impurities and high-energy neutrons caused by the interaction of cosmic rays with the Earth's atmosphere. When an ionizing particle passes through the various material layers of a semiconductor device, it creates an ionization path with free electrons and holes such that charge can be transferred from one node to the other. This may result in a toggling of the state of the memory cell, i.e., changing a logic "1" to a logic "0" or a logic "0" to a logic "1". Unlike a SEL, the SEU itself is not considered permanently damaging to the device [118, 119].

- Single-event transients (SETs): A single event effect may cause one or more transient voltage pulses, i.e., glitches, to propagate through the circuit. We call such events single-event transients. Since the output glitches do not change circuit "state", as in a memory SEU, one differentiates between SETs and SEUs. However, if the temporary glitch propagates through digital circuits and is latched, it may appear as an SEU. Like SEUs, SETs do not cause any permanent damage within the device [69].

- Single-event functional interrupts (SEFIs): A SEFI is an SEE that results in the interference of the normal operation of a complex digital circuit. SEFIs are typically used to indicate failure in support circuits, such as loss of configuration capability, power on reset, JTAG functionality, a region of configuration memory, or the entire configuration [8, 190].

Another class of non-destructive SEE, not shown in Figure 2.1, is referred to as Multiple-Bit Upsets (MBUs). An MBU is defined as any event that causes more than one SEU from a single charged particle [131]. MBUs affect multiple configuration bits simultaneously. The techniques presented in this thesis deal with SEUs. MBUs, which in current process technology devices occur about 5% of the time, will also be dealt with using our techniques when they affect the one module of a TMR component. However, when MBUs affect multiple modules of a TMR component, stronger techniques for detecting the errors in the first place are needed.

2.2 Radiation Environments

Radiation is prevalent in many environments. In this section, we review several important radiation environments where FPGAs are particularly useful, including space environ-

ments, terrestrial environments and high-energy physics environments.

2.2.1 Space Environments

Digital circuits used in space are subjected to many environmental threats that can degrade many circuit components. These threats include vacuum, solar ultraviolet radiation, charged particle (ionizing) radiation, plasma and surface charging and arcing, temperature extremes, vibrations, and so on. Perhaps, the most dangerous threat for digital circuits is the pervasive influence of charged particle radiation [18] since those particles causes single event effects as described in the previous section.

Three primary sources of these particles involve the solar wind and flares, *Galactic Cosmic Rays* (GCRs) and the Van Allen radiation belts [44, 69]. High-energy charged particles, which include protons and electrons are emitted by the sun as part of the solar wind. During intense solar flares, which may last from a few hours to several days, the number of particles emitted can dramatically increase by several orders of magnitude compared to those present under normal conditions. GCRs must also be considered as they are particles similar to those found in the solar wind and in solar flares, but they originate outside the solar system. In many cases, GCRs are much more massive and energetic than particles of solar origin. The Van Allen radiation belts are the third threat and involve charged particles trapped in the Earth's magnetic field. The Van Allen radiation belts extend from 100 km to 65,000 km beyond the Earth's surface and consist mainly of electrons up to a few MeV and protons of up to several hundred MeV energy [44]. Due to the Earth's asymmetric magnetic field, a region in the Atlantic near Argentina and Brazil, known as the *South Atlantic Anomaly* (SAA), has a relatively high concentration of electrons. The SAA is known to cause problems such as: single event upsets in digital circuits [108].

The radiation environment in space is well characterized and varies considerably based on the altitude, inclination, and eccentricity of an orbit. It is also heavily dependent on the transient solar weather and the 11-year solar cycle [121]. One of the challenges of using FPGAs in space is estimating the upset rate of the devices and handling infrequent but harsh space weather events, such as solar flares.

2.2.2 Terrestrial Environments

At terrestrial, Earth-based altitudes, the predominant sources of radiation include both cosmic-ray radiation and terrestrial sources [74]. As mentioned in the previous section,

cosmic radiation may come from sources outside the solar system and is dominated by high- and low-energy neutron-induced reactions between cosmic radiation and atoms in the atmosphere. Terrestrial sources of radiation include naturally occurring materials in the Earth, such as soil, rocks, water and the air. Most of them are relatively low in energy and thus have little impact on electronic systems. However, there is an exception, namely, alpha particles emitted from radioactive impurities in electronic device packaging and chip materials, which can cause soft errors [74]. Fortunately, these can be eliminated using improved manufacturing techniques. For example, the use of a *Dual Interlocked storage CEll* (DICE) design [22] to fabricate the memory cells of SRAM-based FPGAs can improve the soft-error resistance by nearly 1,000 fold [187], while only doubling area, energy consumption and gate delay compared with standard SRAM cells [92].

Problems with terrestrial-based neutron radiation from cosmic rays have become more commonplace. While the incidence of radiation-induced effects in the terrestrial environment is lower than in space, physics, system design and system locations have combined to make systems increasingly vulnerable to terrestrial radiation environments. Therefore, digital circuits operating in the terrestrial radiation environment can also experience adverse radiation effects, leading to their malfunctioning [96]. The impact of these upsets must be considered with regard to high-reliability applications, such as medical devices or systems that contain a large number of FPGAs, such as scientific instruments, data centres and supercomputers [130].

2.2.3 High-Energy Physics Experiments

High-Energy Physics (HEPs) is the branch of physics that studies the properties and interactions of the fundamental particles of nature that constitute matter and radiation. HEP experiments use particle accelerators, such as the *Large Hadron Collider* (LHC), to accelerate charged particles almost to the speed of light before causing them to collide head on. The collisions generate a number of by-products that are studied to learn more about the subatomic structure of matter and the fundamental laws of nature.

High-energy particles collide within HEP experiments to create a radiation environment. The intensity of the radiation environment depends on the type of experiment and on the location within the experiment. Basically, the closer to the particle collision, the higher the radiation level.

In the past, due to their sensitivity to radiation, SRAM-based FPGAs were rarely used in HEP, especially in the inner detector, where the radiation field is very high, and may even

be higher than in some space-based environments [33]. However, in environments with moderate radiation fields, SRAM FPGAs based on 28 nm feature sizes and on *Silicon On Insulator* (SOI) technology can tolerate the radiation present when used with appropriate SEU mitigation methods [33]. SRAM-based FPGAs have thus been used to track sub-atomic particles resulting from collisions because of their fast and dense resources [59, 62].

2.3 SRAM FPGA Architectural Vulnerabilities

In the early days, FPGAs were mainly used to implement the interfaces between devices and sub-systems. However, modern FPGAs enable far more complex operations as they are very sophisticated devices containing a wide variety of heterogeneous resources. These include

- *Configurable Logic Blocks* (CLBs), including programmable *look-up tables* (LUTs) to implement combinational logic, and user flip-flops for storing synchronous logic state;
- *Digital Signal Processing* (DSP) slices provide advanced high-speed arithmetic and comparison functions, including multiplication and accumulation;
- *Block Memory* (BRAM) modules provide large storage capacity in the form of true dual port RAM;
- *Digital Clock Manager* (DCM) blocks provide clock frequency synthesis and de-skew;
- *Phase Lock Loop* (PLL) blocks provide clock distribution delay compensation, clock multiplication/division, coarse/fine-grained clock phase shifting, and input clock jitter filtering;
- Bidirectional *Input/Output Blocks* (IOBs);
- High-speed serial transceivers supporting full-duplex, clock-and-data recovery;
- *PCI Express* (PCIe) endpoints and root port designs, which implement a general purpose serial interconnect that can be used for peripheral device interconnects, chip-to-chip interfaces and bridge to many other protocol standards;
- Hard processors, embedded for processing solutions spanning high performance, low power and very low cost; and

- User configurable *Analog to Digital Converter* (ADC) interfaces with on-chip thermal and supply sensors.

The sensitivity of these resources to radiation-induced effects varies. An understanding of these effects can only be gained through direct radiation testing [153, 154].

Numerous radiation tests on the Xilinx Kintex-7 XC7K325T FPGA have proven the feasibility of deploying commercial SRAM-based FPGAs in radiation environments [17, 68, 90]. Although COTS SRAM-based FPGAs cannot provide high-radiation immunity compared with other FPGA types, they are surprisingly resistant to TID and SELs, especially in *Low Earth Orbit* (LEO), and if proven to be sufficiently reliable, may even have applications in geosynchronous satellites. Hiemstra *et al.* reported that no change in post-irradiation supply current was observed after a Kintex-7 FPGA received an equivalent total dose of 17 krad (SI), which guarantees sufficient device lifetime in space for a short period at moderate radiation levels [68]. Moreover, Lee *et al.* predicted the latch-up event rate to be 9.2×10^{-5} per device day in *Geostationary Earth Orbit* (GEO), and those observed latch-ups are "non-destructive", which can be tolerated by discharging the device through the auxiliary power-rail [90]. More importantly, heavy ions, which could cause latch-ups on commercial SRAM-based FPGAs, are scarce in LEO [90]. In this section, we therefore focus on the effects of soft-errors, primarily SEUs, on COTS SRAM-based FPGAs.

From a fault-tolerance perspective, SRAM-based FPGAs can be thought of as devices that consist of two layers, namely a configuration layer and an application layer. The former comprises the configuration memory and the logic needed to access this memory, while the latter is composed of memory resources that are used by the implemented user designs to store application state. User designs are configured by means of the configuration memory bit settings. Therefore, memory elements in an SRAM-based FPGA device can be classified into two groups: configuration and user memory bits. The configuration memory bits are used to specify the particular circuit mapped onto the FPGA, whereas the user memory bits (e.g., flip-flops and the contents of distributed and block RAM elements) hold the current state of the circuit. Unless the user design is dynamically reconfigurable, the contents of the configuration memory bits should remain unchanged during operation, while the contents of the user memory bits may change on any clock cycle. Moreover, all FPGAs also contain internal state to manage the internal operation and configuration of the FPGA.

This section discusses the effects of radiation-induced SEUs on the configuration memory, block RAMs, user flip-flops, and internal proprietary state of an SRAM-based FPGA.

To aid this discussion, we refer to a Xilinx Artix-7 XC7A200T FPGA to illustrate these issues. Table 2.1 reports the most important user-accessible internal state of the Artix-7 XC7A200T FPGA [184]. This device contains over 91 million bits that are susceptible to radiation-induced upsets. The impact of SEUs within each of these memory types will be described below.

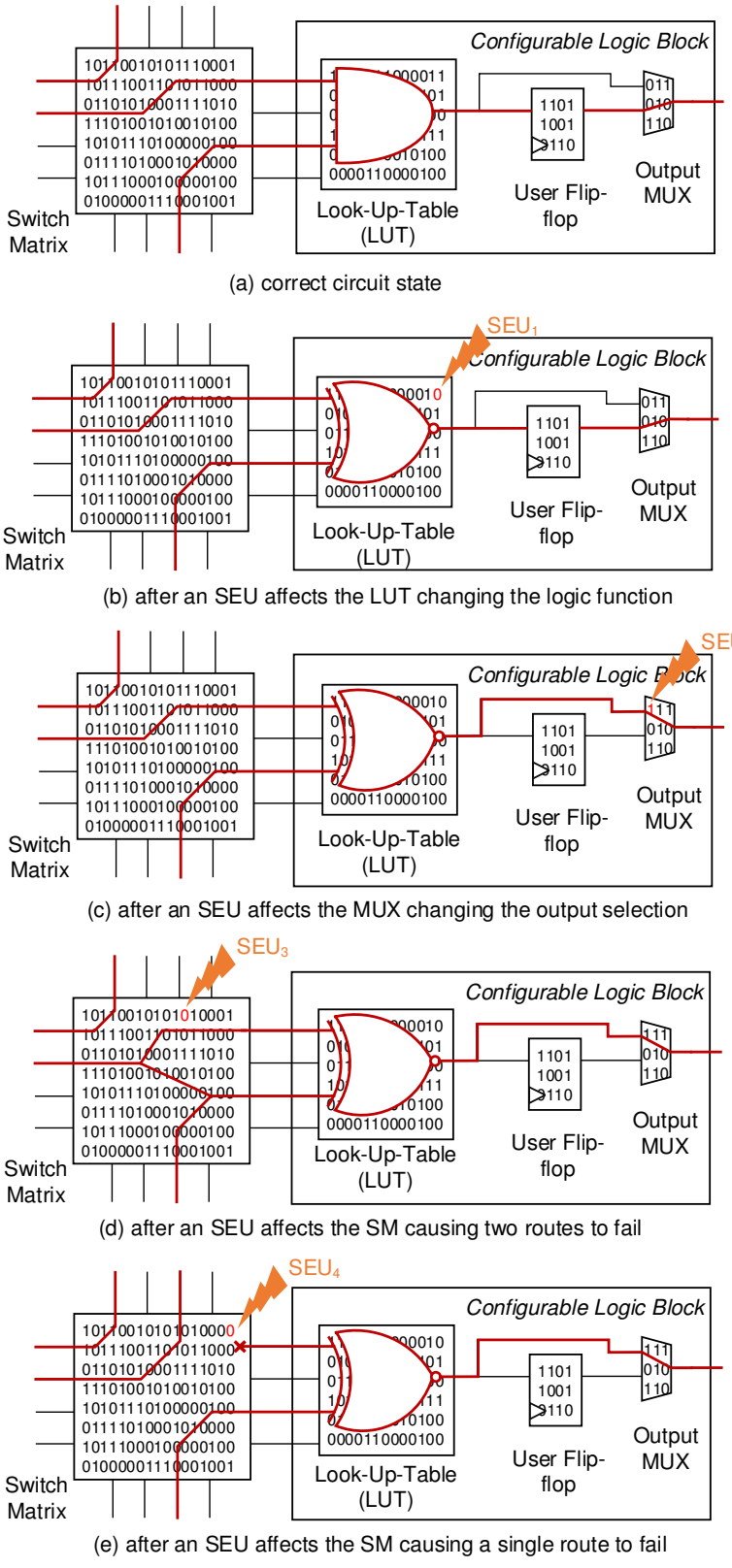
Table 2.1: Memory bits within the Artix-7 XC7A200T Xilinx FPGA

Memory type	**Number of bits**	**Percentage distribution**
Configuration	77,845,216	85.0
Block RAM	13,455,360	14.7
User Flip-flops	269,200	0.3
Total	91,569,776	100.0

2.3.1 Configuration Memory

As can be seen in Table 2.1, the device configuration memory accounts for 85% of the memory latches in the Artix-7 XC7A200T FPGA. This implies that the most susceptible memory type in this device, as in all SRAM FPGAs, is the configuration memory. The configuration memory bits include all LUT values, other internal CLB settings, IOB configuration settings, block RAM control settings, DSP slice settings, clock manager settings, and all *programmable interconnect points* (PIPs). They are used to specify the particular circuit mapped onto the FPGA. Moreover, SRAM-based FPGAs use static memory cells for configuration storage, allowing these devices to be reprogrammed by loading a new configuration into the memory: a different logic function and/or routing organization can be implemented by reconfiguring, or changing, the configuration memory settings.

Since static memory cells are susceptible to radiation-induced SEUs, if SEUs occur within the configuration memory, they may lead to a malfunction of the user circuit design in different ways. Figure 2.2 illustrates four typical circumstances in which a circuit fails due to SEUs occurring in the configuration memory, where Figure 2.2(a) depicts the relationship between the circuit configured on the device and the configuration memory. The data in the switch matrix, LUTs, user flip-flops and the output MUX form a two-input AND gate with a D-type flip-flop before exiting the CLB. In Figure 2.2(b), the upset directly changes the LUT content, which in this case changes the implemented function to an XNOR gate. In Figure 2.2(c), the upset disables the output latch, which changes the circuit mode from synchronous to asynchronous. SEUs in the *Switch Matrix* (SM) may cause two possible effects. As depicted in Figures 2.2(d) and (e), a configuration memory



(a) correct circuit state

(b) after an SEU affects the LUT changing the logic function

(c) after an SEU affects the MUX changing the output selection

(d) after an SEU affects the SM causing two routes to fail

(e) after an SEU affects the SM causing a single route to fail

Figure 2.2: Examples of SEU effects on different types of logic cells [11, 61, 136].

upset may bridge two routes together that affects two routing nets, or may disconnect an input to the CLB [11].

However, not all upsets to the configuration memory will cause the design to fail in its intended operations. In any circuit implementation, only a small portion of configuration memory bits are associated with the circuitry of the design — these are referred to as *essential* bits [89], and a large number of configuration memory bits are unused for specific designs — these are referred to as non-essential bits. SEUs presenting in non-essential bits will not cause any of the above-mentioned failure modes [26]. Note that upsets corrupting essential bits do not necessarily affect circuit operation. The bits that do influence circuit functionality when corrupted are referred to as *sensitive* or *critical* bits, which form a subset of the essential bits. The actual number of sensitive bits varies from design to design. Fault injection is frequently performed to determine the number of sensitive bits in a design [61, 76] (see Section 2.7).

2.3.2 Block RAMs

In the early days, *Block RAMs* (BRAMs) accounted for a small portion of memory cells within an FPGA device [11]. However, modern FPGAs embed more BRAMs to support a variety of operations and computations. BRAMs are distributed across the device providing a large amount of internal memory bandwidth to support high-performance computing and memory buffering. As can be seen in Table 2.1, BRAMs account for 14.7% of on-chip memory, which is the second largest memory component within the example FPGA.

BRAMs, which are implemented using SRAM, are also susceptible to radiation-induced upsets. Upsets in BRAMs may cause application errors. The impact of such data errors on system behaviour depends on how the data are used in the system. Like configuration memory, not all BRAM cells are used in a specific application. If upsets occur in unused memory words, they will not impact on the system. However, if upsets occur within the used memory space, errors can propagate throughout the system and cause a variety of undesirable effects.

2.3.3 User Flip-flops

User programmable *Flip-Flops* (FFs) are an important architectural component of all FPGAs. Most FPGA circuit designs use FFs to implement sequential logic circuits such as *Finite State Machines* (FSM), counters, synchronizers and registers. Upsets in the user

FFs may be caused by SETs or SEUs. SETs cause transient glitches in the intermediate signals and combinational logic gates. Such glitches may be latched into FFs as incorrect values [16], while SEUs may directly affect the internal state and output values of the FFs.

Table 2.1 shows that user FFs account for the smallest portion of accessible memory cells within an SRAM-based FPGA. However, user FFs typically hold values that are very important to correct circuit operation. Upsets in the FFs therefore often cause circuits to malfunction. For example, if the FFs holding the state of an FSM are upset, these cause the FSM to enter an unintended state. Even worse, such upsets may cause the FSM enter non-existent, deadlock states.

2.3.4 Internal Proprietary State

Internal state plays an important role for all FPGAs as it is used to manage the internal operations and configuration of the FPGA [180]. As the name implies, most of this internal state is not visible to the user, but is crucial for the FPGA to operate properly. Fortunately, this internal state makes up a very small portion of the memory cells within an FPGA, thus, it is far less sensitive to radiation than the other memory types. However, any upset occurring in internal registers may lead to strange behaviour for the FPGA. Such behaviour normally cannot be resolved by the FPGA itself and is often classified as a SEFI. For example, the radiation testing in [190] observed a couple of SEFIs, such as clearing of configuration memory and loss of state data (power-on-reset SEFI) and loss of communication with configuration logic (SelectMap SEFI and JTAG SEFI).

2.3.5 Final Remarks on FPGA Architectural Vulnerabilities

The configuration memory bits account for the largest proportion of memory cells within SRAM-based FPGAs, e.g., more than 80% of modern Xilinx FPGA memory is used to configure the device. Since the susceptibility to SEUs of the various memory types on an FPGA is similar [186], there is therefore a roughly four-fold greater probability of SEUs occurring in configuration memory bits than in user memory bits. Since configuration memory upsets have the potential to alter the function of a LUT or the routing between nodes, they can lead to apparently “permanent” errors, such as logic or stuck-at faults, manifesting in user circuits until the altered configuration state is corrected. In this thesis we mainly focus on proposing approaches for detecting and correcting SEUs that have occurred in the configuration memory bits of an SRAM-based FPGA.

2.4 Mitigating SEUs in SRAM FPGAs

Designers have been interested in using SRAM-based FPGAs in radiation environments, such as space and high-energy physics, due to their high-performance, low *Non-Recurring Engineering* (NRE) cost and flexibility. However, they are susceptible to radiation-induced soft errors. Therefore, many studies have been conducted to mitigate the effects of soft errors on SRAM-based FPGAs in order to satisfy reliability requirements. Techniques involving hardware changes to the FPGA fabric or device architecture have been applied by FPGA vendors. For example, the space-grade Xilinx Virtex-5QV FPGA provides radiation-hardened by design (RHBD) techniques to protect the configuration memory cells from single-event upsets [187]. However, these devices are orders of magnitude more expensive than equivalently sized off-the-shelf FPGAs, consume more power to operate and usually lag a couple of process generations behind industry-leading device technologies in terms of performance. Most importantly, because they are much smaller in size, they are more limited and less flexible than COTS devices [20].

This section focuses on reporting the techniques used by FPGA researchers and designers to mitigate soft errors, particularly SEUs, within COTS SRAM-based FPGAs. These techniques involve using hardware redundancy to mask and detect configuration memory SEUs combined with the use of dynamic partial reconfiguration to correct configuration memory SEUs, using *Error Correction Code* (ECC) to protect against SEUs occurring in BRAMs, using redundancy to protect against SEUs in FFs and using system redundancy to mitigate against SEUs occurring in internal proprietary state.

2.4.1 Terminology

Before we proceed further, we first introduce common terminology that describes an abnormal state of a system, including: *fault*, *error* and *failure*. We will use this terminology throughout this thesis. Typically, a *fault* is understood to be the cause of an *error*, and an *error* to be the cause of a *failure*. For example, in functional safety standard ISO 26262 [172], a *fault* is defined as an "abnormal condition that can cause an element or an item to fail". An *error* is defined as the "discrepancy between a computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition". Finally, a *failure* is defined as the "termination of the ability of an element, to perform a function as required". In this thesis, we view an erroneous FPGA output as a failure. Although the failure is always caused by a fault, a fault does not necessarily cause

a failure. In an FPGA circuit design, a bit-flip or an SEU in a memory cell is viewed as a fault.

2.4.2 Hardware Redundancy

During run time, either temporal or spatial redundancy can be applied to the system in order to mask the effects of SEUs in the FPGA configuration memory. Temporal redundancy, like its name implies, uses redundancy in time [70,159]. A computation is performed on the same hardware at least at two different times to mitigate failures that occur during one of the redundant computations. However, the drawback of temporal redundancy is that circuit performance is degraded at least two fold. On the other hand, spatial redundancy involves several identical circuit structures running in parallel in order to remove single points of failures. Morgan *et al.,* showed that the use of spatial redundancy, such as TMR is more reliable than that of temporal redundancy [105].

Two common forms of spatial redundancy include DWC [77,110,152] and TMR [146,167]. DWC duplicates the user design and uses a comparator to compare the outputs of the two replicas in order to detect upsets in the configuration memory and in user memory. DWC logic requires up to 33% less area and results in higher performance than TMR [157]. However, DWC does not allow the corrupted replica to be identified, thus the entire component has to be reconfigured to erase faults in the configuration memory. Therefore, DWC is unsuitable for safety-critical and real-time systems. On the other hand, TMR triplicates the user design, with each module operating in parallel. The outputs of the three identical modules are fed into a voting circuit that arbitrates between them. If any one of the triplicated modules suffers an error, the other two outvote its output provided that they continue to operate normally. Under such circumstances, TMR not only provides the means to identify a faulty unit, but also improves system reliability.

When applied to FPGAs, TMR can be implemented at the system level by using three separate FPGAs. Doing so normally results in a higher cost and a physically larger system that comprises more components and consumes more power than its simplex equivalent. Secondly, TMR can be applied at the system level but within the one FPGA device. *Thirdly, the design can be triplicated at the level of a system's function units that are implemented within the one FPGA device.* Examples of such units are Finite Impulse Response (FIR) filters, Block Adaptive Quantizer (BAQ) compressors or the pipeline stages of a processor. Finally, triplication may be implemented at the level of basic elements in one FPGA such as counters, decoders, multiplexers, and adders, and at the

level of individual registers. In this thesis, we focus on the third type of triplication outlined, namely triplication of a system's function units.

Alternative methods can be used to mitigate radiation-induced SEUs in SRAM-based FPGAs, including reduced precision redundancy [148], design diversity redundancy [155], state machine encoding [117], and special-purpose placement and routing [149]. Such alternatives have not gained as much traction as TMR, perhaps because they are less general or less straightforward to implement.

While TMR is easy to implement and improves system reliability, it has a significant area cost as the resulting circuit is at least three times the size of the original circuit, and normally operates at a slower clock frequency than the unreplicated (simplex) design. A good deal of FPGA-related research has been directed at reducing the cost of TMR by focusing on protecting the important parts of a circuit [78], and optimizing the placement of voters on feedback paths [81], while ensuring multiple errors do not cause the circuit to fail [25].

TMR on its own does not provide a means of correcting a faulty unit. When a triplicated module contains no feedback paths, SEUs are detected as transient errors in the output. However, when an SEU is trapped in a feedback path, or the configuration memory is affected, errors can persist, and some method for eliminating the error is needed. If feedback values are voted on, then they can be masked, and therefore prevented from being recirculated [25]. A simple method for clearing the user circuit is to reset the system. However, a reset may delay the operation of time-critical systems excessively. Typically, TMR is combined with error recovery techniques, which rely on the dynamic partial reconfiguration capabilities of modern FPGAs to correct configuration memory errors.

2.4.3 Dynamic Partial Reconfiguration

Dynamic Partial Reconfiguration (DPR) allows for the selective reading and writing of configuration memory while the device operates [26]. DPR controllers need to access the configuration data through a configuration memory port, such as the *Internal Configuration Access Port* (ICAP), *Processor Configuration Access Port* (PCAP), SelectMap, or using *Joint Test Action Group* (JTAG) [180]. In fault-tolerant systems, the DPR controller, which we refer to as a *Reconfiguration Controller* (RC), can detect and correct configuration memory upsets without any interruption of the user logic function. The configuration memory is read/written in a packet format, and each packet contains a slice

of configuration data that is referred to as a configuration memory frame. A configuration bitstream contains the data for a series of frames together with a bitstream header and footer, and optional frame headers that control the configuration port registers, such as the frame address register, configuration and command registers [180].

To prevent the build-up of upsets within the configuration memory, the upsets in the configuration memory can be repaired using DPR. To date, two upset mitigation techniques that rely on both TMR and DPR have emerged – TMR-Scrubbing and TMR-MER. By triplicating components and voting on their outputs, TMR helps to protect the user data and to detect configuration memory errors, while by reconfiguring the configuration memory, DPR swiftly corrects configuration memory errors. A brief overview of both TMR-Scrubbing and TMR-MER follows.

2.4.3.1 TMR-Scrubbing

TMR-Scrubbing involves repairing upsets within the configuration memory bits by writing the correct configuration data back into the configuration memory [26]. TMR-Scrubbing can be considered to be similar to the scrubbing employed in conventional memory system to preserve the integrity of the memory [139]. TMR-Scrubbing is typically initiated periodically and commonly involves reading back each configuration memory frame in the device, checking for errors, correcting any that are found and writing back the corrected frame, when necessary.

A variety of TMR-Scrubbing methods have been introduced in the literature. These include blind scrubbing and readback scrubbing [63]. Blind scrubbing continuously writes valid configuration frame data into the device over the existing configuration data [26], whereas readback scrubbing involves reading back configuration memory to detect upsets within the bistream and configuring the bitstream only when an upset in the bitstream is detected. Readback scrubbing can further be classified into several variants such as readback and repair with golden copy; readback with ECC and repair with golden copy; readback and repair with ECC; and readback with *Cyclic Redundancy Check* (CRC) [63].

Xilinx provides an *Intellectual Property* (IP) block, the so-called *Soft Error Mitigation* (SEM) controller [188], which can be used for both blind and readback scrubbing as described above. One of the most commonly used modes of the SEM controller is readback with ECC. In this mode, the controller is able to detect soft errors in the configuration memory by reading configuration frames and calculating an ECC based on the built-in FrameECC primitive [179]. In the Xilinx Artix-7 XC7A200T device, the period of a scrub

cycle is 18.3 ms at the maximum ICAP throughput, which is about 1.01 us per frame (the device has 101 words per frame and the ICAP is 32-bit wide running at 100MHz). To correct errors in a frame, the SEM controller can fetch frames from off-chip memory (0.83 ms per frame, as measured on a Nexys Video Artix-7 FPGA board [38]) or flip the incorrect bit by using ECC information (0.61 ms per frame). While the later approach can only correct single errors, it can be further enhanced to be capable of correcting two adjacent errors in a frame via a hybrid algorithm based on CRC and ECC (18.79 ms per frame). Blind scrubbing therefore is quick, but readback and repair is very much slower. Note that most of the above values depend on the device family and system-level design used. They may increase for larger devices, which contain far more configuration frames than an Artix-7 XC7A200T [188].

2.4.3.2 TMR-MER

Modern Xilinx FPGA technology provides the flexibility for in-situ programming and re-programming without going through re-fabrication. DPR provides further flexibility by allowing the run-time modification of a user design via the loading of a partial configuration file. Following a full configuration of an FPGA, a partial configuration file can be downloaded to modify the pre-defined reconfigurable regions in the FPGA without affecting the functions of those parts of the device that are not being reconfigured [183]. TMR-MER utilises this characteristic of modern FPGAs to recover from configuration memory errors. For example, while a TMR component in the system is running, the voters associated with the TMR component can identify which module is in error (see Section 3.1). A corresponding partial configuration file can then be loaded to overwrite the faulty module without compromising the integrity of its two sibling modules, which continue to function correctly [20]. This method is restricted to correcting errors that occur in a single module, and in particular, cannot detect errors in circuits that are not triplicated. Please see Section 2.5 for a more detailed description of TMR-MER.

2.4.4 Error Correction Code for Block RAM Memories

Modern FPGAs provide built-in Hamming code error correction to support the detection and correction of errors within BRAM memories [181]. To assist with error detection and correction of individual words within a memory block, parity and check bits are added to the BRAM memories. For example, in 7-Series Xilinx FPGAs, BRAMs use 8 bits of a 72-bit wide RAM to store the parity check code. Eight protection bits are optionally

generated during each write operation and stored with 64 bits of data into the memory. These bits are used during read operations to correct any single bit error, or to detect (but not correct) any two-bit error.

The built-in error correction code is used to protect the data within the BRAM from single bit upsets during a read operation, but it does not correct the errors stored in the BRAM. To correct the stored BRAM values, some form of memory scrubbing must be employed [93, 139]. For high-level radiation environments, BRAMs are sometimes also triplicated to avoid single points of failures in the memory, such as stuck-at upsets on the write-enable signal, which may flush all data in the memory. Implementing BRAMs using both TMR and memory scrubbing provides a robust memory protection strategy [93].

2.4.5 Flip-flop Mitigation

TMR can be used to protect the state of FFs from SEUs in all FPGA types [94]. For SRAM-based FPGAs, the combinational circuits forming the input logic of the FFs should be protected with TMR, as such circuits are considerably larger and therefore much likelier to suffer a configuration upset than the FFs themselves. When FFs are located within a cyclic circuit, such as an FSM, voters need to be inserted on the feedback path to enable resynchronization of the triplicated state [78, 82]. Some radiation-hardened FPGAs, such as the RTG4 from Microsemi, implement individual FFs using three internal FFs and dedicated voters (transparent to the users) to guard against the effect of SEUs on the FFs [168].

2.4.6 System-level Mitigation

To mitigate SEFIs and to further improve the reliability of FPGA-based systems in harsh radiation environments, system-level redundancy techniques can be used. For example, some systems apply TMR at the system level using triplicated FPGA devices [21]. In such a system, all three FPGAs contain the same functional design and have the outputs voted on in a hardened device. While this approach provides a high-level of SEU tolerance, it is costly, and correctly designing applications using this approach can be a challenge. For example, if one FPGA has failed due to an SEU, resynchronizing it with the other FPGAs requires special design techniques. Alternatives to device redundancy, such as external watch-dog timers, radiation-hardened system monitors and system check-pointing can be used to mitigate SEUs at the system level.

2.4.7 Final Remarks in Mitigating SEUs on SRAM FPGAs

A range of techniques operating at various levels within the system has been proposed to mitigate radiation-induced effects on different types of memory. The above review has shown that the approach using device redundancy for system-level mitigation provides the best approach from a reliability perspective. However, the cost of implementing this approach is prohibitively high in terms of design effort, device cost, mass, and power consumption. With the aim of reducing the overall cost, while still achieving reasonable system reliability, many researchers have been assessing the use of both TMR-Scrubbing and TMR-MER within a single FPGA device, particularly focusing on their impact on overall system reliability. In this thesis, we also focus on these approaches implemented on a single FPGA device and explore new techniques with the aim of improving overall system reliability.

2.5 TMR-Scrubbing and TMR-MER Overview

Both TMR-Scrubbing and TMR-MER rely on DPR to correct configuration memory errors. TMR-Scrubbing involves periodically reading the current state of the configuration memory within the FPGA and writing correct values back into the configuration memory. In contrast, TMR-MER is commonly triggered when repeated errors are detected by the voter associated with a TMR component and involves rewriting the configuration memory for the module that has been found to be in error. TMR-Scrubbing, which could be referred to as a *frame-based* recovery technique, is thus more fine-grained in its corrective action but involves reading or writing the entire configuration memory contents [26, 63]. On the other hand, TMR-MER is more coarse-grained in that the configuration memory contents of a complete module are rewritten. Multiple configuration memory errors affecting the one module can thus be corrected in a single action and correction is typically completed more promptly.

TMR-Scrubbing consumes more energy than TMR-MER [158], because it is invoked periodically rather than when errors are present and it involves reading the entire configuration memory contents. Since it is invoked at regular intervals rather than when errors are detected by voters, and because the configuration memory of the entire device is read, TMR-Scrubbing also has a higher MTTD than TMR-MER. While TMR-MER is triggered by a voter signalling the presence of an error, state-of-the-art FPGAs may include on the order of a billion configuration bits and the time required to read back the entire config-

uration memory during a scrub cycle may take on the order of 100 ms. This means that SEUs will be detected in the system after half the device has been scanned, on average, which could be too long for critical systems.

As it is commonly described in the literature, the TMR-MER technique is not as robust as TMR-Scrubbing [4]. Both TMR-Scrubbing and TMR-MER utilize a controller to operate [66, 152], but TMR-MER also requires extra networking infrastructure, which we refer to as a *Reconfiguration Control Network* (RCN), to carry out its operations. Modern FPGAs, such as those offered by Xilinx, include built-in resources and provide soft IPs to aid TMR-Scrubbing [188], but use of TMR-MER is not supported nearly as well; design complexity and the likelihood of introducing design errors are therefore increased. For example, a designer who wishes to employ TMR-MER needs to design a suitable reconfiguration controller, a method for detecting and signalling reconfiguration requests, storage for partial bitstreams and a data resynchronization mechanism.

The use of TMR-MER affords several benefits over TMR-Scrubbing, but also introduces several additional costs. The main benefits of TMR-MER are that the system is able to respond more dynamically to configuration memory errors and that this, in turn, enhances reliability and availability.

Configuration memory errors are detected by individual voters when faults occur repeatedly within the one module. Rather than having to read back half of the FPGA's configuration frames, on average, to detect an error when incorporating TMR-Scrubbing recovery [141], TMR-MER utilizes the voters of the system to rapidly detect errors within a few tens to hundreds of clock cycles [27]. When an error is detected by a voter, a reconfiguration request that identifies the module to be reconfigured must be conveyed to the RC. The performance of the RCN determines the latency in transmitting this request. As the investigation of this thesis indicates, this is typically possible within a few microseconds at most. Overall, the time to detect errors with TMR-MER is therefore in the order of microseconds, compared to milliseconds for TMR-Scrubbing [4].

Repairing errors by TMR-MER is also typically much faster than for TMR-Scrubbing since TMR-MER is usually achieved by reconfiguring the faulty module, whereas TMR-Scrubbing necessitates reading the entire memory contents of the device. Thus the size of the module to be repaired, rather than the device size, determines the correction time. Most work focuses on modules of the size of a relatively small number of device resources (a few hundred CLBs, FFs and routing) to medium-sized components such as linear filters [29] and a processor stage [129]. Typical module sizes range between 10s and 100s of configuration memory frames (several kB) as opposed to 10,000s of frames (many MB)

involved at device level. It can therefore be expected that error repair also takes two to three orders of magnitude less time for TMR-MER. This responsiveness to errors afforded by TMR-MER can be crucial to the success of a mission when errors occur in rapid succession at critical stages such as during spacecraft manoeuvres, communications with Earth and during remote sensing when particular areas of interest are observed.

Since TMR-MER replaces all the configuration memory contents for a module, it inherently repairs multiple errors, such as *Multiple Bit Upsets* (MBUs), which simple TMR-Scrubbing modes, such as ECC-based *Single Error Correction, Double Error Detection* (SECDED) [188], are generally unable to fix. Only more complex TMR-Scrubbing approaches, such as frame replacement, which comes with many of the costs of TMR-MER, are capable of dealing with MBUs efficiently [188].

TMR-MER comes with some considerable costs. However, most of these have not yet been quantified as they mainly deal with greater design complexity. The designer needs to consider modifications to the voter design to be able to detect configuration memory errors; an RCN is needed to convey reconfiguration requests to an RC; an RC that can process reconfiguration requests and manage modular reconfiguration needs to be designed; secure storage for the module-based partial bitstreams needs to be incorporated; a module-based resynchronization method also needs to be developed. If the designer were to choose a simple TMR-Scrubbing-based configuration memory recovery method instead, most of these design modifications are not needed. Instead, the designer could make use of IP provided by the device vendor and in-built hardware to perform TMR-Scrubbing. This is the case when SECDED is used [188]. If a frame replacement TMR-Scrubbing method is used, then the designer also needs to develop a more sophisticated scrub controller, implement a method for securely storing the device configuration, and implement a method for accessing the golden configuration on a frame-by-frame basis.

Of considerable concern is that much of the additional logic used to implement and support TMR-MER may be implemented in a non-redundant manner and therefore introduces additional single points of failure. Nevertheless, irrespective of the configuration memory error recovery approach taken, FPGA-based TMR systems inevitably include non-redundant components such as clock managers, ICAP and off-chip ports. Periodically invoked recovery, such as TMR-Scrubbing, is likely to deal with configuration memory errors that occur in these components better than TMR-MER does, since non-redundant components don't have voters to trigger TMR-MER. Agiakatsikas *et al.* proposed *Frame-/Module-based Error Recovery* (FMER) that combines both scrubbing and MER to recover configuration memory errors in SoCs that contain both non-redundant and redundant components to

exploit the benefits of both approaches and thereby improve the SoC's availability [3].

2.6 Related Work on Novel Techniques for Improving System Reliability

The literature describes techniques that have been developed to decrease the mean time to detect and correct configuration memory errors in FPGA-based systems with a view to improving system reliability. These techniques are based either on frame-based scrubbing [10, 91, 110, 140, 141, 143] or on TMR-MER [4, 20, 29, 113, 152, 163].

The use of scrubbing has been extensively researched in the literature. Numerous studies have shown that scrubbing approaches that are based on the critical metrics that directly affect system reliability are more reliable than conventional scrubbing, which periodically and systematically scrubs from the first frame of the device to the last. Asadi *et al.* proposed an approach that reduces MTTD by scrubbing only those configuration frames that contain sensitive bits, which affect the circuit operation when they are flipped [10]. In [91], Lee *et al.* presented a heterogeneous scrubbing approach that varies the scrub rates of different components based on the number of sensitive bits they contain. In [110], Nazar *et al.* presented a mechanism that statistically finds an optimal frame to commence scrubbing at in order to reduce the mean time to repair configuration memory errors. In addition, Santos *et al.* showed that overall system reliability is improved when the rate at which scrubbing is performed is based on the criticality of user tasks [140]. In [143], Sari *et al.* presented a methodology that combines check-pointing and scrubbing to guarantee the fault tolerant execution of real-time tasks in FPGA-based embedded systems. They introduced four scrubbing approaches to reduce the time overhead of the scrubbing method. Scrubbing approaches include: *full scan*, which involves the scrubbing the entire FPGA configuration memory; *partial scan*, which considers only the sensitive frames of the SoC design [141]; *constrained scan*, which is similar to the *partial scan*, but the design is constrained with less number of sensitive frames [141]; and *selective scan*, which involves scrubbing operational tasks and excludes scrubbing unused tasks. Last but not least, Schmidt *et al.* applied a netlist analysis using the Xilinx *bitgen* tool to identify essential and critical bits in the configuration memory and constrained the placement and routing of a given design with the aim of reducing the number of frames that need to be scrubbed and thereby improving system reliability [144]. The work we describe in Chapter 5 checks the voters of components for configuration memory errors based not only on the number of essential bits contained in the TMR modules but also on the recovery times

of the modules. This is similar to the approach of [91], but does not involve any further action, such as scrubbing or modular reconfiguration, when errors are not present.

TMR-MER systems utilize an RCN such as a star-based [3, 4, 20, 152], a bus-based [4], or an ICAP-based network [163] to convey the status of the individual TMR component voters to an RC, which determines whether or not configuration memory errors are present. To determine whether any configuration memory upsets have occurred, most TMR-MER systems check the voters of the TMR components in round-robin order [20, 29, 152, 163]. However, doing so increases the response time for checking the voters of highly vulnerable TMR components. Intuitively, the rate at which a TMR component requests error recovery depends on its failure rate or its criticality. Therefore, the RCN arbiter should check components with high failure rates more frequently than those with lower failure rates in order to minimize the chance that the system becomes unusable. In Chapter 4, we develop a method for identifying the next component to check at run time based on the likelihood that it has failed since the last check of the component [113]. In Chapter 5, we report on an off-line approach to determine a fixed voter checking sequence in a TMR-MER system. Our work in Chapter 5 aims to enhance the system's error detection capabilities and thereby raises overall system reliability by checking the TMR component voters for module errors at different pre-determined rates.

2.7 Measuring FPGA SEU Sensitivity

SRAM-based FPGAs are more and more relevant to a growing number of safety-critical applications, ranging from automotive control to aerospace domains. Designers of safety-critical applications demand accurate methods for evaluating the SEU sensitivity of their designs.

The SEU sensitivity profile differs from FPGA design to FPGA design since each design uses a different set of FPGA resources. As mentioned in Section 2.3, modern FPGAs are very complex heterogeneous devices that contain a variety of resources. The behaviour of these resources is defined by a custom combination of configuration memory bits within the device. The number of configuration memory bits associated with an FPGA design determines the SEU sensitivity of the design. Generally, smaller designs utilising fewer FPGA resources are less sensitive to SEUs than larger designs, which make use of more memory resources.

The benefit of any SEU mitigation technique can be determined by accurately gauging the

SEU sensitivity of a given FPGA design. To estimate FPGA design sensitivity, a variety of techniques have been used in the literature. The standard for measuring SEU sensitivity is to use accelerated radiation testing [123]. The benefit of radiation testing is that we can then observe the effect of SEUs on all memory types inside the device under test (DUT), such as internal proprietary state cells, which are not accessible via fault-injection testing. The SEU sensitivity measured by radiation testing can therefore be expected to be closest to that observed in the field.

In radiation testing, high-energy particles are applied to the DUT to gauge the response to upsets within the memory resources of the device. During the radiation test procedure, frame readback is used to count SEUs, which are then recorded and repaired. When an SEU is detected, the output of the DUT is checked for errors and the results are recorded. In order to obtain the configuration memory upset rates, the configuration memory of the DUT is read back and compared to the original state to yield SEU counts after a specific time period. Upsets on the internal proprietary state are observed based on SEFI events, which are extremely rare [190] and which are unique to radiation testing.

On the other hand, many studies have explored the use of fault injection techniques to obtain similar information on SEU sensitivity [123, 133]. Fault injection methods can be grouped into two main approaches, namely, simulation-based fault injection and emulation-based fault injection. Simulation-based fault injection utilizes simulation tools to introduce errors into a model of the target system [19,73]. The basic idea of simulation-based fault injection is to first simulate the fault-free design in order to store the golden results. Each fault is sequentially simulated by loading the state of the design just before fault activation time, injecting the SEU, comparing the current results with the golden one and classifying the effect of the injected fault. However, simulation-based fault injection is a slow process. It may not be feasible to obtain the SEU sensitivity in a reasonable period of time. This is because identifying the SEU sensitivity of a given FPGA design requires injecting a significant number of faults, possibly one for each configuration bit of the device, and simulating the operation of the design with a range of inputs. Therefore, simulation-based fault injection is used to inject a small set of carefully selected faults to outline possible design bugs.

To accelerate the fault injection process, emulation-based fault injection approaches have been developed in recent years [5, 6, 9, 34, 43, 71, 73, 95, 142]. This type of fault injection utilizes FPGA hardware to prototype the *Component Under Test* (CUT) and to support inserting faults into the FPGA rather than using simulation tools.

Using emulation-based fault injection to evaluate the SEU sensitivity of a given FPGA

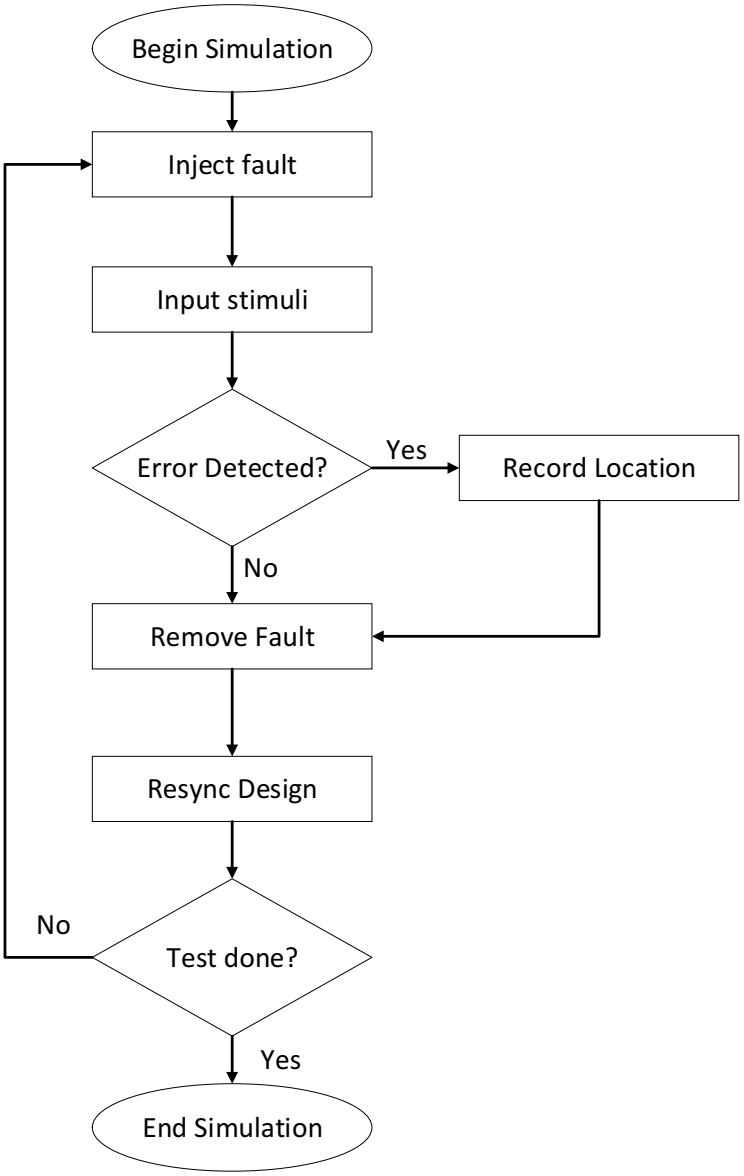


Figure 2.3: Basic emulation-based fault injection algorithm [75, 133]

design has many advantages, including low cost, rapid testing and the ability to customize the experiment. Quinn *et al.,* mentioned that accelerated radiation testing can cost up to $1,500 per hour and that the available facilities for conducting the experiment are limited [133]. For emulation-based fault injection, the total cost includes the support hardware for fault injection and the engineering effort to create and run the emulation system. Since hardware emulation is exploited to boost the simulation performance, the results are obtained far more quickly than with simulation-based approaches. Emulation-based fault injection is about four orders of magnitude faster than simulation-based fault injection [34, 43]. Indeed, when the FPGA designs are more complex and very large, fault injections have to be considered. Moreover, emulation-based fault injection can be used to test different scenarios and experiments since the type and location of faults can be customized by test designers. For example, the test designers can modify the fault injection system to perform *Multiple-Independent Upsets* (MIU) [34, 123, 156], *Multiple-Cell Upsets* (MCU) [32, 132] and target upsets at specific regions of the device [4, 31].

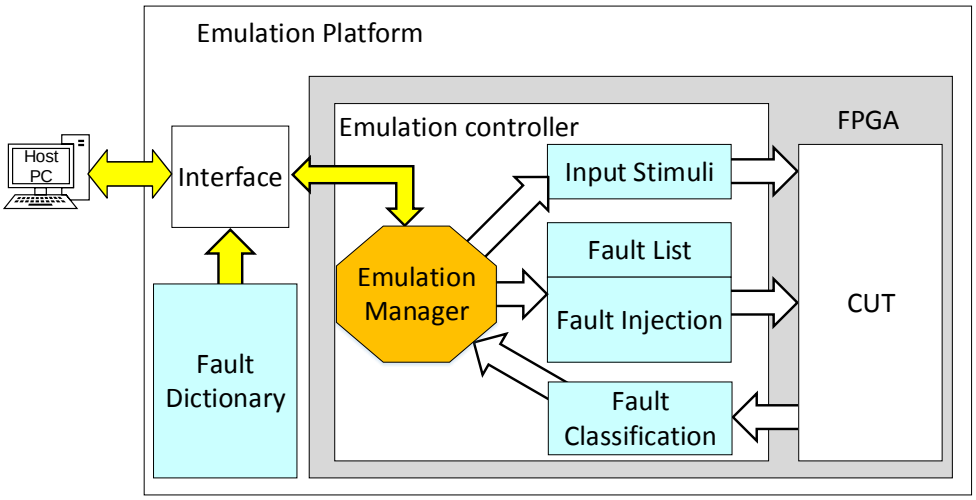


Figure 2.4: Autonomous Emulation System [47]

Figure 2.3 depicts a baseline fault emulation process, while Figure 2.4 illustrates a typical hardware system that is used in fault emulation testing. Such a system includes the CUT and the emulation controller that controls the fault emulation test procedure. As can be seen in Figure 2.3, there are a few common functions that are required in every fault emulation system including [75, 94, 133]:

- The ability to access the internal configuration memory where the faults are inserted;
- The ability to stimulate the input and execute the CUT;

- The ability to report system output faults;
- The ability to clear the error and re-sync the design.

Each of these steps is detailed in the following subsections.

2.7.1 Fault insertion

The first operation of a fault injection system is to insert faults into the CUT. Fault insertion involves three main steps, including reading a segment of the configuration memory, corrupting a bit and rewriting the corrupted memory segment back into the FPGA. A number of methods can be used to access configuration memory in the FPGA via a configuration port such as JTAG, SelectMap, ICAP, and PCAP available in Xilinx FPGA devices [180]. The JTAG boundary scan port is a popular access point for emulation-based fault injection since it is available in most FPGA devices [1, 55, 60, 101, 127, 151]. JTAG provides a serial interface to access the configuration memory that can be used by many tools and software. However, JTAG is relatively slower than the other configuration ports.

Along with JTAG, most FPGA manufacturers also provide other means to access configuration memory at higher throughput. For example, Xilinx FPGAs provide both external and internal parallel configuration ports with which faults can be inserted much faster than with JTAG. On Xilinx devices, the external port is called SelectMap, while the internal ports include ICAP, available from the Virtex II on, and PCAP, available in 7-Series Zynq devices. Xilinx also provides an IP core [188] that has the capability of modifying the configuration memory and implementing fault emulation. In recent years, the use of internal ports to implement emulation-based fault injection has become more prevalent since it is relatively easy to implement such a fault injection system using evaluation boards. These systems only use one FPGA to implement the entire system, including the CUT, the emulation controller and an interface to the host PC. Note that Altera FPGAs also provide similar facilities for fault injection [43].

Emulating faults in the user memory is much more difficult than emulating faults in the configuration memory. As the configuration memory values are static, it is easier to read, write and modify these values, whereas accessing user memory while the circuit is operating is difficult to do directly, and may corrupt user data.

2.7.2 Input Stimulation

Once the fault is inserted, the input is stimulated to test the circuit. It is difficult to cover all possible input values since the input vector set increases exponentially with the input data width. Moreover, some fault locations such as those in sequential logic may only be triggered by a specific sequence of input values. The best techniques create input test vectors that test all functions of the circuit. Such techniques include random testing [40], structural testing [193] and statistical testing [128]. Random testing creates test vectors by randomly sampling all possible input values according to a specific probability distribution. Structural testing uses information about the CUT implementation to select input data for test cases that exercise each coverage element at least one. Statistical testing combines both random testing and structural testing. In other words, statistical testing generates test vector sets by sampling the input domain according to a probability distribution and such vector sets must satisfy the test cases of structural testing. When the input test vectors are difficult to craft, one of the most common alternatives is to use pseudo-random input sequences, which can be generated using linear feedback shift registers, as input stimuli. The larger the number of input stimuli covered, the higher the probability that the inserted faults will be detected.

2.7.3 Error Detection

The next step of the fault emulation system is to detect system output errors. There are two main methods used to detect errors. The first one is to use a redundant design that acts as the golden reference circuit. A golden circuit is designed to be identical to the CUT, but is not subjected to fault injections. The golden circuit can be implemented in a different FPGA device or on the same device with the CUT. The outputs of both the CUT and the golden circuit are compared and any difference between them is identified as a system error in the CUT. Alternatively, only the CUT is implemented and subjected to fault injection. In this case, its outputs are compared with a set of predefined, expected output values.

An important attribute for error detection in an FPGA fault emulation system is the period of time that the FPGA system is permitted to execute to allow errors to be detected after a fault has been inserted. The longer the period of time given, the higher the probability that the inserted fault manifests as an output error that is detected. If the period is too short, it is possible that the inserted fault causes an error, but that the error is not observed at the output of the CUT. This is because the time period is either insufficient

for errors to propagate to the CUT output or the random input stimuli do not cover the circuit element into which the fault was inserted.

2.7.4 Error Clearance

The last step of the fault emulation system is to prepare the system for the next fault to be inserted. The preparation involves clearing the inserted fault and returning the CUT to a known good state. It should be noted that even if an inserted fault does not cause an observable error at the output of the CUT, it is possible that a latent error exists in the CUT. Thus, the next fault insertion may activate the previous faults, thereby causing the CUT to output errors, which leads to incorrect fault attribution. However, this is not the case if MIU injections are considered where multiple faults are inserted and they are removed at prescribed intervals, or once the faults cause the CUT to output errors. To return the CUT to a known, fault-free state, the system resets the state of the user flip-flops to their initial values.

2.7.5 Final Remarks on Measuring FPGA SEU Sensitivity

Compared to accelerated radiation testing and simulation-based fault injection, emulation-based fault injection has lower cost, obtains results faster and allows a wider variety of fault scenarios to be tested. Emulation-based fault injection is particularly beneficial in assessing the SEU sensitivity of complex design where very large fault lists and sets of input values are needed to conduct the experiment. In this thesis, emulation-based fault injection testing is used to validate the work in Chapters 3 and 5.

2.8 Reliability Model

In this section, we outline how we model the reliability of a non-replicated component, the reliability of a TMR component and the reliability of a complete FPGA-based TMR-MER system. Our analysis is based on the number of critical or sensitive bits per component for which we use the number of essential bits reported by the vendor's tools as a worst case estimate.

In the following, we assume that the flip of a single essential bit leads to a module failure if the module is not triplicated. With this assumption, the module failure rate λ_m is given

by the product of the bit error rate, λ_{bit}, and the number of essential bits in module m. We also assume that the three modules of a TMR component have the same failure rate λ_m.

Using SPENVIS [67], the bit error rates of the configuration memory of Xilinx 7–series FPGAs for different orbits and different environmental conditions were calculated as shown in Table 2.2. To determine these error rates, we used the solar min, the worst week, the worst day and the peak 5-minute average flux of the CREME-96 model [161] with 2.54 mm aluminium shielding. The cross–section areas of the 7 series FPGA families was obtained from [68].

Table 2.2: Bit failure rates in different orbits [67]

Orbit	**Altitude (km) Inclination**	**Solar Min**	**Worst Week**	**Worst Day**	**Peak 5-Min**
		λ (SEUs/Bit/s)			
GEO	35,768 0^o	1.71E-13	2.16E-11	7.34E-11	2.66E-10
GPS	20,200 55^o	1.54E-13	1.43E-11	4.84E-11	1.75E-10
LEO	2,000 51.6^o	1.92E-14	7.01E-13	2.33E-12	8.41E-12

We assume that module reliability decreases exponentially over time t as expressed by the function:

$$R_m(t) = e^{-\lambda_m t}, \tag{2.1}$$

whereby the reliability, $R_m(t)$, of a module at time t denotes the probability that the module operates without any failure in the interval $[0, t]$.

When module m is triplicated, its reliability function is given by [167]:

$$R_m^{TMR}(t) = 3R_m^2(t) - 2R_m^3(t). \tag{2.2}$$

The reliability function of a TMR component i with *Module-based Error Recovery* (MER) for an SEU rate of λ_i is given by [100]:

$$R_i^{TMR}(t) = \frac{e^{-\frac{1}{2}(at)}\left(a\sinh\left(\frac{bt}{2}\right) + b\cosh\left(\frac{bt}{2}\right)\right)}{b}, \tag{2.3}$$

with $a = 5\lambda_i + \mu_i$ and $b = \sqrt{\lambda_i^2 + 10\lambda_i\mu_i + \mu_i^2}$. μ_i denotes the repair rate of a module, which is the reciprocal of the time needed to recover the faulty module, and it can be

expressed as:

$$\mu_i = \frac{1}{t_i} = \frac{1}{t_d + t_c + t_{sync}} \approx \frac{1}{t_d + t_c}, \tag{2.4}$$

where t_d denotes the average error detection time, t_c the error correction time and t_{sync} the synchronization time, which is omitted in our study as it normally only accounts for a small fraction of the recovery time. t_d depends on the method used to detect errors while t_c depends on parameters of the target system and the size of the module or the device.

The average failure rate λ_i^{TMR} for a TMR component with repair can be estimated according to [123] by

$$\lambda_i^{TMR} = \frac{1 - R_i^{TMR}(t_i)}{t_i}. \tag{2.5}$$

where t_i is the time needed to recover the faulty module.

Typically, a system contains N interdependent TMR components connected in series such that the failure of any one TMR component causes the system to fail. Note that this definition of a series system is from a reliability perspective and not always an electrical or mechanical one. In other words, the output of the first component is not necessarily connected to the input of the second component, etc. For example, a microprocessor may consist of four units, namely, an instruction fetch unit, a decode unit, an execution unit, and a memory access unit. Although all units are not physically connected in series, the failure of any one unit leads to the malfunction of the microprocessor. The failure rate of a series TMR system, λ_s, is the sum of all component failure rates [85]:

$$\lambda_s^{TMR} = \sum_{i=1}^{N} \lambda_i^{TMR}. \tag{2.6}$$

Using Equations (2.1) and (2.6), the reliability function of a TMR system can be finally calculated as:

$$R_s^{TMR}(t) = e^{-\lambda_s^{TMR}.t}. \tag{2.7}$$

The failure probability of a TMR system over time interval [0, t] can then be estimated as follows:

$$FP_s^{TMR}(t) = 1 - R_s^{TMR}(t). \tag{2.8}$$

Note that Equation (2.6) holds true only if $\mu \gg \lambda$, which ensures repairs are completed independently [145].

In this thesis, we will use this model to evaluate the reliability of systems studied in Chapters 3 and 4.

2.9 Summary

This chapter has provided an overview of radiation-induced effects on SRAM-based FPGAs as well as the radiation sources that may affect such devices in the field. It has also reviewed the architecture of SRAM-based FPGAs that are threatened by radiation-induced soft errors and surveyed common techniques to mitigate such soft errors. Two approaches are commonly used in the literature — TMR-Scrubbing has been extensively researched, but this is not the case for TMR-MER. Intuitively, TMR-MER has many advantages with respect to TMR-Scrubbing, but the design of TMR-MER is more complicated. In this thesis, we explore TMR-MER in detail in order to better understand and compare the relative merits of TMR-MER and TMR-Scrubbing.

Previous works have mainly applied TMR-Scrubbing for improving system reliability. Many novel techniques have been proposed in the literature. In contrast, research into boosting the overall system reliability using TMR-MER is limited. We therefore conclude that there is significant room for improving the reliability of state-of-the-art TMR-MER systems.

Along with fault injection testing, reliability models are useful tools for estimating at an early stage the effectiveness of an FPGA design employing a mitigation technique. The model described in Section 2.8 assumes that the repairs of soft errors are completely independent. While this is reasonable at low error rates, the problem with this assumption at high error rates is that the methods for correcting configuration memory errors are inherently sequential, hence the models do not consider the effect of configuration memory errors on other TMR components while a faulty module is being repaired. Therefore, it is necessary to have accurate reliability models that consider multiple coincident SEUs that may occur in different TMR components and to use these models to analyse the impact of the new approaches on the overall TMR-MER system reliability.

Chapter 3

Reliable TMR-MER System Model

As mentioned in Chapter 2, both upset mitigation techniques — Triple Modular Redundancy with configuration memory scrubbing (TMR-Scrubbing) and Triple Modular Redundancy with Module-based Error Recovery (TMR-MER) — utilize a controller to operate. However, TMR-MER also requires a *Reconfiguration Control Network* (RCN) to relay error requests from the voters in the system to the *Reconfiguration Controller* (RC) [4].

Figure 3.1 illustrates an FPGA-based TMR-MER system. The *voter* associated with each TMR component identifies which module, if any, is suffering from a persistent fault, and raises a *Reconfiguration Request* (RR). Requests from the voters of different TMR components across the device are observed by an RC via an RCN. If the RC observes a fault in one of the modules of any of the system's TMR components, it fetches the partial bitstream corresponding to the module from off-chip memory and reconfigures it by writing the bitstream to the *Internal Configuration Access Port* (ICAP) present in advanced FPGAs from Xilinx. After the faulty module has been reconfigured and resynchronized[1] with the remaining two modules of the TMR component, the voter resumes its normal checking function.

In this chapter, we describe three hardware aspects for designing a reliable FPGA-based TMR-MER system including TMR components, the RC and the RCN. The chapter starts

[1]Resynchronization is readily achieved if the module logic is acyclic [29]

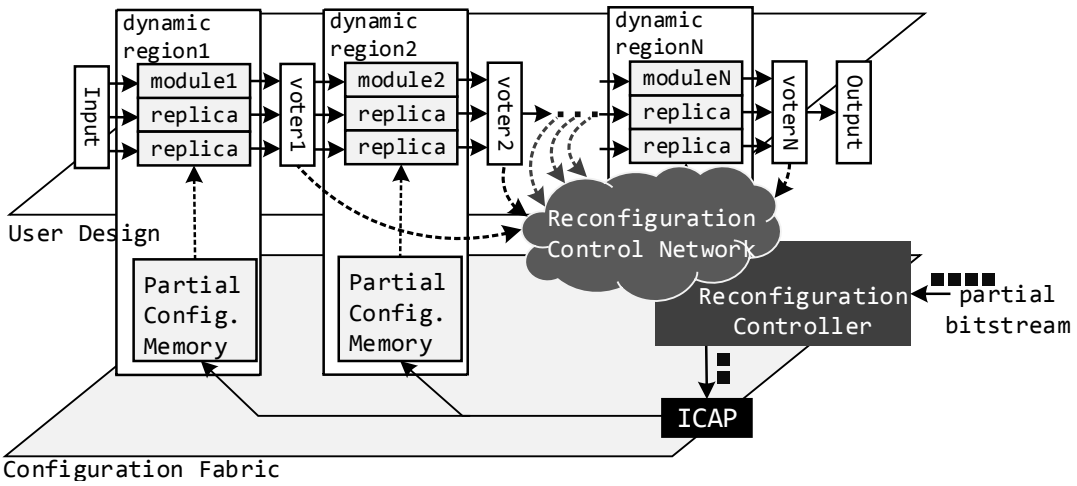


Figure 3.1: An example TMR-MER system diagram

by describing the typical design of a TMR component and the internal structure of a voter needed by the TMR-MER approach (Section 3.1). This is followed by a survey of RCs that have been report for both TMR-Scrubbing and TMR-MER in the literature (Section 3.2). The main contribution of this chapter appears in Section 3.3, which provides details of 4 RCN topologies commonly used by the TMR-MER approach as well as a comparison of the RCNs in terms of reliability, latency and power consumption.

3.1 A TMR Component

A TMR component is created by replicating a critical system unit into three functionally identical units that perform the same operation at the same time. A majority voter is used to generate a single output from the three units that agrees with at least two of the outputs of the triplicated system, as shown in Figure 3.2(a). Such units can be independent sub-systems, such as an FIR filter and a BAQ compressor, or they can be interconnected and interdependent components, such as the stages of a processor, including the instruction fetch unit, the instruction decode unit, the execution unit, and the memory write-back unit [129]. If the triplicated unit contains a cyclic data-path (e.g., counters or accumulators) and the data-path suffers a configuration or data-path memory error, an output error may persist in the feedback path of the cycle leading to ongoing disagreement between the faulty unit with the other remaining functionally correct ones, even after the error has been corrected [78]. To avoid such feedback path errors, majority voters need to be inserted into these feedback paths as illustrated in Figure 3.2(b) [25]. As the majority

voting circuitry is a single point of failure, voters also need to be triplicated to assure high reliability as demonstrated in Figure 3.2(c) and 3.2(d).

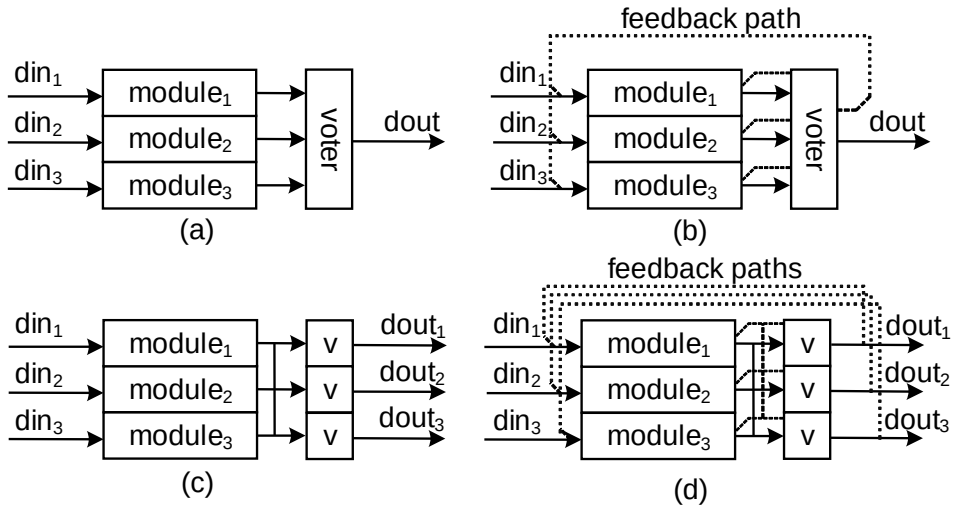


Figure 3.2: Triple Modular Redundancy

When designing a TMR component, it is necessary to consider resynchronization of datapath state after the reconfiguration of a faulty module in order to realign the outputs of the TMR component. Unfortunately, the resynchronization techniques needed differ from TMR design to TMR design. For a typical circuit design, a synchronizing voter can be inserted into the feedback edges of the circuit in order to feedback values that have been voted upon and thus break the recycling of incorrect state [25,78]. This approach is suitable for processing circuitry where the feedback edges in the circuit are easily identified. For triplicated *Finite State Machine* (FSM) circuits, check-points and state prediction logic, as proposed in [125], are more applicable. After the faulty module has been reconfigured, the prediction logic guides the FSM to enter a checkpoint state, in which the correct values are copied to all modules. For soft-processors, the synchronization can be achieved in a number of ways that trade the cost of implementing extra hardware against the time needed for the synchronization process. Such techniques include applying a reset to synchronize the processor internal states, using TMR shared memory to synchronize program and data memory, and using interrupts to minimize the synchronization time for both internal state and the data and instruction memory of the triplicated processors [72,86]. Note that most synchronization techniques require a trigger after the reconfiguration, which calls for something like the RCN used by the TMR-MER system to be able to communicate the synchronization request. More details of RCNs are given in Section 3.3.

In this thesis, we only consider FPGA designs that have synchronization voters inserted

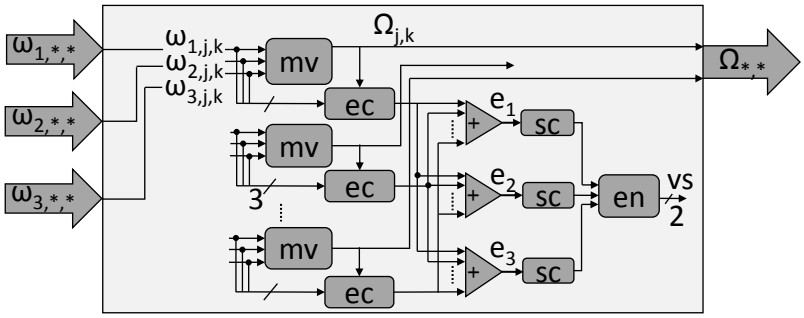


Figure 3.3: Voter internal structure.

on the feedback paths of the circuit. In addition, some extra logic is added to the majority voter for detecting errors and for triggering the reconfiguration of a faulty module. The following describes the internal structure of the voter that we used in the systems described in this thesis.

Figure 3.3 shows the internal structure of a voter, which fulfils two roles:

- Error Detection - detecting errors and distinguishing between "transient" and "permanent" errors, and
- Reconfiguration Trigger - generating a reconfiguration request when a permanent error is detected.

Error-Detection: We refer to the k^{th} bit of output j for module i using the symbol $\omega_{i,j,k}$. We use a *majority voter* (mv) to form the checked output $\Omega_{j,k} = \omega_{1,j,k}\omega_{2,j,k} + \omega_{2,j,k}\omega_{3,j,k} + \omega_{3,j,k}\omega_{1,j,k}$. The corresponding output of each module can then be checked by reference to the checked output i.e. $\forall i,\ e_{i,j,k} = \omega_{i,j,k} \oplus \Omega_{j,k}$ is an error signal for module i, output j, bit k. An *error check* (ec) block for each bit k of each output j produces these signals.

Reconfiguration Trigger: All error signals $e_{i,j,k}$ for module i are OR-ed together (e_i) to detect the presence of an error in any output of module i during the current clock cycle. If the module is acyclic, which it is if all feedback edges are voted on, an error occurring on successive cycles suggests the presence of a permanent error that can only be cleared via reconfiguration. We use an *n-bit saturating up/down error counter* (sc) to count the number of successive errors for each module. If the value of the counter reaches a threshold, e.g., 3 in a 2-bit counter, the output flag will be set high and fed to the *encoder* (en) block. The *voter status* (vs), the output of the encoder, is registered and has four different binary values i.e., 00, 01, 10 and 11 corresponding to the 1^{st} module being in

error, the 2^{nd} module being in error, the 3^{rd} module being in error, and no-error being present, respectively. The voter status is read by the RC to detect the presence of a faulty module before reconfiguring it.

After the faulty module has been reconfigured and resynchronized with its two remaining working modules, the saturating counter for that particular module is reset by the RC to its initial state. For acyclic modules, this occurs after new inputs have reached the output of the reconfigured module i.e. after a period of time corresponding to the latency of the module. Please note that the clock and reset signals for the saturating counters are intentionally not shown in Figure 3.3 for the sake of clarity.

3.2 Reconfiguration Controller

Both TMR-Scrubbing and TMR-MER employ user logic to manipulate configuration bitstreams by reading from and writing to an FPGA configuration port, such as ICAP in advanced Xilinx FPGAs. The RC, which oversees this process, is thus a critical component for fault-tolerant applications in FPGA systems used in high radiation environments. For such applications, it is equally important to reduce the risks of radiation-induced errors to the RC itself and ensuring its reliability is therefore of paramount importance in guaranteeing that the overall system operates reliably.

While various reasons could cause an FPGA design to fail in the presence of radiation, the most common failure is due to the corruption of the configuration memory induced by radiation [135]. User and research interest has focused on devising autonomous methods for detecting and recovering from configuration memory errors as they occur. Broadly speaking, the controllers that oversee fault detection and recovery, and the golden copy of the recovery bitstreams are located either on- or off-chip [66]. In this section, we examine on-chip controllers that fetch externally stored bitstreams and access an internal configuration port to check and overwrite the configuration memory as seen in Figure 3.1. Please note that the controller also needs to ensure that partial bitstreams are reliably read. This can be done by triplicating the memory controller [58] and/or by performing CRC checking before a partial bitstream is fully loaded through the ICAP [176, 191].

The two fundamental operations of the RC are reading and writing the configuration memory to detect and recover from configuration memory errors. For example, in the TMR-MER approach, the existence of an error can be determined by reading voter status by configuration readback via the configuration port [4, 163] (see Section 3.3). After an

error is detected, TMR-MER approaches typically recover from the error by partially reconfiguring the erroneous module [152, 162]. In TMR-Scrubbing, the RC refreshes the FPGA device by periodically rewriting the configuration memory [26,146]. A more efficient scrubbing method relies on ECC stored with each configuration memory frame [188]. The RC reads the configuration memory so as to calculate and check the ECC data, which can isolate single bit errors. A corrected configuration frame is written back to the device when an error is identified. Last but not least, fault injection, which is a commonly-used technique for test and debug purposes, is typically implemented by intentionally writing an erroneous frame to the FPGA device [5,6,9,34,43,71,73,95,142], as detailed in Section 2.7. Since it is specialized for accessing the configuration memory, the RC can also be used to perform this function.

The RC plays an important role in reliable FPGA designs, and there are a variety of different use cases for it. Various proprietary and academic controllers have been developed to meet the general needs of dynamically reconfiguring FPGAs as well as the specific requirements posed by radiation fault-tolerant applications. In the following subsections, we review the RCs that have been reported in the literature.

3.2.1 Commonly Used Reconfiguration Controllers

FPGA manufacturers provide IPs that can be used for fault recovery such as the HWICAP [178] and the SEM controller [188] from Xilinx. HWICAP is a general-purpose IP block that provides a flexible bus-based interface to the ICAP and allows arbitrary configuration data to be written to or read from that port [178]. Software can send any command sequence to the ICAP, including single frame R(ead)/W(rite) as well as partial/complete bitstreams; it can therefore be programmed to perform any fault recovery task the designer wishes to implement. HWICAP is commonly used with a soft processor such as MicroBlaze [177]. Unfortunately, the microprocessor + HWICAP solution is slow and the programmability, which is needed to improve flexibility comes at the cost of a large resource overhead [164]. On the other hand, the SEM controller [188] is a dedicated IP for fault recovery, is light-weight and fast. However, the SEM controller is neither open, nor can it be customized, since it is based on the PicoBlaze processor [175], which does not have an official C compiler and suffers from an extremely small instruction store (1,024 words). Hence, the controller cannot readily be reprogrammed to perform new or different scrubbing functions.

3.2.2 High Performance Reconfiguration Controllers

Several academic development efforts have achieved ICAP throughput approaching the maximum rated capacity of 400 MB/s but invariably compromise on flexibility and/or reliability. Representative efforts include:

- AC-ICAP [23] for Kintex-7, which provides an *Advanced eXtensible Interface* (AXI) interface and therefore can be interfaced to a MicroBlaze or user logic, achieves 380 MB/s ICAP throughput using 1286 LUTs, 1193 FFs and 22 BRAMs, and supports single frame R/W as well as the loading of partial bitstreams, but does not support the loading of arbitrary commands, preventing state capture, for example;
- A self-recovering controller [42], developed for Virtex-4, that has the ability to recover from errors within the controller by loading pre-stored recovery bitstreams, achieves 380 MB/s throughput, performs single frame R/W and loads partial bitstreams, and also supports ECC scrubbing. While fast, this controller's flexibility is compromised by virtue of being PicoBlaze based, as discussed in Section 3.2.1;
- An open source controller [164], developed for Virtex-6, that can be over-clocked to drive the ICAP at up to 838 MB/s, uses 586 LUTs, 672 FFs and 8 BRAMs, but is inflexible as it only supports the loading of partial bitstreams;
- Another open source controller [24], also designed for Virtex-6, that only supports loading of protected bitstreams by performing SECDED at 320 MB/s or CRC at 395 MB/s, which uses about 590 LUTs, 300 FFs, and 1 BRAM; and
- More recent FPGAs, such as the Xilinx Zynq 7-Series SoCs, provide hard-core processors that can configure the FPGA logic using the *Processor Configuration Access Port* (PCAP). The high-speed configuration memory access through the PCAP primitive can be combined with the built-in ECC capabilities of the FPGA to perform fast device scrubbing with a throughput af 145 MB/s [150].

A number of researchers, such as [80], have worked on pure hardware-based RCs for better performance but are more resource intensive to implement than combined software-hardware approaches. Furthermore, hardware-only designs are not programmable and lack the flexibility needed to develop and explore different fault detection and recovery applications.

3.2.3 Reliable Reconfiguration Controllers

Various proprietary and academic RCs have been developed to meet the general needs of dynamically reconfiguring FPGAs as well as the specific requirements posed by fault-tolerant applications in harsh radiation environments. The HWICAP [178] and the SEM controller [188] are Xilinx IPs that can be used for fault recovery. HWICAP, which is commonly used with a soft processor such as a MicroBlaze, suffers from large resource overheads and slow performance [58], while the SEM controller fails to meet flexibility needs as it does not allow alternative user-defined scrubbing functions, such as selective scrubbing of user-nominated regions [3], to be developed. Significantly, the MicroBlaze is not designed to withstand SEUs, and it is not known whether the HWICAP or the SEM controller have been designed to be fault tolerant. Furthermore, without knowing their implementation, applying fault-tolerance techniques, such as TMR, to proprietary IP is non-trivial.

Focusing on the reliability of the RCs themselves, in [41, 42] a self-recovering RC that has the ability to correct any error in its configuration memory by writing a pre-stored recovery bitstream to the ICAP has been proposed. This RC can perform single frame R/W, load partial bitstreams and supports ECC scrubbing. Heiner *et al.* demonstrated an internal readback scrubber by triplicating the ICAP control circuits and by implementing user memory scrubbing to recover from SEUs in the RC's BRAMs [64]. However, the RCs of the approaches detailed in [41, 42, 64] are based on the PicoBlaze processor [175], which does not have an official C compiler and suffers from a very limited instruction space (1,024 words). These RCs cannot therefore readily be reprogrammed to perform new or more sophisticated fault recovery functions [58].

Psarakis *et al.* presented a self-healing RC that is able to perform the reconfiguration process by itself. To enable self-healing, a minimum set of critical instructions and the logic responsible for their execution was hardened by applying TMR or DWC to an OpenRISC processor [129].

3.2.4 Programmable Configuration Controllers

As described in the previous sub-section, a number of investigations have studied techniques for designing fast, light-weight, and easy-to-use RCs for general purpose FPGA-based systems [124]. For space-based applications, in particular, the requirements for low resource utilization (area), high speed and flexibility are not only motivated by, but also

constrained by, the desire to reduce the risk of radiation-induced faults to the RC itself. On the one hand, the more resources devoted to the implementation of the RC, the more area exposed to radiation-induced SEUs, which has a direct impact on the MTTF of the design, the single most important factor determining its reliability. On the other hand, the greater the area of the design in terms of configuration frames used, the longer the MTTR from a configuration memory error via modular reconfiguration or scrubbing. This factor directly impacts on availability, and in a TMR design, upon the reliability of the design as well, since recoveries that are successfully completed before further errors affect the other, correctly functioning replicas, avoid design failure.

It should be noted that system reliability is not solely dependent on the RC's area and speed. There is usually an architectural limit as to how reconfiguration speed influences reliability. For example, in space applications that use slow flash memories for storing bitstreams, the reconfiguration speed is limited by the rate at which the flash memory can be read. Resource utilization is another important criterion. A resource hungry controller introduces more essential bits and is therefore more susceptible to radiation-induced SEUs. Removing unused functionality is an effective means of reducing resource usage but the controller may not remain flexible enough to be reused in a range of applications, or to implement alternative fault detection/recovery algorithms. For example, the surveys in [63, 66, 146] cite a large number of possible scrubbing algorithms. The design space is further enlarged if we also consider TMR-MER as a fault recovery strategy. It is therefore necessary to design an RC that is specifically designed for space-based applications, and that balances the trade-off between performance, resource utilization and flexibility.

To balance this trade-off, Gong *et al.,* proposed a *Programmable Configuration Controller* (PCC) to assist in detecting and recovering from SEU errors in space applications [58]. PCC is a soft *Application Specific Instruction Set Processor* (ASIP) based on the RISC-V instruction specification [169]. It supports all RISC-V integer instructions and benefits from a complete compiler tool chain with a large development community. PCC can run fault detection/recovery software using the general instructions defined by the RISC-V specification while being able to benefit from the high reconfiguration throughput provided by instructions specifically customized for this purpose. The PCC implementation is based on v-Scale [97] and PicoRV [174], two versions of RISC-V that have low resource usage. The PCC can be used in either standalone mode or peripheral mode, as configured at design time by passing Verilog parameters or VHDL generics, to meet different system design requirements. Through 5 case studies, Gong *et al.,* demonstrated that the use of an ASIP architecture for reconfiguration control in applications prone to radiation-induced corruption strikes a good balance between speed, resource utilization and flexibility [58].

However, PCC itself is a single point of failure in a system. Thus, a triplicated PCC was also proposed by Gong *et al.,* [57] to make it more reliable.

3.3 Reconfiguration Control Networks

In this section, we focus on the choice and design of the RCN, which collects reconfiguration requests from the system's TMR voters and communicates these to an internal or external reconfiguration controller [66,146]. The performance and reliability of the RCN is important for a number of reasons. On the one hand, the latency of the RCN has a direct impact on the MTTD errors in the system. On the other hand, if the RCN is implemented in a non-redundant manner, it introduces a single point of failure that compromises system reliability. Furthermore, the RCN can be used to trigger a scrub cycle rather than relying on a periodic or error rate-based trigger. Hence, the results presented in this section are of relevance to any SRAM FPGA-based TMR system irrespective of the type of error recovery method used.

Considering the importance of the RCN in TMR-MER system design, we provide a comprehensive study of the variety of RCNs reported in the literature. The RCN is typically realized by utilizing configurable resources such as *Configurable Logic Blocks* (CLBs) and programmable interconnection resources. However, an RCN can also be implemented using the FPGA's hardwired configuration network. In this work we distinguish between RCNs that are implemented in programmable logic, which we refer to as *soft* networks, and those that are implemented using hardwired non-programmable resources, which we refer to as *hard* networks.

Soft RCNs can be realized with simple star networks [20, 152] or with more complex networks such as bus [109] and token ring networks [28, 192]. In soft networks, routing congestion may occur as the number of TMR components in the system increases. In contrast, hard networks [4, 163] rely on the built-in configuration network of the FPGA to provide access to the state of health of the TMR components. This results in reduced demand for routing resources, and therefore also enhances the reliability of the RCN component.

In this section, we compare four RCNs with respect to reliability, latency, scalability and power consumption. Fault injection experiments are conducted to evaluate the impact of each RCN on system reliability. In a case study that is implemented on the RUSH (*Rapid recovery from SEUs in Reconfigurable Hardware*) payload [30], we demonstrate

that the hard network, which uses the *Internal Configuration Access Port* (ICAP) to read the voter state, achieves the highest reliability. We also show that MTTD is greatest for the ICAP-based approach due to the relatively large latency involved in retrieving user state this way, but we demonstrate an effective optimization that significantly narrows the gap between this hard approach and the soft RCNs. Finally, we assess the reliability of a real system employing module-based recovery relative to the same system using blind scrubbing. Results show that scrub-based error recovery results in higher reliability unless the RCN is itself triplicated and repaired when its configuration becomes corrupted.

The section is organized as follows: sub-section 3.3.1 reviews the literature available on RCN designs, with sub-section 3.3.2 describing the architecture of the various RCN types we studied. Sub-sections 3.3.3 to 3.3.4 describe the fault emulation system we implemented to assess the soft error vulnerability of our designs and the model we used to evaluate the reliability of our implementations. Sub-section 3.3.5 describes our experimental method and reports our findings while concluding remarks and directions for further study are given in Section 3.3.6.

3.3.1 RCN Survey

Several types of networks for aggregating reconfiguration requests from TMR voters have been described in the literature. These include examples of a star network [20,152], a bus network [109], a token ring network [28,192], and an ICAP-based readback approach [163].

Star networks use simple interfaces to directly connect the voter outputs, which are distributed across the device, to a central *Network Controller* (NC) [20,152]. In star networks, the interconnecting wires may need to span the entire device and therefore pass through numerous programmable interconnection resources. This not only increases their susceptibility to SEUs, but also introduces latency. Star networks described in the literature typically involve polling of the remote (voter) interfaces by a central controller implementing a round-robin algorithm. It would be feasible to also consider an interrupt-driven approach whereby voters interrupt a central arbiter to transfer a reconfiguration request.

In [109] the authors utilized the AXI core to transfer the outputs of individual modules to a central voter. While not serving as an RCN, any shared messaging resource, such as this bus, could be used to convey reconfiguration requests from distributed components to a central controller. The use of a shared bus allows new modules to be readily integrated into the system while avoiding the need for dedicated routing resources as used in star networks. However, a bus requires more complex interconnection interfaces than star

networks, which results in an increased soft-error vulnerability, power consumption and latency.

In [28, 192] a token ring network is implemented that spans all voters in a daisy-chained manner. The design uses complex network interfaces that require significantly more logic than the endpoints of the point-to-point connections found in the star networks in [20]. However, token ring topologies usually link neighbouring components and therefore utilize a reduced number of global wires for interconnecting them. In contrast, star and bus topologies realize mixed distance connections and thus utilize various interconnection resources, including both local and global wires. Usually, SRAM-based FPGAs integrate more local than global wires and therefore token ring networks, which tend to utilize more local wires, are considered to be more scalable than star and bus networks. However, in token ring networks, the latency increases with the number of components on the network. Another drawback of this topology is that when a link suffers a configuration memory error the ring no longer functions as intended, whereas the star topology is inherently more robust as all links are independent.

A fourth approach that has been described in the literature makes use of the ICAP to read the outputs of the TMR modules or to read the voter states in a round-robin fashion [163]. The former uses software to centrally compare the outputs of each module in order to reduce the overheads of distributed voting and to reduce the likelihood of the voter mechanism becoming corrupted. An ICAP-based communications scheme eliminates the need for a soft network and therefore reduces routing pressure, implementation time, and improves reliability. Reliability is enhanced since the built-in hard reconfiguration network is utilized to obtain module or voter outputs. This approach has the potential to be scalable as it does not require user routing resources and utilizes a moderate amount of logic to implement the central controller.

3.3.2 RCN Architecture

In this section, we describe the architecture of the four RCN types identified in the literature and provide average latencies for obtaining a reconfiguration request.

Each RCN is composed of distributed *Network Interfaces* (NIs), a central NC and an *interconnection network* between them as illustrated in Figure 3.4. In the figure, each majority voter provides a 2-bit *error info* signal to the corresponding NI. Three possible values of the *error info* signal — 00, 01, and 10 — represent the error states of the triplicated modules respectively, while the value 11 that indicates there is no error. Note

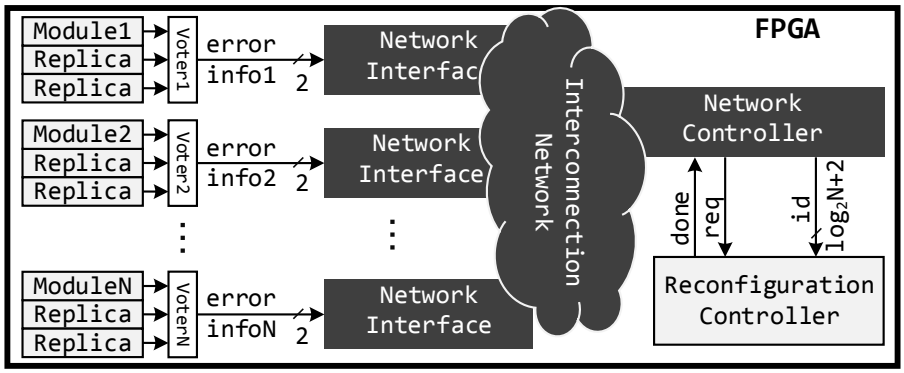


Figure 3.4: Components of an RCN

that a voter raises an RR to the NC by means of the *error info* value. Once the NC receives an RR, it issues a *req* and a $(log_2N + 2)$-bit *module id* signal to the RC, which invokes the MER process to recover the faulty module. (Note that N denotes the number of TMR components in the TMR-MER system.) After MER is completed, the RC asserts a *done* signal to the NC, which is used to commence re-synchronization of the reconfigured module and to resume checking the next component.

3.3.2.1 Star Network

Figure 3.5 illustrates an overview of a typical star network implementation. Each NI contains a 2-bit buffer that connects directly to the NC. The NC consists of two modules, namely an arbiter and a multiplexer. The arbiter selects which voter is to be checked in a round-robin manner while the multiplexer transfers the *error info* from the selected voter to the arbiter. When the arbiter receives an RR, it sends *req* and *module id* signals to the RC. The RC invokes modular reconfiguration for the faulty module before issuing a reconfiguration *done* signal to the arbiter for it to resume voter checking.

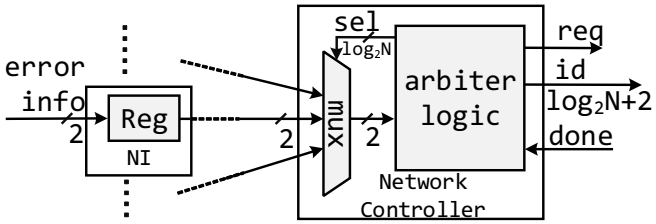


Figure 3.5: The architecture of a star network

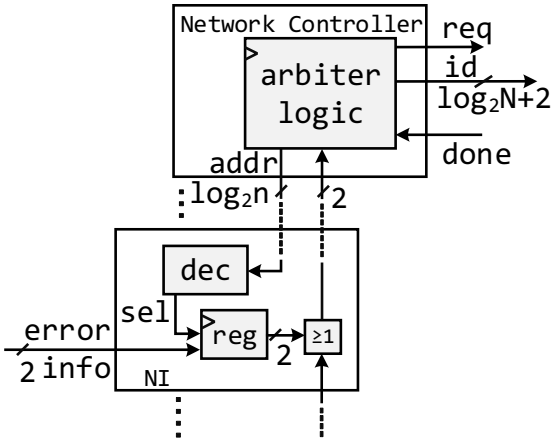


Figure 3.6: The architecture of a bus network

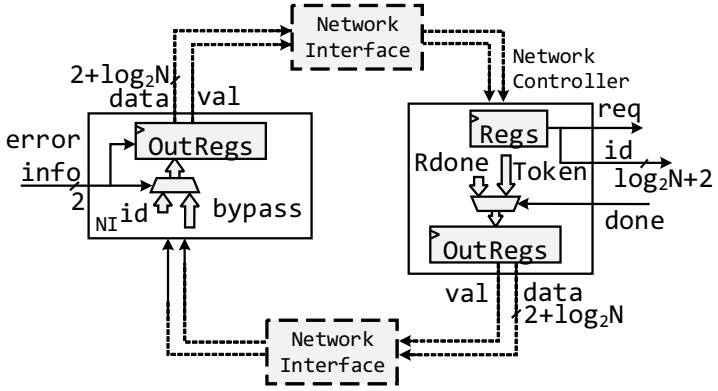


Figure 3.7: The architecture of a token ring network

3.3.2.2 Bus Network

A simple bus network that aggregates the error messages from the voters is illustrated in Figure 3.6. The bus network works similarly to the star network. The multiplexer architecture of the star network is replaced with a global address signal. The registers in the NI are controlled by the *address decoder* (dec) and the register outputs drive the common bus signal through an *OR* gate. If the bus signal indicates an RR, the NC triggers the reconfiguration process in the RC.

3.3.2.3 Token Ring Network

Figure 3.7 shows the basic token ring network architecture. In contrast to the star architecture, the NIs provide a $(log_2N + 2)$-bit message that indicates which module is in error. When the *val* signal toggles and the message on the data signal is a token, the *OutRegs* in the NI latch the token before passing it on to downstream NIs. If an NI needs to send an RR, it keeps the token and its *OutRegs* latch the corresponding *module id*. The downstream NIs pass this message to the NC. The NC operates in a similar manner to that described for the star network. When the reconfiguration of a module has been completed, a *done* message is released to signal to the requesting NI that reconfiguration has finished and that the token should therefore be released.

3.3.2.4 ICAP-based Voter Checking

The ICAP-based readback approach eliminates the soft interconnection network by using a modified RC component to check the TMR component voters directly. The voter *error info* signal is registered and the RC polls these register outputs in a round-robin manner via ICAP configuration frame readbacks. When the RC receives a reconfiguration request, it triggers a reconfiguration operation to correct the faulty module.

Configuration Frame Readback The Xilinx FPGA configuration memory is arranged into frames that are tiled about the device. These frames are the smallest addressable segment of the configuration memory space. The frame size varies among FPGA families; in the case of Xilinx 7–Series FPGAs, it consists of 101 32-bit words. Each frame possesses a unique address that can be used to dynamically read or write to the configuration memory.

Xilinx 4—7 Series devices allow users to read the configuration memory via the ICAP. There are two modes of readback, namely *Readback Verify* (RbV) and *Readback Capture* (RbC) [180]. We use the RbC mode to check the voter state of each TMR component since this mode allows the state of the CLB configuration memory cells to be read. This can be done by issuing a GCAPTURE command to the ICAP so as to sample all CLB register values into configuration memory cells. These values can then be read back along with the configuration frame containing the voter status bits. However, designers must know the frame address and configuration bit offset of the SRAM cell corresponding to the desired output of the voter for this approach to work. These parameters are given in the logic

allocation (`*.ll`) file, which is automatically generated by the Xilinx ISE/Vivado design tools. The logic allocation file includes four fields, namely, a *bit offset*, a *frame address*, a *frame offset*, and *information* for each configured resource as depicted in Figure 3.8. The registers corresponding to the voter status of a TMR component are determined from the information fields that then allow the frame addresses and frame offsets to be extracted.

```
logic allocation file *.ll

<bit offset> <frame addr> <offset> <Information>
Bit 19488835 0x0042021f  3107 Block=SLICE_X4Y48 Latch=AQ Net=voters[6]/status_bits[1]
```

Figure 3.8: Extract of a Xilinx logic allocation file

Xilinx devices expect a specific sequence of commands to be sent to the ICAP in order to read a data frame [180]. A frame read request necessitates the read of a dummy word and a pad frame before the desired data frame can be read. The time to read a frame in Xilinx 7–Series FPGAs is approximately 230 clock cycles. This includes 20 clock cycles for issuing initialization commands, 203 clock cycles for the frame read, and 10 clock cycles for issuing concluding commands [180]. The frame read time depends on the throughput of the ICAP, which supports 32-bit transfers at a rate of 100 MHz. At the maximum rated speed, the time for reading a frame through the ICAP primitive in 7–Series FPGAs is thus approximately 2.3 us. This latency can be reduced if voter registers are placed at the bottom of each clock region so that the voter registers are located at the beginning of the data frame. When this is done, the frame read can be aborted after the voter registers have been read. The frame read time can thus be reduced by as much as 1 us down to about 1.3 us [180].

3.3.2.5 RCN Latency

RCN latency is defined as the average period of time needed for the NC to receive a reconfiguration request from a voter. As described in Section 3.3, all four networks check voters in a round-robin manner. Thus, assuming a system with N TMR components or NIs and one NC, the average latency of the token ring network is given by

$$latency = (N + 1) \times c_{hop} \times \frac{1}{F_{network}}, \tag{3.1}$$

where c_{hop} denotes the number of clock cycles per node hop, and $F_{network}$ denotes the clock frequency of the RCN. Equation (3.1) corresponds to the average time needed for the token to arrive (half the ring) and the time for the request to make it back to the NC (also half the ring).

The RCN latency for all other topologies is given by

$$latency = \frac{N}{2} \times c_{hop} \times \frac{1}{F_{network}}, \tag{3.2}$$

which corresponds to the time it takes to check half the voters in the system, on average, before the one that wishes to raise a reconfiguration request is checked.

3.3.3 Fault Emulation System

In this section, we outline the fault emulation system we implemented to assess the soft error vulnerability of the RCNs we studied. As detailed in Chapter 2, fault emulation involves the use of hardware-based methods to artificially insert faults into FPGAs. A typical fault emulation system needs to provide an ability to access internal memory to inject a fault, an ability to stimulate and execute the circuits, an ability to determine output errors and an ability to clear errors [133].

Figure 3.9 outlines our fault emulation procedure. We use the Xilinx AXI HWICAP IP for flipping configuration memory bits and a MicroBlaze processor to control this process. Once the system has been initialized, the MicroBlaze halts and waits for a fault injection address from a PC host. A uniformly distributed random configuration bit address is generated. The MicroBlaze reads the corresponding frame, flips the addressed bit and writes the frame back using the HWICAP to emulate an SEU. Note that we do not inject faults into either the MicroBlaze or the HWICAP in order to avoid their corruption throughout the fault injection campaign. Of the 18,300 configuration frames in the Artix-7 XC7A200TFBG-484 targetted in our study, 14,250 frames are contained in the region that represents the design under test.

Once a fault is inserted, the circuit is tested using all possible input stimuli. In our case, there are four possible *error info* values that would normally be presented to an NI. An input stimulus is usually provided through external pins on the FPGA. However, we felt that errors injected into the nets leading from the external pins to the NIs of the network would influence the results of our experiments. We therefore developed a different method for stimulating input values. As can be seen in Figure 3.10, the input stimulus of each NI is realized through what we call a "RePin" architecture, which is composed of two SLICEM LUT (LUTM) blocks configured as distributed RAM. Each LUTM provides a single bit of stimulus that can also be changed using the ICAP. Given the site number and logic locations, the positions of the LUTM bits can be obtained from the `*.ll` file as described in Section 3.3.2.4.

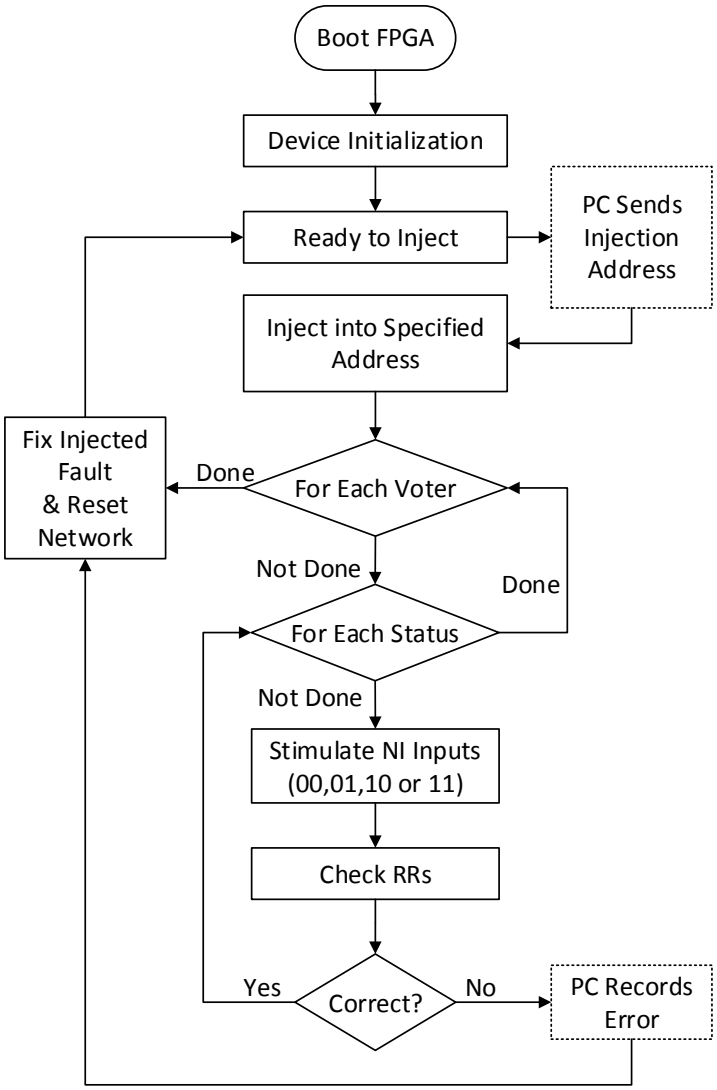


Figure 3.9: Fault injection flowchart

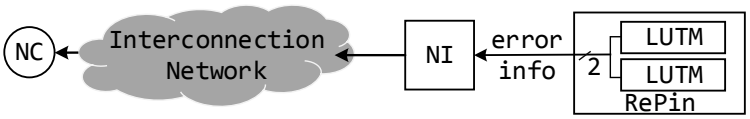


Figure 3.10: RePin architecture for input stimulus

An essential step in the fault emulation procedure involves detecting system errors. The MicroBlaze processor checks the integrity of the design while the LUTMs are manipulated to simulate different *error info* signals arising from a voter. For each NI, we iterate through every possible combination of the *error info* signal while holding the inputs to every other NI constant at "11", which signifies the "no error" condition. Whenever a new *error info* value is written, the MicroBlaze processor checks that the correct RR is received. In the case of the soft networks, we wait for the maximum number of clock cycles required for the NC to receive an RR, then the *req* and *module id* signals are read using the AXI GPIO interface. In the case of the ICAP-based RCN, the MicroBlaze processor utilizes the ICAP to read back the values of each NI's status flip-flops in order to determine the RR. If the RR is as expected, we change the *error info* signal to the next value. When we have cycled through all 4 possible values and there is no unexpected RR, we move on to the next NI. If an unexpected RR is received, an error report is sent to the PC.

The fault emulation tool must also remove the injected fault and return the circuit to a known functioning state before injecting the next fault. In our system, the injected fault is fixed by writing back the frame as it was before injection, all NI inputs are set to "11", the RCN is reset and the software returns to wait for a new fault injection address from the PC.

3.3.4 Reliability Evaluation

The reliability of an FPGA-based system composed of N TMR components that use MER to recover from configuration memory errors and an RCN for aggregating reconfiguration requests can be derived as follows. We model the reliability of the RCN ($R_{RCN}(t)$) using Equations (2.1, 2.2 or 2.3). Respectively, the reliability of each TMR component $R_i^{TMR}(t)$ in the system is modelled using Equation (2.3). Finally, the reliability of the system is given by the product of the reliability of each individual component, namely the RCN and the N TMR components [145]:

$$R_s^{TMR}(t) = R_{RCN}(t) \prod_{i=1}^{N} R_i^{TMR}(t). \tag{3.3}$$

In this derivation, it is assumed that failures follow a Poison distribution and the occurrence of errors in modules or components are statistically independent and uncorrelated. Note that Equation (3.3) holds true only if the recovery rate is much greater than the error rate, which ensures repairs are completed independently [145]. Moreover, since the

main objective of this section is to evaluate the impact of various RCN architectures on the total reliability of FPGA-based designs that incorporate MER, we omit inclusion of the reconfiguration controller and the voters in our reliability analysis as these would have the same impact in each case. In this thesis, we estimate the *relative reliability* of the techniques under study. However, to estimate the *absolute reliability* of the system, the reconfiguration controllers and the voters would need to be included as additional components in the evaluation of Equation (3.3).

3.3.5 Experiments and Results

In this section we evaluate the performance of the networks presented in Section 3.3.2, in terms of resource utilization, latency, operating frequency, power consumption and soft-error vulnerability. All networks were implemented on a Xilinx Artix-7 XC7A200TFBG484-1 FPGA, as hosted on the RUSH experiment board, as described in the Appendix, using the vendor's Vivado 2014.4 implementation tools with default settings. The comparison of the networks is based on data obtained from the implementation tools and also on fault-injection experiments as described in Section 3.3.3.

3.3.5.1 Experiments

As mentioned in Section 3.3.2, an RCN consists of NIs, a central NC and the interconnection network between them. In these experiments the same voter interface and RC designs were used irrespective of the RCN type being tested. The same NI and NC locations were also used for all RCN designs. In the first experiment we studied "synthetic" layouts in which the TMR components, their voters, and thus the NIs were assumed to be distributed in a checkerboard pattern across the majority of the device area. Moreover, the NIs and the NC were always located in partitions that utilized the same FPGA resources irrespective of the RCN topology. To obtain resource utilization and performance results, we initially implemented designs that only contained the components of the RCNs being tested and constrained the implementation tools to prevent optimizations across the port interfaces of the NIs and the NC. To perform the fault injection experiments, we added a MicroBlaze-based RC for injecting faults and distributed RAM-based test vectors to each of the RCNs tested. We tested each RCN type for networks comprising 7, 15 and 31 voters. The synthetic layout of a 31-voter design (in this case for testing the star network topology) is shown in Figure 3.11, in which the design under test into which faults were injected is depicted as the shaded region to the right of the RC.

In the second experiment, we investigated the utilization, performance and system reliability of each RCN when used to collect reconfiguration requests for the RUSH payload. For this case study, we implemented the four network types with the 9 TMR components comprising the RUSH experiment, as described in the Appendix.

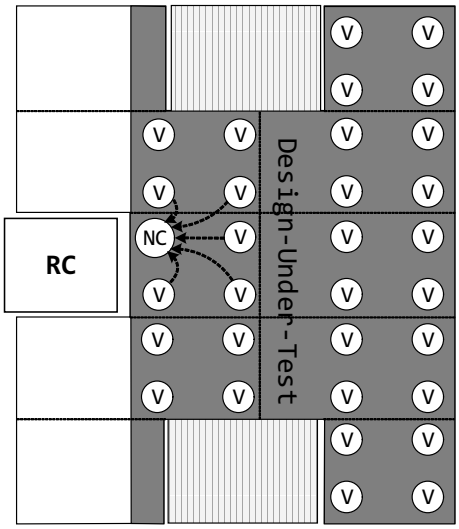


Figure 3.11: Synthetic layout of a 31-voter design

3.3.5.2 Results

Implementation results: Table 3.1 presents information extracted from the vendor's implementation tools. The results are listed according to the resource utilization of the design; the dynamic power consumption and the number of essential bits follow the same pattern. In contrast, the vendor's power analysis tools reported the same amount of static power consumption for all RCN designs. Given that the RCN designs utilized less than 0.2% of the total FPGA resources on average, we believe that the contribution of the RCN to the total static power consumption of the FPGA is negligible, and due to this we have obtained the same result for all RCN designs.

It can be seen from Table 3.1 that the ICAP-based RCN was realized with the fewest resources compared to the other RCN architectures. This is primarily because the ICAP NIs are implemented with just two Flip-Flops (FFs) and a small amount of support logic being mapped to Look-Up Tables (LUTs). As expected, the number of Programmable Interconnection Points (PIPs) and Switch Matrices (SMs) used by the ICAP approach is significantly lower than for the other approaches. As a consequence, the ICAP-based RCN has on average 2.7, 3.6 and 6.0 times fewer essential bits than the synthetic layouts of the

Table 3.1: Results of mapping four RCNs to a Xilinx Artix-7 XC7A200TFBG-484

Type	**ICAP**					**STAR**			**BUS**			**RING**		
Layout	**L1**			**L1***	**L2**	**L1**		**L2**	**L1**		**L2**	**L1**		**L2**
# NIs	7	15	31	31	9	7	15	31	9	7	15	31	9	7	15	31	9
Slices	7	15	31	31	9	12	29	50	18	21	33	60	21	30	50	141	35
LUTs	0	0	0	0	0	14	27	30	16	28	50	108	33	54	130	279	87
FFs	14	30	62	62	18	26	44	77	32	35	61	110	43	61	134	295	87
PIPs	440	889	1,770	1,858	557	1,101	1,996	3,513	1,243	1,341	2,553	4,625	1,729	2,057	3,894	7,986	2,724
SMs	38	62	102	181	55	277	453	792	274	351	616	1074	466	426	496	861	426
Freq. (MHz)	100					112	109	107	126	109	107	104	114	132	203	186	145
Clocks / Hop	230					2			2					1			
# hops	7	15	31	8	9	7	15	31	9	7	15	31	9	8	16	32	10
Latency (us)	8.05	17.25	35.65	9.20	300	0.06	0.14	0.29	0.5	0.06	0.14	0.30	0.5	0.07	0.08	0.18	0.5
Static (mW)	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138	138
Dynamic (mW)	3	4	5	5	3	4	7	7	4	6	7	9	5	4	5	8	4
Ess. bits (Kb)	3.42	5.6	10.4	13.7	4.1	9.4	15.8	26.0	10.1	11.9	20.1	38.6	14.9	18.3	33.7	69.4	24.3

L1: Synthetic layout L2: RUSH layout L1*: optimized ICAP layout

star, bus and ring networks respectively. However, the ICAP-based RCN suffers from high network latency. It requires two to three orders of magnitude more time than the other RCNs to transfer reconfiguration requests to the NC. In contrast, the ring has the lowest latency, since it can achieve a higher operating frequency and only needs 1 clock cycle per node hop. We used Equations (3.1) and (3.2) to calculate the latency for each RCN. The latency of the ICAP approach is on average over 175 times that of the ring and the latency of the star and bus networks was about 1.4 times that of the ring for the synthetic layouts.

We investigated an optimization of the ICAP RCN that entails constraining the registers of those groups of NIs that are located within each clock region. These registers are forced to be placed into a single configuration frame so that they can be accessed in a single frame read. With reference to Figure 3.11, which depicts 4 voters per clock region (the 10 grey rectangles), this optimization resulted in the creation of horizontal wires leading from each voter to a frame that was centrally located in each clock region. Instead of requiring 31 separate frame reads to check all voters, this approach reduced the number of frame reads needed to 8 in total — one for each clock region used by the design. The results of this implementation are reported in Table 3.1 in the ICAP column headed L1*. As can be seen, this optimization reduced the latency of the ICAP approach by a factor of 4 while increasing the number of essential bits used over the unoptimized 31-voter ICAP design by 32%.

Table 3.2: Fault injection results

Type	**ICAP**		**STAR**		**BUS**		**RING**	
# voters	Avg	σ	Avg	σ	Avg	σ	Avg	σ
7	7.0	1.5	8.2	2.3	16.8	2.1	51.0	4.7
15	8.2	3.5	17.0	3.0	36.6	5.0	122.1	16.7
31	20.7	1.4	38.6	4.6	78.6	7.9	213.4	27.3

Avg: average number of observed errors
σ: Standard deviation

Fault injection results: We implemented the fault emulation system described in Section 3.3.3 to conduct fault injection experiments for the synthetic layouts of each of the four RCN types. For each RCN type, we made 5 trials, in each of which we injected one million faults. Table 3.2 tabulates the average number and standard deviation of observed errors per one million injected faults. These results demonstrate that the ICAP-based RCN is more reliable than the other approaches. Additionally, the number of errors that

Table 3.3: Results of mapping 9 TMR components to a Xilinx Artix-7 XC7A200TFBG-484

Design	**Essential Bits (n_e)**	**Failure rate (λ_m) (upsets/s/module)**	**n_f**	**t_c (ms)**
FIR	12,042 (0.02%)	3.25×10^{-6}	65	1.2
FIFO	41,842 (0.07%)	1.13×10^{-5}	192	3.5
BAQ	48,963 (0.08%)	1.32×10^{-5}	73	1.3
BST1	281,604 (0.46%)	7.60×10^{-5}	145	2.6
SR1	285,914 (0.46%)	7.72×10^{-5}	378	6.8
SR2	515,904 (0.84%)	1.39×10^{-5}	474	8.5
BST2	793,534 (1.30%)	2.14×10^{-4}	610	11.0
SR3	1,403,647 (2.30%)	3.79×10^{-4}	1,090	19.6
BST3	1,833,235 (3.00%)	4.95×10^{-4}	1,483	26.7

occur in each RCN is directly proportional to the number of voters and thus the number of essential bits per design.

RUSH case study results: Table 3.3 lists details of the 9 TMR designs of the RUSH payload. These include a single MAC-based 21-tap Finite Impulse Response (FIR) filter with 16-bit signal width, an 8-to-3-bit Block Adaptive Quantizer (BAQ), an 8,096-word deep 32-bit FIFO, three 32-bit Shift Registers (SRs) having different lengths and a range of combinational logic between the stages, and three 32-bit Binary Search Trees (BSTs) of different heights and a variety of combinational logic at each node (please refer to Appendix A.2.2 for additional details of these designs). Table 3.3 also presents the number of essential bits (n_e), the number of frames (n_f) and the correction time (t_c) of each TMR module. The failure rate of each module is calculated assuming $\lambda_{bit} = 2.7 \times 10^{-10}$ upsets/bit/s, which is a high failure rate that corresponds to the peak-5-min condition at GEO, as shown in Table 2.2. This high failure rate was chosen to clearly show the reliability trends in Figure 3.12. We observed similar trends for lower failure rates when time t in Equation 3.3 was increased. Note that in our design, since we were using the AXI HWICAP, the ICAP throughput using the MicroBlaze was limited to 10 MB/s, considerably less than the maximum possible throughput of 400 MB/s, because the AXI bus requires multiple clock cycles to execute an instruction and since the bandwidth to the external flash memory is limited. This reduced ICAP throughput increases the latency for checking a voter using the ICAP to 60 us per voter, and we therefore observe a much higher network latency under these conditions.

Figure 3.12 plots the system reliability for each RCN type and the 9 RUSH applica-

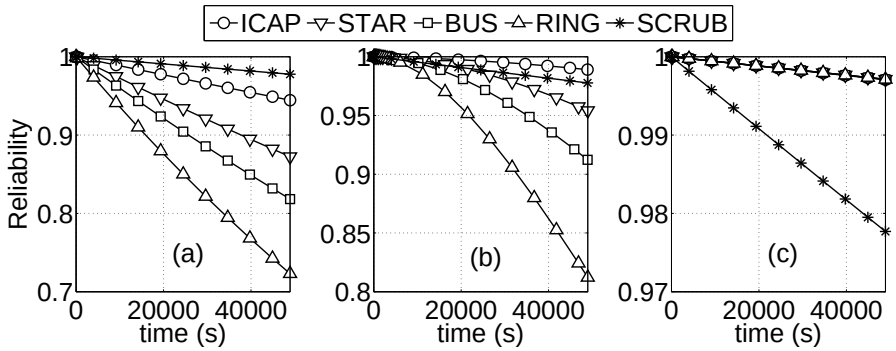


Figure 3.12: Reliability of RUSH payload using (a) Unprotected RCN, (b) TMR triplicated RCN, and (c) TMR triplicated RCN with recovery

tion circuits using Equation (3.3) against the reliability of a blind scrub implemented on the same system. The MicroBlaze RC and off-chip flash configuration storage used by the RUSH system supports a random FPGA configuration frame read latency of 60 us and a sustained frame write period of 18 us per frame. Blind scrubbing, which entails rewriting each configuration frame of the device, therefore takes 330 ms on the Artix-7 XC7A200TFBG-484 used in the RUSH payload, and errors are recovered by scrubbing after 165 ms on average. Please note that in Figure 3.12, the scrub plots only account for the 9 application components; they specifically exclude the RCN component, which is not needed for blind scrubbing.

Figure 3.12(a) assumes the four RCNs are implemented as single, non-replicated components. While the ICAP RCN results in the best reliability for MER, all 4 RCNs weigh down the reliability of the system because they are single points of failure.

Figure 3.12(b) assumes the RCNs are implemented as triplicated components, but that the errors that occur in this component are not repaired. The RCN triplication is done by triplicating the voters and connecting each voter to an RCN. For the ICAP-based RCN, the statuses of triplicated voters can be constrained to one frame so that they are read by the RC at the same time. Some limited error mitigation is therefore in place. Only the ICAP outperforms scrubbing over the time period shown. However, eventually (when $t > 120,000$ s) even this approach succumbs to errors that remain unrepaired and scrubbing once again dominates.

In Figure 3.12(c) we assume that the device is partially reconfigured in its entirety when an error in the triplicated RCN component is detected. This error recovery period is longer than desired, but the approach ensures any error in the network is corrected. Despite the long recovery time (equivalent to reconfiguring the complete device), the reliability

is not significantly affected because errors occur infrequently in the relatively small RCN components.

The results indicate that the system relying on the ICAP-based RCN, despite being affected by severely reduced voter checking frequency, has significantly better reliability than those utilizing one of the three soft network topologies studied. It should be noted that the reliability comparison between the ICAP-based voter checking approach and the other RCNs is not completely fair because the RC in the ICAP-based voter checking approach requires more instruction and data memory to store program and voter addresses for the readback operation. However, this can be alleviated if the RCs of the four approaches are implemented using a radiation-hardened processor.

3.3.6 Final Remarks for Reconfiguration Control Networks

In this section, we have compared four RCN types in terms of reliability, scalability, resource utilization, power consumption and sensitivity to configuration memory errors. The utilization and performance of these RCNs were assessed for networks with 7, 15 and 31 voters. The results demonstrate that the ICAP-based readback approach, which uses the built-in reconfiguration mechanism available in FPGAs, requires the least resources of those networks studied.

The results of a case study that was implemented on the RUSH payload and of fault injection testing indicate that the ICAP-based readback approach has the highest system reliability despite having a relatively high latency. This higher latency may not be too problematic except when radiation levels become much higher than the checking rate assumed in our work or the number of components to be checked becomes very large. It should also be noted that the time to read one frame using this approach is far smaller than the time to recover the module when an error is found. We have shown that the latency of the ICAP approach can be reduced by clustering the registers that are to be read from one clock region into a single frame. This optimization does not have a significant impact on the resource utilization. We have also determined that for the reliability of MER to be competitive with scrubbing in a real system, the RCN must also be triplicated and repaired when errors affect it.

3.4 Summary

This chapter has provided the possible implementation of a TMR component and the survey of RCs as well as a thorough study of RCNs. We have concluded that the use of a hard network has the highest system reliability despite having a relatively high latency. However, it should be noted that the high latency of the ICAP-based network, and the relatively high MTTD that follows, is insignificant compared to the MTTR errors when either scrubbing or MER is used. We have also concluded that a TMR-MER approach is less reliable than TMR-Scrubbing unless the RCN is implemented with redundancy and repaired when it suffers from configuration memory errors.

One direction for further study is to consider the order in which TMR components are checked, as explored in Chapters 4 and 5. Further work is envisaged to derive comprehensive reliability models for complete TMR-MER systems, as described in Chapter 5.

Chapter 4

Dynamic Scheduling of Voter Checks in TMR-MER Systems

As mentioned in Chapter 3, implementing *TMR with Module-based configuration memory Error Recovery* (TMR-MER) requires a *Reconfiguration Control Network* (RCN), which is an infrastructural component that collects status messages from the system's TMR voters and communicates these to an internal or external *Reconfiguration Controller* (RC) [66, 146]. Typically, the RCN involves an arbiter that periodically collects the status messages from TMR voters in a predetermined order, usually in a round-robin manner [4, 28, 152, 163, 192]. However, doing so increases the response time in checking the voters of highly vulnerable TMR components. Intuitively, the rate at which a TMR component requests error recovery depends on its failure rate. Therefore, the RCN arbiter should check components that are likely to suffer higher failure rates more frequently than those with lower failure rates in order to minimize the chance that the system fails.

In this chapter, we propose and evaluate a *Voter Scheduling Engine* (VSE) that dynamically prioritizes and manages the voter checks in a TMR-MER system. The proposed VSE is based on the idea that the TMR component that is most likely to have suffered an error since the last check should be checked next. Moreover, we incorporate the VSE into an RCN that utilizes the *Internal Configuration Access Port* (ICAP) available in advanced Xilinx devices to readback configuration frames that contain the health status of the system's TMR components [4, 163] (so-called ICAP-based RCN). As we have demonstrated in Chapter 3, an ICAP-based RCN requires the least programmable resources and therefore provides the highest system reliability against SEUs compared to the other networks that have been reported in the literature. Notwithstanding, the proposed VSE can be

applied to any RCN that is able to randomly check voter status messages. In particular, a VSE-based RCN could be used to trigger a scrub cycle rather than solely relying on periodic scrubbing. Hence, the results presented in this chapter are of relevance to any SRAM FPGA-based TMR system supporting random voter checks irrespective of the type of error recovery method used.

We assess and compare the reliability of a TMR-MER system in which the TMR voter states are checked in a round-robin fashion with that of the same system implementing VSE. We demonstrate that TMR-MER systems that incorporate the VSE are generally more reliable than those using a round-robin order. This is especially the case when the period between two successive checks is increased e.g. when there is an increased number of TMR components to check or when the check frequency is reduced for the purpose of saving energy. Results show that the failure probability of the TMR system incorporating VSE is up to 50% lower than that of the same system using round-robin voter checks during a simulated 30-day mission at GEO and also during a simulated 10-year mission in LEO.

The chapter is organized as follows: Section 4.1 describes our proposed VSE, its role in an ICAP-based RCN, and how it can be implemented. Section 4.2 describes our experimental method and reports our findings, while conclusions and directions for further study are given in Section 4.3.

4.1 Scheduling Voter Checks

In this section, we detail a VSE along with its implementations in either software or hardware, that respectively require linear or logarithmic time to determine which TMR component to check next.

4.1.1 Voter Scheduling Engine (VSE)

Since the ICAP interface can only read one configuration frame at a time [180], using an ICAP-based RCN, the voter status bits cannot be readback in parallel. In principle, they must be read sequentially. Nevertheless, it should be possible for the checks to be scheduled so as to statistically decrease the error detection time and thereby improve the reliability of the system. To achieve this, a TMR component is checked based on the period of time since it was last checked and the number of sensitive bits it contains. We

use the number of essential bits [89] reported by the vendor's tools as a worst case estimate of the number of sensitive bits.

Scheduling is based on the idea that the likelihood of a configuration memory error being present in a TMR component is proportional to the product of the number of essential bits contained in the component and the amount of time elapsed since it was last checked. Therefore, the components with a large number of essential bits are checked more frequently than those with less essential bits. The VSE is designed to implement this idea.

In more detail, the component that is selected to be checked next is the one that currently has the least probability that all three modules are working. We assume that the three modules of a TMR component contain the same number of essential bits and that SEUs are statistically independent and considered to follow a Poisson distribution. The probability that the three modules of component i are still working after time τ is given by [145]:

$$P_i(\tau) = e^{-3\lambda M_i \tau},\ i = 1..N, \tag{4.1}$$

where λ denotes the bit error rate, M_i denotes the number of essential bits of a module of component i, and N denotes the number of TMR components in a system.

For a constant bit error rate λ, $P_i(\tau)$ is the same over any fixed period τ. In other words, in the arbitrary time window between two consecutive checks of component i, we can consider that $P_i(\tau)$ starts from 1 when the voter of component i is checked and no error is detected ($\tau = 0$) and that $P_i(\tau)$ decreases until the next check. $P_i(\tau)$ can therefore be estimated by

$$P_i(\tau) = e^{-3\lambda M_i \tau} = e^{-3\lambda M_i n_i \Delta t_o}, \tag{4.2}$$

where n_i denotes the number of checks of other components that have been performed since the last time component i was checked and Δt_o denotes the period between any two successive checks of any pair of components (assumed to be a constant).

Ideally, we want the VSE to find the next component to check in constant time. However, since $P_i(\tau)$ changes non-linearly and dynamically, it is difficult to determine the component with the smallest $P_i(\tau)$ at the current moment in constant time.

To illustrate this problem, consider three components I, II and III, with varying $P_i(\tau)$ as depicted in Figure 4.1. Assuming that $P_I(\tau)$, $P_{II}(\tau)$ and $P_{III}(\tau)$ are in ascending order and $P_i(\tau)$ of other components in the system are shown as being contained within the dashed region of the graph area, the voters associated with components I, II and III are checked at times $n\Delta t_o$, $(n+1)\Delta t_o$ and $(n+2)\Delta t_o$, respectively. After each check, $P_I(\tau)$, $P_{II}(\tau)$ and $P_{III}(\tau)$ are reset and proceed to decrease in the next time window according

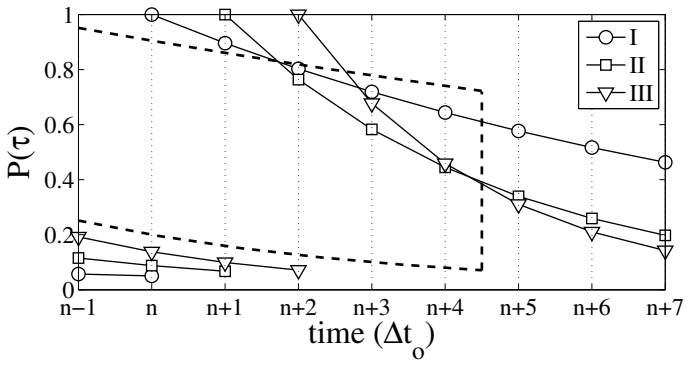


Figure 4.1: Example of component records changing places.

to their error sensitivity. After component III is checked, many other components may have smaller $P_i(\tau)$ and they will need to be checked in subsequent cycles. As can be observed from Figure 4.1, $P_{III}(\tau)$ is higher than $P_{II}(\tau)$ at time $(n+3)\Delta t_o$, however, this relationship changes at time $(n+5)\Delta t_o$. On the other hand, $P_i(\tau)$ of the most recently checked component will always be smaller than that of less recently checked components that were more reliable when it was checked. For example, $P_{II}(\tau)$ is smaller than $P_I(\tau)$ at time $(n+2)\Delta t_o$. Since $P_{II}(\tau)$ decreases faster (higher failure rate) than $P_I(\tau)$, the relationship of components II and I will never change until component II is again checked.

In this work, the VSE simply searches the list of $P_i(\tau)$ of all TMR components to obtain the component with the smallest $P_i(\tau)$, i.e. the component with the highest probability that at least one of the three modules of the TMR component will fail during the next observation period, and therefore should be checked next.

When a configuration memory error is detected in a TMR module through the voter of the component, MER is invoked to reconfigure the faulty TMR module so as to correct the error. During the reconfiguration period, the ICAP cannot be used to read voter statuses according to the usual schedule since it is occupied reconfiguring the module that was found to be in error. Moreover, since the reconfiguration period of a TMR module may be much longer than the time period between voter checks, the likelihood that TMR components have suffered errors will have increased during this period. A feasible approach is to recommence the voter checks starting with the voter of the component with the largest M_i followed by all the successively smaller ones. This is sensible because the larger the component's M_i, the higher the probability that errors will have occurred in it after a substantial period of time. Once the voters of all components have been checked and no errors have been detected, the system resumes the VSE-based schedule.

4.1.2 VSE Implementations

The VSE plays the role of the arbiter of the ICAP-based RCN. The interface between the VSE and the RC can be seen in Figure 4.2. The VSE selects the component that needs to be checked next and asserts the component ID to the RC, while the RC reads the associated configuration frame and checks the status message of the voter(s) associated with the component ID. If the RC does not detect a reconfiguration request from the voter of the asserted component ID, it sends an *Update* signal to the VSE, which commences the search for the component to check during the next cycle. On the other hand, if the RC detects a reconfiguration request, then it commences reconfiguring the faulty module in order to correct the error. After the reconfiguration has been completed, the RC checks all components in descending order of their M_i before it resets the VSE and returns to the normal VSE schedule.

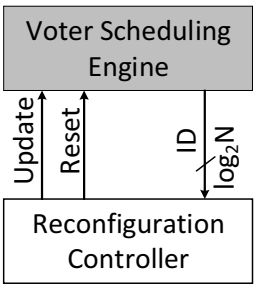


Figure 4.2: Interface between the VSE and the RC.

In the VSE implementation, we use the $M_i n_i$ product in terms of Δt_o units as a proxy for representing $P_i(\tau)$ in Equation (4.2) as it is inversely proportional to $P_i(\tau)$. The $M_i n_i$ product ranks the records in the list of N elements and eliminates the overhead of calculating the exponential function and multiplications. The algorithm involves two steps, which include: (1) finding the largest $M_i n_i$ product, and (2) updating the $M_i n_i$ product for each node. These steps require $O(N)$ time in sequential software.

With the proposed hardware implementation, as depicted in Figure 4.3(a), $N-1$ *Conditional Blocks* (CBs) are connected together as a binary tree in which N records are stored at the leaf nodes. Each record consists of the current $M_i n_i$ product and an identification (ID_i) corresponding to component i. The ID_i are assigned to components according to their M_i values with the component with the largest M_i being assigned 1. The CBs that are connected to the leaf nodes, and which are referred to as the leaf CBs, perform two functions, which involve: (1) comparing the two records and forwarding the record with the greater $M_i n_i$ value to the node in the next level-up, and (2) updating the records at

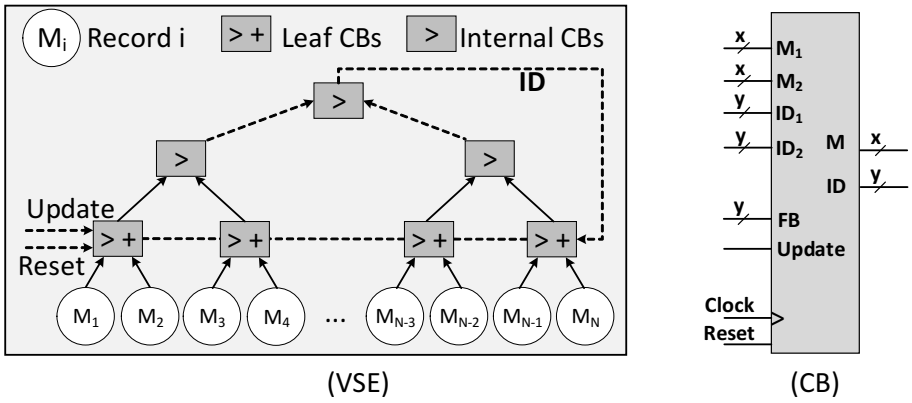


Figure 4.3: (a) VSE and (b) Conditional Block interface.

the leaf nodes. The internal CBs only perform the first function. If the two records contain the same $M_i n_i$ values, the component with the smaller ID_i is chosen. The record with the greatest $M_i n_i$ product reaches the root CB after $log(N)$ steps. The root CB asserts the ID of the selected component to the RC and to the leaf CBs. These CBs update their records after they receive an *Update* signal from the RC. The record of the component that was just selected for checking is reset, while all other records are incremented by their corresponding M_i amount.

The CB interface is depicted in Figure 4.3(b) where the x-bit signals M_1, M_2 and M denote the $M_i n_i$ products, and the y-bit signals ID_1, ID_2, and ID represent the component IDs. x depends on the size of the $M_i n_i$ product while y depends on the number of TMR components in a system.

Since the VSE schedules voter checks based on the $M_i n_i$ product, the M_i values can be modified by the user to impose user-defined priorities, if so desired. One limitation of the VSE is that when the number of TMR components is increased, the search time increases logarithmically while the hardware overhead increases linearly as shown in Section 4.2.2.

4.2 Experimental Analysis

In this section we evaluate the VSE in terms of resource utilization, operating frequency, and scalability. We also evaluate and compare the reliability of a TMR system that checks voters in a round-robin fashion with one that uses the VSE to schedule voter checks. The systems were implemented on a Xilinx Artix-7 XC7A200TFBG484-1 FPGA using Vivado 2014.4 CAD tools.

4.2.1 Experiments

We conducted two types of experiments. In the first, the VSE is implemented as a standalone module on the Artix-7 FPGA in order to obtain resource utilization and performance results. We tested the VSE with systems containing 4, 8, 16 and 32 TMR components.

In the second set of experiments, we compared the reliability of the three configurations as listed in Table 4.1. In the first configuration, we checked the voters in round-robin order. In the second configuration, we assumed that the VSE is implemented in an off-chip radiation-hardened device. And in the third and final configuration, we implemented the VSE on-chip as an additional triplicated system component.

Table 4.1: Configurations for the second set of experiments

Configuration	**# of TMR components**	**Voter Checking**
I	9	Round robin
II	9	Off-chip VSE
III	10 (including the VSE)	On-chip VSE

To test these approaches, three system configurations comprising the 9 TMR components of the RUSH experiment described in the Appendix and listed in Table 4.3 were implemented. Note that the VSE was only included as a TMR component in the third configuration. The VSE used in the second and third configurations was configured with $x = 10$ and $y = 4$. A MicroBlaze (MB) processor was used to implement the RC and the AXI HWICAP IP was used to reconfigure faulty modules in the three configurations.

We injected faults into each TMR component in order to verify the operations of the MB and to measure the correction time (t_c) for each TMR module. The MB controls the fault-injection procedure, which involves reading the corresponding frame, flipping one bit, and writing the frame back using the HWICAP. After injecting a fault, the MB commences checking the voter statuses of all TMR components and reconfigures any faulty module as needed. The correction time accounts for the time interval from the error being detected until the last word of the partial bitstream is written to the HWICAP. Note that the recovery time of a TMR component is the sum of the detection time, the correction time and the synchronization time. In our study, we omitted the synchronization time as it normally only accounts for a small fraction of the recovery time.

Table 4.2: Area and performance of the VSE mapped to a Xilinx Artix-7 XC7A200TFBG-484 FPGA

x – y	N	Slices	LUTs	FFs	Freq (MHz)
32 – 5	32	894 (2.53%)	1922 (1.43%)	1527 (0.57%)	104
32 – 4	16	579 (1.73%)	1595 (1.19%)	968 (0.36%)	117
32 – 3	8	282 (0.84%)	749 (0.55%)	454 (0.16%)	141
32 – 2	4	114 (0.34%)	305 (0.22%)	196 (0.07%)	191
16 – 5	32	419 (1.25%)	1032 (0.77%)	775 (0.28%)	122
16 – 4	16	314 (0.93%)	832 (0.62%)	488 (0.18%)	164
16 – 3	8	146 (0.43%)	380 (0.28%)	230 (0.08%)	197
16 – 2	4	58 (0.16%)	154 (0.11%)	100 (0.03%)	270
8 – 5	32	232 (0.69%)	583 (0.43%)	399 (0.14%)	130
8 – 4	16	158 (0.47%)	426 (0.31%)	248 (0.09%)	174
8 – 3	8	77 (0.23%)	298 (0.14%)	118 (0.04%)	220
8 – 2	4	31 (0.09%)	79 (0.05%)	52 (0.01%)	271

4.2.2 Results

4.2.2.1 VSE Component

Table 4.2 presents the resource utilization and maximum clock frequency of the VSE as the number of TMR components and the sizes, x and y, of M_i and ID_i are varied. As the number of TMR components increases, the resources needed to implement the VSE increase almost linearly. As can be seen in Table 4.2, the VSE utilises a small amount of resources and can readily handle a design with 32 components, in which $x = 32$ and $y = 5$. This small footprint ensures that the placement and routing of other components is not affected and that the VSE can easily be TMR protected for a small overhead.

So as to reduce the wire requirements of the VSE search tree, we normalize the M_i value to the size of the smallest TMR component. This means that the $M_i n_i$ product does not overflow an 8-bit or a 16-bit value. In this case, the CBs with $x = 8$ or $x = 16$ can be used to implement the VSE. As can be observed from Table 4.2, the resource utilization of the VSE with $x = 8$ and of the VSE with $x = 16$ is reduced by approximately 75% and 50%, respectively, with substantially higher clock frequencies than for the VSE with $x = 32$.

In addition, the proposed VSE can update and find which component to check next in $O(log\ N)$ clock cycles. For example, the VSE only needs 5 clock cycles to find the next component to check in a 32-component system. This time is much quicker than the time

Table 4.3: Results of mapping 10 TMR components to a Xilinx Artix-7 XC7A200TFBG-484 FPGA

Design	**Essential Bits M_i**	**Norm M_i** (relative to FIR)	**n_f**	**t_c (ms)**	**t_d (Δt_o)**		
				MBlaze	**I**	**II**	**III**
BST3	1,833,235	152	1,483	26.7	4.5	1.7	1.7
SR3	1,403,647	117	1,090	19.6	4.5	1.7	1.7
BST2	793,534	66	610	11.0	4.5	3.3	3.4
SR2	515,904	43	474	8.5	4.5	4.8	4.8
SR1	285,914	24	378	6.8	4.5	8.2	8.4
BST1	281,604	23	145	2.6	4.5	8.2	8.4
BAQ	48,963	4	73	1.3	4.5	41.0	41.3
FIFO	41,842	3	192	3.5	4.5	56.0	54.7
FIR	12,042	1	65	1.2	4.5	164.4	157.4
VSE	51,293	4	144	2.6	–	–	41.3

needed to check voters using the ICAP-based RCN when both the RC and the VSE operate at 100 MHz (the maximum clock frequency to access the ICAP). The hardware implementation of VSE can scale in size and still be expected to promptly return the ID of the next component to check even with the ICAP operating at maximum frequency. For example, a VSE for a very large system comprising 128 components returns the next ID to check in the minimum time needed to fetch one voter status, as long as the VSE is operated at 7 MHz or faster.

4.2.2.2 Case Study Results

Table 4.3 records the number of essential bits (M_i), the normalized M_i (ratio of each component's M_i relative to the component with the smallest M_i), the number of frames (n_f), the correction time (t_c) and the average detection time (t_d) in number of observation periods (Δt_o) for each TMR component for the three configurations studied. These TMR components include the VSE design and the 9 user application designs of the RUSH payload (see Appendix A.2.2 for details of the designs). In our design, the MB-based RC and off-chip flash configuration storage support a sustained frame write period of 18 us per frame which was used to measure the t_c of each component in Table 4.3.

Figure 4.4 plots the failure probabilities of the three configurations during a 30-day mission in GEO orbit at the peak 5-min radiation condition as Δt_o was varied using Equation (2.8). The bit failure rates were obtained from Table 2.2 as shown in Chapter 2. For clarity, we

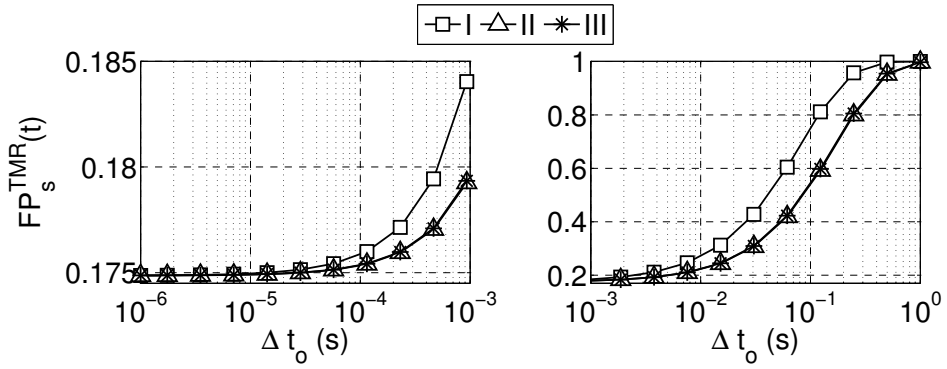


Figure 4.4: Failure probabilities of the three configurations in GEO orbit at the peak 5-min condition during a mission of 30 days.

have plotted 2 sub-figures in which Δt_o ranges from 1 us to 1 ms and then from 1 ms to 1 s. As can be seen in Figure 4.4, configuration II is generally less vulnerable than configuration I. As the observation period (Δt_o) is increased, the gap between the failure probabilities of the two configurations increases, but eventually the gap decreases because of the failure probabilities of the two configurations tend towards 100%. Moreover, the reliabilities of configurations II and III are similar as errors occur infrequently in the relatively small VSE component.

Figure 4.5(a) presents the failure probabilities of configuration III during a 30-day mission for the four GEO orbit conditions as Δt_o is varied. It can be observed that similar system failure probabilities can be achieved at different conditions by adjusting Δt_o. This implies that depending upon the environment that the system is operating in, we can vary the operating frequency of the voter checks to save energy, while maintaining a desired reliability.

Figure 4.5(b) depicts the percentage decrease in the failure probability of configuration III relative to that of configuration I during a 30-day mission for the four GEO orbit conditions as Δt_o is varied. It can be seen that the percentage decrease is similar at different GEO orbit conditions when Δt_o is less than 0.01 second. For the peak 5-minute condition, the highest percentage decrease is 30%, while it is 45% for the worst day and 50% for the worst week and solar min conditions. However, as Δt_o is increased, the percentage decreases under all conditions eventually decline to zero because the failure probabilities of both configurations approach 1. Similar results are also obtained for longer missions in the three orbits. For example, the percentage decrease is up to 50% for four radiation conditions for a 10-year mission in LEO orbit, while it is also up to 50% for solar min and

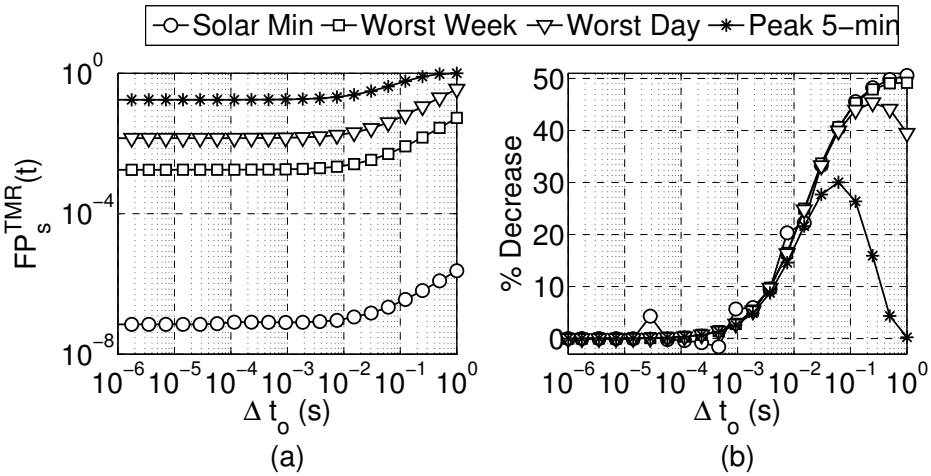


Figure 4.5: (a): Failure probabilities of configuration III with four radiation conditions in GEO orbit during a 30-day mission, (b) Percentage decrease in the probability of failure in configuration III versus configuration I.

worst week, 40% for worst day and 10% for peak 5-min conditions for a 1-year mission in GPS orbit.

Figures 4.4 and 4.5 suggest that one needs to examine the duration of a mission and the voter checking frequency in order to evaluate the robustness of TMR systems that employ either voter checking approach. For the sake of energy saving in space-based applications in long-term missions, the checking frequencies may be decreased [66]. In this case, the VSE is capable of ensuring a higher reliability than round robin.

Table 4.4 shows the failure probabilities of configurations I, II, and III during a 30-day mission in the GEO, GPS and LEO orbits. In this table, we present the failure probabilities of the TMR-MER system when $\Delta t_o = 10$ ms. To compute the values in Table 4.4, we used Equation (2.8) in which the corresponding bit failure rates were obtained from Table 2.2. At these operating points, we can confirm that configurations II and III are generally more reliable than configuration I and that they have similar reliability. Furthermore, the failure probabilities of the three configurations operating in LEO orbit are two to three orders of magnitude less than those in GPS and GEO orbits.

Please note that for the comparison of the VSE and round-robin approaches, the failure probability models are only based on the number of TMR components in the system. As explained in Section 3.3.4, we have not included the RC and the voters in our reliability estimates since they have the same impact on the failure probability of the two systems.

Table 4.4: Failure probabilities of the three configurations when $\Delta t_o = 10$ ms

Orbit\Configuration		**I**	**II**	**III**
GEO	Solar min	$1.23E-7$	$1.00E-7$	$1.03E-7$
	Worst week	$2.89E-3$	$2.31E-3$	$2.32E-3$
	Worst day	$2.28E-2$	$1.83E-2$	$1.83E-2$
	Peak 5-min	$2.68E-1$	$2.21E-1$	$2.22E-1$
GPS	Solar min	$9.98E-8$	$7.79E-8$	$7.81E-8$
	Worst week	$8.75E-4$	$6.99E-4$	$7.02E-4$
	Worst day	$9.98E-3$	$7.98E-3$	$8.02E-3$
	Peak 5-min	$1.23E-1$	$0.99E-1$	$0.99E-1$
LEO	Solar min	Negligible		
	Worst week	$2.10E-6$	$1.68E-6$	$1.69E-6$
	Worst day	$2.33E-5$	$1.86E-5$	$1.87E-5$
	Peak 5-min	$3.03E-3$	$2.42E-3$	$2.43E-3$

4.3 Final Remarks on Dynamic Scheduling of Voter Checks in TMR-MER Systems

In this chapter, we proposed the use of a VSE to schedule the voter checks in an FPGA-based TMR-MER system that employs an ICAP-based RCN. The VSE plays the role of an arbiter in prioritizing checks of critical components. We have also pointed out that any system that uses an RCN that provides random access to component voters can benefit from using a VSE to prioritize the checks of more vulnerable components. We have presented practical algorithms for checking voters in both hardware and software to maximize the system reliability.

A TMR-MER system that includes a VSE is generally more reliable than one that checks voters in a round-robin fashion. The failure probabilities of the former are up to 50% lower than those of the latter for a 30-day mission in GEO orbit with solar min and worst week conditions, and up to 45% and 30% lower with worst day and peak 5-min conditions, respectively.

The proposed approach is fair since each voter will eventually be checked. By modifying the record of the number of essential bits, we can impose a user-defined priority with which to check the voters. The cost of our method is the design and hardware cost of the moderately complex VSE.

Chapter 5

Static Scheduling of Voter Checks in TMR-MER Systems

In Chapter 4, we proposed an on-chip *Voter Scheduling Engine* (VSE) to help the *Reconfiguration Controller* (RC) dynamically adjust the order in which *Reconfiguration Requests* (RRs) from TMR voters are checked based on the likelihood of the next component that is to be checked being in error. The approach was implemented based on the idea that the RRs from the more vulnerable components, i.e., those comprising a greater number of essential bits [89], are checked more frequently than the less vulnerable ones. We found that VSE was able to boost system reliability by up to 50% relative to a static round-robin voter checking schedule. A question that work raised, and which we in this chapter answer in the affirmative, is whether a static voter checking schedule could be found to enhance TMR-MER system reliability by a similar margin or even beyond that possible with the dynamic voter checking method.

It has been noted that while TMR-MER is generally effective for mitigating SEUs affecting the configuration memory [4], it is not well suited to protecting systems against multiple coincident SEUs that affect multiple modules of a TMR component and that thereby defeat the protection afforded by redundancy. In this chapter, we investigate the reliability of TMR-MER systems consisting of multiple triplicated components operating in harsh radiation environments, such as in geosynchronous orbit during solar flares, and in high-energy physics laboratories, like the Large Hadron Collider, where multiple coincident SEUs are more likely [123]. Our main interest in this chapter is in determining the impact of varying the order and rate at which the voter RRs of TMR components are checked for errors on overall system reliability.

Reliability models for TMR-MER systems have not yet been studied in detail. When they are mentioned, Markov models are used to compute the system reliability with the assumption that the recovery of modules of multiple TMR components occurs independently [4]. While acceptable at low error rates, the problem with this assumption at high error rates is that the methods for correcting configuration memory errors are inherently sequential, hence the models do not consider the effect of configuration memory errors on other TMR components while a faulty module is being reconfigured. In this chapter, we develop reliability models that consider multiple coincident SEUs that may occur in different TMR components and use these to analyze the impact of the order in which we check voters for reconfiguration requests.

Several models for estimating the reliability of SRAM FPGAs in the presence of SEUs have been introduced in the literature. Heron *et al.* introduced a reliability model in which the overall reliability of the FPGA is calculated based on physical reliability and SEU reliability [65]. This model parses the netlist of a design to estimate the number of essential bits used for configuring essential items such as LUTs, MUXs, FFs/latches, wires and switch resources in the design. However, the model does not consider the effect of hardware redundancy or the effect of multiple coincident SEUs. Edmonds presented a reliability model of TMR designs without recovery that considers coincident upsets [45]. Ostler *et al.* introduced a reliability model for a one-component TMR design employing TMR-Scrubbing under harsh radiation environments where multiple coincident SEUs are more probable [123]. Their model requires orbit- and condition-specific SEU rates and design-specific estimates of the probability of failure during a single scrubbing period. Our work proposes reliability models of SRAM FPGA designs, also under harsh radiation environments, where multiple coincident SEUs may occur, like [123], but for TMR-MER designs that contain multiple TMR components. The requirements of our models are similar to those of [123], but the probability of failure of a TMR component is estimated during two consecutive observations of the TMR component voters rather than during a single scrub period.

Our contributions in this chapter are:

- To derive reliability models of TMR-MER systems that comprise finitely many TMR components whose voter RRs are checked in round-robin order and at a variable rate (so-called *Variable-Rate Voter Checking* (VRVC)). Previous work has primarily focused on the effects of single events on SRAM FPGA-based systems while our analysis considers the impact of multiple consecutive events, which is an important consideration in providing a more comprehensive and accurate analysis of system

reliability.

- To propose a *Genetic Algorithm* (GA) for finding the optimal fixed rate at which to check all components so as to maximize the MTTF and the reliability of TMR-MER systems. The results show that using round robin for voter checks is not necessarily the best. Instead, scheduling of the voter checks according to component vulnerability substantially improves the system MTTF and reliability. Simulation results indicate that the MTTF of TMR-MER systems can be increased by up to 400% when VRVC rather than round robin, is used.

- To demonstrate that the MTTD is reduced by 44% and 30% on average when VRVC is used instead of a static round robin and dynamic VSE voter checking regimes, respectively.

- To show that the power consumed checking for errors can be reduced by reducing the checking frequency. In this case, VRVC is capable of ensuring a higher system reliability than the static round robin and the dynamic VSE approaches. In this case the reliability of a system using VRVC is far higher than that of the same system using round robin for voter checking.

This chapter is organized as follows: Section 5.1 presents reliability models for TMR-MER systems that consist of finitely many components whose voters are checked in round-robin order or at a variable rate. Section 5.2 presents the results of a small simulation study used to assess the models derived in Section 5.1 on a system comprising two components. Section 5.3 describes two genetic algorithms used to derive a voter checking schedule with the objective of maximizing the system reliability when systems are composed of n components. Section 5.4 describes our simulation experiments to assess the performance of the proposed VRVC relative to VSE and round-robin voter checking, while Section 5.5 details our experimental method, reports on our findings and discusses the results. Concluding remarks and directions for further study are given in Section 5.6.

5.1 Reliability Model

In this section, we introduce models that estimate the reliability of TMR-MER systems. These models are then used to estimate the reliability of FPGA-based designs in harsh

radiation environments when multiple coincident upsets are more probable[1]. We describe a general reliability model that has been widely used to estimate the reliability of FPGA-based systems. Based on this general model, we outline a procedure for estimating the reliability of TMR-MER systems that consist of an arbitrary number of TMR components and whose voters are checked in either round-robin order or at a variable rate.

5.1.1 General Reliability Model

The reliability of a TMR component k over time Δt, $R_k(\Delta t)$, can be expressed w.r.t. the failure probability of the component, $FP_k(\Delta t)$, which is the sum of the individual likelihoods that the component fails for all u SEUs that may affect the device during Δt. These relationships are given in [123] as:

$$\begin{aligned} R_k(\Delta t) &= 1 - FP_k(\Delta t), \\ FP_k(\Delta t) &= \sum_{u=1}^{\infty} P(F_k|E_u)P(E_u, \Delta t), \end{aligned} \tag{5.1}$$

where event F_k is the failure of component k during the period of time Δt and event E_u is that u SEUs have occurred in the device during the period of time Δt. Failure of TMR component k means that at least two of the three modules suffer from errors and that the component's voters therefore fail to produce the correct output.

$P(F_k|E_u)$ can be estimated for various values of u using the number of sensitive bits per component, for which we use the number of essential bits reported by the vendor's tools as a worst case estimate. Sensitive bits are those bits that cause a functional error if they change state, while essential bits are those bits associated with the circuitry of the design [89].

$P(E_u, \Delta t)$, the probability of event E_u occurring during Δt, can be modelled with a Poisson distribution [123],

$$P(E_u, \Delta t) = e^{-\nu}\frac{\nu^u}{u!}, \tag{5.2}$$

where ν is the expected number of SEUs suffered by the device during a period of time Δt and is obtained from the product of the failure rate of one configuration memory bit

[1]Please note that the model presented does not take into account *Multiple-bit upsets* (MBUs) i.e., more than one upset in a configuration word or frame from a single charged particle [131].

of a device (λ_{bit}), the number of configuration memory bits of a device (n_c) and the time period (Δt):

$$\nu = \lambda_{bit} \times n_c \times \Delta t. \tag{5.3}$$

λ_{bit} depends upon the radiation level, the IC process technology and the circuit architecture of the FPGA fabric.

Once the failure probability of component k is known, the failure rate λ_k of component k is given by [123]:

$$\lambda_k = \frac{FP_k(\Delta t)}{\Delta t}. \tag{5.4}$$

Since a TMR component can fail in different scenarios (see Figure 5.1 and associated discussion in Section 5.1.2) with different failure rates (λ_k^i), it is more meaningful to compute the composite failure rate of each component (λ_k^c). This parameter can be calculated for the expected proportions (ρ_k^i) in which each scenario occurs:

$$\lambda_k^c = \sum_{i=1} \rho_k^i \lambda_k^i. \tag{5.5}$$

where $\sum \rho_k^i = 1$.

Typically, a system contains N interdependent TMR components connected in series such that the failure of any one TMR component causes the system to fail. The failure rate of a series TMR system, λ_s, is the sum of all component failure rates [85].

$$\lambda_s = \sum_{k=1}^{N} \lambda_k^c. \tag{5.6}$$

Furthermore, the MTTF of the series TMR system is given by the reciprocal of the system failure rate.

$$MTTF = \frac{1}{\lambda_s} = \frac{1}{\sum_{k=1}^{N} \lambda_k^c}. \tag{5.7}$$

Finally, the system reliability is calculated as follows:

$$R_s(t) = e^{-\lambda_s . t}. \tag{5.8}$$

In this chapter, the main objective is to compare the reliabilities of TMR-MER systems that employ round robin, VSE and VRVC. As explained in Section 3.3.4, we do not therefore consider the impact of non-redundant modules, such as the RCN, the RC, and the voters, on system reliability.

Table 5.1: Notation

Symbol	Definition
N	Number of TMR components in the system.
Ck	Component $k, k = 1..N$.
n_c	Number of configuration memory bits in the device.
n_{ek}	Number of essential bits per module of component k (assumed to be identical for all three modules).
O_{kn}	Ck is observed for the n^{th} time by checking its voter(s).
Δt_o	The time period between successive voter observations (assumed to be constant for a given system setting).
Δt_{dk}	The time period between two consecutive observations of Ck.
Δt_{rk}	The time period to recover a faulty module of Ck.
Δt_k	The total time period over which Ck can fail.
Δt_{dij}	The time period between successive observations of Ci and Cj.
$\Delta t_{d'ij}$	The average time period between two consecutive observations of Ci in the interval between two consecutive observations of Cj.
Δt^g_{dkf}	The time period between two consecutive observations of Ck in group g in which f is the first component suffering and being recovered from an error.

5.1.2 Failure Rates of TMR-MER Systems in which Voters are Checked in Round-Robin Order

5.1.2.1 Two-Component Systems

Based on the general reliability model described in Section 5.1.1, we estimate the failure rate of systems comprised of two TMR components connected in series. Hereafter, we say that if the output of one module of a TMR component repeatedly differs from that of the other two, that the component is suffering from an "error", and if, after the component suffers another one or more SEUs, the outputs of the remaining two modules repeatedly differ, that the component has "failed". We also assume that once a faulty module is detected, it is dynamically reconfigured to correct the error.

In a two-component system, a component may fail in one of four different ways that are classified into two groups as shown in Figure 5.1 using the notation listed in Table 5.1. (Note that Figure 5.1 only describes the modes in which C1 can fail; the modes in which

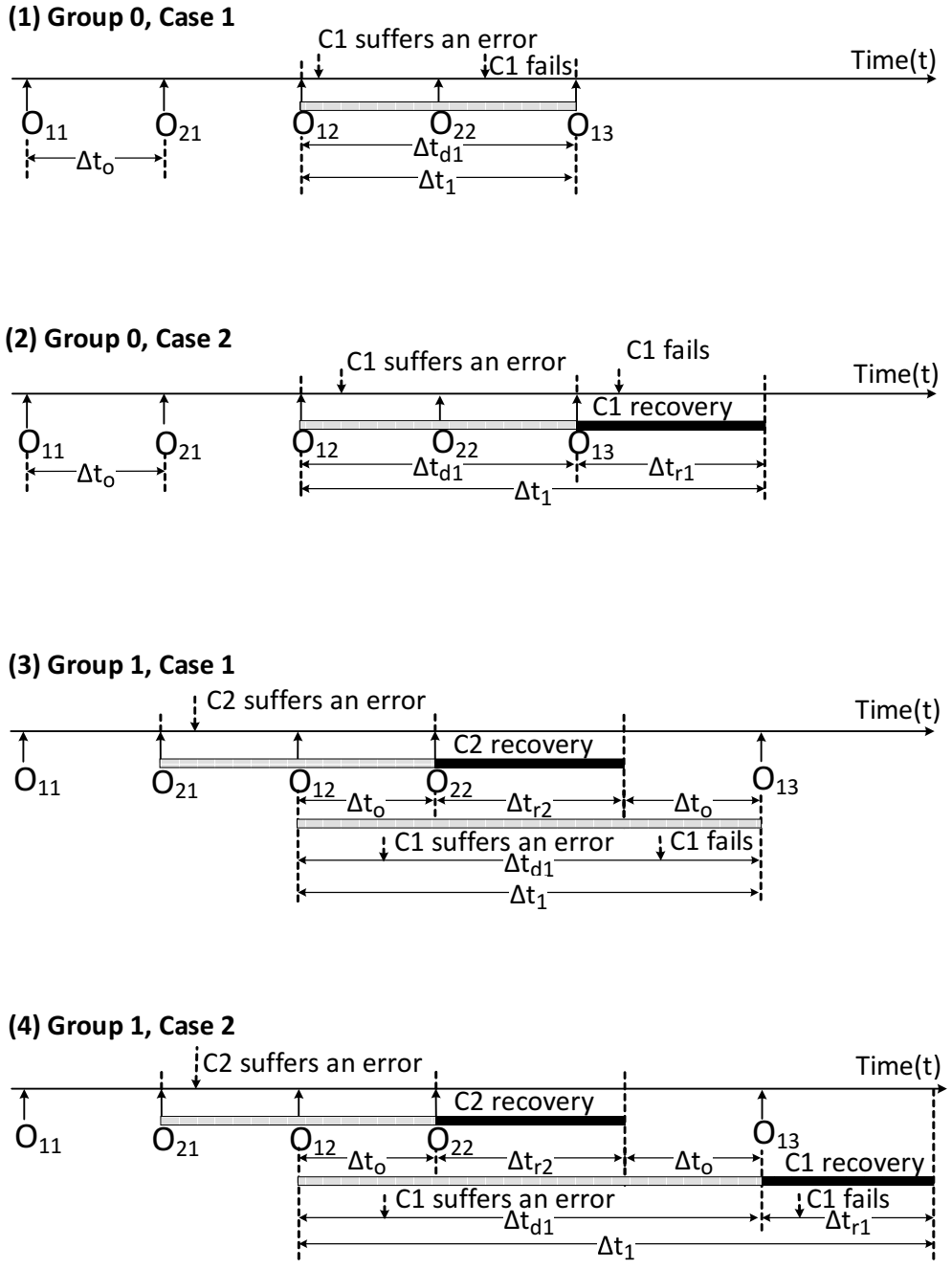


Figure 5.1: Failure mode for component 1 in two-component systems in which the voters are checked in round-robin order.

C2 can fail can be derived in a similar manner.):

Group 0: No other component suffers an error

– Case 1 (Figure 5.1(1)): C1 suffers from two or more SEUs that cause it to fail during the period of time between two consecutive checks of its voters (e.g., during Δt_1 — the period of time between O_{12} and O_{13}).

– Case 2 (Figure 5.1(2)): C1 suffers an error from one or more SEUs during the period of time between two consecutive checks of its voters (between O_{12} and O_{13} in Figure 5.1(2)). Thereafter, C1 fails if one or more SEUs affect its remaining working modules during the period of time during which it is recovering from the previous error (e.g., during Δt_{r1} — from time O_{13} to the end of the recovery process of C1).

Group 1: One other component suffers an error

– Case 1 (Figure 5.1(3)): C1 suffers from two or more SEUs that cause C1 to fail during a period of time between two consecutive checks of its voters that is longer than usual because the system is recovering from an error in C2. C1 fails during the period of time that commences after it is observed to be without an error (at O_{12}), continues while C2 is checked and recovered, and finishes when C1 is observed again at O_{13}.

– Case 2 (Figure 5.1(4)): C1 suffers an error from one or more SEUs during the period of time between two consecutive checks of it (between O_{12} and O_{13}) while the system is recovering from an error in C2. C1 then fails if one or more SEUs affect a second or third module of C1 while it is recovering from the previous error.

To summarize, in case 1 of either group, component k fails, i.e., suffers multiple errors to its different modules, between successive voter checks. In case 2, on the other hand, component k suffers an error to one of its modules during this period, and then fails following subsequent upsets to its other modules while recovering from the first error.

Estimating the Probabilities of Component Failure The failure probability of component k in case 1 of either group $g, g = 0, 1$ (FP^1_{kg}) is computed based on $FP_k(\Delta t)$ in Equation (5.1) with corresponding Δt_k as shown in Figures 5.1(1) and 5.1(3).

$$FP^1_{kg} = \sum_{u=1}^{\infty} P(F_k|E_u)P(E_u, \Delta t_k). \tag{5.9}$$

FP^1_{kg} is calculated by summing the failure probabilities of component k for all u SEUs that occur in the device during the period of time between two consecutive checks of component k. Please see Section 5.1.2.1 for a more detailed explanation.

The failure probability of component k in case 2 of either group $g, g = 0, 1$ (FP^2_{kg}) is the joint probability that event M_k (i.e., that component k suffers an error) occurs during the period of time Δt_{dk} as shown in Figures 5.1(2) and 5.1(4) and that component k fails during the period of time Δt_{rk} given the occurrence of event M_k.

$$FP^2_{kg} = \sum_{u=1}^{\infty} P(M_k|E_u)P(E_u, \Delta t_{dk}) \sum_{u=1}^{\infty} P(F_k|M_k E_u)P(E_u, \Delta t_{rk}), \tag{5.10}$$

where $P(M_k|E_u)$ denotes the conditional probability that event M_k occurs given u SEUs, and $P(F_k|M_k E_u)$ denotes the conditional probability that component k fails given u SEUs following the occurrence of event M_k (see Section 5.1.2.1 for a more detailed explanation).

Estimating the Conditional Probabilities of Component Failure The following conditional probabilities are user-design specific and must be estimated for each design that is implemented.

$P(F_k|E_u)$ is the failure probability of component k given u SEUs in the device. If the number of SEUs is 1, the failure probability of component k is zero because one SEU cannot cause a malfunction of a TMR component. Thus, $P(F_k|E_1) = 0$. Empirically, this failure probability is not exactly zero because there are a few single configuration bits that do indeed cause the TMR design to fail [149]. However, applying the techniques that are described in [149] remove such bits from the TMR design.

If the number of SEUs is u and $u > 1$, component k fails only if one SEU affects one of the three modules first and then at least one of the other SEUs affects one of the other two working modules of this component. In other words, the failure probability of component k given u SEUs in the device is the product of the probability that one SEU affects one of the three modules and the complement of the probability that the $u - 1$ SEUs occur *outside* the other two remaining working modules of component k. The failure probability of component k in this case is calculated as

$$P(F_k|E_u) = \frac{3n_{ek}}{n_c}(1 - \prod_{i=1}^{u-1} \frac{n_c - 2n_{ek}}{n_c}). \tag{5.11}$$

$P(M_k|E_u)$ is the conditional probability that component k suffers an error given u SEUs in the device. If u SEUs occur in the device, component k suffers an error only if one SEU

affects one of the three TMR modules of component k and none of the other SEUs affect the two remaining working modules. Hence, $P(M_k|E_u)$ is estimated as follows:

$$P(M_k|E_u) = \frac{3n_{ek}}{n_c} \prod_{i=1}^{u-1} \frac{n_c - 2n_{ek}}{n_c}. \tag{5.12}$$

$P(F_k|M_kE_u)$ is the conditional probability that component k fails given that u SEUs occur in the device and that component k has suffered an error. If u SEUs occur in the device, the failure of component k occurs only if at least one SEU affects one of the other two remaining working modules of component k. Thus, $P(F_k|M_kE_u)$ is estimated as follows:

$$P(F_k|M_kE_u) = (1 - \prod_{i=1}^{u} \frac{n_c - 2n_{ek}}{n_c}). \tag{5.13}$$

Estimating Failure Rates, λ_{kg}^i Based on (5.4), the failure rate of component k in each case i of group g is estimated as follows:

$$\begin{aligned} \lambda_{kg}^1 &= \frac{FP_{kg}^1}{2\Delta t_o + \sum_{i=1}^{g} \Delta t_{rk'}}, \\ \lambda_{kg}^2 &= \frac{FP_{kg}^2}{2\Delta t_o + \sum_{i=1}^{g} \Delta t_{rk'} + \Delta t_{rk}}, \end{aligned} \tag{5.14}$$

where $k' \neq k$ and k' is chosen based on g. For example for $k = 1$, if $g = 0$, the sums $(\sum_{i=1}^{g} \Delta t_{rk'})$ in (5.14) are invalid and assigned 0; if $g = 1$, then $k' = 2$.

Estimating the Proportion ρ_{kg}^i by which Component k Fails in Each Case i of Group g These parameters are calculated for the likelihood by which component k fails in each case:

$$\rho_{kg}^i = \frac{\alpha_{kg}^i}{\sum_{l=0}^{1} \sum_{j=1}^{2} \alpha_{kl}^j}, \tag{5.15}$$

where $\alpha_{kg}^i, i = 1, 2$ and $g = 0, 1$, denotes the likelihood that case i of group g occurs. We estimate α_{kg}^i as follows: $\alpha_{k0}^i = \frac{3n_{ek}}{n_c}$ and $\alpha_{k1}^i = \frac{3n_{ek'}}{n_c} \frac{3n_{ek}}{n_c}$, because the likelihoods of cases 1 and 2 of group 0 occurring depend upon the likelihood of component k suffering an error, while that of cases 1 and 2 of group 1 occurring depend upon the likelihood of both components k' and k suffering errors.

The composite failure rate of component k, λ_k^c, is calculated by substituting for λ_k^i and ρ_k^i into Equation (5.5). The system failure rate can be computed by summing all λ_k^c.

5.1.2.2 $N-$Component Systems

Based on the results from sub-section 5.1.2.1, we next estimate the composite failure rates, λ_k^c, of all TMR components for a system that consists of N TMR components in which configuration memory errors are recovered by MER. At this juncture in our analysis it is still assumed that the N components are being checked for errors in round-robin order.

Component $k, k = 1..N$, may fail in one of many different ways which we classify into N groups as follows:

- Group 0 involves two cases in which component k fails during the period of time between two consecutive checks of its voter(s) or in which it suffers an error during this period and fails while recovering from that error. This group corresponds to group 0 of the two-component system analysed in Section 5.1.2.1.
- Group 1 involves $2(N-1) = 2\binom{N-1}{1}$ cases that component k fails or suffers an error while the system recovers from an error in one of the other components. This is because there are $N-1$ cases that one other component may be recovered before component k is next checked. This group corresponds to group 1 of the two-component system.
- Group 2 involves $2\binom{N-1}{2}, N > 2$ cases that component k fails or suffers an error while the system recovers from errors in two other components. Since components are checked in round robin order, the order of components that suffer errors is taken into account. With $N-1$ components, there are $\binom{N-1}{2}$ combinations that two other components may need to be recovered before component k is next checked.
- ...
- Group $(N-1)$ involves $2\binom{N-1}{N-1}$ cases that component k fails or suffers an error while the system recovers from errors in the other $N-1$ components. This is because there is only one combination that all other components may need to be recovered before component k is next checked.

From the foregoing, it can be observed that group $g, g = 0..N-1$ involves $2\binom{N-1}{g}$ cases that component k fails or suffers an error while the system recovers from errors in g other components. Since the components are *checked in round-robin* order, if more than one component suffers an error, these are also *recovered in round-robin* order. Therefore, if g other components suffer errors, there are $\binom{N-1}{g}$ combinations that g other components are recovered before component k is next checked.

Let $FP_{kg}^{im}, i = 1, 2$, $m = 1..\binom{N-1}{g}$ and $g = 0..N-1$, be the failure probability of component k in case im of group g. FP_{kg}^{1m} is computed by (5.9) while FP_{kg}^{2m} is computed by (5.10). Thus, based on (5.4), the failure rate of component k in each case in group g is estimated as follows:

$$\begin{aligned} \lambda_{kg}^{1m} &= \frac{FP_{kg}^{1m}}{N\Delta t_o + \sum_{i=1}^{g} \Delta t_{rk'}}, \\ \lambda_{kg}^{2m} &= \frac{FP_{kg}^{2m}}{N\Delta t_o + \sum_{i=1}^{g} \Delta t_{rk'} + \Delta t_{rk}}, \end{aligned} \tag{5.16}$$

where $k' \neq k$ and k' are all the indices of components applicable in case m. For example for $N = 3$ and $k = 1$, if $g = 0$, the sums ($\sum_{i=1}^{g} \Delta t_{rk'}$) of (5.16) are invalid and are assigned 0; if $g = 1$, then $m = 1$ or 2 and $k' = 2$ or 3, respectively; if $g = 2$, then $m = 1$ and $k' = 2$ and 3.

Additionally, the expected proportion (ρ_{kg}^{im}) in which each scenario occurs is calculated as follows:

$$\rho_{kg}^{im} = \frac{\alpha_{kg}^{im}}{\sum_{r=0}^{N-1} \sum_{t=1}^{\binom{N-1}{g}} \sum_{j=1}^{2} \alpha_{kr}^{jt}}, \tag{5.17}$$

where α_{kg}^{im} denotes the likelihood that case im occurs. Therefore, we estimate $\alpha_{kg}^{im} = \frac{3n_{ek}}{n_c} \prod_{i=1}^{g} \frac{3n_{ek'}}{n_c}$ because the likelihood of each case α_{kg}^{im} occurring depends upon the likelihood of g other components suffering errors in a row. Note that k' here is similar to (5.16), except when $g = 0$, the product ($\prod_{i=1}^{g} \frac{3n_{ek'}}{n_c}$) is assigned the value 1.

The composite failure rate of component 1 is calculated by substituting for λ_k^i and ρ_k^i in (5.5) with (5.16) and (5.17). The system failure rate and reliability are calculated using (5.5) and (5.8) with λ_k^c for each component of the system.

5.1.3 Failure Rates of TMR-MER Systems Employing VRVC

VRVC is defined as a periodic schedule in which component voters are checked at specific times and in which the more vulnerable components' voters are checked more frequently than those of the less vulnerable ones. For example in a system of 4 components, one period of a schedule could be 4-3-4-2-4-3-4-2-3-1 in which each digit represents the component whose voters are to be checked. In this case, component 4 is deemed more vulnerable and hence checked more frequently when compared to the other components, and component 1 is deemed least vulnerable and hence checked less frequently. In this sub-section, we

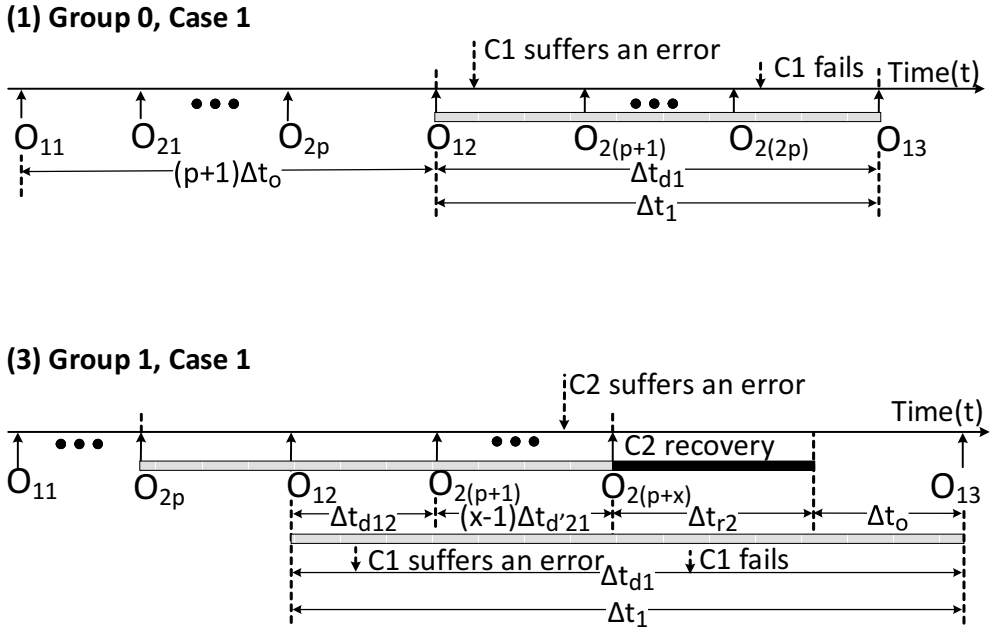


Figure 5.2: Failure of component 1 in systems employing variable-rate voter checking

first consider a system that consists of two components and then generalise the model to obtain the reliability of systems that consist of any number of TMR components.

5.1.3.1 Two-component Systems

Similar to the cases described in Section 5.1.2, we observe that C1 fails in one of four different ways as partly depicted in Figure 5.2 using the notation of Table 5.1. Note that p in Figure 5.2 denotes the nominal number of times that C2 is checked between two consecutive checks of C1 due to its greater susceptibility to SEUs than C1's. In case 1 of group 1 (Figure 5.2(3)), we assume that the system detects an error in C2 x checks after C1 is checked (at O_{12}) where x varies from 1 to p. In this analysis, we associate with $x = 1..p$ the number of checks that the system performs before it detects an error in C2. Thus, each case of group 1 involves p sub-cases that have the same likelihood of both components suffering an error (ρ_k^i). For example, given the following schedule for checking two components: 1-2-2-2-2-1-2-2-2-2-1... where each digit denotes the observation of the corresponding component, Δt_{d1} and Δt_{d2} in group 0 are $5\Delta t_o$ and $1.25\Delta t_o$, respectively. Both Δt_{d12} and $\Delta t_{d'21}$ in group 1 are Δt_o. Furthermore, with such a schedule, there are four checks of C2 during the period of time between two consecutive checks of C1. Thus, $x = 1..4$.

The observations of C2 differ slightly from those of C1. C2 may also fail in one of four different ways, but the number of sub-cases in group 1 is only 1. This is because between any two consecutive checks of C2, C1 is checked at most once.

Let FP^1_{k0} and FP^2_{k0} be the failure probability of component k in cases 1 and 2 of group 0; and let FP^{1x}_{k1} and FP^{2x}_{k1} be the failure probability of component k in cases 1 and 2 of group 1 where $x = 1..p_k$ and p_k denotes the number of times that component k is checked between two consecutive checks of component k' ($k' \neq k$). FP^1_{k0} and FP^{1x}_{k1} are computed by (5.9) while FP^2_{k0} and FP^{2x}_{k1} are computed by (5.10) with corresponding Δt_k in each case, as partly shown in Figure 5.2.

Thus, based on (5.4), the failure rate of component k in each case is estimated as follows:

$$\begin{aligned}
\lambda^1_{k0} &= \frac{FP^1_{k0}}{\Delta t_{dk}}, \qquad \lambda^2_{k0} = \frac{FP^2_{k0}}{\Delta t_{dk} + \Delta t_{rk}}, \\
\lambda^{1x}_{k1} &= \frac{FP^{1x}_{k1}}{\Delta t_{dkk'} + (x-1)\Delta t_{d'kk'} + \Delta t_o + \Delta t_{rk'}}, \\
\lambda^{2x}_{k1} &= \frac{FP^{2x}_{k1}}{\Delta t_{dkk'} + (x-1)\Delta t_{d'kk'} + \Delta t_o + \Delta t_{rk'} + \Delta t_{rk}}.
\end{aligned} \tag{5.18}$$

Additionally, the expected proportion, ρ^i_{kg}, in which each scenario occurs can be calculated as follows:

$$\rho^{ix}_{kg} = \frac{\alpha^{ix}_{kg}}{\sum_{r=0}^{1}\sum_{t=1}^{p_k}\sum_{j=1}^{2}\alpha^{jt}_{kr}}, \tag{5.19}$$

where $\alpha^i_{kg}, i = 1, 2$ denotes the likelihood that case i occurs. Therefore, α^{ix}_{kg} is estimated as $\alpha^i_{k0} = \frac{3n_{ek}}{n_c}$ (x is implicitly assigned 0) and $\alpha^{ix}_{k1} = \frac{3n_{ek'}}{n_c}\frac{3n_{ek}}{n_c}$, because the likelihood of cases 1 and 2 occurring in group 0 depends upon the likelihood of component k suffering an error, while that of cases 1 and 2 occurring in group 1 depends upon the likelihood of both components k' and k suffering errors.

The composite failure rate of component k is calculated by substituting for λ^i_k and ρ^i_k in (5.5) with (5.18) and (5.19).

The above observations allow us to compute the system failure rate.

5.1.3.2 N-component Systems

Without loss of generality, we assume that the components are numbered and ranked $k, k = 1..N$, into increasing vulnerability order, and that component k is therefore checked more frequently than component k–1. After the reconfiguration of a faulty module is

finished, the system checks all other components in descending order of vulnerability before recommencing the planned schedule. For example, in a system of 4 components, after component 2 is recovered, components 4, 3 and 1 need to be checked in that order before resuming the variable-rate schedule. If multiple errors occur in a sequence, the system checks all other components that have not been reconfigured. In doing so, the system is fair in reconfiguring all components of the system if an error is detected.

Based on the results from the previous section, we generalize the approach for estimating the composite failure rate, λ_k^c, of all components in a system containing N components that are checked at different rates.

Before detailing the failure cases of one component in a TMR-MER system that employs variable-rate voter checking, we define ${}^nP_k$ as the well-known k-permutations of n, which is 0 when $k > n$ or $k < 0$, and otherwise is equal to $\frac{n!}{(n-k)!}$. In an $N-$component system, component k may fail in various ways that may be classified as belonging to one of N groups as follows:

- Group 0 involves two cases that component k fails during the period of time between two consecutive checks of it or suffers an error during this period and fails while recovering from that error.
- Group 1 involves $2(N-1) = 2({}^{k-1}P_1 + {}^{N-k}P_1)$ cases that component k fails or suffers an error while the system recovers from an error in one of the other components. This is because there are ${}^{k-1}P_1$ cases that a component with an index less than k and ${}^{N-k}P_1$ cases that a component with index greater than k is recovered before component k is next checked.
- Group 2 involves $2({}^{k-1}P_1 {}^{N-k}P_1 + {}^{N-k}P_2), N > 2$ cases that component k fails or suffers an error while the system recovers from errors in two other components. In the first sub-group, we assume that one of ${}^{k-1}P_1 = (k-1)$ components with indices less than k is the first component to suffer an error. As the system checks all components in decreasing vulnerability order after any module is reconfigured, the second component to suffer an error must belong to the ${}^{N-k}P_1 = (N-k)$ components with indices greater than k. Alternatively, we assume that one of ${}^{N-k}P_1$ components with indices greater than k is the first component to suffer an error. After reconfiguring the faulty module of the first component, the system checks all other components that have indices greater than k that have not yet been reconfigured. Thus, the second component that suffers an error must also belong to the upper group, consisting of $(N-k-1)$ components. Thus in the second sub-group,

we have ${}^{N-k}P_1 {}^{N-k-1}P_1 = {}^{N-k}P_2$ combinations of two components that may need to be recovered before component k is next checked.

- ...
- Group N-1 involves $2({}^{k-1}P_1 {}^{N-k}P_{N-2} + {}^{N-k}P_{N-1})$ cases that component k fails or suffers an error while recovering from errors in $N-1$ other components.

To summarise, the system failure rate can be computed by considering all possible cases in which each component may fail. A component k may also fail in different groups g, g = *0..N–1* in which g other components suffer an error first. Each group involves ${}^{k-1}P_1 {}^{N-k}P_{g-1} + {}^{N-k}P_g$ situations. This is because group g includes ${}^{k-1}P_1 {}^{N-k}P_{g-1}$ situations in which the first component to suffer an error, f, is such that the rank of $f < k$ and ${}^{N-k}P_g$ situations in which the rank of $f > k$. Each situation involves two cases as summarized in Section 5.1.2 and each case involves a number of sub-cases that are schedule dependent.

Let FP_{k0}^{11} and FP_{k0}^{21} be the failure probabilities of component k in cases 1 and 2 of group 0, respectively; and let FP_{kg}^{1mx}, FP_{kg}^{2mx} be the failure probabilities of component k in cases $1mx$ and $2mx$ of group $g, g > 1$ respectively, where $m = 1..{}^{k-1}P_1 {}^{N-k}P_{g-1} + {}^{N-k}P_g$; and x denotes the nominal number of checks of component f between two consecutive checks of component k. FP_{k0}^{11} and FP_{kg}^{1mx} are computed by (5.9) while FP_{k0}^{21} and FP_{kg}^{2mx} are computed by (5.10).

Thus, based on (5.4), the failure rate of component k in each case is estimated as follows:

$$\begin{aligned} \lambda_{k0}^{11} &= \frac{FP_{k0}^{11}}{\Delta t_{dk}}, \qquad \lambda_{k0}^{21} = \frac{FP_{k0}^{21}}{\Delta t_{dk} + \Delta t_{rk}}, \\ \lambda_{kg}^{1mx} &= \frac{FP_{kg}^{1mx}}{\Delta t_{dkf}^{g}}, \quad \lambda_{kg}^{2mx} = \frac{FP_{kg}^{2mx}}{\Delta t_{dkf}^{g} + \Delta t_{rk}}. \end{aligned} \tag{5.20}$$

Note that x in group 0 is implicitly assigned 0.

Δt_{dkf}^{g} is defined as the period of time between two consecutive observations of component k while the system is recovering g other components commencing with f as exemplified in Figure 5.3 with $g = 1$ and with the notation of Table 5.1. $\Delta t_{dkf}^{g}, g > 0$ is given as:

$$\Delta t_{dkf}^{g} = \begin{cases} \Delta t_{dkf} + (x-1)\Delta t_{d'fk} + \sum_1^g \Delta t_{rk'} + (N-k+1)\Delta t_o, & \text{if } f < k \\ \Delta t_{dkf} + (x-1)\Delta t_{d'fk} + \sum_1^g \Delta t_{rk'} + (N-k)\Delta t_o, & \text{if } f > k \end{cases}$$

where $k' \neq k$ and k' involves the indices of all components in each case of group g. This is because Δt_{dkf}^{g} involves the time period between two successive observations of components

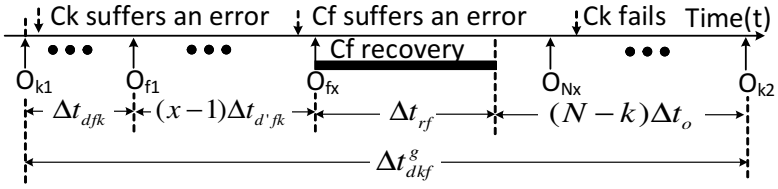


Figure 5.3: Δt^g_{dkf} when $f > k$ and $g = 1$

k and f (Δt_{dkf}), the period between checking component f for the first and the x^{th} times ($(x-1)\Delta t_{d'fk}$), the time period needed to recover the faulty modules of g components ($\sum_{i=1}^{g} \Delta t_{rk'}$), and the time period needed to recheck other components of higher rank than component k.

Additionally, the expected proportion, ρ^{imx}_{kg}, in which each scenario occurs is calculated as follows:

$$\rho^{imx}_{kg} = \frac{\alpha^{imx}_{kg}}{\sum_{r=0}^{N-1} \sum_{t=1}^{k-1 P_1 N-k P_{r-1} + N-k P_r} \sum_{l=1}^{p_{kf}} \sum_{j=1}^{2} \alpha^{jtl}_{kr}}, \tag{5.21}$$

where α^{imx}_{kg} denotes the possibility that case imx in group g occurs. Therefore, we estimate α^{imx}_{kg} as follows:

$$\alpha^{imx}_{kg} = \frac{3n_{ek}}{n_c} \prod_{i=1}^{g} \frac{3n_{ek'}}{n_c} \tag{5.22}$$

because the likelihood of each case α^{imx}_{kg} occurring depends upon the likelihood of g components suffering errors consecutively. Note that k' in (5.22) is similar to its use in (5.20), except when $g = 0$, in which case the product ($\prod_{i=1}^{g} \frac{3n_{ek'}}{n_c}$) is assigned 1.

The composite failure rate of component 1 is calculated by substituting λ^i_k and ρ^i_k in (5.5) with (5.20) and (5.21). The system failure rate and reliability are calculated using (5.5) and (5.8) with λ^c_k for each component of the system.

Most of the timing parameters for computing the composite failure rate of one component (as partly shown in Figure 5.2) differ from those of the others and are not constant. Fortunately, we can statistically determine their average values based on a real schedule as described in Section 5.3

5.2 Simulating a 2-Component System

We interrupt the main flow of our presentation to report on an early simulation we undertook to assess the performance of VRVC. In this section we present results of simulations that aimed to assess the reliability of systems comprising two components using both round-robin and VRVC as the time period between successive voter observations was varied. As will be seen, non-linearities in our results suggest that determining an optimal voter checking schedule is likely to be NP-hard and thus motivate the development of evolutionary approaches to finding good schedules, as discussed in the next section.

In our study, we simulated a system comprising two components, C1 and C2, containing 100,000 and 1,000,000 essential bits and having assumed reconfiguration times of $0.2ms$ and $2ms$, respectively. Simulation parameters were based on a Xilinx Artix-7 XC7A200T device, containing $n_c = 77,845,216$ configuration bits in total, and operating in a Geostationary Equatorial Orbit (GEO) with an anticipated upset rate of 2.66E-10 upsets/bit/s at the peak-5-min condition (Table 2.2). We used the expressions derived in Sections 5.1.2.1 and 5.1.3.1 to calculate the MTTF for a round-robin checking schedule, as well as for VRVC as the number of checks, p, of C2 was varied relative to the 1 check made per period of component C1. These calculations were performed as the voter observation period (Δt_o) ranged from $1\mu s$ to $1s$.

In this case study, we run a brute force search for all possible number of checks for each component. We calculated the ratios of VRVC MTTF to round-robin MTTF on specific voter observation times varying from 1μs to 1s. For each voter observation time, we assign the number of checks for C1 to 1 and increase the number of checks for C2, p, from 1 (i.e., round-robin approach) to until the MTTF ratio starts decreasing. The previous p value before the MTTF ratio is decreased is the one that provides maximum system reliability. We also iterated by assigning number of checks for C2 to 1 and increasing the number of checks for C1. However, the second case always results in system MTTF less than the round robin approach.

Our results, plotted in Figure 5.4(a), show that a better MTTF is achieved by VRVC at all voter observation periods. Figure 5.4(b) shows that the number of checks p of C2 that is needed for each check of C1 to obtain the best MTTF for VRVC relative to round robin varies depending on the voter observation period. The results confirm our intuition that the larger components should be checked more frequently than smaller ones in order to maximize system reliability. With the assumptions made in this study, we found that use of VRVC rather than round robin to check voters led to a significant improvement in

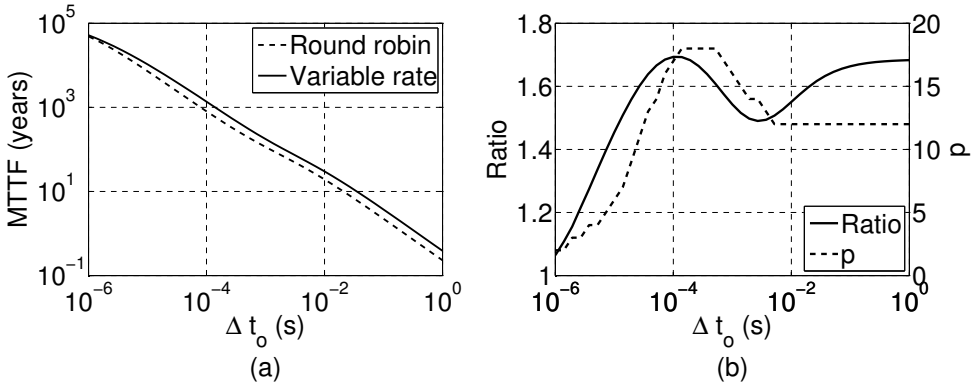


Figure 5.4: (a) MTTF (years) of the VRVC and round-robin voter checking approaches. (b) Peak MTTF ratio achieved when varying the voter checking rate relative to checking voters in round-robin order, and the corresponding rate p at which C2 is checked relative to C1 to achieve the peak MTTF ratio.

system MTTF of up to 70%. It also came as a surprise to us that there was no direct relationship between the module sizes and the optimal rate p at which the larger of the two modules should be checked.

As can also be seen from Figure 5.4(b), the maximum benefit of VRVC (relative to round robin) and the number of checks p of C2 needed to achieve this optimal MTTF are non-linear functions of the voter observation period. Unfortunately, it is infeasible to use an exhaustive search, as we have done in this study, to determine the optimal number of checks to perform on each component as the *number of components* in the system increases. For this reason, in the next section we develop a heuristic approach using evolutionary algorithms to determine a "good" checking schedule.

5.3 Scheduling Voter Checks

Many optimization problems in the real world, such as multi-modality, dimensionality and differentiability, are very complex in nature and quite hard to solve using conventional techniques. Techniques, such as steepest decent, linear programming and dynamic programming, generally fail to solve optimization problems with non-linear objective functions or with many local optima. To solve such hard problems, more powerful optimization algorithms, such as evolutionary algorithms, which include genetic algorithms (GAs), are needed [12].

As can be seen from Figure 5.4(b), clearly, the peak MTTF ratio, which is the function we wish to maximize, is non-linear, and even for a 2-component system, we observe two maxima. We therefore propose to use a GA, which is a probabilistic search method based on an evolutionary approach, to heuristically determine the rate at which all triplicated components in a system should be checked so as to maximize the system reliability. Once the rate at which components should be checked has been determined, we use a second GA, as detailed in [52], to *generate a schedule in which the determined number of voter checks are distributed as evenly as possible* per schedule period. The schedule produced by the second GA is needed to evaluate the fitness of individual solutions to the first GA, which determines the number of checks to be performed. The second GA is therefore nested within the first GA's evaluation function.

Note that apart from evolutionary algorithms, other meta-heuristic optimization techniques have been developed to solve non-linear problems with multiple minima. These include the Artificial Immune Algorithm [48], Ant Colony Optimization [39], Particle Swarm Optimization [84], and several more. In this thesis, we only use GAs to find a static schedule for voter checks. However, it may be worthwhile trialling the above techniques as well to determine which method produces the best results.

5.3.1 Genetic Algorithm

A typical GA requires a genetic representation of the solution domain and a fitness function to evaluate the solution domain. Possible solutions (individuals) are represented by a data structure called a chromosome. A chromosome is composed of simple elements called genes. An initial population of possible solutions is randomly created. As long as the stopping condition (e.g., exceeding a given number of generations) has not been met, a new generation is created. This involves computing the fitness value of each individual in the population. Individuals that represent desirable solutions (e.g., high fitness values or small system failure rates in our case) are selected with high probability to produce offspring. In a crossover process, some parts of the selected individuals (parent chromosomes) are combined to create a new individual (a child chromosome). Furthermore, in a mutation process the child's chromosome is randomly changed to introduce new genetic information. The children created by crossover and mutation are inserted into the new population, thereby replacing other low-fitness individuals. In our work, a GA is used for finding the number of times each TMR component should be checked per schedule period. The algorithm has the following characteristics:

(i) *Representation*: The solution domain is a population (P) in which each chromosome in P is an array consisting of N genes (e.g., $[d_1 \ ... \ d_N]$) representing N TMR components. the values $d_k, k = 1..N$, which are each greater than 0 and arranged into monotonically increasing order, represent the number of times that each TMR component must be checked in one period of the schedule.

(ii) *Initialization*: An initial population is formed of individuals that are created by generating N random numbers between 1 and an upper bound value (e.g., 50) that are sorted into ascending order.

(iii) *Evaluation*: The fitness value is the system failure rate, as estimated using the analysis outlined in Section 5.1, corresponding to each individual in population P. Please note that chromosomes that do not satisfy the constraints on d_k and duplicate chromosomes are removed from the population before proceeding to the next step. Eliminating duplicate chromosomes prevents super chromosomes from dominating the population and helps maintain the diversity of the population [54].

(iv) *Selection*: A *tournament selection* is adopted for the application of the selection procedure [56]. This approach involves running several "tournaments" among a few individuals chosen at random from the population. The individuals with the best fitness are selected for the application of crossover and mutation. Note that selection is performed on the enlarged sample space in which both parents and offspring have the same chance of competing for survival.

(v) *Crossover*: We use a uniform crossover method, which randomly selects genes from the first and the second parent to generate an offspring. For example, with $N = 3$, two chromosomes [1 3 5] and [2 4 6] may create an offspring of [1 4 5] or [2 3 5]. The probability of crossover is a user-defined value e.g., 0.25, by which we expect that on average an offspring inherits 25% genes of the first parent and 75% genes of the second parent.

(vi) *Mutation*: Mutation alters one or more genes with a probability equal to the mutation rate (e.g., 10%) of a parent selected during the tournament. For example, with $N = 3$, assuming the second gene of the chromosome [1 2 5] is selected for mutation, a new value is generated by randomly adding 1 to, or subtracting 1 from, the mutated number. Thus the chromosome after mutation is [1 3 5] assuming addition was selected.

After the mutation function is finished, a new population is created and the evaluation procedure is repeated. When the GA function meets the stopping condition, it terminates

and returns the best individual of the current population.

5.3.2 Scheduling of Voter Checks

Calculating the system failure rate requires all timing parameters, most of which can only be obtained from a real schedule. A real schedule must ensure fair voter checking among all TMR components. These voter checks, in turn, are required to be evenly distributed so that the temporal gap between any two consecutive checks of the same TMR component is as constant as possible. The problem of creating such a sequence of voter checks belongs to the class of *Response Time Variability Problems* (RTVP) [35], which arises whenever products, clients, jobs or, as in this work, voter checks, need to be sequenced in such a way that the variability in the period between the instants at which they receive the necessary resource is minimized.

The RTVP is formulated as follows. Given N positive integers $d_1 \leq ... \leq d_N$ associated with the number of checks of N TMR components, respectively, let $D = \sum_{k=1}^{N} d_k$ and the rates $r_k = \frac{d_k}{D}$ for $k = 1..n$. Let $S = s_1 s_2 ... s_D$, a vector of length D where TMR component k occurs exactly d_k times, be a solution of an instance of the RTVP that consists of a circular sequence of copies of S. For any two consecutive checks of TMR component k we define the distance τ between them as the number of voter checks of other TMR components that separates them plus 1. Since TMR component k is checked exactly d_k times in S, then there are exactly d_k distances $\tau_1^k, ..., \tau_{d_k}^k$ for k. Note that the sequence is circular, s_1 comes immediately after s_D; therefore, $\tau_{d_k}^k$ is the distance between the last occurrence of TMR component k in the preceding cycle and the first occurrence of the same component in the current cycle. Obviously, the two are the same for $d_k = 1$. Since

$$\tau_1^k + ... + \tau_{d_k}^k = D, \tag{5.23}$$

then the average distance $\bar{\tau_k}$ between the TMR component k in S equals

$$\bar{\tau_k} = \frac{D}{d_k} = \frac{1}{r_k}, \tag{5.24}$$

and it is the same for each feasible sequence S. The response time variability for TMR component k is formulated as follows

$$RTV_k = \sum_{j=1}^{d_k} (\tau_j^k - \bar{\tau_k})^2, \tag{5.25}$$

and the objective is to minimize the total response time variability (RTV) that is formulated as follows

$$RTV = \sum_{k=1}^{N} RTV_k = \sum_{k=1}^{N} \sum_{j=1}^{d_k} (\tau_j^k - \bar{\tau_k})^2. \tag{5.26}$$

Unfortunately, the RTVP is NP-hard [35]. To solve our RTVP, we utilize the GA that is detailed in [52] to find the optimal schedule of voter checks.

5.3.3 Mean Time To Detect Errors

The MTTD errors is defined as the average elapsed time interval between a configuration bit being affected by a fault and the instant that the erroneous TMR module is detected. MTTD errors for each component is inversely proportional to its checking rate and is estimated by half the average distance $\bar{\tau_k}$ of component k in the circular sequence S. In addition, the numbers of errors occurring in each component are component dependent; we therefore factor these numbers in when calculating the MTTD for the whole system. The MTTD (in units of Δt_o) can be estimated as follows:

$$MTTD = \frac{\sum_{k=1}^{N} e_k \frac{\bar{\tau_k}}{2}}{\sum_{k=1}^{N} e_k} = \frac{\sum_{k=1}^{N} e_k \frac{D}{2d_k}}{\sum_{k=1}^{N} e_k} \tag{5.27}$$

where e_k denotes the number of errors that occur in component k and $D = \sum_{k=1}^{N} d_k$ denotes the length of one period of the voter checking schedule. If the voters are checked in round-robin order, $d_k = 1, \forall k$ and $D = N$, thus $MTTD = \frac{N}{2}$ in units of Δt_o.

5.3.4 Power Consumption

The power consumption of the components involved in the ICAP-based voter checking approach can be divided into two parts: the power consumed and dissipated by the configuration port and built-in configuration controllers and the power consumed and dissipated by the RC circuitry. Both parts substantially contribute to the overall FPGA power consumption and dissipation [66].

Checking voters continuously provides the quickest response to errors and therefore the highest reliability. When error rates are high, this may be necessary. However, for much of the mission, the error rates may be low enough to allow the checking rate, and thus the power consumption, to be reduced.

5.4 Simulations

Simulation experiments were carried out to assess the performance of the VRVC schedule relative to both round-robin voter checking and the dynamically determined VSE approach presented in Chapter 4. These experiments involved simulating FPGA-based systems comprising sets of TMR components and assessing the MTTF for each voter checking method at low- and high-radiation orbit conditions. In the following, we describe our experimental set-up before presenting the results of the simulations.

5.4.1 Assumptions and Implementation

Our reliability models require an orbit-specific SEU rate and design-specific parameters, such as the total number of configuration bits in a device, the voter observation time, the number of essential bits for each module, the recovery time of each module and the voter observation schedule, which is produced by the genetic algorithms.

Similar to the case study of the two-component system reported in Section 5.2, we simulated a Xilinx Artix-7 XC7A200T device operating in GEO and in Low-Earth Orbit (LEO). The radiation levels of the peak-5-min condition at these orbits are expected to result in 2.66E-10 and 8.46E-12 upsets/bit/s, respectively as reported in Table 2.2. The total number of device configuration memory bits for this device is $n_c = 77,845,216$. We assume that the voter reconfiguration requests are checked at $1\mu s$ intervals, which correspond to the fastest configuration frame read time that the ICAP-based RCN can provide for this device. In order to assess the impact on reliability of reducing the system power consumption by reducing the voter checking frequency, we varied the voter checking period, Δt_o, from $1\mu s$ to $1s$.

We simulated systems comprising 3 – 20 TMR components to assess the performance of the schedules being studied as the number of components was varied. For each simulation, the size of each triplicated module, as measured in number of essential bits, was chosen randomly in a range from 10,000 to 2,000,000 bits using one of three size distributions: uniform, quadratic and exponential. For a given number of system components (3 – 20) and at each orbit level (LEO, GEO), we evaluated the performance of each schedule (VRVC, round robin, VSE) over 5 trials for each module size distribution (uniform, quadratic, exponential). In the following, we report on the average of the 5 trials that were carried out for each distribution.

The recovery time of a TMR module is given as the product of the number of its *Configuration Frames* (CFs) and the reconfiguration time per CF. As the typical ratio of configuration bits not affecting a design to those (essential) bits that do affect a design ranges between 9:1 and 4:1 [7], we made the pessimistic assumption that the ratio is 9:1 in our simulations. In other words, we assumed that 10% of the configuration memory bits per CF were essential. This has the effect of dilating the time to recover module configuration memory errors, which in turn increases (by as much as twice) the likelihood that subsequent errors occur before recovery from a previous error has been completed.

As detailed in Section 5.3, two GAs were implemented to obtain a schedule to yield the best possible system reliability. Fine tuning the parameters of a GA is almost always a difficult and time-consuming task [13,46]. In this section, we undertook experimentation using the following parameter values; further experimentation with the GA parameters is reported on in the next sub-section. The GA to determine the rate at which components should be checked per period was initialised with 100 random chromosomes with the value of each gene being randomly chosen to be in the range 1 – 50 with a uniform distribution. Since the simulation experiments are time consuming, particularly for 20-component systems, we decided that the GA should be terminated after 100 generations. In addition, the crossover rate and the mutation rate were set to 25% and 10%, respectively. As discussed, we implemented the GA from [52] to find the best distribution of checks once the check rates had been determined.

5.4.2 Results and Discussion

The results of our simulations demonstrate that the use of VRVC improves TMR-MER system reliability. The results are detailed in the following:

5.4.2.1 VRVC vs Round Robin

The ratio of MTTFs for systems employing VRVC to those checking voters in round-robin order are consistently greater than 1, and become larger as the voter observation time, Δt_o, is increased (Figure 5.5). It can be observed that the ratios are almost completely independent of the orbital/radiation conditions.

Figure 5.5 also shows that when the number of essential bits of all TMR components are exponentially distributed, the average ratios are more than 80% better for all simulated systems and as much as 400% better in 20 component systems, while when they are

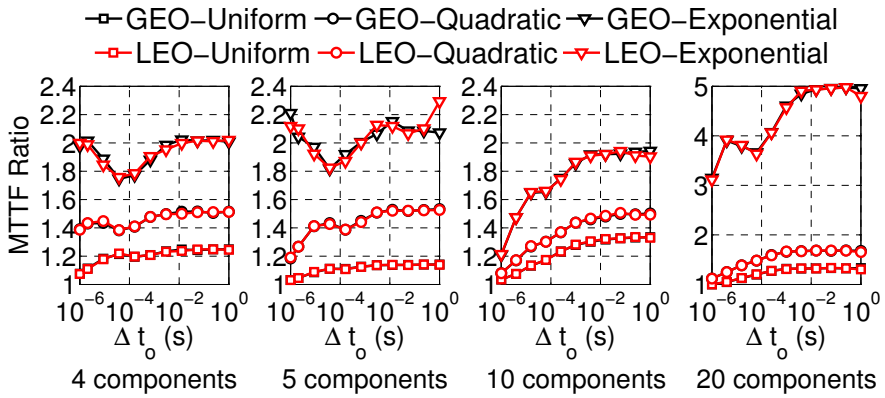


Figure 5.5: Average ratios of MTTFs for VRVC to those for round robin for systems consisting of 4, 5, 10 and 20 components in LEO (red) and GEO (black plots)

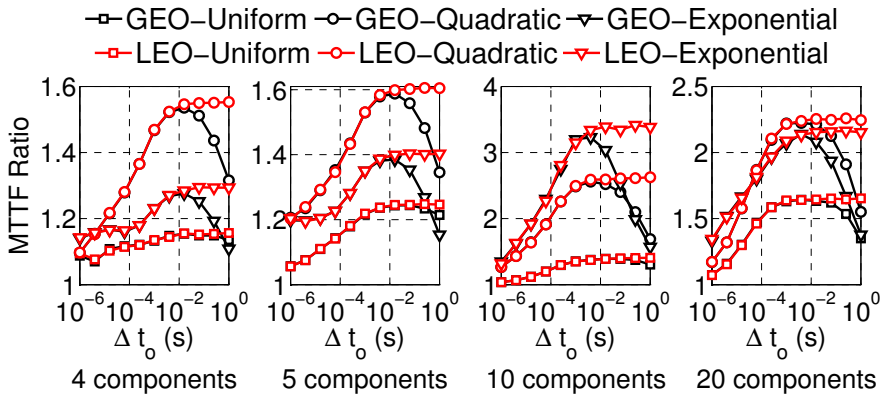


Figure 5.6: Average ratios of MTTFs for VRVC to those for VSE for systems consisting of 4, 5, 10, and 20 components for LEO (red) and GEO (black plots)

uniformly and quadratically distributed, the average ratios are up to 40% and 60% better, respectively.

We observed that the change in MTTF ratios were not consistent as Δt_o was varied. We believe this observation relates to the relative reconfiguration times of the modules, which, as seen in Equations (5.14), (5.16), (5.18), and (5.20), also influence system reliability.

To summarize, the VRVC approach was found to always be beneficial, but it is of greatest benefit when the system contains many TMR components that differ markedly in size.

5.4.2.2 VRVC vs VSE

In all simulated systems involving 4–20 components, the MTTFs for VRVC are higher than those for VSE and the gap between the two approaches increases as the voter observation time is increased (Figure 5.6). However, we also observed that the gap decreases when the voter observation time becomes greater than 10^{-2}s at very high radiation levels, as present in GEO. This is because the MTTFs eventually decline to 0, as partly shown in Figure 5.4(a), when the voter observation time becomes so large as to significantly increase the likelihood of component failures since recoveries are not undertaken frequently enough. Furthermore, since VSE adopts dynamic voter checks based on the number of essential bits per TMR component, the MTTF results of the VSE approach also vary and are unpredictable.

5.5 Experimental Analysis

In this section, we evaluate and compare the performance of the VRVC approach with that of VSE and round-robin voter checking when each are implemented on the experimental CubeSat payload, RUSH, which is described in detail in the Appendix. We assessed the reliability of the system using VRVC, VSE and round robin for voter checking in LEO and GEO. In the course of this assessment, we examined the response of the GA used to schedule VRVC as its parameters were fine tuned. Our experiments also gauged the trade-off between system reliability and power consumption for the three methods studied as the clock frequency of the RC was reduced. Finally, we conducted fault injection testing of the exemplar system in order to evaluate the mean time to detect errors using each of the three methods.

5.5.1 Experiments

5.5.1.1 Implementation

The RUSH payload, described in the Appendix, consists of a token-ring network [28], an RC and the 9 user application TMR components listed in Table 5.2. However, in this experiment, rather than using the token-ring network, we used the ICAP-based voter checking method described in Chapter 3. In addition, the RC was created with minimal features and could be operated at 100MHz, 50MHz, 20MHz, or 10MHz. The RC was used

Table 5.2: Results of mapping 9 TMR components to a Xilinx Artix-7 XC7A200TFBG-484 FPGA

Design	**Essential Bits** n_e	**RC t_r (ms) – # checks (d_k)**			
		100MHz	**50MHz**	**20MHz**	**10MHz**
BST3	1,833,235	26.7 – 47	49.5 – 45	72.4 – 47	118.7 – 49
SR3	1,403,647	19.6 – 41	43.8 – 40	64.0 – 39	104.9 – 46
BST2	793,534	11.0 – 28	24.5 – 31	35.8 – 34	58.7 – 36
SR2	515,904	8.5 – 27	21.7 – 29	31.7 – 33	52.0 – 29
SR1	285,914	6.8 – 26	13.6 – 24	19.9 – 25	32.6 – 25
BST1	281,604	2.6 – 23	5.9 – 23	8.6 – 20	14.0 – 25
BAQ	48,963	1.3 – 15	3.0 – 18	4.4 – 18	7.1 – 14
FIFO	41,842	3.5 – 12	7.8 – 12	11.4 – 13	18.7 – 13
FIR	12,042	1.2 – 08	2.6 – 11	3.9 – 10	6.3 – 11

to reconfigure faulty modules and was also used for flipping configuration memory bits during the fault injection experiment described in the next sub-section. Due to power limitations of the CubeSat that deploys the exemplar system, all components were operated at 10MHz.

The designs were implemented using Vivado 2014.4 with default settings.

5.5.1.2 Fault Injection

We performed a fault injection experiment to assess the MTTD in the RUSH system using each of the three voter checking schedules.

The RC was used to manage the fault injection process. The RC received a random configuration bit address generated by a host PC. The RC read the corresponding frame, flipped the addressed bit and wrote the frame back using the HWICAP to emulate the occurrence of a memory error due to an SEU. Note that we did not inject faults into the RC in order to avoid corrupting it during the experiment. Of the 18,300 configuration frames in the FPGA targeted in our study, 17,330 frames were contained in the design under test. Once a fault was injected, the RC waited for $10ms$ and checked the error status of all voters before reporting which component, if any, was in error.

5.5.2 Results and Discussion

5.5.2.1 Example Design Results

Table 5.2 lists the 9 TMR designs of the RUSH payload (please refer to Appendix A.2.2 for details of the designs). Table 5.2 also reports the number of essential bits (n_e) and the recovery times (t_r[2]) per triplicated module (t_r is the time interval between an error being detected in a module until the last word of the partial bitstream used to recover that module is written back to the FPGA via the AXI HWICAP IP). The table also reports the number of checks (d_k) made of each component per VRVC schedule period so that the system MTTF was maximized when the RC was operated at different clock frequencies under GEO radiation conditions (we observed similar d_k under LEO conditions). The period between successive voter observations (Δt_o) was $71\mu s$, $142\mu s$, $355\mu s$ and $711\mu s$ when the RC was operated at 100MHz, 50MHz, 20MHz and 10MHz, respectively. This is a consequence of the number of clock cycles needed by the RC at that frequency to process the instructions to check a voter.

Table 5.3 reports two metrics. The first is the MTTF in years for systems employing VRVC, VSE and round robin for voter checking in GEO. The second is the power consumption in mW of (i) the RC on its own, and (ii) the RC including the 9 components, when the RC is operated at different clock frequencies. The percentage reduction in power consumption, relative to the RC operating at 100MHz, is indicated in parentheses. This power consumption figure relates to the energy expended checking the voters at intervals of Δt_o, and therefore applies to all schedules equally.

We found that the TMR-MER system using VRVC is more reliable than the same system using round robin when the available power in the system is constrained. Table 5.3 shows that the system reliabilities (as given by the MTTF) are proportional to the rates at which the system recovers from errors. However, for the sake of saving energy in space-based applications during long missions, the voter checking frequency can be reduced [66]. For example, when the RC runs at 10MHz compared to 100MHz, the energy consumption of the RC alone is reduced by 40% and that of the whole system is reduced by 25% (Table 5.3). In this case, the ratio of the MTTF achieved using VRVC to that obtained using round robin for voter checking increases from 118% for the RC operating at 100MHz to 129% at 10MHz. It can also be observed that the MTTFs of systems employing VRVC are greater than those that employ VSE at all four RC clock frequencies.

[2] t_r is also referred to as t_c in Chapters 3, 4 and the Appendix.

Table 5.3: MTTF and power consumption at various RC clock frequencies in GEO

RC operating frequency		**100MHz**	**50MHz**	**20MHz**	**10MHz**
MTTF (years)	Round-robin	103.0(-15%)	49.0(-15%)	28.0(-20%)	16.0(-23%)
	VSE	116.4(-4%)	54.9(-5%)	32.6(-7%)	19.2(-7%)
	VRVC	121.7(0%)	57.6(0%)	35.1(0%)	20.7(0%)
Power (mW)	RC	252(0%)	196(-22%)	163(-35%)	152(-40%)
	RC+TMR comps	456(0%)	394(-14%)	357(-22%)	344(-25%)

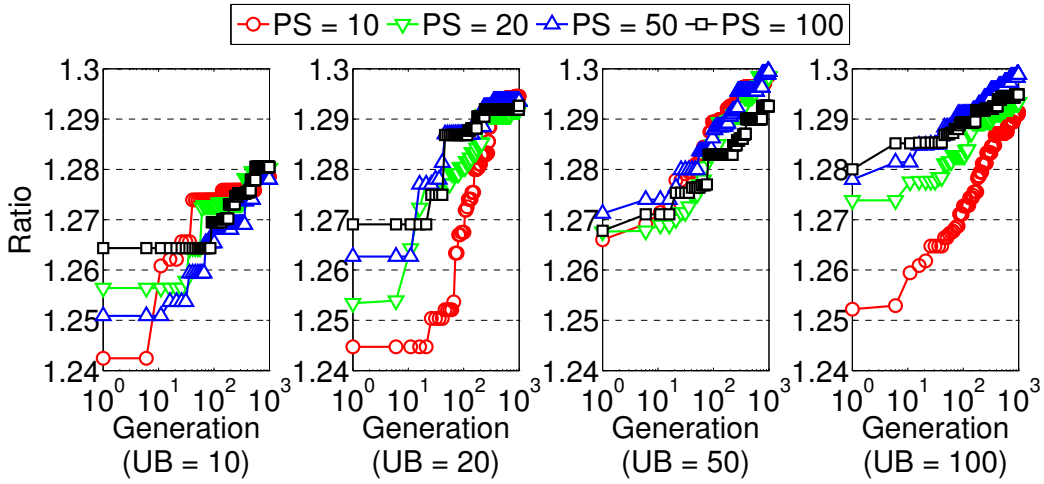


Figure 5.7: Ratio of MTTF for VRVC to MTTF for round robin for the exemplar system while the number of generations, the population size (PS), and the upper bound (UB) of the initial check rate is varied.

Figure 5.7 plots the ratio of the MTTF for systems employing VRVC to the MTTF for those checking voters in round-robin order as the parameters of the proposed GA are varied. The parameters we varied included the maximum number of generations, the population size (PS), and the upper bound value (UB) for the number of checks to be performed per period in the initial population. The crossover rate and the mutation rate were left at 25% and 10%, respectively. Here we show the results of the RC operating at 10 MHz under GEO conditions. As can be seen in Figure 5.7, the GA tends to attain a similar optimal result in most cases. We also observed similar trends when the crossover rate was set to 25%, 50% or 75% with respect to the mutation rate was set to 10%, 50% and 90%.

The experiments show that the UB affects the performance of our proposed GA. This is because when UB is small (e.g., 10), the genetic diversity is limited and the GA is likely to become stuck in a local optimum. When UB is large (e.g., greater than 50), the GA is

more likely to find the global optimum. On the other hand, PS affects the starting point of the GA, but it does not significantly affect the final results.

To summarise, in order to find a schedule that maximizes the system reliability, we found that the proposed GA should be initialized with a small PS (e.g., 10) to aid rapid evaluation, a modest UB (e.g., 50) for the sake of obtaining short schedule lengths and to assist in finding the optimal result, and a large number of generations (e.g., greater than 1000) to have a good chance of attaining the optimal result.

5.5.2.2 Fault Injection Results

On average, VRVC allows errors to be detected 44% faster than with round robin and 30% faster than when VSE is used to check voters. Table 5.4 provides the average number of errors in each component that we found after four trials of one million injected faults. Table 5.5 tabulates the MTTD errors using the round-robin, VSE and VRVC approaches as well as the percentage reduction from round robin and VSE to VRVC when the RC is operated at different clock frequencies. The MTTDs are calculated using Equation (5.27) with the number of checks listed in Table 5.2, and in Table 4.3, together with the average number of errors per component, as tabulated in Table 5.4.

Table 5.4: Average number of errors found in components

Design	**# Errors e_k (%)**
BST3	38,828 (39.1)
SR3	26,701 (26.9)
BST2	13,830 (13.9)
SR2	9,643 (9.7)
SR1	4,522 (4.6)
BST1	4,053 (4.1)
BAQ	684 (0.7)
FIFO	696 (0.7)
FIR	396 (0.4)

5.5.3 Further Discussion

It is of some concern that much of the additional logic used to implement and support TMR-MER, such as the RC, the RCN and voters, may be implemented in a non-redundant manner and therefore introduce single points of failure. Nevertheless, irrespective of the

Table 5.5: Mean time to detect errors

Configuration \ Δt_o	**71**μ**s**	**142**μ**s**	**355**μ**s**	**711**μ**s**
Round robin (μ**s)**	320(-39%)	639(-37%)	1596(-45%)	3200(-53%)
VSE (μ**s)**	290(-26%)	580(-25%)	1451(-31%)	2905(-39%)
VRVC (μ**s)**	230(0%)	465(0%)	1105(0%)	2088(0%)

configuration memory error recovery approach taken, FPGA-based TMR systems inevitably include non-redundant components, such as the clock network, internal state registers, ICAP and off-chip ports, which also introduce single points of failure when used. Therefore, in order to further improve system reliability, the non-replicated modules should be triplicated, if possible, so as to be protected from SEUs along with the other TMR components of the system. Since these components may be distributed across the device, the standard partial reconfiguration design flow [183] cannot be used to recover them in a modular fashion. One solution is to use FMER [3], which combines scrubbing and MER to recover configuration memory errors in SoCs that combine both single points of failure and triplicated sub-systems. In this case, our reliability model can be applied to find a voter checking schedule that enhances, if not maximizes, system reliability.

A limitation of the reliability models proposed in this chapter is that the number of failure cases increases exponentially with the number of TMR components in the system, which impacts on the scalability of our approach. Approximation methods that reduce the complexity of the reliability models should be considered in the future. It should be noted that other approaches, such as Markov models [4,100], face the same problem, since in them the number of states increases exponentially with increasing component numbers.

It is known that if TMR components have similar vulnerability in terms of the number of essential bits and recovery times, checking voters in round robin order maximizes the system reliability. However, most systems consist of a variety of components with different vulnerabilities as illustrated in the example RUSH system, and partitioning these TMR components into similar sizes is a challenge. In this case, checking the component voters at different rates based on the vulnerability of each TMR component generally improves the system reliability while reducing the partitioning burden.

Finally, it should be noted that in our work we have neglected the additional system vulnerability that accrues from the memory used to store the schedule. However, this overhead is small, being on the order of tens of bytes, and therefore does not pose a concern for overall system reliability, particularly since it can also be readily protected using ECC, such as SECDED available in modern FPGAs.

5.6 Final Remarks on Static Scheduling of Voter Checks in TMR-MER Systems

In this chapter, we have presented reliability models for TMR-MER systems that consist of an arbitrary number of components whose voters are checked in either round-robin order or at variable rates. We have proposed a genetic algorithm to derive a voter checking schedule that has the potential to significantly enhance the system reliability. We assert that any FPGA-based TMR system that uses a reconfiguration control network to provide random access to component voters can benefit from using variable-rate scheduling to prioritize checks of more vulnerable components. The benefits become more significant as the radiation level increases and/or as the checking frequency decreases.

The results show that using VRVC improves the mean time for the system to fail by up to 400% compared to checking voters in a round-robin manner. The results also show that the MTTFs of TMR-MER systems employing VRVC are greater than those that employ a VSE to choose the next component to check at run time. Moreover, we have shown that the power consumption of TMR-MER systems can be significantly reduced by reducing the clock frequency of the RC without compromising system reliability. Finally, through fault injection testing, we have demonstrated that the mean time to detect errors can be reduced by 44% and 30%, respectively, when VRVC is used instead of round robin and VSE.

Chapter 6

Conclusions

This chapter concludes this thesis. It starts with concluding remarks on the contributions of this thesis, in which we highlight the main findings of each chapter. Then, we propose three directions for future work to further enhance the system reliability estimation described in Chapter 5 and to further improve system reliability by considering user-defined metrics such as the criticality of a TMR component.

6.1 Concluding Remarks

Reliable space-borne digital systems implemented using COTS SRAM-based FPGAs and programmable SoCs require rapid, low-energy, flexible and reliable SEU mitigation techniques. In this thesis, we focused on the use of TMR combined with *Module-based configuration memory Error Recovery* (MER) to mitigate the effects of configuration memory SEUs. The use of TMR-MER requires a *Reconfiguration Control Network* (RCN) to mediate *Reconfiguration Requests* (RRs) from the system's voters to a *Reconfiguration Controller* (RC). Typically, the RC observes the RRs raised by voters as they are checked in round-robin order. However, TMR-MER systems may contain many different TMR components with different sensitivities. Intuitively, the more highly sensitive components should be observed for errors more frequently in order to detect and correct errors more quickly where they are most likely to occur, and thus improve system reliability. In this thesis, the main objective has therefore been to discover the best order in which to check voters so as to maximize system reliability.

Chapter 2 first explained the radiation challenges for COTS SRAM-based FPGAs deployed

in space, which should be carefully considered by system designers. It then provided information on SRAM-based FPGA architectures (i.e., configuration memory, Block RAM memory, user flip-flops and internal proprietary state) and their vulnerability in radiation environments. Chapter 2 also provided an overview of existing SEU mitigation techniques for each memory type in SRAM-based FPGAs. Since configuration memory accounts for the largest proportion of memory within these devices, more than four times as much as all the other memory types considered together, we mainly provided an overview of the mitigation techniques (e.g., TMR-MER and TMR with configuration memory scrubbing (TMR-Scrubbing)) on configuration memory errors. Intuitively, TMR-MER has several benefits over TMR-Scrubbing in terms of system reliability and power consumption, but the design of TMR-MER systems is more complicated. We therefore focused on researching the TMR-MER approach with the aim of maximizing its system reliability. We also provided the related work with a view to using novel techniques in our work to improve system reliability; this helped us to understand the state-of-the-art techniques and to propose new techniques for improving the reliability of TMR-MER systems. Finally, we provided background information on both practical and theoretical approaches to measuring the effectiveness of SEU mitigation techniques. By reviewing the literature, this chapter highlighted research gaps that we aimed to fill with this thesis.

In Chapter 3, we outlined the implementation of reliable TMR-MER systems including TMR components, an RC and an RCN. We first detailed the implementation of a TMR component as well as the design of the voters that need to be used in TMR-MER systems. We then provided a survey of RCs that have been implemented in the literature. Finally, we provided a comprehensive study of RCNs available in the literature. We concluded that the in-built configuration memory update network provides better overall reliability than any soft network implemented in user logic. The main reason for this is that use of this in-built network avoids use of wire and switch resources in the user logic that would be vulnerable to SEUs, particularly because it is difficult and expensive to protect distributed soft networks using TMR design techniques.

In Chapter 4, we proposed an approach for dynamically scheduling the voter checks in TMR-MER systems based on the vulnerability of each TMR component at the next check time. The proposed approach, which we called the *Voter Scheduling Engine* (VSE), chooses the TMR component that is most likely to have suffered an error to be checked next. We asserted that any RCN that supports random voter checks can benefit from using the VSE. The results showed that use of a VSE is generally more reliable than checking voters in the conventional round-robin fashion. The failure probabilities of the former are up to 50% lower than those of the latter for a 30-day mission in GEO orbit with solar min

and worst week conditions, and up to 45% and 30% lower with worst day and peak 5-min conditions, respectively.

After proposing the VSE for dynamically scheduling voter checks, the question was raised whether it was possible to statically schedule voter checks in order to eliminate the need of the VSE module and further improve system reliability. Chapter 5 answered this question by providing reliability models for TMR-MER systems where voter statuses are checked in round-robin order or at a variable rate and by proposing a genetic algorithm to find a static schedule using *Variable-Rate Voter Checks* (VRVC). Using the derived reliability models and the genetic algorithm, we demonstrated that the use of VRVC can further improve the system reliability compared to systems using the VSE or that the energy used to check the correct functioning of the system can be reduced using VRVC without compromising on reliability.

By using reliability models and fault-emulation test procedures on numerous synthetic applications and on the RUSH payload, this thesis has demonstrated that a TMR-MER system should use an ICAP-based RCN and triplicate the voter statuses in order to improve the system reliability. We also found that the use of round robin to schedule voter checking in TMR-MER systems is not necessarily the best, instead checking voters at specific rates based on the vulnerability of TMR components results in much higher system reliability. We have also devised methods that can cope with all naturally occurring levels of radiation in the vicinity of the Earth by considering multiple coincident SEUs, which are more likely to occur in high radiation environments. Theoretically, we found that TMR-MER systems with triplicated RCNs be more reliable than those employing TMR-Scrubbing (Chapter 3). However, to strengthen this claim and to further understand and compare the relative merits between TMR-MER and TMR-Scrubbing, we aim to conduct practical experiments such as conducting radiation testing on the ground and deploying both methods to enable in-flight comparison in a future CubeSat mission. During these tests, apart from configuration memory errors that can be recovered, we expect that we may observe some unrecoverable errors. These unrecoverable errors may involve (1) errors that have occurred in the internal proprietary state, which have resulted in the clearing of configuration memory and loss of state data (power-on-reset SEFI) and loss of communication with configuration logic (ICAP SEFI); (2) errors that have occurred in the configuration memory of the ICAP interface, which have resulted in loss of communication with configuration logic; and (3) errors that have occurred in the external memory controller interface, which impacts on fetching the correct partial bitstreams in order to recover the faulty modules from faults.

6.2 Future Work

6.2.1 Criticality-aware Scheduling of Voter Checks

The study on scheduling of voter checks is solely based on the number of essential bits and the recovery time needed to repair detected errors with the assumption that the "criticality" of all TMR components is the same. In practice, some components, such as clock managers, may be more critical than others since failures on these components affect the operation of the whole system. One direction for further study is to incorporate the criticality level of components as a user-defined input to the reliability models of TMR-MER systems in order to improve the accuracy of the system reliability estimation and to explore mitigation approaches that boost the reliability of the most critical subsystems.

6.2.2 Adaptive Scheduling of Voter Checks

Another direction for future work involves proposing models to estimate the system reliability in real environments. As we saw in Table 2.2 of the background chapter, there are many different levels of radiation in space environments. For example, in GEO orbit sometimes the radiation level is extremely high and at other times it can be approaching levels normally only present in LEO orbit [123]. Thus, by adapting the scheduling of voter checks based on the radiation level, we aim to maintain the system reliability at a high level while reducing the power consumption. This can be achieved by adjusting the voter checking frequency, which has been demonstrated in both Chapters 4 and 5.

6.2.3 Pre-empting the Recovery of Less Critical Components to Recover Errors in More Critical Components

Last but not least, we suggest studying the merits of schedules that pre-empt lengthy reconfigurations of low criticality components to check critical components so as to help boost reliability and reduce overall energy consumption. This idea could be incorporated with the user-defined criticality level that we suggested above. There are a couple of reasons why we suggest this for future work. The first is that TMR-MER systems may include many differently-sized applications, some of which require a long recovery time but are not critical, whereas others are small and thus have short recovery times, but are also critical to the system. Secondly, we cannot check voter statuses using the ICAP-based

approach while a module is being recovered. Intuitively, to improve system reliability, the critical components need to be prioritized for checking in order to detect errors occurring in them and to correct these errors as soon as possible. Therefore, pre-empting the recovery of modules with long recovery times and low criticality in order to check high-criticality components appears to be worthwhile for further consideration.

Appendix

In this Appendix, we describe the implementation of one of two configurations used in an FPGA-based TMR system deployed on an experimental CubeSat payload [2]. The Appendix provides an overview of the QB50 RUSH (*Rapid recovery from SEUs in Reconfigurable Hardware*) payload, which is one of three in-house designed payloads incorporated into the UNSW-EC0 QB50 CubeSat [2] (Section A.1). Section A.2 details the implementation of the TMR-MER system designed for the RUSH payload, while Section A.3 gives a specific overview of design considerations when the TMR-MER configuration is implemented in the RUSH board. Section A.4 reports the resource utilization of the TMR-MER system and relevant results when the VSE described in Chapter 4 and the VRVC described in Chapter 5 were implemented in the RUSH board. Section A.5 summarizes the Appendix.

A.1 The QB50 RUSH Payload

The QB50 project [165], funded through the European Union Framework Programme 7 (FP7) and overseen by the Von Karman Institute (VKI) in Belgium, envisaged the launch of around 50, 2U and 3U CubeSats into Low Earth Orbit (LEO) with the aim of providing a temporal and spatial image of the largely unexplored lower thermosphere. The individual CubeSats of the QB50 mission have been developed by various universities around the world compliant with the QB50 requirements [166], one of which was to carry one of the three VKI sensor payloads. RUSH is one of three additional payloads that were developed for the UNSW-EC0 QB50 CubeSat [2], which is shown in Figure A.1. The primary objective of the RUSH payload is to demonstrate and validate new approaches to rapidly recovering from SEUs in reconfigurable hardware. The experimental goals of the RUSH payload are to:

- Demonstrate and validate the partial reconfiguration approach to rapidly recovering from SEUs in reconfigurable hardware;
- Compare reconfiguration time and power consumption of TMR-Scrubbing with that of TMR-MER;
- Map SEU event occurrences in the thermosphere; and
- Demonstrate in-orbit reconfiguration.

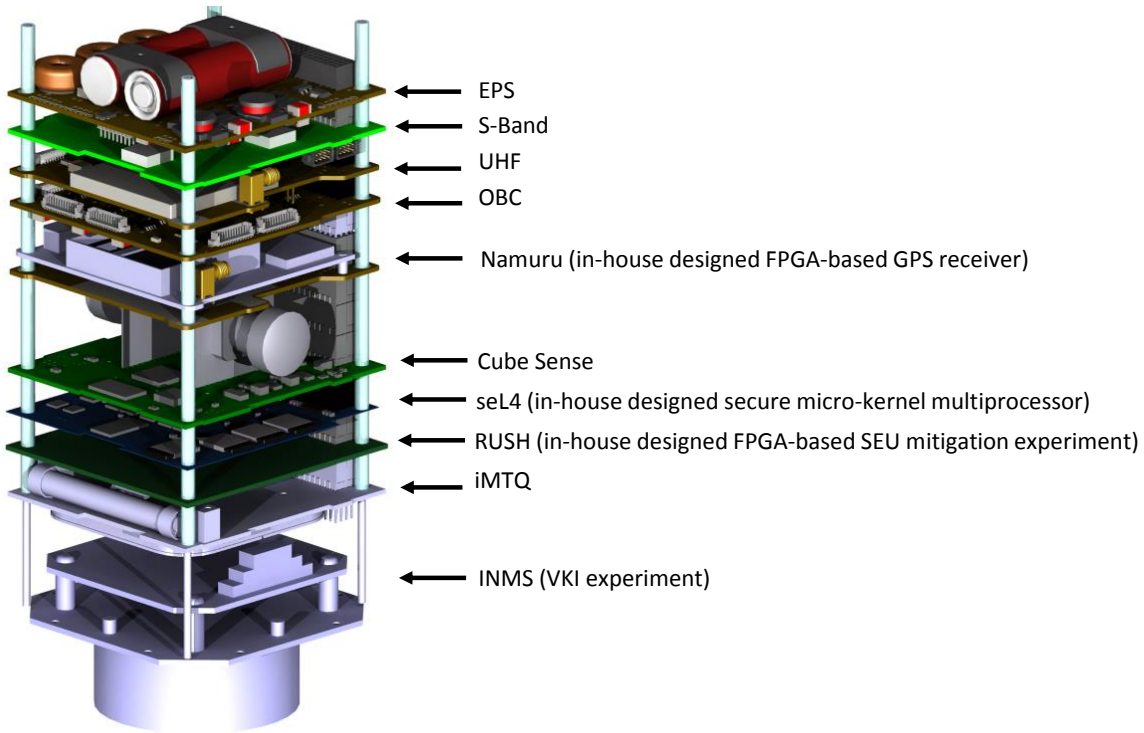


Figure A.1: UNSW-EC0 CubeSat Structure

Figure A.2 illustrates a block diagram of the RUSH payload. As can be observed from Figure A.2, at the heart of the RUSH payload design is a Xilinx Artix-7 XC7A200T FPGA, chosen for its high logic density to power consumption ratio. The FPGA is connected to a flash device, which is partitioned into two sections that are used to store two configurations: one for a TMR-MER configuration, and the other for a TMR-Scrubbing configuration. The FPGA is connected via a UART interface to a *Microcontroller Unit* (MCU), which acts as an interface between the FPGA and the UNSW-EC0 CubeSat system bus, and communicates with the *On-Board Computer* (OBC) via I^2C. The MCU oversees the overall operation of the RUSH payload, controls the power-up/down of the FPGA and logs SEU

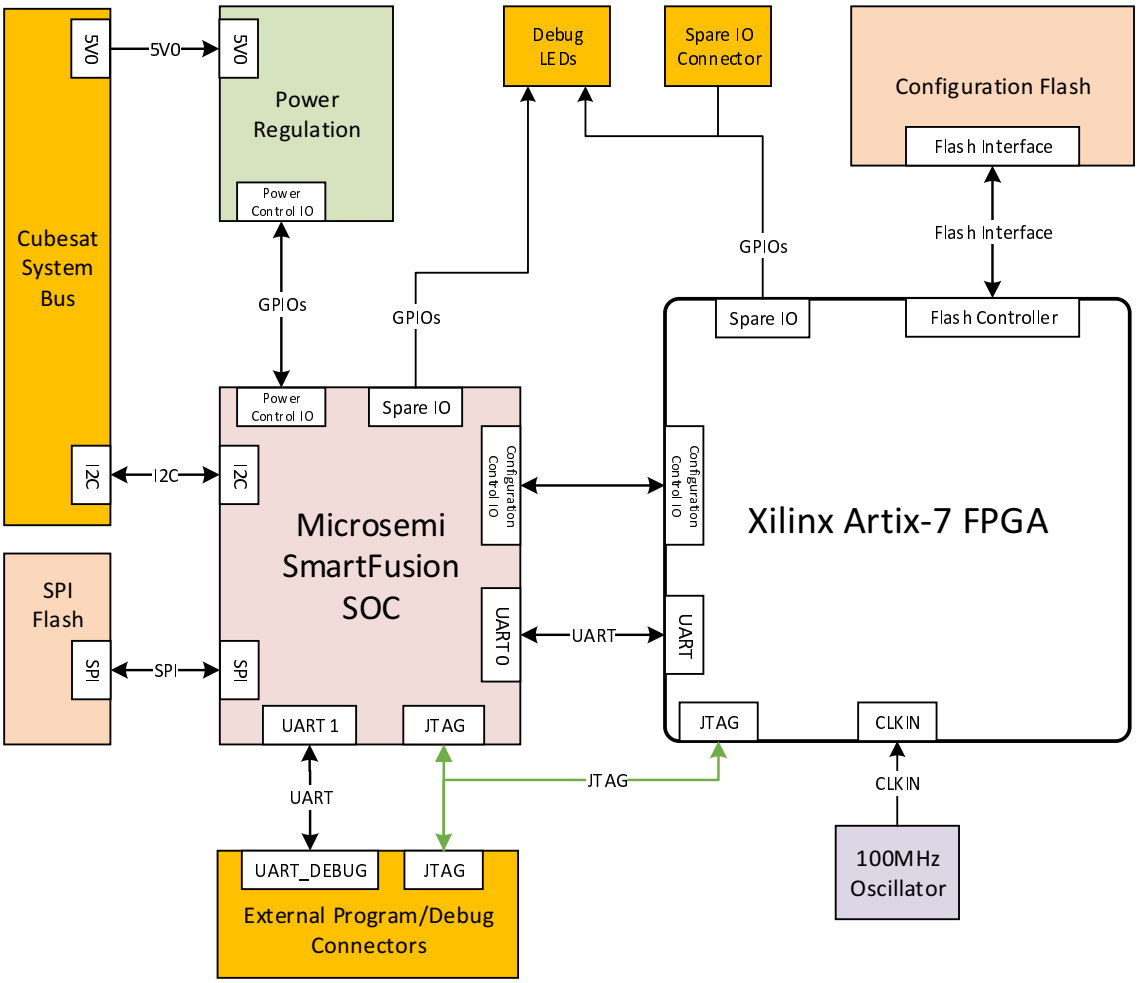


Figure A.2: High level block diagram of RUSH payload

detection and recovery statistics. To fulfil the requirements for the MCU in the proposed design, a Microsemi SmartFusion 2 *System-On-Chip* (SoC) was selected. Furthermore, since the SoC is based on non-volatile FLASH memory it is resilient to SEUs [104]. A small number of additional components provide ancillary functions such as providing regulated power, clock sources, programming interfaces and status indicators.

The primary objective of the RUSH experiment is to test and validate a TMR-MER approach and to compare the performance of TMR-MER with that of TMR-Scrubbing, implemented using the Xilinx SEM controller [188]. To this end, two configurations were developed that are essentially identical in terms of their resource utilization, whereby one configuration employs TMR-MER to guard against and recover from soft errors in user logic and configuration memory, and the other configuration utilizes the Xilinx SEM controller to continuously scan and scrub the FPGA configuration memory. To enable

comparison of SEU susceptibility and recovery, the two configurations incorporate essentially the same user circuitry, but the TMR-Scrubbing configuration does not partially reconfigure its triplicated components.

The experiment is intended to play a vital role in testing the susceptibility of Artix-7 FPGAs in low-earth orbit, and to demonstrate the use of dynamic partial reconfiguration on an FPGA in space. The user design is composed of 9 components chosen for their utilization of all FPGA resource types (LUTs, FFs, DSPs, and BRAMs). These components are replicated to fill the FPGA area, thereby creating the largest possible surface for SEUs to occur with this type of device. During the experiment SEU events are logged by the MCU and the time, location, and time to recover from them will be transmitted to the Earth when UNSW-EC0 passes over any of the ground stations available for the QB50 mission. Due to power limitations of the UNSW-EC0, the RUSH experiment will not run continuously. To deal with this, the available uptime will be evenly distributed between the two configurations. Furthermore, activity of both configurations will be scheduled such that they occur at similar times and locations.

In this Appendix, the payload that employs the TMR-MER approach (the so-called TMR-MER configuration) is described.

A.2 TMR-MER Configuration

In this section, we provide details of the implementation of the TMR-MER configuration. We start by providing an overview of the TMR-MER system that is deployed in the UNSW-EC0 QB50 CubeSat. We then describe the TMR-MER configuration including the implementation of all TMR components, of the reconfiguration controller and of the interface between the FPGA and the micro-controller.

A.2.1 Overview

Figure A.3 depicts an overview of the TMR-MER configuration that was implemented in the RUSH payload. The configuration uses a token-ring network [28] to perform the function of the Reconfiguration Control Network, a *Microblaze System* (MBS) playing the role of the Reconfiguration Controller and the MCU interface. The token-ring network comprises a *Network Arbiter* (NA) node and 9 user application nodes, including a Finite Impulse Response filter, a Block Adaptive Quantiser, and other TMR components. These

user application nodes are triplicated and all outputs are voted on by majority voters. The MBS comprises a MicroBlaze soft processor [177], a DMA controller (DMAC) [182], a Flash controller [185] and a HWICAP [178]. The MBS is used to control the dynamic partial reconfiguration of the FPGA. The MCU interface is a standard logic design implementing an FSM

The typical operation of the TMR-MER configuration is as follows. If an error occurs in one module of a TMR component, a *Reconfiguration Request* (RR) is triggered by the voter and transferred through the token-ring RCN network until the RR reaches the NA node[1]. The NA node decodes the RR and sends the corresponding module ID (M_ID) and the request signal (R_requested) to the MicroBlaze processor. The processor then issues a command to the DMAC that fetches the bitstream associated with the faulty module from the flash memory, and writes the bitstream to the HWICAP in order to reconfigure the erroneous module. The MicroBlaze processor then issues a reconfiguration done signal (R_done) to the NA indicating that state synchronization can commence. Meanwhile, the MCU interface receives signals from the MBS and the NA and sends them to the MCU to record the current TMR-MER system status.

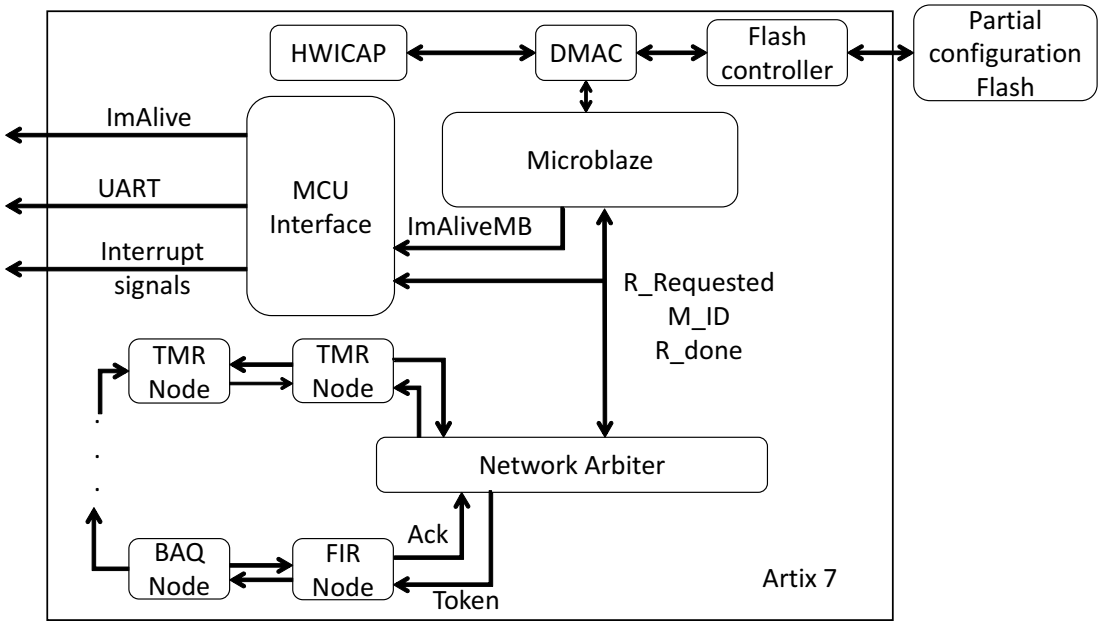


Figure A.3: High level block diagram of TMR-MER payload

[1] The details of how the RR is transferred through the nodes are described in [28]

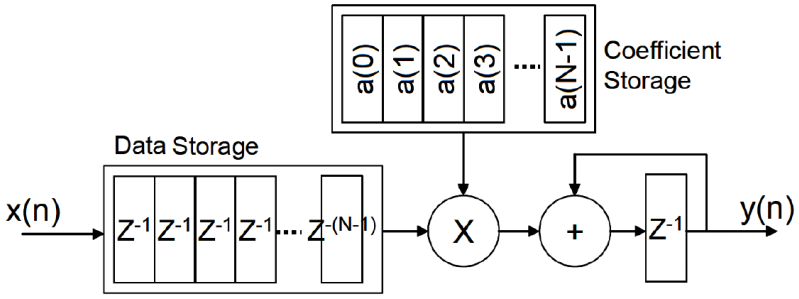


Figure A.4: A single accumulator FIR block diagram

A.2.2 TMR Components

Apart from the NA, the MBS and the MCU interface, the TMR-MER configuration also contains 9 TMR components. These TMR components include a single MAC-based 21-tap *Finite Impulse Response* (FIR) filter with 16-bit signal width, an 8-to-3-bit *Block Adaptive Quantizer* (BAQ), an 8,096-word deep 32-bit *FIFO*, three 32-bit *Shift Registers* (SRs) having different lengths and a range of combinational logic between the stages, and three 32-bit *Binary Search Trees* (BSTs) of different heights and a variety of combinational logic at each node. These components were selected as being representative of circuits that are commonly included in space-based applications and that utilize a mixture of FPGA resources. Note that these components are not interconnected and therefore operate independently. The design of each TMR component is described below.

A.2.2.1 Finite Impulse Response Filter

An N-tap FIR filter can be implemented using any number of accumulators from 1 to N. Figure 4 illustrates a cyclic (temporal) design using just a single accumulator. Note that this design feeds the accumulated sum of the delayed inputs (buffered in the Data Storage block) and scaled by the corresponding coefficients (stored in the Coefficient Storage block) back to an adder. For an N-tap filter with a single accumulator, a filter output value is produced N clock cycles after each sample is input. The contents of the Data Storage block are then shifted by one delay unit and a new input can be processed. Not illustrated

in the figure is a finite state machine including a counter that controls the operation of the filter.

We implemented a 21-tap single accumulator FIR filter with 16-bit signal and 32-bit accumulator values.

A.2.2.2 Block Adaptive Quantizer

The most commonly used technique for on-board Synthetic Aperture Radar (SAR) data compression is the BAQ algorithm [15, 87]. With the BAQ approach, SAR data (both the In-phase (I) and Quadrature (Q) channels) are divided into blocks of size N and the standard deviation of each block is used to adapt the quantizer threshold values for that block. In this way, the raw SAR data, which is originally m-bits wide, is optimally quantized block-by-block using $n < m$-bits [15]. Figure A.5 depicts the BAQ architecture we used.

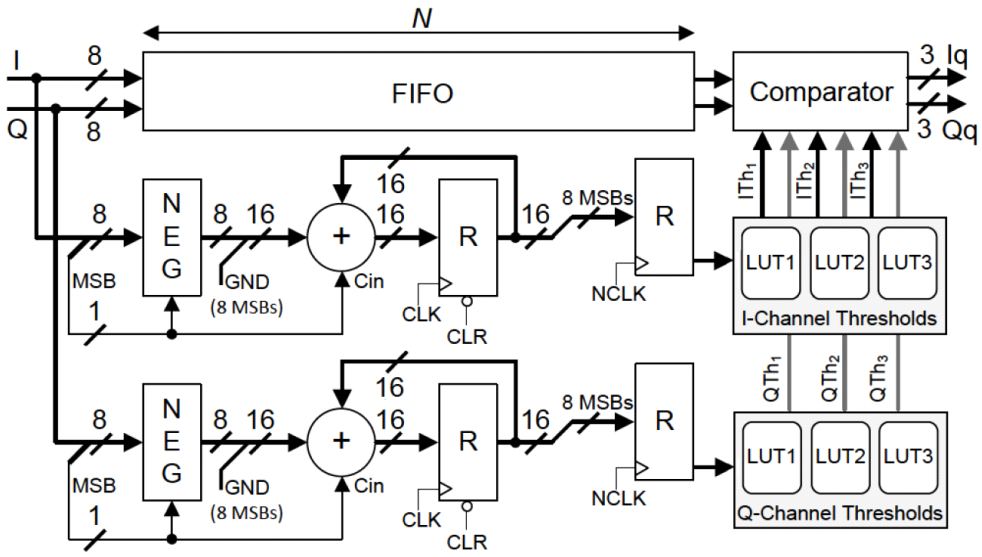


Figure A.5: BAQ architecture

In our design, N was chosen to be 256 samples, $m = 8$ and n was set to 3-bits. The standard deviation calculation was replaced with an average magnitude calculation, $|I|$, $|Q|$, to eliminate the need for a square root operation [87]. The quantization threshold values, ITh/QTh, were determined by calculating the average magnitude for a block of N raw SAR data samples and looking up the corresponding thresholds in Look-Up Tables

(LUTs). The design thus takes 256 cycles to calculate $|I|$ and $|Q|$ for a particular block and a further 256 cycles to realise the quantization. These two operations can be overlapped for successive blocks of N samples to obtain a continuous output stream.

A.2.2.3 Binary Search Trees

Figure A.6 shows an overview of a *Binary Search Tree* (BST) that is implemented in the TMR-MER configuration. The data input is shifted to all the leaves i.e., M_1 to M_N of the tree. The data output is the results of the smallest value of the data. We also implemented addition and multiplication in each node in order to utilize as many resources as possible.

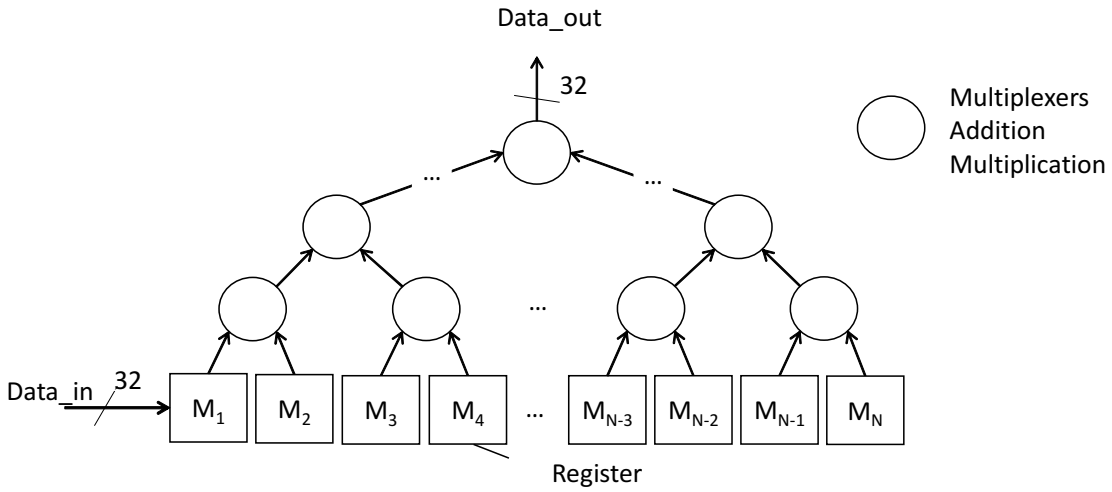


Figure A.6: Binary Search Tree Architecture

In the TMR-MER configuration, three BSTs of different heights and incorporating a variety of combinational functions at each node were implemented.

A.2.2.4 Shift Registers

Similar to the BSTs, we also implemented synthetic *Shift Registers* (SR) as illustrated in Figure A.7. A number of adders, multipliers and multiplexers were included at each stage to maximize the resource utilization.

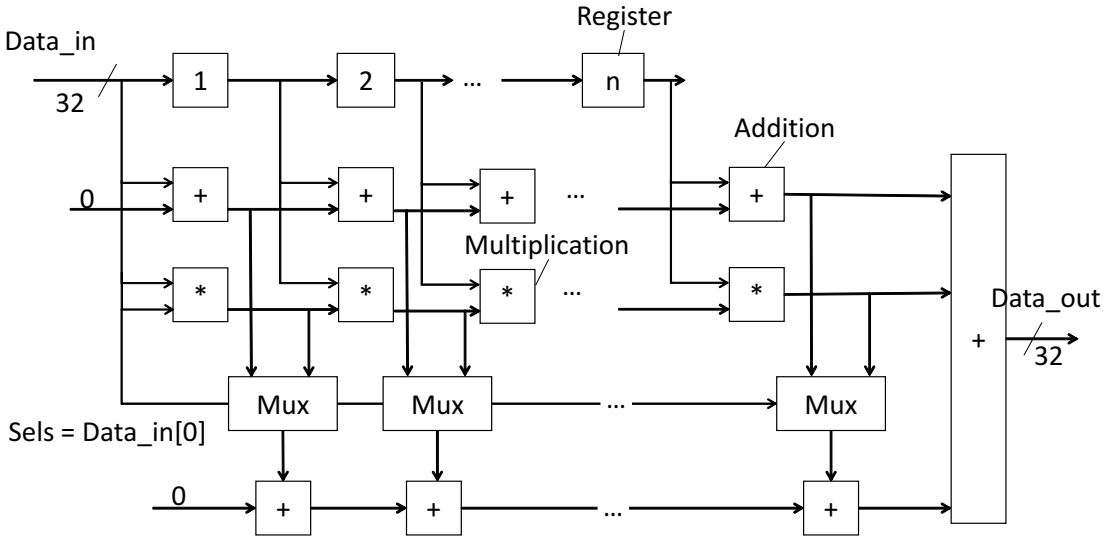


Figure A.7: Shift Register Architecture

A.2.2.5 FIFO

Of the components described so far, only the BAQ utilizes BRAMs in its design. However, the BRAM resources used by the BAQ design account for a very small portion of total block RAMs available in the FPGA. We therefore implemented a FIFO with 32-bit input/output and 8,096 word depth.

A.2.3 MicroBlaze Processor

The MicroBlaze processor in this system plays the role of a reconfiguration controller. The MicroBlaze serves three main functions. The first one is to observe the reconfiguration requests originating from the NA and to obtain the ID of the faulty module. When the MicroBlaze processor receives a reconfiguration request from the NA, it issues a command to the DMAC, which commences fetching the corresponding partial bitstream and writes it to the HWICAP. When reconfiguration is finished, the MicroBlaze processor generates a pulse of the done signal to the NA. This pulse indicates that the TMR component should commence synchronisation with its siblings. The last function is to produce heartbeat pulses for the MCU interface at 10ms intervals. The heartbeat indicates that the MicroBlaze processor is still functioning. If it is no longer detected by the MCU, the MCU power cycles the FPGA.

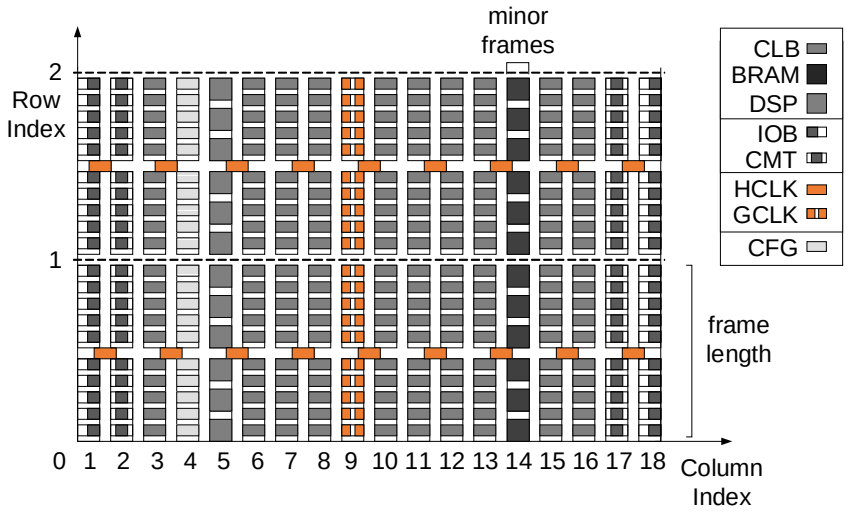


Figure A.8: Simplified 7-Series Xilinx FPGA layout showing two configuration rows and resources distributed across columns.

A.2.4 MCU Interface

The MCU interface is responsible for reporting the current status of the TMR-MER system to the MCU. An IamAlive signal indicates that the TMR-MER system is still operating correctly. UART signals are used to send module ID information, while interrupt signals help the MCU timestamp the error detection time associated with the reconfiguration request signal, and the error correction time associated with the done signal.

A.3 Design Considerations

A.3.1 Floor-planning

Programmable resources on a 7-Series Xilinx FPGA device, such as Configurable Logic Blocks (CLBs), Digital Signal Processing slices (DSPs), BRAMs, and Input/Output Buffers (IOBs), are tiled into columns [183] [180]. A simplified diagram of the layout of a 7-Series Xilinx FPGA device is shown in Figure A.8. The chip comprises several configuration rows, which correspond to the clock regions of the device and are indexed in ascending order from the centre of the device towards both its top and bottom edges. Figure A.8 depicts a section of the upper-left region of the device.

Floor-planning enables resource allocation within a specified region, which may be as

narrow as a single column. The placer can be instructed to only place the logic of part of a design (a logic block) within such a specified region. The router can also be constrained to only use the switch matrices within the region for the internal routing of the logic block. When a logic block and its internal routing are constrained in this way, the design can be partially reconfigured to recover from configuration memory errors that affect the block by just reconfiguring the configuration frames of that region.

A given FPGA-based TMR-MER design can have many different floorplans [20]. However, it is good practice to create a partition that makes most efficient use of interconnect and clocking resources [183]. It is suggested that the height of the partition should align to clock region boundaries, and that the left and right edges of the partition should be placed between two resource columns (for example, CLB-CLB, CLB-BRAM or CLB-DSP) and not between two interconnect columns (INT-INT). This allows the place and route tools the full use of all resources [183]. Please note that more floor-planning constraints can also be found in [183].

A.3.2 Full Bitstream and Partial Bitstream Layout

To implement the TMR-MER configuration, we used the partial reconfiguration design flow from Xilinx, which allows for the dynamic change of modules within an active design [183]. The TMR-MER configuration includes a full bitstream and partial bitstreams for reconfigurable modules.

In the TMR-MER configuration, all bitstreams are stored back-to-back in the flash off-chip memory. The layout of the full and partial bistreams can be seen in Figure A.9. To keep track of the partial bitstreams, the starting address (e.g., Address Module 1_1 denotes the starting address of module 1 of component 1) and the length (e.g., Length Module 1_1 denotes the length of the partial bitstream associated with module 1 of component 1) are used. They are stored in the MicroBlaze processor, which uses them to issue a command to the DMAC to fetch the corresponding partial bitstream of a faulty module.

A.4 Resource Utilization and Layout

Table A.1 presents the resource utilization with respect to LUT, FF, DSP and BRAM counts of the 9 TMR components implementing the TMR-MER configuration. It also reports the number of essential bits, the number of configuration frames and the correction

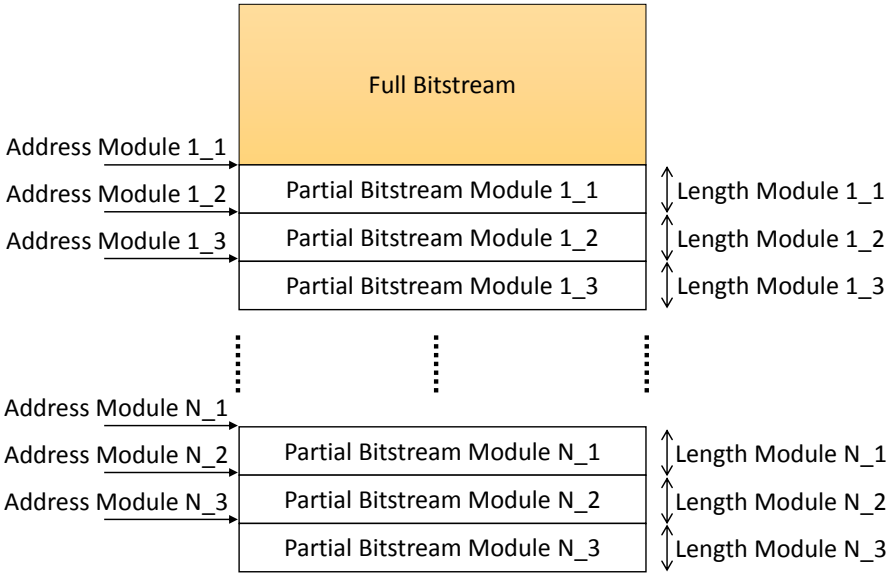


Figure A.9: The layout of a full bitstream and partial bitstreams in the external flash memory

time per module (t_c). Note that t_c is the time interval from when an error is detected in the module until the last word of the partial bitstream used to recover that module is written back to the FPGA via the AXI HWICAP IP. The layout of the 9 components and the location of their voters (-V), as well as that of the MicroBlaze-based reconfiguration controller (RC) is depicted in Figure A.10.

Table A.2 reports the resource utilization of the MBS in terms of LUTs, FFs and BRAMs. Note that the resource utilization does not include DSP usage because the MB was configured with minimal features.

A.5 Summary

In this Appendix, we have described the implementation of the TMR-MER system deployed in the UNSW-EC0 QB50 CubeSat. The design of the 9 TMR user components was presented along with that of the MicroBlaze soft processor and the MCU interface. We have also provided design considerations when a TMR-MER system is implemented in a Xilinx FPGA device and when the partial reconfiguration design flow is used. Finally, we have reported on the resource utilization of the TMR components and the MBS system. The layout of the design on the device has also been depicted.

Table A.1: Results of mapping 9 TMR components to a Xilinx Artix-7 XC7A200TFBG-484

Design	**Utilization**				**Essential Bit** ($\mathbf{n_e}$)	$\mathbf{n_f}$	$\mathbf{t_c}$ **(ms)**
	LUTs	**FFs**	**DSP**	**BRAM**			**MicroBlaze**
FIR	33 (0.02%)	16 (0.01%)	1 (0.13%)	–	12,042 (0.02%)	65	1.2
FIFO	72 (0.05%)	111 (0.04%)	–	7.5 (2.05 %)	41,842 (0.07%)	192	3.5
BAQ	305 (0.22%)	197 (0.07%)	–	–	48,963 (0.08%)	73	1.3
BST1	1,396 (1.04%)	2,519 (0.94%)	–	–	281,604 (0.46%)	145	2.6
SR1	1,619 (1.20%)	3,273 (1.22%)	–	–	285,914 (0.46%)	378	6.8
SR2	2,630 (1.96%)	5,499 (2.05%)	20 (2.70%)	–	515,904 (0.84%)	474	8.5
BST2	3,779 (2.84%)	6,198 (2.32%)	31 (4.18%)	–	793,534 (1.30%)	610	11.0
SR3	7,022 (5.24%)	14,573 (5.44%)	40 (5.40%)	–	1,403,647 (2.30%)	1,090	19.6
BST3	9,126 (6.82%)	12,214 (4.56%)	31 (4.18%)	–	1,833,235 (3.00%)	1,483	26.7

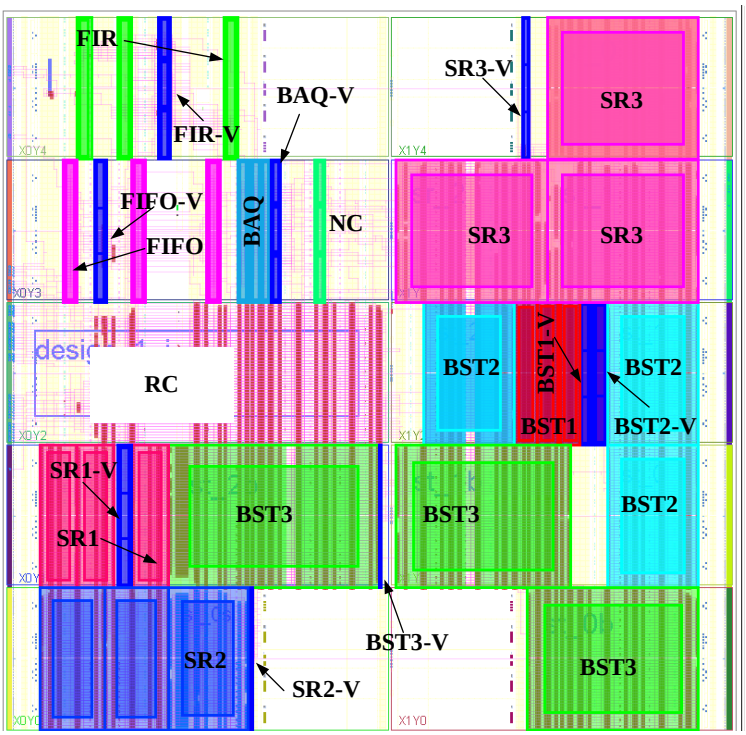


Figure A.10: System layout of RUSH payload

Table A.2: Results of mapping the MBS to a Xilinx Artix-7 XC7A200TFBG-484

Module	LUTs	FFs	BRAM
AXI DMA	1286 (0.96%)	1837 (0.68%)	2 (0.55%)
AXI EMC	821 (0.61%)	465 (0.17%)	0 (0%)
AXI HWICAP	338 (0.25%)	966 (0.36%)	1 (0.27%)
MicroBlaze	2165 (1.61%)	1799 (0.67%)	5 (1.37%)
AXI Peripheral	1715 (1.27%)	973 (0.36%)	0 (0%)
Local memory	8 (0.01%)	4 (0%)	8 (2.19%)
Others	16 (0.01%)	33 (0.01%)	0 (0%)
MBS (total)	6349 (4.72%)	6077 (2.26%)	16 (4.38%)

References

[1] "Open On-Chip Debugger – Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing." [Online]. Available: http://openocd.org/

[2] ACSER, "The UNSW-EC0 QB50-AU02 CubeSat." [Online]. Available: http://www.acser.unsw.edu.au/QB50

[3] D. Agiakatsikas, E. Cetin, and O. Diessel, "FMER: A hybrid configuration memory error recovery scheme for highly reliable FPGA SoCs," in *International Conference on Field-Programmable Logic*, Sept 2016, pp. 88–91.

[4] D. Agiakatsikas, N. T. H. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration control networks for TMR systems with module-based recovery," in *International Symposium on Field-Programmable Custom Computing Machines*, May 2016, pp. 88–91.

[5] M. A. Aguirre, J. N. Tombs, A. Torralba, and A. Fernández-León, "FT-UNSHADES: A new system for SEU injection, analysis and diagnostics over post synthesis netlist," in *International Conference of Military and Aerospace Programmable Logic Devices*, Sep 2005.

[6] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, D. M. Codinachs, S. Pastore, C. Poivey, G. R. Sechi, G. Sorrenti, and R. Weigand, "Experimental validation of fault injection analyses by the FLIPPER tool," *IEEE Transactions on Nuclear Science*, vol. 57, no. 4, pp. 2129–2134, Aug 2010.

[7] G. Allen, "Mitigation selection and qualification recommendations for Xilinx Virtex, Virtex-II, and Virtex-4 Field Programmable Gate Arrays," Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, Tech. Rep. 09-35 12/09, Dec 2009.

[8] G. Allen, G. Swift, and C. Carmichael, "Virtex-4 VQ static SEU characterization summary," Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration., Tech. Rep. 08-16 4/08, Apr 2008.

[9] L. Antoni, R. Leveugle, and B. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2000, pp. 405–413.

[10] G. H. Asadi and M. B. Tahoori, "Soft error mitigation for SRAM-based FPGAs," in *VLSI Test Symposium*, May 2005, pp. 207–212.

[11] G. Asadi and M. B. Tahoori, "Soft error rate estimation and mitigation for SRAM-based FPGAs," in *International Symposium on Field-programmable Gate Arrays*, 2005, pp. 149–160.

[12] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford, UK: Oxford University Press, 1996.

[13] T. Back, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*, 1st ed. Bristol, UK: IOP Publishing Ltd., 1997.

[14] H. J. Barnaby, "Total-ionizing-dose effects in modern CMOS technologies," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3103–3121, Dec 2006.

[15] U. Benz, K. Strodl, and A. Moreira, "A comparison of several algorithms for SAR raw data compression," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 5, pp. 1266–1276, Sep 1995.

[16] M. Berg, J. J. Wang, R. Ladbury, S. Buchner, H. Kim, J. Howard, K. LaBel, A. Phan, T. Irwin, and M. Friendlich, "An analysis of Single Event Upset dependencies on high frequency and architectural implementations within Actel RTAX-S family Field Programmable Gate Arrays," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3569–3574, Dec 2006.

[17] M. D. Berg, K. A. LaBel, and J. Pellish, "Single event effects in FPGA devices 2014-2015," Tech. Rep. GSFC-E-DAA-TN24313, Jun 2015.

[18] N. W. Bergmann and A. S. Dawood, "Reconfigurable computers in space: problems, solutions and future directions," in *Military and Aerospace Applications of Programmable Logic Devices Conference*, 2000.

[19] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Lopez, M. S. Reorda, and G. Squillero, "An industrial environment for high-level fault-tolerant structures insertion and validation," in *VLSI Test Symposium*, 2002, pp. 229–236.

[20] C. Bolchini, A. Miele, and C. Sandionigi, "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1744–1758, 2011.

[21] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," Mar 2008.

[22] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874–2878, Dec 1996.

[23] L. A. Cardona and C. Ferrer, "AC ICAP: A flexible high speed ICAP controller," *International Journal of Reconfigurable Computing*, Sep 2015.

[24] S. D. Carlo, P. Prinetto, P. Trotta, and J. Andersson, "A portable open-source controller for safe dynamic partial reconfiguration on Xilinx FPGAs," in *International Conference on Field Programmable Logic and Applications*, Sept 2015, pp. 1–4.

[25] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," *Xilinx Application Note (XAPP197)*, Jul 2006.

[26] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single event upsets through Virtex partial configuration," *Xilinx Application Note (XAPP216)*, 2000.

[27] E. Cetin, O. Diessel, L. Gong, and V. Lai, "Towards bounded error recovery time in FPGA-based TMR circuits using dynamic partial reconfiguration," in *International Conference on Field programmable Logic and Applications*, Sept 2013, pp. 1–4.

[28] E. Cetin, O. Diessel, L. Gong, and V. Lai, "Reconfiguration network design for SEU recovery in FPGAs," in *IEEE International Symposium on Circuits and Systems*, June 2014, pp. 1524–1527.

[29] E. Cetin and O. Diessel, "Guaranteed fault recovery time for FPGA-based TMR circuits employing partial reconfiguration," in *International Workshop on Computing in Heterogeneous, Autonomous 'N' Goal-oriented Environments*, 2012.

[30] E. Cetin, O. Diessel, T. Li, J. Ambrose, T. Fisk, S. Parameswaran, and A. Dempster, "Overview and investigation of SEU detection and recovery approaches for FPGA-based heterogeneous systems," in *FPGAs and Parallel Architectures for Aerospace Applications*, 2016, pp. 33–46.

[31] G. Cieslewski and A. D. George, "SPFFI: Simple portable FPGA fault injector," in *Military and Aerospace Programmable Logic Devices Conference*, 2009.

[32] G. Cieslewski, A. D. George, and A. Jacobs, "Acceleration of FPGA fault injection through multi-bit testing," in *International Conference of Engineering Reconfigurable System Algorithms*, 2010, pp. 218–224.

[33] M. Citterio, "FPGAs and High-Energy Physics," *presented at the SEPL workshop, Lausanne, Switzerland*, Sep 2016.

[34] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "Exploiting circuit emulation for fast hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 48, no. 6, pp. 2210–2216, Dec 2001.

[35] A. Corominas, W. Kubiak, and N. M. Palli, "Response time variability," *Journal of Scheduling*, vol. 10, no. 2, pp. 97–110, 2007.

[36] N. R. Council, *Earth Science and Applications from Space: National Imperatives for the Next Decade and Beyond*. Washington, DC: The National Academies Press, 2007.

[37] DDTC, "The International Traffic in Arms Regulations (ITAR)," Apr 2017.

[38] Digilent, "Nexys Video Artix-7 FPGA: Trainer board for multimedia applications."

[39] M. DORIGO, "Optimization, learning and natural algorithms," *Ph.D. Thesis, Politecnico di Milano, Italy*, 1992.

[40] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE Transactions on Software Engineering*, no. 4, pp. 438–444, 1984.

[41] A. Ebrahim, T. Arslan, and X. Iturbe, "On enhancing the reliability of internal configuration controllers in FPGAs," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2014, pp. 83–88.

[42] A. Ebrahim, K. Benkrid, X. Itrube, and C. Hong, "A novel high-performance fault tolerant ICAP controller," in *NASA/ESA Conference on Adaptive Hardware and Systems*, 2012, pp. 259–263.

[43] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A fast, flexible, and easy-to-develop FPGA-based fault injection technique," *Microelectronics Reliability*, vol. 54, no. 5, pp. 1000 – 1008, 2014.

[44] ECSS, "Space environment," *ESA-ESTEC, Standard ECSS-E-ST-10-04C*, 2008.

[45] L. D. Edmonds, "Analysis of single-event upset rates in triple-modular redundancy devices," Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, Tech. Rep. 09-6, Feb 2009.

[46] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999.

[47] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil, "Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection," *IEEE Transactions on Computers*, vol. 61, no. 3, pp. 313–322, March 2012.

[48] J. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Physica D: Nonlinear Phenomena*, vol. 22, no. 1, pp. 187 – 204, 1986, proceedings of the Fifth Annual International Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016727898690240X

[49] J. A. Felix, P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and G. L. Hash, "Radiation response and variability of advanced commercial foundry technologies," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3187–3194, Dec 2006.

[50] R. C. Ferguson and R. Tate, "Use of Field Programmable Gate Array technology in future space avionics," in *Digital Avionics Systems Conference*, vol. 2, Oct 2005.

[51] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula, "Radiation testing update, SEU mitigation, and availability analysis of the Virtex FPGA for space reconfigurable computing," in *IEEE Nuclear and Space Radiation Effects Conference*, 2000.

[52] A. Garca-Villoria and R. Pastor, "Solving the response time variability problem by means of a genetic algorithm," *European Journal of Operational Research*, vol. 202, no. 2, pp. 320 – 327, 2010.

[53] R. B. Gardenyes, "Trends and patterns in ASIC and FPGA use in space missions and impact in technology roadmaps of the European Space Agency," *Master's thesis, TU Delft, Delft, The Netherlands*, 2012.

[54] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, 1997.

[55] E. A. Ghazaani, Z. Ghaderi, and S. G. Miremadi, "A non-intrusive portable fault injection framework to assess reliability of FPGA-based designs," in *International Conference on Field-Programmable Technology*, Dec 2013, pp. 398–401.

[56] D. Goldberg, K. Deb, and B. Korb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex systems*, no. 3, pp. 493–530, 1989.

[57] L. Gong, A. Kroh, D. Agiakatsikas, N. T. H. Nguyen, E. Cetin, and O. Diessel, "Reliable SEU monitoring and recovery using a programmable configuration controller," in *International Conference on Field Programmable Logic and Applications*, 2017.

[58] L. Gong, T. Wu, N. T. H. Nguyen, D. Agiakatsikas, Z. Zhao, E. Cetin, and O. Diessel, "A programmable configuration controller for fault-tolerant applications," in *International Conference on Field-Programmable Technology*, Dec 2016, pp. 117–124.

[59] A. Gregerson, M. J. Schulte, and K. Compton, "High-Energy Physics," *Handbook of Signal Processing Systems*, pp. 135–169, 2013.

[60] A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed programmable FPGA Configuration through JTAG," in *International Conference on Field Programmable Logic and Applications*, Sep 2016, pp. 1–4.

[61] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, "Estimating soft processor soft error sensitivity through fault injection," in *International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 143–150.

[62] M. Havener and N. Hartl, "500+ Xilinx FPGAs Search for Elusive Higgs Boson at 1.5 Terabytes per Second," *Xcell journal*, 2002.

[63] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 99–104.

[64] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *IEEE Aerospace Conference*, 2008, pp. 249–258.

[65] O. Heron, T. Arnaout, and H. J. Wunderlich, "On the reliability evaluation of SRAM-based FPGA designs," in *International Conference on Field Programmable Logic and Applications*, Aug 2005, pp. 403–408.

[66] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for Xilinx Virtex FPGA configuration memory scrubbers," *IEEE Transactions on Nuclear Science*, vol. 60, no. 1, pp. 376–385, Feb 2013.

[67] D. Heynderickx, B. Quaghebeur, E. Speelman, and E. Daly, "ESAs SPace ENVironment Information System (SPENVIS): a WWW interface to models of the space

environment and its effects," *American Institute of Aeronautics and Astronautics*, vol. 371, 2000.

[68] D. M. Hiemstra and V. Kirischian, "Single event upset characterization of the Kintex-7 field programmable gate array using proton irradiation," in *IEEE Radiation Effects Data Workshop*, July 2014, pp. 1–4.

[69] A. Holmes-Siedle and L. Adams, *Handbook of radiation effects*. New York, NY (United States); Oxford University Press Inc., Jan 1993.

[70] Y. M. Hsu and E. E. Swartzlander, "Time redundant error correcting adders and multipliers," in *International Workshop on Defect and Fault Tolerance in VLSI Systems*, Nov 1992, pp. 247–256.

[71] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, Apr 1997.

[72] Y. Ichinomiya, S. Tanoue, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Improving the robustness of a softcore processor against SEUs by using TMR and partial reconfiguration," in *International Symposium on Field-Programmable Custom Computing Machines*, May 2010, pp. 47–54.

[73] K. Iniewski, *Radiation effects in semiconductors*. CRC Press, 2010.

[74] JEDEC, "Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices," *JEDEC Solid State Technology Association*, vol. 1, no. 6, p. 8, 2006.

[75] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2147–2157, Dec 2003.

[76] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *International Conference on Engineering of Reconfigurable Systems and Algorithms*. CSREA Press, 2002, pp. 68–73.

[77] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in FPGA-based designs," in *Aerospace Conference*, March 2008, pp. 1–11.

[78] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *International Symposium on Field Programmable Gate Arrays*, 2010, pp. 249–258.

[79] A. H. Johnston, "The influence of VLSI technology evolution on radiation-induced latchup in space systems," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 505–521, Apr 1996.

[80] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," in *International Conference on Field Programmable Logic and Applications*, Aug 2005, pp. 223–228.

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

[82] R. Katz, R. Barto, P. McKerracher, B. Carkhuff, and R. Koga, "SEU hardening of Field Programmable Gate Arrays (FPGAs) for space applications and device characterization," *IEEE Transactions on Nuclear Science*, vol. 41, no. 6, pp. 2179–2186, Dec 1994.

[83] R. Katz, K. LaBel, J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift, "Radiation effects on current field programmable technologies," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 1945–1956, 1997.

[84] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings., IEEE International Conference on Neural Networks*, vol. 4, Nov 1995, pp. 1942–1948.

[85] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[86] U. Kretzschmar, J. Gomez-Cornejo, A. Astarloa, U. Bidarte, and J. D. Ser, "Synchronization of faulty processors in coarse-grained TMR protected partially reconfigurable FPGA designs," *Reliability Engineering and System Safety*, vol. 151, pp. 1 – 9, 2016.

[87] R. Kwok and W. T. K. Johnson, "Block adaptive quantization of Magellan SAR data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 27, no. 4, pp. 375–383, Jul 1989.

[88] B. LaMeres, "Radiation Tolerant Computer Mission on the ISS (RTcMISS)," May 2017.

[89] R. Le, "Soft error mitigation using prioritized essential bits," *Xilinx Application note (XAPP538)*, 2012.

[90] D. S. Lee, M. Wirthlin, G. Swift, and A. C. Le, "Single-event characterization of the 28 nm Xilinx Kintex-7 Field-Programmable Gate Array under heavy ion irradiation," in *IEEE Radiation Effects Data Workshop*, July 2014, pp. 1–5.

[91] J. Y. Lee, C. R. Chang, N. Jing, J. Su, S. Wen, R. Wong, and L. He, "Heterogeneous configuration memory scrubbing for soft error mitigation in FPGAs," in *International Conference on Field-Programmable Technology*, Dec 2012, pp. 23–28.

[92] T. Li, H. Yang, G. Cai, T. Zhi, and Y. Li, "A CMOS triple inter-locked latch for SEU insensitivity design," *IEEE Transactions on Nuclear Science*, vol. 61, no. 6, pp. 3265–3273, Dec 2014.

[93] Y. Li, B. Nelson, and M. Wirthlin, "Reliability models for SEC/DED memory with scrubbing in FPGA-based designs," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2720–2727, Aug 2013.

[94] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *European Conference on Radiation and Its Effects on Components and Systems*, Sept 2001, pp. 275–282.

[95] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena, "Autonomous fault emulation: A new FPGA-based acceleration system for hardness evaluation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 1, pp. 252–261, Feb 2007.

[96] D. Lyons, *Sun screen*, Nov 2000. [Online]. Available: https://www.forbes.com/global/2000/1113/0323026a.html

[97] A. Magyar, *RISC-V in Verilog*, 2015.

[98] J. McCollum, "Radiation tolerant flash FPGA," Nov. 27 2001, US Patent 6,324,102.

[99] J. McCollum, "ASIC versus antifuse FPGA reliability," in *IEEE Aerospace Conference*, March 2009, pp. 1–11.

[100] D. McMurtrey, K. S. Morgan, B. Pratt, and M. J. Wirthlin, "Estimating TMR reliability on FPGAs using Markov models," *All Faculty Publications*, no. 149, Dec 2008.

[101] H. Michel, A. Belger, T. Lange, B. Fiethe, and H. Michalik, "Read back scrubbing for SRAM FPGAs in a data processing unit for space instruments," in *NASA/ESA Conference on Adaptive Hardware and Systems*, June 2015, pp. 1–8.

[102] Microsemi, "RT ProASIC3 FPGAs – the industry's first flash-based, radiation-tolerant FPGA for low-power space applications." [Online]. Available: https://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rt-proasic3

[103] Microsemi, "Radiation-Tolerant ProASIC3 Low Power Spaceflight Flash FPGAs with Flash*Freeze Technology," 2012, Datasheet Rev. 5.

[104] Microsemi, "SmartFusion Customisable System-on-Chip (cSoC)," 2013.

[105] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, Dec 2007.

[106] K. Morris, "FPGAs in Space - Programmable Logic in Orbit," in *Electronic Engineering Journal*, Aug 2004.

[107] K. Morris, "FPGAs in Space - Microsemi Launches New Space-bound Family," *Electronic Engineering Journal*, Apr 2015.

[108] NASA, "Space radiation effects on electronic components in low-earth orbit," Apr 1996.

[109] B. Navas, J. berg, and I. Sander, "The upset-fault-observer: A concept for self-healing adaptive fault tolerance," in *NASA/ESA Conference on Adaptive Hardware and Systems*, July 2014, pp. 89–96.

[110] G. L. Nazar, L. P. Santos, and L. Carro, "Fine-grained fast Field-Programmable Gate Array scrubbing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 23, no. 5, pp. 893–904, May 2015.

[111] N. T. H. Nguyen, E. Cetin, and O. Diessel, "Scheduling considerations for voter checking in TMR-MER systems," in *International Symposium on Field-Programmable Custom Computing Machines*, April 2017, pp. 30–30.

[112] N. T. H. Nguyen, D. Agiakatsikas, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration control networks for TMR systems with module-based recovery," *submitted to Microprocessors and Microsystems journal*, 2017.

[113] N. T. H. Nguyen, E. Cetin, and O. Diessel, "Dynamic scheduling of voter checks in FPGA-based TMR systems," in *International Conference on Field-Programmable Technology*, Dec 2016, pp. 169–172.

[114] N. T. H. Nguyen, E. Cetin, and O. Diessel, "Improving reliability of FPGA-based systems by scheduling checks for configuration memory errors," *submitted to IEEE Transactions on Aerospace and Electronic Systems*, 2017.

[115] N. T. H. Nguyen, E. Cetin, and O. Diessel, "Scheduling Considerations for Voter Checking in FPGA-based TMR Systems," School of Computer Science and Engineering, UNSW Sydney, Tech. Rep. UNSW-CSE-TR-201705, 05 2017.

[116] N. T. H. Nguyen, E. Cetin, and O. Diessel, "Scheduling voter checks to detect configuration memory errors in FPGA-based TMR systems," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct 2017.

[117] S. Niranjan and J. F. Frenzel, "A comparison of fault-tolerant state machine architectures for space-borne electronics," *IEEE Transactions on Reliability*, vol. 45, no. 1, pp. 109–113, Mar 1996.

[118] E. Normand, "Single-event effects in avionics," *IEEE Transactions on Nuclear Science, vol. 43, no. 2*, pp. 461–474, 1996.

[119] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science, vol. 43, no. 2*, pp. 2742–2750, 1996.

[120] C. D. Norton, T. A. Werne, P. J. Pingree, and S. Geier, "An evaluation of the Xilinx Virtex-4 FPGA for on-board processing in an advanced imaging system," in *IEEE Aerospace Conference*, March 2009, pp. 1–9.

[121] R. Nymmik, "Relationships among solar activity SEP occurrence frequency, and solar energetic particle event distribution function," in *International Cosmic Ray Conference*, vol. 6, 1999, p. 280.

[122] T. R. Oldham and F. B. McLean, "Total ionizing dose effects in MOS oxides and devices," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 483–499, June 2003.

[123] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, H. Quinn, and M. Wirthlin, "SRAM FPGA reliability analysis for harsh radiation environments," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3519–3526, Dec 2009.

[124] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: A survey and a cost model," *ACM Transactions Reconfigurable Technology Systems*, vol. 4, no. 4, pp. 36:1–36:24, Dec. 2011.

[125] C. Pilotto, J. R. Azambuja, and F. L. Kastensmidt, "Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications," in *Symposium on Integrated Circuits and System Design*, 2008, pp. 199–204.

[126] P. J. Pingree, "Advancing NASAs on-board processing capabilities with reconfigurable FPGA technologies," in *Aerospace Technologies Advancements*. InTech, 2010.

[127] M. Portela-Garcia, C. Lopez-Ongil, M. G. Valderas, and L. Entrena, "Fault injection in modern microprocessors using on-chip debugging infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, pp. 308–314, March 2011.

[128] S. Poulding and J. A. Clark, "Efficient software verification: Statistical testing using automated search," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 763–777, Nov 2010.

[129] M. Psarakis, A. Vavousis, C. Bolchini, and A. Miele, "Design and implementation of a self-healing processor on SRAM-based FPGAs," in *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct 2014, pp. 165–170.

[130] H. Quinn and P. Graham, "Terrestrial-based radiation upsets: a cautionary tale," in *Symposium on Field-Programmable Custom Computing Machines*, April 2005, pp. 193–202.

[131] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455–2461, Dec 2005.

[132] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037–2043, Dec 2007.

[133] H. Quinn and M. Wirthlin, "Validation techniques for fault emulation of SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 62, no. 4, pp. 1487–1500, Aug 2015.

[134] H. Quinn, "Radiation effects in reconfigurable FPGAs," *Semiconductor Science and Technology*, vol. 32, no. 4, p. 044001, 2017.

[135] H. Quinn, P. Graham, K. Morgan, Z. Baker, M. Caffrey, D. Smith, and R. Bell, "On-orbit results for the Xilinx Virtex-4 FPGA," in *IEEE Radiation Effects Data Workshop*, 2012, pp. 1–8.

[136] H. Quinn, P. S. Graham, K. Morgan, J. Krone, M. P. Caffrey, and M. J. Wirthlin, "An introduction to radiation-induced failure modes and related mitigation methods for Xilinx SRAM FPGAs," in *International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2008, pp. 139–145.

[137] H. Quinn, D. Roussel-Dupre, M. Caffrey, P. Graham, M. Wirthlin, K. Morgan, A. Salazar, T. Nelson, W. Howes, E. Johnson, J. Johnson, B. Pratt, N. Rollins, and J. Krone, "The Cibola Flight Experiment," *ACM Transactions on Reconfigurable Technology Systems*, vol. 8, pp. 3:1–3:22, Mar 2015.

[138] D. Ratter, "FPGAs on Mars," in *Xcell Journal*, Aug 2004.

[139] A. M. Saleh, J. J. Serrano, and J. H. Patel, "Reliability of scrubbing recovery-techniques for memory systems," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 114–122, Apr 1990.

[140] R. Santos, S. Venkataraman, and A. Kumar, "Scrubbing mechanism for heterogeneous applications in reconfigurable devices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, pp. 33:1–33:26, Feb 2017.

[141] A. Sari and M. Psarakis, "Scrubbing-based SEU mitigation approach for Systems-on-Programmable-Chips," in *International Conference on Field-Programmable Technology*, Dec 2011, pp. 1–8.

[142] A. Sari and M. Psarakis, "A fault injection platform for the analysis of soft error effects in FPGA soft processors," in *International Symposium on Design and Diagnostics of Electronic Circuits Systems*, April 2016, pp. 1–6.

[143] A. Sari, M. Psarakis, and D. Gizopoulos, "Combining checkpointing and scrubbing in FPGA-based real-time systems," in *VLSI Test Symposium*, April 2013, pp. 1–6.

[144] B. Schmidt, D. Ziener, J. Teich, and C. Zllner, "Optimizing scrubbing by netlist analysis for FPGA configuration bit classification and floorplanning," *Integration, the VLSI Journal*, vol. 59, pp. 98 – 108, 2017.

[145] M. L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design.* New York, NY, USA: John Wiley & Sons, Inc., 2002.

[146] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," *ACM Computing Surveys*, vol. 47, no. 2, pp. 37:1–37:34, Jan 2015.

[147] A. R. Simpson, G. C. Dandy, and L. J. Murphy, "Genetic algorithms compared to other techniques for pipe optimization," *Journal of water resources planning and management*, vol. 120, no. 4, pp. 423–443, 1994.

[148] J. D. Snodgrass, "Low-power fault tolerance for spacecraft FPGA-based numerical computing," *Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, USA*, 2006.

[149] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, June 2006.

[150] A. Stoddard, A. Gruwell, P. Zabriskie, and M. Wirthlin, "High-speed PCAP configuration scrubbing on Zynq-7000 All Programmable SoCs," in *International Conference on Field Programmable Logic and Applications*, Aug 2016, pp. 1–8.

[151] M. Straka, J. Kastil, and Z. Kotasek, "SEU simulation framework for Xilinx FPGA: First step towards testing fault tolerant systems," in *Euromicro Conference on Digital System Design*, Aug 2011, pp. 223–230.

[152] M. Straka, J. Kastil, Z. Kotasek, and L. Miculka, "Fault tolerant system design and SEU injection based testing," *Microprocessors and Microsystems, vol. 37, no. 2*, pp. 155–173, 2013.

[153] G. Swift and G. Allen, "Virtex-5QV static SEU characterization summary," NASA Jet Propulsion Laboratory, Tech. Rep., May 2013.

[154] G. Swift, C. Carmichael, G. Allen, R. Monreal, G. Madias, and E. Miller, "Virtex-5QV architectural features SEU characterization summary," NASA Jet Propulsion Laboratory, Xilinx, Tech. Rep., Aug 2013.

[155] L. A. Tambara, F. L. Kastensmidt, J. R. Azambuja, E. Chielle, F. Almeida, G. Nazar, P. Rech, C. Frost, and M. S. Lubaszewski, "Evaluating the effectiveness of a diversity TMR scheme under neutrons," in *European Conference on Radiation and Its Effects on Components and Systems*, Sept 2013, pp. 1–5.

[156] J. Tarrillo, F. L. Kastensmidt, P. Rech, C. Frost, and C. Valderrama, "Neutron cross-section of N-modular redundancy technique in SRAM-Based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 61, no. 4, pp. 1558–1566, Aug 2014.

[157] J. Teifel, "Self-voting dual-modular-redundancy circuits for single-event-transient mitigation," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3435–3439, Dec 2008.

[158] J. Tonfat, F. L. Kastensmidt, P. Rech, R. Reis, and H. M. Quinn, "Analyzing the effectiveness of a frame-level redundancy scrubbing technique for SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 3080–3087, Dec 2015.

[159] W. J. Townsend, J. A. Abraham, and E. E. Swartzlander, "Quadruple time redundancy adders," in *Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov 2003, pp. 250–256.

[160] R. Trautner, "ESA's roadmap for next generation payload data processors," in *International Space System Engineering Conference*, May 2011, pp. 1–5.

[161] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2150–2160, Dec 1997.

[162] A. Vavousis, A. Apostolakis, and M. Psarakis, "A fault tolerant approach for FPGA embedded processors based on runtime partial reconfiguration," *Journal of Electronic Testing*, vol. 29, no. 6, pp. 805–823, Dec 2013.

[163] F. Veljkovic, T. Riesgo, and E. de la Torre, "Adaptive reconfigurable voting for enhanced reliability in medium-grained fault tolerant architectures," in *NASA/ESA Conference on Adaptive Hardware and Systems*, June 2015, pp. 1–8.

[164] K. Vipin and S. A. Fahmy, "A high speed open source controller for FPGA partial reconfiguration," in *International Conference on Field-Programmable Technology*, Dec 2012, pp. 61–66.

[165] VKI, *https://www.qb50.eu/*.

[166] VKI, "QB50 System Requirements and Recommendations and Interface Control Document, Issue 7," Feb 2015.

[167] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[168] J. J. Wang, S. Kaptanoglu, S. Varela, N. Rezzak, S. Cui, H. Pan, J. McCollum, F. Hawley, and E. Hamdy, "SEU and SET in radiation-hardened flash-based FPGA RTG4," *presented at the SEPL workshop, Lausanne, Switzerland*, Sep 2016.

[169] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, *The RISC-V Instruction Set Manual Volume I: User-Level ISA*, 2014.

[170] D. Weigand and M. Harlacher, "A radiation-tolerant low-power transceiver design for reconfigurable applications," in *the Earth Science Technology Conference*, 2002.

[171] Wikipedia, "Ionizing radiation."

[172] Wikipedia, "ISO 26262."

[173] M. Wirthlin, "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond," *The IEEE*, vol. 103, no. 3, pp. 379–389, March 2015.

[174] C. Wolf, "PicoRV32 - a size-optimized RISC-V CPU," 2016.

[175] Xilinx, "PicoBlaze 8-bit Embedded Microcontroller User Guide (UG129)," 2011.

[176] Xilinx, "Partial Reconfiguration User Guide (UG702)," Jul 2012.

[177] Xilinx, "MicroBlaze Processor User Guide (UG984)," 2014.

[178] Xilinx, "AXI HWICAP Product Guide (PG134)," 2015.

[179] Xilinx, "Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide (UG953)," Jun 2015.

[180] Xilinx, "7–Series FPGAs configuration User Guide (UG470)," Sep 2016.

[181] Xilinx, "7–Series FPGAs memory resources User Guide (UG473)," Sep 2016.

[182] Xilinx, "AXI DMA Product Guide (PG021)," Oct 2016.

[183] Xilinx, "Vivado design suite user guide: Partial reconfiguration User Guide (UG909)," Apr 2016.

[184] Xilinx, "7–Series FPGAs overview, Data Sheet (DS180)," Mar 2017.

[185] Xilinx, "AXI EMC Product Guide (PG100)," Apr 2017.

[186] Xilinx, "Device Reliability Report (UG116)," Apr 2017.

[187] Xilinx, "Radiation-hardened, space-grade Virtex-5QV family data sheet: Overview (DS192)," Apr 2017.

[188] Xilinx, "Soft Error Mitigation Controller Product Guide (PG473)," Apr 2017.

[189] X. Yao, N. Hindman, L. T. Clark, K. E. Holbert, D. R. Alexander, and W. M. Shedd, "The impact of total ionizing dose on unhardened SRAM cell margins," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3280–3287, Dec 2008.

[190] C. Yui, G. M. Swift, C. Carmichael, R. Koga, and J. S. George, "SEU mitigation testing of Xilinx Virtex II FPGAs," in *Radiation Effects Data Workshop*, 2003, pp. 92–97.

[191] A. Zeineddini and J. Wesselkamper, "PRC/EPRC: Data integrity and security controller for partial reconfiguration," *Xilinx Application Note (XAPP887)*, Jun 2012.

[192] Z. Zhao, "Reconfiguration control networks for FPGA-based space applications," *Master's thesis, UNSW Sydney, Australia*, Nov 2016.

[193] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, no. 4, pp. 366–427, Dec. 1997.