Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (37^{th}cycle)

AI-oriented Design Methods for Intelligent Aerospace Computing Systems

By

Andrea Portaluri
******

Supervisor:
Prof. Luca Sterpone

Doctoral Examination Committee:
Prof. Juan Antonio Clemente, Universidad Complutense de Madrid
Prof. Luca Fanucci, Università di Pisa
Prof. Lana Josipovic, ETH Zurich
Prof. Bartolomeo Montrucchio, Politecnico di Torino
Prof. Carsten Trinitis, TUM

Politecnico di Torino
2025

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Andrea Portaluri
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

To my loving parents, mamma e papà, for being my best source of inspiration.

Acknowledgements

I am immensely grateful to everyone who has supported me throughout this journey. I would like to express my sincere appreciation to my supervisor, Prof. Luca Sterpone, for allowing me to work on my research and being an exceptional guide, mentor, and source of motivation throughout this period. I am deeply grateful to Prof. Sarah Azimi and Dr. Corrado De Sio who have been an endless source of advice and inspiration for me over these years and from whom I have learned more than any other. I would like to thank Prof. Matteo Sonza Reorda and the CAD group for the opportunity they have given me to work with them and my lab colleagues for the invaluable help they have provided me all these years. Finally, I would like to thank my entire family for the support and encouragement they have given me over the years and throughout my whole life. I want to thank my friends for being exactly the people I needed in my life.
Thank you all.

Abstract

This dissertation explores methodologies and techniques to enable the development, evaluation, and optimization of radiation-hardened Field Programmable Gate Arrays (FPGAs) for high-performance computing in radiation-intense environments. A comprehensive suite of tools is introduced to enhance both reliability and computational efficiency, addressing the challenges posed by Single Event Upsets and other radiation-induced phenomena.

The research contributions span multiple levels of abstraction, from device-level characterization to system-level fault tolerance, employing fault injection, accelerated irradiation testing, and analytical modeling. Dedicated analysis frameworks have been developed to assess the sensitivity of FPGA designs to SEUs, with fault emulation platforms and irradiation campaigns targeting SRAM- and FinFET-based devices. These studies have revealed critical insights into the resilience and performance trade-offs inherent in FPGA-based systems, highlighting the limitations of traditional mitigation techniques such as redundancy and scrubbing.

Particular emphasis has been placed on soft-core processors, custom accelerators, and data-intensive applications, including artificial intelligence workloads, where fault propagation was shown to significantly impact both throughput and correctness. Interconnect reliability, especially in AXI-based data transfer modules, has also been rigorously evaluated, with results demonstrating that silent data corruption remains a major concern despite redundancy.

The outcomes of this work culminate in a framework dedicated to the analysis of European radiation-hardened FPGA platforms, laying the foundation for future research on placement and routing strategies tailored to these architectures.

This dissertation aims to advance the state of the art in radiation-tolerant reconfigurable computing, providing methodologies and insights that support the reliable deployment of FPGAs in aerospace and other high-reliability domains.

Contents

List of Figures **x**

List of Tables **xiii**

Nomenclature **xv**

1 Introduction **1**

1.1 Motivations 1

1.2 Aims of the Thesis 2

1.3 Contributions 3

1.4 Thesis Organization 3

I Context and Background **5**

2 Reconfigurable Devices **7**

2.1 Introduction 7

2.2 Programmable Logic and Interconnections 8

2.3 Design Flow 9

2.4 Reconfigurable System-on-Chips 11

3 Radiation-Induced Effects on SRAM-based Devices **13**

3.1 Introduction 13

3.2 Radiation Environments 14
3.2.1 Space Environment 14
3.2.2 High-Altitude and Avionics Radiation Environment 14
3.2.3 Terrestrial Radiation Environment 15
3.3 Single-Event Effects 15
3.4 Summary 17

4 Radiation-Hardened Technologies 19
4.1 Introduction 19
4.2 Design-Level Hardening 20
4.2.1 Triple Modular Redundancy 20
4.2.2 Configuration Scrubbing 20
4.2.3 Error Detection and Correction 21
4.2.4 Low Power and Radiation-Tolerant Clocking 21
4.3 Intrinsic Hardness: CMOS vs. FinFET 21
4.3.1 Experimental Setup 23
4.3.2 Test Results and Analysis 25
4.4 Process and Material Hardening 27

II Radiation-Induced Effects: Analysis and Observations 29

5 SEUs Analysis Methodologies 31
5.1 Introduction 31
5.2 Fault Injection Methods 32
5.3 Evaluation of Soft-Processors Reliability 34
5.3.1 State of the Art of Soft-processors Reliability 34
5.3.2 Experimental Flow for Soft-Processors Reliability Analysis 36

5.3.3 Results of Fault Injections on Hardened Soft-Processor . . . 39
5.4 Evaluation of Data Transfer in Programmable Logic 43
5.4.1 State of the Art of AXI Reliability 43
5.4.2 Experimental Flow for AXI DMA Reliability Analysis . . . 44
5.4.3 Experimental Results of Fault Injection on DMA 48

III Design Techniques for Reliable Computing in Space 55

6 Design Methods for COTS FPGAs in Space 57
6.1 Analysis and Mitigation of Cross-Domain Errors 57
6.1.1 State of the Art of Cross-Domain Failure Mitigation 58
6.1.2 Methodology of the Experiment 59
6.1.3 Experimental Setup 64
6.1.4 Results of Cross-Domain Errors Mitigation 66
6.2 Evaluation of Domain-based Isolation 70
6.2.1 Methodology of the Experiment 71
6.2.2 Comparison between Plain Design and Domain-based Isolation Results 74

7 Design Methods for Rad-Hard FPGAs in Space 79
7.1 Toward Rad-Hard FPGA-based High-Performance Computations . . 79
7.1.1 State of the Rad-Hard FPGAs and Tools 80
7.1.2 Proposed Model of NanoXplore Architecture 83
7.1.3 NXRouting: A Placement & Routing Tool for NanoXplore . 88
7.1.4 NXRouting Performances and Timings 93
7.2 Performance Analysis 93
7.3 Evaluation of a Neural Network Accelerator on Rad-Hard FPGA . . 95

7.3.1 The r-VEX Architecture 95
7.3.2 Convolution in VEX Assembly 96
7.3.3 Experimental Analysis and Results 98

8 Conclusions and Future Directions 108

8.1 Conclusions 108
8.2 Future Directions 109

Bibliography 112

List of Figures

2.1 FPGA design flow, achieving the generation of the bitstream. 12

4.1 Block scheme of the benchmark. 24

4.2 Cross section comparison between CMOS and FinFET. 25

4.3 SEMU patterns of CMOS and FinFET technologies. 26

5.1 View of the baseline and hardened version of the software. 37

5.2 Comparison between baseline and hardened software reliability against injected SEUs. 42

5.3 Examples of errors in DMA due to radiation-induced bitflips. (a) Corruption of the BD causing wrong output products. (b) Corruption of routing bits causing output unavailability. 45

5.4 Experiment flow for radiation experiment. 47

5.5 Particle cross-section per DMA and CORDIC. 50

5.6 Cross-section of MBU patterns. 51

5.7 Cross-section of common-mode errors. 52

5.8 Failure rate with the accumulation of fault injections and radiation test data. 54

6.1 Representation of the isolation concept used to reduce common resources among different modules. 61

6.2 Simplified view of an isolated design. The sum of CLBs in yellow represents the isolation overhead (R_{tot}) 62

6.3 Isolation resources overhead vs. number of cores for partial and fully isolated layouts with different q values and fixed parameters. 63

6.4 Vivado Implementation views of the three layouts: (a) Default (b) Matrix (c) Columns. A different color is given to each of the cores. . 67

6.5 SEU cross section per unmasked CRAM bit for the three layouts (95% confidence error bars). 68

6.6 Heatmaps of particle cross-section per core ($\cdot 10^{-11}$ cm^2) of the three layouts with 95% confidence intervals. 69

6.7 Block scheme of the Standard design. 76

6.8 Block scheme of the domain-based design. 76

6.9 Block scheme of the non-domain-based design. 76

7.1 (a) Hierarchical structure of the NanoXplore programmable logic. (b) Example of `Device:LUT315` and `Plug:I1` in the NG-MEDIUM variant. 84

7.2 Example of Networks and Devices distribution within a Tile zone. . 85

7.3 Detail of a Tile in the NG-MEDIUM architecture with focus on the FE and its components. The Tile includes also additional logic such as Carry logic (CY), High-performance LUT (X-LUT), Register File and others. 86

7.4 The figure explains the difference between the two routing models inside a Tile. (a) Routing by means of switch matrices, where the delay is dependent on the physical distances among the routed logic blocks; (b) Routing by means of shuffling matrix, where the delay only roughly depends on how many times the route enters the matrix. 87

7.5 Example of architectural exploration through pseudo-Python of the NXRouting tool. This feature mainly focuses on the user experience, presenting a well-known objected-oriented interface. 89

7.6 Scheme of NXRouting showing the features and the databases associated. 92

7.7 a) Scheme of r-VEX architecture. b) Different instructions supported by the VEX ISA. 97

7.8 Architecture of the AlexNet and its sub-section selected for the acceleration through the r-VEX. 97

7.9 Plot of the data delay of the 10 longest paths of the 1-core design compared with the same paths of the other design. The dotted lines represent the average values. 103

7.10 Plot of the data delay of the 10 longest paths of the 1-core design compared with the same paths of the other design with timing constraints. The dotted lines represent the average values. 104

7.11 Cores layouts analyzed for a) 1-core b) 2-core c) 3-core. Each colored block represents a constrained region for a single core. . . . 106

7.12 Plot of the data delay of the 10 paths analyzed in Section V.C of the constrained layouts. L1 versions minimize the distance from the output buffer, while L2 maximizes it. 107

List of Tables

3.1 Space radiation environments and dominant particle populations . . 18

3.2 Single Event Effects (SEE) and their impact on electronics 18

4.1 FPGA utilization for the Benchmark Circuits 24

5.1 FPGA utilization for the PULPissimo Soft Processor implemented on Nexys Video (Artix-7) 37

5.2 Error categorization resulting from fault injection campaigns on baseline software 41

5.3 Error categorization resulting from fault injection campaigns on hardened software 41

5.4 Zynq Resource Utilization 49

5.5 Radiation test conditions: energy, flux and fluence. 49

5.6 Particle cross-section per DMA and CORDIC 49

6.1 Resources utilization and Essential Bits for Matrix, Columns, and Default layouts on ZCU104 67

6.2 Essential bits of standard and IDF configurations 74

6.3 Distribution of errors for standard and IDF designs (10,000 injections) 74

6.4 Resource utilization for standard and IDF designs 75

6.5 EB of standards, domains-based, and non-domains-based configurations 77

6.6 Resources Utilization for Standard and Isolated Designs 77

6.7 Distribution of errors for the standard TMR, domains-based and non-domains-based configurations for 10,000 SEUs 78

7.1 Timing comparison between Full Loading and Request Loading configurations. 93

7.2 Number of nets to be routed for each benchmark 94

7.3 Routing time comparison across different router configurations. . . . 95

7.4 Parameters of the Implemented Network Layers 98

7.5 Implemented r-VEX Configurable Parameters 100

7.6 Resource Utilization of Each Design 101

7.7 Average Transition Timing of Internal Routing Resources 102

7.8 Average Transition Timing of General Routing Resources 102

7.9 Average Transition Timing of Low Skew Routing Resources 102

7.10 Maximum Operating Frequency for Each Design 105

Nomenclature

Acronyms / Abbreviations

AI Artificial Intelligence

AXI Advanced eXtensible Interface

BRAM Block RAM

CAD Computer-Aided Design

CLB Configurable Logic Block

CMOS Complementary Metal-Oxide-Semiconductor

CORDIC Coordinate Rotation Digital Computer

COTS Commercial Off-The-Shelf

CRAM Configuration RAM

DMA Direct Memory Access

DSP Digital Signal Processor

DUT Device Under Test

EDA Electronic Design Automation

ELF Executable and Linking Format

FF Flip-Flop

FPGA Field-Programmable Gate Array

GEO Geostationary Orbit

GPU Graphic Processor Unit

HDL Hardware Design Language

HLS High-Level Synthesis

HPC High-Performance Computer

IOB Input/Output Block

IP Intellectual Property

LEO Low Earth Orbit

LUT Look-Up Table

MBU Multi-Bit Upset

RAM Random Access Memory

SEE Single-Event Effects

SEFI Single-Event Functional Interrupt

SEL Single-Event Latch-up

SEMU Single Event Multiple Upset

SET Single-Event Transient

SEU Single Event Upset

SoC System-on-Chip

SRAM Static RAM

TID Total Ionizing Dose

Chapter 1

Introduction

1.1 Motivations

Today, computer systems play a crucial role in shaping our society. Since the invention of the transistor, these systems have rapidly permeated nearly every aspect of human life and culture. To address the unique needs and requirements of various industries and applications, they have evolved into a diverse array of computational systems built upon distinct architectures and paradigms.

From the earliest days of computing, ensuring system reliability has been a critical concern to ensure proper functionality. This focus intensified since the beginning of the first space programs and became a key feature with the advent of the so-called New Space Era, where the stringent reliability demands for manned missions and programs led to the need for advanced reliability assurance techniques, while keeping an eye on the budget. This term refers to the contemporary transformation of the space sector characterized by the rise of private companies, reduced launch costs, and a shift from purely government-driven programs to a dynamic, market-oriented ecosystem. Unlike the traditional space age, now the market is also fueled by commercial innovators which focus on reusability and rapid development. This new paradigm emphasizes accessibility, sustainability, and economic competitiveness in space, enabling unprecedented opportunities.

As the demand for robust and reliable systems grew, there was a parallel push for high-performance systems capable of processing vast amounts of data, performing real-time operations, and tackling complex tasks efficiently. This also accelerated,

among the others, the integration of Artificial Intelligence (AI)-based applications, enabling advancements such as predictive analytics, autonomous systems, and intelligent decision-making frameworks, which further expanded the potential of computer systems across industries.

The advent of hardware-reconfigurable devices and Systems-on-Chip has marked a significant milestone in computing, enabling the creation of systems that combine high performance, low power consumption, and exceptional flexibility. These systems have found applications across a wide range of fields, including space exploration, healthcare, automotive, and beyond.

However, the architecture of these systems introduces vulnerabilities and a heightened sensitivity to faults and errors. Furthermore, the inherent complexity of such heterogeneous systems poses significant challenges for evaluating their robustness. Despite these challenges, ensuring fault tolerance and robustness remains a vital design objective, particularly for safety-critical systems and applications.

This need is especially pronounced in space applications, which demand systems capable of enduring extreme conditions and operating reliably in radiation-rich environments over extended periods. Similarly, equipment used in radiation facilities, as well as critical systems in the automotive and aerospace industries, requires stringent reliability standards. Field Programmable Gate Arrays and Reconfigurable Systems-on-Chip offer a promising solution for these scenarios due to their reprogrammability, which supports adaptation to evolving requirements and working parameters, extends device longevity, and facilitates in-field updates.

1.2 Aims of the Thesis

The primary aim of this thesis is to address the challenges posed by radiation-induced Single-Event Upsets (SEUs) on Static Random Access Memory (SRAM)-based devices, with a particular focus on their application-level behavior. The study begins by conducting detailed analyses and radiation tests to evaluate the impact of radiation on these devices, especially in systems that handle and compute large volumes of data in parallel. This emphasis is motivated by the need to mimic the operational patterns of AI-based applications, which often require significant computational power and efficient data management.

The objective then shifts to radiation-hardened devices as a potential alternative to mitigate radiation-induced issues. While these devices are designed to enhance reliability in harsh environments, they often struggle to achieve performance levels comparable to their Commercial Off-The-Shelf (COTS) counterparts. This work aims to explore and, possibly, to unlock the potential of these devices, enabling the development of high-end applications on radiation-hardened chips.

1.3 Contributions

To bridge the performances gap, the thesis proposes a series of optimizations and tools aimed at enabling radiation-hardened Field-Programmable Gate Arrays (FPGAs) to effectively support AI-based applications and other tasks demanding high computational power. These innovations are designed to balance reliability and performance, ensuring that radiation-hardened devices can meet the stringent requirements of safety-critical applications without compromising their ability to process complex workloads efficiently.

By addressing these challenges, the thesis contributes to the development of more robust and high-performing computational systems for use in radiation-rich environments, such as space exploration, avionics, and other safety-critical industries. The findings and solutions presented in this work aim to pave the way for broader adoption of radiation-hardened FPGAs in advanced computational tasks, ultimately enhancing their applicability in both current and future technological landscapes.

1.4 Thesis Organization

The organization of this thesis is structured into three main parts, each addressing a distinct aspect of the research.

Part I provides the foundational context for the work presented in the subsequent sections. Specifically, it introduces the paradigm of reconfigurable hardware and the effects of radiation on SRAM-based devices and radiation-hardened devices. These topics are covered in Chapters 2, 3, and 4, respectively.

Part II focuses on the methodologies and experiments conducted on reprogrammable devices to investigate the impact of radiation-induced effects. It examines how these effects influence high-computational power applications. This section includes results from radiation test campaigns and analyses of different SRAM-based technologies. In this context, Chapter 5 emphasizes experiments with radiation campaigns and hardware-based faults emulation.

Part III centers on optimizing the design of COTS and radiation-hardened devices for demanding applications in space, proposing methods and analysis tools. Chapter 6 presents performance analyses of techniques focusing COTS devices, while Chapter 7 details the development of a tool for architecture and netlist exploration and analysis tailored to the first European radiation-hardened FPGAs and its applications.

Finally, Chapter 8 concludes the thesis, summarizing the main findings and discussing potential future developments to enhance the performance of radiation-hardened devices for AI-based applications.

Part I

Context and Background

Chapter 2

Reconfigurable Devices

2.1 Introduction

SRAM-based FPGAs have become a cornerstone of modern digital design, offering unparalleled time-to-market, performance-over-costs ratio and flexibility for a wide range of applications. These devices are built around an array of Configurable Logic Blocks (CLBs) connected through programmable interconnects, with their functionality defined by configuration data stored in SRAM cells [1, 2]. This unique architecture allows designers to implement complex digital circuits and reprogram them as needed, enabling rapid prototyping, customization, and in-field updates.

Over the past decade, SRAM-based FPGAs have undergone significant technological advancements, driven by the demand for higher performance, lower power consumption, and greater versatility. The integration of cutting-edge process technologies has allowed for the development of FPGAs with increased logic density, faster clock speeds, and reduced power requirements. These improvements have positioned SRAM-based FPGAs as key enablers for applications ranging from telecommunications and data centers to automotive systems and AI.

One of the most notable advancements in recent years has been the incorporation of heterogeneous elements into FPGA architectures [3]. Modern SRAM-based FPGAs now include dedicated hardware blocks such as hardwired processors, Digital Signal Processors (DSPs), high-speed transceivers, and memory interfaces. These enhancements allow FPGAs to handle computationally intensive tasks, such as real-time data processing and AI inference, with greater efficiency. Furthermore, in-

novations in interconnect design and routing algorithms have significantly improved the speed and scalability of these devices, enabling them to meet the demands of high-performance computing and parallel processing.

In the following subsections, the building blocks, architecture, and design flow of SRAM-based FPGAs will be presented in detail. Additionally, an introduction to reconfigurable System-on-Chips (SoCs) and the integration of embedded processors with the FPGA resource matrix will be provided.

2.2 Programmable Logic and Interconnections

The architecture of SRAM-based FPGAs is designed to offer a high degree of flexibility and configurability, enabling the implementation of diverse digital circuits tailored to specific application requirements. At the core of this architecture are CLBs, programmable interconnects, and Input/Output Blocks (IOBs). These components work together to create a versatile and reprogrammable hardware platform.

CLBs are the fundamental building blocks of an FPGA [4]. Each CLB generally contains a set of Look-Up Tables (LUTs), Flip-Flops (FFs), and multiplexers. The LUTs are used to implement combinational logic, while the flip-flops store state information for sequential circuits. Multiplexers provide flexibility in routing signals within the CLB. By configuring these elements, designers can implement a wide range of logic functions and circuit behaviors.

IOBs provide the interface between the FPGA and external devices or systems. They support various signaling standards and are configurable to meet the voltage, speed, and direction requirements of different applications. IOBs also include dedicated circuitry for high-speed serial communication, which is critical for data-intensive tasks.

In addition to the basic architecture, SRAM-based FPGAs often include specialized hardware blocks such as DSP slices, embedded memory, and high-speed transceivers. These elements are tightly integrated with the CLBs and interconnects to support computationally intensive and data-centric applications.

The programmable interconnects form the communication backbone of an FPGA, connecting CLBs, IOBs, and other specialized blocks. These interconnects consist of a dense network of wires and programmable switches that can be configured to

establish specific signal paths. Advanced routing algorithms are employed during the design process to optimize these connections for performance, power efficiency, and resource utilization. Modern FPGAs also include hierarchical interconnect structures, such as local, regional, and global routing networks, to balance flexibility and scalability [5].

All the above blocks are configured by means of a binary data array, called bitstream, which is stored in a volatile configuration memory (or CRAM). The bitstream is a critical component of SRAM-based FPGAs, as it encodes the data required to define the functionality of the device. This data includes information about the logic implemented in the CLBs, the routing of interconnects, the settings of IOBs and other specialized blocks. As volatile memory, the CRAM requires a constant power supply to retain its contents, making the device susceptible to data corruption in the event of power and voltage-related fluctuations (e.g., radiation-induced effects). To address this, several are the available approaches including radiation-hardened devices and mitigation techniques, later discussed in this manuscript.

2.3 Design Flow

The design flow of FPGAs is a structured process that transforms a high-level design concept into a fully functional hardware implementation on the programmable fabric. It is widely supported by advanced vendor and third pary Electronic Design Automation (EDA) and Computer-Aided Design (CAD) tools, which guide designers through each step while ensuring optimal performance, resource utilization, and reliability. The FPGA design flow can be divided into several key stages:

Design Entry

This initial stage involves defining the intended functionality of the FPGA. Designers can describe their designs using Hardware Description Languages (HDLs), which provide a precise, low-level specification of digital circuits. Alternatively, High-Level Synthesis (HLS) tools allow designers to use high-level programming languages but they are out of the scope of this thesis. During this phase, constraints such as timing, power, and area requirements are often defined to guide the subsequent design stages.

Synthesis

In the synthesis phase, the HDL or high-level design is converted into a gate-level netlist. The netlist represents the logical components and their interconnections, abstracted from the physical FPGA resources. Synthesis tools perform extensive optimization to balance trade-offs between resource usage, performance, and power consumption. Advanced algorithms are used to minimize the logic required for the design while ensuring adherence to constraints. The resulting netlist is a critical intermediary that forms the basis for the later mapping and placement steps.

Mapping and Optimization

The synthesized netlist is mapped onto the FPGA resources, including LUTs, FFs, and embedded hardware blocks such as DSPs and embedded memories like Block RAMs (BRAMs). This process involves translating logical components into the specific physical resources available on the target FPGA. During mapping, tools analyze the design to maximize the utilization of specialized blocks, such as DSP slices for arithmetic operations or BRAMs for memory-intensive tasks. Optimization techniques are applied to reduce resource contention and improve the overall performance of the design, ensuring that critical paths are kept as short as possible.

Placement

Placement determines the physical location of each mapped component on the FPGA fabric. The goal is to reduce signal delays and ensure efficient utilization of the FPGA resources. Placement algorithms carefully consider timing constraints, resource availability, and connectivity to achieve an optimal arrangement. For example, components that communicate frequently are placed closer together to minimize routing delays. Modern placement tools may also leverage machine learning and heuristic techniques to handle the complexity of large designs, achieving superior performance and reliability.

Routing

After placement, the design enters the routing phase, where the interconnections between components are established. Programmable interconnects are configured to create paths for signals, adhering to constraints such as timing, signal integrity, and resource limitations. Routing tools ensure that signals are routed efficiently, minimizing congestion and delays. Advanced routing algorithms can adapt to highly congested areas of the design, finding alternative paths to maintain performance. This phase is crucial, as the quality of routing directly impacts the maximum achievable clock frequency of the design.

Bitstream Generation

Once the design is fully verified and optimized, the final step is to generate the bitstream. The bistream generation engine is responsible to encode all the configuration data according the vendor policy. This data defines the logic, interconnects, and settings of the FPGA components. The bitstream is then loaded into the device CRAM through an external programming interface. Modern tools allow for partial bitstream generation, enabling updates or modifications to specific regions of the FPGA without affecting the rest of the design.

Although these steps are common among all kinds of FPGAs, their execution is not always straightforward as they may require preemption, iterative tries or additional substeps to achieve the optimized solution. Figure 2.1 summarizes all the FPGA design flow steps.

2.4 Reconfigurable System-on-Chips

A SoC is an integrated circuit designed to incorporate an entire system onto a single chip. Like most emerging technologies, SoCs have evolved significantly over time. Early designs featured simple architectures with a limited number of components, such as basic microcontrollers, minimal memory, and a few peripherals. Today, modern SoCs are high-performance systems that integrate multiple microprocessors, on-chip memory, accelerators such as Graphic Processor Units (GPUs), FPGAs, controllers, network-on-chip (NoC) architectures, and more.

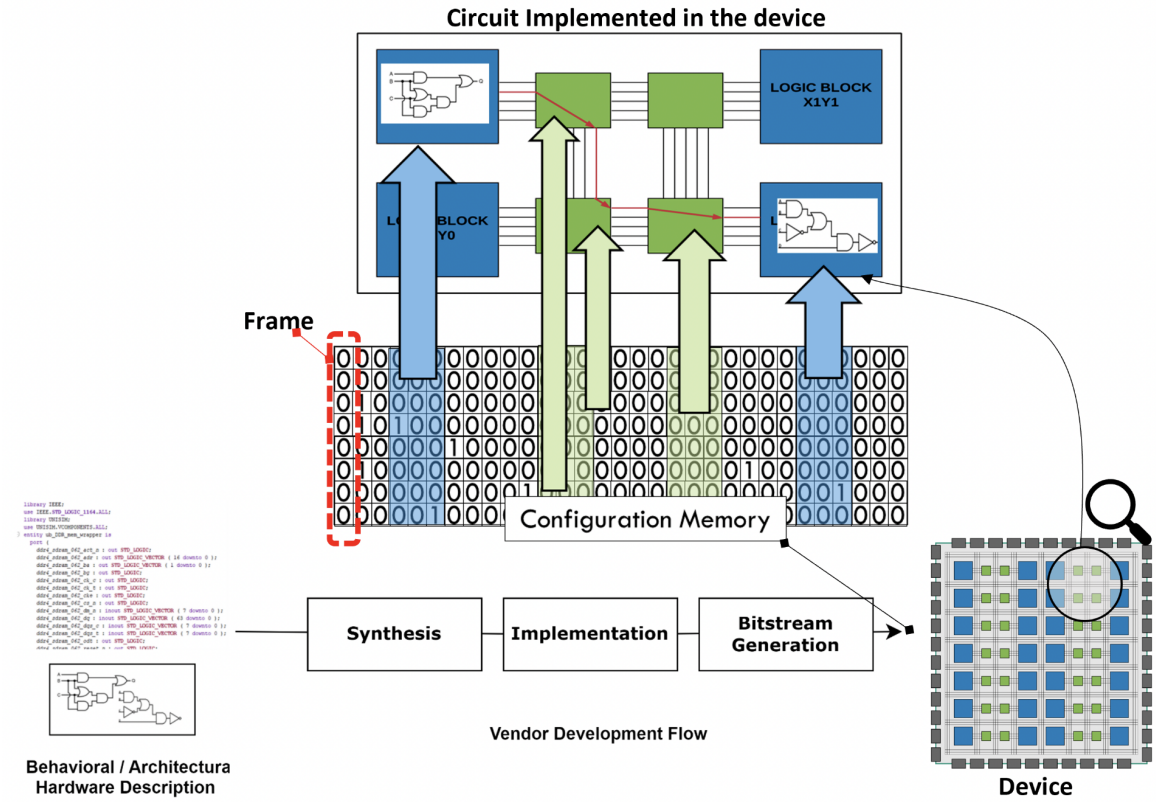

Fig. 2.1 FPGA design flow, achieving the generation of the bitstream.

Hardware-Reconfigurable System-on-Chip devices extend this concept by including an FPGA-like section within the same chip, alongside processors and memory. This FPGA portion is typically utilized as a hardware accelerator or coprocessor, enabling enhanced software and hardware performance in the same chip.

Most of the FPGA vendors today have expanded their offerings to include reconfigurable SoC devices in their product catalogs. These devices often represent the flagship models in their respective portfolios, combining the flexibility of FPGA fabrics with the performance and versatility of embedded processors. Notable examples include the SoC families from Xilinx, NanoXplore, and Intel Altera, which integrate embedded cores alongside their programmable logic [6][7][8]. This combination allows for high-performance computing and real-time processing while maintaining adaptability, making these SoCs ideal for applications such as AI, edge computing, and advanced communication systems.

Chapter 3

Radiation-Induced Effects on SRAM-based Devices

3.1 Introduction

Electronic devices operating in radiation-prone environments, such as space or high-altitude regions, are subject to radiation-induced effects that can compromise their functionality and reliability. In this context, FPGAs susceptibility to radiation effects is further exacerbated by the the device extensive silicon requirements to support its high degree of reconfigurability. These effects are broadly classified into hard errors and soft errors. Hard errors are permanent damages to the device's physical structure, often resulting from mechanisms like erroneous high-current states or long-term degradation due to Total Ionizing Dose (TID). Such failures typically render the device unusable or require physical repair or replacement. In contrast, soft errors are transient phenomena that do not cause permanent damage but can temporarily disrupt the operation of the device. Soft errors are particularly critical in SRAM-based FPGAs, where a single upset in the configuration memory can alter the functionality of the programmed circuit.

The focus of this thesis is primarily on soft errors, as they are the most common radiation-induced effects [9] and pose significant challenges for maintaining the reliability of electronic systems, especially in mission-critical applications. The following subsections will explore the different types of Single-Event Effects (SEEs),

their mechanisms, their impact on FPGA-based systems, and a brief overview on the available mitigation approaches.

3.2 Radiation Environments

Radiation environments present a significant challenge for reconfigurable SoC devices, particularly those based on SRAM-based FPGAs due to their vast usage of Silicon. High-energy particles, such as cosmic rays, protons, and heavy ions, can induce soft errors and permanent damage in electronic circuits. Understanding the characteristics of different radiation environments is crucial for designing robust and fault-tolerant systems.

3.2.1 Space Environment

The space environment is one of the most challenging for electronic systems due to the high intensity of ionizing radiation [10, 11]. Key radiation sources include:

- Galactic Cosmic Rays (GCR): High-energy protons and heavy ions originating from outside the solar system. They can penetrate deep into electronic components and induce SEEs.
- Solar Particle Events (SPE): Intense bursts of protons and heavy ions emitted by the Sun, typically during solar flares. These particles can increase the radiation exposure of space systems significantly for hours or days.
- Trapped Radiation Belts: The Van Allen Belts, surrounding Earth, contain high-energy electrons and protons trapped by Earth's magnetic field. Satellites in Low Earth orbit (LEO) or Geostationary Orbit (GEO) must be designed to withstand prolonged exposure to this environment.

3.2.2 High-Altitude and Avionics Radiation Environment

Aircraft and high-altitude applications, such as drones and weather balloons, are exposed to increased radiation levels due to their proximity to the upper atmosphere, where cosmic ray interactions with atmospheric particles generate secondary

neutrons. These neutrons can cause SEUs and Multi-Bit Upsets (MBUs) in reconfigurable SoCs used in flight control systems, radar, and communication equipment [12].

3.2.3 Terrestrial Radiation Environment

Even at ground level, radiation effects are not negligible, especially in High-Performance Computers (HPCs), data centers, and automotive applications. Atmospheric neutrons, generated by cosmic rays interacting with the Earth's atmosphere, can lead to sporadic SEUs in electronic devices. The impact is particularly noticeable in safety-critical systems, such as autonomous vehicles [13], and financial servers [14], where data integrity is paramount.

3.3 Single-Event Effects

SEEs encompass several distinct phenomena such as Single-Event Latchup (SEL), Single-Event Functional Interrupt (SEFI), Single-Event Transient (SET), and the already cited Single-Event Upset and their occurrences depend on several factors such as location, altitude, and solar activity, including events like solar flares and coronal mass ejections.

Single-Event Latchup

A SEL is a severe type of soft error that occurs when a high-energy particle causes a parasitic thyristor-like structure within the semiconductor material to activate [15, 16]. This results in a low-impedance path between the power supply and ground, leading to a substantial increase in current. Unlike other soft errors, SELs may also manifest as not transient; they persist until the device is powered down or damaged due to thermal runaway caused by the excessive current. Modern reconfigurable SoCs and FPGAs employ several design strategies to mitigate SEL risks, such as incorporating silicon-on-insulator (SOI) technology to isolate parasitic elements, adding current-limiting circuits, and monitoring for abnormal current levels with hardware protection mechanisms

Single-Event Functional Interrupt

A Single Event Functional Interrupt (SEFI) is a severe form of soft error that temporarily disrupts the operation of critical components in an FPGA or SoC, such as configuration controllers, clock managers, or power management units [17]. SEFIs often result in system-wide failures, requiring a reset or reconfiguration to restore functionality. These errors are caused by high-energy particles affecting sensitive control logic rather than regular data paths. Mitigation strategies include hardened designs for control blocks, frequent state monitoring, and the use of watchdog timers to detect and recover from SEFI-induced interruptions.

Single-Event Transient

A Single Event Transient (SET) is a momentary glitch in a circuit's electrical signals caused by the interaction of a high-energy particle [18, 19]. Unlike SEUs, which change stored data, SETs manifest as temporary voltage spikes or dips that propagate through combinational logic. If an SET reaches a flip-flop or latch, it can become an SEU. SETs are particularly concerning in high-speed circuits, as their propagation can disrupt normal operations. To mitigate SETs, designers often use filtering techniques, such as low-pass filters or redundant circuitry, to ensure signal integrity. Timing analysis tools also play a role in identifying vulnerable paths during the design phase.

Single-Event Burnout

Single Event Burnout (SEB) is a destructive effect caused by high-energy particles, such as protons or heavy ions, striking power semiconductor devices like MOSFETs or bipolar transistors. Unlike SEU or SET, SEB produces permanent failure, often resulting in short circuits. It occurs when a particle induces a high current that exceeds the device's thermal limits, damaging the semiconductor.

SEB is particularly critical in power electronics for space applications, such as DC-DC converters or motor drivers. Mitigation strategies include using robust devices, protection circuits (fuses, current limiters), radiation shielding, and careful thermal design. Testing and simulations help determine voltage and current thresholds that lead to burnout.

Single-Event Upset

SEUs, among the most common SEEs in SRAM-based FPGAs, result in bit-flips within the Configuration RAM [17, 15]. These events are particularly critical as they can alter the functionality of the programmed circuit. and they represent often the main focus of mitigation approaches due to their prevalence and significant impact on FPGA reliability. The underlying mechanism of SEUs begins when ionizing radiation generates electron-hole pairs within the oxide layer of MOS devices. This process disturbs the threshold voltage and increases leakage currents in the affected transistors. The disturbance produces an SET, which, if it occurs with the appropriate timing and amplitude, can propagate and flip a bit in the CRAM. When this bit-flip occurs in a programmed cell that defines the circuit's netlist, the circuit's structure and behavior are directly altered, potentially leading to functional failures. Understanding and mitigating these effects are essential for deploying FPGAs in radiation-prone environments, especially for mission-critical applications where system reliability is paramount.

3.4 Summary

The Table 3.1 summarizes the effects, the sources and the altitude described in this chapter [20]. Table 3.2, instead, gives an overview on the discussed SEEs and their effects on electronics.

Table 3.1 Space radiation environments and dominant particle populations

Altitude / Orbit	*Radiation Sources*	*Radiation (typical energies)*
LEO (200-800 km)	Trapped particles, GCR, SPE	Protons 10-400 MeV, electrons 100 keV-10 MeV, GCR (protons, HZE ions)
Van Allen belts	Trapped particles	Protons up to hundreds of MeV, electrons hundreds of keV/MeV
GEO (36,000 km)	Relativistic electrons, GCR, SPE	Electrons up to 5-10 MeV, solar protons 10-100 MeV, GCR
Deep Space	GCR, SPE	GCR: protons (85-90%), α (10-12%), ions (1-2%), MeV-TeV; SPE: protons 10-300 MeV

Table 3.2 Single Event Effects (SEE) and their impact on electronics

SEE Type	*Effect on Electronics*
Single Event Latch-up	Activation of parasitic thyristor structures in CMOS devices, causing high current states. Can lead to permanent damage if not cleared by power cycling or protection circuits.
Single Event Functional Interrupt	Temporary interruption of device functionality (e.g., processor or FPGA core). Requires reset or reconfiguration to restore normal operation.
Single Event Transient	Transient voltage/current pulse in combinational logic or analog circuits. Can propagate as glitches or spurious signals, potentially triggering incorrect operations.
Single Event Burnout	Destructive failure in power devices due to high-energy particle strikes, causing permanent short-circuits or catastrophic failure.
Single Event Upset	Bit flip in memory cells, registers or flip-flops. May cause corruption of data or program instructions, usually recoverable.

Chapter 4

Radiation-Hardened Technologies

4.1 Introduction

The necessity of assuring error-proof applications in mission-critical contexts arose the need for new technologies, when common mitigation approaches fail to provide complete protection. Radiation-hardened FPGA chips are proposed as the main solution to this problem, as they exploit the hardening of the architecture that lies behind the implemented design [21, 22]. This includes intrinsic redundancy of resources, which ensures that critical functions can continue operating even if a fault occurs. Additionally, increased robustness of the memory cells minimizes the likelihood of configuration corruption due to radiation-induced bit flips. Another critical enhancement is the increase in the number of transistors in a logic cell, which allows more complex error correction and fault-tolerant computing mechanisms.

To ensure effective mitigation of radiation faults, these enhancements come at the cost of slightly increased power consumption and manufacturing complexity, leading to higher production costs. However, these trade-offs are considered acceptable in mission-critical applications where system failure is not an option. Due to these architectural advancements, the majority of rad-hard FPGA vendors can claim near-complete tolerance of their devices to radiation-induced effects, making them a preferred choice for aerospace, defense, and other radiation-intensive domains.

4.2 Design-Level Hardening

At the design level, several fault-tolerant techniques can be used to ensure that SEEs do not propagate through the system. In COTS devices, research has focused on ways to improve robustness of the design itself such as redundancy of the modules, critical registers, and flip-flops (e.g., Triple Modular Redundancy, Duplication with Comparison) and optimizing logic placement to minimize corruption of cores (e.g., AMD Xilinx Isolation Design Flow). Other approaches include clock and power management, ensuring stability in presence of SEFIs by using fault tolerant clock networks and redundant power regulation.

4.2.1 Triple Modular Redundancy

One of the most widely used techniques for radiation hardening at the design level is Triple Modular Redundancy. TMR works by triplicating the critical modules of an FPGA and using a majority voting system to resolve discrepancies among the outputs. This redundancy ensures that if one of the modules fails due to a SEU, the other two modules can provide the correct output. TMR is particularly effective in mitigating SEUs, as it reduces the likelihood of a single radiation event leading to a system failure. However, TMR increases area overhead and power consumption, which must be considered in mission design [23]. Moreover, the necessity of a voter also introduces a single point of failure, which must be taken care of involving additional effort.

4.2.2 Configuration Scrubbing

Configuration scrubbing is another key technique employed to protect FPGA designs from radiation-induced SEUs in the configuration memory. FPGAs rely on volatile SRAM cells to store their configuration data, making them susceptible to SEUs. Configuration scrubbing periodically rewrites the FPGA's configuration memory with a known good copy to reset any errors caused by radiation. This technique ensures that the FPGA's functionality is restored if an SEU alters the configuration, thus minimizing the chance of a persistent fault. The scrubbing cycle can be optimized to balance performance and error mitigation [24].

4.2.3 Error Detection and Correction

Error detection and correction (EDAC) techniques are essential in ensuring the integrity of data in FPGA designs. Implementing ECC in critical data paths and registers allows for the detection and correction of errors caused by radiation-induced events. The most common approach involves using Hamming codes or Reed-Solomon codes, which can detect and correct single-bit and multi-bit errors. These codes are typically applied in the memory blocks of FPGAs and in the interconnects between logic blocks. EDAC techniques are crucial for maintaining system reliability, particularly in applications that require high fault tolerance, such as satellite communication systems and payload processors [25] .

4.2.4 Low Power and Radiation-Tolerant Clocking

The clocking system in an FPGA can also be a source of vulnerability in the radiation environment. High-frequency clock signals are prone to radiation-induced faults, which can propagate through the entire design. To mitigate this, designers often employ low-power and radiation-tolerant clocking strategies, including using multiple clock domains, redundant clock paths, or clock gating techniques. These strategies reduce the likelihood that a radiation event will cause widespread errors across the FPGA's components [26].

4.3 Intrinsic Hardness: CMOS vs. FinFET

An additional method to ensure better reliability is the sensivity of the technology itself. Considerations about a technology being more or less sensitive than others can save a lot of additional work. The case of CMOS and FinFET sensitivity is presented in the following section as they widely represent the vast majority of the adopted technologies for FPGAs to this day.

The evolution of programmable hardware has closely followed the trajectory predicted by Moore's Law, with successive generations of devices leveraging advances in semiconductor technology to achieve increased integration density and improved performance. In this context, Xilinx SRAM-based FPGAs have transitioned from the 28 nm Complementary Metal-Oxide-Semiconductor (CMOS) technology employed

in the Series 7 family to more advanced FinFET-based multi-gate transistor architectures. This transition is evident in the UltraScale and UltraScale+ device families, which are fabricated using 20 nm, 16 nm, and most recently 7 nm process nodes. The adoption of FinFET technology introduces fundamental changes to the electrical behavior of transistors due to the altered gate geometry and device structure. These changes, in turn, impact the susceptibility of the devices to radiation-induced effects. Factors such as operating voltage, transistor geometry, and material properties all contribute to the overall radiation sensitivity of a given technology node.

While extensive research has been carried out to characterize the radiation response of FinFET and CMOS transistors, particularly in the context of application-specific integrated circuits (ASICs), there remains a relative lack of comprehensive studies focused on the radiation tolerance of reconfigurable FPGA platforms employing these technologies. This gap is especially notable considering the ongoing technological transition in modern FPGA architectures. To address this, a comparative radiation sensitivity evaluation has been carried on of two Xilinx FPGA platforms representative of distinct fabrication technologies: the 28 nm CMOS-based Zynq device and the 16 nm FinFET-based UltraScale+ device [27]. The evaluation was performed through two dynamic proton irradiation campaigns carried out at the Paul Scherrer Institute (PSI). Protons have been chosen to mimic one of the most present radiation source in the space environment, as shown in Chapter 3. The test setup involved a hardware-accelerated multi-core computation engine implemented on each device, enabling dynamic assessment of system reliability under irradiation. In parallel, the static CRAM contents were monitored to evaluate susceptibility to SEUs.

The experimental results reveal a notable improvement in SEU resistance in the 16 nm FinFET device, which exhibited an order-of-magnitude reduction in SEU cross-section compared to its 28 nm CMOS counterpart. However, this improvement in SEU immunity is offset by a higher propensity for critical failure modes, such as SEL, observed in the FinFET-based device. Furthermore, a detailed characterization of Single Event Multiple Upsets (SEMUs) was conducted for both technologies, offering additional insights into the fault mechanisms and spatial sensitivity associated with advanced semiconductor processes.

4.3.1 Experimental Setup

To enable a rigorous and meaningful comparison of radiation sensitivity across fabrication technologies, an identical benchmark was designed and deployed on two distinct Xilinx Zynq SoC platforms, representative of CMOS and FinFET-based configuration memories, respectively.

For the CMOS-based implementation, the PYNQ-Z2 development board was selected. This platform incorporates a Xilinx Zynq-7020 device, fabricated using a 28 nm CMOS process, and integrates a dual-core ARM Cortex-A9 processing system alongside the programmable logic fabric. As a counterpart based on FinFET technology, the Xilinx ZCU104 board was chosen, featuring a Zynq UltraScale+ ZU7EV device. This SoC is manufactured using a 16 nm FinFET process and includes a heterogeneous processing subsystem comprising a quad-core ARM Cortex-A53 application processor, a dual-core ARM Cortex-R5 real-time processor, and advanced programmable logic resources.

A benchmark workload was developed to emulate the computational architecture of a hardware accelerator targeted at AI applications. The design implements a parallel computing engine within the programmable logic, tightly coupled with the embedded processing system present on both SoCs. The computational cores utilized in the benchmark are based on COordinate Rotation DIgital Computer (CORDIC) Intellectual Property (IP) cores, well-suited for evaluating arithmetic-intensive operations in hardware. The benchmark leverages both the programmable logic and the processing system components [28]. The processing system communicates with the programmable logic through the Zynq High-Performance (HP) ports, interfaced via a Master Advanced eXtensible Interface (AXI). Two AXI Direct Memory Access (DMA) IP cores, each supporting multiple channels, are used to facilitate high-throughput data transfer between the processor and the hardware accelerator. The accelerator itself comprises 16 CORDIC cores per DMA unit, yielding a total of 32 CORDIC cores distributed across the programmable logic fabric. Figure 4.1 shows a block scheme of the relevant benchmark.

This architecture enables a balanced and scalable test environment to evaluate the behavior of the system under radiation, allowing for the dynamic observation of functional disruptions across both logic and processing subsystems in devices built

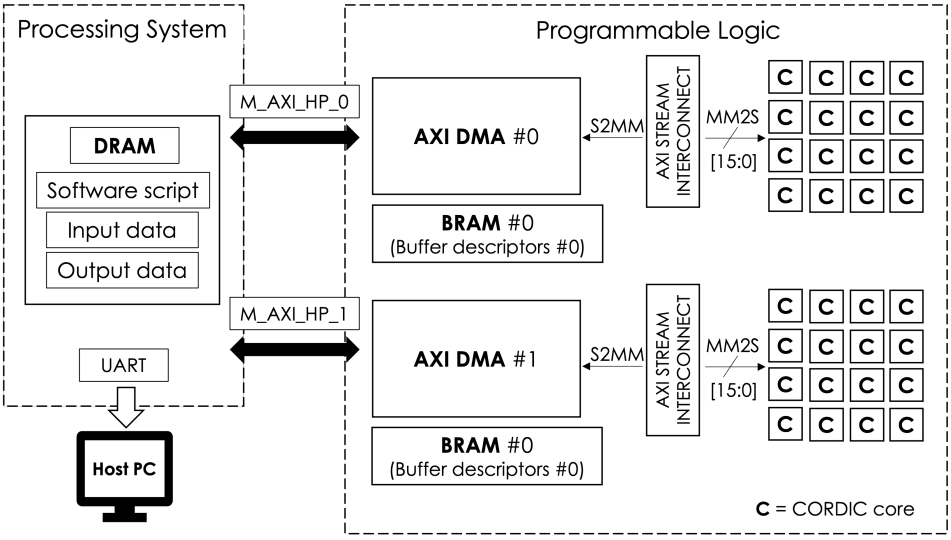

Fig. 4.1 Block scheme of the benchmark.

on distinct technology nodes. The resources utilization for both devices is shown in Table 4.1

Table 4.1 FPGA utilization for the Benchmark Circuits

Resources	**28nm CMOS Zynq (7 Series)**		**16nm FinFet Zynq (US+)**	
	Available	Used (%)	Available	Used (%)
LUTs	53,200	28,413 (53.41%)	230,400	30,633 (13.30%)
Flip-Flops	106,400	33,317 (31.31%)	460,800	34,031 (7.39%)
BRAM	140	14 (4.39%)	312	10 (3.21%)

Two proton irradiation campaigns were carried out at the Proton Irradiation Facility of the Paul Scherrer Institute (PSI) in Switzerland, aimed at assessing the radiation response of the selected programmable platforms.

Throughout the experiments, system monitoring and configuration were managed using the PyXEL experiment manager [29], operating on a dedicated host computer located in the control room. Communication with the DUTs was established via a serial interface, enabling remote control and supervision during irradiation. The PyXEL framework facilitated both real-time acquisition of output data from the benchmark application running on the DUTs and periodic readback of the configuration memory contents. By comparing the original (golden) configuration bitstream with the readback data collected during irradiation, the framework enabled the

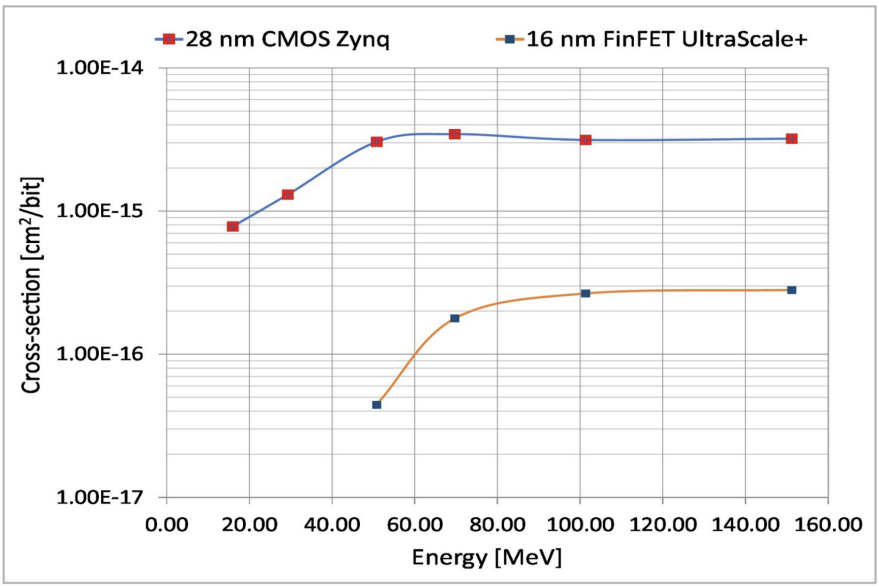

Fig. 4.2 Cross section comparison between CMOS and FinFET.

real-time detection and quantification of SEUs in the configuration memory. This capability was instrumental in assessing the dynamic fault behavior of each device and supported precise evaluation of radiation-induced errors in the programmable logic fabric.

4.3.2 Test Results and Analysis

The measured SEU cross-sections for both the 16 nm FinFET and the 28 nm CMOS technologies are presented in Figure 4.2. A notable observation is that, for the 16 nm FinFET-based device, no Single Event Upsets were recorded at proton energies below 50 MeV. Consequently, the reported cross-section values for this technology are valid only for incident energies equal to or exceeding 50 MeV.

The 16 nm FinFET technology exhibits a significantly lower SEU cross-section compared to the 28 nm CMOS device. This indicates a reduced sensitivity to radiation-induced bit-flips in the configuration memory. The observed difference in SEU cross-section (expressed in terms of events per bit) highlights the enhanced robustness of FinFET-based programmable logic against ionizing radiation, particularly at higher energy levels.

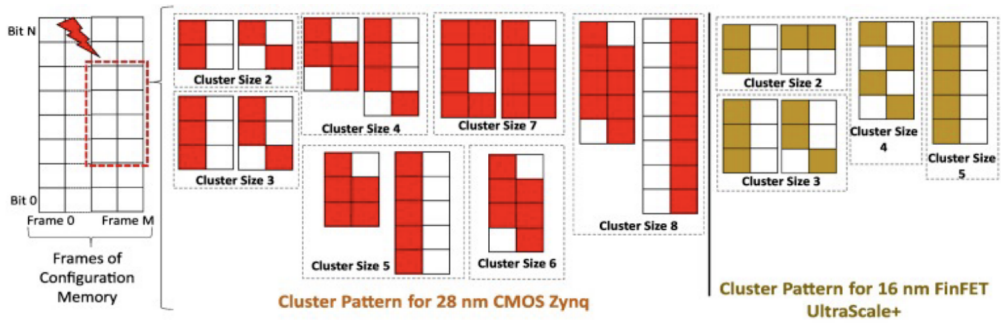

Fig. 4.3 SEMU patterns of CMOS and FinFET technologies.

The configuration memory readback process was performed continuously throughout the irradiation campaigns, with a readback duration of approximately 5 seconds for the Zynq-7020 (28 nm CMOS) and 12 seconds for the ZU7EV (16 nm FinFET) devices. Initial tuning runs have been performed in order to fine-tune the proton flux; doing so, the experimental conditions were optimized such that only a limited number of bitflips occurred in the configuration memory between consecutive readback intervals (in the range of 20-30 bitflips per readback acquisition). This approach enabled the identification of spatially and temporally correlated SEU events, thereby facilitating the detection of SEMUs. The patterns identified are reported in Figure 4.3.

Given the substantial size of the configuration memory, exceeding 32,335,848 bits for the Zynq-7020 and more than 154,472,256 bits for the ZU7EV, coupled with the ability to perform continuous, high-resolution readback, it was possible to observe bit-flip clusters with a high probability of being induced by SEMUs.

A detailed analysis of the readback data revealed distinct SEMU patterns within the configuration memory. The results indicate a clear dependency of SEMU morphology on the underlying fabrication technology, with notable differences in both the extent and geometric distribution of the clusters between the 28 nm CMOS and 16 nm FinFET devices.

During the proton irradiation campaigns, both SELs and SEFIs were observed, affecting the ZU7EV and Zynq-7020 devices, respectively. SELs, which were recorded exclusively on the ZU7EV (16 nm FinFET), are critical events characterized by the onset of high current paths within the silicon, potentially leading to a complete loss of functionality or permanent damage if the current exceeds safe thresholds.

In non-destructive cases, a full power cycle is required to return the device to its nominal operational state.

Conversely, SEFIs, detected only in the Zynq-7020 (28 nm CMOS), typically result in the disruption of system operation without causing permanent damage. These events often necessitate a soft reset or, in more severe cases, a power cycle to restore normal functionality. However, should a SEFI corrupt or reset the configuration memory, the device will enter a non-functional state, requiring a full reconfiguration to resume normal operation. It is thus important to recognize that, despite the improved resistance of the FinFET-based device to SEUs, its susceptibility to SELs introduces a critical reliability concern, particularly in radiation-rich environments. In contrast, the SEFIs observed in the 28 nm CMOS device pose comparatively less severe consequences, albeit still requiring mitigation strategies for mission-critical applications. Specifically, during the irradiation tests, the ZU7EV experienced up to 18 SEL events, whereas the Zynq-7020 recorded 7 SEFIs. Notably, no SELs were detected on the Zynq-7020, and no SEFIs were observed on the ZU7EV, underscoring a distinct technology-dependent failure mode.

The comparative analysis of FPGAs fabricated with 16 nm FinFET and 28 nm CMOS technologies, through targeted proton radiation testing, has led to several important insights. While the FinFET-based configuration memory exhibited a significantly lower cross-section for SEUs and SEMUs, indicating enhanced resilience to soft errors, the higher incidence of SELs in this technology poses a substantial challenge for its deployment in radiation-prone environments. This trade-off must be carefully considered when selecting programmable logic technologies for space and other mission-critical applications.

4.4 Process and Material Hardening

Process hardening involves altering the manufacturing process of FPGAs to increase their tolerance to radiation effects. One of the most common approaches is the use of Silicon-on-Insulator (SOI) technology, which isolates the active transistor regions from the substrate using an insulating layer. This design reduces the likelihood of radiation-induced current leakage, thus minimizing the risk of SELs, which can cause catastrophic failures. SOI-based FPGAs are also known to exhibit enhanced

tolerance to TID due to the reduction in the parasitic capacitance between the active and substrate layers.

Another significant process hardening technique is the use of Rad-Hard-by-Design (RHBD) principles, where the design of the FPGA is optimized to mitigate radiation effects without relying on the underlying process technology. This includes the use of radiation-hardened libraries for critical components such as flip-flops, SRAM cells, and logic gates, which are specifically designed to reduce susceptibility to SEUs [30]. In some applications, the use of radiation-hardened FPGAs, which are pre-engineered to withstand radiation effects, is the most reliable option. Manufacturers such as Xilinx and Microsemi provide FPGAs that are specifically designed and tested for space applications. These devices are constructed using radiation-hardened processes and often come with built-in radiation mitigation features, such as increased TID tolerance and hardened SRAM cells. Such devices are ideal for space missions requiring the highest level of reliability and performance in radiation-prone environments

Part II

Radiation-Induced Effects: Analysis and Observations

Chapter 5

SEUs Analysis Methodologies

5.1 Introduction

The vulnerability of SRAM-based programmable hardware to SEUs is especially pronounced due to its configurability. In these devices, the functionality of the hardware is dictated by the contents of the configuration memory, which consists of millions, or even billions, of bits that define the behavior of logic and sequential elements. The high density of these memory cells significantly increases their susceptibility to radiation-induced faults. Moreover, since configuration memory is typically written only during initialization or reconfiguration, faults may accumulate over time, progressively corrupting the implemented architecture and ultimately leading to functional failures. While mitigation strategies such as TMR and configuration scrubbing are commonly adopted, they impose non-negligible trade-offs in terms of area overhead, power consumption, system availability, and design complexity. Compounding the challenge, FPGA vendors provide little to no documentation regarding the internal structure and encoding of configuration memory, thereby limiting the development of efficient fault-tolerant techniques and detailed reliability models.

To evaluate the resilience of FPGA-based systems against SEUs, fault injection has become an essential methodology. By artificially reproducing the effects of SEUs, fault injection enables the systematic assessment of system robustness and the validation of mitigation strategies. Fault injection techniques can be broadly categorized into fault simulation and fault emulation. Fault simulation is performed in a software environment, often at register-transfer level (RTL) or gate level, where

faults are modeled and injected in a controlled fashion. This approach provides high flexibility and full observability of the system state but suffers from poor scalability due to the high computational cost of simulating large and complex designs. On the other hand, fault emulation leverages the reconfigurability of hardware platforms to physically reproduce the fault effects on the target FPGA, either by perturbing the configuration memory or forcing signals during execution. Emulation offers higher fidelity and dramatically accelerates fault campaigns, although it comes with more limited observability and additional setup complexity.

The coexistence of these two approaches reflects the inherent trade-off between accuracy, speed, and observability in dependability evaluation. While simulation is particularly effective for early-stage design exploration and exhaustive small-scale fault campaigns, emulation is better suited for large-scale experiments and validation under realistic operating conditions. In this chapter, the principles, methodologies, and challenges of SEU fault injection on FPGAs are introduced, emphasizing emulation, forming a comprehensive framework for reliability assessment in safety-critical applications.

5.2 Fault Injection Methods

Fault injection techniques have emerged as a fundamental approach for analyzing the susceptibility of FPGAs to SEUs and evaluating the effectiveness of mitigation strategies. Given the increasing complexity of modern SRAM-based FPGAs, fault injection enables controlled experimentation and systematic assessment of soft error resilience without requiring exposure to actual radiation environments. Several fault injection techniques have been developed to model the effects of SEUs in FPGA-based systems such as software-based, hardware-based and laser-based approaches. The software-based method manipulates the FPGA bitstream or configuration memory to simulate SEUs [31][32]. It is widely used due to its accessibility and non-intrusive nature, allowing researchers to evaluate fault tolerance mechanisms without additional hardware. However, it often lacks real-time execution fidelity. This flaw is partially overcome by the hardware-based injection, where single or multiple bitflips can be emulated physically on the bitstream which will be downloaded into the chip [29]. This approach provides high-precision SEU emulation but requires deep knowledge of the bitstream architecture, often hidden or

vendor-proprietary. Last, laser-based fault injection uses high-energy laser pulses for inducing localized charge collection, simulating SEUs in FPGA transistors. This approach is particularly effective for studying fine-grained fault effects but requires specialized equipment [33].
Accelerated radiation testing constitutes a fundamental methodology for assessing the reliability of electronic components and systems intended for operation in radiation-prone environments, such as outer space. By emulating, within a compressed timeframe, the cumulative effects of radiation exposure, this approach enables the early identification, evaluation, and mitigation of potential system vulnerabilities prior to field deployment. This is particularly critical in the context of safety-critical applications, where system failures could compromise expensive infrastructure, such as satellite platforms, or jeopardize human safety, for instance in disaster-prevention systems. Compared to alternative approaches such as fault injection or software-based simulations, radiation testing offers unique advantages. Despite the necessity of employing elevated particle fluxes to accelerate the testing process, it exposes the actual Device Under Test (DUT) to real physical phenomena, thereby yielding results with higher fidelity and representativeness of the operational environment. Radiation testing can be tailored to evaluate system sensitivity to two primary classes of radiation effects: TID and SEEs. The focus of the experimental work presented in the following chapter is on SEU testing, which investigates the behavior of electronic systems under exposure to high-energy particles, typically protons or heavy ions, accelerated to simulate the harsh conditions encountered in space. SEE evaluation can be conducted through static characterization, such as quantifying the SEU rate in memory elements, or through dynamic characterization, wherein the functional integrity of the entire system is assessed during irradiation.

Nonetheless, radiation testing is not without its challenges. The increasing architectural complexity of modern integrated systems complicates the isolation of faults to specific components, particularly in system-level experiments. Furthermore, such testing requires access to specialized facilities equipped with particle accelerators, which introduces significant logistical constraints, elevated costs, and potential delays due to limited facility availability. Additionally, the exposure of devices to high-energy particles may result in permanent damage or functional degradation, thereby limiting re-usability. Successful execution of radiation experiments demands substantial effort in the design and validation of both the experimental setup and

the data acquisition infrastructure, to ensure the reliability of collected data and minimize the risk of destructive events.

In recent years, commercial reconfigurable SoC platforms have gained increasing traction for advanced applications in aerospace and high-energy physics, because of their high performance and versatility. Consequently, the assessment of their resilience to ionizing radiation has become an area of significant research interest. However, system-level evaluations are increasingly complicated by the integration of diverse subsystems, such as programmable logic, embedded processors, and volatile memories, on a single silicon die.

Although extensive studies have been carried out to investigate the radiation tolerance of configuration memory, widely acknowledged as a critical vulnerability in SRAM-based FPGAs, there remains a relative paucity of research focusing on the reliability of processor subsystems, shared memory architectures, and full-system functionality under dynamic radiation exposure. Bridging this gap is essential to develop a comprehensive understanding of the behavior of modern SoC platforms in radiation environments.

5.3 Evaluation of Soft-Processors Reliability

5.3.1 State of the Art of Soft-processors Reliability

In recent years, programmable hardware devices have been increasingly adopted in mission-critical applications. Their high performance, combined with the flexibility and cost advantages over ASICs, has made them a compelling choice for domains such as automotive and space applications. One of the most common implementations within programmable hardware is the use of soft-core microprocessors, which provide an efficient means of integrating computational capabilities with hardware acceleration. By leveraging an IP-based microprocessor core, designers can achieve a balance between hardware-level performance and the adaptability of software-based processing.

Soft microprocessors are frequently implemented in programmable hardware to enable a balance between hardware acceleration and software flexibility. Even when an SoC is already equipped with a hardwired microprocessor, the inclusion of

a soft-core processor can be advantageous for offloading computational tasks from the primary processor or providing redundancy. Among the available architectures, RISC-V soft processors have gained significant traction due to their open-source licensing and extensive community support. The Riduced Instruction Set (RISC) offers a small and highly efficient base instruction set that can be extended with standardized or custom modules for floating-point computation, vector processing, security, and artificial intelligence. Its fast spreading is mainly due to RISV-V being royalty-free, allowing users to build, design and commercialize their own customized processor and enabling fosters innovation in industry and academia [34] [35] [36].

In mission-critical applications, the deployment of soft processors necessitates careful consideration of their susceptibility to SEUs, which can lead to system malfunctions. Various techniques have been developed to enhance reliability against SEUs, with hardware redundancy being the most effective. However, redundancy-based approaches impose significant overhead in terms of design complexity, area utilization, and power consumption. Alternatively, software-based fault tolerance strategies offer a more accessible and less resource-intensive solution. These methods rely on data replication in memory, redundant execution of operations, and consistency verification during runtime. While software-based approaches do not achieve the same level of robustness as hardware redundancy, they provide a cost-effective means of mitigating SEU-induced failures.

Soft-core processors implemented in FPGAs present an additional layer of vulnerability compared to hardwired processors due to their reliance on CRAM. This memory defines the netlist of the implemented circuit, and corruption due to SEUs can result in structural faults at the hardware level. Consequently, software replication techniques, which are typically effective in protecting against memory errors, may be less efficient in mitigating SEUs in soft processors due to their susceptibility to architectural faults, an issue that is significantly less pronounced in fixed-hardware microprocessors.

In recent years, both hardware- and software-based methodologies have been extensively investigated to meet the stringent reliability requirements of mission-critical systems. One of the earliest proposed approaches for fault tolerance enhancement was software hardening-by-replication [37]. Although this method is less effective than complex hardware-based strategies such as Triple Modular Redundancy (TMR), its ease of implementation has contributed to its widespread adoption. Notably, as

early as the 2000s, NASA recommended software-based techniques for enhancing the fault tolerance of spaceborne applications [38].

Prior research has examined the resilience of both software and hardware platforms in the presence of SEUs. For instance, in [39], a methodology for detecting soft errors in code and data through software replication was proposed. Similarly, [40] introduced a software-based approach capable of both error detection and correction via data and code replication. Furthermore, [41] conducted a comparative analysis of hardware and software mitigation techniques using fault injection campaigns targeting microprocessors, including FPGAs. However, these studies primarily focused on faults affecting storage elements, neglecting the potential impact of SEUs on configuration memory and other FPGA-specific components.

Among the notable software-based mitigation strategies, [42] introduced SWIFT, a fault-detection technique that leverages instruction-level parallelism. Additionally, [43] expanded on software-level detection and correction mechanisms through data and code replication. The authors of [44] further evaluated both hardware- and software-based strategies against SEUs through fault injection experiments on microprocessors. Despite including FPGA-based designs, their analysis was limited to memory corruption without considering faults affecting configuration memory or other critical programmable resources.

Software hardening-by-replication is fundamentally based on the redundancy of data and computational processes. To enhance resilience against SEUs, input data can be stored in multiple copies within memory, and the execution of computations can be repeated using redundant data to verify correctness. Depending on the required level of reliability and the acceptable performance overhead, detection and correction checkpoints can be strategically inserted into the software. At these checkpoints, intermediate computational results are cross-checked, enabling the identification and correction of discrepancies through majority voting, either during execution or upon final output computation.

5.3.2 Experimental Flow for Soft-Processors Reliability Analysis

A RISC-V soft processor serves as the foundation for reliability evaluations in our experiment. To facilitate this analysis, a benchmark suite comprising a set of software applications has been selected. Each application is available in both an unhardened

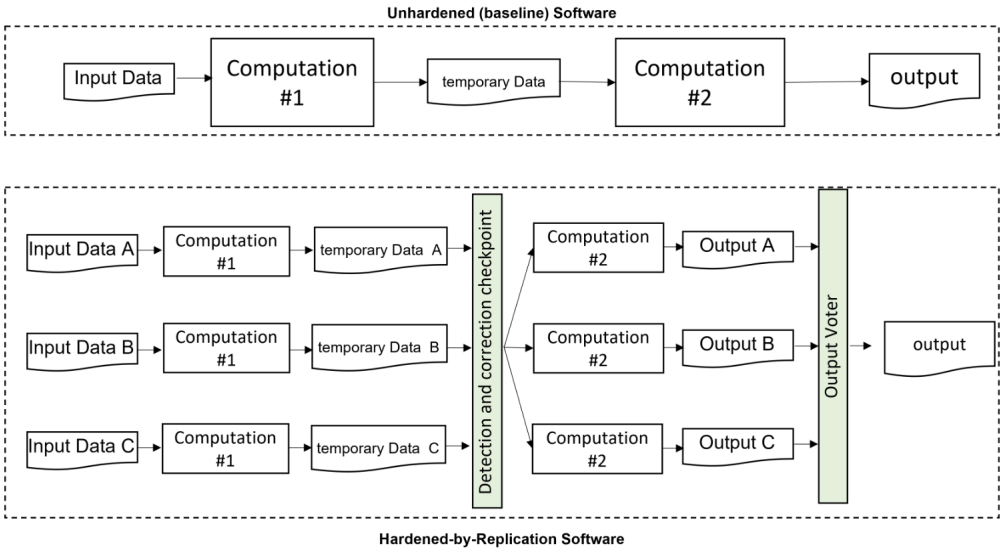

Fig. 5.1 View of the baseline and hardened version of the software.

(baseline) version and a hardened-by-replication variant. The reliability assessment is conducted using a fault injection platform, where fault models are simulated in both the processor's memory and the CRAM while the software executes on the system. Figure 5.1 shows the two designs.

Hardware Platform

To evaluate reliability, we selected PULPissimo as the microcontroller architecture for our analysis. PULPissimo is a single-core platform developed as part of the PULP project, a collaboration between ETH Zurich and the University of Bologna [45]. As the target programmable hardware device, a Nexys Video Artix-7 FPGA has been chosen. The resource utilization for implementing PULPissimo on this FPGA is detailed in Table 5.1

Table 5.1 FPGA utilization for the PULPissimo Soft Processor implemented on Nexys Video (Artix-7)

Resources	**Available [#]**	**Used [#] (%)**
Logic Slices	33,650	14,150 (42.05%)
Flip-Flops	106,400	21,531 (8.00%)
BRAMs	140	128 (35.07%)
DSPs	740	12 (1.62%)

Software Benchmarks

Four benchmark software applications were selected to evaluate the system's behavior under radiation-induced fault conditions. These applications span multiple computational domains, including signal and image processing, and are widely recognized in performance and reliability evaluation studies. The selected benchmarks are as follows:

- *CoreMark*: A comprehensive workload that includes list processing, matrix manipulation, finite state machine execution, and cyclic redundancy check (CRC) operations. It is designed to assess the performance of central processing units in embedded systems.
- *Dhrystone*: A synthetic benchmark primarily targeting integer and string processing tasks. It excludes floating-point operations and focuses on evaluating general-purpose computing performance.
- *FFT*: An implementation of the Fast Fourier Transform, which is a fundamental algorithm in digital signal processing for frequency-domain analysis.
- *Sobel*: Implements the Sobel operator, a widely used edge detection algorithm in image processing applications.

Both the FFT and Sobel benchmarks are drawn from the MiBench Benchmark Suite, a collection of embedded applications tailored for evaluating real-world workloads.

To assess the effectiveness of software-level fault tolerance mechanisms under radiation exposure, a hardened version of each application has been developed. The hardening methodology followed the one proposed in [38], where single-version software fault tolerance techniques were applied. These techniques involve replicating input data, program variables, and critical functions in memory. The modified applications execute redundant operations across these replicated elements and incorporate periodic checkpoint routines for error detection and correction. This approach enables the mitigation of transient faults through majority voting or consistency verification, enhancing the overall system reliability without requiring hardware modifications.

Methodology of the Experiment

To evaluate the impact of radiation-induced faults on system behavior, two distinct fault injection platforms were employed to emulate Single Event Upsets (SEUs) in both the main memory and the configuration memory of microprocessor-based systems.

For emulating SEUs in main memory, a Python-based fault injection framework was developed. This platform operates directly on the Executable and Linking Format (ELF) file, injecting faults into the memory contents to simulate the effects of SEUs. The modified ELF binaries are subsequently loaded into the memory of the PULPissimo microcontroller platform. The system integrates with Open On-Chip Debugger (OpenOCD) and GDB, which interface with the RISC-V debug module via the JTAG interface to facilitate memory manipulation and program execution. The framework automates the entire injection and observation process, including application loading, fault injection, result acquisition via the serial port, and timeout handling to detect processor hangs caused by injected faults.

To emulate SEUs in the configuration memory of FPGAs, the PyXEL platform was adopted, described in Subsection 4.3.1 [29]. In this study, PyXEL was applied to the Artix-7 XC7A200T FPGA, enabling targeted bit-level corruption in the configuration memory, effectively simulating SEU scenarios. The platform automates key stages of the injection campaign, including configuring the FPGA with a bitstream implementing a RISC-V soft-core processor, resetting the system in response to fault-induced halts, loading and executing software benchmarks on the soft processor, and collecting computation results.

Together, these platforms provided comprehensive coverage of fault injection scenarios, allowing for the detailed assessment of system behavior under SEU conditions in both volatile and non-volatile memory elements.

5.3.3 Results of Fault Injections on Hardened Soft-Processor

To assess the benefits and limitations of hardening-by-replication techniques for software running on soft-core microprocessors, a series of reliability analyses were performed. Specifically, the study focused on evaluating both baseline and hardened versions of several benchmark applications under the effects of SEUs, injected in

both the main memory and configuration memory of a reconfigurable hardware platform. All software benchmarks were executed in a bare-metal environment, i.e., without the presence of an operating system, to isolate the effects of injected faults.

The outcomes of the fault injection experiments were categorized into the following distinct classes:

- *Correct*: The benchmark completed successfully, and the output was consistent with the reference (golden) output.
- *Silent Data Corruption (SDC)*: The benchmark terminated, but the produced output differed from the expected result.
- *Halt*: The benchmark failed to complete execution, typically due to system-level faults such as infinite loops, illegal instruction execution, or processor crashes. These failures may result from corruption at either the software (binary code) or hardware (architecture-level) layer.

To quantitatively assess system reliability, an error rate was defined as the proportion of fault injection experiments resulting in either SDC or Halting behavior. This metric provides an indication of the likelihood that a system subjected to radiation-induced faults will produce incorrect or incomplete results.

A first fault injection campaign targeted the main memory of the soft-core microprocessor to assess the reliability of unprotected (baseline) software. For each of the four selected benchmark applications, 10,000 independent fault injection experiments were performed. In each experiment, a single bit within the memory image of the application was randomly flipped, simulating an SEU. This approach enabled a statistical evaluation of the application's robustness against transient memory faults. A second fault injection campaign focused on SEUs affecting the configuration memory of the FPGA hosting the soft-core processor. Corruption in configuration memory can result in permanent alterations to the hardware architecture of the system until a reconfiguration or power cycle is performed. Given that only a fraction of the FPGA's configuration bits are utilized by a specific design, the observed error rate for configuration memory faults tends to be lower than that for main memory faults. Nevertheless, such faults are critical in the context of reliability, as they can lead to persistent malfunctions. In this campaign, 10,000 single-bit faults were randomly injected into the configuration memory bitstream. Each injected

bitstream was then used to configure the FPGA before executing one of the software benchmarks. The variability in the logic resources utilized by each application results in differing sensitivity to the same configuration fault across benchmarks. This underscores the importance of application-specific analysis in evaluating system reliability. The obtained results are shown in Table 5.2 and Table 5.3, reporting the absolute frequencies of the obtained errors over 10,000 faulty bitstreams.

Table 5.2 Error categorization resulting from fault injection campaigns on baseline software

Fault	*SEU in Processor Memory*			*SEU in Configuration Memory*		
	Mask [#]	*SDC [#]*	*Halt [#]*	*Mask [#]*	*SDC [#]*	Halt *[#]*
Coremark	9,673	172	155	9,933	20	47
Dhrystone	9,861	69	70	9,946	12	42
FFT	9,813	80	107	9,926	28	46
Sobel	9,845	81	84	9,926	10	64

Table 5.3 Error categorization resulting from fault injection campaigns on hardened software

Fault	*SEU in Processor Memory*			*SEU in Configuration Memory*		
	Mask [#]	*SDC [#]*	*Halt [#]*	*Mask [#]*	*SDC [#]*	Halt *[#]*
Coremark	9,711	122	167	9,932	19	49
Dhrystone	9,878	41	81	9,942	11	47
FFT	9,836	56	108	9,909	30	61
Sobel	9,850	80	70	9,921	9	70

The subsequent two fault injection campaigns are aimed at assessing the effectiveness of the hardening technique applied to the software benchmarks. These campaigns replicate the methodology employed in the previous analyses; however, they are conducted on the hardened versions of the software applications. The objective is to quantify the improvement in resilience to SEUs introduced by the application of software-level fault tolerance mechanisms. The comparison between the reliability of the baseline and hardened software yielded notable insights, as illustrated in Figure 5.2. In particular, the application of software hardening techniques led to a modest improvement in reliability across all benchmarks when subjected to SEUs in the processor memory. Conversely, this positive effect was not mirrored in the case of SEUs in the configuration memory. In fact, the hardened versions exhibited a slight decrease in reliability, suggesting that the added complexity and

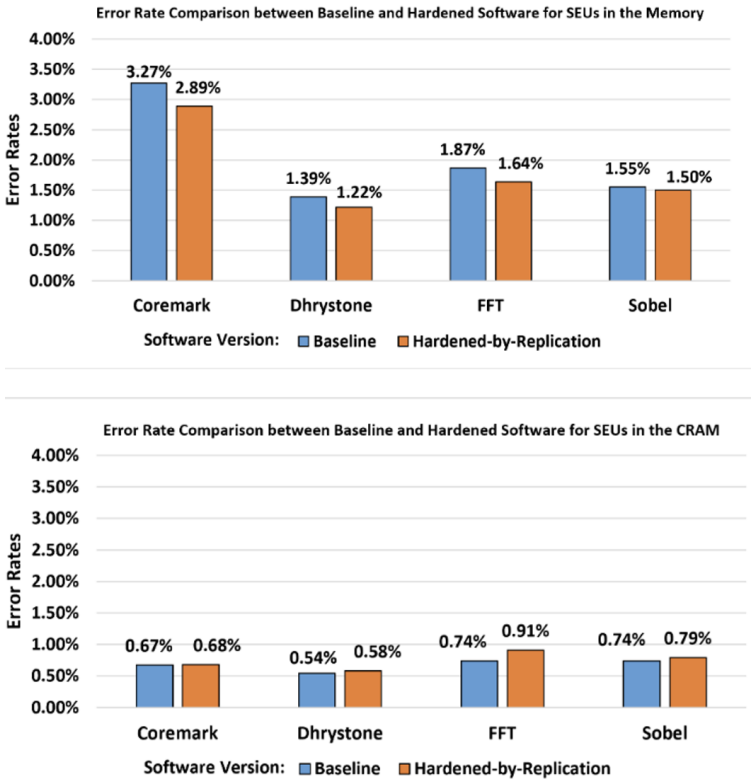

Fig. 5.2 Comparison between baseline and hardened software reliability against injected SEUs.

hardware usage introduced by the hardening mechanisms may have inadvertently increased the system's vulnerability to faults in the underlying hardware architecture.

Two main hypotheses have been formulated to explain these results. First, software-based replication techniques are inherently limited in mitigating faults that affect the hardware elements of the microprocessor. When the same computation is redundantly executed on faulty hardware resources, the outcome is likely to be consistently erroneous across replicas. While exceptions exist, like when a fault affects the readout of a specific memory cell and the replicated data is stored in distinct memory locations, such cases are relatively rare, and the overall mitigation effectiveness remains limited compared to faults confined to memory. Second, the additional logic introduced to implement detection and correction checkpoints may activate portions of the hardware that remain unused in the baseline software execution. This increased hardware utilization could expose new failure modes,

potentially causing faults that were previously dormant or non-propagating to now manifest as observable deviations in the application's output.

5.4 Evaluation of Data Transfer in Programmable Logic

5.4.1 State of the Art of AXI Reliability

Different studies have investigated and analyzed data from radiation tests on reconfigurable devices, although most of them have primarily focused on the interaction with heavy ions. For example, [46] characterized a Xilinx Kintex-7 device, based on 28 nm CMOS technology, under ultra-high energy heavy ion irradiation, providing insights into the vulnerability of this technology node in extreme conditions. The work in [47] explored the behavior of BRAMs in a Zynq-7000 device when exposed to heavy ions, an important and complementary contribution to this work, as BRAMs will store the Buffer Descriptors (BDs), as discussed later. Additional investigations in [48] have extended this analysis to both embedded memories and processors within the Zynq-7000 platform, under both heavy ion and proton exposure, offering a broader understanding of component-level sensitivity in SoCs. The authors in [49] have also evaluated the reliability of the Advanced eXtensible Interface (AXI) in Xilinx Reconfigurable SoCs. This interface is the standard adopted by ARM for the communication within SoCs, thus the one available for Xilinx devices that embeds ARM cores. One such study quantified the error rates and analyzed the behavior of AXI Interconnect IP blocks when affected by SEUs, identifying potential failure modes and reliability concerns. The study in [50] focused on the interaction between AXI Stream and Memory-Mapped flows, including an evaluation of hardened circuit versions to assess mitigation effectiveness. However, despite the technical rigor of these contributions, they typically simulate SEUs by randomly flipping bits in the CRAM to emulate soft errors, which may not fully capture the spatial and temporal correlation of real radiation-induced upsets. As highlighted in these references, heterogeneous architectures that leverage DMA are particularly susceptible to soft errors. Nonetheless, while memory blocks and processor cores have been extensively studied in previous research, the DMA subsystem remains a critical component that has not yet been deeply analyzed, warranting further investigation.

A DMA architecture enables hardware subsystems to directly access memory without CPU intervention, thereby allowing the processor to continue working on other tasks concurrently while memory operations are executed in parallel. In this context, the term DMA refers specifically to the controller, implemented in the programmable hardware, that orchestrates data transfers, rather than the physical data channels themselves. DMA has become a critical component in reconfigurable devices due to its capability to handle data transfers between different types of interfaces, such as AXI Memory-Mapped and AXI Stream interfaces. These interfaces follow the AXI specification, a standard widely supported by both Xilinx and ARM that facilitates high-bandwidth communication and seamless integration of hardware and software components in complex SoC architectures. In FPGAs, DMA controllers are typically instantiated in the programmable logic and interact with fixed, hard-wired memory and interface resources on the board. As such, the implementation of DMA in reconfigurable SoCs consumes a significant amount of CRAM, making it especially susceptible to corruption from SEUs. This vulnerability becomes even more critical in architectures using the Scatter-Gather (SG) engine, a DMA operation mode in which memory transfers are defined using BDs. Each BD describes critical parameters such as the source and destination addresses, transfer direction, data length, and the number of 32- or 64-bit words to be moved. Unlike the Direct Register Mode (also known as Simple DMA), where the CPU must initiate each transfer, the SG engine allows BDs to be configured once and then automatically accessed by the controller, enabling autonomous and uninterrupted transfer sequences. Radiation-induced effects such as SEUs are especially problematic in modules such as DMA controllers, where logic complexity and CRAM footprint are substantial, increasing the probability of a critical bit flip. In Figure 5.3, a typical failure scenario in reconfigurable DMA functioning due to bitflip is shown.

5.4.2 Experimental Flow for AXI DMA Reliability Analysis

In order to evaluate the radiation-induced effects on reconfigurable DMA architectures, a dedicated analysis workflow was adopted. Initially, a proton irradiation campaign was conducted on an FPGA implementing a DMA-based data transfer system interfacing with hardware accelerators. The test outputs were collected and analyzed to extract key parameters such as cross-section values, failure rates, and observed patterns of MBUs. These empirical data were subsequently used to guide

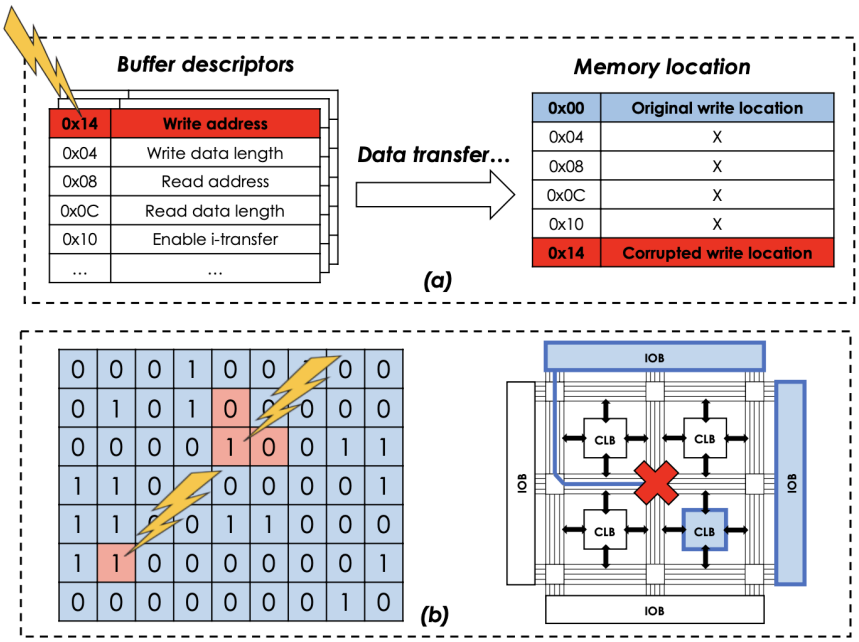

Fig. 5.3 Examples of errors in DMA due to radiation-induced bitflips. (a) Corruption of the BD causing wrong output products. (b) Corruption of routing bits causing output unavailability.

targeted fault injection campaigns aimed at reproducing and understanding the impact of radiation effects in a controlled environment. The detailed methodology and steps followed throughout this analysis are described in the following subsections.

Application Benchmark

Considering the typical use of reconfigurable devices in hardware acceleration scenarios, a benchmark was developed to stress the data transfer and computation capabilities, with a particular focus on maximizing the usage of the DMA infrastructure rather than evaluating the accuracy of the computational output. Each DMA module operates at full capacity, utilizing all 16 available channels and the maximum burst length for data transfers. The benchmark orchestrates the transfer of data to and from a matrix of hardware accelerators implemented in the programmable logic. The system architecture includes a microprocessor connected via Master AXI High-Performance ports to two AXI DMA IP blocks configured in multichannel mode. Each DMA controls a 16-core hardware accelerator composed of multiple CORDIC IP cores. These IP cores implement the CORDIC algorithm, a resource-efficient

method for computing trigonometric functions, vector rotations, and square roots without requiring multiplication operations. In this setup, the cores specifically compute the sine and cosine of incoming data streams. For each DMA, separate BDs are allocated in dedicated BRAMs to handle Memory Mapped-to-Stream (MM2S) and Stream-to-Memory Mapped (S2MM) transfers. Interrupt signals from the DMAs are aggregated and managed appropriately. A software routine running on the processor is responsible for printing the output buffer contents, configuring the BDs, and handling the interrupts.

Radiation Experiment

The proton irradiation campaign was conducted at the Proton Irradiation Facility of the Paul Scherrer Institute (PSI), an irradiation facility for medical purposes that allows industries and academia to exploit Throughout the following sections, the terminology adopted is as follows:

- *Energy*: refers to the kinetic energy carried by a single proton. Measured in eV
- *Flux*: indicates the rate at which particles cross a one-square-centimeter surface per unit of time. Measured in $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$
- *Fluence*: denotes the total number of particles that crossed such a surface during the exposure. Measured in cm^{-2}

The term cross-section represents the likelihood that a specific component (e.g., a DMA module, a CORDIC core, a bit in CRAM) will experience a functional failure as a consequence of radiation. For a given energy level E, let us assume N irradiation runs were performed. The cross-section per computational core (σ) is then calculated by dividing the total number of observed events across all runs by the product of the cumulative fluence and a normalization constant k (e.g., the number of cores), as expressed in Equation 6.1:

$$\sigma = \frac{\sum_{i=1}^{N} \mathrm{events}_i(E)}{\sum_{i=1}^{N} F_i(E) \cdot k} \tag{5.1}$$

Several irradiation energies and flux levels were employed, enabling the observation of diverse device behaviors and ensuring coverage of a wide range of LEO

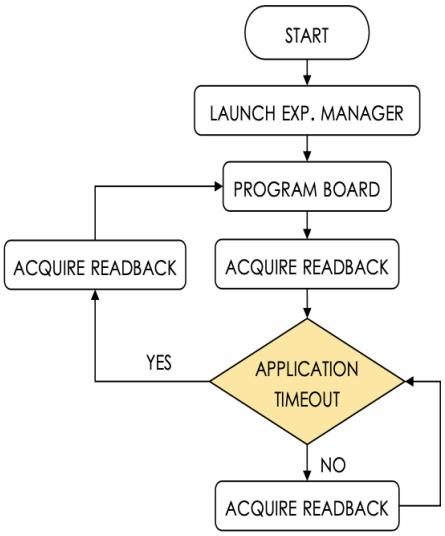

Fig. 5.4 Experiment flow for radiation experiment.

radiation scenarios. The device under test was mounted and precisely aligned with the proton beam using an adjustable frame, assisted by a cross-laser sight. The experimental setup involved continuous monitoring and control of the board via a host computer connected through a serial interface. A dedicated Python script running on the host system handled the collection of output data and the periodic acquisition of the CRAM content, known as the readback procedure. This acquisition was carried out at the maximum permitted frequency (approximately every 3 seconds) and concurrently with the execution of the application on the board. By comparing the original bitstream with the readback data, it was possible to detect SEUs in real-time. Application execution was halted only in the event of a timeout, in order to allow bitflip accumulation over time. A periodic check was performed to assess the availability of output data, and in the absence of a response, the board was reprogrammed. From the control room of the facility, irradiation sessions of arbitrary duration could be executed, and power cycles were applied to the board when required. The flow of the experiment is presented in Figure 5.4.

Fault injection Analysis

Following the analysis of the radiation test data, patterns of observed MBUs were identified. To assess whether this information could improve the accuracy of fault injection campaigns, two distinct campaigns were conducted. The first employed

a traditional SEU fault model, while the second incorporated both SEU and experimentally derived MBU patterns. These campaigns were aimed at evaluating the effectiveness of fault prediction based on actual irradiation outcomes.

To this end, a dedicated fault injection platform was developed to emulate fault accumulation within the configuration memory. The platform automates both the fault generation process and the evaluation of resulting output data. The in-house developed PyXEL framework was utilized to perform bit-level corruption within the CRAM content. PyXEL facilitates the injection of single or multiple bit-flips by decoding and manipulating the device's bitstream. The bitstream is progressively altered, one bit at a time, and reprogrammed into the FPGA until a critical failure results in a system crash. Faults are injected either at random or based on specific MBU patterns identified from the radiation data. A concurrent monitoring thread collects output data via the serial interface, which is subsequently parsed to produce a report detailing the occurrence and nature of the faults. In order to derive reliability curves and enable comparison with the experimental irradiation results, the fault injection algorithm later illustrated in Algorithm 1 was implemented. This algorithm supports both stochastic fault injection and pattern-based injection using insights gained from the post-irradiation analysis.

5.4.3 Experimental Results of Fault Injection on DMA

For the experiment, an off-the-shelf Digilent PYNQ-Z2 Development Board was utilized. This development board features a 28nm CMOS Z7020 Zynq 7-Series FPGA, which is integrated with a dual-core ARM Cortex-A9 Processor. The resources allocated for the benchmark implementation on the PYNQ-Z2 are summarized in Table 5.4. Due to the nature of programmable hardware, not all bit-flips that occurred resulted in errors in the output. In fact, some bit-flips may have affected unused resources of the device, which would not impact the functionality of the implemented design. Initially, the data from the radiation experiments have been analyzed, calculating the error rate, the particle cross-section of the peripheral, and the patterns of MBU. Following this, a comparison was made between the results of the fault injection and the radiation test data.

Table 5.4 Zynq Resource Utilization

Resources	*Used [#]*	*Available [#]*	*Usage [%]*
LUTs	28,413	53,200	53.41
Flip-Flops	33,317	106,400	31.31
Memories	2,080	17,400	11.95
BRAM	14	140	10.00

Radiation Data

Table 5.5 presents the values of energies and fluxes used during the radiation test experiment. Core errors were identified using a software voter running on the microprocessor. Golden values were rewritten every cycle to prevent any possible corruption. DMA failures, on the other hand, were tracked through interrupt signals. Table 5.6 provides the cross-section values for each energy, along with those for the CORDIC cores, while Figure 5.5 illustrates these values.

Table 5.5 Radiation test conditions: energy, flux and fluence.

Energy [MeV]	*Flux [$cm^{-2}s^{-1}$]*	*Fluence [cm^{-2}]*
29.31	4.124×10^{7}	9.173×10^{10}
50.80	4.024×10^{7}	6.064×10^{10}
69.71	4.110×10^{7}	2.124×10^{10}
101.34	4.319×10^{7}	2.415×10^{10}
151.18	4.094×10^{7}	1.226×10^{10}
200.00	4.144×10^{7}	3.942×10^{10}

Table 5.6 Particle cross-section per DMA and CORDIC

Energy [MeV]	*DMA Cross-section [cm^2]*	*CORDIC Cross-section [cm^2]*
29.31	2.725×10^{-11}	1.703×10^{-12}
50.80	1.567×10^{-10}	4.484×10^{-11}
69.71	2.354×10^{-10}	5.592×10^{-11}
101.34	2.899×10^{-10}	8.671×10^{-11}
151.34	3.263×10^{-10}	6.627×10^{-11}
200	3.171×10^{-10}	9.831×10^{-11}

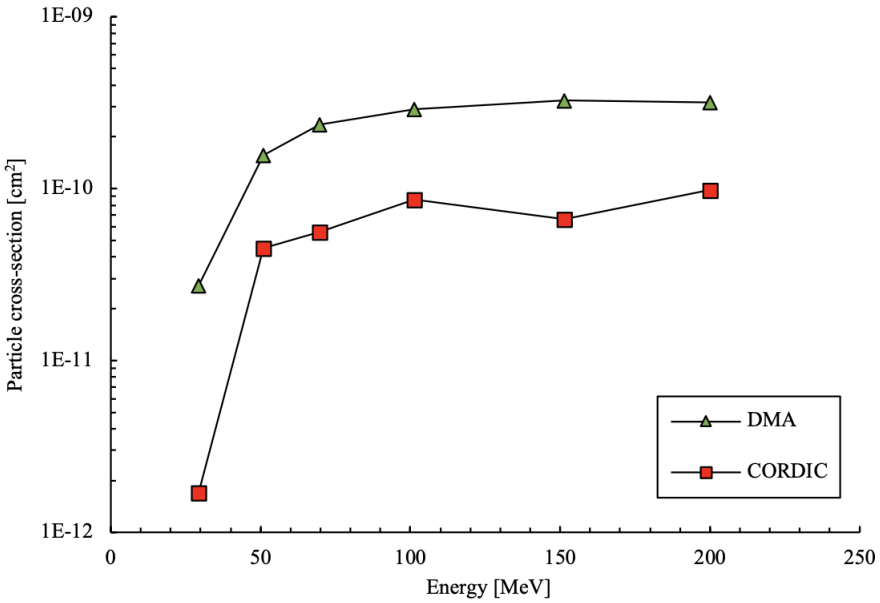

Fig. 5.5 Particle cross-section per DMA and CORDIC.

Such a consideration helps the designer identify the most critical parts of the system. In fact, DMA data transfer on the PYNQ-Z2 exhibits a higher sensitivity to proton-induced effects compared to the computational cores, as the cross-section is, on average, 20 times higher. Therefore, DMA modules must be considered critical and should be hardened accordingly. Further analysis has been conducted on MBU events and their correlation with DMA failures. An MBU is described as an event where a single particle striking the FPGA's silicon surface causes multiple adjacent bit-flips. Two bit-flips, a and b, are considered close when their Euclidean distance is less than or equal to $\sqrt{2}$. The distance is calculated as:

$$\text{dist}(a,b) = \sqrt{(bit(a) - bit(b))^2 + (frame(a) - frame(b))^2} \tag{5.2}$$

where $bit(\cdot)$ and $frame(\cdot)$ identify the coordinates of the bit-flip in the CRAM memory. By analyzing consecutive readbacks, we were able to isolate several MBU patterns that occurred during each readback cycle. The small number of bit-flips accumulating during each time window, combined with the large size of the CRAM (more than 32,335,848 bits), allowed us to detect groups of SEUs with a strong correlation both in time and space. Figure 4.3, already discussed in Chapter 4.3.2 illustrates the detected patterns, grouped into clusters based on the number of bit-flips.

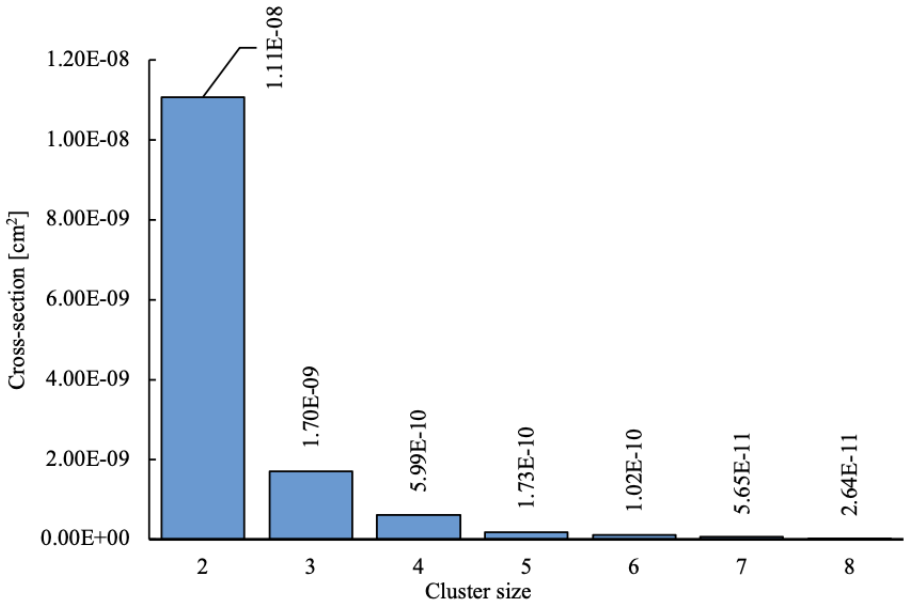

Fig. 5.6 Cross-section of MBU patterns.

The cross-section of the MBU clusters, shown in Figure 5.6, was computed as the ratio between the occurrence of a particular cluster size and the total number of particles that hit the board. The data indicate that the cross-section rate decreases as the cluster size increases, which suggests that larger MBU clusters require higher energy, making them rarer events.

Additionally, common-mode errors were observed and analyzed. We define common-mode faults as those where two or more modules fail simultaneously due to the same SEU or MBU. Three types of common-mode errors were distinguished:

- *DMA-DMA (DD)* errors occur when two DMA modules fail simultaneously due to the same SEU or MBU.
- *DMA-CORDIC (DC)* errors happen when DMA and one or more computational cores fail simultaneously due to the same SEU or MBU.
- *CORDIC-CORDIC (CC)* errors occur when two or more CORDIC cores fail simultaneously due to the same SEU or MBU.

Figure 5.7 presents the distribution of common-mode faults per MBU cluster size. As shown by the data, MBU occurrences can significantly impact the correctness of the application, leading to common-mode errors. The majority of these errors

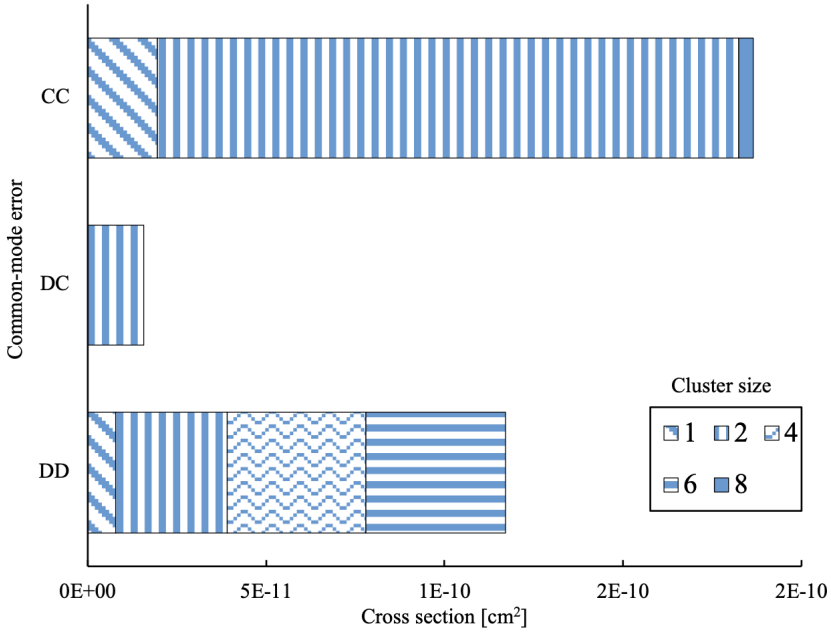

Fig. 5.7 Cross-section of common-mode errors.

occur with a 2-cluster MBU, mainly due to a higher frequency of this cluster size. Moreover, CC common-mode errors are more likely to occur because of the elevated number of cores and their proximity in the programmable logic. It's important to note that such corruption can also result from single bit-flips. Previous studies have demonstrated that physical isolation of resources can reduce these errors.

Fault Injection Data

In order to assess whether the data obtained from radiation tests can be leveraged to refine fault injection predictions, two separate campaigns were conducted. The first campaign involved 15,000 single-bit fault injections, with error accumulation, targeting bitstreams that led to DMA corruption, following the fault injection procedure outlined in Algoritm 1. The second campaign focused on injecting MBU patterns, derived from their occurrence rates observed in the radiation tests. These patterns were combined with single-bit injections, resulting in another set of 15,000 bit corruptions. The results of these campaigns, shown in Figure 5.8, highlight how fault injection using single-bit faults with accumulation can complement radiation test data, though minor discrepancies were noted. These differences are likely due to the MBU effects, which are more challenging to model accurately in fault injection

scenarios. A single particle strike may cause multiple bit-flips within a localized region, affecting the configuration differently than an equivalent number of randomly injected bit-flips. This behavior is complex to replicate in fault injection models due to variables such as device architecture and chip technology, which influence the pattern of bit corruptions. Consequently, fault injections based purely on SEU models tend to underestimate the failure rate.

Algorithm 1 Fault Injection with Accumulation

Require: bitstream, no_failures

1: Instantiate object `injector` of class `FaultInjector` with `bitstream` as argument
2: Instantiate object `listener` of class `SerialListener`
3: **for** $i = 0$ to $no\_failures$ **do**
4: Duplicate and target `bitstream`
5: **while** True **do**
6: Perform random bit-flip (single or multiple) from patterns
7: Program board with modified `bitstream`
8: **if** state of `listener` = `serial timeout` **then**
9: **break**
10: **end if**
11: **end while**
12: **end for**

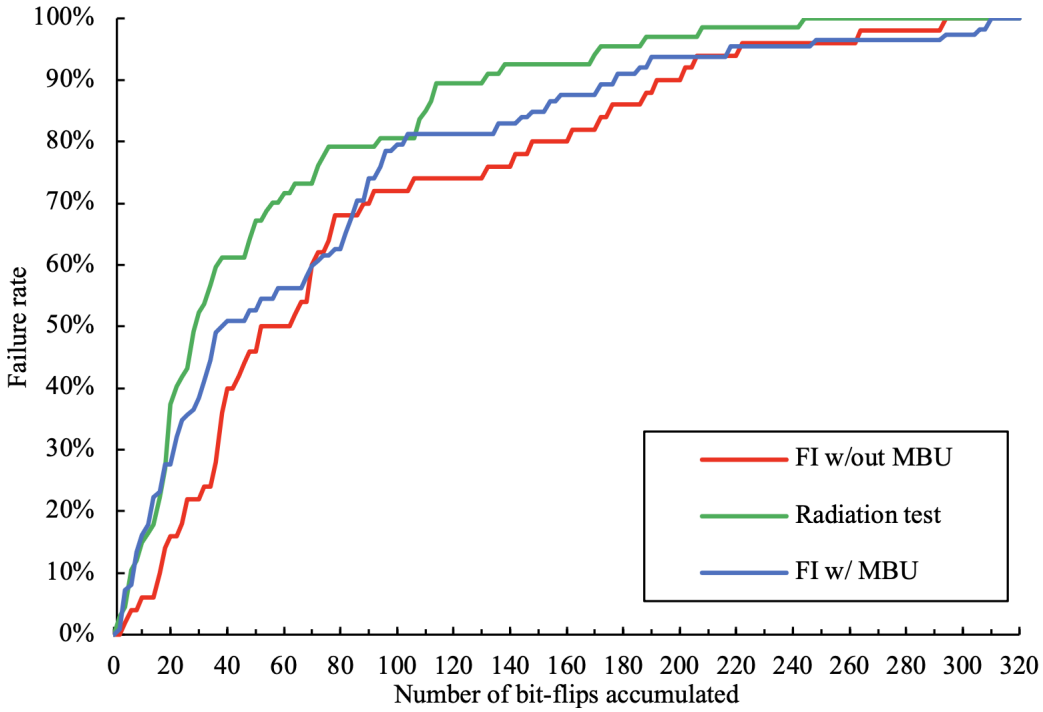

Fig. 5.8 Failure rate with the accumulation of fault injections and radiation test data.

Part III

Design Techniques for Reliable Computing in Space

Chapter 6

Design Methods for COTS FPGAs in Space

6.1 Analysis and Mitigation of Cross-Domain Errors

Cross-domain errors represent one of the most critical failure scenarios in radiation-hardened systems, as they undermine the effectiveness of redundancy schemes by simultaneously altering the behavior of two or more independent computational domains. This issue becomes particularly severe in the context of multi-core accelerators, where multiple identical processing units operate in parallel to enhance performance. In such systems, the corruption of multiple cores not only compromises individual outputs but effectively negates the benefits of hardware parallelism, making it imperative to devise strategies that can mitigate these effects.

To address this challenge, the primary contribution of this research is the development and evaluation of a set of layout design strategies tailored for multi-core computational engines [51]. These strategies are specifically aimed at enhancing system reliability in the presence of cross-domain faults. The proposed approaches leverage varying levels of resource isolation, with the objective of minimizing the use of shared logic and routing elements among cores. By reducing interdependencies at the physical level, the likelihood that a single radiation event affects multiple computational blocks is significantly diminished.

The effectiveness of these design strategies was validated through their implementation on a 16nm FinFET AMD UltraScale+ FPGA. A comprehensive proton irradiation campaign was conducted at the Paul Scherrer Institute (PSI) Proton Facility in Villigen, Switzerland, with proton energies spanning from 40 MeV to 150 MeV. Experimental results demonstrated that adopting a fine-grained isolation approach can reduce the cross-section of errors by up to 36% compared to standard layout techniques. This improvement is largely attributed to a decreased vulnerability of shared routing resources. Furthermore, these layout-based strategies can be effectively integrated with traditional redundancy mechanisms to further enhance overall system robustness.

6.1.1 State of the Art of Cross-Domain Failure Mitigation

One of the most established methods for mitigating radiation-induced faults in digital systems is hardware redundancy. Among the most widely adopted techniques are Duplication with Comparison and Triple Modular Redundancy (TMR), in which multiple instances of the same computational module, referred to as domains, are deployed in parallel. The outputs of these domains are subject to a voting mechanism (e.g., majority voting) to ensure that, even in the presence of a fault in one domain, the system can still retrieve the correct result from the majority.

However, one of the major vulnerabilities of these redundancy schemes lies in the phenomenon of cross-domain corruption. This occurs when a single particle-induced event, such as a SEU, simultaneously affects multiple redundant modules. For example, this may happen if the upset targets a configuration bit shared across domains, such as a routing resource or interconnect, thereby invalidating the fault tolerance provided by the voting mechanism.

To address these issues, several mitigation strategies have been developed, both at the software and hardware levels. Software-based solutions include code replication, checkpoint and rollback systems, or hybrid approaches combining both techniques. On the hardware side, fault detection and correction mechanisms such as scrubbing (e.g., partial reconfiguration) and majority voting logic are employed to counteract the effects of SEUs.

Despite these efforts, cross-domain errors remain a critical threat, as highlighted in previous studies that analyzed the vulnerability of redundant architectures to

shared-resource upsets. Some works have proposed custom design tools that improve resilience against such errors by controlling physical placement and connectivity, though these are often tailored to specific devices or configurations, limiting their general applicability.

More recently, placement and layout strategies have attracted growing interest as a complementary approach to traditional redundancy. For instance, vendor-specific flows like the Xilinx Isolation Design Flow (IDF) allow the designer to implement trusted routes and enforce physical separation between functional blocks, enhancing communication safety and reducing the likelihood of shared-resource corruption [52] [53]. These methods typically require minimal changes to the circuit netlist and can be used alongside other fault-tolerance techniques.

Nonetheless, a major limitation of these approaches is their dependency on proprietary features and internal architectural details that are not publicly documented. As such, they are not easily portable to non-Xilinx devices. Some researchers have attempted to generalize the concept by developing simplified isolation strategies that relax implementation complexity, but their effectiveness remains restricted to specific FPGA platforms.

To date, there is no broadly applicable placement methodology designed with reliability as a primary objective that can be integrated into mainstream FPGA design tools. This highlights the need for more accessible and device-agnostic placement strategies that can enhance system robustness in radiation-prone environments without relying on vendor-specific infrastructures.

6.1.2 Methodology of the Experiment

The design guidelines that were developed and subsequently applied to mitigate cross-domain errors in redundant systems are now explained. These strategies were specifically conceived to minimize the likelihood of shared configuration bits or routing paths between computational domains by physically increasing the spatial separation among modules.

The core principle underpinning the proposed methodology is module isolation. The main goal of this approach is to prevent the existence of common resources, such as configuration or functional bits, across domains, which could otherwise become potential single points of failure. If a shared bit is corrupted due to a radiation-

induced event, it may simultaneously compromise the operation of multiple domains, rendering redundancy ineffective.

Previous studies and vendor guidelines have highlighted that maintaining a physical separation of approximately 2 to 6 CLBs (introduced in Section 2.2) between modules significantly reduces the probability of such single points of failures. In particular, internal analyses by FPGA manufacturers suggest that this range provides a reliable isolation threshold that ensures no critical bit is shared between adjacent regions, thereby preserving the independence of the modules.

Ideally, a fully isolated layout, where each domain is completely decoupled at the granularity of individual configurable logic primitives, would guarantee the highest level of resilience against cross-domain errors. However, such an approach introduces substantial drawbacks in terms of resource utilization and implementation complexity. Specifically, the area overhead increases, placement becomes more constrained, and routing delays may rise, negatively affecting system performance.

As a result, a trade-off must be found between the degree of isolation and the associated overhead in terms of area, latency, and design time. To address this, we propose two distinct placement strategies tailored for multi-core accelerator architectures: fine-grained isolation and coarse-grained isolation. These strategies are analyzed in terms of their impact on resource sharing and system robustness, considering the balance between isolation granularity and implementation overhead. The subsequent subsections elaborate on the rationale behind each method and present a quantitative evaluation of the respective costs and benefits.

Proposed Isolation Approaches

In contemporary FPGA design workflows, designers frequently utilize proprietary pre-wrapped modules provided by FPGA vendors, known as Intellectual Properties (IPs). These modules are typically delivered as black boxes, offering limited or no visibility into their internal structure. Consequently, achieving isolation at the single-resource level becomes impractical or infeasible.

To overcome this challenge, we extend the isolation concept to a higher level of granularity by treating each computational core as an atomic unit. This leads to two main strategies, visually presented in Figure 6.1:

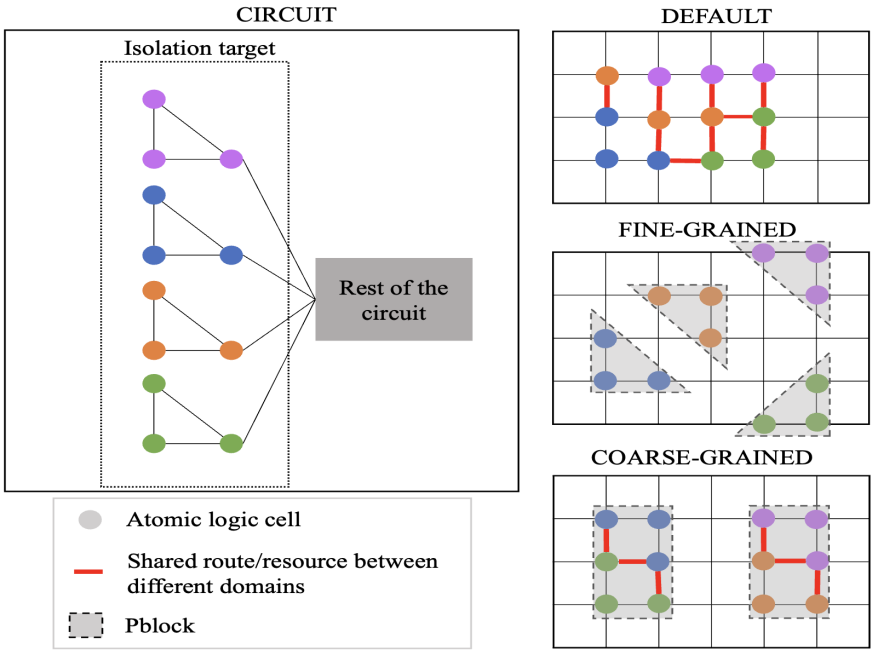

Fig. 6.1 Representation of the isolation concept used to reduce common resources among different modules.

- *Fine-grained isolation*, where each core is physically separated to minimize shared resources.
- *Coarse-grained isolation*, where a tunable number of cores are grouped, allowing some degree of resource sharing within defined placement regions.

Modern FPGA CAD tools enable explicit control over resource placement and routing. Among the available tools, placement blocks (pblocks) are designer-defined shapes used to constrain both logic and routing resources associated with specific modules. These constraints facilitate custom placement strategies with predefined isolation distances and core grouping policies.

However, implementing such placement strategies introduces a certain amount of overhead. Let us assume the need to instantiate a multi-core accelerator composed of n cores on the FPGA fabric. Let R_{tot} denote the total number of CLBs required for the isolation overhead. We also define:

- d_{iso}: isolation distance between adjacent pblocks (in CLBs),
- d_{pblock}: dimension of a square-shaped pblock (in CLBs),

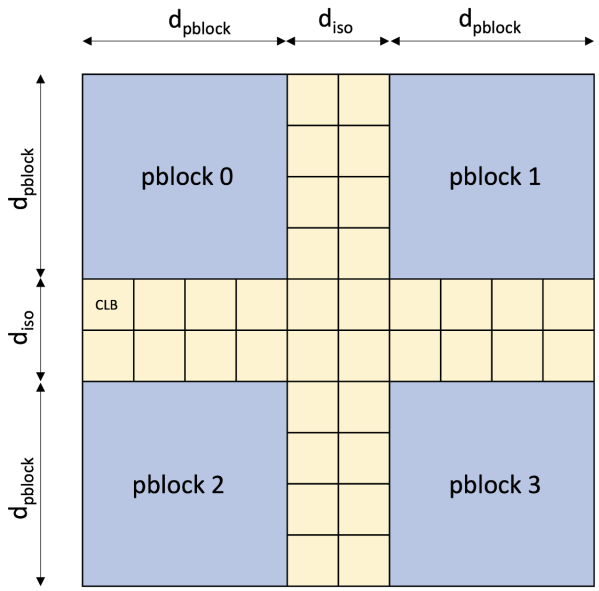

Fig. 6.2 Simplified view of an isolated design. The sum of CLBs in yellow represents the isolation overhead (R_{tot})

- q: the number of cores that are allowed to share the same pblock (i.e., $q = 0$ for full isolation).

The total resource overhead R_{tot} can be modeled as follows:

$$R_{\text{tot}} = \begin{cases} (2n - 2\sqrt{n}) \cdot d_{\text{iso}} \cdot d_{\text{pblock}} + (\sqrt{n} - 1) \cdot d_{\text{iso}}, & \text{if } q = 0 \\ q \cdot (n-1) \cdot d_{\text{iso}} \cdot d_{\text{pblock}}, & \text{if } q > 0 \end{cases} \tag{6.1}$$

This model assumes a square arrangement of cores. The values of d_{iso} and d_{pblock} are derived from experimental constraints and empirical guidelines. It is worth noting that Equation 6.1 yields an exact result only if n is a perfect square. Otherwise, a minor approximation error (around $\pm 2\%$) is introduced. As a comparison, a default implementation without isolation yields $R_{\text{tot}} = 0$. The physical meaning of the parameters is presented in Figure 6.2

While this overhead is non-negligible, it remains significantly lower than the $\sim 3\times$ overhead typically associated with modular redundancy approaches. Therefore, the proposed isolation strategy represents a viable trade-off between area cost and fault resilience, especially in radiation-hardened design contexts. The equation for different values of the parameters is plotted in Figure 6.3.

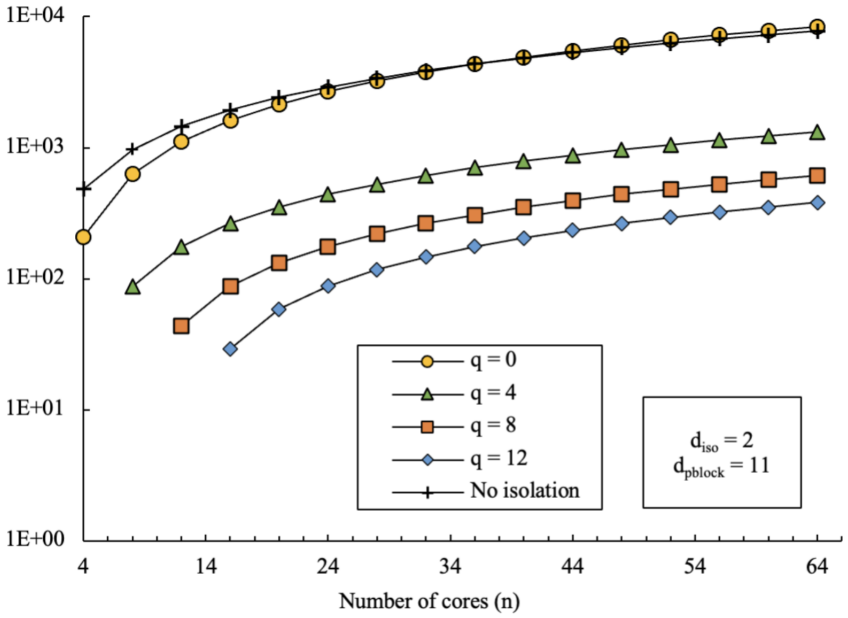

Fig. 6.3 Isolation resources overhead vs. number of cores for partial and fully isolated layouts with different q values and fixed parameters.

Application Benchmark

During the validation phase of the design, a benchmark was developed to manage both data transfer and computation in order to emulate the behavior of AI-oriented hardware accelerators based on multi-core processors. The benchmark leveraged both the programmable logic and the microprocessor system of the device.

Within the programmable logic, multiple CORDIC cores were implemented. This algorithm is widely used for computing transcendental functions, square roots, and vector rotations using only hardware operations (i.e., additions, subtractions, and bit-shifts).

Data transfer was handled by two DMA engines, configured in multichannel Scatter-Gather mode. Each DMA utilized its full capacity, operating all 16 available channels and using the maximum burst length per packet. These engines managed data transfer to and from 16 CORDIC cores each, for a total of 32 cores in the system. Each output from the CORDIC cores was tagged with an identifier (i.e., the DMA channel ID) to indicate which core had produced the data.

The hardwired microprocessor was connected to the two DMA engines through Master AXI High-Performance ports. Data was transferred using two directions:

Memory Mapped-to-Stream and Stream-to-Memory Mapped, each controlled by BDs allocated in dedicated BRAM memories. These BRAMs are volatile memories integrated into the programmable logic to enable faster access.

DMA interrupt signals were concatenated and managed properly to distinguish and track any potential corruption originating from the DMA modules.

A bare-metal software routine ran on the processor to print the contents of the output buffers on the host computer, write the buffer descriptors, and handle DMA interrupts. To ensure robustness, buffer descriptors were rewritten every cycle to scrub any potential corruption. Additionally, the processor performed TMR software voting on the outputs of the CORDIC cores. This process enabled the identification and logging of failing cores through their assigned IDs for further analysis.

6.1.3 Experimental Setup

The next step in our workflow has been carried out by exploiting the developed hardware benchmark to implement several solutions according to the methodology proposed in order to observe how different isolation parameters behave. Two of these configurations adopt fine- and coarse-grained isolation strategies, and are referred to as Matrix and Columns, respectively. A third configuration, named Default, has been implemented for comparison purposes.

To enforce placement constraints, pblocks were created to restrict the core resources to specific areas of the programmable logic. This allowed for accurate post-implementation analysis since the spatial position of each core is known in all three layouts. The configurations were implemented as follows:

- *Default*: This layout leverages the default placement strategy provided by the CAD tool, without any user-defined constraints. As a result, no isolation is applied among the cores.
- *Matrix*: This configuration adopts a matrix-structured, fine-grained isolation strategy with the accelerators placed in an 8×4 grid. Each core is isolated within its own pblock, placed 2 CLBs ($d_{\text{iso}} = 2$) away from adjacent blocks in both horizontal and vertical directions. The size of each pblock is 11×11 CLBs, which contains the necessary resources for a single core. Cores are assigned sequentially from the top-left of the matrix (e.g., row 0 contains cores

0–3; row 1 contains cores 4–7, etc.). This layout is characterized by $n = 32$ and $q = 0$, as in Equation 6.1.

- *Columns*: In this layout, cores are grouped into 8 columns, each containing 4 cores, with resource sharing allowed within each column, corresponding to a coarse-grained isolation approach. Each column is separated from the next by 2 CLBs ($d_{\text{iso}} = 2$) and is placed within a 44×11 CLB pblock (i.e., four times the size of the single-core pblock). Cores are sequentially placed in each column (e.g., column 0 includes cores 0–3; column 1 includes cores 4–7, etc.). This layout is characterized by $n = 32$ and $q = 4$.

It is important to highlight that the dimensions of the pblocks instantiated for both the Matrix and Columns layouts are the minimum required to accommodate the necessary core resources. As a result, the total number of CLBs used for implementing the cores within the pblocks is the same in both configurations, ensuring a fair comparison.

Radiation Test Setup

To evaluate the reliability of the proposed solutions under radiation-induced cross-domain errors, a dedicated radiation test campaign was carried out. Radiation testing provides the closest approximation to space environments by simulating radiation effects using high-energy particle beams.

Specialized facilities allow control over the energy and fluence of the particle flux, directing it at specific devices under test, such as circuit boards or silicon chips. Due to the hazardous nature of the radiation environment, data acquisition and power management for the device under test were fully automated using a custom in-house developed experimental platform.

In the irradiated room, the board was mounted and centered with respect to the beam using an adjustable frame. No components were removed, such as heatsinks or device lids, in order to replicate a realistic application environment and preserve the board in its COTS condition.

Data and power lines were routed to the control room to allow for board reconfiguration, output acquisition, and power cycling. The device was monitored and

configured by a dedicated host computer located in the control room and connected via a serial interface.

To manage the experiment, a Python-based application was developed within the PyXEL framework. This platform automatically handled output data acquisition and periodically captured the CRAM content through the readback process, achieving the maximum readback frequency supported by the device under test (approximately one readback acquisition every 10 seconds).

The configuration memory state and output data were periodically collected via the Universal Asynchronous Receiver-Transmitter (UART) port and were only halted at the end of each beam run. In the event of a UART timeout, typically caused by corruption of critical design bits, the system was able to autonomously detect the failure, reset the board and configuration memory, and automatically reprogram both the CRAM and the processor. This process effectively removed any corruption accumulated up to that point, without requiring dynamic scrubbing. In addition, UART logs were continuously printed out from the ARM processor that performed a basic comparison with golden outputs (triplicated in local memory) to detect which CORDIC core failed.

The radiation fluxes were initially tuned to induce approximately 2 to 5 bit-flips per readback acquisition. This rate was selected to ensure the collection of a statistically significant number of events over the course of the beam time, while still maintaining enough granularity to analyze individual occurrences between successive readback cycles.

The radiation beam runs were configured to deliver various energy levels, with the primary goal of emulating the range of radiation conditions typically encountered in LEO scenarios.

6.1.4 Results of Cross-Domain Errors Mitigation

First, the resource utilization for the implementation of the three placement layouts is presented in Table 6.1, together with information about Essential Bits. These bits are a circuit-dependent subset of the CRAM bits that are reported by the vendor tool as bits that, if corrupted, may lead to errors in the application. In this instance, AMD Xilinx design environment, Xilinx Vivado, has been used. This tool allows the user to design, synthesize and implement designs into AMD Xilinx FPGAs and

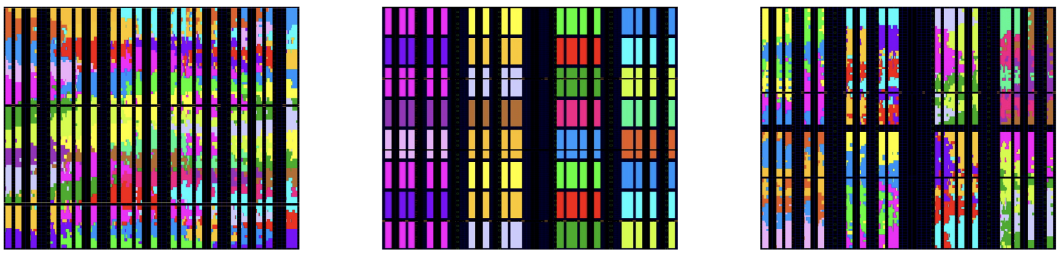

Fig. 6.4 Vivado Implementation views of the three layouts: (a) Default (b) Matrix (c) Columns. A different color is given to each of the cores.

SoCs. All these data underline a negligible difference among the solutions in terms of resource utilization and Essential Bits, thus the placement choice does not involve any overhead in terms of resources actually utilized. The actual implementations are presented in Figure 6.4 as in the Vivado Implementation view.

Table 6.1 Resources utilization and Essential Bits for Matrix, Columns, and Default layouts on ZCU104

Layout	*LUT*			*FF*			*BRAM*			*Essential Bits*		
	Used [#]	*Avail. [#]*	*Utiliz. [%]*	*Used [#]*	*Avail. [#]*	*Utiliz. [%]*	*Used [#]*	*Avail. [#]*	*Utiliz. [%]*	*Used [#]*	*Avail. [#]*	*Utiliz. [%]*
Default	30,633	230,400	13.30	34,032	460,800	7.39	14	312	4.39	10,791,647	131,763,200	8.19
Matrix	30,631	230,400	13.29	34,031	460,800	7.39	14	312	4.39	11,000,622	131,763,200	8.35
Columns	30,638	230,400	13.29	34,032	460,800	7.39	14	312	4.39	11,142,122	131,763,200	8.46

Finally, notice that the constraints we imposed involved only the cores, whereas the other modules (e.g., DMA, BRAMs) were left unconstrained. However, we ensured that the placement of this supporting logic did not involve any resources in the neighborhood of the isolation targets.

At first, we computed the SEU (Single Event Upset) cross-section per CRAM bit of the three layouts. The cross-section allows quantifying the probability of a given unit (e.g., CRAM bit, computational core) failing under the effect of a radiant excitation. In particular, the SEU cross-section σ_{SEU} is calculated as:

$$\sigma_{\text{SEU}} = \frac{N_{\text{SEU}}}{\Phi}$$

where N_{SEU} is the number of occurred SEUs and Φ is the total particle fluence for a given energy. Moreover, these values have been normalized by the total number of unmasked CRAM bits in the design.

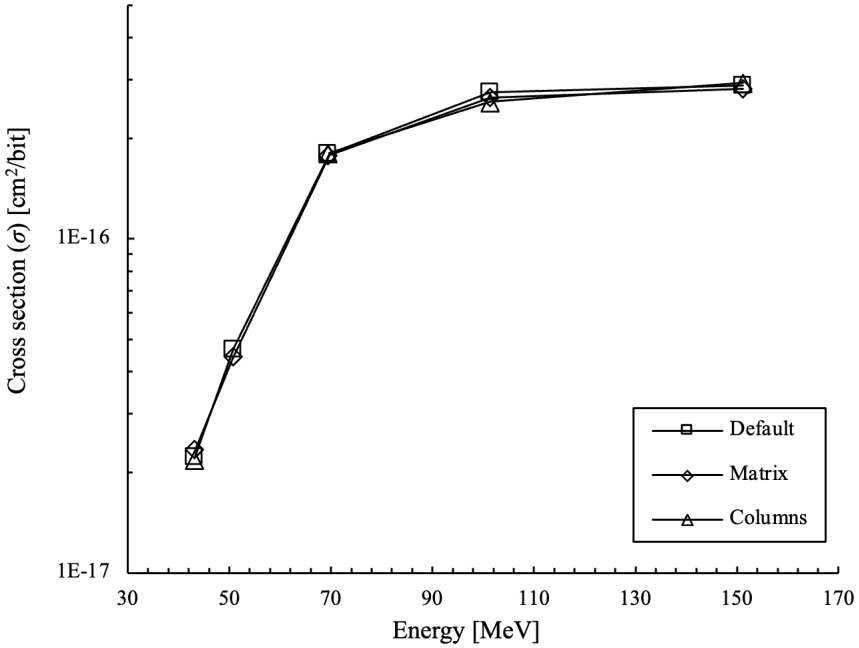

Fig. 6.5 SEU cross section per unmasked CRAM bit for the three layouts (95% confidence error bars).

Xilinx refers to unmasked bits as those configuration bits that may remain changeable, as opposed to the masked ones configuring the dynamic contents (FFs, LUTRAMs, and BRAMs), which are hidden by applying a binary mask on the bitstream. The distinction between masked and unmasked bits is necessary in order to collect actual radiation-caused bitflips, instead of bit modifications due to nominal circuit functioning.

The data in Figure 6.5 show a plateau of $2.8 \cdot 10^{-16}$ cm^2/bit, which represents a cross-section value in line with previous experiments on the same FPGA technology. As expected, the difference in SEU cross-section among the layouts is minimal, as it does not depend on the design. However, we can exploit this result to state that each of the three different layouts was subject to the same ratio of SEU per particle during the test and thus justify the comparison among the three implementations.

In order to quantify the reliability of the design with different layouts, the cross-section per single core has been computed. Thus, the ratio between the failures of a given core and the number of particles, normalized by the total amount of cores, together with the error bars with 95% confidence level and 10% fluence uncertainty, has been calculated. Each cell represents a core identifier, and for the Matrix layout

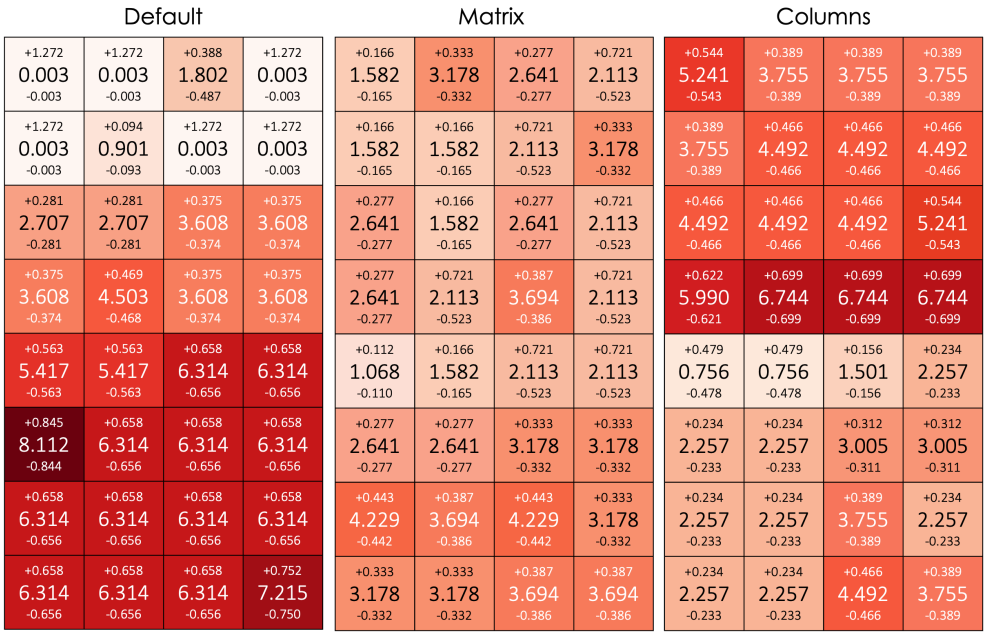

Fig. 6.6 Heatmaps of particle cross-section per core ($\cdot 10^{-11}$ cm^2) of the three layouts with 95% confidence intervals.

only, the identifier reflects the effective CORDIC placement on the programmable logic array, as shown in Figure 6.6

Computing the total number of core failures normalized by the particle, the Matrix layout obtained the lowest value equal to $2.655 \cdot 10^{-11}$ cm^2, followed by the Columns configuration ($3.675 \cdot 10^{-11}$ cm^2), and eventually the Default one ($4.139 \cdot 10^{-11}$ cm^2). As expected, concerning the Columns layout, the variance among failure rates of the same column is minimal, especially in the first half. This is due to the corruption of common resources within a given column that causes the failure of the cores belonging to that column.

The same behavior can be seen in the lower half of the Default layout cores, although spreading in a wider area since no isolation contains the propagation of the faults. The small differences among core failures in these regions may depend on bitflips affecting resources that program only that specific core, thus not shared.

This highlights the fact that there is no reason to believe that single-core corruption can be lowered using this approach, as this study is focused on the containment of cross-domain faults. For this reason, from the UART output log files, we collected

the number of common-mode errors that affected the design and the core identifiers associated with them.

As can be seen from the results, a fine-grained solution appears to be the safest one due to the lowest and less biased distribution of failures, although the isolation is not performed at the atomic level. Thus, the finest isolation of the cores seems to decrease the error propagation towards the closest routes and short lines, especially common routing resource corruption, since the neighborhood of the isolated modules is characterized by unused resources that do not affect the core computation, as opposed to a more congested placement.

Such a result may be of extreme importance when the data are streamed sequentially among the cores, so the result of a computation is dependent on the previous computation (i.e., no embarrassingly parallel problems).

6.2 Evaluation of Domain-based Isolation

Few research works have focused on Isolation Design Flow (IDF) or IDF-based architectures. One of the first notable contributions in this domain was proposing a technique that leverages IDF together with partial reconfiguration for Xilinx SRAM-based FPGAs, enabling online module relocation [54]. Their work highlighted the potential of combining logical isolation with runtime flexibility in reconfigurable systems. Building upon these ideas, the authors in [55] introduced a method aimed at easing bitstream relocation under IDF constraints, significantly simplifying the integration of isolation-aware modular design flows. More recently, [56] focused on integrating secure off-chip communication with reconfigurable sections of a design, thereby addressing trust and security aspects in dynamically reconfigurable FPGA architectures.

Although each of these approaches has proven to be highly effective in addressing specific challenges, current commercial FPGA design tools such as Xilinx Vivado still do not natively support partial reconfiguration combined with IDF. As a result, designers are often required to interface with external tools, which increases design time and complexity.

Regarding general design-for-reliability strategies, the work in [57] provides a comprehensive overview of commonly adopted techniques, including hardware

redundancy, error correction coding, and configuration memory scrubbing. Among these, TMR remains one of the most widely used and effective solutions for mitigating SEUs, particularly in safety-critical and radiation-prone environments.

However, to the best of our knowledge, no research to date has specifically explored the combined impact of IDF-based techniques and TMR on improving design reliability. This integration is the main focus of our investigation.

6.2.1 Methodology of the Experiment

A set of design practices aimed at reducing the complexity of floorplanning when applying replication-based mitigation techniques is now proposed. These guidelines are specifically conceived to simplify the floorplanning phase of the IDF, which is typically left to the designer and becomes exceedingly complex when state-of-the-art IDF is applied to replicated modules. Such complexity often leads to violations of the strict topological constraints imposed by IDF, thereby forcing the designer to abandon isolation entirely.

This approach mitigates this issue by reducing the number of isolated blocks through the strategic grouping of multiple modules within the same isolated region. However, careless relaxation of isolation constraints and indiscriminate aggregation of modules may nullify the benefits introduced by IDF. For instance, in the case of coarse-grained TMR, modules are replicated to perform the same computation independently. The results of these computations are then fed into a voter circuit to detect and correct potential errors. Since the voter filters out faults that affect only one replica, isolation should be prioritized between modules contributing to distinct inputs of the voter. Conversely, isolation constraints can be relaxed for modules that contribute to the same voter domain.

Reducing the number of isolated regions consequently lessens the floorplanning constraints and associated complexity. The integration of domain-based IDF into a traditional FPGA design flow can be organized into the following four stages:

- *Pre-Synthesis*: Isolated regions are defined. Unlike standard IDF approaches, modules intended to be grouped within the same isolated region must be reorganized within the design hierarchy. During this phase, it is crucial to avoid aggregating modules from different voter domains. Clock signals and

other inter-region connections must be declared in the placement constraint file to allow safe crossing between isolated areas.

- *Post-Synthesis*: Modules previously identified for isolation are marked with the corresponding property to inform the CAD tool. This step ensures that communication between isolated regions occurs exclusively via trusted routes.
- *Floorplanning*: Placement blocks (pblocks), which are collections of physical resources such as LUTs and PIPs, are manually instantiated to allocate the logic and routing of isolated modules. Each module's logic is restricted to its assigned pblock. It is essential to respect fencing rules during this step to ensure effective isolation. Design tools typically provide an estimate of the required resources to prevent overflow within a pblock.
- *Post-Implementation*: After implementation, the correctness of isolation is verified using a tool such as Vivado Isolation Verifier (VIV), which checks for violations in isolation rules, including misplaced modules, improper fence placement, or physical overlaps.

Benchmark Design

As an application under test, we developed a hardware-accelerated system targeting the Zynq-7000 platform. The system leverages the Xilinx CORDIC IP Core, which is implemented on the programmable logic and is widely used in aerospace applications for the computation of transcendental functions and digital signal processing. The CORDIC IP Core is managed by a software routine executed on the Cortex-A9 processing system (PS). Communication between the software and hardware components is achieved via AXI4 Interconnection Cores. Data transfers are handled through AXI DMA. When triggered by the fault injection platform running on the host computer, the software routine activates the cores on the PL and verifies their correctness. Specifically, the routine supplies a test vector to the CORDIC IP Core, compares the hardware output with the expected values, and sends an experiment report to the results collector module running on the host computer. The programmable logic design includes three AXI cores for communication and one computational core, namely the CORDIC module. This IP Core is connected to the PS via the AXI interface. To enhance fault tolerance, a triplicated (TMR-hardened) version of the benchmark circuit has been implemented. The CORDIC core and its associated

communication interfaces have been replicated three times. Each replica is accessible from the PS through the AXI Interconnect. The software routine performs majority voting on the outputs of the three replicas to determine the final result.

Error Classification

The software routine running on the processor system stimulates the cores on the programmable logic. The results obtained from the hardware are compared with the expected (golden) results to identify potential misbehaviors. If a mitigation approach based on replication is adopted, the software routine executes the computation on each replicated module. The outputs are then compared and voted to detect and potentially correct errors.

The misbehaviors observed during fault injection campaigns have been classified into four main categories:

- *Data Unavailability (DU)*: This condition occurs when no results can be received from the PL, typically due to faults affecting the communication modules.
- *Silent Data Corruption (SDC)*: This refers to errors in the results produced by the PL that are not immediately detectable, and can only be identified by comparing with the expected outputs. This occurs in the absence of replication, or when the voting mechanism selects an incorrect result.
- *Recoverable Data Corruption (RDC)*: In this case, different results are returned by the replicated cores, but the correct output is successfully recovered through the voting process.
- *Detectable Data Corruption (DDC)*: This occurs when the replicated cores return inconsistent results and it is not possible to determine a majority (e.g., in a TMR system, two cores produce differing outputs and the third is unavailable).

6.2.2 Comparison between Plain Design and Domain-based Isolation Results

Two versions of the plain benchmark design (i.e., without TMR) were implemented using the standard design flow and the state-of-the-art IDF. The reliability of both implementations was assessed through fault injection campaigns simulating SEUs in the configuration memory. In the IDF-based implementation, the top-level blocks (Zynq PS, AXI DMA, AXI Interconnect, and CORDIC) were isolated according to the guidelines provided in the IDF application notes.

Table 6.2 Essential bits of standard and IDF configurations

Design	*CM bits [#]*	*EB [#]*	*EB in CM [%]*
Standard	32,345,856	717,873	2.22
IDF	32,345,856	810,181	2.50

Table 6.2 reports the number of configuration memory (CM) bits and essential bits (EB) for both designs, as well as the percentage of EB with respect to the total CM bits. Both designs have the same number of CM bits, but the IDF implementation includes a slightly higher number of essential bits, increasing the percentage from 2.22% to 2.50%.

Table 6.3 Distribution of errors for standard and IDF designs (10,000 injections)

Error Type	*Standard*	*IDF*
DU [#]	103	97
SDC [#]	194	148
Total [#]	297 (2.97%)	245 (2.45%)

Each fault injection campaign involved 10,000 injections targeting only the essential bits of the respective design. The resulting error distributions are summarized in Table 6.3. The standard design experienced 297 errors, corresponding to a 2.97% error rate, while the IDF-based design showed 245 errors, reducing the error rate to 2.45%. This represents a 17.51% improvement in reliability achieved solely through changes in the placement constraints.

The overhead in terms of hardware resources is detailed in Table 6.4. Although the IDF-based design requires slightly more flip-flops and LUTs (approximately 10%

Table 6.4 Resource utilization for standard and IDF designs

Resources	*Standard*		*IDF*	
	Used [#]	*Utiliz. [%]*	*Used [#]*	*Utiliz. [%]*
LUTs	3,257	6.16	3,539	6.65
Flip-Flops	4,196	3.94	4,555	4.28
Memories	5	3.57	5	3.57

more), the overall resource utilization remains low and manageable. Nonetheless, for more complex designs, this increase might become more significant.

In order to evaluate the benefits introduced by IDF for replicated designs, we carried out additional fault injection analyses on the TMR version of the benchmark design. In particular, we evaluated the reliability of the circuit resulting from the standard design flow, proposed domains-based IDF, and non-domains-based IDF. Please note that even if the benchmark design under test is very small (i.e., less than 7% of resource utilization), it has not been possible to implement the state-of-the-art IDF. Indeed, the number of modules to isolate when TMR is applied makes it unfeasible to satisfy the isolation constraints.

The analyzed benchmark implementing TMR is described as follows:

- *Standard (unconstrained) configuration*: this benchmark has been implemented without IDF constraints, thus the modules are not isolated in this version. The block scheme of the modules at the higher level of the hierarchy is presented in Fig. 6.7.

- *Domains-based IDF configuration*: this design implements the domains-based isolation flow. The modules of a domain are grouped together in a block. The blocks are isolated using IDF. A single isolated block is composed of an AXI SmartConnect block, AXI DMA, and the CORDIC IP, as represented in Fig. 6.8.

- *Non-domains-based IDF configuration*: this last isolation pattern, represented in Fig. 6.9, couples together modules by task. AXI SmartConnect, DMA, and CORDIC blocks of the different domains are grouped between them. IDF is applied to these groups. This configuration has been proposed to evaluate the benefits of using the proposed domain-based aggregation policy with respect

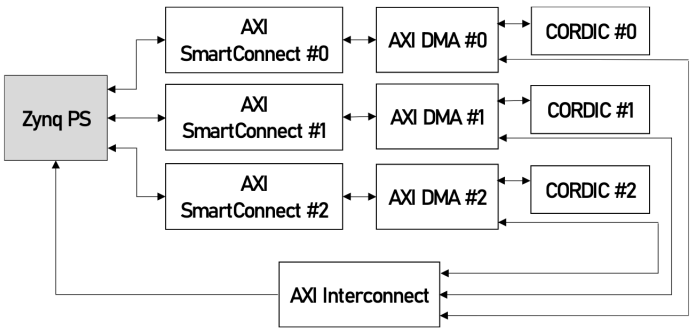

Fig. 6.7 Block scheme of the Standard design.

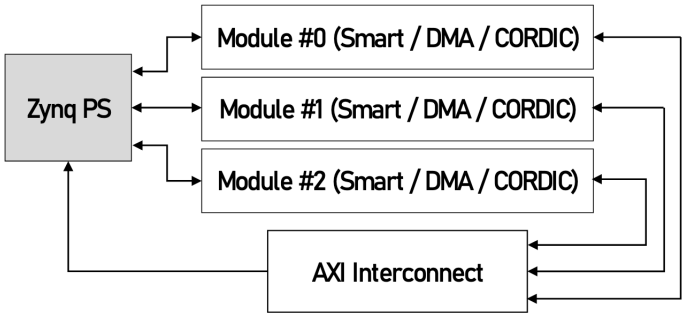

Fig. 6.8 Block scheme of the domain-based design.

to an aggregation policy aiming only to minimize the number of modules to be placed, without taking into account the concept of domains. Both of the IDF configurations isolate singularly the AXI Interconnect Module as it is recognized as a weak point of the design.

The fault injection campaigns consist of 10,000 injections emulating SEUs in configuration memory, affecting the essential bits of different versions of the TMR-hardened benchmark circuit implemented using different design flows. The total number of CM bits and the EB for each design has been reported in Table 6.5.

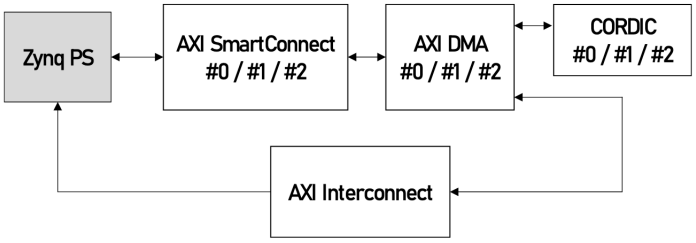

Fig. 6.9 Block scheme of the non-domain-based design.

Table 6.5 EB of standards, domains-based, and non-domains-based configurations

Design	*CM bits [#]*	*Essential Bits [#]*	**EB in CM [%]**
Standard TMR	32,345,856	2,811,321	8.69
Domains-based	32,345,856	2,930,999	9.06
Non-domains-based	32,345,856	3,002,114	9.28

Concerning the resource utilization, we observed a slightly higher requirement of logic resources when adopting IDF, similarly to what was obtained with the plain benchmark. In particular, IDF needs almost 1.5% of LUTs and 2% more FFs than the standard configuration when using IDF, as is reported in Table 6.6.

Table 6.6 Resources Utilization for Standard and Isolated Designs

Resources	*Standard TMR*		*Domains-based*		*Non-domains-based*	
	Used [#]	*Utiliz. [%]*	*Used [#]*	*Utiliz. [%]*	*Used [#]*	*Utiliz. [%]*
LUTs	12,878	24.21	13,064	24.56	13,204	24.82
Flip-Flops	17,706	16.64	17,713	16.65	18,037	16.95
Memories	15	10.71	15	10.71	15	10.71

We carried out the fault injections randomly targeting the EB of the designs. Due to the definition of EB, not all the injections will affect bits programming the used resources of the design. As a matter of fact, only some of them will cause an error in the output of the application. This can happen as a result of a fault injected in the used resources or the activation of an unused resource that leads to a conflict. Considering the three possible configurations we observed the following results:

- Standard configuration: in this configuration, we detected a percentage of 5.37% faulty behaviors. Of these, 50.47% were RDCs, 29.61% DDCs, 15.27% SDCs, and 4.65% of DUs.
- Domains-based IDF configuration: in this case, the fault injection campaign produced 2.89% of faulty outputs: in particular, 65.74% are RDCs, 30.10% DDCs and 4.16% consists in the DUs. No SDCs have been detected.
- Non-domains-based IDF configuration: the experiment resulted in a 3.13% error rate. In detail, we observed 45.37% of RDCs, 32.59% of DDCs, 2.87% of SDCs, and 19.17% of DUs.

The collected data are reported in detail in Table 6.7. Comparing both of the configurations with the unconstrained design, it can be observed that due to the IDF implementation, the total error rate is slightly dropped with focus on the silent errors (no SDC and 0.09% of the total with the domains-based and non-domains-based configurations, respectively).

Table 6.7 Distribution of errors for the standard TMR, domains-based and non-domains-based configurations for 10,000 SEUs

Error type	*Standard TMR*	*Domains-based*	*Non-domains-based*
RDC [#]	271	190	142
DDC [#]	159	87	102
SDC [#]	82	0	9
DU [#]	25	12	60
Total [#]	537	289	313

The domains-based design produced the lowest error rate as well as the highest RDC ratio among the analyzed implementations. Such achievements are also supported by the absence of SDCs, which represent the worst possible behavior due to their undetectability. The non-domains-based implementation also brought the decrease of SDC with respect to the standard design. However, this configuration appeared to be very sensitive to the DU, which occurred more than three times with respect to the domains-based case. This is likely due to its by-task aggregation, where a fault in one of the communication modules propagates to the communication infrastructure of all the domains.

Chapter 7

Design Methods for Rad-Hard FPGAs in Space

7.1 Toward Rad-Hard FPGA-based High-Performance Computations

In recent years, the computational demands and data transfer requirements of modern deep space applications have grown significantly. Traditional radiation-hardened (rad-hard) CPUs and controllers often fall short in meeting these evolving needs due to their limited performance. While Radiation-Hardened-by-Design (RHBD) ASICs offer high reliability and performance, their high development and production costs limit widespread adoption.

Conversely, SRAM-based FPGAs have emerged as a compelling alternative, offering a favorable balance of performance, flexibility, and cost-effectiveness. These devices support dynamic hardware reconfiguration, enabling advanced functionalities such as in-field error correction, parameter tuning, hardware-accelerated computation, and data conversion, all while maintaining low power consumption. However, their primary drawback lies in their vulnerability to radiation-induced soft errors. High-energy particles such as protons, neutrons, and heavy ions can interact with the silicon substrate, causing SEUs in configuration memory and SETs in combinational logic, potentially leading to critical system malfunctions.

To address these challenges, RHBD FPGAs combine the reprogrammability of SRAM-based devices with hardened design techniques that provide immunity to SEUs, although they remain susceptible to SETs. Despite the advantages, the availability of space-grade FPGAs is limited, particularly in the European market. Among the most prominent high-performance space-grade FPGAs are Xilinx Virtex-5QV (SRAM, 65 nm) and RT Kintex UltraScale (SRAM, 20 nm), as well as Microchip's RTG4 (Flash, 65 nm) and RT PolarFire (SRAM, 28 nm).

Within this context, the NanoXplore NG-Medium FPGA, recently certified as space-grade by the European Cooperation for Space Standardization (ECSS), represents a noteworthy addition. This 65 nm SRAM-based RHBD device integrates space-specific features such as a SpaceWire interface [58] and an on-chip scrubber module, along with traditional programmable logic. Furthermore, recent advancements in radiation-hardened technologies, including high-performance BRAMs and DSP blocks, have significantly expanded the range of applications that FPGAs can support, pushing the boundaries of their performance.

In this work, the implementation and timing optimization of a convolutional accelerator deployed on the r-VEX soft-core processor are investigated (details on r-VEX in Section 7.3). The accelerator executes convolution operations, Rectified Linear Unit (ReLU) activations, and Max Pooling using custom-developed assembly code for the VEX Instruction Set Architecture (ISA). Both single-core and multi-core implementations are explored to assess the trade-offs between performance and core count. The design is synthesized and evaluated on the programmable logic fabric of the NanoXplore NG-Medium FPGA.

7.1.1 State of the Rad-Hard FPGAs and Tools

The current body of literature primarily focuses on cross-section measurements and radiation testing reports that characterize the susceptibility of RHBD FPGAs to soft errors and SEEs [59–62]. In parallel, a growing number of studies have explored the deployment of artificial intelligence (AI) accelerators on these devices. For example, the authors in [63] present an AI architecture implemented on the RT Kintex UltraScale FPGA, leveraging Xilinx AI processing units, while in [64], AI models are deployed on the RT PolarFire device using the VectorBlox software development

kit. The comparative study in [65] evaluates various embedded platforms executing AI workloads, including a NanoXplore FPGA among the test devices.

Specifically regarding NanoXplore components, the vendor reports a Single-Event Upset (SEU) cross-section of approximately $1.00 \times 10^{-10}\,\mathrm{cm}^2\,\mathrm{bit}^{-1}$ for the NG-Medium FPGA when the Configuration Memory Integrity Check (CMIC) module is enabled [66]. This scrubber mechanism autonomously monitors and repairs corrupted configuration memory frames, thus enhancing the device's resilience to radiation effects. Additional studies include [67], where the NG-Medium is evaluated under high TID conditions such as those found in CERN environments, and [68], which investigates the performance of the NG-Large device with a strong emphasis on its DSP capabilities.

Despite these advancements, the feasibility and implementation of Machine Learning (ML) accelerators specifically tailored for rad-hard FPGAs, particularly recent devices like those developed by NanoXplore, remain underexplored. This gap in the literature is likely attributable to the limited availability of detailed architectural documentation, constraints in placement and routing flexibility, and relatively immature customization options in the corresponding Computer-Aided Design (CAD) toolchain when compared to more established FPGA vendors.

The development of custom Computer-Aided Design (CAD) tools has long been a flourishing area of research, primarily aimed at extending the capabilities or improving the efficiency of vendor-provided toolchains. In the context of Commercial-Off-The-Shelf (COTS) FPGAs, particularly for space applications, numerous efforts have been dedicated to optimizing placement and routing strategies with a focus on error mitigation. For instance, the authors in [29] introduce a custom Python library capable of analyzing the bitstream and performing targeted fault injections on AMD Xilinx hardware. Similarly, in [69], a tool developed in C extracts reliability metrics from the place-and-route model of AMD Xilinx devices, enabling hardened implementations. The work in [70] explores the routing architecture of Xilinx Series 7 FPGAs, while other studies [71, 72] investigate bitstream manipulation to support and automate partial reconfiguration for error recovery.

Several reverse-engineering attempts have also been reported [73–76], though with limited success. These efforts often target outdated FPGA families and achieve only partial results due to the complexity of proprietary formats. Notably, the absence of any public documentation regarding the NanoXplore bitstream format

makes such techniques infeasible for this family of devices. Furthermore, many of these approaches rely on the now-deprecated Xilinx Design Language (XDL) (i.e., previous version of the Vivado command-line set of instructions) or require intricate interfaces with the Vivado Design Suite, inherently restricting their applicability to AMD Xilinx architectures.

State-of-the-art FPGA routing methodologies typically involve two steps that can be summarised as:

1. the abstraction of the device architecture into a routing graph, where logic blocks are modeled as nodes and interconnects as edges

2. the routing phase, which applies algorithmic strategies to establish connections

Among the most widely adopted routing algorithms are Maze, A*, and Pathfinder [77, 78]. The Maze algorithm guarantees the shortest path in a grid-based representation but does not account for net contention, making its performance highly sensitive to net ordering. The A* algorithm introduces the Manhattan distance heuristic to enhance efficiency by pruning the search space. The Pathfinder algorithm dynamically balances path length and congestion by iteratively adjusting routing costs.

A prevalent trend in routing research involves parallelization, with strategies generally categorized as fine-grained or coarse-grained. Fine-grained parallelism accelerates the routing of individual nets, while coarse-grained approaches distribute sets of nets across different processing units to globally speed up the routing process. Coarse-grained methods often exploit architectural and spatial information to minimize synchronization overhead. For example, the authors in [79, 80] assign subsets of nets to individual CPU cores for parallel routing. While this boosts performance, it is inherently limited by the number of available cores. Further scalability can be achieved via GPU-based implementations, which enable massive parallelism. The fine-grained technique described in [81] similarly faces limitations due to CPU resource constraints.

As discussed later in this work, the routing model of NanoXplore devices diverges significantly from standard paradigms and imposes a set of additional constraints. These limitations restrict the use of conventional algorithms and necessitate the implementation of simpler yet effective alternatives. Finally, the complete absence of third-party CAD tools for NanoXplore FPGAs underscores the scarcity of publicly

available data beyond the front-end capabilities of the official Impulse tool (e.g., timing analysis and resource utilization). Consequently, no existing solution in the literature supports placement and routing analysis for NanoXplore devices. To the best of our knowledge, NXRouting represents the first tool to successfully address these challenges and achieve functional placement and routing analysis on this FPGA family.

7.1.2 Proposed Model of NanoXplore Architecture

Resource Hierarchy

The following section is dedicated to the description of the model we developed to overcome the limitations discussed earlier. The choices have been driven by details extracted using NanoXplore APIs. Particular focus is given to the routing of the core logic such as LUTs and FFs, since they represent the largest part of the routing effort in FPGAs.

The reconfigurable matrix is loosely based on the island-style FPGA model, where arrays of logic blocks are interposed with routing channels. IOBs, clock generators and, in the case of newer devices (i.e., NG-ULTRA), processor interfaces are located at the edges of the programmable logic.

The logic resources are spatially organized in a descending tree structure, with the highest level being the *Plane*, followed by *Zone*, *Network*, *Device*, and *Plug*, as shown in Figure 7.1(a). The *Plane* represents the entire programmable logic matrix and corresponds to the model of the FPGA chip (namely, the *Variant*). The *Zone* is the first subset of resources, which can include logic blocks (e.g., Tile and CGB), global routing resources (e.g., Mesh), or interfaces (e.g., Fence, JTAG, IOB). *Networks* are further subdivisions within a Zone, while *Devices* and *Plugs* represent physical resources and their pins, respectively.

For instance, an input pin (e.g., `Plug:I1`) of a LUT (e.g., `Device:LUT315`) in the NG-MEDIUM variant is shown in Figure 7.1(b). The *Plugs* of a *Device* can be further categorized as *Emitters* (i.e., output pins) and *Receivers* (i.e., input pins).

It is important to note that *Zone* names are globally unique (e.g., `TILE[15×10]` univocally identifies the Tile located in the 10th row and 15th column of the reconfigurable matrix), whereas *Network*, *Device*, and *Plug* names may be reused across

different Zones. Nevertheless, these identifiers remain unique within any given Zone. For example, a *Device* named `LUT315` may appear in both `TILE[15×10]` and `TILE[2×2]`, but within each Zone, it remains unique.

This hierarchical structure allows each element to be uniquely identified using a qualified tuple such as:

$$[\texttt{zone\_a}, \texttt{netwk\_b}, \texttt{dev\_c}, \texttt{plug\_d}]$$

which univocally identifies one and only one element.

Figure 7.4 illustrates an example of *Networks* and *Devices* distribution inside a *Tile* Zone. This structure is repeated consistently for every *Tile* in the programmable logic fabric.

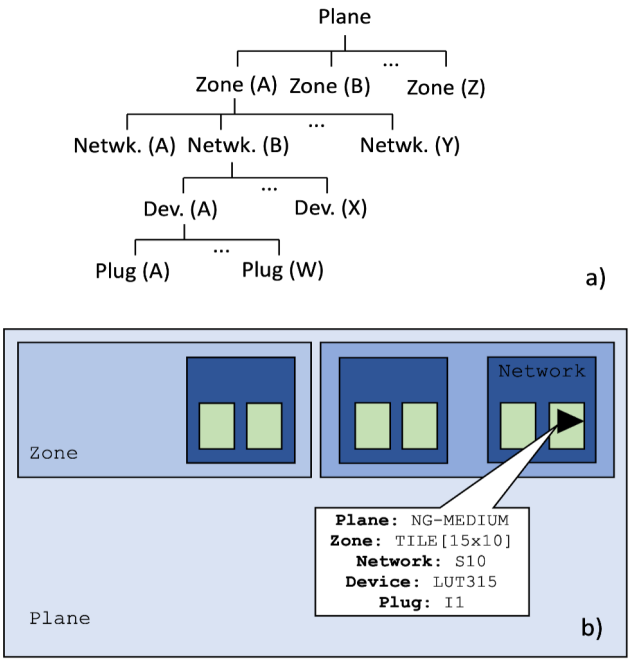

Fig. 7.1 (a) Hierarchical structure of the NanoXplore programmable logic. (b) Example of `Device:LUT315` and `Plug:I1` in the NG-MEDIUM variant.

Routing Model

Concerning the interconnection of the core logic, a clear distinction between routing inside and outside a *Tile* must be made. These represent the two cases of internal

Fig. 7.2 Example of Networks and Devices distribution within a Tile zone.

and general routing, cited in the NanoXplore documentation [82], and they impact timing differently. In the following subsection, we present how the model has been developed to address these two scenarios.

When dealing with routing inside a *Tile*, we first need to introduce the concept of the Functional Element (FE), which is a hardwired coupling of a LUT and a FF, physically placed in close proximity [82]. All common LUTs and FFs in the programmable logic are arranged this way, meaning that the output of the LUT must be connected to the input of the FF. For the NG-MEDIUM architecture, Figure 7.3(a) presents a simplified view of the core logic within a *Tile* and the FE structure. The output of the FE (i.e., the output of the FF) can then be routed to further routing structures.

This structural choice implies that the FF can be used either as a proper memory element (i.e., a D-type FF) or as a simple routing component (i.e., Bypass FF). Similarly, LUTs can serve as logic resources or behave as bypass elements when configured with an identity truth table. This scenario represents the vast majority of FE routing in the programmable logic. Although in some cases, the output of certain LUTs can also be directly routed to CY blocks for high-speed chain computations or to other LUTs (only in NG-ULTRA and NG-ULTRA 300) to fulfill specific timing constraints, these scenarios have been omitted from this study as they represent

usually a minor and optional part of a whole routed design. However, this extension is planned as future work in order to provide a full coverage of routing possibilities.

Timing analysis of intra-Tile routing shows that the signal delay is almost independent of the FE's position within the *Tile*, meaning that the delay between two adjacent FEs is comparable to that between two opposite corners of the *Tile*. This observation excludes the possibility of a classic routing model where routing matrices are interposed between each logic block and directly connect different resources [82]. Instead, it suggests the presence of a common routing entity shared by all resources within a *Tile*.

To ensure proper connectivity to all four inputs of the LUTs and to avoid conflicts in routing, we model each *Tile* with four dedicated routing matrices, one for each LUT input. Within each of these, the routing matrix is abstracted with two *Devices*, enabling the signals to be routed arbitrarily and reach any FE in the *Tile* without creating conflicts. Figure 7.4(b) shows the proposed model, while Figure 7.4(a) compares it to a common routing model (as adopted in most AMD Xilinx architectures).

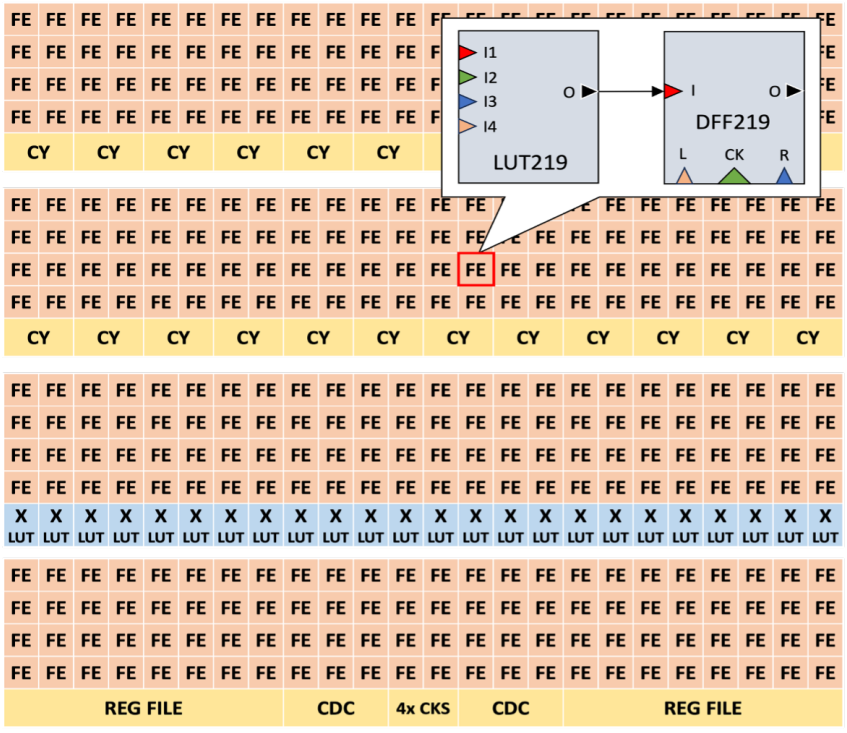

Fig. 7.3 Detail of a Tile in the NG-MEDIUM architecture with focus on the FE and its components. The Tile includes also additional logic such as Carry logic (CY), High-performance LUT (X-LUT), Register File and others.

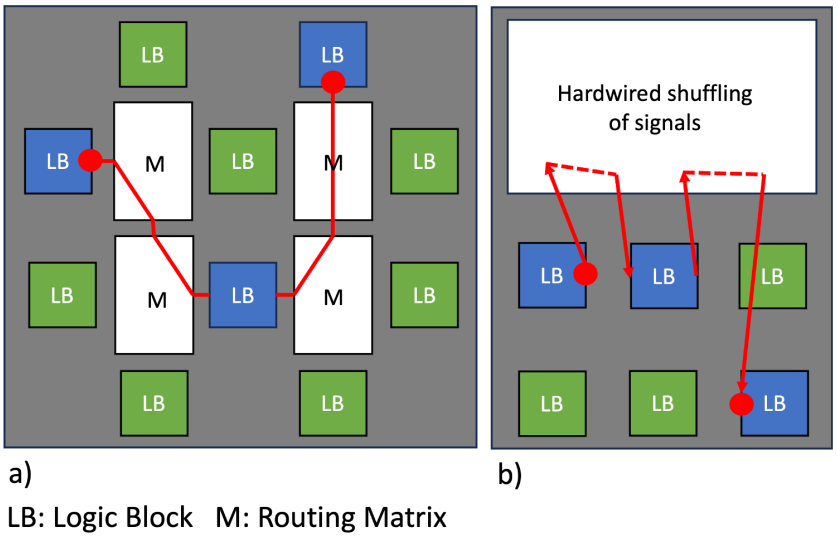

Fig. 7.4 The figure explains the difference between the two routing models inside a Tile. (a) Routing by means of switch matrices, where the delay is dependent on the physical distances among the routed logic blocks; (b) Routing by means of shuffling matrix, where the delay only roughly depends on how many times the route enters the matrix.

As designs become more complex and require more logic to be routed, a single *Tile* may not contain enough resources to fulfill the scope. Therefore, NanoXplore refers to general routing when dealing with *Tile*-to-*Tile* signals [82]. In order to manage the access to these external routing channels from the *Tile*, two blocks have been modeled and added to it.

Moreover, observing the displacement of *Tiles* and the zones that manage the general routing (namely, Meshes) in the configuration memory of NanoXplore devices, it is safe to assume that a *Tile* can be reached both from a Mesh located at the bottom and one located at the top. This assumption is also validated by the Impulse tool Graphic User Interface (GUI), where the *Tile* shows two sets of access points as well as two exiting points, both coupled on the upper and lower sides of the *Tile*.

When in the Mesh, the signal can now be routed globally, crossing other Meshes with a span of several ones in the horizontal direction (depending on the Variant) and ± 1 in the vertical direction. This solution allows the signal to reach any Mesh in the configurable plane starting from any point.

7.1.3 NXRouting: A Placement & Routing Tool for NanoXplore

Database Management

According to the routing directives described before, a database of resources and available connections has been built for each variant. Each row of the database gives the coordinates of a source point and a target point in the naming convention described in Section 7.1.2. Therefore, the database includes eight columns (i.e., two pairs of four coordinates). These data have been extracted through C++ NanoXplore APIs and their size is dependent on the variant and the amount of resources within (e.g., ranging from 76.8 MB for NG-MEDIUM up to 1.32 GB for NG-ULTRA).

To efficiently deal with the database exploration, two different loading configurations have been realized for the user to choose from. The first one, named *Full Loading*, performs the load of the entire contents of the database into cache memory, allowing the best performances in terms of computational speed. The drawbacks of this solution are an initial time overhead to load the full database (in the order of tens of seconds for the NG-MEDIUM up to tens of minutes for NG-ULTRA) and the limited size of cache memory available (some systems may have difficulties loading GBs of data).

The second solution is to load data only when requested (*Request Loading*); the system calls a load method that retrieves specific data from the database at runtime. This approach does not require any time overhead at start-up but it may involve several performance decreases during the routing exploration. Both solutions are available, and the user must decide which one to use at launch time depending on the needs.

Data Organization and Structures

Once the database loading routine has been chosen, the system starts to fetch data accordingly. A top structure is created and associated with the *Plane*, given the variant as input argument. The *Variant* class couples the *Plane* with the respective database unless the latter is manually overwritten by the user. The plane contains a Python dictionary that has the first level of hierarchy as keys (i.e., *Zones*) and the objects themselves as values, as well as other useful getter methods (e.g., `getName()`, `getID()`, `getZone(name)`, and so on). This approach is necessary in order to

avoid any duplication of a class object referred to the same element, so it will be unique as well as its attributes. This structure is repeated for each level of the hierarchy, allowing an agile exploration of the whole programmable logic. Each object also contains the reference to its parent, so that it is possible to retrieve it at any moment. This interface fully integrates with the Python object-oriented programming paradigm, for instance, exploiting iterators that allow the user to use cycles and indexing on the returned objects. Figure 7.5 gives an example of the flexibility of the exploration from the Python command line using NXRouting.

```
>>> plane = Plane(variant_name = 'NG-MEDIUM')
>>> plane.getDatabase().getName()
'NG_MEDIUM.db'
>>> zone_0 = plane.getZone(zone_name = 'TILE[15x10]')
>>> netw_0 = zone_0.getNetwork(netw_name = 'S10')
>>> for device in netw_0.getDevices():
>>>         device.getName()
>>>
LUT315
LUT316
[...]
>>> netw_1 = zone_0.getNetwork('S10')
>>> netw_0.getZone() == netw_1.getZone()
True
```

Fig. 7.5 Example of architectural exploration through pseudo-Python of the NXRouting tool. This feature mainly focuses on the user experience, presenting a well-known objected-oriented interface.

Routing Points

Among the architectural exploration, as stated before, a routing feature has been added to the tool in order to retrieve all the connection branches between two points in the programmable logic. When integrating the detailed model explained in Section 7.1.2 to this possibility, the tool allows the user to analyze a specific net; for instance, the observation of a route may suggest that a high delay is due to too many

global jumps among meshes and so it might be solved by constraining the placement within closer tiles.

The problem of routing this specific architecture can be seen as a directed graph where each node has one or more directed edges (i.e., the direction of the edge is defined as apriori, and it does not allow the signal to travel arbitrarily) linked to other nodes. In our solution, *Plugs* represent the nodes and the connections listed in the databases identify the edges and their direction. However, each *Plug* cannot allow incoming and outgoing branches since it is classified either as an emitter or receiver. To overcome this issue, the data structures defined in Section 7.1.3 may come in handy, returning the emitters associated with a *Device* and its receivers. This approach extends the concept of the node to an object that includes a *Device*, *Emitters*, and *Receivers*, although still allowing the user to route specific *Plugs*.

For the routing algorithm, the *Breadth-First Search* (BFS) has been chosen due to its well-known structure and performance [83]. In particular, the BFS algorithm allows to search for a target by exploring all the nodes at the present depth of the graph before moving to the next depth level, in contrast to the *Depth-First Search* (DFS) that analyses a single node branch in its full depth before expanding to the next one. The implementation of the algorithm is described below in Algorithm 2. The implementation loosely follows the standard BFS, although it contains substantial differences. Each emitter will be searched through by popping it out from the queue list. A for loop iterates over every possible receiver that can be reached from the analyzed node and each emitter of the same device of that receiver will be added to the queue as well. The algorithm stops searching when the target receiver has been found.

However, the standard algorithm faces some limitations. In particular, plain BFS is not able to manage graphs with loops as they may arise infinite cycles of search for such branches. To overcome this problem, three flags have been added to the *Plugs*: visited, queued, and routed. The visited one keeps track of which plug has already been completely searched while the queued flag identifies a plug already in the queue waiting to be searched. The routed flag, instead, has been added to allow multiple consequent routings and to know which node has already been taken by a net or is still vacant. At the end of the routing routine, the first two flags will be reset. The results are saved in a *Net* object that contains each branch the algorithm followed to

Algorithm 2 | Algorithm for BFS of NanoXplore Architectural Graph

Input: *source_emitter, target_receiver*

queue = [*source_emitter*]

```
1:  while len(queue) > 0 do
2:      node = queue.pop(0)
3:      if not node is routed then
4:          if not node is visited then
5:              for receiver in node.receivers do
6:                  if not receiver = target_receiver then
7:                      if not receiver is visited then
8:                          for emitter in receiver.getDevice().getEmitters() do
9:                              if not emitter is queued then
10:                                 update emitter parent nodes
11:                     queue.append(emitter)
12:                     emitter is queued
13:                 else:
14:                     update node parent nodes
15:                     break from while
16:     node is visited
```

reach the target (i.e., parent nodes in Algorithm 2). In Figure 7.6, a scheme presents the whole flow of the NXRouting described so far.

GPU Acceleration

This paragraph describes the implementation of the routing algorithm and its integration within the NVIDIA Cuda-C environment. CUDA (Compute Unified Device Architecture) is the most widely used framework for GPU parallel programming due to its maturity, extensive hardware deployment, and rich ecosystem of optimized libraries [84]. Its tight integration with NVIDIA GPUs enables high and predictable performance, establishing CUDA as the de facto standard in HPC, AI, scientific simulations, and advanced graphics. This study leverages two levels of parallelization: fine-grained and coarse-grained. The fine-grained parallelization handles concurrent searching among the nodes in the queue during the routing of a single net, while the

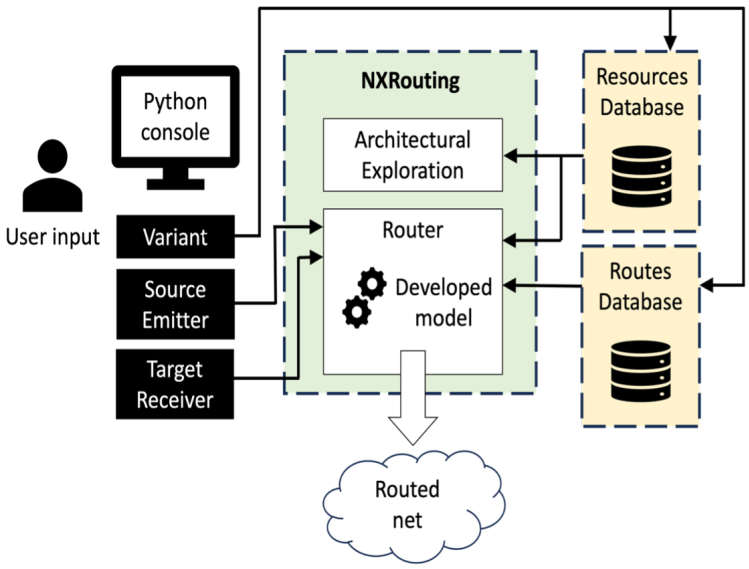

Fig. 7.6 Scheme of NXRouting showing the features and the databases associated.

coarse-grained parallelization manages the routing of multiple nets at once. To support GPU parallelization, new data structures have been introduced, which need to be allocated in memory N times, where N is the number of nodes to explore in parallel, differing from the structures introduced in Section 7.1.2. The main data structures include: a *local labeling array* that marks nodes when they are discovered for the first time and keeps track of which nodes were already found in previous iterations; *previous* and *next buffers*, a double buffer that at each iteration contains the indices of nodes discovered in the previous iteration and the indices of nodes discovered in the current iteration, with their roles being swapped at each iteration; and a *backtracking array* that saves, for each discovered node, which node discovered it. This array allows the shortest path from source to sink to be reconstructed once the sink is found, by traversing the backtracking array from target to source. Before starting the next iteration, the BFS algorithm swaps the pointers of the two buffers (previous and next) and clears the content of the next buffer. This clearing is done by setting an index indicating that the length of the valid content is zero, without copying or deleting the data, thus optimizing performance using the *ping-pong buffering* technique. When the second iteration starts, the previous buffer contains the nodes discovered in the previous iteration, and the GPU kernel launches as many threads as there are new nodes. If a valid path is found, the iterations stop and the shortest path is reconstructed using the backtracking array. Local data structures can either

be reinitialized for other nets or deallocated when no longer needed. The next step is to extend the algorithm to handle the routing of multiple nets in parallel. A key issue in this case is that when one net finds its shortest path, it needs to remove the resources it used from the pool. The other nets, which are still performing their routing, must then exclude any nodes discovered after traversing a node removed from the pool, but this check could significantly reduce performance, especially as the number of parallel routes increases. To avoid this performance loss, information about the FPGA architecture, as described in Section 7.1.2, can be leveraged. The coarse-grained technique consists of dividing the list of nets into $N+1$ sets, where N is the number of Tiles. One set contains nets with source and sink located in different Zones, while the other N sets contain nets with source and sink located within the same Tile. Each set will route its nets sequentially, creating $N+1$ sequential routing flows. The main advantage of this approach is that the N sets containing nets within the same Tile can be routed in parallel, as they use distinct routing resources. When one of the $N-1$ sets removes resources from the pool, these resources are used only by nets in that set, avoiding conflicts. The $N+1$ set, which contains nets with source and sink in different Tiles or Zones, cannot be routed concurrently with the others because it may remove shared resources. Therefore, this set should either be routed first, before the other N sets, or after the other sets have completed their routing.

7.1.4 NXRouting Performances and Timings

7.2 Performance Analysis

This section presents the analyses and results collected to quantify the performances of our tool. First, profiling the two database loading configurations provided an efficient way to discriminate the choice according to the feature the user wants to utilize. The time the system has taken to fetch each level of hierarchy is presented in Table 7.1 for two variants (NG-MEDIUM and NG-ULTRA, respectively).

Table 7.1 Timing comparison between Full Loading and Request Loading configurations.

Config.	*Start-Up [µs]*		*getZone [µs]*		*getNetwork [µs]*		*getDevice [µs]*		*getEmit. [µs]*	
	NG-MED	*NG-ULT*	*NG-MED*	*NG-ULT*	*NG-MED*	*NG-ULT*	*NG-MED*	*NG-ULT*	*NG-MED*	*NG-ULT*
Full Load.	$19 \cdot 10^6$	$452 \cdot 10^6$	4.768	5.323	4.034	4.873	4.733	5.543	4.768	5.323
Req. Load.	-	-	189.452	378.574	245.829	201.551	387.002	568.719	639.402	739.321

As stated in Section 7.1.3, the Full Loading configuration mainly affects the start-up timing as all the data are loaded into memory. In particular, the NG-ULTRA analysis takes up to 8 minutes to complete the loading, a result not optimal for a quick peek into the architecture. However, this overhead is required only at the start of the tool and does not affect any subsequent computations. The Request Loading configuration does not require any start-up overhead, although getters take significantly more time compared to the Full Loading. A slight increase in the time as the hierarchy level goes from Zone to Plug has also been registered in this configuration. This is due to the more complex queries and filters that the system demands from the database as the hierarchy level of the objects decreases.

Moreover, an analysis of the timing required for NXRouting to fully route several benchmarks using the `routePoints()` method on the NG-MEDIUM architecture has been performed. Four benchmarks have been selected from the ITC'99 benchmark suite [85], specifically B03, B06, B09, and B12. These designs have been chosen for their well-known architecture and behavior, and for covering a wide section of the routing complexity spectrum. Table 7.2 shows the number of nets to be routed for each benchmark.

Table 7.2 Number of nets to be routed for each benchmark

Benchmark	*Nets [#]*
B03	253
B06	68
B09	284
B12	1688

Results about the routing time have been collected from three different routing configurations exploiting parallelization at various levels. The first configuration, sequential, implements no parallel computation, performing the routing of signals in series. The second and third configurations exploit fine and coarse-grained parallelization described in the previous sections. Unfortunately, no other third-party router is available in the literature for comparison at the time being. Table 7.3 presents the obtained results. The tool has been implemented on an NVIDIA Jetson Nano GPU [86].

The results show a decrease in routing time using the Fine-Coarse-Grained configuration with respect to the other ones. However, the timing still seems to scale with the number of nets to route. This is mainly due to the unfeasibility of issuing a

Table 7.3 Routing time comparison across different router configurations.

Benchmark	*B03 [s]*	*B06 [s]*	*B09 [s]*	*B12 [s]*
Sequential Router	143.17	62.73	300.05	433.14
Fine-Grained Router	0.81	0.47	0.89	4.62
Fine-Coarse-Grained Router	0.60	0.47	0.58	2.64

number of kernels equal to the amount of nets, thereby preventing an embarrassingly parallelization of the problem. Moreover, these results demonstrate the feasibility of GPU acceleration for routing purpouses compared to the CPU-based one. As a future work, further investigations on the comparison among different GPU architectures are planned.

7.3 Evaluation of a Neural Network Accelerator on Rad-Hard FPGA

7.3.1 The r-VEX Architecture

The r-VEX processor is a Very Long Instruction Word (VLIW) soft core designed by Delft University, based on the VLIW EXample (VEX) Instruction Set Architecture (ISA), introduced in [87], and loosely modeled after the HP/STMicroelectronics LX/ST200 ISA for VLIW embedded cores. It offers scalable technology, including instruction width, instruction set, and functional units, all written in VHDL. The Very Long Instruction Word (VLIW) architectures execute multiple operations in parallel by packing them into a single, wide instruction word determined entirely at compile time [88]. This simplifies the processor's control logic and can lead to high performance and energy efficiency when parallelism is abundant. The r-VEX processor is an appealing choice for research studies as benchmark because it offers a fully customizable architecture that can be easily reconfigured to explore different parallelism and pipeline structures. Moreover, the availability of a dedicated toolchain and cycle-accurate simulator makes r-VEX suitable for academic exploration of performance, energy efficiency, and fault tolerance in VLIW-based systems.

By default, the core includes 4 Arithmetic and Logic Units (ALUs), 2 Multiply Logic Units (MULs), 1 branch control unit, 1 memory access unit, 64 32-bit general-purpose registers (GRs), and 8 1-bit branch registers (BRs). It also supports a 32 kB

data and instruction memory cache. As a VLIW processor, each instruction includes (also parameterizable) 4 32-bit long syllables, which encode opcode bits, register address bits, and metadata bits in a single syllable. These syllables can be seen as individual RISC instructions.

The VEX standard specifies three types of immediate operands: short immediate operands, branch offset immediate operands, and long immediate operands. Each syllable in the VEX standard has a switch field for immediates, comprising 2 bits to identify the type of immediate operand within the syllable. In r-VEX, syllables also include additional metadata bits, L and F, where the L bit indicates whether the syllable is the final syllable in an instruction, and the F bit indicates whether it is the first syllable in an instruction.

The default configuration of the r-VEX and the syllable layouts are shown in Figure 7.7. The standard set of instructions includes 73 operations, with the possibility to extend it through custom instructions. Data and instructions can be directly edited in their respective memories by modifying the associated hardware description files. After the traditional design flow (synthesis, implementation, and bitstream generation), the design can be implemented on the FPGA.

7.3.2 Convolution in VEX Assembly

In order to effectively leverage the computational capabilities of the r-VEX processor, a convolutional accelerator based on a reduced version of the AlexNet architecture was implemented as the benchmark application. AlexNet is a convolutional neural network (CNN) particularly effective for handwritten digit recognition, and is composed of multiple layers, including 2D convolution, Max Pooling, and ReLU activation functions [89]. The network receives a square input image and processes it through a sequence of convolutional, pooling, and activation layers to return a probability distribution over the digits 0 through 9.

Due to the limited data memory available on the r-VEX processor (implemented in Block RAM of the FPGA), only a sub-section of the full network could be implemented. However, care was taken to ensure that this subset included representative instances of the various types of layers present in the original architecture. As illustrated in Figure 7.8, the implemented structure comprises two convolutional

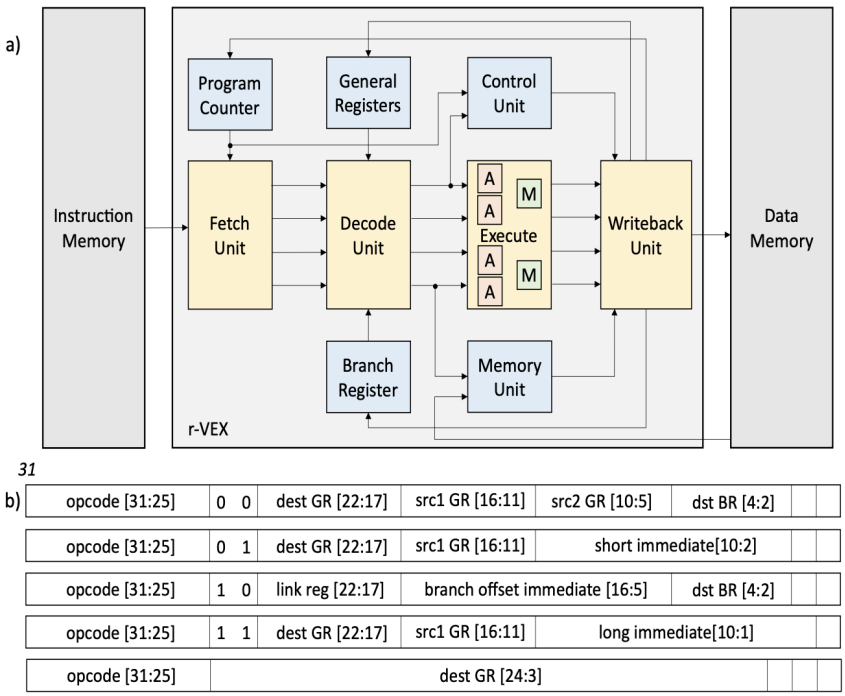

Fig. 7.7 a) Scheme of r-VEX architecture. b) Different instructions supported by the VEX ISA.

layers interleaved with two Max Pooling layers. The specific parameters of the implemented network section are detailed in Table 7.4.

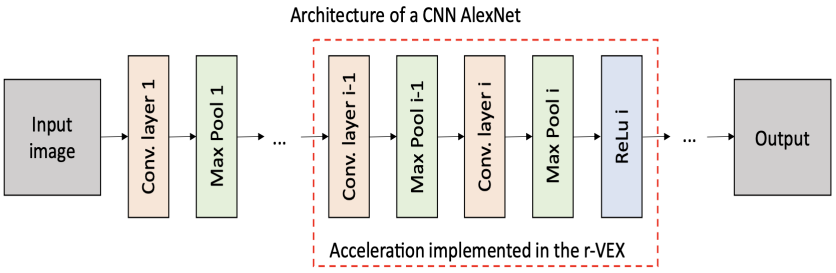

Fig. 7.8 Architecture of the AlexNet and its sub-section selected for the acceleration through the r-VEX.

Since the r-VEX requires programming in assembly language, custom assembly code based on the VEX instruction set was developed to implement the convolution operation. The implementation leverages branch registers to compare row and column indices during computation. Additionally, the code was designed to be parametrizable, allowing for dynamic adjustment of stride and padding values without requiring changes to the core structure of the code. The algorithm developed

Table 7.4 Parameters of the Implemented Network Layers

Layer	*Input*	*Stride*	*Padding*	*Filter*	*Output*
Conv. Layer 1	12×12	1	1	3×3	12×12
Max Pool 1	12×12	0	1	2×2	11×11
Conv. Layer 2	11×11	0	1	3×3	9×9
Max Pool 2	9×9	0	1	2×2	8×8

is presented in Algorithm 3. First, the input image has been uploaded from the data memory to the registers in order to minimize further accesses, then various functions for the different layers have been developed and pointed through the *goto* opcode in case of need. In the implemented configuration, the computation involved a 12×12 input image, producing an 8×8 output.

7.3.3 Experimental Analysis and Results

In order to perform a comprehensive evaluation of the implemented accelerator, a set of analyses has been conducted on the design. Specifically, three different implementations of the benchmark have been investigated, varying the number of cores (namely, 1-core, 2-core, and 3-core configurations) in addition to the baseline design. The objective of this study is to identify an optimal trade-off between performance improvements and the number of computing cores deployed.

The analysis is structured in two main phases. First, a detailed evaluation of resource utilization and mapping directives has been performed, leveraging the NXmap design toolchain. Second, a timing analysis has been conducted through Static Timing Analysis (STA) to assess the critical path and timing behavior of the designs.

The following subsections provide a detailed description of the methodology adopted and present the results obtained from these evaluations.

Hardware Benchmark

First, the configurable parameters of the r-VEX processor were selected. In particular, the number of registers, the number of syllable issues, and the available computational units were maximized according to the specific r-VEX version used

Algorithm 3 Algorithm for convolution, Max Pool and ReLu in VEX assembly.

Input: *image[N][N]*

Load image from Data Memory:

```
while branch_col_register do
    while branch_row_register do
        mov image[i][j], gen_register j+N*i
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto Convolution
```

Convolution:

```
while branch_col_register do
    while branch_row_register do
        execute convolution in row_register, col_register
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto Max Pool
```

Max Pool:

```
while branch_col_register do
    while branch_row_register do
        max gen_register i, gen_register i+1
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto ReLU
```

ReLu:

```
while branch_col_register do
    while branch_row_register do
        max gen_register i, 0
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
```

in this work. The selected configuration parameters are reported in Table 7.5. These settings remained constant across all the cores implemented in the designs.

Table 7.5 Implemented r-VEX Configurable Parameters

Parameter	*Value*
General Purpose Registers [#]	64
Branch Registers [#]	8
Arithmetic Logic Units (ALUs) [#]	4
Multipliers [#]	2
Syllable Issues [#]	4
Data Memory [kB]	32

The design was implemented using the NanoXplore toolchain. Specifically, the *NXmap* tool was used to perform the synthesis, placement, and routing phases. During synthesis, appropriate mapping directives were added to ensure correct management of specific components; for instance, multi-read/write port RAMs, which cannot be automatically handled by the tool, were explicitly mapped as DFF-based structures. Placement was entirely managed by the tool, without any user-defined placement constraints.

The NanoXplore toolchain provides comprehensive reports on resource utilization and timing analysis, in addition to graphical visualizations of the synthesized design. These visual aids highlight resource instances and routing paths, providing a global overview of the implemented architecture.

Following synthesis and implementation, the bitstream was generated. The bitstream was subsequently programmed into the Brave NG-Medium NX1H35S CLGA625 DevKit [82] using the NXbase software.

Resource utilization results extracted from the NXmap reports for the three different designs are presented in Table 7.6. As expected, the resource utilization scales proportionally with the number of cores implemented.

It is worth noting that the Brave NX-Medium NX1H35S device does not include an embedded Universal Asynchronous Receiver-Transmitter (UART) controller. Therefore, a configurable UART peripheral was developed and integrated into the circuit to enable the visualization of the r-VEX output data. The UART output was

Table 7.6 Resource Utilization of Each Design

Design	*4-LUT*	*DFF*	*Carry Logic*	*DSP*	*BRAM*
1-core	6,098 (19%)	1,901 (6%)	298 (4%)	3 (3%)	4 (8%)
2-core	11,714 (37%)	3,801 (12%)	596 (8%)	6 (6%)	8 (15%)
3-core	17,946 (56%)	5,703 (18%)	894 (12%)	9 (9%)	12 (22%)

mapped to a 3.3V programmable I/O pin, and an external USB-to-UART converter module was employed to transmit the data to a host PC, operating at a baud rate of 115,200.

All benchmarks were executed using the 25 MHz clock provided by the onboard oscillator. For performance analysis, several reports were extracted from the NXmap tool, including the Static Timing Analysis (STA) netlist and the routed VHDL netlist. The following subsections describe the methodology adopted for timing analysis and performance evaluation.

Evaluation of Transition Delays

The NXmap tool is capable of generating Standard Delay Format (SDF) reports, following the IEEE specification that represents circuit delays using an ASCII format. Typically, SDF files report delays associated with $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions for the circuit nets, although the format supports up to 12 different delay values corresponding to all possible transitions among logic states (0, 1, Z, and X).

For the purpose of timing analysis, the average of the maximum and minimum delays for the $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions was computed for each design to highlight their timing performances. Moreover, timing results were categorized according to the type of interconnection, as described in [82], by cross-referencing the SDF reports with the routed netlist.

Thus, transition timing data were collected and divided into three categories:

- *Intra-tile resources*: internal routing
- *Inter-tile resources*: general routing
- *Global routing resources*: low-skew resources for clock and reset signals

The obtained results are summarized in Tables 7.7, 7.8, and 7.9, respectively. As can be observed, transition delays tend to increase as the spatial distribution of resources within the programmable logic becomes broader.

Table 7.7 Average Transition Timing of Internal Routing Resources

Design	$0 \rightarrow 1_{min}$ *[ns]*	$0 \rightarrow 1_{max}$ *[ns]*	$1 \rightarrow 0_{min}$ *[ns]*	$1 \rightarrow 0_{max}$ *[ns]*
1-core	2.278	2.378	2.278	2.378
2-core	2.497	2.601	2.497	2.601
3-core	2.537	2.642	2.537	2.642

Table 7.8 Average Transition Timing of General Routing Resources

Design	$0 \rightarrow 1_{min}$ *[ns]*	$0 \rightarrow 1_{max}$ *[ns]*	$1 \rightarrow 0_{min}$ *[ns]*	$1 \rightarrow 0_{max}$ *[ns]*
1-core	2.742	2.827	2.742	2.827
2-core	2.801	2.922	2.801	2.922
3-core	2.905	2.985	2.905	2.985

Table 7.9 Average Transition Timing of Low Skew Routing Resources

Design	$0 \rightarrow 1_{min}$ *[ns]*	$0 \rightarrow 1_{max}$ *[ns]*	$1 \rightarrow 0_{min}$ *[ns]*	$1 \rightarrow 0_{max}$ *[ns]*
1-core	6.605	6.667	6.605	6.667
2-core	6.587	6.646	6.587	6.646
3-core	6.683	6.743	6.683	6.743

As shown in the tables, the 1-core design exhibits the lowest overall delay values. However, the maximum variation observed across the designs occurs in the internal routing resources, with a difference of approximately $0.3 \cdot 10^{-9}$ s, corresponding to roughly 0.75% of the clock period. Therefore, such a variation can be considered negligible.

An exception is observed for the low-skew routing delays, where the 2-core design demonstrates slightly lower values compared to the others. This result can be attributed to a more uniform distribution of resources around the clock and reset sources in the 2-core layout.

Data Delay

The STA report provides a detailed evaluation of the timing performance of a circuit by decomposing the design into timing paths and computing the signal propagation delay along each path. This methodology allows the detection of possible timing violations and the assessment of the criticality of different paths. Each timing path consists of a start point, an endpoint, and the combinational logic in between.

In the context of this work, the main motivation for conducting an STA was to observe the evolution of timing performances as the number of implemented cores increased. The analysis was carried out under typical scenario conditions for all three designs. Specifically, the data delays of the ten longest timing paths in the 1-core design were identified and compared against the corresponding paths in the 2-core and 3-core implementations. The results of this comparison are depicted in Figure 7.9.

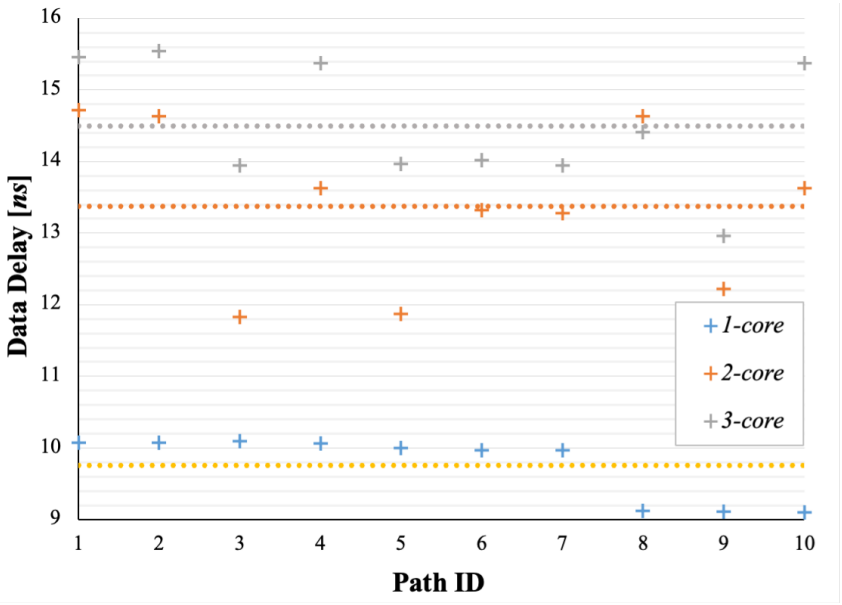

Fig. 7.9 Plot of the data delay of the 10 longest paths of the 1-core design compared with the same paths of the other design. The dotted lines represent the average values.

As can be observed, data delays are clearly influenced by the increased resource utilization. Notably, the displacement in the average data delay between the 1-core design and the multi-core designs is approximately 3.620 ns (+37.10%) for the 2-core configuration and 4.742 ns (+48.59%) for the 3-core configuration. However, it is

important to highlight that this increase in delay is not strictly proportional to the rise in resource utilization.

Furthermore, the maximum operating frequency was also affected by placement congestion, as summarized in Table 7.10. The NXmap tool provides the capability to create and edit a timing constraint file. This file allows designers to specify constraints on nets and timing paths, aiming to improve timing performances as long as no violation flags are raised.

For this study, timing constraints were applied to attempt optimization of the 2-core and 3-core designs. The focus was on the same ten critical paths previously analyzed. A constraint iteration process was performed, applying a 0.05 ns step reduction on the maximum allowed data delay for each path until a timing violation was encountered. Figure 7.10 presents the results of this constraint-driven optimization.

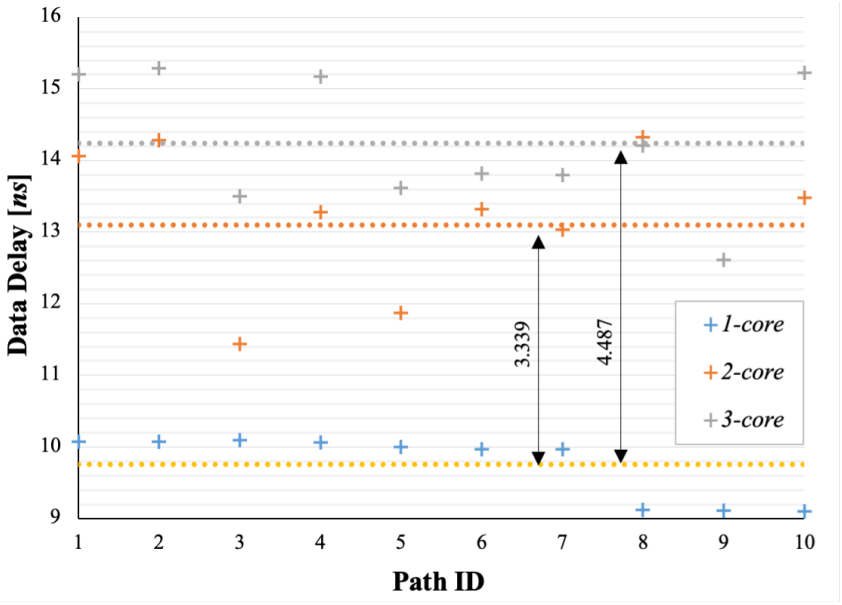

Fig. 7.10 Plot of the data delay of the 10 longest paths of the 1-core design compared with the same paths of the other design with timing constraints. The dotted lines represent the average values.

The application of timing constraints led to a decrease in data delays across all ten paths for both the 2-core and 3-core designs. After optimization, the average delay differences from the 1-core baseline were reduced to 3.339 ns (+34.21%) for the 2-core and 4.487 ns (+45.98%) for the 3-core designs. However, it is worth

noting that no improvements in the maximum working frequencies were observed despite the reduction in individual path delays.

Table 7.10 Maximum Operating Frequency for Each Design

Design	Maximum Frequency [MHz]
1-core	32.753
2-core	27.586
3-core	23.384

Placement Constrained Data Delay

In typical hardware-accelerated applications, the programmable logic of an FPGA is often densely populated with interconnections, memories, processors, and additional control logic managing data flow to and from the accelerator. Consequently, the resources dedicated to accelerated computations are typically constrained within congested sub-sections of the configurable logic fabric. These placement constraints can have a substantial impact on timing performance, due to the varying proximity to critical sources such as clock and reset signals, as well as output pin interfaces.

In this study, placement constraints were deliberately applied to the cores in order to emulate placement congestion. Using the NXRouting tool, regions within the programmable logic were created and assigned to the cores. After iterative trials of implementation, the minimum region size necessary to accommodate a single core was determined to be 11×6 tiles, according to the NXmap coordinate system. If the size is smaller, the implementation results in failure.

Two distinct layout configurations, denoted as L1 and L2, were proposed and analyzed for each design. These layouts vary primarily in their distance from the I/O buffers, which are located at the top-left corner of the device. In L1 layouts, cores are placed closer to the top-left corner, while in L2 layouts, cores are positioned to maximize their distance from this region. Figure 7.11 illustrates the two placement strategies. It is worth noting that, in the case of the 3-core designs, two core regions overlap due to the limited availability of programmable resources, leading to a more congested placement with intrinsic worst timing.

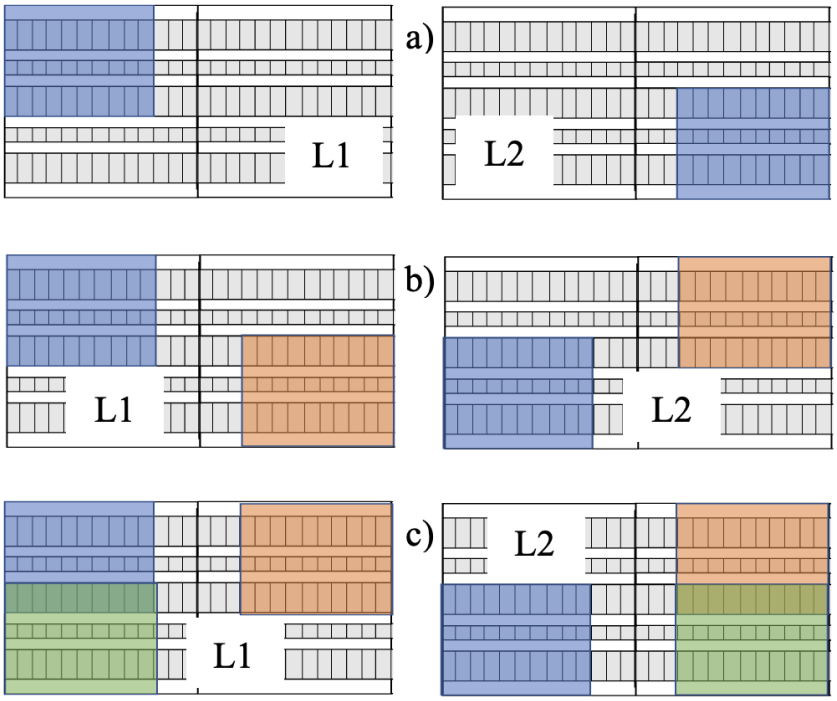

Fig. 7.11 Cores layouts analyzed for a) 1-core b) 2-core c) 3-core. Each colored block represents a constrained region for a single core.

An STA was performed for each layout configuration, focusing on the same ten critical timing paths analyzed previously. The timing delay data were collected and represented using a box plot, shown in Figure 7.12.

Overall, the results indicate that increasing the distance from the I/O buffers correlates with a deterioration in timing performance. The most pronounced difference was observed in the 1-core layouts, where the maximum degradation reached approximately 6 ns. This observation is consistent with the expectation that longer routing paths to the output buffers introduce additional delay. However, in the 2-core and 3-core designs, the impact appears less severe due to relatively shorter routing paths.

These findings underline the strong dependence of timing-sensitive modules, such as accelerators, on their physical placement within the FPGA fabric. Moreover, as previously shown, timing constraints alone may not be sufficient to compensate for the performance degradation caused by placement-induced congestion. Therefore, careful placement analysis and optimization are essential for fully exploiting hardware acceleration in complex and resource-congested programmable logic envi-

ronments. Finally, it is important to note that the application of placement constraints did not lead to any significant increase in overall resource utilization.

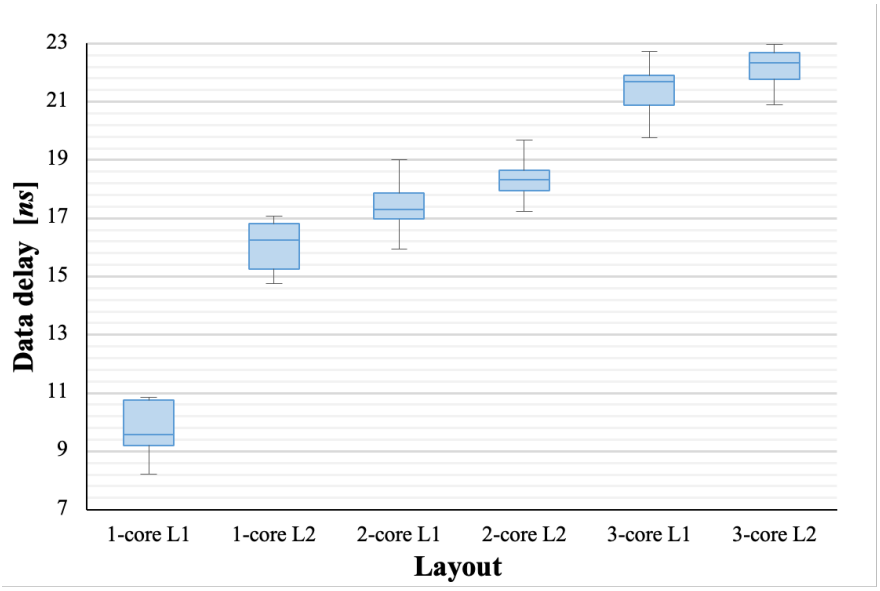

Fig. 7.12 Plot of the data delay of the 10 paths analyzed in Section V.C of the constrained layouts. L1 versions minimize the distance from the output buffer, while L2 maximizes it.

Chapter 8

Conclusions and Future Directions

8.1 Conclusions

This dissertation has presented and investigated a range of methodologies and techniques aimed at enabling the development, evaluation, and analysis of radiation-hardened Field Programmable Gate Arrays (FPGAs) for high-performance computing applications. Throughout this work, a comprehensive suite of tools and methods has been proposed and developed to enhance both the reliability and computational efficiency of reconfigurable hardware platforms intended for operation in radiation-intense environments.

Significant contributions have been made to the field of radiation-tolerant reconfigurable computing, addressing critical aspects of fault mitigation and performance optimization. Comprehensive methodologies encompassing diverse fault models, device technologies, and computational architectures have been introduced, demonstrating the effectiveness of the proposed approaches in supporting the deployment of FPGAs in demanding high-performance applications. Extensive methodology development and evaluation campaigns have been conducted across multiple levels of abstraction, from physical device characterization to system-level fault tolerance, leveraging techniques such as fault injection, accelerated radiation testing, and analytical modeling. The techniques developed within this research have provided valuable insights into the resilience and performance trade-offs inherent to FPGA-based systems, representing a significant advancement toward their adoption in both aerospace and terrestrial high-reliability applications.

Particular focus has been placed on the evaluation of the sensitivity of high-performance FPGA designs to Single Event Upsets, supported by the development of dedicated analysis frameworks. The thorough characterization and mitigation of SEUs affecting both configuration memory and user logic have been addressed, underpinned by fault emulation platforms and experimental irradiation campaigns targeting state-of-the-art SRAM- and FinFET-based FPGA technologies.

Additionally, special emphasis has been given to the assessment of soft-core processors, custom accelerators, and the integration of fault-tolerant designs within data-intensive applications, such as artificial intelligence workloads. The analysis has systematically highlighted the impact of radiation-induced faults on high-throughput designs, revealing how fault propagation can degrade both system performance and correctness, even in architectures employing traditional redundancy techniques.

The reliability of interconnect architectures has also been rigorously evaluated, with a particular focus on AXI-based data transfer modules, which were identified as potential sources of systemic failures under radiation exposure. Experimental findings have shown that silent data corruption remains a critical challenge, despite the presence of architectural redundancy, emphasizing the necessity of carefully engineered interconnect protection strategies for radiation-hardened systems.

Finally, this research culminated in the development of a comprehensive framework dedicated to the analysis of European radiation-hardened FPGA platforms. This advancement paves the way for future research efforts focusing on custom placement and routing strategies tailored for these architectures, an area that remains largely unexplored.

8.2 Future Directions

The methodologies, approaches, and results presented in this dissertation have explored the potential of radiation-hardened reconfigurable devices for fault-tolerant, high-demanding, and AI-based computing applications, as well as the challenges associated with analyzing fault sensitivity and the effects of radiation-induced faults in such systems. The outcomes of this research provide an important foundation for further work in this domain and enable numerous potential directions for future exploration.

First, further research efforts are needed to develop increasingly efficient fault detection, diagnosis, and analysis strategies specifically tailored for radiation-hardened, high-performance reconfigurable platforms. The initial steps taken in this dissertation toward proposing robust evaluation flows to be integrated early and seamlessly into the design process should be expanded to include new methodologies, particularly static analysis techniques. Low-level, implementation-aware static analysis, capable of predicting potential points of failure and fault propagation paths resulting from configuration memory upsets, presents a promising lightweight methodology to be employed in the early stages of design for AI and HPC applications on radiation-hardened FPGAs.

Second, the in-depth knowledge of the mapping between configuration memory faults and the corresponding hardware resources developed during this research can be leveraged to enable more accurate and effective fault impact analyses. This understanding can be instrumental in the development of custom placement and routing strategies specifically designed for radiation-hardened devices, aiming to enhance the intrinsic robustness of AI accelerators and high-performance computing architectures by strategically optimizing the physical layout to minimize vulnerability to radiation-induced faults.

Third, additional investigation is required to design and evaluate fault-tolerant systems with high reliability and robustness, particularly focusing on the interplay between redundancy mechanisms and radiation-induced fault behaviors. The detailed analysis of cross-domain errors emerged as a critical area needing deeper understanding. Future work should build on the methodologies developed in this dissertation to detect and mitigate such vulnerabilities, thereby enhancing the effectiveness of redundancy-based hardening approaches in AI-based and high-throughput computing systems.

Furthermore, specific emphasis should be placed on the evaluation of AI models deployed on radiation-hardened FPGAs, where hardware-aware reliability assessment frameworks must be developed to comprehensively capture the sensitivity of deep learning workloads to hardware-level faults. Traditional software-centric reliability evaluation methods must be augmented by fault models derived from physical effects in programmable fabrics to ensure the deployment of truly resilient AI systems in space and high-radiation environments.

The knowledge, methodologies, results, and tools developed throughout this research will serve as a fundamental basis for future advancements in the field, supporting the creation of more resilient, efficient, and high-performance reconfigurable computing systems capable of operating reliably under radiation exposure.

Bibliography

[1] Ian Kuon, Russell Tessier, and Jonathan Rose. 2008.

[2] Roger Woods, John McAllister, Gaye Lightbody, and Ying Yi. *Current FPGA Technologies*, pages 94–115. 2017.

[3] Corrado De Sio and Luca Sterpone. Toward fault-tolerant applications on reconfigurable Systems-on-Chip. In *2024 IEEE International Test Conference (ITC)*, pages 197–206, 2024.

[4] J. L. V. Ramana Kumari, V. Kranthi Kumar, M. Abhignya, and P. Shiva Rama Krishna. Design and performance analysis of Configurable Logic Block (CLB) for FPGA using various circuit topologies. In *2024 3rd International Conference for Innovation in Technology (INOCON)*, pages 1–5, 2024.

[5] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Bala Jayadev, Jeff Cuppett, Abbas Morshed, Brian Gaide, and Ygal Arbel. Versal Network-on-Chip (NoC). In *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 13–17, 2019.

[6] AMD Adaptive Computing. *Zynq UltraScale+ Device Technical Reference Manual (UG1085)*. AMD Adaptive Computing, 2025. Revision 2.5, March 21, 2025.

[7] Intel Corporation. *Cyclone V Device Handbook, Volume 1: Device Interfaces and Integration (Document 683375)*. Intel Corporation, 2025. Available as PDF at Intel's Programmable website, accessed December 4, 2025.

[8] Nanoxplore. NG-ULTRA Configuration Guide. https://files.nanoxplore.com/f/5bb62031ec6a4bdeb00a/?dl=1, 2023.

[9] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5:305 – 316, 10 2005.

[10] Dionysios Tompros and Dionysios E Mouzakis. Space environment effects on equipment and structures—current and future technologies. *The Journal of Defense Modeling and Simulation*, 21(3):283–291, 2024.

[11] Ygor Quadros de Aguiar, Frédéric Wrobel, Jean-Luc Autran, and Rubén García Alía. *Radiation Environment and Their Effects on Electronics*, pages 1–28. Springer International Publishing, Cham, 2025.

[12] Yasuda Mitsuyoshi, Funada Tomoya, Sato Hisaya, and Kato Kyoichi. Minimizing the exposure to the eye lens of radiologic technologists assisting patients during chest x rays: A phantom study. *Radiation Protection Dosimetry*, 193(1):43–54, 03 2021.

[13] Paolo Rech. Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions. *IEEE Transactions on Nuclear Science*, 71(4):377–404, 2024.

[14] Daniel Alfonso Goncalves De Oliveira, Laercio Lima Pilla, Mauricio Hanzich, Vinicius Fratin, Fernando Fernandes, Caio Lunardi, José María Cela, Philippe Olivier Alexandre Navaux, Luigi Carro, and Paolo Rech. Radiation-induced error criticality in modern HPC parallel accelerators. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 577–588, 2017.

[15] Ygor Quadros de Aguiar, Frédéric Wrobel, Jean-Luc Autran, and Rubén García Alía. *Introduction to Single-Event Effects*, pages 29–47. Springer International Publishing, Cham, 2025.

[16] Marty Shaneyfelt, James Schwank, Paul Dodd, and James Felix. Total ionizing dose and single event effects hardness assurance qualification issues for microelectronics. *Nuclear Science, IEEE Transactions on*, 55:1926 – 1946, 09 2008.

[17] Vitor A. P. Aguiar, Saulo G. Alberton, and Matheus S. Pereira. Radiation-induced effects on semiconductor devices: A brief review on single-event effects, their dynamics, and reliability impacts. *Chips*, 4(1), 2025.

[18] Suge Yue, Xiaolin Zhang, Yuanfu Zhao, Lin Liu, and Hanning Wang. Modeling and simulation of single-event effect in CMOS circuit. *Journal of Semiconductors*, 36:111002, 11 2015.

[19] Zhikang Yang, Lin Wen, Yudong Li, Jie Feng, Dong Zhou, Bingkai Liu, Zitao Zhao, and Qi Guo. Heavy ion single event effects in CMOS image sensors: SET and SEU. *Electronics*, 12(13), 2023.

[20] ECSS Secretariat. Ecss-e-st-10-04c rev.1: Space environment. ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands, June 2020. 15 June 2020 edition.

[21] Leonard Rockett, Dinu Patel, Steven Danziger, Brian Cronquist, and Jih-Jong Wang. Radiation hardened FPGA technology for space applications. pages 1 – 7, 04 2007.

[22] Zhe Liu, Zukun Lu, Long Huang, Zhiwei Yao, Zhaojun Lu, and Jiliang Zhang. Recent advances on reliability of FPGAs in a radiation environment. *Microelectron. J.*, 148(C), June 2024.

[23] E. Ardelean et al. Radiation tolerant fpga design techniques for space applications. *Journal of Aerospace Engineering*, 33(3):123–145, 2020.

[24] Bernhard Schmidt, Daniel Ziener, Jürgen Teich, and Christian Zöllner. Optimizing scrubbing by netlist analysis for FPGA configuration bit classification and floorplanning. *Integration*, 59:98–108, 2017.

[25] Constantin Anton, Laurentiu Ionescu, Ion Tutanescu, Mazare Alin, and Gheorghe Serban. Implementation and analysis of an error detection and correction system on FPGA. *International Journal of Intelligent Computing Research*, 4:334–339, 09 2013.

[26] Melanie Berg. FPGA design strategies for the space radiation environment. Technical report, NASA Goddard Space Flight Center, 2006. Presented at the Single Events Symposium, 2006.

[27] S. Azimi, C. De Sio, A. Portaluri, D. Rizzieri, and L. Sterpone. A comparative radiation analysis of reconfigurable memory technologies: FinFET versus bulk CMOS. *Microelectronics Reliability*, 138:114733, 2022. 33rd European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[28] Jack E. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, 1959.

[29] L. Bozzoli, C. De Sio, L. Sterpone, and C. Bernardeschi. PyXEL: An integrated environment for the analysis of fault effects in SRAM-based FPGA routing. In *2018 International Symposium on Rapid System Prototyping (RSP)*, pages 70–75, 2018.

[30] Andrew G. Schmidt and Thomas Flatley. Radiation hardening by software techniques on FPGAs: Flight experiment evaluation and results. In *Proceedings of the 2017 IEEE Aerospace Conference*, pages 1–8. IEEE, 2017.

[31] Leonardo Passig Horstmann and Antônio Fröhlich. A fault injection framework for real-time multicore embedded systems. pages 1–8, 11 2020.

[32] Martin Hiller, Arshad Jhumka, and Neeraj Suri. Propane: an environment for examining the propagation of errors in software. *SIGSOFT Softw. Eng. Notes*, 27(4):81–85, July 2002.

[33] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Regis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model. In *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 1–6, 2018.

[34] Enfang Cui, Tianzheng Li, and Qian Wei. RISC-V instruction set architecture extensions: A survey. *IEEE Access*, 11:24696–24711, 2023.

[35] Yasin Yilmaz, Yavuz Selim Tozlu, and Berna Örs. Design and implementation of a 32-bit RISC-V core. In *2021 13th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 460–464, 2021.

[36] Ankit P. Navik, Sachin Kumar Tiwari, Vipin Anand, Brijesh Yadav, Jonglae Park, and Ha Jun Sung. RISC-V: Redefining the future of computing, architecture, innovations, and beyond. In *2025 8th International Conference on Electronics, Materials Engineering Nano-Technology (IEMENTech)*, pages 1–5, 2025.

[37] F. Abate, L. Sterpone, and M. Violante. A new mitigation approach for soft errors in embedded processors. In *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, pages 1–6, 2007.

[38] Wilfredo Torres-Pomales. *Software Fault Tolerance: A Tutorial*. NASA, 1983.

[39] Felipe G. H. Leite, Roberto B. B. Santos, Nilberto H. Medina, Vitor. A. P. Aguiar, Renato C. Giacomini, Nemitala Added, Fernando Aguirre, Eduardo L.A. Macchione, Fabian Vargas, and Marcilei A. G. da Silveira. Ionizing radiation effects on a COTS low-cost RISC microcontroller. In *2017 18th IEEE Latin American Test Symposium (LATS)*, pages 1–4, 2017.

[40] Mehran Amrbar, Farokh Irom, Steven M. Guertin, and Greg Allen. Heavy ion single event effects measurements of Xilinx Zynq-7000 FPGA. In *2015 IEEE Radiation Effects Data Workshop (REDW)*, pages 1–4, 2015.

[41] Weitao Yang, Xue-Cheng Du, Yong-Hong Li, Chaohui He, Gang Guo, Shu-Ting Shi, Li Cai, Sarah Azimi, Corrado De Sio, and Luca Sterpone. Single-event-effect propagation investigation on nanoscale system on chip by applying heavy-ion microbeam and event tree analysis. *Nuclear Science and Techniques*, 32, 10 2021.

[42] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D.I. August. Swift: software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*, pages 243–254, 2005.

[43] M. Rebaudengo, M.S. Reorda, and M. Violante. A new software-based technique for low-cost fault-tolerant application. In *Annual Reliability and Maintainability Symposium, 2003.*, pages 25–28, 2003.

[44] Cristiana Bolchini, Antonio Miele, Fabio Rebaudengo, Fabio Salice, Donatella Sciuto, Luca Sterpone, and Massimo Violante. Software and hardware techniques for SEU detection in IP processors. *J. Electronic Testing*, 24:35–44, 06 2008.

[45] Pasquale Davide Schiavone, Davide Rossi, Antonio Pullini, Alfio Di Mauro, Francesco Conti, and Luca Benini. Quentin: an ultra-low-power PULPissimo SoC in 22nm FDX. pages 1–3, 2018.

[46] Boyang Du, Luca Sterpone, Sarah Azimi, David Merodio Codinachs, Véronique Ferlet-Cavrois, Cesar Boatella Polo, Rubén García Alía, Maria Kastriotou, and Páblo Fernandez-Martínez. Ultrahigh energy heavy ion test beam on Xilinx Kintex-7 SRAM-based FPGA. *IEEE Transactions on Nuclear Science*, 66(7):1813–1819, 2019.

[47] Vasileios Vlagkoulis, Aitzan Sari, John Vrachnis, Georgios Antonopoulos, Nikolaos Segkos, Mihalis Psarakis, Antonios Tavoularis, Gianluca Furano, Cesar Boatella Polo, Christian Poivey, Veronique Ferlet-Cavrois, Maria Kastriotou, Pablo Fernandez Martinez, Ruben Garcia Alia, Kay-Obbe Voss, and Christoph Schuy. Single event effects characterization of the programmable logic of Xilinx Zynq-7000 FPGA using very/ultra high-energy heavy ions. *IEEE Transactions on Nuclear Science*, 68(1):36–45, 2021.

[48] Lucas Antunes Tambara, Alexey Akhmetov, Dmitriy V. Bobrovsky, and Fernanda Lima Kastensmidt. On the characterization of embedded memories of Zynq-7000 All Programmable SoC under single event upsets induced by heavy ions and protons. In *2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–4, 2015.

[49] C. De Sio, S. Azimi, and L. Sterpone. On the analysis of radiation-induced failures in the AXI interconnect module. *Microelectronics Reliability*, 114:113733, 2020. 31st European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, ESREF 2020.

[50] Fabio Benevenuti and Fernanda Lima Kastensmidt. Reliability evaluation on interfacing with AXI and AXI-S on Xilinx Zynq-7000 AP-SoC. In *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pages 1–6, 2018.

[51] Sarah Azimi, Eleonora Vacca, Corrado De Sio, Luca Sterpone, and Andrea Portaluri. Mitigating cross-domain SEU corruption in FPGA-based AI accelerators for space applications: Invited paper. In *Proceedings of the 22nd ACM International Conference on Computing Frontiers: Workshops and Special Sessions*, CF '25 Companion, page 193–198, New York, NY, USA, 2025. Association for Computing Machinery.

[52] AMD. Isolation Design Flow for Zynq UltraScale+ MPSoCs and UltraScale+ FPGAs. Application Note XAPP1335, AMD, May 2023. Release date: 2023-05-15.

[53] Arsalan Ali Malik, Anees Ullah, Ali Zahir, Affaq Qamar, Shadan Khan Khattak, and Pedro Reviriego. Isolation Design Flow effectiveness evaluation methodology for Zynq SoCs. *Electronics*, 9(5), 2020.

[54] L. Gantel, M.E.A Benkhelifa, F. Lemonnier, and F. Verdier. Module relocation in heterogeneous reconfigurable Systems-on-Chip using the Xilinx Isolation Design Flow. In *2012 International Conference on Reconfigurable Computing and FPGAs*, pages 1–6, 2012.

[55] Jens Rettkowski, Konstantin Friesen, and Diana Göhringer. RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2016.

[56] Khoa Pham, Edson Horta, Dirk Koch, Anuj Vaishnav, and Thomas Kuhn. IPRDF: An isolated partial reconfiguration design flow for Xilinx FPGAs. In *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 36–43, 2018.

[57] Michael Wirthlin. High-reliability FPGA-based systems: Space, high-energy physics, and beyond. *Proceedings of the IEEE*, 103(3):379–389, 2015.

[58] S.M. Parkes and P. Armbruster. Spacewire: a spacecraft onboard network for real-time communications. In *14th IEEE-NPSS Real Time Conference, 2005.*, pages 6–10, 2005.

[59] Ádria B. de Oliveira, Fabio Benevenuti, Luis A. C. Benites, Gennaro S. Rodrigues, Fernanda L. Kastensmidt, Nemitala Added, Vitor A.P. Aguiar, Nilberto H. Medina, Marcilei A.G. Silveira, and Cédric Debarge. Analyzing the influence of using reconfiguration memory scrubber and hardware redundancy in a radiation hardened FPGA under heavy ions. In *2018 18th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–4, 2018.

[60] Guy Mantelet, Michel Briet, Guy Rouxel, Said Hachad, Bernard Bancelin, and Dominique de saint Roman. ATMEL ATF280E rad hard SRAM based reprogrammable FPGA SEE test results. In *2009 European Conference on Radiation and Its Effects on Components and Systems*, pages 606–608, 2009.

[61] Paulo R. C. Villa, Roger C. Goerl, Fabian Vargas, Letícia B. Poehls, Nilberto H. Medina, Nemitala Added, Vitor A. P. de Aguiar, Eduardo L. A. Macchione, Fernando Aguirre, Marcilei A. G. da Silveira, and Eduardo A. Bezerra. Analysis of single-event upsets in a Microsemi ProAsic3E FPGA. In *2017 18th IEEE Latin American Test Symposium (LATS)*, pages 1–4, 2017.

[62] N. Battezzati, F. Decuzzi, M. Violante, and M. Briet. Application-oriented SEU sensitiveness analysis of Atmel rad-hard FPGAs. In *2009 15th IEEE International On-Line Testing Symposium*, pages 89–94, 2009.

[63] Pierre Maillard, Yanran P. Chen, Jason Vidmar, Nicholas Fraser, Giulio Gambardella, Minal Sawant, and Martin L. Voogel. Radiation-tolerant Deep learning Processor Unit (DPU)-based platform using Xilinx 20-nm Kintex UltraScale FPGA. *IEEE Transactions on Nuclear Science*, 70(4):714–721, 2023.

[64] K. O'Neill, A. Severance, and D. Nandi. Using the VectorBlox software development kit to create programmable AI/ML applications in radiation-tolerant RT PolarFire FPGAs. In *Proc. Eur. Workshop Board Data Process. (OBDP)*, pages 1–8, 2021.

[65] Vasileios Leon, George Lentaris, Dimitrios Soudris, Simon Vellas, and Mathieu Bernou. Towards employing FPGA and ASIP acceleration to enable onboard AI/ML in space applications. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–4, 2022.

[66] NanoXplore SAS. From radiation hardening to BRAVE FPGA devices. In *RADiation and reliability challenges for electronics used in Space, Avionics, on the Ground and at Accelerators (RADSAGA)*, 2017.

[67] G. Tsiligiannis, C. Debarge, J. Le Mauff, A. Masi, and S. Danzeca. Reliability analysis of a 65nm rad-hard SRAM-based FPGA for CERN applications. In *2019 19th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–8, 2019.

[68] Vasileios Leon, Ioannis Stamoulias, George Lentaris, Dimitrios Soudris, David Gonzalez-Arjona, Ruben Domingo, David Merodio Codinachs, and Isabelle Conway. Development and testing on the European space-grade BRAVE FPGAs: Evaluation of NG-Large using high-performance DSP benchmarks. *IEEE Access*, 9:131877–131892, 2021.

[69] Sarah Azimi, Boyang Du, Luca Sterpone, D. Merodio Codinachs, and L. Cattaneo. SETA: A CAD tool for Single Event Transient analysis and mitigation on flash-based FPGAs. In *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–52, 2018.

[70] Morten B. Petersen, Stefan Nikolić, and Mirjana Stojilović. Netcracker: A peek into the routing architecture of xilinx 7-series fpgas. FPGA '21, page 11–22, New York, NY, USA, 2021. Association for Computing Machinery.

[71] Steve Guccione, Delon Levi, and Prasanna Sundararajan. JBits: Java based interface for reconfigurable computing. 01 2000.

[72] Khoa Dang Pham, Edson Horta, and Dirk Koch. BITMAN: A tool and API for FPGA bitstream manipulations. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 894–897, 2017.

[73] Travis Haroldsen, Brent Nelson, and Brad Hutchings. Rapidsmith 2: A framework for BEL-level CAD exploration on Xilinx FPGAs. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, page 66–69, New York, NY, USA, 2015. Association for Computing Machinery.

[74] Chris Lavin and Alireza Kaviani. Rapidwright: Enabling custom crafted implementations for FPGAs. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 133–140, 2018.

[75] Tao Zhang, Jian Wang, Shize Guo, and Zhe Chen. A comprehensive FPGA reverse engineering tool-chain: From bitstream to rtl code. *IEEE Access*, 7:38379–38389, 2019.

[76] Florian Benz, André Seffrin, and Sorin A. Huss. Bil: A tool-chain for bitstream reverse-engineering. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 735–738, 2012.

[77] F. Mo, A. Tabbara, and R.K. Brayton. A force-directed maze router. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pages 404–407, 2001.

[78] L. McMurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for FPGAs. In *Third International ACM Symposium on Field-Programmable Gate Arrays*, pages 111–117, 1995.

[79] Minghua Shen and Guojie Luo. Accelerate FPGA routing with parallel recursive partitioning. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 118–125, 2015.

[80] P.K. Chan and M.D.F. Schlag. Acceleration of an FPGA router. In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*, pages 175–181, 1997.

[81] Marcel Gort and Jason H. Anderson. Deterministic multi-core parallel routing for FPGAs. In *2010 International Conference on Field-Programmable Technology*, pages 78–86, 2010.

[82] NanoXplore SAS. *NanoXplore Wiki*, 2024.

[83] Vigneshwaran Palanisamy and Senthooran Vijayanathan. Cluster-based multi agent system for Breadth First Search. In *2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 54–58, 2020.

[84] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. In *2011 International Conference on Parallel Processing*, pages 216–225, 2011.

[85] S. Davidson. ITC'99 benchmark circuits - preliminary results. In *International Test Conference 1999. Proceedings (IEEE Cat. No.99CH37034)*, pages 1125–1125, 1999.

[86] Hoyeong Lee and Pilsung Kang. Performance evaluation of modern gpu accelerator-based edge systems: A holistic approach. *IEEE Internet of Things Journal*, 12(23):51716–51729, 2025.

[87] Stephan Wong, Thijs As, and Geoffrey Brown. r-VEX: A reconfigurable and extensible softcore VLIW processor. pages 369 – 372, 01 2009.

[88] Joseph A. Fisher, Paolo Faraboschi, and Cliff Young. VLIW processors: once blue sky, now commonplace. *IEEE Solid-State Circuits Magazine*, 1(2):10–17, 2009.

[89] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.