CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Evaluation of Fault-Tolerance Methods in Commercial Off-The-Shelf FPGAs Used in Harsh Environments

Master's thesis in Embedded Electronic System Design

Benjamin Björklund
Michael Ngibuini

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Evaluation of Fault-Tolerance Methods in Commercial Off-The-Shelf FPGAs Used in Harsh Environments

BENJAMIN BJÖRKLUND
MICHAEL NGIBUINI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Evaluation of Fault-Tolerance Methods in Commercial Off-The-Shelf FPGAs Used in Harsh Environments
BENJAMIN BJÖRKLUND

MICHAEL NGIBUINI

Supervisor: Lena Peterson, Department of Computer Science and Engineering
Company advisor: Martin Rönnbäck, Frontgrade Gaisler
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Evaluation of Fault-Tolerance Methods in Commercial Off-The-Shelf FPGAs Used in Harsh Environments
BENJAMIN BJÖRKLUND
MICHAEL NGIBUINI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The increased desire for space-based services has led companies to develop the systems using cheaper electronics while providing a large processing power. It poses a challenge where companies must consider that the system should be radiation resistant. The purpose of the thesis is to explore three fault tolerance methods and implement two in the NOEL-V processor, which runs on a commercial-off-the-shelf SRAM FPGA followed by a concept of fault fault injection in the processor during runtime. After testing the system with the created fault injection, triple modular redundancy (TMR) and lockstep were implemented to compare their clock rate, power consumption, and resource utilisation with a base design. The results show a functioning concept of fault injection and that the lockstep method consumes a moderate amount of resources and power while maintaining the same clock rate as the base design. Findings suggest that the current fault tolerance solution should be expanded to lower levels of the processor and increasing support for the fault injection to target specific registers.

Keywords: Fault-tolerance, RISC-V, FPGA, Triple-modular redundancy, Lockstep, fault injection, radiation

Acknowledgements

We would like to thank our supervisor, Lena Peterson and our examiner Per Larsson-Edefors for their guidance and suggestions during the thesis. We would like to thank our advisor from Frontgrade Gaisler, Martin Rönnbäck for his help and guidance during the thesis. We are also thankful to the employees in Frontgrade Gaisler for their guidance and support throughout the project.

Benjamin Björklund, Michael Ngibuini, Gothenburg, October 2023

List of Abbreviations

Below is the list of abbreviations that have been used throughout this thesis listed in alphabetical order:

AHB	Advanced high-performance bus
AMBA	Advanced microcontroller bus architecture
APB	Advanced peripheral bus
ARM	Advanced RISC Machines
ASIC	Application specific integrated circuits
COTS	Commercial-off the shelf
CMOS	Complementary metal oxide semiconductor
DDR	Double data rate
DSP	Digital signal processing
ECC	Error-correcting codes
EEPROM	Electrically erasable programmable read-only memory
FPGA	Field-programmable gate array
FSM	Finite-state machine
FT	Fault tolerant
HDL	Hardware design language
I/O	Input/output
IP	Intellectual property
ISA	Instruction set architecture
JTAG	Joint test action group
LUT	Look up table
MBU	Multi-bit upsets
ODRG	On-demand redundancy grouping
PCIe	Peripheral component interconnect express
RAM	Random access memory
RISC-V	Reduced instruction set computer five
SDC	Synopsys design constraints
SEE	Single event effects
SEU	Single event upset
SIHFT	Software-implemented hardware fault tolerance
SRAM	Static random access memory
TCL	Tool command language
TMR	Triple modular redundancy
UART	Universal asynchronous receiver transmitter
USB	Universal Serial Bus
VHDL	Very high-speed integrated circuit hardware description language

Contents

1

Introduction

Space electronics have become even more advanced due to the increased desire for space-based services such as communication, earth-observation, and navigation that require more functions in the satellites. The miniaturisation of components has also driven the development of cheaper devices capable of providing a large processing power. The desire for cheaper electronics used for space systems has led to exploration of the role of commercial-off-the-shelf SRAM field programmable gate arrays (COTS SRAM FPGAs). They offer a low-cost solution while providing state-of-art components such as digital signal processors (DSPs) and reconfigurability as opposed to application-specific integrated circuits (ASICs). Using COTS SRAM FPGAs means that systems can be built at a lower cost whilst maintaining the required performance. One potential disadvantage of using COTS SRAM FPGAs in space applications is that they are more susceptible to single-event upsets (SEU) in harsh environments than other alternatives. However, it is crucial to note that all electronic systems in space face similar environmental challenges and have to be designed to minimise the effect of SEUs. Techniques that increase the system-level tolerance towards SEU are available, such as triplicating the logic, but cause a significant increase in area usage and power consumption.

The purpose of this project is to explore techniques for minimising the effects for SEUs in one particular COTS SRAM FPGA developed by Microchip, known as Polarfire. The FPGA is engineered for space and military applications, with reliability and security as top priorities. Frontgrade Gaisler develops and supports the GRLIB-integrated very high-speed integrated circuit hardware description language (VHDL) intellectual property (IP) library that is available as open-source. The NOEL-V processor created for space applications is available in the library, and a similar version available for the Polarfire was used to implement and benchmark a mitigation strategy.

1.1 Background

As mentioned before, the desire for low-cost alternatives has led companies to consider the role of COTS SRAM FPGAs in space applications. COTS FPGAs offer the ability for post-fabrication reconfiguration, high performance with dense logic, and built-in digital signal processing (DSP) blocks, which are all desirable features for space applications [1].

However, COTS SRAM FPGAs are vulnerable to SEUs because they affect the programmable logic and RAM. The programmable logic and RAM can be protected against SEU by various techniques. These techniques protect them from detrimental effects caused by SEUs. The SEUs are caused by radiation particles that induce a current pulse in the P-N junction and affect the functionality of the transistor [2]. One common technique is triple modular redundancy(TMR), which involves triplicating the circuit and voting to determine the correct output. However, TMR requires a significant amount of resources that affects performance while only hiding faults from being detected [3].

A hybrid format of TMR called on-demand redundancy grouping (ODRG) can switch between two fault-tolerant cores to perform critical tasks or individual cores for more performance. The hardware required must contain additional processor cores to have the ability to deploy ODRG followed along with a configurable voter that decides the correct result [4].

Other techniques are based on software and hardware redundancy for error detection. One of them being lockstep which uses multiple checkpoints at the software level with the ability to roll back while also including duplication of the processor with additional circuits to check the logic [5]. Lockstep can also be applied on the hardware level where the first processor runs a few cycles ahead on the second one where comparison occurs until the second processor produces a result [6].

One of the purely software-implemented techniques is software-implemented hardware fault tolerance (SIHFT) which implements additional instructions at the software level comparing values to detect SEUs without modification of existing hardware. This technique relies on making adjustments to the existing code by adding more instructions to compare values or handling branch executions [7].

1.2 Problem description

The implementation of fault tolerance is heavily dependent on the system's architecture. Testing the system also presents an issue because it relies on direct contact with radiation. While there are programs capable of performing fault injection, they require restarting the system to inject errors. With this problem, the thesis will attempt to answer the following research questions:

- Which fault tolerance method is suitable for the NOEL-V?
- How can the fault injection be performed on the NOEL-V to test the method chosen and is it suitable?
- How does the method perform compared to the baseline design?

1.3 Aim and objectives

The thesis project explores various options for system-level fault tolerance and their implementation on the NOEL-V, both in the market and academia. Additionally, the project investigates and implements a concept of fault injection to test the chosen fault tolerance method on the NOEL-V. Based on the aims of exploring system-level fault tolerance and implementing fault injection testing, the following objectives were formulated:

- Develop and test a concept of performing fault injection for the NOEL-V.
- Implement a mitigation strategy and benchmark against the base design.

The thesis will delve into the different methods for fault tolerance, among which one will be implemented and benchmarked against the base design. Since it is well known that TMR can guard against SEU, fault injection will only be performed on the chosen method and will not involve exposing the FPGA to radiation. Other known effects that can occur when exposing the FPGA to radiation will not be observed.

1.4 Thesis outline

Chapter 2 introduces the theoretical framework, covering topics such as SEU and its effects on electronic systems, the NOEL-V processor and its architecture, and various fault tolerance methods used. Chapter 3 provides an overview of the methods and tools used in the thesis project. Chapter 4 discusses design choices for implementing the fault tolerance method with a detailed account of the fault injection process used to test system robustness. Chapter 5 presents the results, followed by Chapter 6 that presents a more in-depth analysis of the design choices and offers suggestions for future work. The thesis concludes in Chapter 7, with a presentation of the implications and key findings.

2

Theory

This chapter provides a detailed background on processors, with a particular emphasis on the NOEL-V architecture. It covers topics such as benchmarking, fault tolerance methods, and SEUs (single event upsets), while also presenting various approaches to fault tolerance through examples.

2.1 Central processing unit (CPU)

In the context of computer architecture, the CPU is responsible for executing instructions, which it does in three stages referred to as a pipeline: fetching, decoding, and execution. The arithmetic logic unit (ALU) inside the CPU implements the operations defined in the instruction set architecture (ISA) while registers temporarily store data and signals. The control unit manages instruction fetching, and an internal bus connects the ALU, registers, and control unit [8]. The CPU's operations are synchronized by a master clock that generates a fixed frequency by pulsing between high and low states. The master clock is determined by the slowest component in the CPU and generated by an oscillator. This signal is then divided or multiplied to generate an internal clock signal that coordinates the fetching, decoding, and execution of instructions.

Since the ISA decides what the CPU should do, it heavily impacts the overall architecture and dictates how the data and instructions should be processed. The processor NOEL-V used in the thesis project uses the ISA called RISC-V for its instructions because of its philosophies to have an open ISA capable of supporting any hardware [9, 10].

There are two primary CPU architectures, Von Neumann and Harvard. The Von Neumann architecture utilises the same memory space for instructions and data, allowing for more complex instructions. In contrast, the Harvard architecture uses separate memory spaces for instructions and data [8]. The Harvard architecture is used for the NOEL-V pipeline where there are separate data and instruction caches along with the interfaces for each pipeline stage available in the CPU core. The cores are a part of the NOEL-V subsystem which is illustrated in Figure 2.1.

2.1.1 NOEL-V Processor

The NOEL-V processor, developed by Gaisler, is a synthesizable RISC-V architecture processor with a 7-stage pipeline capable to support 32-bit or 64-bit architectures. It is a dual-issue processor meaning that it can issue two instructions per clock cycle. Additionally, the NOEL-V contains a subsystem that connects to different peripherals, which enables the creation of a NOEL-V-based system-on-chip. Illustrated in Figure 2.1 is a block diagram of the subsystem along with the controllers for the processor bus [11]. The controllers in the subsystem take care of the communication with external IP connected to the AHB and APB buses. The debug system in Figure 2.1 is connected to multiple debug ports available within the CPU core that access the control and status registers [11].

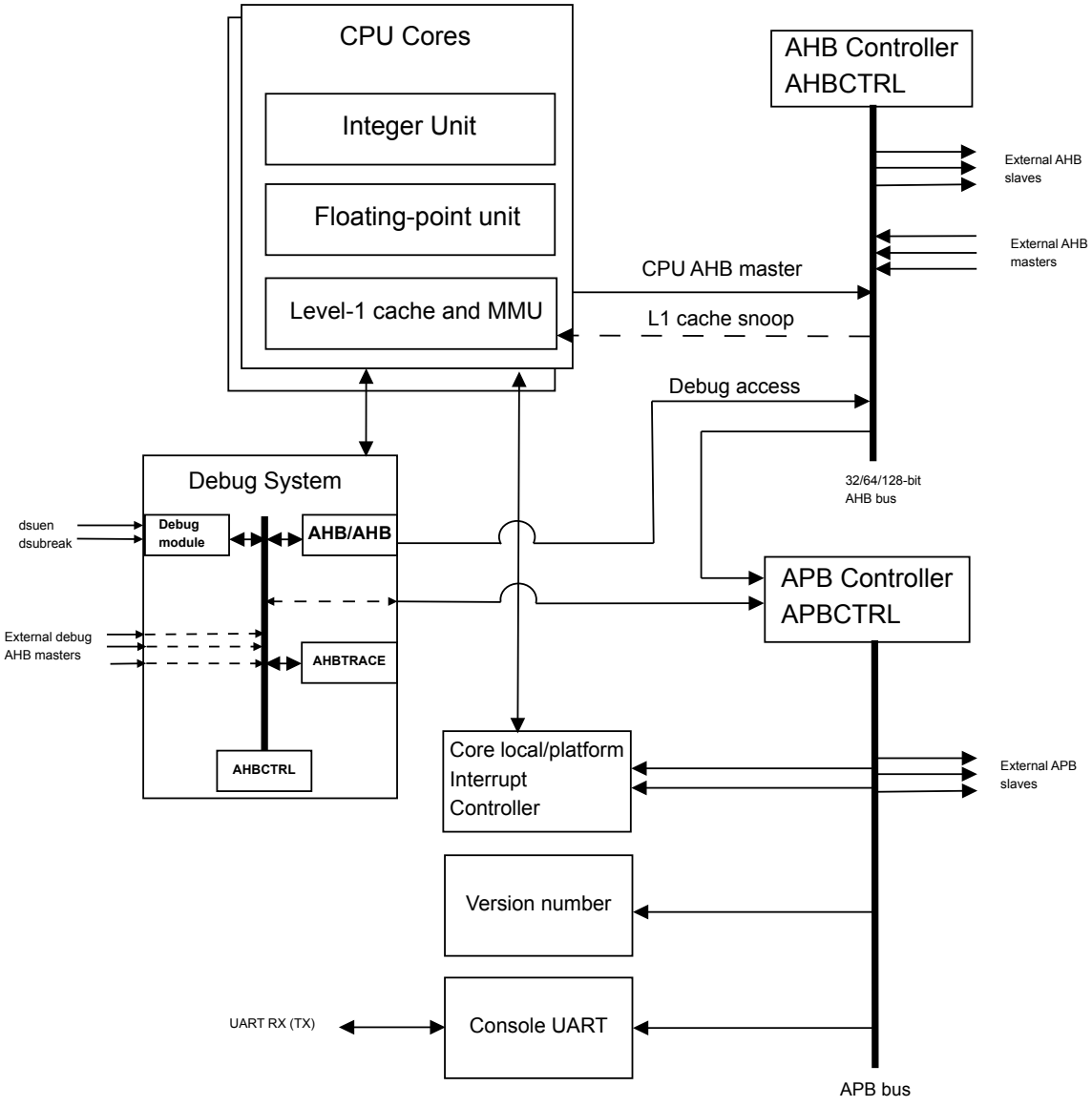

Figure 2.1: Subsystem of the NOEL-V processor [11].

The RISC-V architecture provides a high degree of flexibility and compatibility with existing commercial software. Thanks to recent advances in fault-tolerant techniques, the space industry has become interested in exploring the potential of RISC-V processors. This interest has motivated the development of the NOEL-V, which incorporates advanced features such as an enhanced branch predictor and a wide datapath that supports dual issue operations. Overall, the NOEL-V represents a promising new direction for fault-tolerant computing in space applications [9]. Connections with the rest of the intellectual property (IP) block in the GRLIB library is done through the advanced microprocessor bus architecture (AMBA) that also provides an easy interaction with microprocessors. The specific parts that handle the interaction are the APBCTRL and AHBCTRL which are located in the NOEL-V subsystem as illustrated in Figure 2.1.

2.1.2 Advanced microprocessor bus architecture (AMBA)

AMBA is a bus specification created by ARM used to define on-chip communication when designing high-performance embedded microprocessors. There are multiple buses defined within the AMBA specification; the common ones are the advanced high-performance bus (AHB), advanced system bus (ASB), and the advanced peripheral bus (APB). The AHB bus is considered the backbone thanks to the connection it provides for on, and off-chip memories while also interfacing with other low-power functions [12]. It has write and read buses that are 64-bit wide with space for extension.

The APB bus is a specification tailored to low-power peripherals with reduced interface complexity while being low-cost. The lack of a pipeline makes the APB bus simple where every transfer takes two cycles to complete. The APB has write and read buses that are 32-bit wide but extended with some configuration [13].

2.1.3 Processor benchmarking

Measuring processor performance involves running benchmarks and the user evaluates the time it takes for a specific task or workload to be processed. Benchmarks are carefully crafted software programs that stress different aspects of a system or component, allowing consumers to compare and evaluate performance [14]. They can be tailored towards solving specific problems or designed to be more generic, providing valuable information about the system. In addition to assessing systems or components, benchmarks can be used to measure the performance of individual software components or algorithms [14].

Some benchmarks are designed to test specific elements and are known as synthetic benchmarks. Unlike more general tests, synthetic benchmarks focus on one aspect of the system's performance. As a result, they are used to measure the performance of a particular subsystem or hardware component. However, they do not reflect the actual behaviour of programs or workloads that run on the system. Although synthetic benchmarks do produce a measurable result, they do not involve any computing tasks and may not accurately represent real-world usage scenarios [15]. Synthetic benchmarks may not reflect actual program behaviour, but they can match instructions for large programs and reduce simulation time. By targeting specific subsystems, they provide insights into system performance for optimisation and faster evaluation of changes [16].

2.2 Types of FPGAs

FPGAs are devices that can be programmed to represent any system designed digitally. They typically consist of logic blocks, interconnection networks, and configurable I/O blocks. The logic blocks are divided into two parts, one part consists of sequential operations while the other handles the combinatorial parts [17]. The interconnection networks are programmed by the user to connect the logic blocks with the I/O blocks. The I/O blocks are used to connect the pins located on the FPGA with the programmed logic [18].

In most modern FPGAs, additional circuits are added to support logic that is too large to be implemented in the logic blocks. Some of them are DSP blocks that support complex mathematical operations, on-chip memory for storing data temporarily, phase-locked-loops that help maintain low timing variations for clocks, and various data processing blocks that support protocols such as USB [19].

The major difference between FPGAs is the memories used for programming the configuration cells. The configuration cells store the configuration data that describes the connection between the various blocks in the FPGA [20]. These cells can be made using different technologies where each technology has advantages and disadvantages.

2.2.1 Anti-fuse programming

Anti-fuse programming uses metal-to-metal structures that are fused together by applying a large current. The structures create a link that stores the configuration which cannot be altered after applying a large current eliminating the need to reprogram the device after power cycling and the possibility for reconfiguration [17].

Anti-fuse programming is also considered very secure due to the lack of reconfiguration making it resilient to radiation and other attacks [21]. The data in an FPGA containing anti-fuse programming is difficult to read because the electrical short creates permanent connections making it difficult to read the data and detect any manufacturing faults. One benefit is that the connection requires no silicon area but can be done on metal-to-metal reducing the area required for programming [21], [17].

2.2.2 Flash/EEPROM memory programming

Flash/EEPROM programming uses electronically erasable memory that is non-volatile meaning that the contents loaded are still stored even when powered down. The connections are made using two transistors that share a gate floating above that is electrically isolated. Since the gate is isolated, any charge placed there will not discharge for a long time [17].

As mentioned before, the non-volatility of the memories stores the configuration removing the need for reconfiguration when powered up. Flash memory is considered as a low power technology thanks to its non-volatile characteristic [21]. Flash memories also consume a small amount of area due to the circuits required to program the cells are far smaller [22], [20].

One disadvantage is the limited amount of times that they can be reprogrammed due to the isolation. This prevents the cells from being fully erased. Another disadvantage is the lack of miniaturisation caused by the use of transistor based switches limiting them from using the standard CMOS manufacturing process [17], [22].

2.2.3 SRAM programming

SRAM programming is the latest technology used in FPGAs for programming. It uses the latest CMOS manufacturing process that allows for faster access time to memory, high density of cells in small area, and low power [21]. SRAM FPGAs also implements the logic blocks with the same cells used to control the interconnection networks that connect the logic and I/O blocks [22].

The memory used to store the configuration is volatile whereby the contents are lost when the FPGA is power cycled. The configuration data is stored first in non-volatile manner by the programmer allowing the user to reconfigure the FPGA for an unlimited number of times [21]. SRAMs also benefit from the latest CMOS manufacturing process that provides the highest performance and low-power consumption [20].

One disadvantage is the amount of transistors required to create the cell that holds the configuration data. The amount of transistors consume a large area compared to anti-fuse and flash technologies [20]. This issue makes SRAM FPGAs highly susceptible to SEU that can cause failures in the memory cells [23]. Despite the cells being re-programmable an indefinite amount of times, the configuration data must be load everytime upon power up. This raises concerns regarding the security of the data which can be intercepted during programming [21].

2.3 Single Event Upsets (SEU)

SEU are a class of soft-error variety of single event effects (SEE) that occur when a single radioactive particle hits a device with enough charge to cause a sudden change in the logic. The particle that hits the device induces a shift in the voltage level within the memory, register, latch, or flip-flop. The region in the device that is struck with the particle experiences a change in the logic where it changes to either "0" or "1". This bit flip can stay and propagate through the system leading to a loss of the original value [24].

There are two types of SEU: single bit upsets (SBU) and multi-bit upsets (MBU). An MBU occurs when a single particle strikes the device at a certain angle allowing it to strike two places causing multiple bits to change [25]. In contrast, SBU only induces one bit change and it is the most common phenomenon out of the two types [26].

FPGAs suffer from SEU due to the method used to store data. SRAM FPGAs are most susceptible to SEU due to the memory cells in the SRAM requiring power to retain their states. Error correcting codes (ECC) can be added to the memory that assist the user to implement scrubbing mechanisms [27].

2.4 Methods used to Mitigate SEU

As mentioned in section 1.1, there are methods used to mitigate SEU to protect COTS FPGAs from radiation. Each technique is presented along with their advantages, disadvantages, and challenges when applying them.

2.4.1 Triple modular redundancy (TMR)

Triple-modular redundancy (TMR) involves triplicating the logic in a design and adding voters that vote on the output that aid in masking errors. It can be done either through using design tools or manually [28]. TMR is effective when mitigating SEUs but it has some drawbacks, one of which is its high area consumption due to the additional logic required. The added logic also causes other problems such as increased power consumption and lower clock rate because the path lengths increase [29]. Illustrated in Figure 2.2 are modules that have been triplicated where their output is sent to a voter to decide the correct output.

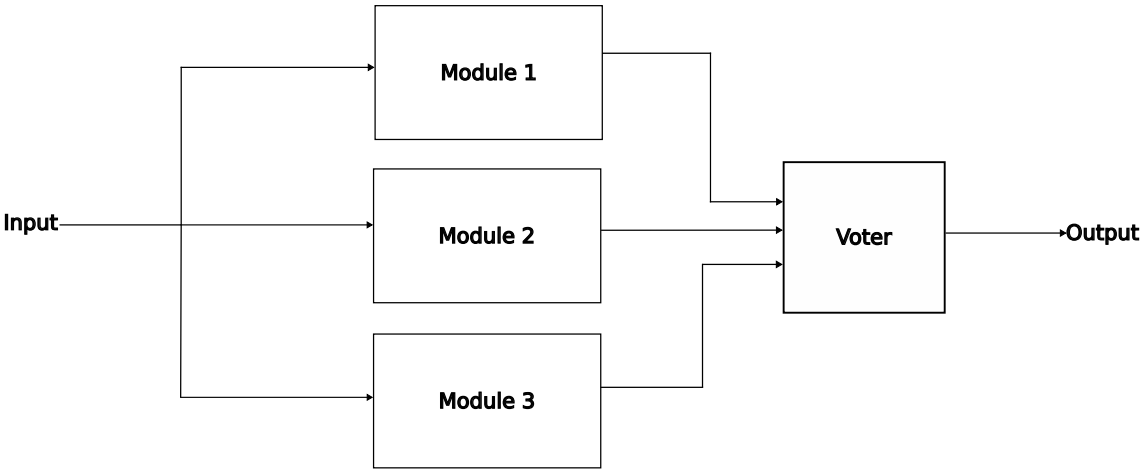

Figure 2.2: Triplicated modules that each receive the same input and vote to decide the output [28].

TMR alone masks the memory errors caused by SEU and reconfiguration techniques are added to prevent a build-up of multiple errors. Therefore TMR cannot withstand a large amount of faults as they would build up and break the redundancy [30]. The solution is to combine it with some configuration technique that reads the configuration data, detects errors, and rewrites the data with the correct one. This technique is called scrubbing [31]. Scrubbing cannot be applied to sequential circuits since the state information is saved [32].

Due to limiting factors such as IP control and resource limitations, TMR cannot be applied to every circuit in SRAM FPGAs. This limitation requires careful consideration regarding where to apply TMR as it can impact the FPGAs tolerance towards SEUs. This method is called partial TMR and it provides a trade-off between resource utilisation and error mitigation [33]. Partial TMR allows the designer to target the parts of the design that contain configuration bits that can cause errors when SEUs occur to ensure no important data is lost. It involves identifying the areas most sensitive to SEUs and applying TMR to achieve a desired reliability level [34].

2.4.2 Lockstep

Lockstep is a technique that uses identical processors that run the same application in parallel. The processors are synchronised from the start of the application and they receive the same inputs. The state of the application should be the identical for the processors for every clock cycle unless anomalies occur [35]. The principle is to detect errors by comparing the processor states after the application has been executed for a certain duration. These states are referred to as verification points where the running application is locked to check the processors state. Should discrepancies occur, the whole application will be stopped and the processors are reset to a state without any errors. The processors re-execute the program again from the last verification point with no errors [36].

The method of saving the error free states is referred to as checkpointing. The states of the processors are saved in secure memories that are protected against SEU with the help of error correcting codes [35] The re-execution of the programs can occur in two ways: one can roll-back to the last error-free execution that has been saved or perform a roll-forward operation that restores to pre-calculated error free state to set the processors to a consistent state [35].

The important aspects to consider when performing the roll-back operation is that an interrupt is sent to the processor and that there has to be a sanity check that occurs before creating the checkpoint. A trade-off occurs where if too many sanity checks are implemented, the interruptions can cause a loss of performance. If there are few sanity checks, the number of faults can increase which prevents the program from executing properly [37].

Lockstep itself as a technique is not limited to any specific type of hardware. Researchers have managed to implement their own version of lockstep on different processor architectures [38], [39].

2.4.3 Software-implemented hardware fault tolerance (SIHFT)

Software-implemented hardware fault tolerance (SIHFT) is a technique that leverages the ability to detect SEUs during the execution of the program by using additional instructions to either check the values in the variables during load and store instructions or checking the program's flow [40]. It can be done manually or with the help of different tools that automatically inject these instructions into the code.

For SIHFT, it is assumed that the faults caused by SEUs affect the system and did not cause the errors in the deployed software. Depending on the chosen method, one can opt to either protect the data or the flow of the program [41]. Methods such as instruction level duplication leverage added instructions that check the values of the variables. There is also the possibility of applying the instructions on specific values to ease the cost of memory and performance [41].

Another method involves implementing control flow checks, which partition the program into simple blocks that do not have any branching in the code. Comparisons of the result from the simple blocks are made by comparing the computational time with pre-computed ones to then decide whether or not to progress to the next block [42]. This method makes it easier for the programmer to verify the functionality and works regardless of the hardware used.

SIHFT has disadvantages such as the added instructions introduce an increased performance overhead where they consume a large amount of resources [43]. The execution of the added instructions can also increase the processing time of the application running, thus affecting the execution time and introducing delays. Another consideration is that there can be an increase in overhead time depending on the added instructions. Therefore, it is important to consider where the instructions are added to reduce the added overhead time [43].

2.4.4 On-demand Redundancy Grouping (ODRG)

On-demand redundancy grouping (ODRG) is a relatively new mitigation technique developed for the parallel ultra low power (PULP) platform, which is a joint effort from the universities of Zürich and Bologna, with the goal of exploring different architectures to achieve low power consumption [4]. ODRG builds on the principle of allowing multiple RISC-V cores to be grouped on the fly based on the demand. This technique achieves similar functionality as TMR at the core level where a majority voter is added to vote on the output of the cores.

ODRG uses the additional cores by adding extra elements that connect up to three cores to the same output allowing three cores to execute the same process. The results from the three cores are then sent to a majority voter that checks the result. The user can choose whether to have the cores operating individually for maximum performance or group the multiple cores into two larger cores connected to a majority voter for added fault tolerance. An example of ODRG is illustrated in Figure 2.3 where three cores are grouped along with a majority voter.

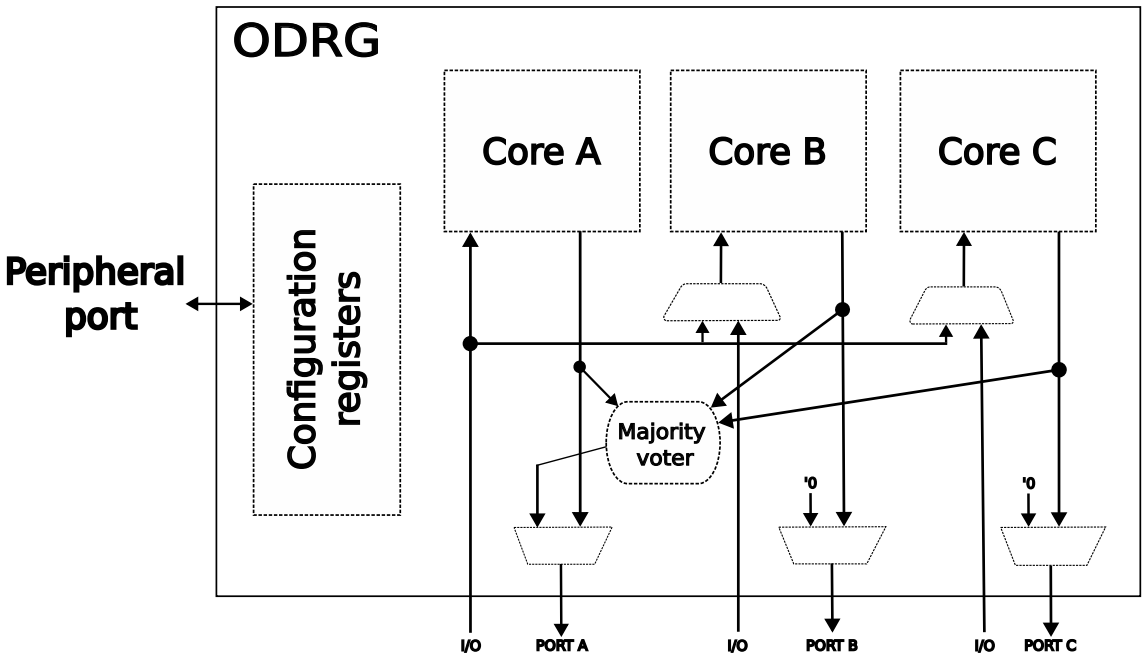

Figure 2.3: Diagram over the ODRG with 3 cores wrapped with a majority voter [4].

When the cores are in the performance mode, they each have individual inputs allowing them to operate in parallel and send data to the caches. In the fault-tolerant setting, three cores function in a lock-step operation where they all have the same input to compute and communicate with the rest of the system as a cluster where the result goes to a voter for comparison.

When a core produces a mismatch, the three cores stop and a re-synchronisation process is started to restore the core that caused the mismatch. Should all cores produce a mismatch, the re-synchronisation process takes longer to find a stable state where the cluster produced the same result. During this period, the host also has the ability to send an interrupt to reset the cluster.

One advantage with this method is the ability for the user to switch between six cores for more processing power and three cores for fault tolerance eliminating the need for additional configuration to protect against SEU. One drawback is that all cores must share the same instruction cache requiring modification to any architecture that has separate instruction caches in the cores themselves.

3

Methods

This chapter presents the tools, software, and the steps taken during the thesis project. It also contains details regarding the choice of which method to compare against the baseline and the reasons behind this choice.

3.1 Software and Hardware

To test and deploy the NOEL-V processor, various hardware and software are needed such as the benchmarking program, the COTS SRAM FPGA, and software for accessing the processor itself.

3.1.1 Polarfire Splash Kit

The Polarfire splash kit is an SRAM FPGA kit provided by Microchip that contains general purpose interfaces that are used for evaluation and development. The kit itself comes with gigabit ethernet, peripheral component interconnect express (PCIe), universal serial buss (USB), and onboard double data rate 4 (DDR4) memory together with the MPF300T_ES-FCG484I that is a part of the MPF300T device family provided by Microchip [44]. The family is a part of the low power, cost-optimised FPGAs that Microchip offer for applications within cellular infrastructure, defence, commercial aviation, and industrial automation [45]. The FPGA was chosen because it is currently used for space applications.

3.1.2 GRMON3

GRMON3 is a hardware monitor developed by Gaisler, optimized for their processors. It provides an environment where the system can be monitored through a graphical user interface that also supports scripting. The program supports multiple debugging links and can read and write the processor's registers and memory [46]

3.1.3 Dhrystone

Dhrystone is a synthetic benchmarking software created by Reinhold P. Weicker to measure the performance of computer systems by testing their integer performance [47]. The benchmark consists of 12 procedures in a single measurement loop to produce the performance rating in Dhrystones per second. Dhrystone itself is light and does not require any operating system to run while fitting in on-chip caches [48]. It was picked because it's been a choice in the industry for many years due to its ease of implementation and reliability. Benchmark results from Dhrystone are often expressed as Dhrystone million instructions per second (MIPS) or DMIPS/MHz where lower values are a better score.

3.1.4 Microsemi Libero

`Libero` is a comprehensive collection of design and verification tools that enable the implementation of various fault tolerance methods in digital systems. The software supports a wide range of FPGA devices, including the Polarfire FPGA. `Libero` offers an integrated simulation module with `Modelsim`, allowing users to test the functionality of the processor. In addition, the software integrates `synplify pro` for synthesis that provides a suite for additional settings such as re-timing and clock constraints [49], [50].

3.1.5 Commercial and fault tolerant (FT) version of the NOEL-V

Introduced in section 2.1.1, the NOEL-V is a processor developed by Gaisler for space applications. Gaisler provides multiple versions of the NOEL-V but two versions were made available: the commercial version and FT version. The main difference between the two versions is that the FT version has added ECC in the on-chip memories for added protection against SEUs [51].

3.2 Procedure

This section outlines the criteria used to select the method for comparison against the baseline. These criteria encompass FPGA resource availability, practicality within allocated time frames, and the potential impact on the NOEL-V base design. Additionally, it includes an evaluation of ODRG, SIHFT, and lockstep against these criteria to ensure a meaningful comparison.

3.2.1 Choosing the fault tolerance method

The criteria to decide which method would be compared against the baseline were based on the following:

- Does the FPGA have an adequate amount of resources (4LUTs, Flip-flops)?
- Is the method applicable within the allocated time?
- How will the fault-injection process take place in the new method?
- Are additional tools required?
- Will the new method affect the base design of the NOEL-V?

These criteria evaluate the feasibility, practicality, and impact of a chosen method in the context of FPGA-based development and were decided upon via discussion with the company and the project group.

The chosen method would need to be added to the NOEL-V and requiring minor modifications to the original design to create space for the benchmarking software and the fault injection process. The three methods ODRG, SIHFT, and lockstep mentioned in section 2.4 were analysed based on the points above to ensure that the comparison with the baseline and TMR design provided a significant difference.

Investigating other designs that used TMR like ODRG, it was discovered that the resource consumption was higher which introduced some misgivings regarding lockstep and ODRG. Since the method of ODRG relies on six separate processors that can run individually or grouped, there was a risk that the resources in the FPGA would not be enough.

Due to the additional logic required for ODRG, especially for the majority voter and core grouping, ODRG was not considered. This decision was made because the area consumption was too high based on the results obtained with two cores added. If an attempt to add additional cores in the design to resemble the ODRG core, there would be a few resources remaining for the additional logic such as the majority voter and registers to control the grouping of the cores.

The base functionality of the NOEL-V would also be affected since the cores would run in parallel requiring modifications to the original design to accommodate for parallel execution. Additionally, the fault injection would have to provide a choice of which processor to inject errors to, thus requiring adding means of identifying each core individually.

While investigating SIHFT with the listed criteria, we found that it would be the best in terms of resource consumption and did not affect the base design of the NOEL-V, but it would require looking into the internals of the compiler when it came to configure specific things such as branching. This approach would limit the added fault tolerance to a specific program instead of being applicable for any program. In addition to compiler adjustments, duplication of instructions would increase the amount of memory used which could cause delays when instructions are being processed. Since the majority of the load and store instructions require duplication, there is a high chance that performance is lost due to the increased time taken to process instructions. While the fault injection process would not be affected since the base design would be used, there is a chance of losing memory performance to the added instructions.

The method ultimately chosen for testing was lockstep which can be implemented at any level. Depending on the chosen implementation, more or less resources can be consumed when implementing the synchronisation logic. In addition, the design time to implement the lockstep method depended entirely on which level it is applied on. Since it requires two processors, the configuration of the NOEL-V made it easier to configure another processor core within the original design.

3.2.2 Testing equipment and process

The first step involved loading the NOEL-V design onto the FPGA to get a simple program up and running. The process helped us understand the necessary steps to load the processor to the card and providing a starting point for evaluating resource consumption before introducing fault-tolerant measures. Files containing the processor design for the FPGA, as well as a file that created the entire project in `libero`, were available. Before any programming could be conducted on the FPGA, it was essential to understand the specific settings related to the tools used for synthesis and technology mapping leading to a simplified testing procedure and ensuring a fair comparison.

Synthesis and technology mapping were conducted through `synplify pro` since it was the default option set by `libero`. Constraint settings such as FSM compiler, and re-timing allowed the synthesis tool to optimise the design for increased performance. Included in synplify was a detailed report regarding the area consumption, recommended clock rate for the module, and the set clock rate. The attribute parameter in `synplify pro` allows for special commands that are viable depending on the board used. The parameter *syn_radhardlevel* applies TMR in the design based on the desired level.

Files with pre-written C code were compiled through Gaisler's internal compiler that could program the NOEL-V. Choosing which clock rate to set was limited by the different peripheral units in the design and the values of crystal oscillators available by manufacturers. It should be possible to run the entire processor through the crystal instead of the internal clock in the board. During the programming, Dhrystone manages to fill both instruction and data registers where `GRMON` allowed the observation of changes in the registers in contrast to an operating system.

3.2.3 Fault injection process

The fault injection process started by first analysing the NOEL-V processor core to see where the RAM required access to the layer that wraps the defined memory blocks to the MPF300T. The error-injected RAM received commands through `GRMON` via the joint test action group (JTAG) port. Pre-existing software tracked whether the fault injection to the RAMs in the NOEL-V occurred correctly. Emulation of SEU meant emulating a bit-flip in the RAM by changing only one bit. Access to the error-injected RAM meant connecting a port to the top level where the core has access to the debugging tool.

In addition, the logic for deciding when the fault injection would begin since some processor functions would interfere with the data written to the RAM. It meant analysing how the existing RAM functioned before any attempts at fault injection were made. To assess the success of the fault injection, it was determined through visible printouts or a message confirming that the processor was in debug mode.

4

Design

This chapter provides the design aspects undertaken during the thesis project and the decisions made during the implementation.

4.1 Application of TMR on the NOEL-V

To successfully apply TMR on the processor, additional functions such as ECC are required in the core to scrub the faults that have built up. The ECC are included in the FT-version of the NOEL-V processor where direct access to the core was limited because of the VHDL files were encrypted making it impossible to add modifications for the fault injection. Adding TMR to the design was done by launching `synplify pro` through `libero` to apply the attributes. The attribute parameter in `synplify pro` allows the user to direct the tool regarding the optimisation of the design during the compilation stage during synthesis. One can either add the attributes directly in the hardware design language (HDL) source code or through a constraint file (.sdc). The attribute *syn_radhardlevel* applied the radiation-resistant technique through the constraint file to decrease the time spent performing synthesis. One thing to note that this parameter is only applicable on FPGAs such as the MPF300T that has radhard devices.

The added attribute has three different values to choose from:

- *cc*: Combinatorial cells with feedback as a method of storage instead of using conventional flip-flops or latches.
- *tmr*: Triple-module redundancy is used for registers. Three flip-flops or latches are used for each register that votes on the state of the register.
- *tmr_cc*: Triple-module redundancy is used along with combinatorial cells with feedback for each voting register.

We decided to chose *tmr* as that was the most familiar one compared to the others. Application of the attribute occurred at the top level of the design so that TMR is applied on every instance in the hierarchy. The tool would then target the registers and add voters without affecting the overall ports, clock, and the memory controller.

In addition, there were several advanced features that can further enhance the performance and efficiency of the design. For instance, the FSM compiler can optimise any state machines in the design, while the FSM explorer allows for testing of different encoding styles for these state machines. The timing report generates a log file that documents any changes made to the design providing a valuable record for analysis and troubleshooting purposes. By utilising these advanced options, one could observe how the tool applied a clock frequency based on the specific settings. Upon completing the synthesis process, `synplify Pro` generated data regarding the number of cells and LUTs consumed that closely matched those obtained through `libero` after running place and route.

4.2 Lockstep implementation of the NOEL-V

There are two ways the lockstep method could be implemented within our two-core system: within the core themselves or outside each core. If implemented inside the core themselves, each stage within the pipeline would be observed to compare the results stored after each instruction is processed. This method would check for any inconsistencies in the pipelines and force a re-fetching of the instruction that caused the error. When implemented outside the cores, the AMBA bus connecting the two cores would be compared to detect any inconsistencies and force a re-execution to process the data again.

The approach taken was to implement the lockstep method outside each core because it avoids affecting the original functionality of the processor while preserving the original design. Since the AMBA signal contains the data sent to the processors, it is easier to analyse them without adding extra signals that can affect the overall functionality. The idea is illustrated in Figure 4.1 along with the signals sent to the synchronisation module.

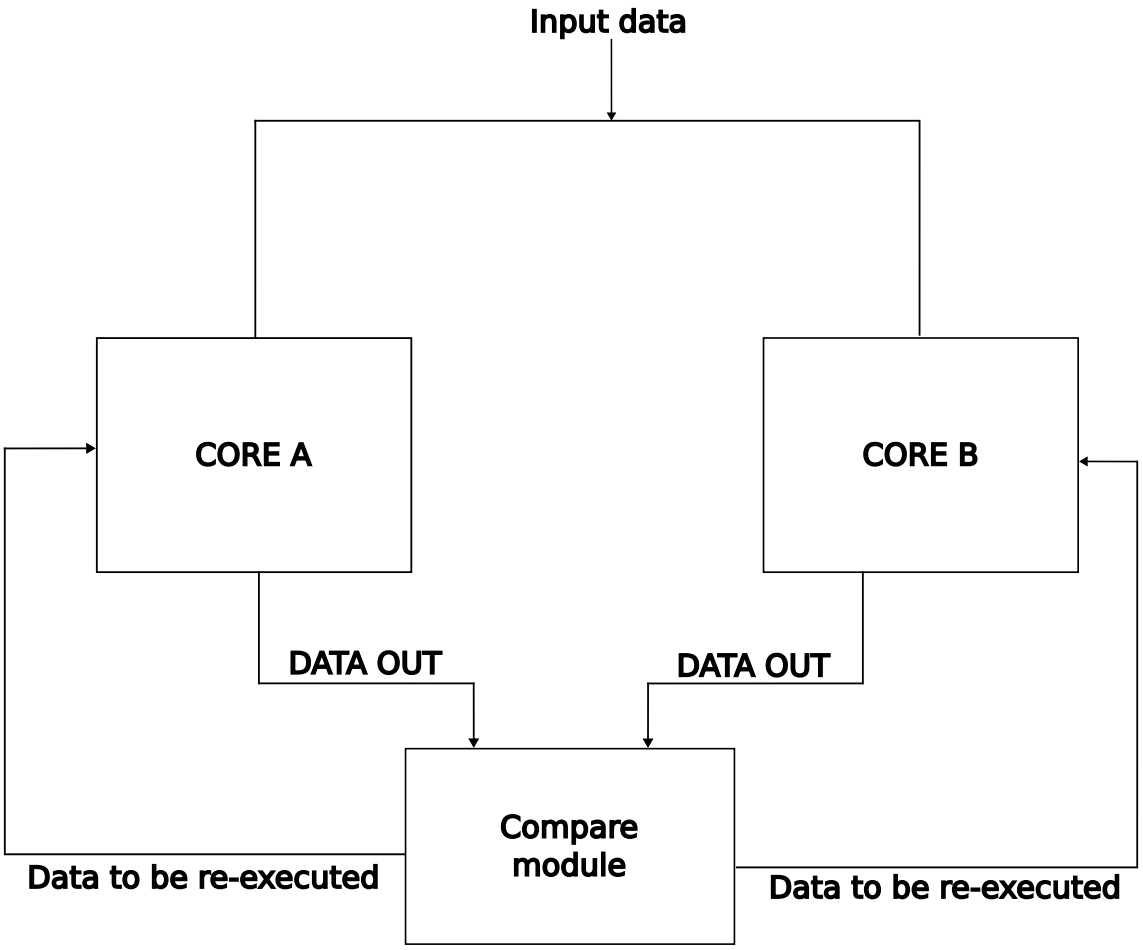

Figure 4.1: Configuration of the lockstep implementation.

The added core is simply a copy of the first core. The two cores share the same input data that is included in the AMBA signal which allows for the same instructions and data to be executed. Similar to the input data, data out from the core is included in the AMBA signal that is sent to the compare module.

4.2.1 Synchronisation module for lockstep

Adding another core to the design means that the data being sent via the AMBA bus could be compared to detect differences between them. A method to synchronise the two cores using the input data itself was required. The idea was to compare the data on the AMBA bus from both cores and compare them with each other. The two cores would continue executing as usual and data would be re-loaded when a mismatch is detected.

Efforts applying the lockstep configuration began by looking at the current design with one core and the different signals that were used to send data to the AMBA bus. A program that controlled the amount of processors was provided and used as a starting base regarding the testing of the NOEL-V with multiple cores.

The first version was implemented with a state machine where the output data dictated which state to jump to. When a mismatch occurred, the data from the first core would be sent out to the AMBA bus. There were some issues such as the loaded program not running as intended because the connection to the AMBA bus was lost and even the connection to the processor itself. The root issue was that the added state machine caused synchronisation issues with the data in the AMBA bus resulting in a lose of connection.

The tested variant was a simple implementation using if-statements in VHDL to check the output of the core which was expanded to save the data that was sent to both cores. The data would then be returned to the cores to be processed again where the inputs have to match if the process is to continue. Illustrated in Figure 4.2 is the output of the two cores are sent to be compared and store the input data if they produce a similar output.

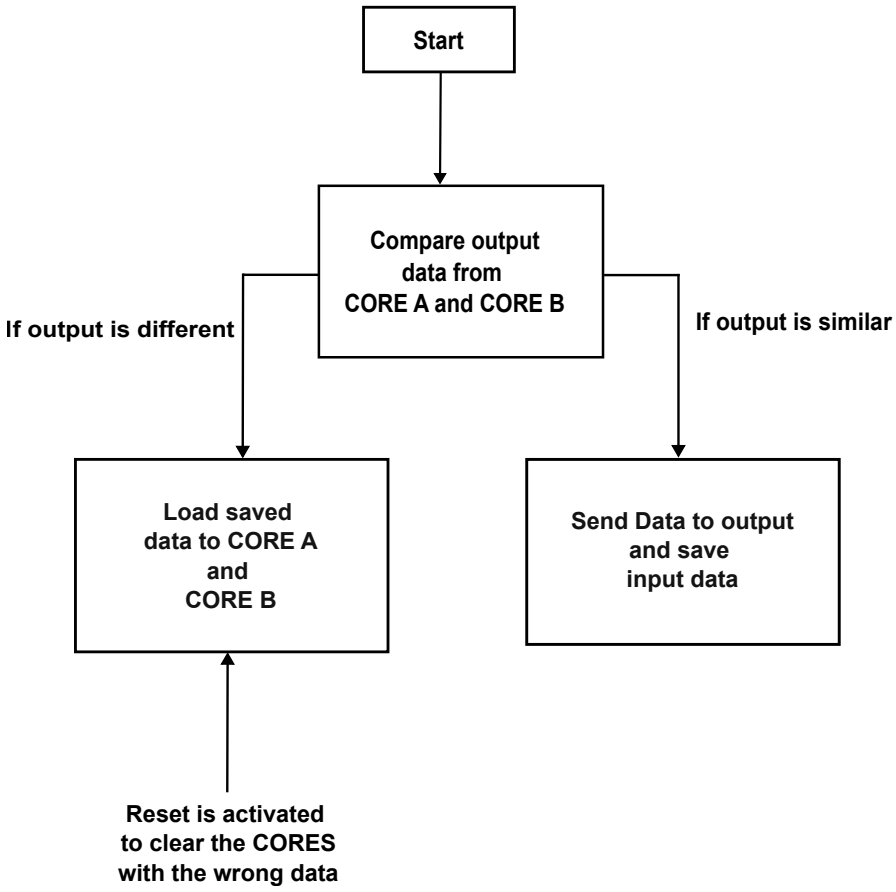

Figure 4.2: General idea of the synchronisation logic implemented to check the data from the cores.

The input data that produced the similar output would be saved to re-load it if the next output data caused a mismatch on one core. The implementation did not require any synchronous components and could be implemented in combinatorial logic. Testing of the synchronisation logic meant initially performing the fault injection on both cores to observe the effects followed by only one core.

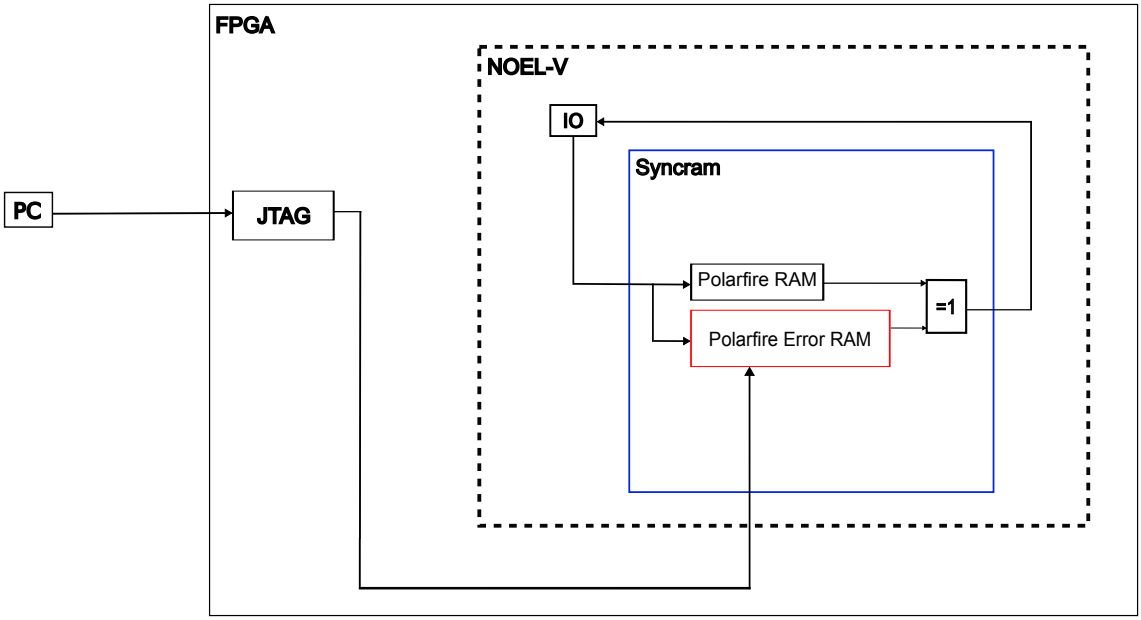

Figure 4.3: Block diagram of the fault injection to the NOEL-V core where the red block is the modified syncram for the polarfire version and the black one being the regular one.

4.3 Fault Injection in the NOEL-V core

To facilitate the fault injection into the NOEL-V core, a copy of the RAM was created with inputs to receive the data and address of where the mask would be sent to. This process is carried out using `GRMON`, which accesses various registers peripherals in the processor. Communication with peripherals for the NOEL-V was done through the AMBA protocol accessible via the JTAG port.

Inside the design was a module communicating with the RAM model specifically for the Polarfire FPGA. The module called *syncram* creates different RAM models depending on the type of FPGA used. The syncram is used to create the cache and register file for the NOEL-V where a dedicated version for the polarfire FPGA was modified.

Data from both RAMs is transmitted to the I/O using an xor-gate to detect the differences of the data read from the RAMs. This approach offered the advantage of simplifying the existing RAM while preserving the original structure since the RAMs for various FPGAs were available in the file. This concept is depicted in Figure 4.3, showing the communication from the technology specific RAMs being sent to the processor's I/Os.

Before creating the new RAM, we analysed signals used to communicate with the RAM to understand their specific function. The error RAM had similar signals as the regular one with the only difference being the use of the *testin* signal used to send the data. The IP block provided by Gaisler could map to any technology-specific RAM block depending on the FPGA used. Since the internals of the RAM were already defined, the only requirement was to declare and instantiate the error RAM.

The modification seen in Figure 4.3 was only made on the commercial version of the NOEL-V because of limited access in the FT-version making the subsystem in Figure 2.1 only visible. By adding a copy of the declared RAM, we could access the cache and register file could be accessed for fault injection. Since the provided syncram module contained a version of the RAM model specifically for the Polarfire, this way of adding the modification would ensure the effect would be applied on the various modules that used the RAM to create the memory storage.

The newly made copy of the Polarfire RAM was based on the existing one but with added logic to handle inputs coming from the user. The input from the user would contain the bit-mask and a write signal. The write signal was used to prevent the bit-mask written to the error RAM after every write signal sent to the ordinary RAM. The initial contents of the error RAM is zero that allowed for normal functionality. In addition, the user should be able to write to the error RAM through the general purpose register (GRGPREG) IP that was mapped to the APB address bar.

As mentioned in section 2.1.2, the APB bus takes care of multiple peripherals connected to the NOEL-V cores through the AHB bus as illustrated in Figure 2.1. Since the APB bus is used multiple times in the NOEL-V design, it is important to decide on the address where the signal should be placed to avoid collisions with addresses used by other modules. Therefore, the output of the GRGPREG was connected to the provided address. In `GRMON`, the *info sys* command provided an overview of the entire system, including the different modules connected and their respective addresses.

The error RAM initiated error injection with minimal data, using only a bit-mask and a write signal. Initially, it contained zeros to maintain normal functionality, with assistance from an XOR gate. Upon adding a controller, data with the bit-mask and write signal was sent directly to the controller via the GRGPREG. The controller processed the input with the specified data and write signal to the error RAM.

4.3.1 Controller for fault injection

The primary function of the controller was to enable fault injection based on user requests. A module to control the input of data and apply the write signal was created and added to the design. The bit mask was generated initially through a linear-feedback shift register, which generated a pseudo-random bit position in the data to flip it. It was used to test whether the system detected the other RAM and whether any changes occurred when data was being read from the RAM. The linear-feedback shift register was later replaced with data being sent from the GRGPREG IP that enabled the user to send bit-masks and a write signal.

The controller ultimately was based on combinatorial logic that separated the write signal from the bit mask. The write signal was used to write the bit mask to the RAM. The bit mask in this instance would be a series of bits where one was set to 1 while the rest were set to 0. It was sent to the error RAM to later affect the output from the ordinary RAM. Data to the controller itself would be received from the GRGPREG IP that was mapped to the APB address bar and accessed via `GRMON`.

Figure 4.4 illustrates the controller with the signals it sends to and receives from the error-injected RAM within the Polarfire Error RAM file. The main function of the write signal is to allow the user to directly write bit masks to the error RAM when a write process occurs in the ordinary RAM.

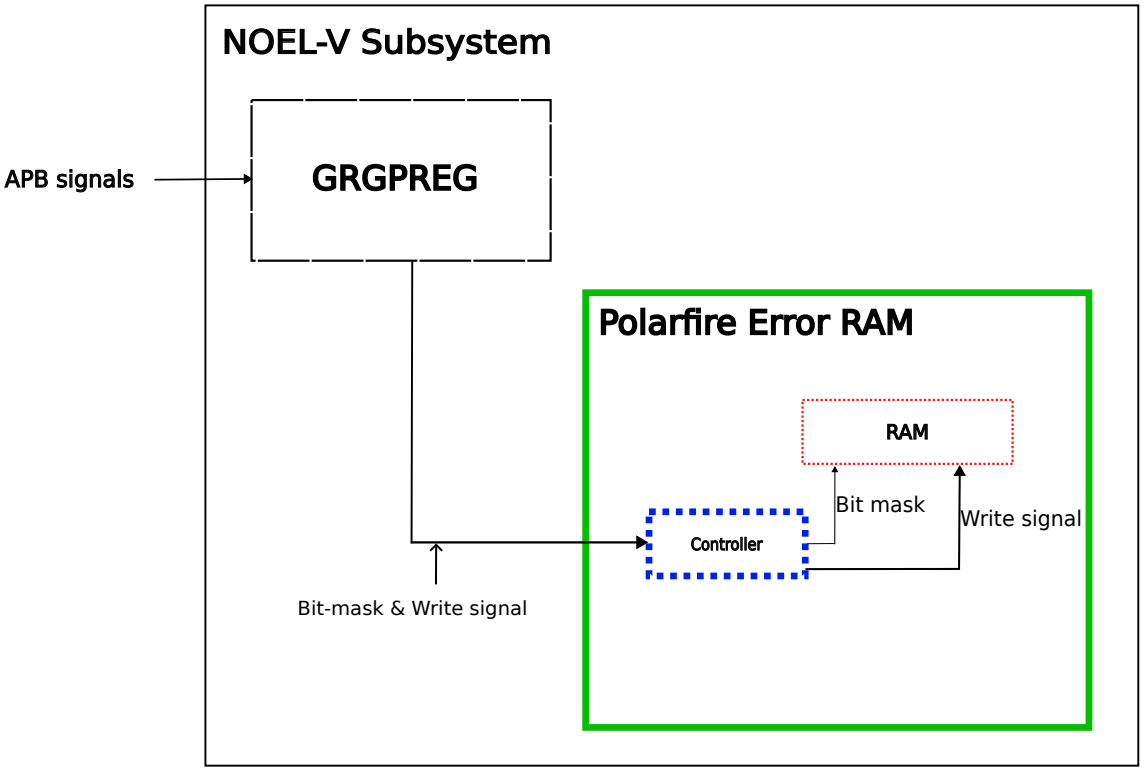

Figure 4.4: Interaction of the controller (blue) with the RAM (red) available in the Polarfire error RAM

The main function of the controller is to capture the input data from the GRGPREG IP and route it to the error RAM. The bit-mask is sent directly to the data bits of the RAM where they are loaded in. The data was loaded based on the write signal received from the user and the ordinary RAM. The general idea is to use an AND gate to control when the error RAM is written to. The write process to the error RAM is done when the user sends the write command and when the ordinary RAM is written to. To prevent any errors when running programs, the error RAM is filled with zeros when the write signal from the user is not sent.

Including the GRGPREG to the design started by sending a signal that would activate a simple LED. It showed that communication via `GRMON` to the processor core was possible and that the chosen address for the GRGPREG did not interfere with the other peripheral units in the APB address bar. Expanding the communication from the GRGPREG to the controller itself was simple by creating a signal from the subsystem itself as seen in Figure 4.4.

Since the GRGPREG is a peripheral unit, it accesses the same signals as the console UART through the APBCTRL. Adding it to the design consists of changing the number of peripherals in the APB bus and providing an appropriate address on the APB bus to communicate with it. After configuration, the GRGPREG was visible in `GRMON` with the address to access for sending data to the controller that loaded data to the error RAM.

5

Results

This chapter provides results based on the designs illustrated in chapter 4. Section 5.1 covers the utilisation and power consumption of the different designs of the NOEL-V processor. Section 5.2 and 5.3 cover the benchmark results of the designs and the fault injection in the single core and lockstep design.

5.1 NOEL-V design utilisation and power consumption

Before performing any benchmarks on the three designs, the resource utilisation and power consumption was observed to see the differences. The clock rate of the designs was found by performing static timing analysis in `libero` with a breakdown of the slack time in the different parts of the design.

5.1.1 Utilisation

Three different designs of the NOEL-V were tested after performing synthesis and timing verification to observe the utilisation and clock rate for each design. A baseline of the processor without any modifications for further comparison with the other two designs. The major difference between the designs is the TMR has added ECC in the RAMs. Illustrated in Table 5.1 is the utilisation for the different parameters for the three designs.

Table 5.1: Clock-rate and resource utilisation of the designs

Type	**4LUT**	**DFF**	**Logic Element**	**Clock Rate (MHz)**
Baseline design	92819	39078	99200	50.0
TMR design	152609	109527	207783	35.0
Dual-core design	159657	59754	168276	50.0

The major difference seen in Table 5.1 is the clock rate for the designs. The TMR design ran at 35 MHz before experiencing issues with timing. As expected, the utilisation numbers for the dual-core and TMR designs were larger than the baseline design. Figure 5.1 shows the utilisation numbers seen in Table 5.1 where the difference between the designs is more noticeable.

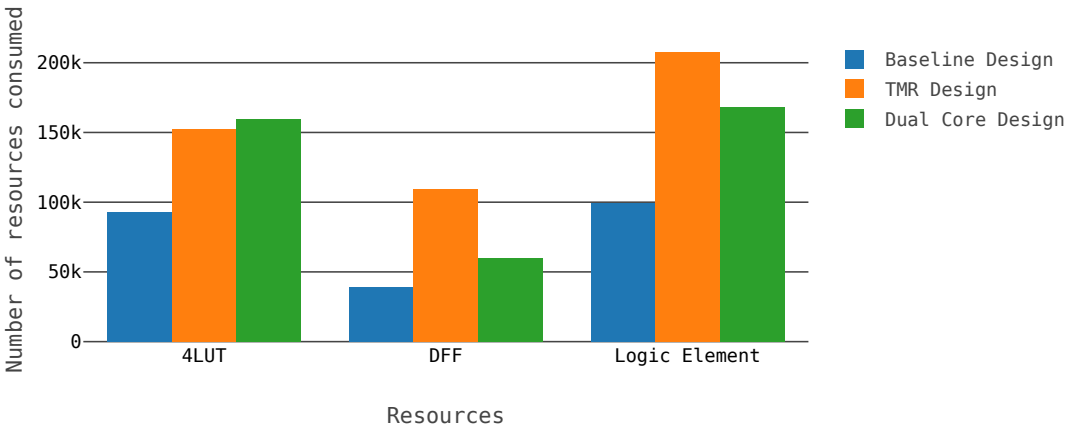

Figure 5.1: Resource utilisation values over the three designs

The amount of D-type flip-flops (DFFs) and logic elements showed a large difference between the three designs. The TMR designs used the largest number of DFFs because they are used to apply majority voters on each instance. Looking at the dual-core design, it used the most of the 4LUTs to implement another core while still maintaining a high clock rate. The unusual high number of logic elements on the TMR design is caused by the added voters on instances in the overall design including the memory controllers for the Polarfire FPGA.

5.1.2 Power

The dynamic power was measured for each design to investigate if the added resources caused a power increase. The baseline design was again used as a comparison point against the other two designs. Illustrated in Table 5.2 is the power consumption for the three designs.

Table 5.2: Dynamic power and total power consumption for the three designs

	Baseline design	**TMR design**	**Dual-core design**
Dynamic Power (mW)	463	930	588
Total Power (mW)	1204	1672	1330

The TMR design consumed more than twice the dynamic power of the baseline design. The added DFFs cause a large increase in the dynamic power on account of the voters added to each instance in the design. The dual-core design showed a minor increase in dynamic power compared to the baseline design despite having the added core. Since the static power was similar for all three designs, the only parameter that affected the total power was the dynamic power and looking at the consumption in Table 5.1, the design with the most DFFs had the highest consumption of overall power. A different illustration of the numbers is seen in Figure 5.2.

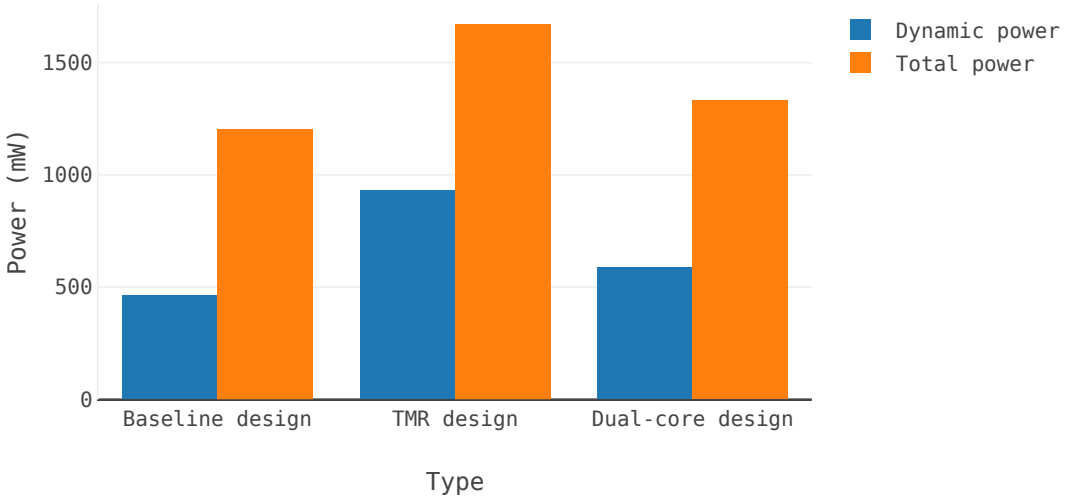

Figure 5.2: Power consumption over the three designs

One thing to note is that the static power for the designs is large due to the memory controllers for the Polarfire being active and used to generate the clocks to access the memory. The static power is the same amount for the baseline and TMR design but roughly doubles for the dual-core because of the added core. Since the difference between the three designs is at the core level, the power consumption was still higher for the TMR design compared to the baseline and dual-core design. The static power was almost double for the dual-core design compared to the baseline design. Illustrated in Table 5.3 is the power consumption for the three designs at the core level.

Table 5.3: Static and dynamic power at the core level for the three designs

	Baseline design	**TMR design**	**Dual-core design**
Static Power (mW)	0.543	0.543	0.991
Dynamic Power (mW)	140	295	264

5.2 Benchmarks

All three designs were benchmarked with the program dhrystone due to its simplicity and requiring no operating system to run. Table 5.4 illustrates minor differences between the designs in terms of resource utilisation, specifically regarding the benchmark results from Dhrystone for the three designs.

Table 5.4: Benchmark results for the three designs

	Baseline design	**TMR Design**	**Dual-core design**
Dhrystone per second	189392.5	187264.4	189392.5
Dhrystone MIPS	107.8	106.6	107.8
DMIPS/MHz	2.156	3.046	2.156

As expected, the TMR design performed worse because of its lower clock rate. The added DFFs increase the latency in the design causing it to fail at higher clock rates which in turn forces the design to run at lower clock rates. The baseline and dual-core designs performed similarly despite the added core. Included in Table 5.4 is the DMIPS/MHz score that shows the performance difference between the designs more clearly showing that the baseline design and dual-core design provide better performance compared to the TMR design.

5.3 Fault injection

Testing began on the baseline design to observe whether the added GRGPREG could send data to the error RAM. This meant sending the bit-mask without the write signal to see the effect. Two programs were available for testing: a simple hello world program that displays the text *hello world* on the screen and the dhrystone benchmark itself. Before sending the run command to `GRMON`, the *verify* command was used to check whether the program was uploaded correctly. The effect of the mentioned test above is illustrated in Figure 5.3 where the black arrow shows the bit-mask fed to the error program along with the *verify* command.

The loaded program was unable to execute properly and upon using the *verify* command, multiple errors were detected showing different values in the address. The errors themselves are illustrated in Figure 5.3 where the values for the higher addresses had conflicting values. The reason it happened is that the processor reads random data from the XOR gate which meant that the error RAM was not empty preventing the program from running as intended.

```
grmon3> wmem 0xfc003000 0x00000003

grmon3> load hello.elf
                0 .text             19.1kB /  19.1kB   [================>] 100%
             4C6C .rodata            116B              [================>] 100%
             4CE0 .init_array          4B              [================>] 100%
             4CE4 .fini_array          4B              [================>] 100%
             4CE8 .data              1.1kB /   1.1kB   [================>] 100%
             5158 .sdata              28B              [================>] 100%
             5174 .eh_frame            4B              [================>] 100%
  Total size: 20.37kB (790.75kbit/s)
  Entry point 0x00000000
  Image /home/michaeln/Desktop/Master_Thesis/ncc-1.0.4-gcc-linux64/ncc-1.0.4-gcc

grmon3> verify hello.elf
                0 .text             19.1kB /  19.1kB   [================>] 100%
             4C6C .rodata            116B              [================>] 100%
             4CE0 .init_array          4B              [================>] 100%
             4CE4 .fini_array          4B              [================>] 100%
             4CE8 .data              1.1kB /   1.1kB   [================>] 100%
             5158 .sdata              28B              [================>] 100%
             5174 .eh_frame            4B              [================>] 100%
  Total size: 20.37kB (692.32kbit/s)
  Entry point 0x00000000
  Image of /home/michaeln/Desktop/Master_Thesis/ncc-1.0.4-gcc-linux64/ncc-1.0.4-

grmon3> run

  Interrupted!
  0x00000170: 34202073  csrr    zero, mcause  <__bcc_trap_table+0>

grmon3> verify hello.elf
                0 .text               0B              [>                ]   0%
  Verify error at address 00000030 - Read: 0xa7030000, Expected: 0x93824235
  Verify error at address 00000038 - Read: 0xf3540000, Expected: 0x13132400
  Verify error at address 0000003c - Read: 0x03000000, Expected: 0x97020000
  Verify error at address 00000040 - Read: 0x03000000, Expected: 0x9382420c
  Verify error at address 00000044 - Read: 0x03000000, Expected: 0xb3025300
```

Figure 5.3: Error injection (black arrow) to the processor without the write logic along with the run command (red arrow).

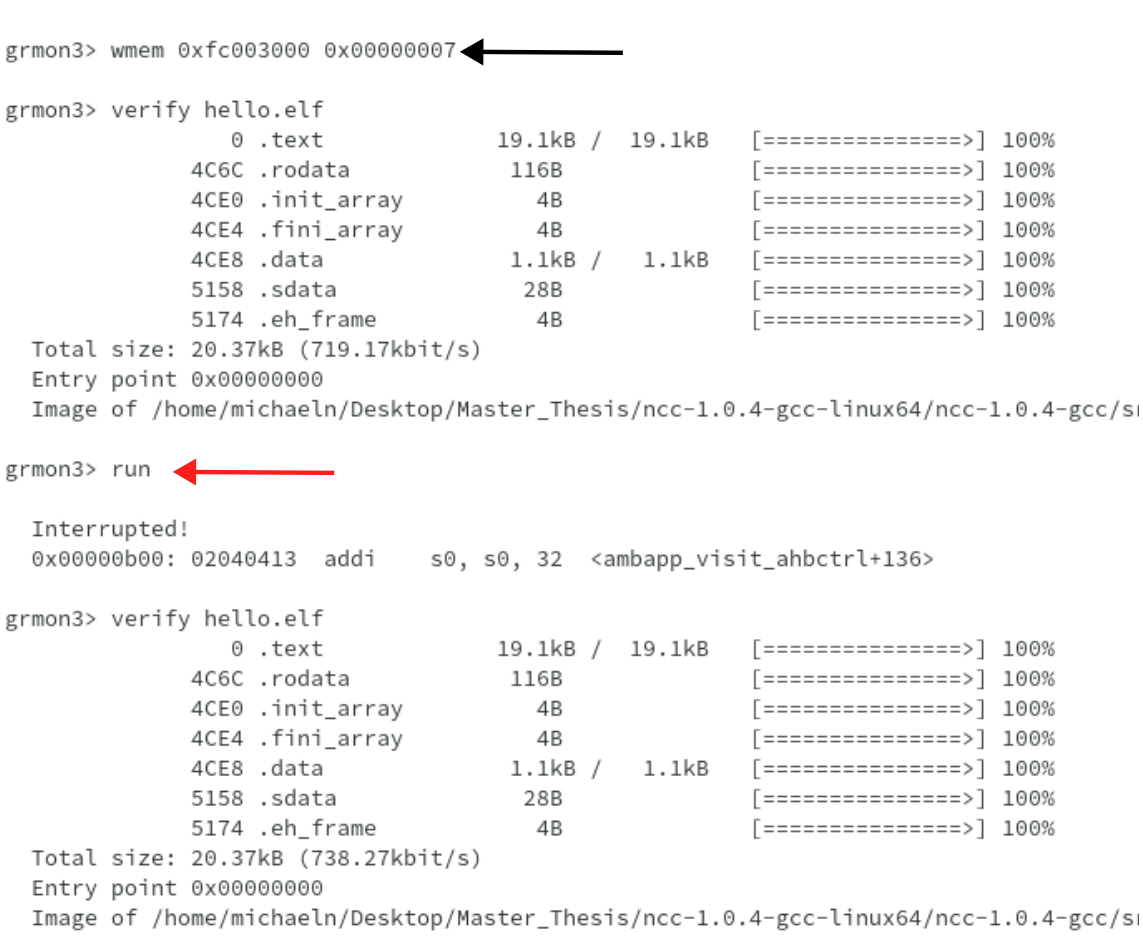

Figure 5.4: Error injection (black arrow) to the processor with the write logic added along with the run command (red arrow).

Tests continued where the write signal was sent with the bit-mask. The goal of the added write logic was to ensure that the error RAM would not affect the normal functionality when no bit-mask was sent. Same procedure was performed where the *hello world* program was uploaded and verified but the data sent is the bit-mask along with the write signal. Illustrated in Figure 5.4 is the effect seen upon verifying whether the program has been loaded correctly. According to `GRMON`, the program was loaded correctly but no message is displayed apart from the instruction it was interrupted on.

The difference is that the added write logic affects the memories that store the uploaded program by ensuring it is stored properly and can be read by GRMON. This means that the program is stored in the ordinary RAM when performing the fault injection. The current implementation shown in Figure 5.4 was used for the dual-core fault injection where the bit-mask could be sent to either core.

5.3.1 Dual-core fault injection

The fault injection was also performed on both cores to test the implemented lock-step method illustrated in Figure 4.1 with Dhrystone. During testing, only one of the core were fault injected to see if the same effect seen in Figure 5.4 was achieved. Performing the fault injection on only one core caused an effect where the program would execute continuously and require interruption. Illustrated in Figure 5.5 is the effect when fault injection is performed on one of the cores. The black and red arrows still point to similar instances as Figure 5.3 and 5.4 but the green arrow represents the output received after running the program for some time. This value meant that there was a loss of communication with the system.

The effect seen is the program constantly jumping back to the top of the RAM address where the program is initially loaded. When executed, the JTAG is disconnected from the board because the same data that caused the error is constantly fed to the cores while the rest of the system is executing normally. Figure 5.5 also shows the effect when the entire system is reset physically. The original data is still available in the RAM and when the system is reset meaning that the peripheral units are not reset when the cores themselves are reset. A reset signal was added in the compare module that was sent to the mapping of the ports. Figure 5.6 shows the effect of the reset signal being added to force the re-execution.

The black arrow in Figure 5.6 marks the bit-mask sent to the first core, the data is separated into separate vectors that allows only one value to sent in GRMON to be separated later on. The effect seen in GRMON is the benchmark constantly landing on a reset point outside of the program stack. The red arrows in Figure 5.6 show two attempts at running the program which leads to it landing on a reset point in GRMON until the error was physically cleared through the reset button. Despite mapping the reset from the compare module to the ports of the FPGA, the physical button for the reset was still needed because the added peripherals were not registering the reset signal from the compare modules.

```
grmon3> load dhrystone.elf
                0 .text             72.8kB /  72.8kB   [===============>] 100%
            12330 .rodata            2.7kB /   2.7kB   [===============>] 100%
            12e08 .init_array          4B              [===============>] 100%
            12e0c .fini_array          4B              [===============>] 100%
            12e10 .data              1.5kB /   1.5kB   [===============>] 100%
            133e8 .sdata             160B              [===============>] 100%
            13488 .eh_frame          180B              [===============>] 100%
  Total size: 77.30kB (779.87kbit/s)
  Entry point 0x00000000
  Image /home/michaeln/Desktop/Master_Thesis/ncc-1.0.4-gcc-linux64/ncc-1.0.4-gcc

grmon3> wmem 0xfc003000 0x00030000

grmon3> run
^C^C^C^C^C^C^C^C^[[A^[[B^C^C^C^C^C^C^C   û

^C
^C
e  0x00000000: 0
eai
grmon3>
grmon3> run
Execution starts, 1000000 runs through Dhrystone
Total execution time:                          5.3 s
Microseconds for one run through Dhrystone:    5.3
Dhrystones per Second:                      189392.5

Dhrystones MIPS      :                       107.8

  Forced into debug mode
  0x0000fb90: 00100073  ebreak  <_exit+0>

grmon3> wmem 0xfc003000
  wrong # args: should be "wmem  ?options? addr data... ?cpu#?"

grmon3> mem 0xfc003000
  0xfc003000  00000000  00000000  00000000  00000000    ................
  0xfc003010  00000000  00000000  00000000  00000000    ................
  0xfc003020  00000000  00000000  00000000  00000000    ................
  0xfc003030  00000000  00000000  00000000  00000000    ................
```

Figure 5.5: Error injection (black arrow) to one of the cores without the reset signal added

```
grmon3> load dhrystone.elf
                0 .text             72.8kB /  72.8kB   [===============>] 100%
            12330 .rodata            2.7kB /   2.7kB   [===============>] 100%
            12e08 .init_array          4B              [===============>] 100%
            12e0c .fini_array          4B              [===============>] 100%
            12e10 .data              1.5kB /   1.5kB   [===============>] 100%
            133e8 .sdata             160B              [===============>] 100%
            13488 .eh_frame          180B              [===============>] 100%
  Total size: 77.30kB (778.91kbit/s)
  Entry point 0x00000000
  Image /home/michaeln/Desktop/Master_Thesis/ncc-1.0.4-gcc-linux64/ncc-1.0.4-gcc

grmon3> wmem 0xfc003000 0x00030000

grmon3> run

  Interrupted!
  0xc00000a4: 00100073  ebreak

grmon3> reset

grmon3> run

  Interrupted!
  0xc00000a4: 00100073  ebreak

grmon3> verify dhrystone.elf
                0 .text             72.8kB /  72.8kB   [===============>] 100%
            12330 .rodata            2.7kB /   2.7kB   [===============>] 100%
            12e08 .init_array          4B              [===============>] 100%
            12e0c .fini_array          4B              [===============>] 100%
            12e10 .data              1.5kB /   1.5kB   [===============>] 100%
            133e8 .sdata             160B              [===============>] 100%
            13488 .eh_frame          180B              [===============>] 100%
  Total size: 77.30kB (725.38kbit/s)
  Entry point 0x00000000
  Image of /home/michaeln/Desktop/Master_Thesis/ncc-1.0.4-gcc-linux64/ncc-1.0.4-

grmon3> run
Execution starts, 1000000 runs through Dhrystone
Total execution time:                          5.3 s
Microseconds for one run through Dhrystone:    5.3
Dhrystones per Second:                      189392.5

Dhrystones MIPS      :                       107.8

  Forced into debug mode
  0x0000fb90: 00100073  ebreak  <_exit+0>
```

Figure 5.6: Result of the execution of the benchmark with fault injected (Black arrow) to one of the cores after adding the reset signal.

6

Discussion

This chapter discusses the fault injection and the chosen fault tolerance method. More importantly, sections 6.1 and 6.2 cover the fault injection process and the tested fault tolerance method where the drawbacks of the implementation and testing are mentioned. Section 6.3 presents overall ethical concerns and implications. Section 6.4 discusses the future work regarding the fault injection and fault tolerance method.

6.1 Fault injection process

The fault injection process was simple in terms of modification but the communication part was troublesome because of the added component required to communicate with `GRMON`. The GRGPREG was added on the APB bus because of the extensive documentation available but multiple signals were required to send the value down to the error RAM. One benefit of the current solution is that it targets all parts in the design that use the syncram component because the fault injection is sent to the error RAM that contains a RAM model tailored specifically for the Polarfire FPGA.

The drawbacks of this method is that it is tailored for the Polarfire and is not applicable to other FPGAs, limiting the ability of performing fault injection in another FPGAs. Despite the added logic for the bit-mask and write signal being added, the RAM models themselves are different for different FPGAs.

One issue is that the fault injection was performed only on the commercial version of the processor limiting the testing to memories without ECC. Therefore, we could not test whether the existing ECC manages to detect and correct memory errors which is important especially for TMR. While it may not be possible to include the current configuration in the FT-design, there is a chance that the ECC in the memory would detect the difference.

The connection made through `GRMON` via the GRGPREG component presented some limitations regarding the address offset made available for sending data to the RAMs that limited testing to the 32-bit version of the processor. While the data sent was limited to 4 bits, it was impossible to fully test the implementation with 32 or 64-bit masks sent to the RAM. Since there was no available tool command language (TCL) script capable of sending the bit-masks in a rapid manner, the use of the *wmem* command made it possible to communicate with the error RAM.

6.2 Suitable fault tolerance method

Based on the values found in Table 5.1, we found that TMR will cause a significant decrease in performance caused by the added logic to implement the voting structure seen in Figure 2.2. Since the added logic contains multiple flip flops to fully implement the voters, they each require additional time to store and process the information leading to the decreased clock rate. As illustrated in Figure 5.1 the number of flip flops were the highest for the TMR design compared to the other two designs.

The utilisation numbers also show that for TMR, when applied in a general manner, the amount of resources consumed will be high. Since `synplify pro` only allowed the triplication to occur on all design modules, there is a high chance that some parts were triplicated despite not being susceptible to SEU compared to the register file or caches. Issues with the tool regarding the application of TMR in the specific files were found because `libero` would detect the commands as a syntax error despite `synplify pro` detecting them as valid commands.

The utilisation numbers and benchmarks show that adding another core with extra logic is the suitable method for the NOEL-V. The design managed to maintain the same clock frequency as the baseline design despite a minor increase in the number of logic elements. Looking again at Figure 5.1 the number of DFFs for the dual-core design is lower than TMR showing that a large number of flip flops in the design require a low clock rate to achieve the same functionality.

The different versions of the designs also impact the utilisation numbers because of the different RAMs used. The FT version of the NOEL-V contains added ECC in the memories that are used to create the register files and caches. The added ECC also has a chance of impacting performance negatively causing the design to fail at higher clock rates because of the added protections. Combined with TMR, the latency increment leads to a decreased performance. When compared to the dual-core version, the FT-version does not have any problems regarding synchronisation and detection of errors despite no fault injection being done.

Since the register files and caches are the critical components within the processor that require protection, the dual-core design must be capable of reading the signals that are sent between the two cores for comparison. The chosen version only checks the AMBA bus between the processors leading to potential errors being missed or detecting the same errors. Errors can occur in the stages where the processor access the cache to store or load values in the same address where the cores can retrieve or store different values in the same address.

6.3 Ethical concerns

The selected approach to applying the fault tolerance method imposes the question regarding how well it can mitigate SEU. Despite prior research pointing that lockstep is a suitable choice, the question that arises is whether the system can still mitigate SEU when exposed to real radiation and the program being deployed. If the program deployed is critical to the system, the more protection is required leading to stringent requirements of the system. Since the model heavily relies on the data fed to it, the question regarding the validity of the bit-masks arises as it affects the end result. The current method lets the user dictate the bit-mask that will be sent providing the ability to see the effect based on different data.

Overall, the fault injection model presents more benefits as it can be used to create a more accurate method capable of being used in different types of COTS FPGAs providing the ability to test designs against radiation in a cheaper manner lowering the cost barrier regarding the manufacturing of space electronics and providing an easier approach to testing systems against radiation.

6.4 Future work

Despite the fault injection being successful in affecting the functionality of the NOEL-V, the current fault tolerance method relies on the data within the AMBA bus to be similar including the bus clock. An improvement would be to access the data that is sent between the cache and register file to observe how faults propagate in the pipeline and affect the data. In addition, the current architecture for the fault injection can be extended to target the pipeline as well.

There is also the exploration of different levels of the lockstep implementation whereby one could either use the pipeline between two cores to compare the results from each stage to check whether the data has changed or not. Significant changes to the current architecture would be required such as adding additional signals that access the pipeline stages for comparison. The detection of errors that occur between the pipeline stages would improve.

The current fault injection method is only functional for the Polarfire FPGA because of the added controller for controlling the data and write signal added in the RAM model. Expanding the same functionality to other RAM models for FPGAs would provide the ability to see whether different effects appear.

Regarding the generation of bit-masks, an interface capable of communicating with external programs such as MATLAB or PYTHON would allow for the statistical modelling of the bit-masks allowing for further testing. The requirement to support this would be to extend the fault injection to the different versions of the NOEL-V models with added protections such as ECC. Providing support for larger bit-masks would provide the opportunity to test the system fully.

7

Conclusion

The thesis evaluates fault tolerance on a COTS SRAM-FPGA using the NOEL-V processor. Different fault tolerance methods are presented to find the best approach for comparison with the base design and the conventional triple modular redundancy. The project also develops a concept of fault injection to test the capabilities of the implemented lockstep version that could be implemented at any level within the processor.

We compared NOEL-V with two fault tolerance methods: the conventional TMR method with ECC and a lockstep version. In this comparison, the lockstep implementation outperformed TMR regarding the clock rate, resource utilisation, and power consumption.

The current lockstep implementation has limitations such as not detecting faults in the lower parts of the processor because it only monitors signals higher in the processor's bus. The fault injection method is limited to testing with only 4 bits of data preventing the maximum width of the registers from being used. The fault injection method is constrained to the specific FPGA board used in the project limiting its general applicability to other FPGAs. The implication for using COTS SRAM FPGAs in space is the required testing to see whether the deployed system can withstand SEUs from affecting the standard functionality while leveraging the performance benefits.

For future work, the current lockstep implementation would require improvements to target the lower parts of the bus that connect the cache, pipeline, and the memory controllers. Furthermore, the fault injection should be expanded to other FPGA technologies to enable a more comprehensive evaluation of the fault tolerance method's capabilities. Finally, it would be worthwhile to investigate the system's behaviour with added ECC in memory when faults occur to further test the ability to detect and correct errors.

Bibliography

[1] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of radiation effects in SRAM-based FPGAs for space applications," *ACM Comput. Surv.*, vol. 47, no. 2, jan 2015, doi: 10.1145/2671181.

[2] E. Canessa, "High-energy physics fault tolerance metrics and testing methodologies for SRAM-based FPGAs," M.S. thesis, Politecnico di Torino, 2018.

[3] Á. B. de Oliveira *et al.*, "Evaluating soft core RISC-V processor in SRAM-based FPGA under radiation effects," *IEEE Transactions on Nuclear Science*, vol. 67, no. 7, pp. 1503–1510, 2020, doi: 10.1109/TNS.2020.2995729.

[4] M. Rogenmoser, N. Wistoff, P. Vogel, F. Gürkaynak, and L. Benini, "On-demand redundancy grouping: Selectable soft-error tolerance for a multi-core cluster," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 398–401, doi: 10.1109/ISVLSI54635.2022.00089.

[5] Á. B. de Oliveira *et al.*, "Applying lockstep in dual-core ARM Cortex-A9 to mitigate radiation-induced soft errors," in *2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS)*, 2017, pp. 1–4, doi: 10.1109/LASCAS.2017.7948063.

[6] F. Bas *et al.*, "SafeDE: a flexible diversity enforcement hardware module for light-lockstepping," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021, pp. 1–7, doi: 10.1109/IOLTS52814.2021.9486715.

[7] E. Chielle *et al.*, "Reliability on ARM processors against soft errors through SIHFT techniques," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2208–2216, 2016, doi: 10.1109/TNS.2016.2525735.

[8] T. Noergaard, Ed., *Embedded Systems Architecture*, 2nd ed. Newnes, 2013, pp. 153–172, doi: 10.1016/B978-0-12-382196-6.00004-2.

[9] J. Andersson, "Development of a NOEL-V RISC-V SoC targeting space applications," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 66–67, doi:10.1109/DSN-W50199.2020.00020.

[10] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V instruction set manual," *Volume I: User-Level ISA', version*, vol. 2, 2014.

[11] Frontgrade Gaisler, "GRLIB IP core user's manual," 2022, accessed: 25-Apr-2023. [Online]. Available: https://www.gaisler.com/products/grlib/grip.pdf

[12] ARM, "AMBA specification (ver. 2.0)," 1999, accessed: 27-Apr-2023. [Online]. Available: https://developer.arm.com/documentation/ihi0011/a/

[13] ——, "AMBA APB product specification," 2003, accessed: 27-Apr-2023. [Online]. Available: https://developer.arm.com/documentation/ihi0024/latest/

[14] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao, "How to build a benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 333–336, doi: 10.1145/2668930.2688819. [Online]. Available: https://doi.org/10.1145/2668930.2688819

[15] K. L. John and L. Eeckhout, *Performance Evaluation and Benchmarking*, 1st ed. CRC Press, 2006, pp. 25–43, doi: 10.1201/9781315220505.

[16] A. Joshi, L. Eeckhout, and L. John, "The return of synthetic benchmarks," in *2008 SPEC Benchmark Workshop*, 2008, pp. 1–11.

[17] H. Yang, J. Zhang, J. Sun, and L. Yu, "Review of advanced FPGA architectures and technologies," *Journal of Electronics (China)*, vol. 31, no. 5, pp. 371–393, Oct 2014, doi: 10.1007/s11767-014-4090-x.

[18] E. Tlelo-Cuautle, J. d. J. Rangel-Magdaleno, and L. G. De la Fraga, *Engineering Applications of FPGAs: Chaotic Systems, Artificial Neural Networks, Random Number Generators, and Secure Communication Systems*. Cham: Springer International Publishing, 2016, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-319-34115-6_1

[19] H. Quinn, "Radiation effects in reconfigurable fpgas," *Semiconductor Science and Technology*, vol. 32, no. 4, p. 044001, Mar 2017, doi: 10.1088/1361-6641/aa57f6.

[20] M. Iida, *Principles and Structures of FPGAs*. Singapore: Springer Singapore, 2018, pp. 23–45, doi: 10.1007/978-981-13-0824-6_2. [Online]. Available: https://doi.org/10.1007/978-981-13-0824-6_2

[21] R. Druyer, L. Torres, P. Benoit, P. Bonzom, and P. Le-Quere, "A survey on security features in modern FPGAs," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2015, pp. 1–8, doi: 10.1109/ReCoSoC.2015.7238102.

[22] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, pp. 135–253, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:51557856

[23] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015, doi: 10.1109/JPROC.2015.2404212.

[24] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, and K. Velusamy, "Understanding radiation effects in SRAM-based field programmable gate arrays for implementing instrumentation and control systems of nuclear power plants,"

Nuclear Engineering and Technology, vol. 49, no. 8, pp. 1589–1599, 2017, doi: 10.1016/j.net.2017.09.002.

[25] F. L. Kastensmidt, L. Carro, and R. A. da Luz Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*. Boston, MA: Springer US, 2006, pp. 13–17, doi: 10.1007/978-0-387-31069-5.

[26] D. White, "Considerations surrounding single event effects in FPGAs, ASICs, and processors," *Xilinx White Paper*, 2012.

[27] D. E. Johnson, K. Morgan, M. J. Wirthlin, M. P. Caffrey, and P. S. Graham, "Detection of configuration memory upsets causing persistent errors in SRAM-based FPGAs," in *7th Annual Military and Aerospace Programmable Logic Devices International Conference*. NASA Office of Logic Design, 2004.

[28] N. Rollins, "Evaluating TMR techniques in the presence of single event upsets," *Proc. Military and Aerospace Programmable Logic Devices, Sept. 2003*, 2003. [Online]. Available: https://cir.nii.ac.jp/crid/1571980075963082752

[29] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A comparison of TMR with alternative fault-tolerant design techniques for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, 2007, doi: 10.1109/TNS.2007.910871.

[30] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*, 2006, pp. 226–232, doi: 10.1109/RELPHY.2006.251221.

[31] V. Vlagkoulis *et al.*, "Configuration memory scrubbing of SRAM-based FPGAs using a mixed 2-D coding technique," *IEEE Transactions on Nuclear Science*, vol. 69, no. 4, pp. 871–882, 2022, doi: 10.1109/TNS.2022.3151977.

[32] Y. Ichinomiya *et al.*, "Improving the robustness of a softcore processor against SEUs by using TMR and partial reconfiguration," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 47–54, doi: 10.1109/FCCM.2010.16.

[33] A. M. Keller and M. J. Wirthlin, "Partial TMR for improving the soft error reliability of SRAM-based FPGA designs," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1023–1031, 2021, doi: 10.1109/TNS.2021.3070856.

[34] A. J. Sánchez-Clemente, L. Entrena, and M. García-Valderas, "Partial TMR in FPGAs using approximate logic circuits," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2233–2240, 2016, doi: 10.1109/TNS.2016.2541700.

[35] S. Kasap, E. W. Wächter, X. Zhai, S. Ehsan, and K. D. McDonald-Maier, "Novel lockstep-based fault mitigation approach for SoCs with roll-back and roll-forward recovery," *Microelectronics Reliability*, vol. 124, p. 114297, 2021, doi: 10.1016/j.microrel.2021.114297.

[36] F. Abate, L. Sterpone, C. A. Lisboa, L. Carro, and M. Violante, "New techniques for improving the performance of the lockstep architecture for SEEs mitigation in FPGA embedded processors," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 1992–2000, 2009, doi:10.1109/TNS.2009.2013237.

[37] F. Abate, L. Sterpone, and M. Violante, "A new mitigation approach for soft errors in embedded processors," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2063–2069, 2008, doi: 10.1109/TNS.2008.2000839.

[38] E. W. Wächter, S. Kasap, X. Zhai, S. Ehsan, and K. McDonald-Maier, "Survey of lockstep based mitigation techniques for soft errors in embedded systems," in *2019 11th Computer Science and Electronic Engineering (CEEC)*, 2019, pp. 124–127, doi: 10.1109/CEEC47804.2019.8974333.

[39] Á. B. de Oliveira *et al.*, "Lockstep dual-core ARM A9: Implementation and resilience analysis under heavy ion-induced soft errors," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1783–1790, 2018, doi: 10.1109/TNS.2018.2852606.

[40] J. R. Azambuja, F. Sousa, L. Rosa, and F. L. Kastensmidt, "The limitations of software signature and basic block sizing in soft error fault coverage," in *2010 11th Latin American Test Workshop*, 2010, pp. 1–8, doi: 10.1109/LATW.2010.5550346.

[41] O. Goloubeva, M. Rebaudengo, M. Reorda, Sonza, and M. Violante, *Software-Implemented Hardware Fault Tolerance*. Boston, MA: Springer US, 2006, pp. 37–62, doi:10.1007/0-387-32937-4_2.

[42] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Soft-error detection using control flow assertions," in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 581–588, doi:10.1109/DFTVS.2003.1250158.

[43] M. Nikseresht, J. Vankeirsbilck, D. Pissoort, and J. Boydens, "A selective soft error protection method for COTS processor-based systems," in *2021 XXX International Scientific Conference Electronics (ET)*, 2021, pp. 1–5, doi: 10.1109/ET52713.2021.9579862.

[44] Microchip, "Polarfire splash kit," 2023, accessed: 12-Feb-2023. [Online]. Available: https://www.microchip.com/en-us/development-tool/MPF300-SPLASH-KIT#

[45] ——, "MPF300T: Lowest power, cost-optimised, mid-range fpgas," 2023, accessed: 14-Feb-2023. [Online]. Available: https://www.microchip.com/en-us/product/MPF300T

[46] Frontgrade Gaisler, "GRMON3," 2023, accessed: 13-Feb-2023. [Online]. Available: https://www.gaisler.com/index.php/products/debug-tools/grmon3

[47] L. Lavagno, I. L.Markov, G. Martin, and L. K. Scheffer, Eds., *Electronic Design Automation for IC System Design, Verification, and Testing*, 2nd ed. CRC Press, 2016, vol. 1, ch. 10.6.4.

[48] A. R. Weiss, "Dhrystone benchmark," *History, Analysis,„Scores "and Recommendations, White Paper, ECL/LLC*, 2002, accessed: 10-Mar-2023. [Online]. Available: https://johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf

[49] Synopsys, "Synplify pro for microsemi edition user guide," 2020, [Online], Accessed: 09-Feb-2023.

[50] Microchip, “Libero® design flow user guide,” 2023, accessed: 09-Feb-2023. [Online]. Available: https://onlinedocs.microchip.com/pr/GUID-AE5CCA5C-A93D-4362-B6A3-D684C83E863C-en-US-4/index.html?GUID-D3BE74A4-8846-442A-9F67-834757001FE2

[51] Frontgrade Gaisler, “NOEL-V brief„” 2023, accessed: 10-Feb-2023. [Online]. Available: https://www.gaisler.com/products/noel-v/Product%20Brief_NOEL-V_20230301_FOR%20WEB.pdf