Reconfiguration Control Networks for FPGA-based Space Applications

Zhuoran Zhao

A thesis in fulfillment of the requirements for the degree of

Masters by Research

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

November 2016

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Zhao**

First name: **Zhuoran** Other name/s:

Abreviation for degree as given in the University calendar: **Masters by Research**

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

Title: Reconfiguration Control Networks for FPGA-based Space Applications

Abstract

Space processing applications deployed on SRAM-based Field Programmable Gate Arrays (FPGAs) are vulnerable to radiation-induced Single Event Upsets (SEUs). To efficiently mitigate configuration memory upsets, Triple Modular Redundancy (TMR) with Module-based Error Recovery (MER) has been proposed. TMR-MER systems rely on a Reconfiguration Control Network (RCN) to aggregate reconfiguration requests from voters that are distributed across the device to a central Reconfiguration Controller (RC), which performs partial reconfiguration on an affected module. This thesis focuses on the choice and impact of these RCNs on the system's reliability and performance.

This thesis evaluates the suitability, utilization and performance of a token ring network in fulfilling the requirements of an RCN as the number of network connections is increased. As RCNs are not limited to a ring, we then carry out a comprehensive study and comparison of four possible RCN topologies, namely, a star, a bus, a token ring and an ICAP-based readback approach, with respect to their utilization, latency and SEU sensitivity as the network capacity increases. We evaluate these networks by implementing them on synthetic layouts as well as an experimental payload based on a Xilinx Artix-7 device and by injecting faults into these implementations. We show that of the RCN topologies studied, the ICAP-based approach is the most reliable, despite having the highest latency, while the star network also proves to be a strong contender due to its high performance.

In our study, we found the reliability of TMR-MER to be significantly less than that of TMR-S unless the RCN was also triplicated and recovered. To more effectively localize errors in RCNs or other triplicated sub-components, this thesis proposes a fine-grained module error recovery scheme that without additional system hardware can localize and correct errors that classic MER fails to identify. We evaluate our proposal via a fault-injection campaign on a Xilinx Artix-7 application circuit and compare the reliability, the recovery latency and the energy cost of repairing errors, of our proposed method with those of a conventional MER approach and with periodic and on-demand blind scrubbing. We find the reliability of our proposal to be the highest and the energy expenditure to be the lowest amongst those methods considered.

The thesis concludes with directions for further study.

Signature **Zhuoran Zhao** Witness **Oliver Diessel** Date **Nov. 16, 2016**

FOR OFFICE USE ONLY Date of completion of requirements for Award

Originality Statement

Zhuoran Zhao
Nov. 30, 2016

Abstract

Space processing applications deployed on SRAM-based Field Programmable Gate Arrays (FPGAs) are vulnerable to radiation-induced Single Event Upsets (SEUs). To efficiently mitigate configuration memory upsets, Triple Modular Redundancy (TMR) with Module-based Error Recovery (MER) has been proposed. TMR-MER systems rely on a Reconfiguration Control Network (RCN) to aggregate reconfiguration requests from voters that are distributed across the device to a central Reconfiguration Controller (RC), which performs partial reconfiguration on an affected module. This thesis focuses on the choice and impact of these RCNs on the system's reliability and performance.

This thesis evaluates the suitability, utilization and performance of a token ring network in fulfilling the requirements of an RCN as the number of network connections is increased. As RCNs are not limited to a ring, we then carry out a comprehensive study and comparison of four possible RCN topologies, namely, a star, a bus, a token ring and an ICAP-based readback approach, with respect to their utilization, latency and SEU sensitivity as the network capacity increases. We evaluate these networks by implementing them on synthetic layouts as well as an experimental payload based on a Xilinx Artix-7 device and by injecting faults into these implementations. We show that of the RCN topologies studied, the ICAP-based approach is the most reliable, despite having the highest latency, while the star network also proves to be a strong contender due to its high performance.

In our study, we found the reliability of TMR-MER to be significantly less than that of TMR-S unless the RCN was also triplicated and recovered. To more effectively localize errors in RCNs or other triplicated sub-components, this thesis proposes a fine-grained module error recovery scheme that without additional system hardware can localize and correct errors that classic MER fails to identify. We evaluate our proposal via a fault-injection campaign on a Xilinx Artix-7 application circuit and compare the reliability, the recovery latency and the energy cost of repairing errors, of our proposed method with those of a conventional MER approach and with periodic and on-demand blind scrubbing. We find the reliability of our proposal to be the highest and the energy expenditure to be the lowest amongst those methods considered.

The thesis concludes with directions for further study.

Acknowledgements

First of all, I would like to thank my supervisor, Assoc. Prof. Oliver Diessel, for continuously pushing me towards finishing this work. Thank you for your patience and timely feedback, as well as your effort in correcting my language errors and intellectualizing my direction whenever I lost my way.

I would also thank his colleague, Dr Ediz Cetin, who helps me on discussing my research, reviewing my papers, and most importantly, cheering me up when I was defeated by life. I would like to thank my co-supervisor, Dr Guo Hui, who was always willing to listen to my complaints. I thank all of my fellows in the Reconfigurable System Group for their help throughout my study.

Lastly, special thanks to my parents who support and understand all of my self-willed decisions, and kindly listen to my whispers about the hardship of life and work.

Abbreviations

ASIC Application-Specific Integrated Circuit

BAQ Block Adaptive Quantizer

BST Binary Search Tree

CLB Configurable Logic Block

COTS Commercial, Off-The-Shelf

CRC Cyclic Redundancy Check

DICE Dual Interlocked storage Cell

DMA Direct Memory Access

DPR Dynamic Partial Reconfiguration

DUT Design-Under-Test

ECC Error Correcting Code

FDPR Fine-grained Dynamic Partial Reconfiguration

FF Flip-Flop

FIR Finite Impulse Response

FPGA Field Programmable Gate Array

FSM Finite State Machine

GEO Geostationary Earth Orbit

ICAP Internal Configuration Access Port

IP Intellectual Property

ITAR International Traffic in Arms Regulations

LEO Low Earth Orbit

LUT Look-Up-Table

MTTR Mean Time To Recovery

NC Network Controller

NI Network Interface

NRE None-Recurring Engineering

PIP Programmable Interconnection Point

RC Reconfiguration Controller

RCN Reconfiguration Control Network

Rdone Reconfiguration done

RReq Reconfiguration Request

RRhdr Reconfiguration Request header

RUSH Rapid recovery from SEUs in Reconfigurable Hardware

SD Standard Deviation

SEL Single Event Latchup

SET Single Event Transient

SEU Single Event Upset

SEM Soft Error Mitigation

SerDes Serializer/Deserializer

SM Switch Matrix

SoC System-on-Chip

SOI Silicon-On-Insulator

SR Shift Register

TID Total Ionizing Dose

TMR Triple Modular Redundancy

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

TMR-S Triple Modular Redundancy with configuration memory error Scrubbing

VLSI Very Large Scale Integration

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital systems for space have traditionally been implemented using radiation-hardened devices. In recent years, interest has grown in using Commercial, Off-The-Shelf (COTS) SRAM-based Field Programmable Gate Arrays (FPGAs), particularly for low-cost Low Earth Orbit (LEO) applications, to reduce the cost and to enhance the flexibility and performance of these systems. These applications commonly rely on Triple Modular Redundancy (TMR) to mask the effects of radiation-induced Single Event Upsets (SEUs) in the application circuits, in which user data that is stored in flip-flops may become corrupted by ionizing radiation [1]. The configuration memory of these devices, which constitute 90% of the device resources, are also susceptible to radiation-induced corruption. The errors in the configuration memory may result in changes to the circuit state and the implemented circuit itself. Due to the accumulation of errors in the configuration memory, the triplication of the circuit's application may fail when more than two modules of the TMR component fail.

Two approaches have emerged in the literature to deal with this problem. Configuration memory scrubbing (TMR-S) periodically scans the entire device and corrects configuration memory errors by rewriting the memory frames containing them [2]. While scrubbing occurs periodically, whether or not errors are present, substantial time and energy are wasted on overwriting frames that are correct. With the decreasing feature size of CMOS transistors, upsets occur more readily in the configuration memory, but scrub cycles take more time due to the increased device capacity. Module-based error recovery (TMR-MER), on the other hand, reconfigures the frames of a TMR module when an error in its configuration memory is detected. This approach relies upon repeated detection of an error by the same TMR voter to trigger a reconfiguration of the module presenting the error [3].

Both methods are based on Dynamic Partial Reconfiguration (DPR), which facilitates the dynamic reading and writing of the configuration memory without interruptions.

There are a number of TMR-MER systems that have been reported in the literature, e.g. [3–11]. Most of these systems rely upon a dedicated Reconfiguration Control Network (RCN) to relay error requests from the voters in the system to a central Reconfiguration Controller (RC) [12]. 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 Recovery (MTTR) from errors in the system, and the sooner the module is recovered, the lower the likelihood that the protection provided by TMR fails. 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, in this thesis, we focus our attention on the design of an RCN for high performance and low resource utilization so as to reduce its sensitivity to upsets. 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.

To achieve high system reliability, it is also possible to triplicate the RCN and to detect whether or not an RCN is corrupted due to SEUs. The triplication of the RCN generates a set of error reports which we refer to as an error signature, and as shown in this thesis, when errors manifest in different components, such as modules, voters, RCN components, and other interconnection routes, different error signatures are generated. In order to localize these errors, the thesis demonstrates how the triplicated RCNs contribute to improving the precision and robustness of the TMR-MER technique. However, the error recovery of RCN components is complicated by the conventional design flow of TMR-MER not supporting the recovery of non-block oriented components. The interconnection routes of the RCN cannot therefore be recovered using this method. Whereas the work presented in [12] triggers a scrub cycle when a configuration error is detected in the RCN, this thesis proposes a fine-grained dynamic partial reconfiguration (FDPR) design flow that enables recovering routing components without resorting to scrubbing.

1.1 Contributions

The work described in this thesis aims to discover improved RCN designs with respect to utilization and performance, to demonstrate how triplicated RCNs contribute to localizing errors, and to describe how a corrupted RCN is recovered other than by scrubbing the device. The key contributions of this thesis are:

- To study and explore the Token Ring Network design proposed in [13] for the purpose of reconfiguration control with the aim of evaluating its utilization and performance as the number of network connections is increased;
- To carry out a comprehensive study and comparison of four alternative and realistic RCN topologies, namely, a star network, a bus, a token ring network and an ICAP-based readback approach, as proposed in the literature — the networks are compared in terms of logic and routing utilization, latency and SEU sensitivity as the network capacity increases — and to choose the best approach from among the interconnection methods studied;
- To demonstrate how the RCN is triplicated to enhance reliability; to explain how the response to the triplicated error requests should be prioritized; and to describe a fine-grained method for dynamically reconfiguring an incorrect RCN component;
- To further evaluate this fine-grained method and to compare the reliability, latency and energy cost of correcting configuration memory errors using the proposed approach with (1) an equivalent TMR-MER system that resorts to complete scrubbing of the device when errors are detected outside the TMR modules, (2) on-demand scrubbing of the device when any SEU is detected in the system, and (3) periodic scrubbing of the device as a fault prevention and correction strategy.

1.2 Publications

- D. Agiakatsikas, N. T. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration Control Networks for TMR Systems with Module-based Recovery," in *2016 IEEE 24rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 2016, pp. 88–91 [12]
- N. T. 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 Special Issue on Reconfigurable Computing and FPGA Technology, Journal of Parallel and Distributed Computing [14]
- Z. Zhao, D. Agiakatsikas, N. T. Nguyen, E. Cetin, and O. Diessel, "Fine-grained Module-based Error Recovery in FPGA-based TMR Systems," Accepted to 2016 International Conference on Field-Programmable Technology (FPT) [15]

My contribution towards developing the first and the second publications [12, 14] involves the investigation of four possible RCN architectures, including star networks, buses, a token ring network and an ICAP-based approach, as described in the literature; the redesign of these network topologies for evaluation; implementation of the four networks in a checkerboard-like synthetic layout and on the RUSH experimental payload [16]; and the setup of the fault injection campaign.

In the third publication [15], my contribution includes proposing the fine-grained module-based error detection and recovery scheme that enhances the error localization and reconfigurability of the TMR-MER technique; to demonstrate the advantages of the proposed method using a test circuit; a fault injection campaign in order to identify the critical bits of different components and to record error signatures caused by these errors; and lastly, to compare the reliability, latency and energy cost of correcting configuration memory errors using the proposed approach, with those of the three more conventional approaches previously mentioned.

1.3 Thesis Outline

The thesis is organized as follows. Chapter 2 provides motivation for the use of Commercial, Off-The-Shelf (COTS) SRAM-based FPGAs to the development of low-cost satellite systems. Chapter 2 also explains the radiation challenges and how radiation experiments prove that COTS FPGAs can be an alternative to radiation-hardened devices. We then summarize the advantages of the rapid SEU mitigation approach — TMR-MER over the traditional TMR-S.

Chapter 3 explains the early explorations on the Token Ring Network design [13] and evaluates its performance and utilization as the network size scales up. Thereafter, we propose an optimized alternative that has higher performance and lower resource utilization, when compared with the original design.

Chapter 4 investigates possible RCN topologies from the literature and proposes four network architectures based on the star topology, a bus, a token ring network and using the in-built FPGA configuration network (ICAP) for evaluation. We compare the latency, utilization, and upset sensitivity of the four alternatives by mapping them to a synthetic layout and to the RUSH experimental payload [16], as well as by injecting faults into each network.

Chapter 5 presents a fine-grained module-based error detection and correction approach based on triplicated RCNs, which improves the precision of the error detection of the TMR-MER technique and enhances its error correction capabilities in order to avoid resorting to scrubbing. We evaluate our method by implementing it on a typical application circuit, and via a fault injection campaign, we identify the critical bits of the various system components and validate the error correction ability in order to evaluate its reliability, latency and energy consumption. We also compare our results with those of more conventional approaches.

The last chapter concludes the thesis and outlines future research based on the work described in this thesis.

The thesis has two appendices. Appendix A includes an overview of the Xilinx 7-Series device configuration architecture, as discovered through our experiments, and describes the correlation between hardware resources and configuration memory frames, while Appendix B presents an overview of the configuration port facilities of 7-Series devices in order to explain how configuration memory frames are accessed using the internal registers to achieve high performance.

Chapter 2

Background

This chapter summarizes the research context including a summary of the benefits when Commercial Off-The-Shelf (COTS) Field Programmable Gate Arrays (FPGAs) are deployed in satellite applications, the radiation challenges in space, and the reasons why COTS FPGAs can be considered to be a feasible alternative to radiation-hardened FPGAs.

2.1 Why FPGAs in Space

FPGAs are Very-Large-Scale Integrated (VLSI) devices that are designed to be reprogrammed for user-specified functions after their manufacture. Typical user circuits range from merely simple logic gates, wires or buses to complex microprocessors and Systems-on-Chip (SoCs). A potentially important application area is implementing satellite payload applications. Compared with its alternative technology, Application-Specific Integrated Circuit (ASIC) devices, FPGAs have several advantages.

- **Low Non-Recurring Engineering (NRE) costs**. The hardware platforms for space missions are often built for specific purposes that have never been developed before, and most of which need to be heavily customized. The small market size, stringent development cycle and energy budget create significant NRE costs that cannot be offset [17]. FPGAs provide ideal hardware platforms that can be programmed to adapt their internal hardware resources to the type of logic circuit required. Compared with FPGAs, ASICs that also have prefabricated logic resources, have to be re-fabricated for each application, which results in a long prototyping cycle and high development risk [18].

- **Suitable for onboard processing applications**. Remote sensing payloads require increasingly high data rates to reduce the amount of raw data that is to be sent for ground-based processing, but the radio link is low bandwidth and often unavailable [17, 19]. Under these conditions, FPGAs are widely used for low-level signal processing and compression applications [20].
- **Field-upgradeable**. FPGAs can be reprogrammed without having to be retrieved. The circuit implemented by FPGAs can be reconfigured remotely, and allows for in-orbit hardware upgrades. Ground-based communication stations can upload new configuration files for the FPGA on the satellite throughout its lifetime [21].
- **Dynamically reconfigurable**. State-of-the-art FPGAs allow dynamic hardware changes on part of its logic [22]. In dynamic hardware designs, the reconfigurable function blocks are only loaded when they are needed, which significantly reduces the power consumption and allows the hardware design space to be explored more completely [23].

2.2 Challenges for Deploying FPGAs in Space

Space FPGAs need to be able to deal with three aspects of harsh environment conditions. These devices must survive the high mechanical and acoustic vibration during the rocket launch, and withstand high temperature ranges due to the vacuum in space, in which industry or military temperature grade devices, using appropriate packaging technology, are able to work safely. Moreover, the high level of ionizing radiation present beyond the Earth's magnetic field is the most serious threat to these devices [17].

2.2.1 Radiation Effects

Just like other VLSI devices, FPGAs in space need to deal with radiation-induced effects, including Total Ionizing Dose (TID), Single Event Latchup (SEL), Single Event Upset (SEU) and Single Event Transient (SET), which are mainly induced by ionizing radiation caused by heavy ions, high-energy electrons and protons etc. striking the PN junctions of CMOS devices. Making matters worse is that for SRAM-based devices, SEUs may also cause configuration memory errors, which can change the circuit state and/or behavior [24]. The detailed description of these radiation effects is as follows.

- TID can create parasitic leakage paths as more electric charges are trapped in the gate insulation layers of CMOS transistors. TID biases the gate and changes the threshold voltage Vth, which gradually defeats the transistors [25]. TID tolerance of a CMOS device is a factor that indicates its lifetime in space, which is also determined by the radiation harshness and manufacturing technique of the device.
- SEL is a potentially destructive effect that has long been a bane to CMOS circuits [26]. Latchup is a sequence of events occurring very quickly that change CMOS circuit behavior to become equivalent to that of a thyristor, causing failures of circuit function, which may furthermore destroy the device [27]. Among all cosmic rays, heavy ions are the main cause of latchup events, while high energy protons get less chances to produce latchup in silicon devices [28]. However, in modern CMOS processes, the susceptibility to SELs is reduced due to a reduction in Vcc [25].
- SEUs change the state of dynamic cells on electronic devices, which are always called soft-errors. In FPGAs, upsets can modify the data being processed by the circuit, as well as the configuration memory that determines the implementation of the circuit. The consequence of this effect is that it permanently corrupts the circuit function if the error persists in feedback cycles or in the configuration memory [2].
- SET is an upset due to a transient pulse induced in the wiring or combinational logic between storage cells arriving at the input of a latch on the latching edge of the clock, and the faster the clock speed, the greater the probability for latches to latch a transient signal [29].

2.2.2 Radiation-hardening

In order to prolong the working life and to avoid errors and interrupts on the processing circuit, space-borne FPGAs are hardened in four aspects — architecture, manufacturing process, logic cell design, and redundancy scheme. Modern FPGAs are mainly based on SRAM, but anti-fuse FPGAs and flash-based FPGAs are more robust and common in space applications. Anti-fuse FPGAs do not require configuration memories and therefore SEUs will not corrupt the state of the circuit. Flash-based FPGAs, due to their higher reprogram voltage, are far less sensitive to configuration memory SEUs [30]. However, all three types of FPGAs are prone to the other effects and sacrifice adaptivity and dynamic reprogram ability. Hardening approaches applied to the manufacturing process, such as the thin epitaxial wafer process, or Silicon-On-Insulator (SOI) technique [31, 32], can greatly reduce or completely preclude leakage current, parasitic capacitors and transistors, and trapped electron-induced degradations, and all of these cause the two destructive effects, TID and SEL [26, 33]. To overcome errors caused by SETs and SEUs, storage cells and configuration memory in space-borne SRAM-based FPGAs could be fabricated based on the Dual Interlocked storage Cell (DICE) design [34], which greatly improves the SEU resistance by nearly 1,000 times [32] while only doubling area, energy consumption and gate delay compared with standard SRAM cells [35]. Flip-flops could be preceded with SET filters, which can significantly reduce proton-induced short transients, but they may impose extra delay [36]. However, all of the above-mentioned hardening approaches are costly and cannot completely immunize the device against upsets, and as the transistor feature size shrinks, state-of-the-art FPGAs are becoming more vulnerable to soft errors [37]. Applying redundancy schemes such as Triple Modular Redundancy (TMR) is more important and effective than hardening logic cell designs [1]. As only latches are sensitive to upsets in anti-fuse and flash-based FPGAs, a unique flip-flop hardening approach using voter gate logic that is described in [38] has been applied to mitigate upsets in user flip-flops. With the majority of voting circuitry provided by the Triple Modular Redundancy (TMR) technique, a triplicated latch block can mask and detect errors occurring in one of the latches, while the error can also be corrected by asynchronous voting logic [38]. For SRAM-based FPGAs, as both latches and configuration memory cells are sensitive to upsets, combinational logic also needs to be triplicated and aided with configuration memory scrubbing approaches, which periodically sweep away errors that have accumulated in the device configuration memory [39].

Deploying Commercial, Off-The-Shelf (COTS) SRAM-based FPGAs has became more attractive than using radiation-hardened devices, as numerous radiation tests on Xilinx Kintex-7 XC7K325T FPGA [28, 40, 41] have proven this to be feasible. Table 2.1 provides a comparison of typical anti-fuse, flash-based, radiation-hardened SRAM-based and commercial-grade FPGA models in terms of their manufacturing process, logic cell features and density, performance, available soft-error mitigation approaches, and radiation-tolerance. Commercial FPGAs cannot provide high radiation immunity, but 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. As reported in [28], no change in post-irradiation supply current was observed after the FPGA received an equivalent total dose of 17 krad(Si), which guarantees enough device lifetime in space for short duration and moderate radiation level missions, while the authors of [41] predict the latchup event rate to be 9.2×10^{-5} per device day in Geostationary Earth Orbit (GEO),

Table 2.1: Overview of FPGA Parameters & Radiation Immunity

Technique	Anti-fuse	Flash	Rad-hard SRAM	Standard SRAM
Device	RTAX4000D	RT4G150	XQR5VFX130	XC7K325T
Family	RTAX	RTG4	Virtex-5QV	Kintex-7
Vendor	Microsemi	Microsemi	Xilinx	Xilinx
Process	0.15 µm	65 nm	65 nm	28 nm
Logic Cell	C-Cell	LUT-4	LUT-6	LUT-6
Density	36K	151K	131K	326K
Speed	230 MHz	300 MHz	450 MHz	600 MHz
TID Immunity	300 krad	100 krad	1,000 krad	17 krad
Latchup Threshold	110 MeV·cm^2/mg	110 MeV·cm^2/mg	100 MeV·cm^2/mg	15 MeV·cm^2/mg
GEO Orbit Configuration Memory SEU Rate	N/A	Immune [30]	3.80×10^{-10} upsets/bit·day	1.52×10^{-8} upsets/bit·day
Embedded Mitigation Approach	TMR-FF	TMR-FF	XTMR + Scrubbing	Scrubbing

and those observed latchups are non-destructive, which can be mitigated by discharging the device through the auxiliary power-rail. More importantly, in LEO, there are scarcely any heavy ions that could cause SELs on these commercial devices [28].

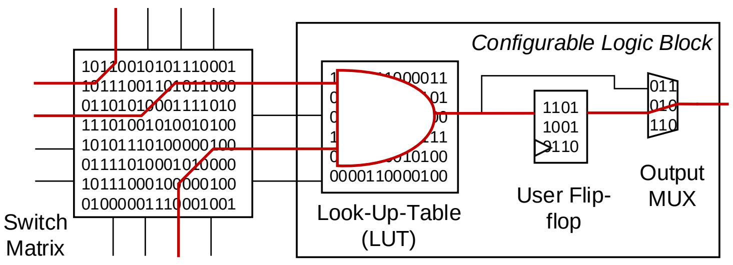
Commercial FPGAs are on the order of two magnitudes more sensitive to upsets than the other space-grade models. The SEU rate for a Kintex-7 device in the table is obtained from [28] using CREME96 [42] assuming a GEO orbit, solar minimum conditions, and 100 mils (2.54mm) of aluminum shielding, while that of the space-grade Virtex-5 device is from its datasheet [32] assuming a geosynchronous orbital altitude. As anti-fuse and flash architectures are not sensitive to configuration memory SEUs, their vendors only need to harden them by triplicating the flip-flops to protect user data. For the radiation-hardened SRAM-based FPGA model listed, Xilinx provides the XTMR tool and a configuration memory scrubber (IP block) [2] to boost the development of their space-grade FPGAs. However, the commercial-grade device model lacks ready-to-use upset mitigation approaches to deal with more intense soft-error rates. Since ground-based critical systems also have stringent reliability constraints [43], Xilinx has made the Soft Error Mitigation (SEM) controller [44] available for non-radiation hardened devices. However, commercial FPGAs are in urgent need of more flexible and comprehensive, reliable and portable SEU mitigation solutions. In general, commercial devices also need to rely on TMR techniques to mask and detect errors in user circuits, but approaches for correcting SEUs in configuration memory are the subject of ongoing research among FPGA research communities. These investigations focus on using configuration memory scrubbing to aid TMR (TMR-S) or using Module-based Error Recovery (TMR-MER). More detailed descriptions and literature review of these two approaches appear in Section 2.3.

COTS FPGAs can bring plenty of benefits to satellite systems, as they are state-of-the-art and always two or more generations ahead of radiation-hardened FPGAs. As listed in Table 2.1, the COTS device has the highest logic density, and each logic cell is configured using an adaptive 6-input look-up-table that can be further configured into two 5-input look-up-tables. By contrast, the anti-fuse model is based on special C-Cells to perform combinational logic, which only have 4,000 possible functions, and such devices only integrate 36,000 cells, which is only 10% of the logic density of the SRAM device due to the outdated silicon process used. The maximum clock speed of commercial programmable logic cells is 2.5 times greater than the slowest technology (anti-fuse). Moreover, commercial FPGAs range in logic capacity and function, which provides flexibility for development, while there are only limited choices for space-qualified FPGAs, and they are not available in all markets due to ITAR (International Traffic in Arms Regulations) restrictions.

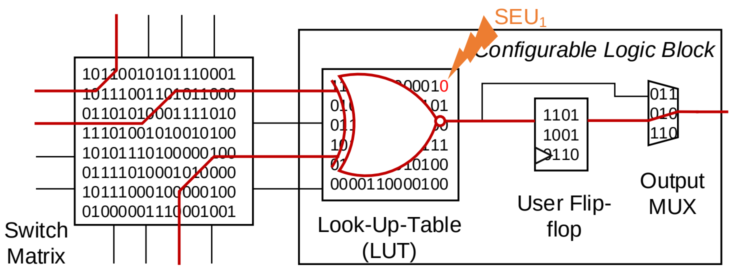
2.2.3 Upsets in SRAM-based FPGA

Upsets can affect user data stored in both user flip-flops and block memories of SRAM-based FPGAs. The incorrect data may directly cause a one-hot state machine to enter an unsafe state, or cause wrong state branches in the state machine logic. For system buses such as AXI [48], Wishbone [49], etc. that are the main interconnection method for FPGA-based SoCs, SEUs in user latches may transfer incorrect data or fail to handshake correctly, while for soft processors, SEUs may affect their pipeline registers, program pointers and register file, and both instruction and data memory and caches are unsafe.

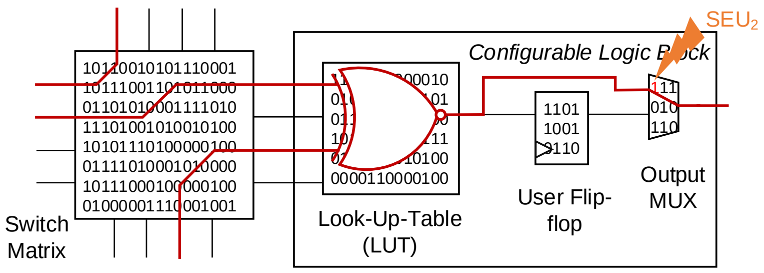
SEUs are more probably present in the device configuration memory as more than 98% of the memory latches in commercial FPGA comprise the configuration memory [46]. Figure 2.1 illustrates four typical circumstances in which a circuit fails due to SEUs, where Figure 2.1(a) depicts an AND gate logic with a D flip-flop before exiting the CLB. The numbers inside the logic cells represent their configuration data. In Figure 2.1(b), the upset directly changes the LUT content, which changes the implemented logic to an XNOR gate. In Figure 2.1(c), the upset disables the output latch, which changes the circuit from synchronous to asynchronous. SEUs in the Switch Matrix (SM) may cause two possible effects. As depicted in 2.1(d) & (e), a configuration memory upset may



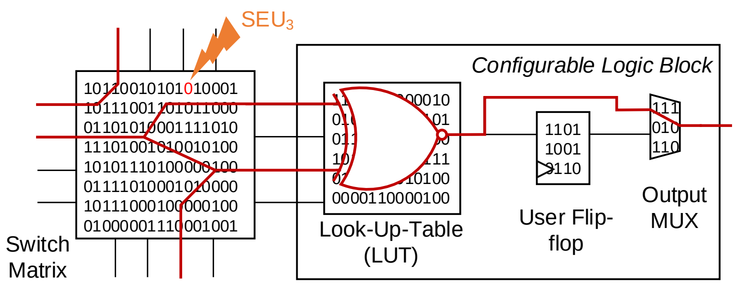
(a) correct circuit state



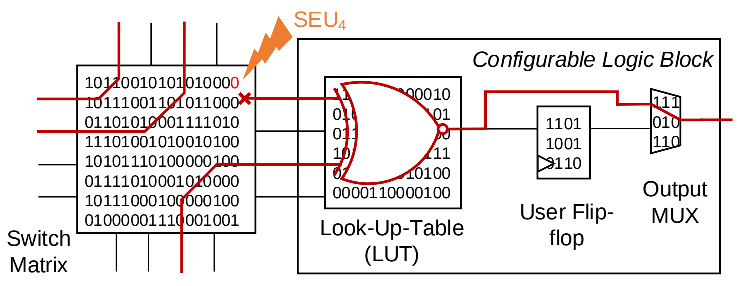
(b) after an SEU affects the LUT changing the logic function



(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 2.1: An example of SEU effects on different types of logic cells [45–47]

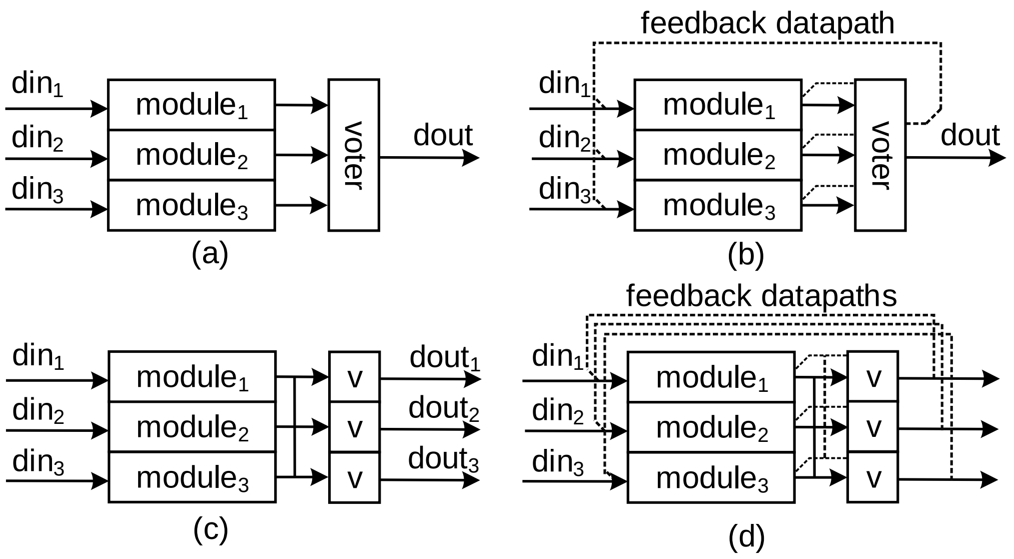


Figure 2.2: Triple Modular Redundancy

bridge two routes together that affects two routing nets, or may disconnect an input to the CLB [46]. Moreover, for a circuit implementation, only a few configuration memory bits are associated with the circuitry of the design - these are referred to as essential bits. SEUs present in non-essential bits will not cause any of the above-mentioned failure modes [2].

2.3 Mitigating Upsets on SRAM-based FPGAs

To date, two upset mitigation technique have emerged — Triple Modular Redundancy with configuration memory scrubbing (TMR-S) and Triple Modular Redundancy with Module-based Error Recovery (TMR-MER) both of which are proven to be able to mitigate errors in COTS SRAM-based FPGAs. Both TMR-S and TMR-MER rely on the TMR technique to provide protection for the user data, and Dynamic Partial Reconfiguration (DPR) to correct configuration memory upsets.

2.3.1 Triple Modular Redundancy

TMR implements triplicated critical system units that perform the same operation at the same time. The results of all three units are then processed by a majority voter to generate a single output that agrees with at least two outputs of the triplicated system as shown in

Figure 2.2(a). As demonstrated in [50], if the module contains logic cycles (e.g. counters or accumulators), errors may persist in the feedback path of the cycle, which causes one of the units to become unsynchronized if it suffers datapath errors. Therefore, as depicted in Figure 2.2(b), to avoid persistent errors, the majority voter also processes these feedback paths [1]. As the majority voting circuitry is also a single point of failure, voters are always triplicated for high reliability as demonstrated in Figure 2.2(c) & (d). However, TMR itself does not provide any means of correcting a faulty unit. For this reason, TMR is always implemented with scrubbing or Module-based Error Recovery (MER) to eliminate these configuration memory errors.

2.3.2 Dynamic Partial Reconfiguration

Both scrubbing and module-based configuration error recovery are based on Dynamic Partial Reconfiguration (DPR)[1], which allows reading and writing configuration memory while the device operates. DPR controllers need to access the configuration data through configuration memory ports such as the Internal Configuration Access Port (ICAP), SelectMap, or using JTAG [51]. 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 a series of frames with a necessary bitstream header and footer, and optional frame headers that control the configuration port registers such as the frame address register, configuration command register, etc. [51]. Legal bitstream formats of Xilinx 7-Series devices are discussed in Appendix B.

2.3.3 Configuration Memory Scrubbing

Configuration memory scrubbing periodically scans the entire device and corrects configuration memory errors by rewriting the memory frames containing them. Scrubbing can be implemented by simply overwriting all of the configuration frames, or reading and comparing these frames with a golden copy, to replace any frame that is found to be in error. The scrubbing speed of this approach is restricted to the maximum throughput

[1]It should be noted that the DPR capability has until recently only been explained in advanced FPGAs manufactured by Xilinx. In this thesis, we have therefore targeted the development of radiation mitigation techniques in Xilinx devices.

of the memory device, interface and memory controller implemented on the FPGA. To avoid the need for data retrieval, single frame errors can be detected and corrected using Error Correcting Codes (ECC) embedded in the frame. This approach can be further implemented with a device-level Cyclic Redundancy Check (CRC) to determine whether or not a complete reconfiguration of the device is required [44].

The Soft Error Mitigation (SEM) controller provided by Xilinx [44] detects soft errors in configuration memory by reading configuration frames and calculating Error Correcting Codes (ECC) based on the built-in FrameECC primitive [52]. A typical period of a scrub cycle on 7-Series devices is 18.3 ms for an Artix-7 XC7A200T device with maximum throughput of ICAP, which is about 1.01 us per frame as the device has 101 words per frame and the ICAP is 32-bit wide running at 100MHz. This figure may increase as the transistor density of devices increases. To correct errors in a frame, the SEM controller can fetch frames from off-chip memory (0.83 ms per frame) or flip the incorrect bit as instructed by the ECC (0.61 ms per frame). While the later can only correct single errors, it can be further enhanced to be able to correct two adjacent errors in a frame via a hybrid algorithm based on CRC and ECC (18.79 ms per frame).

Since scrubbing has to run in the background periodically, a substantial amount of energy is wasted when errors are not present [53]. To reduce the energy consumed by periodic scrubbing, a scrub cycle can be triggered via a dedicated error transmission network (referred to as the Reconfiguration Control Network (RCN) in TMR-MER). Moreover, the overall mean-time-to-repair (MTTR) of scrubbing can be further reduced by prioritizing frames scanned for different error signatures generated by the network [54]. Nazar *et al.* [54] proposed a heuristic signature translation algorithm to reduce the memory overhead of storing prioritized frame addresses for different error signature.

2.3.4 Module-based Error Recovery

Module-based configuration memory Error Recovery (MER) reconfigures the frames of a TMR module when an error in its configuration memory is detected. While scrubbing occurs periodically, MER relies upon the error detection circuitry implemented together with majority voters to trigger a reconfiguration of the module presenting the error [3]. With an increasing integration of TMR modules, TMR-MER necessitates a network to aggregate these error requests which we refer to as a Reconfiguration Control Network (RCN) [3].

The biggest advantage of TMR-MER over TMR-S is that rather than having to overwrite the whole configuration memory, TMR-MER only needs to reconfigure a portion of the configuration memory frames when errors are detected by individual voters and when faults occur repeatedly within one module. The reconfiguration request needs to be relayed through the RCN. The performance of the RCN determines the latency in transmitting error reports from voters to the RC. As the investigation in [12] indicates, this is typically possible within a few microseconds at most, while the detection and correction time for TMR-S corresponds to half the time needed to reconfigure the entire device - on the order of 25 ms depending on the device capacity [44]. While in TMR-MER the configuration memory contents of a complete module are rewritten, energy is therefore wasted on rewriting frames that are correct, the overall number of frames that are rewritten is typically far lower than the number that are checked in a scrub cycle. Therefore, the time and the energy consumption to detect and correct an error are largely reduced in comparison to TMR-S. Moreover, multiple configuration memory errors [37] affecting the one module can be corrected in a single action and correction is typically completed more promptly.

TMR-MER must incorporate circuit re-synchronization approaches when the reconfiguration of the faulty module is finished, which makes it more responsive to recovering from errors than TMR-S. Table 2.2 summarizes the features of existing synchronization approaches [4, 5, 8, 50]. Johnson *et al.* [50] described a synchronizing voter architecture and insertion algorithms which always synchronizes triplicated modules in the background. When the error is recovered, the new data passing through the circuit can clear errors in the flip-flops. This approach is suitable for processing circuitry where the feedback edges in the circuit are easily identified. The second approach is based on check-points and state prediction logic as proposed in [8], which is more applicable to state-machines. After the faulty module is corrected, the state-machine enters a checkpoint as instructed by the state prediction logic, and when the remaining two modules enter the same checkpoint, the state prediction forces a branch of the state-machine and thus the module is resynchronized. As described in [4], the third re-synchronization method targets packet processing units, where the modules are equipped with packet encoder and decoder logic. Before a packet is received, the units are synchronized by a local reset signal at the end of the preceding packet [4]. The maximum synchronization latency is the time for the units to process the longest packet, and in the meantime, error signals are ignored. For soft-processors, as the datapaths of the internal pipelines are complex, the ideal re-synchronization method is state migration as presented by Tanoue *et al.* [5]. When the reconfiguration of an incorrect soft-processor is finished, a processor interrupt is needed to let all three processors enter the synchronization program, in which the correct register file data and processor special-purpose registers are migrated to the recently-recovered processor via shared

Table 2.2: Overview of Re-synchronization Approaches

Methods	Flush Through [50]	Check-points & State Prediction [8]	Periodic Flow Control [4]	State Migration [5]
Adaptivity	Processing circuitry	State machines	Packet processing cores	Soft-processors
Latency	fast	medium	medium	slow
Area	low	low	high	medium
Design complexity	medium	hard	medium	low
Trigger	Passive	Active	Passive	Active

memory [5]. However, the check-points & state prediction and state migration approaches require a trigger after the reconfiguration, which calls for something like the RCN used by the TMR-MER system to be able to toggle the synchronization logic.

Until now, as it has been described in the literature, the TMR-MER technique is not as robust as TMR-S. Modern FPGAs, such as those offered by Xilinx, include built-in resources such as the FrameECC primitive and provide soft IP, such as the SEM controller, to aid scrubbing [44]. But the use of a TMR-MER approach is not supported nearly as well, and design complexity and the likelihood of introducing design errors are noticeably increased. For example, a designer who wishes to employ TMR-MER needs to design a suitable RC, a proper RCN, storage for partial bitstreams and a data synchronization mechanism. Many TMR-MER SoCs in the literature have made efforts to solve some of the above technical problems, such as the rapid error detection, recovery and synchronization TMR-MER framework presented in [3]; the automatic TMR-MER voter insertion and floorplanning framework in [6]; the generic reconfiguration controller in [4]; a comprehensive study of different RCN topologies [12]; the above listed synchronization techniques, and so on. However, these academic techniques cannot readily be combined since they have been developed on different platforms and lack portability.

The biggest drawback of existing TMR-MER approaches is that most of them can only detect and correct errors inside the TMR modules, while errors inside the majority voters, affecting the interconnecting nets between voters and modules, and the RCNs are almost always neglected. Some partitioning approaches may partially solve these problems as demonstrated in [4, 6], but the system has to sacrifice reconfiguration time. Furthermore, the TMR-MER techniques rely upon the vendor's partial reconfiguration flow [22] to generate partial bitstreams, but this flow is not able to generate partial bitstreams for non-block-oriented designs. It is therefore not possible for the flow to generate partial bitstreams for the nets between TMR modules.

TMR-MER Model

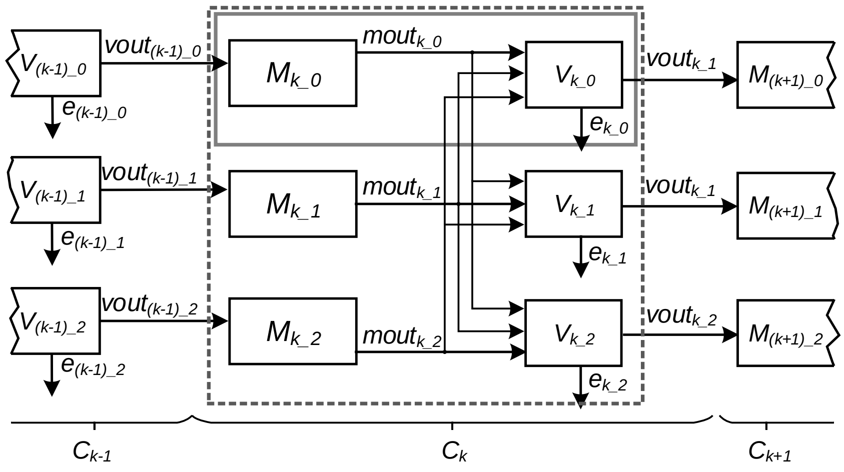
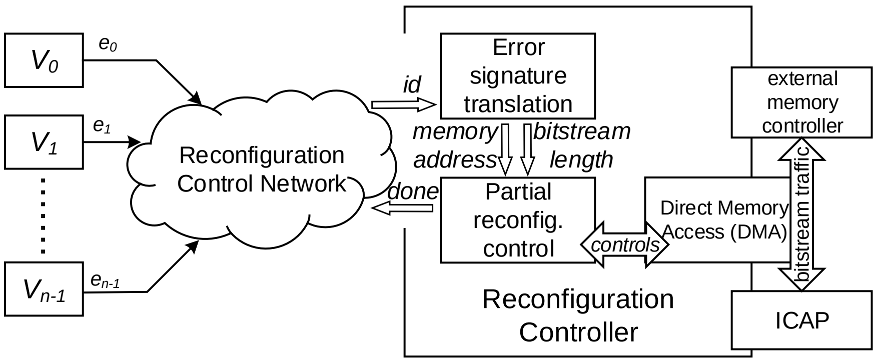


Figure 2.3: TMR-MER Model

In general, a triplicated system model is as depicted in Figure 2.3, in which we assume the user circuit comprises n TMR components that are acyclic, as discussed in [50], and that cyclic components are implemented by allowing voter outputs to re-enter a component as module inputs. Each set of triplicated modules, voters and interconnecting nets (between circuit modules and voters, between voters and downstream components, and between voters and the reconfiguration controller) comprise a single component (C_k, $k \in \{0, 1, ...n-1\}$). While Figure 2.3 illustrates a linear sequence of TMR components, in general, there may be several immediate predecessors and immediate successors of C_k. There are at least three possible approaches to partitioning the component into reconfigurable modules that can be independently recovered when they are found to be in error. Most commonly in the literature, only SEUs in the circuit modules themselves (indicated as $M_{k\_0,1,2}$ in Figure 2.3) are localized and corrected. This approach leaves the voters and interconnecting nets to be recovered by other means — perhaps via scrubbing, as proposed in [12], or via fine-grained modular reconfiguration, as studied in Chapter 5. Another approach includes the corresponding voter and some of the interconnecting nets, as illustrated by the grey box surrounding M_{k_0} and V_{k_0}. This approach allows both sub-components and some of the interconnecting nets to be recovered by modular reconfiguration, but does not provide a means for recovering all of the nets comprising $mout_{k\_0,1,2}$. A third alternative, therefore, includes all sub-components of a triplicated component in the reconfigurable partition, as



V0
e0
V1
e1
Vn-1
en-1
Reconfiguration Control Network
id
done
Error signature translation
memory address
bitstream length
Partial reconfig. control
controls
Direct Memory Access (DMA)
external memory controller
bitstream traffic
ICAP
Reconfiguration Controller

Figure 2.4: Reconfiguration Control Network (RCN) & Reconfiguration Controller (RC) for TMR-MER Systems

illustrated by the dashed box surrounding C_k. This approach can be used to cover all the nets except for the voter outputs, which must cross partition boundaries, but suffers a significantly larger reconfiguration delay, which translates to increased Mean Time To Repair (MTTR).

While the voter blocks in TMR-S only fulfill the role of providing a majority output that agrees with at least two outputs generated by the triplicated modules, those in TMR-MER are enhanced to identify the module whose output differs from the majority based on [55]. The voter logic discussed in this dissertation not only protects the output of the user circuit, but also detects which module output ($mout_{k\_j}$) is incorrect in the minority and raises a 2-bit error report (e_{k_i}) identifying that module, where $i, j \in \{0, 1, 2\}$. Values 00, 01 and 10 identify the erroneous module ($M_{k\_0,1,2}$), while the value 11 indicates the absence of an error in C_k.

Both TMR-S and TMR-MER require a Reconfiguration Controller (RC), as illustrated in Figure 2.4, to perform dynamic partial reconfiguration, but in TMR-MER, the RC should also be able to translate the module-id aggregated by a dedicated Reconfiguration Controller Network (RCN) into the memory address and bitstream length of the corresponding partial bitstream file stored in the external memory. To achieve high throughput and flexibility, a generic RC uses Direct Memory Access (DMA) to transfer bitstream data from an external memory controller to the ICAP via commercial system buses such as AXI. When the reconfiguration is done, the *done* signal is toggled in order to acknowledge the RCN. As the circuit is acyclic, the flush through of new data synchronizes the modules.

2.4 Discussion

In this chapter, we have summarized the advantages of the deployment of COTS FPGAs in space and explained the challenges caused by ionizing radiation which are most crucial. We then provided an overview of existing SEU mitigation approaches — TMR-MER and TMR-S — with the aim of explaining the benefits of TMR-MER over the conventional TMR-S. These are that TMR-MER is more time- and energy-efficient and responsive for correcting errors in the user circuit than TMR-S. Lastly, we have outlined that the design of the RCN also determines the performance and robustness of TMR-MER technique.

Chapter 3

Token Ring Network

COTS FPGAs can provide more hardware resources than radiation-hardened devices to allow integrating more functional units for space-borne applications. For TMR-MER systems on these devices, in order to deal with the a potentially large number of error signals generated by the TMR voters used to protect these functional units, a multirate token ring network design that was first proposed in [3] and further described in [13] is introduced to aggregate these signals to a central Reconfiguration Controller (RC), which coordinates the partial reconfiguration of TMR modules suspected of being affected by configuration memory errors.

The work described in this chapter includes:

- a re-implementation of the token ring network according to the description detailed in [13], with the aim of evaluating its utilization and performance as the number of network connections scales up;
- a proposal for a simplified token ring network that fulfills the basic feature of the network in the literature, but greatly reduces the utilization, and to compare its performance and utilization with the initially proposed token ring network.

3.1 Token Ring Network Review

In order to make the token ring network generic, flexible and extensible, Cetin *et al.* [13] designed a multirate token ring network that has various features. To ensure high scalability and portability, the architecture of the token ring network connects voters with the RC in a daisy-chain manner. Network interfaces are attached to each of the voters in the system, and the links between network interfaces are asynchronous. To reduce the use of routing resources, the interface only use 3 signals to communication with its closest upstream and downstream interfaces, which includes two signals for performing handshakes to avoid meta-stability [56]. Flits are transmitted bit-by-bit with a long handshake delay, which calls for SerDes circuitry and handshake state machines. When an error is detected by a voter in a TMR module, the network interface of the voter encodes the voter error status into a reconfiguration request (RReq) message which is transferred to the RC via downstream interfaces. These interfaces relay this message until the RC receives this information. Then, the module-id is decoded by the RC, which directs the partial reconfiguration of the indicated module. At the end of the reconfiguration, the network instructs the module to synchronize state with its siblings by passing a reconfiguration done (Rdone) message generated by the RC. RReq messages and the other messages are transferred via flits, and to protect the message that might be corrupted by upsets, flits transferred by the network are encoded with a (7,4) Hamming code, which supports correcting a single bit error [13] in a 7-bit flit. The flits have a constant length, but for RReq messages, the network interface transmits multiple flits as it includes the module-id of the incorrect module, and to further maximize the performance of the network, the network supports wormhole routing (pipelining the flit transfers). However, as reported in the following section, the token ring network has high utilization and low performance. While flit transfers are protected from transient effects on the links through the use of Hamming codes, upsets occur more frequently in the configuration memory and state machine flip-flops. As a consequence, the complexity of the interface is greatly increased, which increases their area, degrades the performance, introduces high recovery delay, and increases the sensitivity of the network to upsets as higher resource utilization results in a greater number of sensitive configuration bits.

3.2 Revised Token Ring Network Design

The original token ring network interface design [13] comprise three components — network interfaces, communication ports and transceivers, as shown in Figure 3.1. Each of these designs are under the supervision of state machines. The transceivers send or receive bits using 3-stage synchronizers [57] to synchronize the link, while the communication ports serialize or deserialize the flits. The network interfaces contain a complex FSM that controls the protocol of the network, and also includes logic for performing error

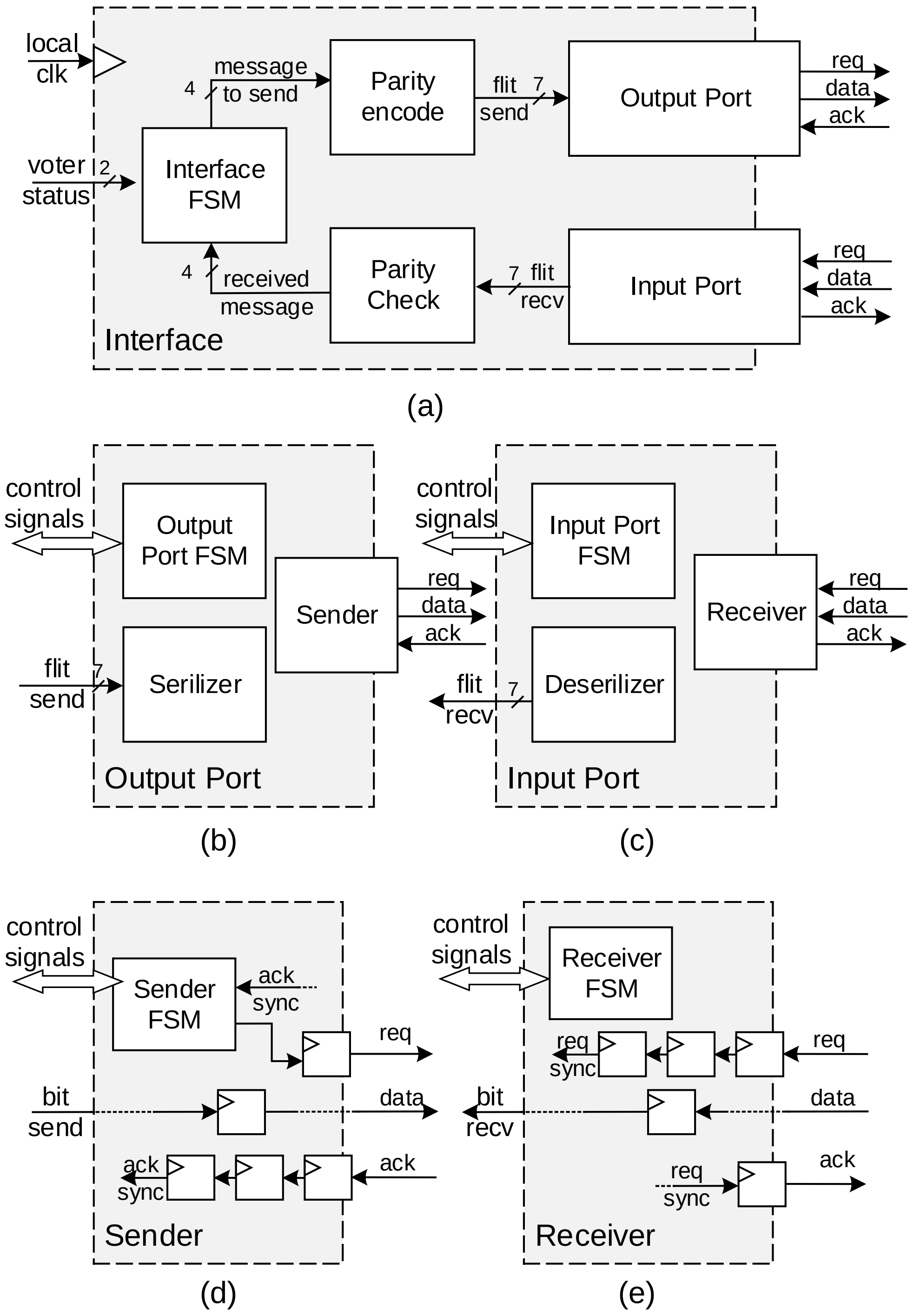


Figure 3.1: Overview of token ring network components: interface (a), communication ports (b) (c), and transceivers (d) (e) [13]

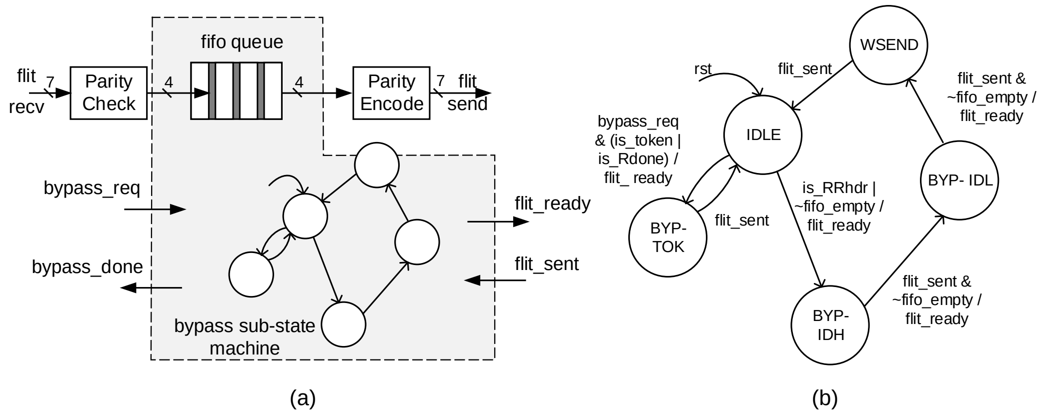


Figure 3.2: Bypass sub-state machine design: (a) block overview; (b) sub-state machine design

correction based on the Hamming code. However, the details of how wormhole routing is implemented on the network are not explained.

3.2.1 The Implementation

Our implementation follows the same design structure, as shown in Figure 3.1, FSM and protocol definitions, as described in [13], but omits the FSM design in the communication ports. The state machines are based on one-hot encodings, which generally yield higher performance than those sequential state machines based on binary or gray codes. This approach may result in slightly increased flip-flop usage, and reduced look-up-table usage. In order to allow the network interfaces to run at clock speeds above 400 MHz, the inter-component signals are latched with flip-flops, which significantly improves the clock speed, but introduces latency.

The RReq message comprises three flits that includes an RReq header (RRhdr), and two flits that contain the higher 4 bits and lower 4 bits of the module-id, respectively. Thus, the size of the module-id is set to 8 bits, where the lower two bits represent the voter status and the upper six bits indicate the interface-id. Therefore, the network is capable of providing reconfiguration management for up to 64 voters.

In order to let our design support wormhole routing, a FIFO queue and a sub-state machine are attached to the interface hierarchy as shown in Figure 3.2. As the upstream and downstream network interfaces may run at different clock frequencies, which brings about a complex synchronization problem when transferring error messages that consist of multiple flits such as RReq, the FIFO queue temporarily stores the received flits when the downstream interface works slower than the upstream one. When a token is decoded by the main state machine and there is no reconfiguration request from the attached voter, the received token is then pushed into the FIFO queue and the sub-state machine enters the BYP-TOK state to pop out the token data to the output port. On the other hand, when the header of a reconfiguration request is detected, the sub-state machine relays this flit and waits for the following two flits.

The original state machine design can detect communication errors and raises a communication alarm, typically when the handshake times out, or when the protocol fails. However, the network itself does not generate alarm flits to inform the system supervisor (e.g. the RC). The user logic may need to provide support to aggregate these alarms. In our implementation, the communication error detection logic is removed as these failures can easily be detected by initiating a watch-dog timer on the RC interface and allowing each pass through of the token to reset the timer. When the RC is reconfiguring an incorrect module, the timer is disabled. The timeout value should be higher than the time between the first node receiving a token from the RC and the time to transmit an RReq from this node to the RC.

3.2.2 Experimental Analysis

In order to evaluate utilization and performance of the redesigned token ring network, we performed three experiments based on a medium-sized FPGA — an Artix-7 XC7A200T — using Vivado 2015.4 software with the default implementation settings. In the first experiment, we implemented the RC interface and voter interface separately on the device to obtain the resource utilization and maximum clock speed. The second experiment mapped out the design of the system layout described in [13], and measured different aspects of its transmission delay using static timing simulation, which was further compared with the simulation results demonstrated in [13]. The layout as shown in Figure 3.3 includes a Finite Impulse Response (FIR) node running at 400 MHz and a Block Adaptive Quantizer (BAQ) node running at 333 MHz, while the RC interface is fixed at 100 MHz as the maximum clock speed of the ICAP of the device is only 100 MHz. The network interfaces are clocked by local clocks. Lastly, we obtained the transmission delays of networks comprising 8, 16, 32 and 64 nodes running at 100 MHz.

The result of the first experiment is shown in Table 3.1. The reimplemented network interfaces support user applications running at a very wide frequency range (100 – 400

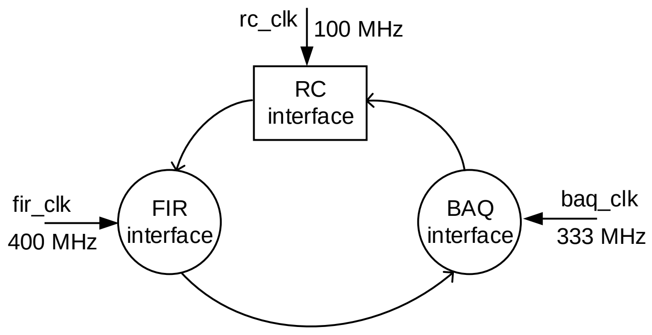


Figure 3.3: System layout [13]

Table 3.1: Utilization Result

	LUTs	Flip-Flops	Slices	Max Freq.
Original RC Interface [2]	121	102	42	345 MHz
Original Voter Interface [2]	131	97	48	338 MHz
Reimplemented RC Interface	119	112	47	464 MHz
Reimplemented Voter Interface	100	86	39	441 MHz

MHz). The overall utilization of these interfaces is not very significant, but if the system contains 64 nodes, the total area used by the token ring network would be 3,047 slices, which accounts for 23% of all of the available slices on the medium-sized device used.

In the second experiment, we simulated the token ring network on Vivado, measured the communication delays of the reimplemented design from the timing waveform, and compared it with the simulation results reported in [13]. In Table 3.2, the first section shows the transmission latency for a 7-bit flit (such as a Token flit) on different network links. As the redesigned architecture contains more latches on the internal signals to speed up the critical timing paths, its performance is a bit lower than that of the original network. The reconfiguration request initiation time is given by the average time taken by the token arrive at the voter to allow the voter to raise an error request. We also report the time to transmit a 3-flit reconfiguration request message from the FIR node to the RC. Since the network is wormhole-routed, this transmission delay is very close to the delay for the RRhdr flit to arrive at the RC node plus the delay of receiving the remaining flits over the BAQ-RC link.

Table 3.3 lists the results of the third experiment. The first section of the table reports the average Token flit arrival delay that is the time to traverse half of the ring, and the time

[2]Note that to obtain the utilization for the design of [13], the circuits were retargeted to an Artix-7 XC7A200T.

Table 3.2: Token Ring Network Performance Comparison

Parameter	**Original [13]**	**Redesigned**
FIR-BAQ flit transmission time	278 ns	252 ns
BAQ-RC flit transmission time	541 ns	620 ns
RC-FIR flit transmission time	591 ns	519 ns
Initiate reconfiguration request time	1,195 ns	1,430 ns
Transmit reconfiguration request msg.	1,901 ns	2,164 ns
Transmit reconfiguration done msg.	591 ns	519 ns
TOTAL FIR communications time	3.7 µs	4.1 µs

Table 3.3: Token Ring Network Performance with 8, 16, 32 and 64 Nodes

Parameter	**8 nodes**	**16 nodes**	**32 nodes**	**64 nodes**
Average Token arrival time	4.9 µs	9.3 µs	18.1 µs	35.7 µs
Worst-case RReq transmission time	11.1 µs	19.9 µs	37.5 µs	72.7 µs
Worst-case Rdone transmission time	1.0 µs	1.0 µs	1.0 µs	1.0 µs
Worst-case communications time	17.0 µs	30.2 µs	56.6 µs	109.4 µs
Best-case RReq transmission time	3.3 µs	3.3 µs	3.3 µs	3.3 µs
Best-case Rdone transmission time	8.7 µs	17.5 µs	35.1 µs	70.3 µs
Best-case communications time	16.9 µs	30.1 µs	56.5 µs	109.3 µs

needed to relay reconfiguration request flits initiated by the first node of the network (first node downstream from the RC), as well as the delay for transmitting the reconfiguration done message to that node. While in the best case, the last node (upstream of the RC) raises reconfiguration requests that will immediately be received by the interface of the reconfiguration controller without any downstream interfaces to be bypassed. However, by contrast, the time spent on transmitting the reconfiguration done message is then much longer since it has to be relayed by all of the upstream voters of the last node in the ring. The worst-case and best-case total communications time are equal but depend on the network size due to the symmetry of communicating around the ring, but these times are relatively long. The total communications time for 64 nodes is 109.3 µs, which is enough time to allow small modules that only span 2 major columns (about 72 frames according to Appendix A) of reconfigurable resources to be reconfigured using the maximum throughput of the ICAP that is 400 Mbyte/s.

The layout of the network interfaces may cause slight changes to these performance figures. The signal delay may increase as the distance between the two closest network interfaces increases, but these changes are on the order of a few nano-seconds, which only results in one or two more cycles delay before the synchronizers in the transceiver of the network interfaces latch the data.

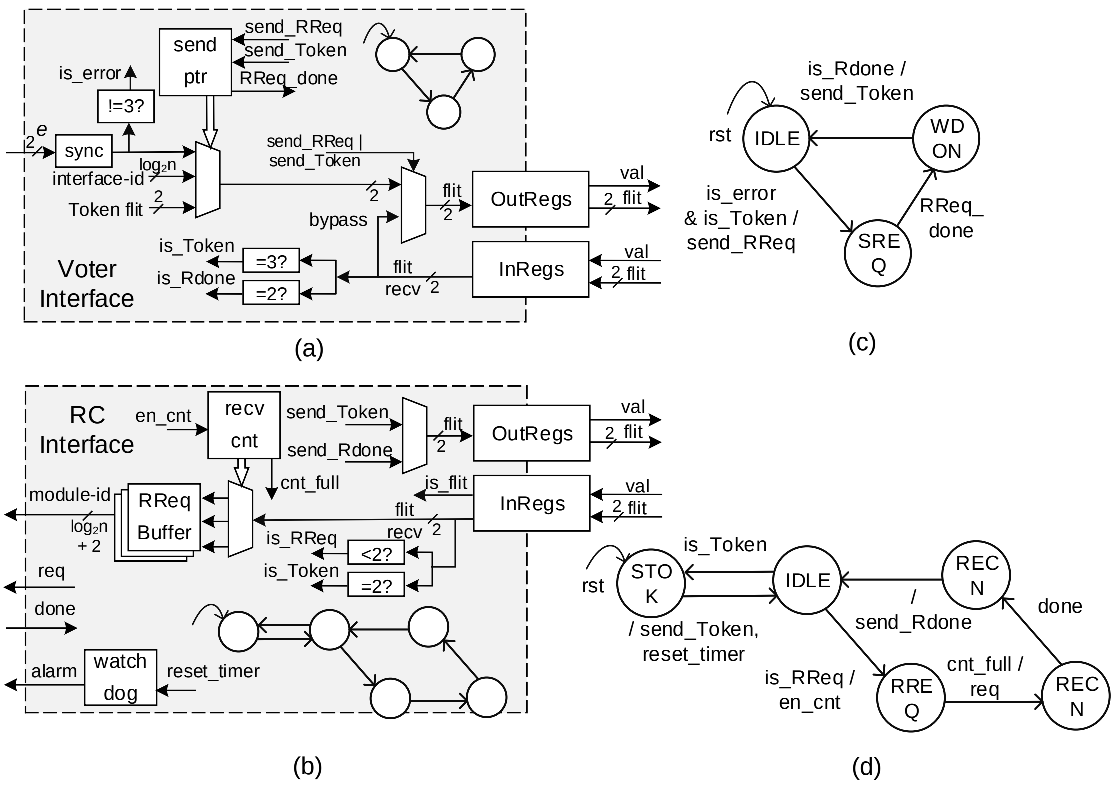


Figure 3.4: Architecture of network interface (a) for voters and (b) for the RC, as well as the details of their state machine designs (c) and (d).

3.3 Simplified Token Ring Network Design

As reported in Section 3.2, the token ring reconfiguration control network design from the literature [13] is flawed due to high utilization and low performance. This is mainly because its design concerns are excessively constrained. To pursue its extreme generic nature, the links between network interfaces are asynchronous, whereas, the network could be implemented synchronously to reduce the design complexity and greatly improve the latency, while the signals between network interfaces and voters can be left to be asynchronous. To reduce the use of routing nets, the network transmits flits bit-by-bit, which requires SerDes circuitry, and multi-flit messages are wormhole-routed, which calls for complex state machine design as demonstrated in Section 3.2. The flit format can also be simplified since the communication functions the network performs are relatively simple. Therefore, in this section, a synchronous token ring network design that fulfills the needs of a reconfiguration control network is presented that uses fewer resources and achieves high performance.

Table 3.4: Utilization Result

	LUTs	Flip-Flops	Slices
Voter Interface	16	15	5
RC Interface	24	23	17

3.3.1 Network Interfaces Architecture

The design of the network interfaces for the reconfiguration controller and for the voters are illustrated in Figure 3.4. The flits are only two-bit lines, and when the *val* toggles, the data on the two-bit flit signal is assumed to be valid. The Token flit is defined with a binary value of "11", while "10" indicates an Rdone flit. The RReq flits that have a msb of "0" are transmitted in a wormhole-routed manner. Each flit of an RReq contains one bit of the module-id of the suspect module. The voter generates a two-bit error signal (depicted as e) which is synchronized before it is processed by the interface's main state machine. If e indicates a decimal value of 3 then there is no error in the voter. Otherwise, if a Token flit is received and e is lower than 3, the state machine enters the SREQ state in which module-id bits will be sent bit-by-bit as a send pointer value is incremented. The arrival of the Rdone flit from the RC interface forces the state machine to enter the idle state and to resume monitoring the voter error status. When there is no error, the input register is connected to the output register, and because of this, the flit transmission can be wormhole-routed. As the whole network is synchronous, the FIFO design and the bypass state machine design described in Section 3.2 are not required. The RC interface is a bit more complex than the voter interface. In order to detect communication breakdown caused by upsets in the network configuration memory, a watchdog timer is included, and whenever a token passes through the RC interface, the timer is reset.

3.3.2 Experimental Evaluation

The design of the proposed network interface was hand-coded in Verilog and targeted at an Artix-7 XC7A200T using Vivado 2015.4 software with the default settings. The code is parameterized for different network sizes, and the length of the module-id is set to $\log_2 n+2$ where n is the total number of interfaces required by the system. Table 3.4 lists the utilization result for the proposed network interfaces. Compared with the utilization results of the multirate token ring network design reported in Table 3.1, the area of the voter interface is reduced by about 90%, and if a TMR-MER system contains 64 voters, the total area required by the network is only 344 slices which is only 2.6% of the available

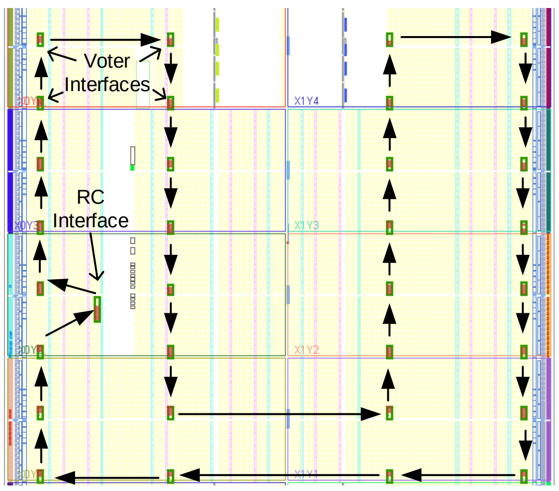
Table 3.5: Maximum Clock Speed and Routing Utilization Results

	PIPs	**SMs**	**Max Freq.**
Scattered layout	1,089	442	262 MHz
Compact layout	926	350	341 MHz

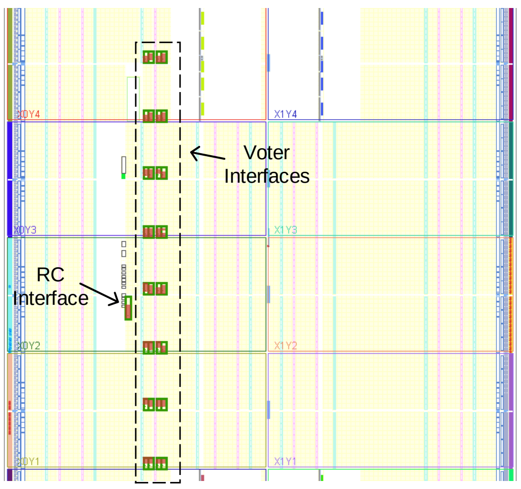
slices in the target device.

As the network is synchronous, the internal buffers of the network are clocked by a dedicated network clock signal. Thus, the token hop rate and the reconfiguration request transmission delay depend on the circuit's static timing and the maximum clock frequency, which is mainly determined by the floorplan of the network interfaces. The network has a low flit hop latency of 2 clock cycles. Therefore, the timing latency for the average Token flit arrival time can be calculated by $T_{token} = 2 \times \frac{n}{2} \times T_{clk}$, where n is the number of nodes. As the RReq flits are transferred in a wormhole-routed manner, the RReq transmission time for an arbitrary node in the network can be calculated by $T_{RReq} = (2 \times (log_2 n + 2) + 2 \times d) \times T_{clk}$, where d is the number of downstream nodes before the RC interface. The delay for transmitting an Rdone flit to an arbitrary voter interface is $T_{Rd} = 2 \times (n - d) \times T_{clk}$. To obtain the maximum clock frequency the synchronous network can achieve, we have implemented the proposed token ring network in two different layouts, namely, a scattered layout and a compact layout of 32 voters, as depicted in Figure 3.5. The interconnection direction of the network in the compact layout is the same as that of the network in the scattered layout. Table 3.5 lists the routing resource utilization of the links between network interfaces and the network clock speed for the two different layouts. In the checkerboard-like scattered layout, the maximum clock speed of the network is limited by the two longest critical links as shown towards the centre, at the bottom of the figure. The Programmable Interconnection Point (PIP) utilizations of the links in the two networks are relatively similar, while the utilization of Switch Matrices (SMs) in the compact layout is only 75% of that for the scattered layout.

The derived communication delay of the synchronous token ring network for 32 voters is shown in Table 3.6. The scattered layout is 30% slower than the compact layout. While compared with the performance result of the original token ring network as reported in Table 3.3, the performance of the simplified network is considerably higher, and the utilization is significantly lower.



(a)



(b)

Figure 3.5: Experiment 32-node layouts: (a) scattered layout and (b) compact layout

Table 3.6: Simplified Token Ring Network Performance Results for 32 Voters

Parameter	**In scattered layout**	**In compact layout**
Average Token arrival time	122.3 ns	93.8 ns
Worst-case RReq transmission time	171.9 ns	131.9 ns
Worst-case Rdone transmission time	7.7 ns	5.9 ns
Worst-case communications time	301.9 ns	231.6 ns

3.4 Discussion

In this chapter, we first evaluated the utilization and performance of the token ring network as described in the literature [3, 13]. The results of the experimental evaluation demonstrate that the token ring network is relatively heavy in terms of resource utilization, and through stress testing in which we implemented the network with 64 nodes, we found that the communication time of the network is relatively high with respect to the reconfiguration time. We therefore improved the token ring network to be fast and light-weight. The experimental results show that these improvements brought about a 200-fold speedup and also saved 90% of the hardware resources used. However, the token ring network could be further simplified by allowing more parallel signals in the links. The reconfiguration request transmission delay and utilization could thereby be further improved.

Chapter 4

Reconfiguration Control Network

TMR-MER systems require a Reconfiguration Control Network (RCN) which is an infrastructure component that relays error requests that may arise from any voter in the system to an internal or external RC. The performance and reliability of the RCN are of great importance for the following reasons. First, the latency of the RCN directly impacts the latency of recovering faulty TMR modules in the system. Secondly, the RCN is often implemented as a single non-redundant component and therefore introduces a single point of failure that may greatly compromise system reliability.

In this chapter, we provide a comprehensive study of the variety of RCNs mentioned in the literature. The RCN is typically implemented by utilizing configurable resources such as Configurable Logic Blocks (CLBs) and Programmable Interconnection Pins (PIPs) that reside in the Switch Matrices (SMs). It can also be implemented using the hardwired configuration network and the Dynamic Partial Reconfiguration (DPR) technique to communicate. 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 logic, which we refer to as a "hard" network.

Soft RCNs can be realized with simple point-to-point connections, referred to as star networks [4,6], or using shared, single access media such as a bus or token ring network [3]. In soft networks, routing congestion may occur as the number of TMR components in the system increases. As a result, the RCN may use more routing resources and suffer from lower performance. In contrast, hard networks rely on the dynamic reconfiguration of the FPGA to provide access to state of health of the TMR modules as reported by the voters. This method reduces the demand for routing resources, but increases latency as the number of voters increases.

We propose four different RCN topologies that meet the minimal requirements of the RCN, and then describe synthetic experiments in order to compare them with respect to reliability, latency, scalability and power consumption. The four networks are also mapped into a real satellite system — the RUSH payload [16] — for a similar assessment. We performed a fault injection experiment to further validate the reliability result we gathered from the synthetic experiments.

The work described in this chapter is part of the published paper [12] and the paper-under-review [14], where the contributions of this author were:

- To help survey all of the RCN designs described in the literature;
- To propose optimized RCN architectures for experimental evaluations;
- To implement the synthetic layout and RUSH layout derived from [16] and to obtain implementation details including design logic and routing utilization, latency, power estimation and essential bit reports from the implementation tool (Vivado 2014.4) for experimental evaluations.
- To propose the "RePin" approach to generate test vectors for fault injection experiments, and to assist in implementing the fault injection campaign.

4.1 RCN Survey

A star network interconnects voters with an RC via simple interfaces and simple wires. Each of the connections is dedicated and as the number of TMR modules increases, the RC will need a greater number of input ports to handle these wires, which compromises the performance of the network as the processing delay of these signals increases. It is possible to implement a central Network Controller (NC) to reduce the complexity of the RC and to maximize the performance. Straka *et al.* described an RC architecture which integrates dedicated NC logic to handle up to 200 error request signals aggregated via a star network [4]. Errors signals are multiplexed to an error register file which is accessed by an Error Encoder Unit in round-robin order. If an error request is detected, the unit translates the error into an address and the length of the corresponding partial bitstream file in off-chip memory. By accessing and loading this partial bitstream file, the RC performs partial reconfiguration of the faulty module. Each reconfigurable module has an error signal, whereby, to reduce the need for routes, the error signal is compressed into a binary format [6]. It is also feasible to implement an interrupt-driven NC, whereby voters interrupt a central arbiter to transfer an error request whenever it is raised.

Most of the buses presented in the literature are not specifically designed to be used as an RCN as described here. The TMR modules in those MER SoCs are more probably running as processor peripherals, where buses that are the main interconnection method may or may not fulfill the role of an RCN as well, and because of this the processor is an essential component of the system. Navas *et al.* [9] designed a fault-tolerant system based on an AXI bus, where the TMR modules are mapped into different bus addresses, and the processor has to access these processing peripherals via drivers with redundancy algorithms. The software-based TMR algorithm masks the faulty data if one of the triplicated modules is affected by SEUs, and automatically detects and corrects the module via partial reconfiguration. Jacobs *et al.* [7] proposed a fault-tolerant controller, which acts as an intermediate interface between a system bus and TMR peripherals. The controller votes on the output data of the TMR cores, which replaces the process of accessing the bus three times in order to write or read data from the triplicated modules, and thus, the processor only needs to access the bus once to complete the communication with TMR peripherals. This significantly increases the throughput of data processing. When SEUs manifest in peripherals, the controller detects which module is incorrect, then toggles a processor interrupt signal and writes the module-id and error info of the module in error into the register file, which is accessed when the fault recovery process is invoked by the interrupt handler. The processor then runs as an RC to reconfigure the faulty module according to the error info. To further improve the throughput of the bus transfer, the processor bus can be replaced by Fast Simplex Link (FSL) interconnections [58] which can be directly inserted into the pipeline to reduce the handshake delay imposed by the bus protocol [10]. However, buses in these TMR-MER SoCs are heavy, as they require more complex interconnection interfaces than a star, which results in an increased soft-error vulnerability and power consumption and latency. To adapt the bus topology into our TMR-MER framework, it is possible to implement simplified bus interconnections based on the shared bus topology. The common signals have to span the whole device, but this solution can be very scalable as scaling up the capacity of the bus may not require adding extra global signals.

A token ring network connects voters with the RC in a daisy-chain manner [3]. The voters and the RC require complex network interfaces with ports to connect with upstream and downstream voters because of which the ring network requires significantly more logic than the endpoints of the star network. Also, the ring network needs to pass a token for arbitrating the use of the network. The protocol increases the latency for transferring a reconfiguration request. However, the ring topology usually only interconnects neighboring components and therefore utilizes a reduced number of global wires for interconnecting all of the nodes in the network. In contrast, star and bus topologies realize mixed distance connections and thus utilize various interconnection resources, which involves both local and global wires. Usually, the order in which components are connected by the ring network has to be modified after floorplanning in order to minimize the routing utilization and improve the speed. For this reason, the ring network tends to use more local wires, and is considered to be more scalable than a star network.

While in a star network, each of the voters has a dedicated connection to the RC, both bus and ring topologies suffer more flaws on effects when a link suffers from a configuration memory error. In a bus, an SEU may corrupt all or part of the voters' connections depending on where the error occurs. However, in a ring, when a link suffers from a configuration memory error the ring can no longer function as intended.

The fourth approach that has been described in the literature makes use of the ICAP to monitor the voter status registers [59]. Advanced SRAM-based FPGAs support dynamic reconfiguration that allow reading and writing configuration memory via configuration ports. The flip-flop values can be found in configuration memory cells when a global flip-flop value capture command is issued [51]. Therefore, an ICAP-based communication scheme eliminates the need for a soft network, and reduces routing pressure, implementation time and improves reliability. This approach has the potential to be scalable as it does not require user resources to form the network and only utilizes a moderate amount of logic to implement the central controller.

4.2 Optimized RCNs for Comparison

In this section, we describe the RCN architectures we use for evaluation based on the four RCN types found in the literature and provide average latencies for obtaining a reconfiguration request.

Each RCN is uniformly composed of distributed Network Interfaces (NIs), a central Network Controller (NC) and an interconnection network between them as illustrated in Figure 4.1. In the figure, each majority voter provides a 2-bit error signal (referred to as e_i, where $i \in \{1, 2, ...n\}$) to the corresponding NI. Three possible values of the error signal — 00, 01, and 10 — represent the error states of the triplicated modules respectively,

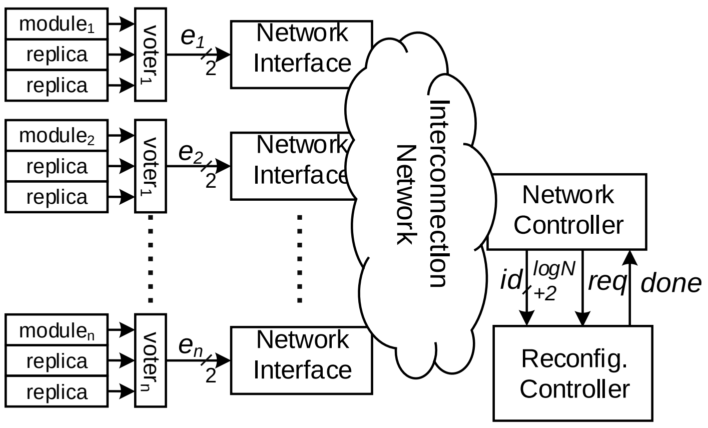


Figure 4.1: Sub-components of an RCN

while the value 11 indicates that there is no error. Note that a voter raises a reconfiguration request to the NC by means of the error signal value. Once the NC receives any valid error report, it issues a *req* and provides a $\log_2$(N+2)-bit module *id* signal to the RC, which invokes the module-based recovery process to recover the faulty module. After the module-based error recovery is completed, the RC asserts a *done* signal in order to resume the checking of the NC.

4.2.1 Star Network

Figure 4.2(a) illustrates, at an overview level, a typical star network implementation. Each NI contains a 2-bit buffer that connects directly to the NC. The NC consists of three modules, namely a check counter (*cnt*), a multiplexer and a comparator. The error counter selects which voter is to be checked next in a round-robin manner, while the multiplexer transfers the error request from the selected voter to the comparator. When the value is not equal to “11”, the comparator toggles the *req* signal and the *id* signal is composed of the selected voter output and the current counter value. The RC invokes partial reconfiguration for the faulty module before issuing a reconfiguration *done* signal to the counter to resume voter checking.

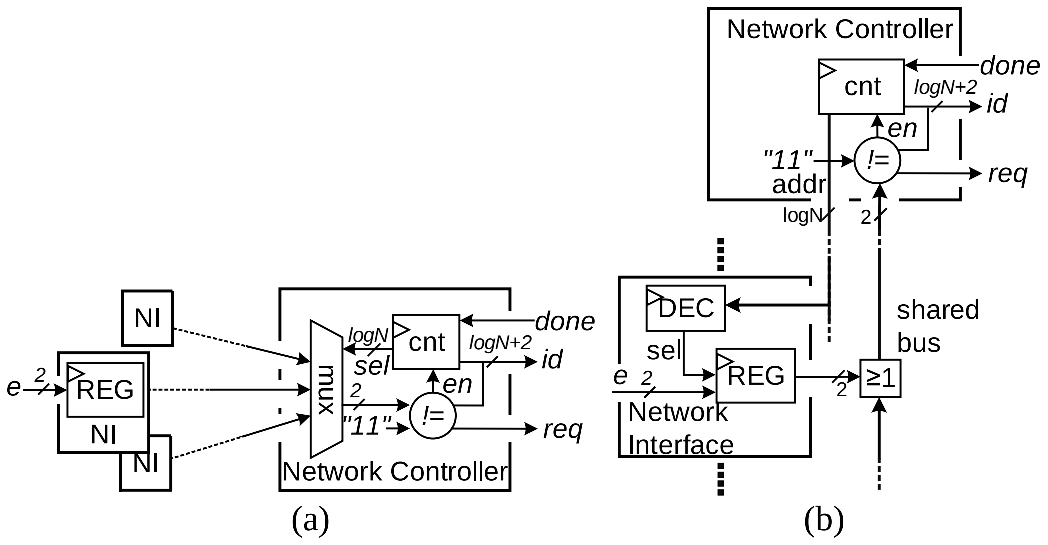


Figure 4.2: The architecture of a star network (a) and of a bus (b)

4.2.2 Bus

We detail a simple bus that aggregates the error messages from the voters as illustrated in Figure 4.2(b). The bus works similarly to the star network. The multiplexer architecture of the star network is replaced with a global address network. 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 error, the NC triggers the reconfiguration process in the RC.

4.2.3 Token Ring Network

Figure 4.3 shows the architecture of a token ring network as detailed in Section 3.3, but modified to transfer RReq as a parallel word rather than bit-serially. In contrast to the star architecture, each NI has a pre-defined id, and combining it with the 2-bit error signal provided by the voter, the NI forms a ($\log_2 N$ + 2)-bit Reconfiguration Request (RReq) message that indicates which module is in error. When the *val* signal toggles and the message on the parallel data signal is a token, the OutRegs in the NI latches the token before passing it onto downstream NIs. If an NI needs to send an RReq, it keeps the token and its OutRegs latch the current RReq message. The downstream NIs pass this message to the NC. The NC operates in a similar manner to that described for the star network. When a modular reconfiguration has occurred, a Reconfiguration done (Rdone) message

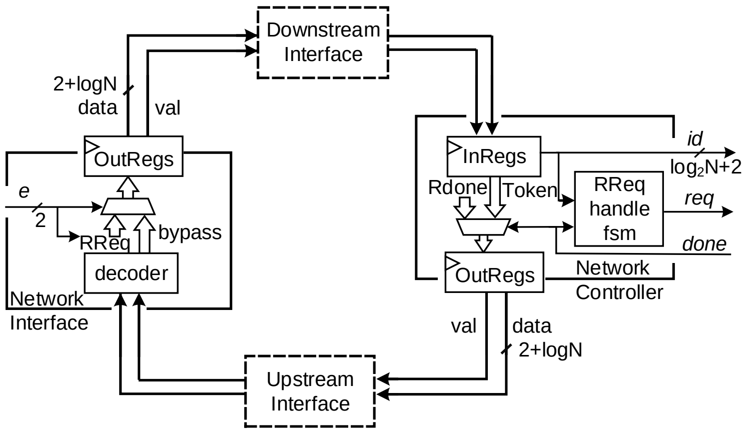


Figure 4.3: The architecture of a token ring network

is released to signal the requesting NI that the reconfiguration has been completed and to release the token.

4.2.4 ICAP-based Voter Checking

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

Configuration Frame Readback

Xilinx FPGA configuration memory is arranged in frames that are tiled about the device (for more information about the configuration memory architecture, please refer to Appendix A). Frames are addressable and accessible via configuration ports such as ICAP. In Xilinx 4–7 Series devices, frames for Configurable Logic Blocks (CLBs) contain all of the configuration bits for flip-flops and look-up-table values, whereas, the state of flip-flops cannot be directly accessed except by performing a Readback Capture as described in [51]. In order to know the frame address and configuration bit offsets of the memory cell that stores the corresponding register value when the GCAPTURE command (for details refer to Appendix B.3.3) is issued, Xilinx ISE/Vivado tools can automatically generate a logic location (*.ll) file during the "write bitstream" procedure that provides these parameters. The controller of the ICAP-based voter checking approach needs to be able to filter the desired flip-flop states from frame data according to the logic location file. In order to read a frame that contains the desired flip-flop value, Xilinx devices expect a specific sequence of commands to be sent to the ICAP [51]. A discrete frame read request necessitates the read of a dummy word and a pad frame before the frame can be read. The time to read a frame in Xilinx 7-Series FPGAs is approximately 230 clock cycles. This includes about 20 clock cycles for issuing initialization commands, 203 clock cycles for the frame read, and 10 clock cycles for issuing concluding commands. The frame read time depends on the throughput of the ICAP, which supports 32-bit transfers at a rate of 100MHz. At the maximum rated speed, the time for reading a frame on its own through the ICAP primitive in 7-Series FPGAs is thus approximately 2.3 us. This latency can be further reduced as much as 1 us via a pipelined readback method. For more details about both single and sequential reconfiguration operations, please refer to Appendix B.3. For the ICAP-based approach, if voter status registers are placed in different CLB columns, this latency is required for each voter check, but all of the voter values can be read at once if the voter registers are placed in the one column of CLBs so that all of the flip-flop values are concentrated in only a few frames or, at best, just one frame.

4.2.5 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 above, all four networks check voters in a round-robin manner. Thus, assuming a system with N TMR voters or NIs and one NC, the average latency of the token ring network is given by

$$latency = (N+1) \times c_{hop} \times \frac{1}{F_{network}} \tag{4.1}$$

where c_{hop} denotes the number of clock cycles per node hop, and $F_{network}$ denotes the maximum clock frequency of the RCN. Equation (4.1) corresponds to the average time needed for the token to arrive (half of 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

$$latency = \frac{N}{2} \times c_{hop} \times \frac{1}{F_{network}} \tag{4.2}$$

which corresponds to the time it takes to check half the voters in the system.

4.2.6 Synchronization Support

Synchronization methods after reconfiguration can be divided into two types, namely, "passive" synchronization and "triggered" synchronization. The passive approach does not require initiation after the recovery of a faulty module [4, 50]. The SEU-induced error effects are cleaned up when a periodic module reset toggles, or when new data flushes the pipeline. This kind of module synchronization approach does not require the RCN to be able to distribute a reconfiguration done message to voters. However, for triggered synchronization methods such as check-point and state prediction or state migration approaches, as discussed in Section 2.3.4, RCNs need to be capable of two-way communication. The star network, bus and token ring network can easily be extended, whereby the star and bus just need to add an extra signal for issuing a *done* signal to the voter after reconfiguration, and the token ring network just needs to toggle a *done* signal when the Rdone message is received by the network interface. It is possible for the ICAP-based voter checking approach to write commands to the voter without using programmable resources. An easier approach is to use the RePin architecture and the dynamic reconfiguration technique, as described in the following Section 4.3.2. The delay to instruct a voter that the reconfiguration has been done via RePin comprises at least a single frame read and a single frame write operation, which is roughly twice the latency of a single frame write or 4.6 us in total. An alternative but less straightforward approach is based on the special column frame described in Appendix A.4, which allows us to write values into flip-flops within a major column via dynamic reconfiguration. This approach involves creating a global mask that controls which flip-flops are read or written to in the current frame via GCAPTURE and GRESTORE instructions [51].

4.3 Experimental Evaluation

In this section, we evaluate the performance of the networks presented in Section 4.2, in terms of resource utilization, latency, operating frequency, power consumption and soft error vulnerability. All networks were implemented on a Xilinx Artix-7 XC7A200T FPGA, as hosted on the RUSH experiment board [16], 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 the results of fault injection experiments.

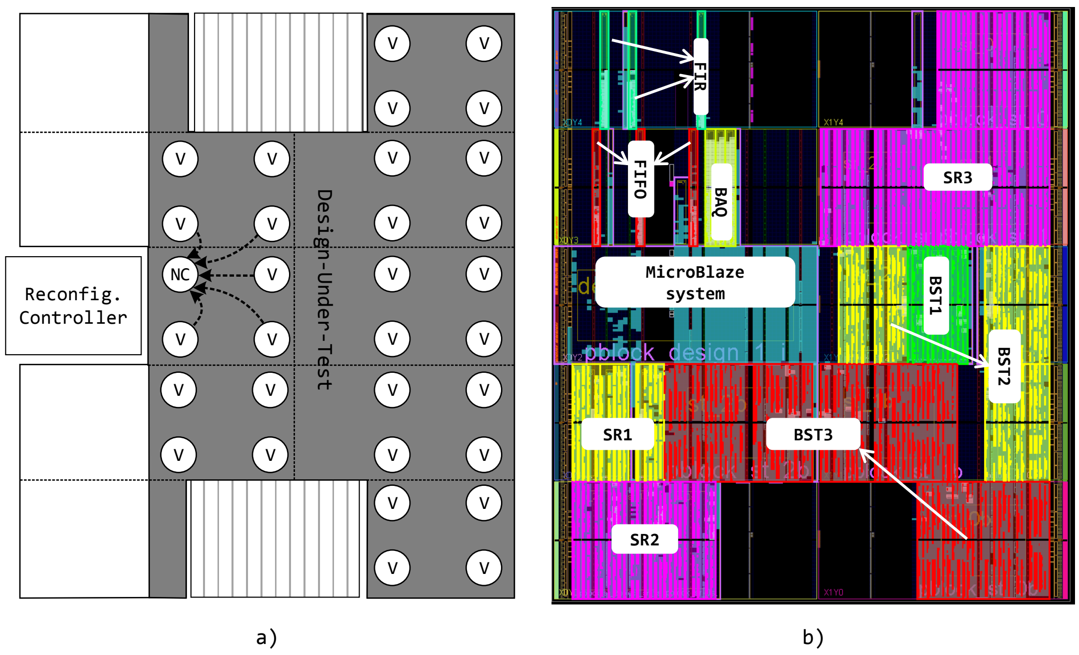
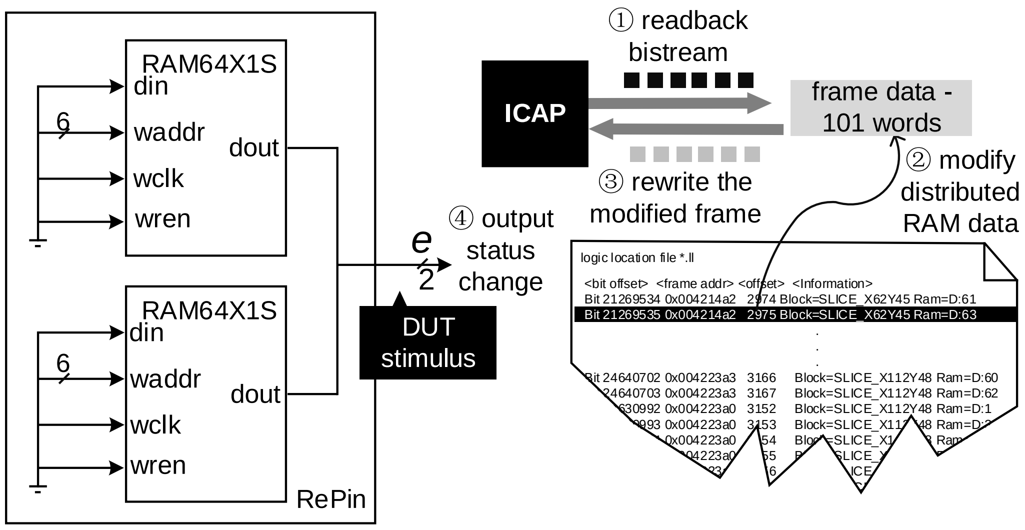


Figure 4.4: (a) Synthetic layout of a 31-voter design and (b) RUSH floorplan

4.3.1 Synthetic Experiment

As mentioned in Section 4.2, an RCN consists of NIs, a central NC and the interconnection network between them. In our experiments, the same voter interface and RC designs were used in each experiment irrespective of the RCN type being tested. The same NI and NC locations were also used for all RCN designs. The NIs and the NC were allocated to partitions that utilized the same FPGA resources irrespective of the RCN topology. We first studied "scattered" 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. To obtain resource utilization and performance results, we initially implemented designs that only contained the components of the RCNs being tested and constrained the design 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 distributing test vectors to each of the RCNs. We tested each RCN type for networks comprising 7, 15 and 31 voters. The scattered layout of a 31-voter design (in this case for testing the star network topology) is shown in Figure 4.4(a), in which the design under test into which faults were injected is depicted as the shaded region to the right of the RC.



```
logic location file *.ll

<bit offset>  <frame addr> <offset>  <Information>
Bit 21269534 0x004214a2   2974 Block=SLICE_X62Y45 Ram=D:61
Bit 21269535 0x004214a2   2975 Block=SLICE_X62Y45 Ram=D:63
                               .
                               .
                               .
Bit 24640702 0x004223a3   3166    Block=SLICE_X112Y48 Ram=D:60
 24640703 0x004223a3   3167    Block=SLICE_X112Y48 Ram=D:62
   630992 0x004223a0   3152    Block=SLICE_X112Y48 Ram=D:1
    0993 0x004223a0   3153    Block=SLICE_X11 48 Ram=D:
        0x004223a0    54    Bloc =SLICE_X1   Ram
          04223a0     55    B   SLICE_
              23       6         ICE
```

Figure 4.5: RePin architecture for input stimulus

4.3.2 RUSH Experiment

In a second experiment, we investigated the utilization and performance of each RCN when used to gather reconfiguration request for the RUSH experiment [16]. For this case study, we implemented the four network types with the 9 TMR components comprising the RUSH payload. These 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 31-bit Shift Registers (SRs) having different lengths and a range of combinational logic between the states, and three 32-bit Binary Search Tree (BSTs) of different heights and a range of combinational logic at each node. A MicroBlaze processor was used to implement the RC and the AXI HWICAP IP was used to reconfigure faulty modules. The layout of this design is depicted in Figure 4.4 (b).

4.3.3 Fault Injection Methodology

In this section, we outline the fault injection methodology and the system we used to assess the soft-error vulnerability of the RCNs studied on the proposed synthetic layout. Fault injection involves the use of Dynamic Partial Reconfiguration to artificially insert faults into the FPGA's configuration memory. When a fault is injected, the effect of the error is accessed by checking the output of the Design-Under-Test (DUT).

We use the Xilinx HWICAP IP for flipping configuration bits as directed by the MicroBlaze processor. Once the system has been initialized, the processor halts and waits for a random frame address and configuration bit offset from a host PC program. It then reads the corresponding frame, flips the designated address bit and writes the frame back to emulate an SEU. The detailed way to read and write a frame discreetly is described in Appendix B.3.2. Note that errors will only be injected into the DUT region. The host PC program parses the partial bitstream of the DUT region in order to obtain the list of frames to inject into. Of the 18,300 frames in the device targeted in our study, 18,092 frames were contained in the region that represents the DUT.

Once a fault is inserted, the circuit is tested using all possible input stimuli. In our case, there are four possible error signal values that would normally be presented to a Network Interface (NI). An input stimulus is usually provided through external pins on the FPGA or GPIO of the test engine. However, we felt that errors injected into the nets leading from those IOs would influence the results of our experiments. Therefore, we developed a "wireless" way of generating input vectors for testing the effect of a fault, as shown in Figure 4.5. The approach is also realized through dynamic reconfiguration through what we called a "RePin" architecture, which is composed of two SLICEM LUTs (LUTMs) configured as distributed RAM. Each LUTM can provide one of two bits of stimulus that can be changed using the ICAP. Given the site number of logic locations, the positions of the LUTM contents can be obtained from the logic location files as described in Section 4.2.4.

After injecting the fault, the processor checks the integrity of the design while the RePins are manipulated to stimulate the voter behavior. Whenever a new RePin output is set, the MicroBlaze checks that the correct reconfiguration request is received. If the NC reports the correct error value as expected, we change the RePin to the next possible error output value. When we have cycled through every possible voter status, and there is no unexpected output, we move on to the next NI. If an unexpected value is detected, an error report is sent to the PC, and further testing of the bitflip is abandoned. Testing proceeds with the next configuration bit until a million random bits in the DUT have been tested.

Table 4.1: Results of mapping four RCNs to 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

4.4 Results

4.4.1 Implementation Results

Table 4.1 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 used follow the same pattern. In contrast, the vendor's power analysis tools reported the same amount of static power consumption for all RCN designs. However, given that the RCN designs utilized less than 0.2% of the total programmable resources on average, we believe that the contribution of the used resources is negligible, and due to this, we have obtained the same result for all designs.

It can be seen 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 that is 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 star, bus and ring networks respectively. However, the ICAP-based RCN suffers from high network latency. It requires two or 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 (4.1) and (4.2) 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 latencies of the star and bus networks were 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 4.4(a), which depicts 4 voters per clock region (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 4.1 in the ICAP column headed L1*. As can be

Table 4.2: Number of critical bits detected from 1 million random fault injections performed in the four types of network with different number of voters over 5 trials.

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

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%.

4.4.2 Fault Injection Results

Essential bits are those configuration bits that are used to configure a user circuit. If an essential bit changes, it changes the design circuitry. However, it might not necessarily affect the function of the design [44, 60]. The purpose of the fault injection campaign was to estimate, for each RCN type, the number of critical bits, which, if changed, result in a failure of the network function [60]. This establishes a more accurate measure of the sensitivity of a design to faults, since the essential bit reckoning provided by Xilinx typically overestimates the vulnerability of user designs.

We implemented the fault emulation system described in Section 4.3.2 to conduct fault injection experiments for the synthetic layouts of each of the four RCN types. We carried out 5 trials of injecting one million random faults into each RCN type. The coverage of each trial was restricted to one million of the 59 million configuration bits in total present in the DUT. Of the total number of configuration bits in the DUT, about 5 million bits — those pertaining to the fault injector and the test harness — were excluded from the test so as to avoid corrupting the fault injector. During each fault injection cycle, the RCN implementation was verified by toggling the RePins as described in Section 4.3.2 and the output from the network controller was checked against the desired value.

Table 4.2 tabulates the average number of errors and their standard deviations (SD) as found after five trials of one million random fault injections. These results demonstrate that the ICAP-based RCN is more reliable than the other approaches. Additionally, the number of critical bits that occur in each RCN is directly proportional to the number of voters and thus the number of essential bits per design.

Table 4.3 compares the essential bit results predicted by the implementation tool and the

Table 4.3: Comparison between fault injection results and essential bit results

Type	**ICAP**			**STAR**			**BUS**			**RING**		
#voters	7	15	31	7	15	31	7	15	31	7	15	31
% critical bits	7.0E-4	8.2E-4	2.1E-3	8.2E-4	1.7E-3	3.9E-3	1.7E-4	3.7E-3	7.9E-3	5.1E-3	1.2E-2	2.1E-2
% essential bits	5.8E-3	9.5E-3	1.8E-2	1.6E-2	2.7E-2	4.4E-2	2.0E-2	3.4E-2	6.5E-2	3.1E-2	5.7E-2	1.2E-1
Sensitivity	0.121	0.086	0.117	0.051	0.063	0.089	0.085	0.109	0.122	0.165	0.211	0.175

fault injection results. The third row of the table lists the percentage of critical bits found in one million random fault injections, while the fourth row reports the ratio of essential bits to total configuration bits (about 59 million) as a percentage. The sensitivity of an RCN to faults is derived by dividing the critical bit result by the essential bit result. We have found that despite experiencing the lowest SEU rate, the ICAP-based approach promises a low and stable trend on fault sensitivity. The star-type RCNs have the lowest fault sensitivity. By contrast, the ring network experiences a fault rate when SEU occur.

4.5 Discussion

In this chapter, 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 synthetic layouts of 7, 15 and 31 voters, and with an experimental design layout having 9 voters mapped to the RUSH board. 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 ICAP-based approach has the highest reliability despite having a relatively high latency, but the latency of this approach can be reduced by clustering into a single frame the registers that are to be read from the one clock region. This optimization does not significantly impact on resource utilization. The star network is also a strong contender due to its high performance, which contributes to reducing the MTTR and to a relatively high reliability.

Chapter 5

Fine-grained Module-based Error Detection and Correction

In [12], we found that FPGA SoCs that solely rely on TMR-MER have lower reliability than those relying on TMR-S unless the RCN is also triplicated and corrected when configuration memory errors become evident in it. Due to the distributed nature of RCN resources, we resorted to scrubbing the whole device when an RCN was affected by configuration memory errors. Although error recovery using scrubbing is slow and energy is wasted checking/reconfiuring frames that are not in error, it was found that scrub operations were only occasionally needed since the triplicated RCN had a relatively low susceptibility to errors due to the comparatively few resources utilized by this system component.

The work described in this chapter, which has been submitted to ICFPT2016 [15], aims to address the considerable latency and energy used scrubbing the device when the RCN is affected by configuration memory errors. Our contributions are:

1. To localize configuration memory errors more precisely than has previously been reported in the literature and to explain how the response to error signals should be prioritized;
2. To describe a fine-grained method for dynamically reconfiguring sub-components that are suspected of containing configuration memory errors; and
3. To compare the reliability, latency and energy cost of correcting configuration memory errors using the proposed approach with (a) TMR-MER and complete scrubbing

of the device when errors are detected outside the TMR modules, (b) on-demand scrubbing of the device when an SEU is detected in the system, and (c) periodic scrubbing of the device as a fault prevention strategy.

This work takes a significant step towards solving the drawbacks of TMR-MER as discussed in Section 2.3.4. In this chapter, a repair strategy is used to enhance the localization of single errors within fully-triplicated systems. Inspired by the fact that scrubbing is able to rewrite a frame without resorting to the Partial Reconfiguration Flow [22], a fine-grained reconfiguration approach is applied to determine configuration frame sets pertaining to either block or non-block sub-components after a system is implemented. To save on the cost of storing partial bitstreams, we describe and demonstrate a dynamic bitstream composition method that retrieves arbitrary frame data from the full bitstream at run time. These ideas have, to our knowledge, not previously been reported in the literature.

5.1 Fine-grained Error Detection Strategy

5.1.1 Fully-triplicated Circuit Model

Our target system model (Figure 5.1) is based on the model we described in Section 2.3.4, where we assume the user circuit comprises n TMR components forming a linear sequence of TMR components. In order to maximize the reliability, each component consists of three identical circuit replicas, referred to as modules, and additional sub-components, including voters, intra-component nets that connect between circuit modules and voters, and inter-component nets that form the link between voters and downstream components, as well as between voters and the reconfiguration controller. The voter architecture is derived from [50] which only fulfills the role of providing the majority of triplicated module outputs and indicates which module is in the minority as an error signal. The error detection circuitry does not detect errors in the voter block.

The error reports from the triplicated voter blocks are aggregated by a central Reconfiguration Controller (RC) through three identical star-type Reconfiguration Control Networks (RCNs). The RC thus receives triplicated error reports from each component. Figure 5.2(a) illustrates the triplicated RCN. The RC manages the triplicated Network Controllers (NCs) in order to check the error state of a particular component. The RCNs are synchronous and all NCs operate in lock step to check the voter of each component in turn according to the selection of the RC.

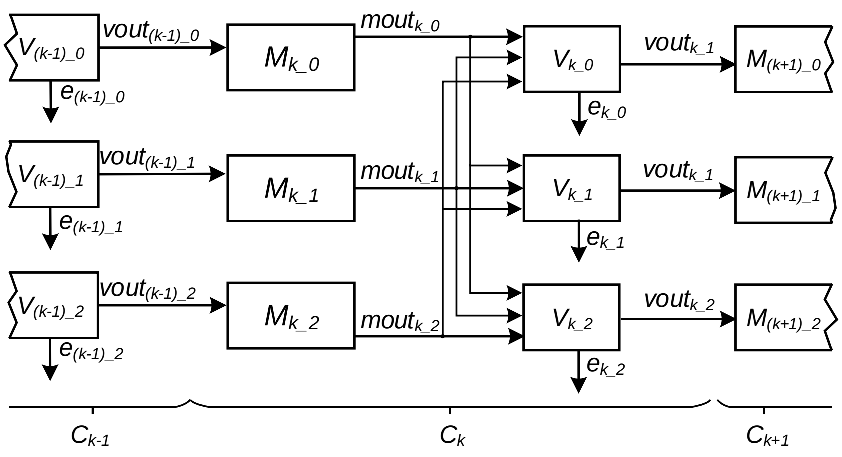


Figure 5.1: Target TMR Model

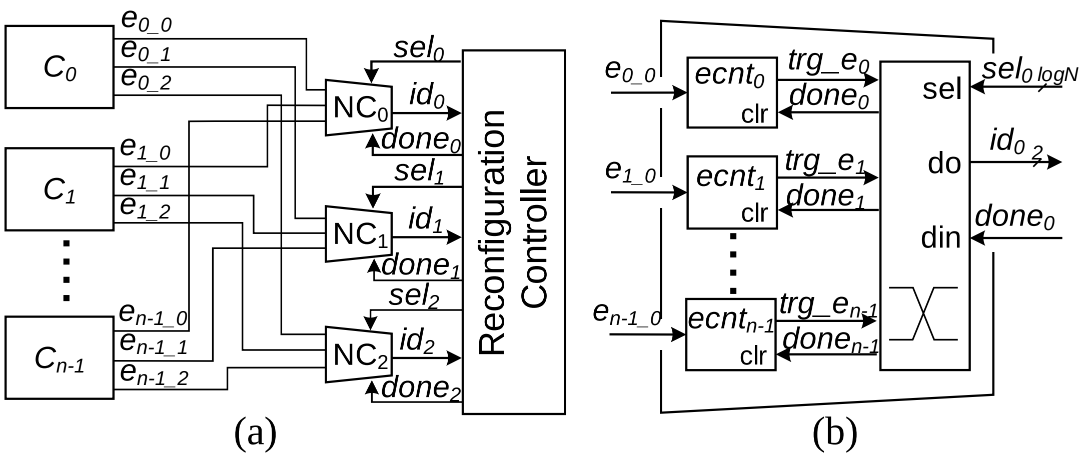


Figure 5.2: (a) Triplicated Star-type RCN, (b) Elaborated NC logic

The NC aggregates error signals and distinguishes between transient errors in the user circuit and permanent errors that are likely due to configuration memory corruption. Figure 5.2(b) elaborates the architecture of an NC. Each of the error signals inputs to a saturating up-down counter. A permanent error is associated with repeated error signals that are expected to saturate the counter and trigger an error report trg_e_i to the RC [3]. Transient errors will result in both up- and downward counts and will therefore not saturate the counter. A switch selects between the error triggers of the individual counters.

The RC controls the *sel* signal to the NC switch to select the output of the desired error counter, which triggers the partial reconfiguration of the indicated module. The RC then retrieves the frame data of the indicated module from external memory and writes these to the configuration memory space. When the reconfiguration is complete, the *done* signal is asserted to clear the corresponding error counter.

5.1.2 Persistent Errors

Permanent errors reported by the counters in the NCs may be, as indicated, due to configuration memory errors present in one of a component's modules. But a substantial number of reported errors are caused by permanent errors present in the voters, the intra-component nets connecting modules and voters, the nets connecting voters to downstream components, as well as in the wires of the RCNs and the NCs themselves. In this chapter, we study the effect of configuration memory SEUs on TMR-MER systems and propose a remediation scheme that enhances the reliability of TMR-MER that is more responsive, and saves considerable energy with respect to equivalent systems employing scrubbing.

Configuration memory errors in different parts of a component, as shown in Figures 5.1 & 5.2, result in a range of error symptoms. The error reports received by the RC due to errors in the various sub-components are as follows:

- A bitflip in the configuration of M_{k_j} may cause $mout_{k\_j}$ to differ from $mout_{k\_i}$, $i \neq j$. In this case all three error outputs for C_k should report M_{k_j} to be in error.
- Bitflips in the net $vout_{(k-1)\_j}$ or the majority voter logic of $V_{(k-1)\_j}$ can cause the input for M_{k_j} to be incorrect. The three RCNs will then report that M_{k_j} is incorrect but cannot correctly determine that the error lies with the logic of the upstream voter or with its corresponding outgoing nets.
- If an error is present in the minority voter logic of V_{k_i}, the voter may assert that a bitflip is present in C_k while the other two voters report no error. The RC thus only receives one error report for C_k instead of three.
- A fault present in the branch of $mout_{k\_j}$ can cause the connected voters to incorrectly indicate the presence or absence of an error in M_{k_j}. If the fault is present in the main trunk of the net, all three voters, or just two of them, may assert an error. Errors in the region of $mout_{k\_j}$ where it branches may also cause two different voters to indicate that different modules are in error. This is because a single error in a switch matrix can affect more than two nets [46]. If a bitflip is present in the switch matrix used by any two of $mout_{k\_0,1,2}$, two voters may report different modules to be in error.

- If a configuration memory error is present in one of the triplicated RCNs, that RCN may raise or mask errors for a random module in some component.

It is evident that errors in the various sub-components of C_{k-1} and C_k or in RCN$_j$ result in different behaviors at the outputs of the triplicated RCNs which we refer to as error signatures. A single error may cause one of three different types of error signature to be observed for C_k. These are: (i) three identical error reports ($id_{0,1,2} = M_{k\_j}$), (ii) one error report ($id_i = M_{k\_j}$), and (iii) all others.

If the error signature is of type (i) and the reconfiguration of the module indicated by the RCNs fails to clear the error, the same error signature will present after the recovery operation, in which case the error is deemed to be persistent. It should then be suspected that the error is in $V_{(k-1)\_j}$, $vout_{(k-1)\_j}$, $mout_k$ or $mout_{(k-1)}$. For error signatures of type (ii), besides $mout_k$, V_{k_i} and the RCN logic (NC$_i$ & e_{k_i}) are suspects. For error signatures of types (iii), the only suspect is $mout_k$.

Other possible RCN outputs are caused by an accumulation of SEUs. Two or more errors may then be present in different sub-components. We do not consider these effects in this thesis, but all could be dealt with by triggering a complete scrub of the device when they appear.

5.1.3 Repair Strategy

We propose a repair strategy to guide the design of an RC that is capable of detecting and repairing persistent errors in systems using fine-grained dynamic reconfiguration. The repair strategy includes a repair flow and a recovery sequence as determined by the flow.

Repair Flow

Figure 5.3 depicts the proposed flow for recovering from persistent errors in the system. An error check and correction cycle commences at entry point A, when the component number K and the error signature E are cleared. The first component is checked after initialization. K is incremented while the RCN reports no errors for the component currently being checked. When an error is reported for C_k the error signature is compared with the previously recorded signature E and the check index Chk is incremented, if so. The error signature and check index indicate which sub-component R is to be reconfigured (Table

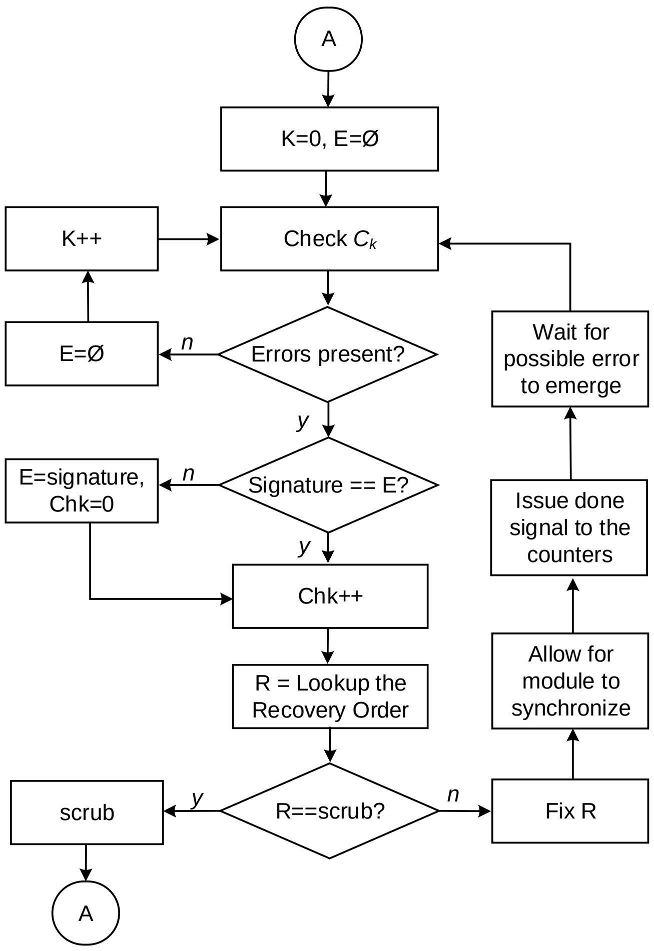

A
K=0, E=Ø
K++
Check C_k
E=Ø
n
Errors present?
y
E=signature, Chk=0
n
Signature == E?
y
Chk++
R = Lookup the Recovery Order
scrub
y
R==scrub?
n
Fix R
A
Wait for possible error to emerge
Issue done signal to the counters
Allow for module to synchronize

Figure 5.3: Repair Flow

5.1). After the dynamic reconfiguration of the sub-component has been performed, the RC waits for the component to be resynchronized [3] and issues a done signal to the RCN to reset the error counters for C_k. After the error counters have been cleared, the RC waits for a period of time to allow any residual error to once again manifest itself by saturating the error counters. This period depends upon the saturation level of the counter and the latencies of component C_k and the RCN. These wait times are of the order of a few µs [3, 12]. The RC checks whether C_k is still affected by errors, and if the same error signature is decoded, Chk is incremented in order to recover the next sub-component indicated by the recovery sequence. If errors persist after all suspect sub-components have been reconfigured, a scrub is performed to sweep away all accumulated errors, and the recovery flow is re-initialized by returning to entry point A.

Table 5.1: Recovery Sequence

Error Signature	**Number of Checks (Chk)**					
	1	2	3	4	5	6
i. $id_{0,1,2} = M_{k\_j}$	M_{k_j}	$V_{(k-1)\_j}$	$vout_{(k-1)\_j}$	$mout_k$	$mout_{(k-1)}$	scrub
ii. $id_i = M_{k\_j}$	V_{k_i}	$mout_k$	NC_i	e_{k_i}	scrub	
iii. others	$mout_k$	scrub				

Recovery Sequence

Table 5.1 lists the repair order we use with the flow. The recovery sequence is based on the suspects identified for each error signature. For each type of signature, the priority is to recover logic blocks first, followed by the component nets. The main reason for this preference is that the programmable bits are more densely located in the logic blocks than in the nets, while the nets are more vulnerable at their source and destination terminals, which are always included in the logic blocks. Reconfiguring the block components first results in higher repair efficiency.

- For signature (i), M_{k_j} is to be recovered first. If the recovery of the module does not clear the error, the error is deemed persistent and $V_{(k-1)\_j}$ is recovered next, followed by the three component nets $vout_{(k-1)\_j}$, $mout_k$ and $mout_{(k-1)}$.
- For signature (ii), since the utilization of routing and switch resources for dedicated nets and NC-related logic is far lower than for the logic sub-components, the recovery sequence commences with logic blocks first and follows up with recovery of the RCN sub-components: V_{k_i} and $mout_k$ are recovered first, followed by NC_i and the dedicated routing for e_{k_i}.
- For signature (iii), it is only worthwhile recovering $mout_k$ before resorting to a complete scrub of the device.

5.1.4 Triggered Scrub

Systems that solely rely on scrubbing to correct configuration memory errors usually perform a scrub periodically as they do not usually monitor system outputs to determine when configuration memory errors may be present. However, when the outputs of system components are monitored using the architecture outlined in Section 5.1.1, it is feasible to trigger a scrub cycle when a configuration error is detected. In this case, the error correction flow of Figure 5.3 can be adapted to perform a scrub whenever an error is detected, and to restart the flow after the scrub is finished. In this case the prioritized recovery sequence is ignored.

5.1.5 MER with Scrubbing (MER/Scrub)

Classic TMR-MER systems need to resort to scrubbing when errors are not present in TMR modules, typically these occur in the RCNs, as identified in [12], and in the voters or interconnecting nets between voters and modules, as explained in this section. For error signature (i), the system should be able to detect repeated recovery action to avoid repeating the same recovery cycle again and again, and should then perform a scrub cycle as the remedy. MER/Scrub also relies on the flow of Figure 5.3 to determine when to perform a scrub cycle. However, the recovery sequence is changed to perform a scrub when the Chk exceeds 1 for error signatures of type (i), and to perform a scrub whenever an error of type (ii) or (iii) is detected.

5.2 Fine-grained Dynamic Reconfiguration

Conventional TMR-MER SoCs (e.g. [3], [6], [4]) rely on the vendor's partial reconfiguration flow [22] to generate partial bitstreams for the component modules. These partial bitstreams are stored in external memory and a lookup table is created for the RC to index these files [3,4]. When a permanent error is detected, the RC fetches the indexed file from memory and writes it to the ICAP. However, as indicated in Section 2.3.4, the flow is not flexible enough for robust and efficient SEU recovery in fault-tolerant systems. This is mainly because the original intention of the flow was to create a partial region to allow for dynamic hardware changes. Partition pins have to be placed on the boundary of the region, which result in extra delay for the interface signals, as well as slower compilation and longer design cycles. Most importantly, in the architecture described in Section 5.1.1, the RCN and the interconnecting nets between voters and modules are not amenable to modular reconfiguration using the partial reconfiguration flow.

In this section, we propose a fine-grained dynamic partial reconfiguration approach (FDPR). Our approach uses the standard FPGA project flow while overcoming the drawbacks of the vendor's partial reconfiguration flow when applied to fault-tolerant systems. The approach includes a design method for identifying the sets of frames pertaining to the sub-components such as the module outputs $mout_k$, voter outputs $vout_{k\_j}$, RCN nets e_{k_i}, voters and network controllers. We also describe a bitstream composition method that enables retrieving arbitrary sets of frame data from a full bitstream at run time.

5.2.1 Major Columns

We describe the proposed FDPR approach on the Xilinx 7-series devices. Programmable resources on a 7-Series device are tiled into major columns [22, 51]. A major column is a column of resources in a single clock region, of which there are several in a device. It also includes a column of switch matrices that provides access to the general routing matrix. A major column is configured via a contiguous set of configuration frames, each of which forms a bit slice of the configuration memory for the major column. The size of the contiguous set of frames used to configure a major column depends upon the type of resource it contains. Every pair of major columns shares a local clock buffer. The detailed device model based on major columns and conversion between major column index and frame address is detailed in Appendix A.

5.2.2 Configuration Memory Boundaries

An FPGA application circuit is typically implemented using a number of CLBs. Interconnecting signals are routed via switch matrices to their destination CLBs. Floorplanning enables resource allocation within a specified region, which may be as narrow as a single major 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. If the design is synchronous, then local clock buffers are also used, and the other major column that shares the clock buffer must also be reconfigured.

The detailed routing of nets that interconnect different logic blocks can be retrieved from the implementation database of the design. The information stored in the database includes the name of any entry or exit Programmable Interconnect Pin (PIP) the net uses, the switch matrices that own these PIPs, and the segments of the general routing matrix forming the net. The major columns that implement the net can be extracted from the

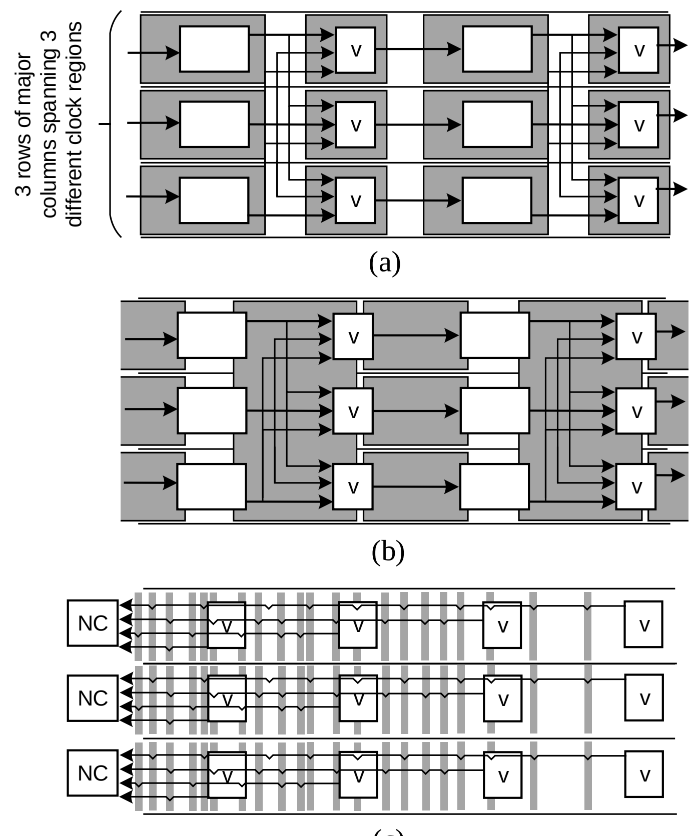


Figure 5.4: One possible floorplaning (a) logic, (b) intra-component nets, (c) RCN

names of the switch matrices used. Configuration memory errors in a net can thus also be recovered by simply reconfiguring the configuration frames of the major columns it uses.

5.2.3 Floorplanning

Proper floorplanning before implementation can minimize the number of frames per sub-component, and can provide isolation between the copies of a triplicated design.

Figure 5.4 illustrates one possible floorplan for the proposed TMR architecture outlined in Section 5.1.1. The circuit spans three rows of major columns. Modules and voters are constrained to be placed in different major columns as shown in Figure 5.4(a). Figure 5.4(b) presents the major columns relating to the interconnection nets between the modules and the voters. As the output nets from the triplicated modules ($mout_{k\_0,1,2}$) are intertwined, we treat them as a single sub-component for recovery purposes. The router automatically routes voter outputs and module outputs in the gaps between the closest module and voter. Figure 5.4(c) shows the major columns relating to the triplicated RCN. The NCs are also isolated and constrained to different rows of major column. The NCs connect to the voters via error signals. The signal sinks correspond to switch matrices. The major columns that contain these switch matrices will be recovered if the RCN signals are determined to be in error. The routes of the triplicated RCN are isolated because they are implemented in different major column rows. However, in large-scale systems, the RCN connects voters from different regions of the device. It may then become impossible to isolate the RCN replicas from each other in this way. Although few single errors would cause more than two RCN replicas to fail, a better design method is then needed to avoid the use of PIPs that will bridge two nets if corrupted.

To improve area usage, small modules can be implemented in a single clock region using a similar layout. However, in this case, each copy should be provided with its own clock signal to ensure ongoing protection if an SEU disables a clock buffer.

5.2.4 Dynamic Repair Bitstream Composition

A full bitstream is written when the device boots. This bitstream comprises a bitstream header and configuration frame data. The bitstream header is the command sequence that is loaded before the frame data are written. After the commands are issued to the device fabric, the data for the first frame, at frame address 0, are written. The frame address register auto-increments and the data of the second frame are written straight after the last word of the first frame.

Usually a full bitstream is difficult to index. However, compared with a typical full bitstream, the full bitstream for critical systems has single-frame CRC checking enabled. Frame headers are then inserted after every single frame of data. The header contains the CRC check sequence as well as the address of the previous frame. We use this information to extract frame addresses and to index them according to their byte offset in the file. When the bitstream is loaded into external memory, we can then extract the data for arbitrary frames according to their byte offsets.

Storing the frame index potentially necessitates a large context memory for the RC. To minimize the amount of memory required, we only index the address of the first frame for each major column that is needed for the recovery strategy. The range of frame addresses for the sub-components can be calculated off-line and stored with the index. The frame write operation is done as for the full bitstream with single frame CRC check enabled. First, issue the command sequence. Second, write the first frame data, which is stored in the internal frame buffer of the configuration port. Then, set the frame address of the first frame's data. Writing the second frame pushes the first frame's data to its destination, and so on. To complete the reconfiguration of a major column, a pad frame is written in order to push the data of the last frame of the major column to its destination.

5.3 Experimental Evaluation

In this section, we evaluate the recovery latency, energy used in correcting configuration memory, and the reliability of the proposed approach, and compare the results with those obtained for MER when scrubbing is used to recover from errors that occur outside the TMR modules, with on-demand scrubbing that is triggered by SEUs located anywhere in the system, and with periodic scrubbing. We evaluate the fine-grained dynamic reconfiguration method described in Section 5.2 via an exhaustive fault injection experiment on a typical hardware implementation of the system model of Section 5.1.1, based on a Nexys-4 Video board implemented using Vivado 2015.4. The board includes a Xilinx Artix-7 XC7A200T device, which is clocked at 100MHz.

The purpose of the fault injection experiment was to find the critical bits of the test circuit, to log their locations, and to record the error signatures they gave rise to. The results were used to validate the persistent error effects described in Section 5.1.2, to evaluate the soft-error vulnerability of the sub-components in the test circuit, and to evaluate the performance of the proposed recovery method.

5.3.1 Experimental Setup

Test Circuit

Figure 5.5 depicts a block diagram of the test system implemented on the Nexys-4 Video board. The test circuit comprises two TMR components representing both cyclic state-machine logic and acyclic datapath logic [3]. To represent a typical state machine, we chose a 64-bit Linear-Feedback Shift Register (LFSR). The LFSR serves as a random test vector generator for the DUT and emulates the upstream voters and nets illustrated in Figure 5.1. The acyclic datapath logic was represented by a Shift Register (SR) module in which all logic paths travelled without feedback from the input to the output of the module. The module comprised 8 stages of 64-bit registers with a variety of arithmetic

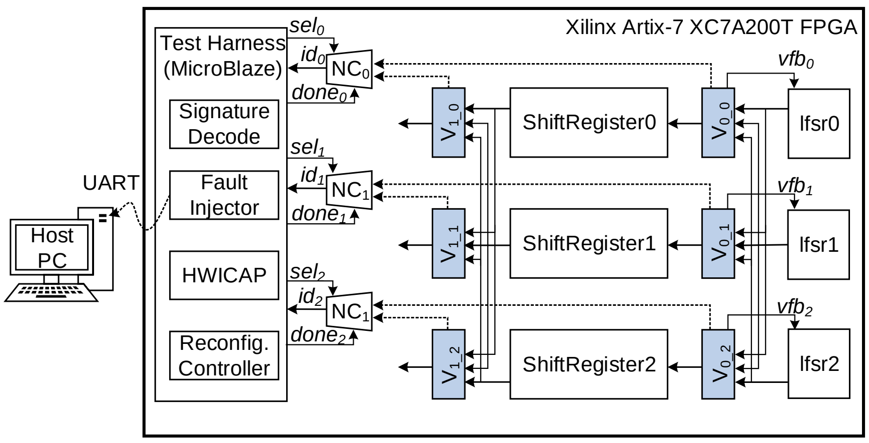


Figure 5.5: Experimental Setup

operations mapped into the look-up tables (LUTs) of each stage. In our experimental setup, the SR module processed the data generated by the LFSR module.

The LFSR modules, their voters and module outputs comprise C_0 of the design. As there is feedback from the LFSR voter back to the LFSR input, the LFSR voter also behaves as an upstream voter with respect to the LFSR component. The SR modules, its voters, and their connection to the RCN form C_1 of the Design Under Test (DUT).

We floorplanned the test circuit according to the guidelines provided in Section 5.2.3. The DUT contained 26 sub-components in total. These included three identical LFSR modules ($M_{0\_0,1,2}$) and three SR modules ($M_{1\_0,1,2}$), six voters ($V_{0,1\_0,1,2}$), triplicated NCs ($\mathrm{NC}_{0,1,2}$), two sets of nets connecting the modules to their voters ($mout_{0,1}$), three voter output signals ($vout_{0\_0,1,2}$), and the dedicated error signal for each voter ($e_{0,1\_0,1,2}$). The voter output of C_1 was not evaluated because it did not connect to any downstream logic in this design. The frame sets for these sub-components were extracted using the method described in Section 5.2.2.

Test Harness

The test harness implemented a fault injector, an error signature decoder, and an RC program running FDPR of Section 5.2.4.

Since the full bitstream was stored in an SPI flash memory from which the device booted, we implemented a flash controller to access this memory. The controller allowed a maximum read throughput of 25 Mbyte/s through quad SPI protocol running at 50MHz. We

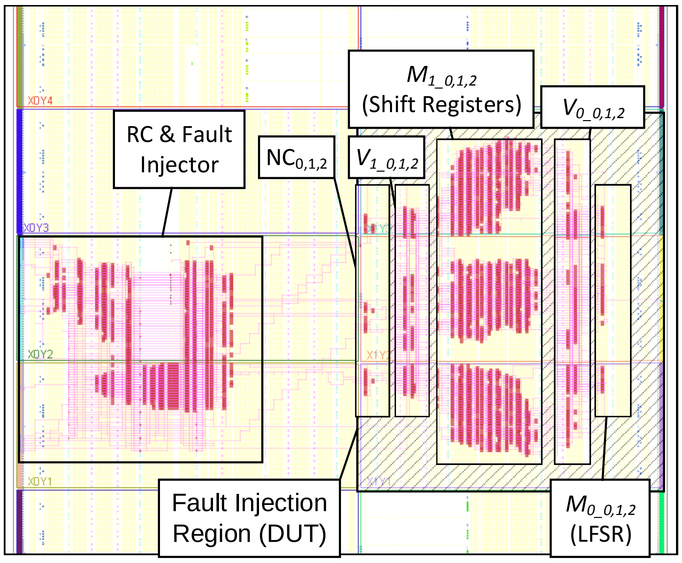


Figure 5.6: Floorplanning of the test harness, and test circuit in three clock regions on the right-hand side of the device

stored the byte offset of the first frame of each major column in an array. The total program memory needed for the array was 530 entries, corresponding to the 5×106 major columns available in the device. The list of the major columns used by the sub-components was also stored in program memory, the size of which depended on the number of major columns they used. When the reconfiguration of a sub-component was triggered, the data for the frames of the corresponding major columns were fetched from the flash memory and transferred to the ICAP with the required frame headers using a DMA engine.

We measured the reconfiguration latency of different sub-components and used the same hardware setup to measure the latency of a complete blind scrub of the device. The frames to be scrubbed were also extracted from the full bitstream as described in Section 5.2.4.

Figure 5.6 shows the layout of the test system on the device. The test circuit was implemented in the three shaded clock regions depicted on the right side of the device — we injected faults (configuration bit flips) into this region using dynamic partial reconfiguration. The MicroBlaze was implemented in the two central clock regions on the left side of the device. This region was not subjected to fault injection. In total, we injected 16,134,144 faults into 4,992 frames of the DUT, thereby injecting a fault into every configuration bit of the DUT.

After a fault was injected, the RC was programmed to wait for 1 µs in order to let the error emerge. Only then did it check the output of the RCNs. For this DUT, a bit was deemed

Table 5.2: Block Sub-component Area & Recovery Time

Sub-component	**LUTs**	**Flops**	**Major Columns**	**Frames**	**Recovery Time (ms)**
$M_{0\_0,1,2}$	8	64	2	64	2.0
$V_{0,1\_0,1,2}$	153	66	2	72	2.3
$M_{1\_0,1,2}$	2,548	512	20	712	22.6
$\mathrm{NC}_{0,1,2}$	43	34	2	72	2.3

not to be sensitive if the RCN did not report an error. Otherwise, the RC reported the sensitive bit location and the error signature to the host. Thereafter, the faulty bit was corrected by reversing the injected bit flip. Between each injection cycle, the RC waited another 1 μs to allow the correct data to flush through the test circuit.

The host PC used the bit locations reported as causing errors to determine which sub-component of the design was affected and, together with the reported error signature, determined which sub-components would have been reconfigured to clear the error according to the repair strategy described in Section 5.1.3. The total number of frames that would consequently have been reconfigured and the reconfiguration latency that would have been incurred to recover the observed errors were logged against each error report. Note, therefore, that for this experiment the repair strategy was not actually running on the board. Instead, the host simulated the strategy based on the error location and signatures contained in the error reports it received and assuming that the error would have persisted until the sub-component containing the error, as identified by the error location, would have been reconfigured.

5.3.2 Results

Fine-grained Dynamic Reconfiguration of Subcomponents

Tables 5.2 & 5.3 report the utilization, number of major columns and number of configuration frames for each of the sub-components in the design, as well as the reconfiguration times that we measured using the proposed fine-grained dynamic bitstream composition method. Using this method, on our platform we found that we could sustain a transfer rate of one frame every 32 us. This is a reasonably good result considering the constraints imposed by the board architecture (SPI flash) and the use of Microblaze and AXI-HWICAP.

Table 5.3: Net Sub-component Area & Recovery Time

Sub-component	**PIPs**	**Switch Matrices**	**Major Columns**	**Frames**	**Recovery Time (ms)**
$mout_0$	10,690	632	35	1,188	39.7
$mout_1$	10,358	637	50	1,712	55.8
$vout_{0\_0}$	3,077	161	12	416	13.7
$vout_{0\_1}$	3,050	143	13	452	14.8
$vout_{0\_2}$	3,131	172	12	416	13.7
e_{0_0}	41	11	11	380	12.3
e_{0_1}	41	18	18	624	20.3
e_{0_2}	41	19	16	552	18.0
e_{1_0}	22	7	4	136	4.6
e_{1_1}	23	7	5	172	5.7
e_{1_2}	22	4	3	100	3.4

While the LFSR modules are relatively small, since they are synchronous, we need to reconfigure the neighboring major columns to recover from any errors affecting their clock buffers, as explained in Section 5.2.2. In our design, the outputs of the LFSR voters were registered to separate the feedback path from the module output. For efficiency's sake, we included the feedback path in the $mout_0$ sub-component.

On our platform, we found that to perform a blind scrub of the device we had to overwrite 18,300 frames in total. The latency for a scrub cycle was 432 ms, which corresponds to a sustained transfer rate of one frame every 24 us. This performance is limited not just by the board and circuit architecture, but also by the need in our test to retrieve each frame individually from the indexed complete bitstream. It should be noted that the use of a purpose-designed scrubber could be expected to use considerably less than 432 ms to scrub the device.

On the tested circuit, the worst-case repair time using our proposed repair strategy was 135 ms, which involved the reconfiguration of one of the SR modules, its upstream voter and voter output, the nets between the SR modules and their voters ($mout_1$), and those of the LFSR modules and their voters ($mout_0$ of the upstream component). This maximum repair time is approximately 1/3 of the scrub cycle latency, which represents a substantial reduction in the repair time. At most, we found we had to reconfigure just over 4,136 frames, which is less than 1/4 of those needed to perform a scrub.

Table 5.4: Fault Injection Result

Sig	**Number of Checks (Chk)**						**Sub-component Reconfiguration Sequence**
	1	2	3	4	5	6	
i_0	4,882	959	N/A	627	N/A	0	$M_{0\_0,1,2} \rightarrow V_{0\_0,1,2} \rightarrow mout_0$
ii_0	15,862	6,623	159	44	0		$V_{0\_0,1,2} \rightarrow mout_0 \rightarrow \mathrm{NC}_{0,1,2} \rightarrow e_{0\_0,1,2}$
iii_0	3,656	0					$mout_0$
i_1	633,901	4,788	2,038	748	23	0	$M_{1\_0,1,2} \rightarrow V_{0\_0,1,2} \rightarrow vout_{0\_0,1,2} \rightarrow mout_1 \rightarrow mout_0$
ii_1	18,248	7,282	115	0	0		$V_{1\_0,1,2} \rightarrow mout_1 \rightarrow \mathrm{NC}_{0,1,2}$
iii_1	4,364	0					$mout_1$

Fault Injection Results

Table 5.4 reports on the exhaustive fault injection experiment. The subscripts indicate the signatures for the LFSR and SR components (0 & 1) respectively. The table reports for each error signature (i_0 - iii_1) how many reconfigurations of the sub-components were triggered (Number of Checks) according to our repair strategy. The sub-components reconfigured for each error signature and their priority, as proposed by the repair sequence, is also listed.

As can be seen from the table, a complete scrub of the device was not required to recover any emulated SEU during our fault injection experiment. We therefore conclude that the repair strategy is effective at quickly recovering from single errors within the test circuit, and that in the worst case, the strategy is able to recover from errors substantially faster and using considerably less energy than a scrub cycle.

In total, 680,931 errors were reported for 16,134,144 fault injections. For error signature (i), the number of reports for the LFSR and SR modules differ greatly because their utilizations differ greatly. For error signatures (ii) & (iii), both components show the same trend on the number of checks per signature. It is clear that voters and the interconnecting nets between voters and modules are much more prone to errors than the RCNs. Since the LFSR component is further away from the NC, its routing net utilization is greater than for the SR component, and thus, its RCN nets present more errors. However, since the reconfiguration frames for these sub-components are different, the average number of frames recovered for these sub-components differ.

Table 5.5: Frames, MTTR and Energy Result

	MER/FDPR	MER/Scrub	Triggered Scrub
Frames	695	1,950	18,300
MTTR (ms)	22	36	216
Energy (mJ)	0.37	1.04	9.79

5.3.3 Frames, MTTR & Energy

Table 5.5 compares the average number of frames reconfigured per error, the average recovery time and the energy expended to repair the error assuming each frame write consumes 535 nJ [53]. The proposed fine-grained DPR approach to MER is listed as MER/FDPR and its performance is compared with, on the one hand, a more conventional approach to MER in which errors that occur outside the reconfigurable modules are recovered by scrubbing the device (MER/Scrub) [12], and on the other, by triggering a scrub whenever any error is detected in the system (Triggered Scrub). In each case, the test system is used to trigger reconfigurations or scrubs as outlined in Section 5.1. While triggered scrubbing needs to scrub 18,300 frames, MER/FDPR only needs to reconfigure 695 frames on average. While these results are application and device dependent, they are representative of the gains that are possible. In our case, the MTTR for MER/FDPR was only 10% of that for scrubbing. MER/FDPR is also the most energy efficient approach of those we have studied. As energy consumption is assumed to be proportional to the number of frames that are rewritten, in our study we found the fine-grained DPR approach was 2.8× more efficient than MER/Scrub and 26× more efficient than triggered scrubbing. A periodic scrubbing approach is likely to expend even more energy or compromise MTTR if the period between scrubs is reduced.

5.3.4 Reliability

Table 5.6 summarizes the critical bits we found in the test circuit via the fault-injection experiment and the average recovery time for these sub-components using MER/FDPR, MER/Scrub and Triggered Scrub. The average recovery times for sub-components using MER/FDPR and MER/Scrub were derived from Table 5.4, which was used to calculate the number of times an error was found in each sub-component, and to identify which sub-components were reconfigured according to the reconfiguration sequence. The average

Table 5.6: Sub-component Critical Bits Summary Using Proposed Recovery Sequence

Sub-component	**Critical Bits Found**	**Avg Recovery Latency (ms)**		
		MER/ FDPR	**MER/ Scrub**	**Triggered Scrub**
$M_{0\_0,1,2}$	1,307	2.0	2.0	216.0
$mout_{0\_0,1,2}$	3,590	41.5	216.2	216.0
$V_{0\_0,1,2}$	4,107	7.4	221.1	216.0
$vout_{0\_0,1,2}$	430	38.9	238.6	216.0
$e_{0\_0,1,2}$	15	48.3	216.0	216.0
$M_{1\_0,1,2}$	211,300	22.6	22.6	216.0
$mout_{1\_0,1,2}$	4,058	59.9	216.8	216.0
$V_{1\_0,1,2}$	3,655	2.3	216.0	216.0
$\mathrm{NC}_{0,1,2}$	77	52.4	216.0	216.0

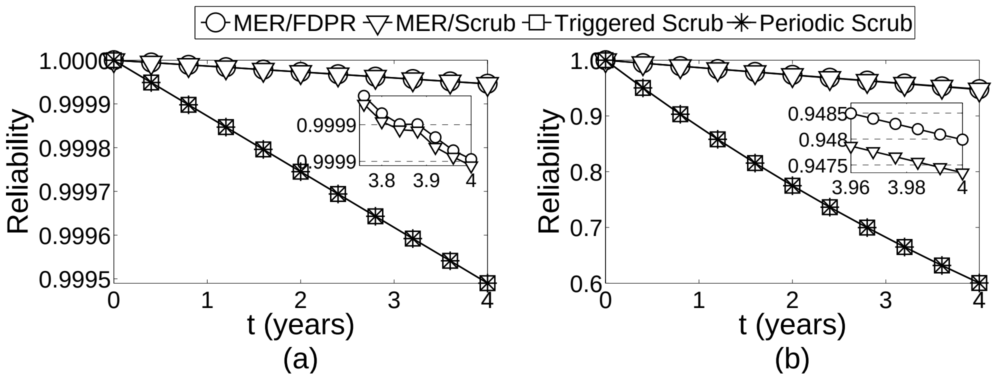


Figure 5.7: Reliability Results: (a) $\lambda_{bit} = 8.41E-12$ (LEO), (b) $\lambda_{bit} = 2.66E-10$ (GEO)

recovery time was used as input to a Markov reliability model derived from [12,61]. Figure 5.7 plots the resulting reliability of the test circuit using the proposed MER/FDPR, the more conventional MER/Scrub, Triggered Scrub and periodic scrubbing (Periodic Scrub) in LEO and GEO over a 4-year mission. Furthermore, we have zoomed in to certain parts of the reliability plots to show how the MER/FDPR performance compares with MER/Scrub. Note that the Periodic Scrub plot needs the same recovery time as Triggered Scrub, but that it excludes the RCN sub-components ($\mathrm{NC}_{0,1,2}$ and $e_{0\_0,1,2}$), as these are not needed for this approach. In our analysis, we have assumed high bit failure rates using the peak 5-min CREME96 model [42] with 2.54 mm aluminum shielding for these two orbits. Thus, in LEO, we assume 8.41E-12 upset/bit·s, while in GEO, this figure is 2.66E-10 upset/bit·s. We find that both MER/FDPR and MER/Scrub approaches can be expected to be more reliable than scrubbing because of the significant reduction in the MTTR of critical system sub-components.

5.4 Discussion

In this chapter, we have proposed a fine-grained module-based error detection and dynamic reconfiguration scheme that can detect and recover any error presented in TMR-based SoCs without additional hardware or resorting to scrubbing. We evaluated our method by injecting faults into a typical test circuit. The result shows that with the proposed method, we can achieve a slight improvement in reliability over the more conventional module-based configuration memory error recovery scheme that triggers a scrub cycle when an error repeats or the triplicated reconfiguration control networks disagree [12], but it is also much more energy efficient — in our case we found an improvement of almost two-thirds in the energy consumption. The advantages of the proposed approach in terms of energy efficiency and reliability over triggered and periodic scrubbing are even greater.

As a future investigation on the energy consumption and reliability of this approach, it would be best to implement an actual satellite payload based on the fine-grained approach and to expose the final design to heavy ion bombardment or neutron flux to observe the behavior of the payload and its self-healing ability. The fault injection experiment could also be extended to be able to inject accumulated bit-flips in a random way in order to simulate the particle flux of a given environment.

Chapter 6

Conclusion

This chapter concludes this thesis. It starts with concluding remarks on the contributions of each chapter, in which we highlight the main findings of each part. Then, we propose two possible directions for further enhancing our investigation in Chapters 3 & 4, which can also be automated as part of the TMR-MER design flow. The study of Chapter 5 motivates vendor support for the proposed fine-grained dynamic reconfiguration approach so as to improve the performance and flexibility of the TMR-MER technique.

6.1 Concluding Remarks

Reliable space-borne digital systems implemented using COTS FPGAs and programmable SoCs require rapid, low-energy, flexible and reliable SEU mitigation approaches. In this thesis, we focused on the deployment of Triple Modular Redundancy (TMR) for error detection and tolerance and Module-based Error Recovery (MER) for configuration memory error recovery. Use of MER calls for a fast, resilient, energy- and area-efficient Reconfiguration Control Network (RCN) to aggregate recovery requests arising from any of the voters in the TMR-MER system. Thus, in this thesis, the main objective has been to discover the best RCN solution among all possible RCN architectures.

Chapter 2 summarized the motivation and the research context for deploying COTS FPGAs in space and explained the radiation challenges which should be carefully considered by system designers. It then provided an overview of existing SEU mitigation approaches - TMR-MER and Triple Modular Redundancy with configuration memory scrubbing (TMR-S) and explained the advantages of TMR-MER over the classic TMR-S, which are that it is faster, more energy-efficient and responsive for correcting errors in the user circuit. The conclusion of this chapter is that the RCN plays a very important role in TMR-MER systems because it has a direct influence on the recovery delay and the recovery precision of the system.

In Chapter 3, we implemented and evaluated an existing token ring based RCN architecture [13], and we found that although this architecture is highly flexible and generic, it suffers from poor performance and high resource utilization due to its high design complexity. These shortcomings can, in turn, compromise recovery time and system reliability. Thus, with a view to reducing the area overhead and improving the circuit speed, we have explored alternative designs for the token ring network that fulfill the purpose of connecting the voters with the RC in a daisy-chain manner, for which we obtained about a 200-fold speedup and 90% area reduction.

Since the RCN can be implemented in various topologies, in Chapter 4, we surveyed four possible RCN architectures that have been reported for TMR-MER systems described in the literature, and proposed four RCN architectures for experimental evaluation. By implementing these networks in a typical checkerboard-like synthetic layout and on the RUSH satellite payload [16], and by injecting faults to identify the sensitive bits, we concluded that both the ICAP-based hard network that reads voter status via in-built FPGA configuration hardware and the star-type RCN, that comprises dedicated point-to-point connections for each of the voters in the system, are the best candidates as they have fewer sensitive bits, low demand for hardware resources and are easier to interpret when errors occur. Both approaches scale well as the size of the network increases. However, the ICAP-based approach has a noticeably greater latency (on the order of micro-seconds), while star networks only need a few clock cycles (depending upon the size of the network) to report errors. In [12], it was found that the reliability of TMR-MER was significantly less than that of TMR-S unless the RCN was also triplicated and recovered. In [12], we examined using a complete scrub to repair the RCN when errors occurred in that component.

In Chapter 5, we examined and studied a triplicated star-type RCN in a fully-triplicated TMR-MER system, and proposed a fine-grained module-based error detection and correction framework that is able to localize errors more precisely than has previously been demonstrated in the literature. We also studied the prioritized reconfiguration of sub-components in response to the error signature provided by the triplicated RCNs for efficiency's sake. This approach comprises a repair strategy and a repair order which can narrow down the actual incorrect sub-component causing a particular error signature. The method dynamically reconfigures the suspected sub-component, which can be a module, a set of routing nets or even an RCN component using a fine-grained reconfiguration method. The evaluation of this approach was performed by injecting faults into typical processing circuitry that is triplicated using the proposed approach, and collecting the error signatures caused by these bitflips. From the results, we were able to determine the performance, energy consumption, and reliability of the proposed method, which in the case of the studied circuit lead to a 65% energy saving, a 29% reduction on the overall mean-time-to-repair and a slight improvement in system reliability over conventional TMR-MER systems that resort to a scrub of the whole device when a bitflip is detected in the RCNs. These improvements can be even greater compared to the more conventional periodic scrub and triggered scrub approaches.

This thesis compares various types of RCN that are used by the TMR-MER systems in the literature and demonstrates how a way of triplicating the RCN can assist in localizing and recovering the errors in the system. The comparison between different RCNs began with a comprehensive study of a token ring network dedicated to controlling the reconfiguration of TMR modules. By optimizing away unnecessary features and design concerns, we redesigned four interconnection networks, namely, a star network, a bus, a token ring network and an ICAP-based approach for area, performance and reliability comparisons. From the result of these experiments, we concluded that the ICAP-based voter check and star-based networks are the ideal choices for TMR-MER systems despite the long communication delay of the ICAP-based approach and relatively higher resource utilization and upset sensitivity of the star network. From [12], we found that TMR-MER systems with triplicated RCNs can have a very high reliability over TMR-S systems. Since the traditional TMR-MER does not cover the error recovery of supporting logics such as RCNs, interconnection routes and, in some cases, voters, which makes the TMR-MER technique not as robust as TMR-S, in this thesis, we proposed a fine-grained MER scheme that greatly improves the recovery precision, and substantially reduces the energy and repair delay over traditional TMR-MER systems, which have to resort to scrubbing when unrecoverable errors are detected.

6.2 Future Work

The study on comparing different RCNs demonstrated in this thesis has provided a comprehensive evaluation of four typical RCNs — star network, bus, token ring network and the ICAP-based approach that were used in the literature. To our knowledge, some other forms of interconnection method such as binary-tree, fat trees or many Network-on-Chip topologies as listed in [62] are not evaluated, but they all have the potential to be scalable, fast, low-cost and even fault-tolerant in providing reconfiguration management for voters. Therefore, one of the further directions is to compare these possible interconnection methods with the four existing networks.

Another direction is to propose a flexible and automatic RCN insertion and mapping flow. This approach should be finalized as an extension of CAD tools for critical systems that can identify and connect voter status signals with the Reconfiguration Controller (RC). Such enhancements should be incorporated as a part of a TMR-MER design flow, where given a user circuit, the dataflow of the circuitry needs to be first analyzed before being partitioned with a proper voter insertion algorithm that avoids internal cycles, and adding an RCN to interconnect these voters with a ready-to-use RC component. The flow should also fulfill the role of mapping module-IDs to all of the TMR modules and sub-components which are exported to the flow for generating the internal RC look-up-table to partial bitstreams stored in external memory.

Using current memory media and interface techniques, the performance of module-based configuration memory error recovery implemented in this thesis is limited to a throughput of 10 – 25 MB/s, which is far lower than the theoretical maximum throughput of ICAP (400 MB/s). Furthermore, based on this data rate, each scrub cycle may take a few hundred milliseconds, which is enough for SEM-based readback scrubbing to scrub the whole device more than 10 times. Even when given a digital system for space implemented using DDR3 memory for bitstream storage, the peak data rate is still only half of that of the ICAP, and the memory controller needed is more complex, and therefore more vulnerable to SEUs. If an SEU affects the bitstream traffic, the whole system may be affected if corrupted data is written into the configuration memory. In order to achieve high recovery performance and to reduce the design complexity of bitstream traffic, the MER technique could also incorporate use of the built-in FrameECC primitive [52] to read and check the frameset of TMR sub-components extracted using the method proposed in Section 5.2.2, and whenever an incorrect frame is detected, the single bit error within a frame could be corrected as instructed by the FrameECC, while the other errors could be fixed by overwriting the incorrect frames based on the full bitstream index methodology proposed in Section 5.2.4.

As discussed in Section 5.2, due to the limitation of the vendor's partial reconfiguration flow, TMR-MER cannot be used to recover all of the components in the system. Thus, FPGA vendors could provide similar CAD tool support as we have proposed for the fine-grained reconfiguration of TMR-MER systems that can identify the configuration bits for arbitrary system sub-components and can also provide a method for retrieving arbitrary frame data from the full bitstream. Furthermore, the TMR-MER technique can be made more efficient and precise if the configuration memory addressing can be more fine-grained. Rather than accessing the memory through frames, which only contain a slice of configuration bits for a column of logic resources, as an ideal, these frames needs to be partitioned and addressed for different logic cells such as CLBs, while the dynamic reconfiguration can reconfigure a CLB in one operation without the need for writing a set of frames pertaining to the whole CLB column. The fine-grained reconfiguration method could then identify which set of CLBs pertain to a system component, and when a fault manifests in this component, the RC would only need to reconfigure the relevant set of CLBs.

Appendix A

A.1 7-Series Device Model

Xilinx FPGAs are tiled into major columns [22, 51, 63] each of which is indexed by a row and column address. A major column contains a column of programmable resources, of which there are several in a device. It also includes a column of switch matrices that provides access to the general routing matrix.

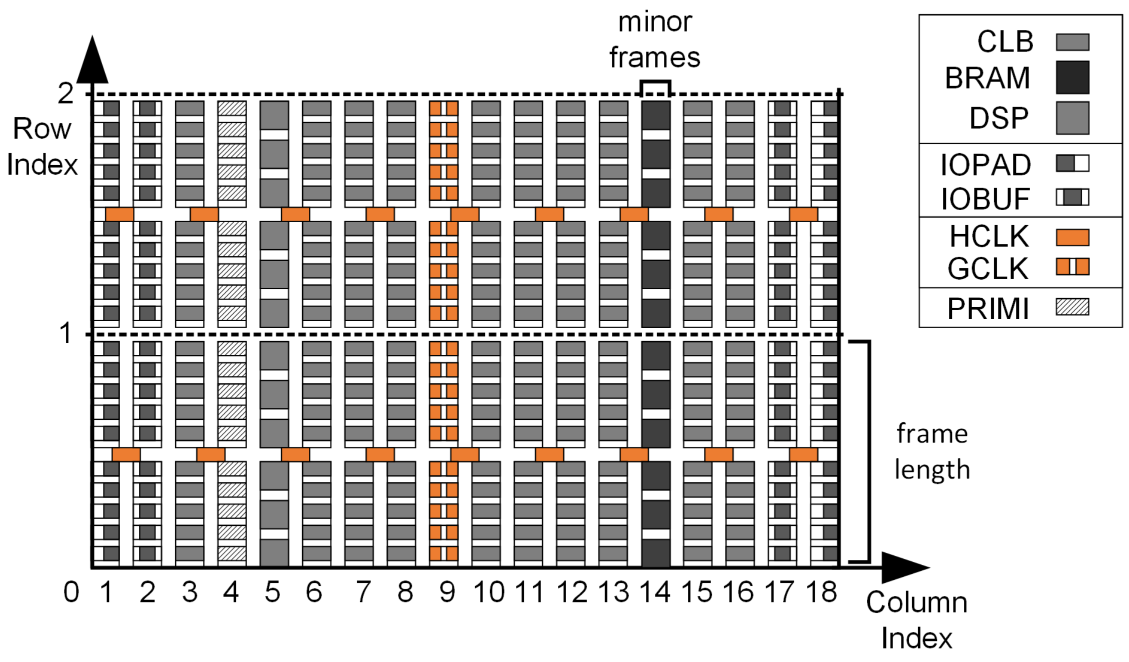


Figure A.1: Device Model

Figure A.1 illustrates the major columns present in Xilinx 7-Series devices. As shown in the figure, 7-Series devices comprise major columns for CLB, DSP, BRAM, and IO pads that contains versatile IO ports, IO buffers featured in high-speed memory interfaces such as SPI, DDR, etc., and special columns for user primitives such as ICAP, BSCAN, etc.. The IOBUF column also contains PLL primitives used for generating clocks. The dashed lines in the figure indicate the horizontal boundaries between configuration rows. The middle columns contains the backbone of the clock regions (GCLK) dividing each row of major columns into two clock regions.

Clock signals are propagated from the central column of the device to each of the local clock buffers in a clock region via a dedicated clock network. The GCLK column buffers and routes the clock signals to different clock regions. At the middle of each major column, a local clock buffer (HCLK) is inserted to relay the desired clock signals to the major column. Every pair of major columns shares a local clock buffer.

A.2 Frame Addressing

A major column is configured via a contiguous set of configuration frames, each of which forms a bit slice of the configuration memory for the major column. In 7-Series devices, a configuration frame has 101 words, the middle word of which has configuration bits for HCLK and frame ECC. The number of the contiguous frames used to configure a major column depends upon the type of resource it contains. CLB tiles stretch over 36 configuration frames and DSP tiles use 28 frames. BRAM tiles have two sets of frames: one for configuring the BRAM and its closest switch matrix that comprises 28 frames, and another 128 frames that are used to store the BRAM contents. The GCLK, IOPAD, and PRIMI columns are configured via 30 frames, while IOBUFs use 42 frames.

A.3 Frame Address Conversion

The row index of a major column increases from the bottom-most clock region to the top-most one. A column index increments from the left-most major column to the right-most one within a row. The frame address of the major column can be ascertained according to the description of the frame address register (Table 5-24 in the vendors configuration user guide [51]). A frame address is derived from five sub-addresses. These are the Block Type, a Top/Bottom Bit, the Row Address, the Column Address and a Minor Address. Configuration frames for major columns have a Block Type of zero. Apart from this, BRAM content frames have a Block Type of one. The Top/Bottom Bit and Row Address are related to the row index of the major column. The bit select distinguishes between rows in the top half of the device (0) and those in the bottom half (1). The Row Address is the relative distance of the row from the horizontal centre of the device. It increments as the row index of the major column increases when the top half of the device is selected, or decreases when the major column is in the bottom half of the device. The Column Address is just the column index of the desired major column. The Minor Address indexes into the contiguous set of frames for a major column. Its range depends upon the type of programmable resource.

A.4 Special Major Column Frames

Derived from the Virtex-5 family, 7-Series devices also have special frames which contain configuration bits used for dynamic partial reconfiguration [51, 63]. These bits gate the global dynamic reconfiguration commands such as GCAPTURE, which is used for reading configuration status, GRESTORE that forces the user flip-flops to load the state stored in its configuration memory cell, the SHUTDOWN command, which deactivates the write operation of user flip-flops and RAMs and sets all interconnects to High-Z state, and the STARTUP sequence that activates the device. They are seated in the middle word of a special frame that has a Block Type of 2 and Minor Address of 0. As every major column has a special frame, the Top/Bottom Bit, Row and Column Address are the same as the major column it is in charge of. When these bits are set to high, the influence of these dynamic reconfiguration commands will not affect the programmable resources within that major column.

In the partial reconfiguration flow [22], the partial bitstreams for normal partial reconfiguration modules will not involve special frames, but when the reconfigurable module enables the reset-after-reconfiguration feature, the configuration memory of all available special frames of the device will be rewritten before the configuration data of the logic is loaded. All the other major columns beside the major columns that implement the reconfigurable module will be protected via asserting the dynamic reconfiguration bits. Thus, the SHUTDOWN, GRESTORE and STARTUP sequences, that implement the reset-after-reconfiguration feature, will only affect the dynamic major columns.

Through experimentation we have found that in advanced dynamic reconfiguration of 7-Series devices, to mask a single major column, the special frame should be rewritten with a pad frame with the word "0xE00009BC" in the middle. The 3 msbs are the reconfiguration configuration bits, while the 12 lsbs are the corresponding frame ECC.

A.5 Artix-7 XC7A200T Device Parameters

A XC7A200T device contains 18,300 frames arranged in a 5×106 major column matrix. Major columns are arranged as follow:

- GCLK major columns are placed at column index 55.
- Column indices of 0 and 105 are IOPADs, each of which indexes 52 IO ports, while column indices 1 and 104 are for IOBUFs.
- The device has 45 DSP tiles and 45 BRAM tiles, which are arranged in 9 columns of major columns. The DSP tiles are at column indexes 9, 14, 31, 43, 48, 61, 66, 91 and 96, while the BRAM tiles are at indexes 6, 17, 28, 40, 51, 58, 69, 88 and 99.

- User primitives are tiled in the major columns with column index 24. Although not all of these major columns contain the same primitives, they are all configured using the same number of frames.
- The remaining major columns are CLB columns.

Appendix B

B.1 Overview of Internal Configuration Facilities

The configuration memory content of Xilinx FPGAs can be accessed via a special configuration port which processes configuration packets, and the bitstream commands are executed by reading from or writing to the configuration registers [51, 63]. Users can communicate with the configuration port via ICAP, SelectMap, or JTAG. Through our experiments, we have derived the internal architecture of the configuration port as shown in Figure B.1. At the frontend of the port, packet decoder logic is used to distinguish between Type-1 packets, Type-2 packets, and configuration data. The CRC circuit is attached to the packet decoder to automatically calculate the CRC value of the data entering and leaving the port. When the CRC register is written, the user CRC value will be compared with the value calculated by the CRC circuit, and if they disagree, then a CRC error flag is asserted so as to prevent the device startup or shutdown operations. Bitstream and Dynamic Partial Reconfiguration commands are executed through writing command codes into the command register. Valid commands are WCFG and RCFG that directly select between the frame data registers (Frame Data Register Input (FDRI) and Output (FDRO)), GRESTORE and GCAPTURE that are the global flip-flop control signals, and START and SHUTDOWN that perform the device startup and shutdown operations. The value of the CRC circuit can also be cleared when the Clear CRC code is written into the CMD register.

Before reading or writing configuration frames into the configuration memory space, the user needs to first load the frame address into the FAR register, issue a WCFG/RCFG command that controls the configuration data direction, and lastly specify the number of words of the frame data by writing the value to FDRI or FDRO. Then, the following write/read operations are directly bypassed to the internal configuration memory network. Each row of major columns has a frame buffer, which in the Xilinx 7-Series contains 101 words. The frame data is first loaded into those individual buffers. In order to write/read a frame into/from the configuration memory, the user needs to write a pad frame to push the frame data into its destination or skip a pad frame before reading the actual frame data. A pad frame contains 102 dummy words, which comprise a 101-word dummy frame and an extra dummy word to flush the word buffer inside the FDRI/FDRO block [2]. The addressing of these frame buffers and configuration frames depends upon the FAR

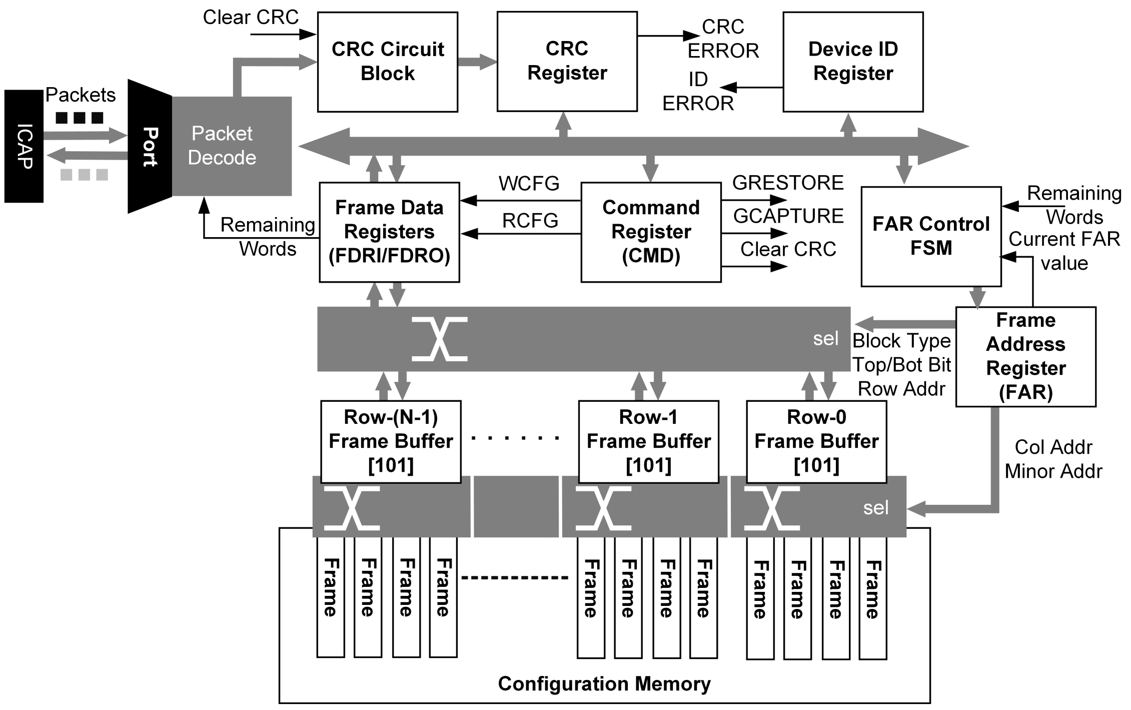


Figure B.1: Overview of Configuration Facilities

value, where the upper addresses (Block-Type, Top/Bot bit, and Row address) select the frame buffer, while the lower addresses (the Column address and Minor address) direct the frame data stored in the frame buffer to the desired configuration memory latches. Due to the architecture of the switch network, the frames with the same upper addresses can be transferred in pipeline mode, in which the time to transfer a frame can be reduced by a half because the pad frame only needs to be issued once. Branching between different frame buffers involves reading or writing a pad frame. To further maximize the throughput of the configuration port, the FAR FSM that automatically increments the FAR register reduces the need to change the FAR register for each frame. The FSM can automatically skip invalid frame addresses, and at the end of the row, the FAR will be delayed in order to write or read a pad frame.

B.2 Full Bitstream Composition

A full bitstream, typically stored in flash memory, is automatically loaded when the device boots. The full bitstream contains all of the necessary commands and all of the configuration frame data. There are two types of full bitstreams that can be generated by the vendor's design toolkit, namely, a normal full bitstream and a bitstream with the single frame CRC check feature included. The sequences of both are shown in Figure B.2. The normal full bitstream utilizes the FAR FSM to control the FAR register, while the later

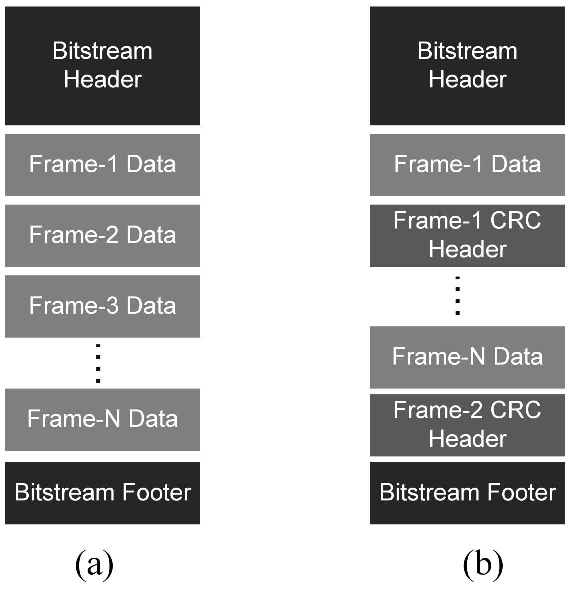


Figure B.2: Composition of bitstream formats: (a) Normal full bitstream (b) Bitstream with single frame CRC check feature

manually specifies the FAR value. Therefore, as explained in Section 5.2.4, bitstreams with the single frame CRC check feature included are ideally suited to being indexed so that the individual frame data can be readily accessed for repairing SEUs in the configuration memory. The bitstream comprises frame CRC check headers after the frame data. The frame headers include commands to load the FAR register sequence, then check the CRC register sequence, and to write the FDRI for the next frame. The descriptions of the bitstream header and footer can be found in the corresponding device Configuration User Guide [51], while the composition of the frame header is as follows:

Configuration Words (Hex)	Descriptions	Section
`0x30002001`	Write FAR	
`0x????????`	Frame-n FAR	
`0x30000001`	Write CRC Reg	Frame-n Header
`0x????????`	CRC value	
`0x30004065`	Write FDRI (word count: 101)	

B.3 Advanced Dynamic Reconfiguration Operations

Customized configuration memory scrubbing applications, or fine-grained dynamic partial reconfiguration, as proposed in this thesis, requires advanced dynamic reconfiguration operations that are capable of reading or writing arbitrary frame data from/to the configuration memory at a high bandwidth. These techniques involve reading/writing a single frame and reading/writing a series of frames.

B.3.1 Single Frame Write/Read

The following is the bitstream sequence for a single frame write, where the obtained frame data should be put into the frame data section.

Configuration Words (Hex)	Description	Section
`0xFFFFFFFF`		Bitstream Header
`0xAA995566`	Sync Word	
`0x20000000`	Noop	
`0x20000000`	..	
`0x30008001`	Write CMD	
`0x00000007`	Clear CRC	
`0x20000000`	Noop	
`0x20000000`	..	
`0x30018001`	Confirm ID	
`0x????????`	Device ID	
`0x30002001`	Write FAR	
`0x????????`	Frame-1 FAR value	
`0x30008001`	Write CMD	
`0x00000001`	WCFG	
`0x300040??`	Write FDRI (word count: 101 $\times$ 2)	
`0x????????`	Frame Word-1	Frame Data
`....`	....	
`0x????????`	Frame Word-101	
`0x00000000`	Frame Word-1	Pad Frame
`....`	....	
`0x00000000`	Frame Word-101	
`0x00000000`	Extra Dummy Word	

The bitstream sequence for single frame read is as follows:

Configuration Words (Hex)	Description	Section
`0xFFFFFFFF`		Bitstream Header
`0xAA995566`	Sync Word	
`0x20000000`	Noop	
`0x20000000`	..	
`0x30008001`	Write CMD	
`0x00000007`	Clear CRC	
`0x20000000`	Noop	
`0x20000000`	..	
`0x30018001`	Confirm ID	
`0x????????`	Device ID	
`0x30002001`	Write FAR	
`0x????????`	Frame-1 FAR value	
`0x30008001`	Write CMD	

`0x00000004`	RCFG	
`0x280060??`	Write FDRO (word count: 101 × 2)	
`0x00000000`	Extra Dummy Word	Pad Frame
`0x00000000`	Frame Word-1	
....	....	
`0x00000000`	Frame Word-101	
`0x????????`	Frame Word-1	Frame Data
....	....	
`0x????????`	Frame Word-101	

B.3.2 Serial Frame Write/Read

To write/read a number of frames such as all of the frames in a major column, the performance of this dynamic reconfiguration operation can be improved by using the FAR FSM to increment the FAR pointer instead of manually loading it, which saves at least 3 cycles per frame. The bitstream format of this operation is the same as for single frame operations except that the FDRI/FDRO should be loaded with a word count of 101 × (n+1) where n is the number of frames to be transferred and the frames should be arranged from lowest to highest address. However, due to the limit of the Type-1 packet that can only hold word count less than 2048, the load FDRO/FDRI can be altered using a combination of Type-1 and Type-2 packets that are `0x30004000 0x50??????` for writing a sequence of frames, and `0x28006000 0x48???????` for reading the sequence.

To write/read a series of frames that are not adjacent, we suggest writing frames by one with frame headers. The optimal frame header format is as follows:

Configuration Words (Hex)	Descriptions	Section
`0x30002001`	Write FAR	Frame-n Header
`0x????????`	Frame-n FAR	
`0x30004065/0x28004065`	Write FDRI/FDRO (word count: 101)	

When the next FAR is in another row, a pad frame should be written or read before loading the next FAR and its frame data.

References

[1] C. Carmichael, “Triple module redundancy design techniques for Virtex FPGAs,” *Xilinx Application Note XAPP197*, vol. 1, 2001.

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

[3] E. Cetin, O. Diessel, L. Gong, and V. Lai, “Towards bounded error recovery time in FPGA-based TMR circuits using dynamic partial reconfiguration,” in *2013 23rd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2013, pp. 1–4.

[4] 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.

[5] S. Tanoue, T. Ishida, Y. Ichinomiya, M. Amagasaki, M. Kuga, and T. Sueyoshi, “A novel states recovery technique for the TMR softcore processor,” in *2009 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2009, pp. 543–546.

[6] C. Bolchini, A. Miele, and M. D. Santambrogio, “TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs,” in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*. IEEE, 2007, pp. 87–95.

[7] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, “Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, p. 21, 2012.

[8] C. Pilotto, J. R. Azambuja, and F. L. Kastensmidt, “Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications,” in *Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design (SBCCI)*. ACM, 2008, pp. 199–204.

[9] B. Navas, J. Öberg, and I. Sander, “The upset-fault-observer: A concept for self-healing adaptive fault tolerance,” in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2014, pp. 89–96.

[10] S. Yousuf, A. Jacobs, and A. Gordon-Ross, "Partially reconfigurable system-on-chips for adaptive fault tolerance," in *2011 International Conference on Field-Programmable Technology (FPT).* IEEE, 2011, pp. 1–8.

[11] K. Paulsson, M. Hübner, M. Jung, and J. Becker, "Methods for run-time failure recognition and recovery in dynamic and partial reconfigurable systems based on Xilinx Virtex-II Pro FPGAs," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06).* IEEE, 2006.

[12] D. Agiakatsikas, N. T. Nguyen, Z. Zhao, T. Wu, E. Cetin, O. Diessel, and L. Gong, "Reconfiguration Control Networks for TMR Systems with Module-based Recovery," in *2016 IEEE 24rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 2016, pp. 88–91.

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

[14] N. T. 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 Special Issue on Reconfigurable Computing and FPGA Technology, Journal of Parallel and Distributed Computing.

[15] Z. Zhao, D. Agiakatsikas, N. T. Nguyen, E. Cetin, and O. Diessel, "Fine-grained Module-based Error Recovery in FPGA-based TMR Systems," Accepted to 2016 International Conference on Field-Programmable Technology (FPT).

[16] E. Cetin, O. Diessel, T. Li, J. A. Ambrose, T. Fisk, S. Parameswaran, and A. G. Dempster, "Overview and Investigation of SEU Detection and Recovery Approaches for FPGA-Based Heterogeneous Systems," in *FPGAs and Parallel Architectures for Aerospace Applications.* Springer, 2016, pp. 33–46.

[17] N. W. Bergmann and A. S. Dawood, "Reconfigurable computers in space: problems, solutions and future directions," in *The 2nd Annual Military and Aerospace Applications of Programmable Logic Devices (MAPLD'99) Conference*, 2000.

[18] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[19] R. Trautner, "ESA's roadmap for next generation payload data processors," in *Proc. DASIA Conf*, vol. 1, 2011.

[20] A. S. Dawood, S. Visser, and J. Williams, "Reconfigurable FPGAs for real time image processing in space," in *14th International Conference on Digital Signal Processing*, vol. 2. IEEE, 2002, pp. 845–848.

[21] P. Bergsman, "Xilinx FPGA blasted into orbit," *Xcell Journal*, vol. 46, pp. 86–88, 2003.

[22] *UG909: Vivado Design Suite User Guide - Partial Reconfiguration*, Xilinx Inc., 2015.

[23] J. Becker, M. Huebner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 438–452, 2007.

[24] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 37, 2015.

[25] J. Wang, "Radiation effects in FPGAs," in *Proceedings of the 9th Workshop on Electronics for LHC Experiments*, 2003, pp. 34–43.

[26] *Understanding Latch-Up in Advanced CMOS Logic*, Fairchild Inc., 1989. [Online]. Available: `https://www.fairchildsemi.com/application-notes/AN/AN-600.pdf`

[27] W. Morris, "Latchup in CMOS," in *2003 41st Annual IEEE International Reliability Physics Symposium (IRPS) Proceedings.* IEEE, 2003, pp. 76–84.

[28] D. M. Hiemstra and V. Kirischian, "Single Event Upset Characterization of the Kintex-7 Field Programmable Gate Array Using Proton Irradiation," in *2014 IEEE Radiation Effects Data Workshop (REDW).* IEEE, 2014, pp. 1–4.

[29] M. J. Gadlage, R. D. Schrimpf, J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Sibley, K. Avery, and T. L. Turflinger, "Single event transient pulse widths in digital microcircuits," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3285–3290, 2004.

[30] N. Rezzak, J.-J. Wang, D. Dsilva, and N. Jat, "TID and SEE Characterization of Microsemi's 4th Generation Radiation Tolerant RTG4 Flash-Based FPGA," in *2015 IEEE Radiation Effects Data Workshop (REDW).* IEEE, 2015, pp. 1–6.

[31] D. R. Alexander, "Transient ionizing radiation effects in devices and circuits," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 565–582, 2003.

[32] *Radiation-Hardened, Space-Grade Virtex-5QV Family Overview*, Xilinx Inc., 2014.

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

[34] T. Calin, M. Nicolaidis, and R. Velazco, "Upset-hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, pp. 2874–8, 1996.

[35] 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, 2014.

[36] G. Swift, C. Carmichael, G. Allen, G. Madias, E. Miller, and R. Monreal, "Compendium of XRTC radiation results on all single-event effects observed in the Virtex-5QV," *Proceedings of NASA Military and Aerospace Programmable Logic Devices (MAPLD)*, pp. 1–33, 2011.

[37] P. Graham, H. Quinn, J. Krone, M. Caffrey, and S. Rezgui, “Radiation-Induced Multi-Bit Upsets in SRAM-Based FPGAs,” in *IEEE Nuclear and Space Radiation Effects Conference (NSREC)*, vol. 5. IEEE, 2005.

[38] *WP0191: Mitigation of Radiation Effects in RTG4 Radiation-Tolerant Flash FPGAs*, Microsemi Inc., 2015.

[39] B. Bridgford, C. Carmichael, and C. W. Tseng, “Single-event upset mitigation selection guide,” *Xilinx Application Note, XAPP987 (v1. 0)*, 2008.

[40] M. D. Berg, K. A. LaBel, and J. Pellish, “Single Event Effects in FPGA Devices 2014-2015,” *NASA Technical Report Server (NTRS)*, 2015. [Online]. Available: `http://ntrs.nasa.gov/search.jsp?R=20150015964`

[41] 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 *2014 IEEE Radiation Effects Data Workshop (REDW)*. IEEE, 2014, pp. 1–5.

[42] A. J. Tylka, J. Adams, P. R. Boberg, B. Brownstein, W. F. Dietrich, E. O. Flueckiger, E. L. Petersen, M. A. Shea, D. F. Smart, and E. C. 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, 1997.

[43] P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and G. L. Hash, “Neutron-induced latchup in SRAMs at ground level,” in *2003 41st Annual IEEE International Reliability Physics Symposium (IRPS) Proceedings*. IEEE, 2003, pp. 51–55.

[44] *PG036: Soft Error Mitigation Controller v4.1 LogiCORE IP Product Guide*, Xilinx Inc., 2015.

[45] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, “Estimating soft processor soft error sensitivity through fault injection,” in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2015, pp. 143–150.

[46] G. Asadi and M. B. Tahoori, “Soft error rate estimation and mitigation for SRAM-based FPGAs,” in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2005, pp. 149–160.

[47] 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 *Proceedings of the 2008 International Conference on Engineering of Reconfigurable Systems & Algorithms (ERSA)*, 2008, pp. 139–145.

[48] *AMBA AXI and ACE Protocol Specification*, ARM Inc., 2003.

[49] R. Herveille, “WISHBONE system-on-chip (SoC) interconnection architecture for portable IP cores,” 2002. [Online]. Available: `http://cdn.opencores.org/downloads/wbspec_b3.pdf`

[50] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA).* ACM, 2010, pp. 249–258.

[51] *UG470: 7 Series FPGAs Configuration User Guide*, Xilinx Inc., 2013.

[52] *UG953: Vivado Design Suite 7 Series FPGA Libraries Guide*, Xilinx Inc., 2012.

[53] J. Tonfat, F. Kastensmidt, and R. Reis, "Energy efficient frame-level redundancy scrubbing technique for SRAM-based FPGAs," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS).* IEEE, 2015, pp. 1–8.

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

[55] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and G. R. Sechi, "Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems," in *Proceedings of 1998 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT).* IEEE, 1998, pp. 233–240.

[56] P. Alfke, "Metastable recovery in Virtex-II Pro FPGAs," *Xilinx, XAPP094. Feb*, 2005.

[57] C. E. Cummings, "Synthesis and scripting techniques for designing multi-asynchronous clock designs," in *SNUG 2001 (Synopsys Users Group Conference, San Jose, CA, 2001) User Papers*, 2001.

[58] *Fast Simplex Link (FSL) V20 Bus (v2.11f)*, Xilinx Inc., 2012.

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

[60] R. Le, "Soft error mitigation using prioritized essential bits," *Xilinx XAPP538 (v1.0)*, 2012.

[61] D. McMurtrey, K. S. Morgan, B. Pratt, and M. J. Wirthlin, "Estimating TMR reliability on FPGAs using Markov models," 2008. [Online]. Available: `http://scholarsarchive.byu.edu/facpub/149`

[62] S. Werner, J. Navaridas, and M. Luján, "A survey on design approaches to circumvent permanent faults in networks-on-chip," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 59, 2016.

[63] *UG191: Virtex-5 FPGA Configuration User Guide*, Xilinx Inc., 2006.