Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (35th cycle)

Toward Fault-Tolerant Applications on Reconfigurable Systems-on-Chip

By

Corrado De Sio

******

Supervisor(s):
Prof. Luca Sterpone, Supervisor

Politecnico di Torino
2023

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.

Corrado De Sio
2023

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

A Mamma e Papà,
who have been a light to me in dark places

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, who has 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 Maria for standing by my side throughout this journey. Without her, I could not have completed this journey.

Abstract

Thanks to their performance, reduced power consumption, and adaptability, Programmable Hardware devices, particularly Reconfigurable Systems-on-Chip, have emerged as a cutting-edge platform for many performance-oriented applications, including embedded ones. However, additional efforts are needed to ensure the correct system functionality for applications where reliability is the main concern. In particular, space exploration requires highly reliable systems that can operate in extreme conditions and environments such as the space radiation environment. Indeed, electronic devices, and especially programmable hardware, are sensitive to radiation-induced faults and errors, necessitating fault tolerance in critical applications.

This dissertation proposes and explores methodologies and techniques to enable accurate fault analysis and reliability evaluation for Hardware-Reconfigurable Systems-on-Chip, with a particular focus on safety-critical systems. It addresses the challenges in analyzing radiation sensitivity in complex systems and applications, such as the need for efficient fault detection and diagnosis strategies and the development of dedicated tools and methodologies.

Methodologies, evaluation flow, and tools for analysis and characterization of radiation-induced faults, such as Single Event Transients and Single Event Upsets, are proposed. They include both physical and electrical simulation approaches and radiation test analysis.

Techniques and methodologies for assessing the reliability of heterogeneous systems-on-chips, which includes both processor systems and accelerators based on reconfigurable hardware paradigms, are also presented. Research efforts cover the sensitivity of different modules against fault models resulting from physical analysis, electrical analysis, and radiation test experiments. Evaluated modules include soft and hard processors, host-device interfacing systems, and neural network accelerators.

The approaches, methodologies, and results presented in this dissertation aim to enable the development of highly reliable and fault-tolerant systems in a wide range of applications, particularly those requiring operation in extreme environments, such as space exploration. Additionally, the proposed methodologies and techniques can be used to analyze and evaluate the reliability of Reconfigurable Systems-on-Chip. This research wants to provide the necessary means to develop and analyze the reliable and efficient operation of such systems in extreme conditions and open the door for new opportunities for the development of advanced, reliable, and efficient systems based on a comprehensive and detailed analysis of the elements of heterogeneous computational platforms including programmable hardware, providing the methodology as well practical tools for reaching this goal.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

ALU Arithmetic-Logic Unit.

ANN Artificial Neural Network.

ASIC Application Specific Integrated Circuit.

BNN Binarized Neural Network.

BRAM Block RAM.

CLB Configurable Logic Block.

CMOS Complementary Metal-Oxide Semiconductor).

CNN Convolutional Neural Network.

CORDIC COordinate Rotation DIgital Computer.

COTS Commercial Off-the-Shelf.

CRAM Configuration RAM.

DD Displacement Damage.

DNN Deep Neural Network.

DRPM Dynamically Reconfigurable Processing Module.

FF Flip-Flop.

FinFET Fin Field-Effect transistor.

FPGA Field-Programmable Gate Array.

GCR Galactic Cosmic Ray.

GSO Geosynchronous Orbit.

HEO High Earth Orbit.

IC Integrated Circuit.

IOB Input-Output Block.

LEO Low Earth Orbit.

LUT Look-up Table.

MBU Multiple Bit Upset.

MEO Medium Earth Orbit.

MPSoC Multiprocessor System-on-Chip.

MUX Multiplexer.

NN Neural Network.

OCM On-Chip Memory.

OS Operating System.

PIP Programmable Interconnection Points.

PL Programmable Logic.

PREDA Placed-and-Routed Electronic Design Analyzer.

PS Processor System.

RAM Random Access Memory.

RTOS Real-Time Operating System.

SDC silent Data Corruption.

SEE Single Event Effect.

SEFI Single Event Functional Interrupt.

SEMU Single Event Multiple Upset.

SET Single Event Transient.

SEU Single Event Upset.

SoC System-on-Chip.

SRAM Static RAM.

TID Total Ionizing Dose.

TMR Triple Modular Redundancy.

UART Universal Asynchronous Receiver-Transmitter.

US UltraScale.

US+ UltraScale+.

Chapter 1

Introduction

1.1 Motivations

In the modern era, computer systems have become essential to our society. Starting with the transistor invention, they have quickly breached and influenced nearly any aspect of human life and modern culture. As a consequence, they evolved to target the specific needs and requirements of industries and applications. This trend led to the rise of a multitude of different computational systems based on different architectures and paradigms.

From the beginning of the computer era, the reliability of computer systems has been a major concern to ensure the correct functioning of the systems. However, starting in the 1960s with the NASA space program, the reliability requirements required for safety-critical applications responsible for human lives have pushed the need for reliability assurance to an even higher level.

Along with the need for more robust and reliable systems, we moved toward more high-performant systems that can fastly process large amounts of data, perform real-time operations, and deal with complex tasks. This has led to the development of more complex computer systems that require the use of multiple components working together.

The emergence of Hardware-Reconfigurable Systems-on-Chip has enabled the development of complex systems with high performance, low power consumption, and high flexibility. They soon found their way into many fields, such as space exploration, healthcare, automotive, and more.

However, the architecture of these systems makes them vulnerable and particularly sensitive to faults and errors. Additionally, the intrinsic complexity of these heterogeneous systems makes evaluating their robustness challenging. Nevertheless, fault tolerance and robustness is an important design goal for these systems that need to be evaluated and met, as it is mandatory for critical safety systems and applications.

This is especially true for space applications that require systems that can operate in extreme conditions and work properly even in a radiation environment for long periods of time, as well as equipment for radiation facilities and safety-critical systems such as systems used in the automotive and avionic industries. Hardware-Reconfigurable Systems-on-Chip provides a promising platform for these applications due to their ability to be reconfigured to meet the changing requirements, increasing the duration of the device and enabling in-field updates.

1.2 Aims and Objectives

The research work exposed in this thesis aims to identify and develop methodologies and techniques to evaluate and improve robustness and fault tolerance in Hardware-Reconfigurable Systems-on-Chip applications, with a major focus on space systems. It addresses the challenges associated with analyzing radiation sensitivity in Hardware-Reconfigurable Systems-on-chips applications. These challenges include the need for efficient fault detection, analyses, and diagnosis strategies and the development of dedicated tools and methodologies.

In addition, it aims to explore the potential of programmable hardware for building resilient applications and the potential for using Hardware-Reconfigurable Systems-on-Chip to develop fault-tolerant systems with high reliability and robustness. By exploring the potential of Hardware-Reconfigurable Systems-on-Chip for fault tolerance systems, the goal is to enable the development of reliable and robust systems that will benefit a wide range of applications.

1.3 Contributions

As main contributions, the research proposes and explores a range of tools and methods for analyzing, evaluating, and improving the reliability of Programmable Hardware-Reconfigurable devices, including fault injection methodologies, accelerated radiation testing, and emulation. The research covers different types of computational systems, fault models, and techniques, showing the potential of these tools and methodologies to analyze critical applications and modules in detail. Techniques dedicated to analyzing the sensitivity of systems to Single Event Transients and Single Event Upsets in programmable hardware are proposed. They are achieved by taking into account the devices' specific characteristics and susceptibility to radiation-induced faults by relying on radiation testing, electrical characterization, fault modeling, and novel methodologies.

The robustness of modules that build complex systems based on reconfigurable systems-on-chips, such as Hard and Soft processing systems, interconnection modules, and accelerators, with a particular focus on hardware-accelerated neural network applications, are addressed as well. The research shows the potential of reconfigurable hardware to be used to analyze applications and modules in detail, proposing platforms for analyzing the reliability of systems and exploring the real hardware device itself.

1.4 Thesis Organization

The thesis is structured as follows:

Part I is dedicated to introducing the context of the work presented in Parts II and III. Hardware-Reconfigurable Systems-on-chip are introduced in Chapter 2, while Chapter 3 is dedicated to Radiation Effects on Electronics devices.

Part II is dedicated to modeling and evaluating basic radiation-induced effects on Reconfigurable Systems-on-chip. In particular, Chapter 4 presents the analysis of Single Event Transient effects, followed by Chapter 5, which focuses on the analysis of Single Event Upset effects. Chapter 6 presents the results of two proton test experiments.

Part III of the thesis focuses on methodologies and techniques for evaluating the elements composing a heterogeneous system based on programmable hardware, considering the various elements composing the system, such as processors, hardware-accelerated unit, and their interfacing. Chapter 7 presents the evaluation of reliability for Embedded Processor Systems, considering both hard and soft processors. Chapter 8 focuses on the evaluation of reliability for Host-Accelerator Interfacing, while Chapter 9 then presents the evaluation of the reliability of Hardware-Accelerators for Neural Networks implemented in programmable hardware.

Finally, Chapter 10 concludes the thesis and provides insights into future research directions in Hardware-Reconfigurable Systems-on-chip, providing a summary of the findings and suggesting potential areas for further research in this field.

1.5 Publications

Part of the results presented in this thesis has been reported in the following publications:

Journal Papers

- **C. De Sio**, S. Azimi, L. Sterpone, and B. Du. "Analyzing radiation-induced transient errors on sram-based fpgas by propagation of broadening effect". *IEEE Access*, 7:140182–140189, 2019.
- **C. De Sio**, S. Azimi, L. Bozzoli, B. Du, and L. Sterpone. "Radiation-induced single event transient effects during the reconfiguration process of sram-based fpgas". *Microelectronics Reliability*, 100-101:113342, 2019.
- **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.
- S. Azimi, **C. De Sio**, D. Rizzieri, and L. Sterpone. "Analysis of single event effects on embedded processor". *Electronics*, 10(24), 2021.

- 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.

- **C. De Sio**, S. Azimi, and L. Sterpone. "Firenn: Neural networks reliability evaluation on hybrid platforms". *IEEE Transactions on Emerging Topics in Computing*, 10(2):549–563, 2022.

- S. Azimi, **C. De Sio**, A. Portaluri, D. Rizzieri, E. Vacca, L. Sterpone, and D. Merodio Codinachs. "Exploring the impact of soft errors on the reliability of real-time embedded operating systems". *Electronics*, 12(1), 2023.

Conference Papers

- 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.

- **C. De Sio**, S. Azimi, and L. Sterpone. "On the evaluation of the pipb effect within sram-based fpgas". In *2019 IEEE European Test Symposium (ETS)*, pages 1–2, 2019.

- B. Du, S. Azimi, **C. De Sio**, L. Bozzoli, and L. Sterpone. "On the reliability of convolutional neural network implementation on sram-based fpga". In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2019.

- **C. De Sio**, S. Azimi, and L. Sterpone. "On the evaluation of seu effects on axi interconnect within ap-socs". In A. Brinkmann, W. Karl, S. Lankes, S. Tomforde, T. Pionteck, and C. Trinitis, editors, *Architecture of Computing Systems – ARCS 2020*, pages 215–227, Cham, 2020. Springer International Publishing.

- **C. De Sio**, S. Azimi, and L. Sterpone. "An emulation platform for evaluating the reliability of deep neural networks". In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4, 2020.

- **C. De Sio**, S. Azimi, A. Portaluri, and L. Sterpone. "Seu evaluation of hardened-by-replication software in risc- v soft processor". In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2021.

- **C. De Sio**, S. Azimi, L. Sterpone, and D. Merodio Codinachs. "Analysis of proton-induced single event effect in the on-chip memory of embedded process". In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2022.

- A. Portaluri, S. Azimi, **C. De Sio**, D. Rizzieri, and L. Sterpone. "On the reliability of real-time operating system on embedded soft processor for space applications". In M., Carsten Trinitis, N. Papadopoulou, and T. Pionteck, editors, *Architecture of Computing Systems*, pages 181–193, Cham, 2022. Springer International Publishing.

Part I

Context

Chapter 2

Hardware-Reconfigurable Systems-on-Chip

2.1 Reconfigurable Hardware

In recent years, Field Programmable Gate Arrays (FPGAs) have become the leading solution among reconfigurable hardware systems. Reconfigurable Hardware is a computer architecture with the capability to be customized by end users or developers, differently from traditional hardware where functionality is defined during fabrication.

Thanks to this characteristic, reconfigurable hardware provide some valuable advantages compared to traditional fixed-a-function Application-Specific Integrated Circuit (ASIC) devices, such as low costs, flexibility, and fast time-to-market. ASICs still outperform reconfigurable systems on specific tasks, but the huge amount of resources that modern FPGAs can provide, along with the introduction of hardwired blocks (e.g., Digital Signal Processors, Artificial Intelligence engines, and microprocessors) are significantly reducing the performance gap. Additionally, the flexibility provided by reconfigurability ensures a smaller turnaround time, a longer lifetime, and in-field bug fixing and updates.

Additionally, in the past decade, hardware-reconfigurable Systems-on-chip (SoC) hit the market, making adaptive computing and heterogeneous computing a viable solution, thanks to commercial off-the-shelf solutions that reduced the costs and the required expertise by providing a higher level of abstraction and automated design process integrated with the development environments. As a consequence of that, these devices are nowadays widely adopted in many industries and sectors, such as space missions, avionics, automotive, server acceleration, and healthcare.

The traditional FPGA architecture consists of thousands of configurable logic tiles. These tiles can be configured to implement custom logic functions and then connected by using dedicated interconnection tiles to other logic blocks or the output pins of the system, virtually making it possible to implement any digital circuit in the fabric of the device. Nowadays, modern devices include also macro blocks directly inside the programmable fabric, such as Digital Signal Processors (DSPs), and Block Rams (BRAMs) that can be connected with the programmable elements.

Starting from a description of the circuit to implement on the programmable hardware, the development flow for the design to be implemented in programmable hardware includes several steps, such as synthesis, place and route, and bitstream generation. During synthesis, the high-level description of the design is translated to simple logic elements, such as logic gates and primitives, that are then associated with specific hardware resources in the FPGA fabric during place and route. During place and route also connection between these elements is performed. During the bitstream generation, a bitstream containing the configuration data for implementing the circuit on the target device is generated.

The configuration data of the device are stored within SRAM cells of the named configuration memory (CRAM) of the device. In modern FPGAs, the configuration memory can be either SRAM-based or Flash-based. Even if Flash-based FPGAs present advantages due to low power consumption and non-volatile configuration memory cells, SRAM-based solutions can take advantage of the last standard fabrication processes (FinFET and CMOS) that provide higher density and performance.

Configuration bitstreams are binary files that are used to program programmable hardware. In modern devices, a bitstream consists of hundreds of millions of bits containing metadata, configuration data, and programming instructions [1, 2]. The structure of the hardware is composed of multiple tiles, each of which has basic hardware resources that can be configured to execute a certain function. These resources include programmable interconnects that link two routing paths and Look-Up Tables (LUTs) that can implement custom logic functions. Each tile is associated

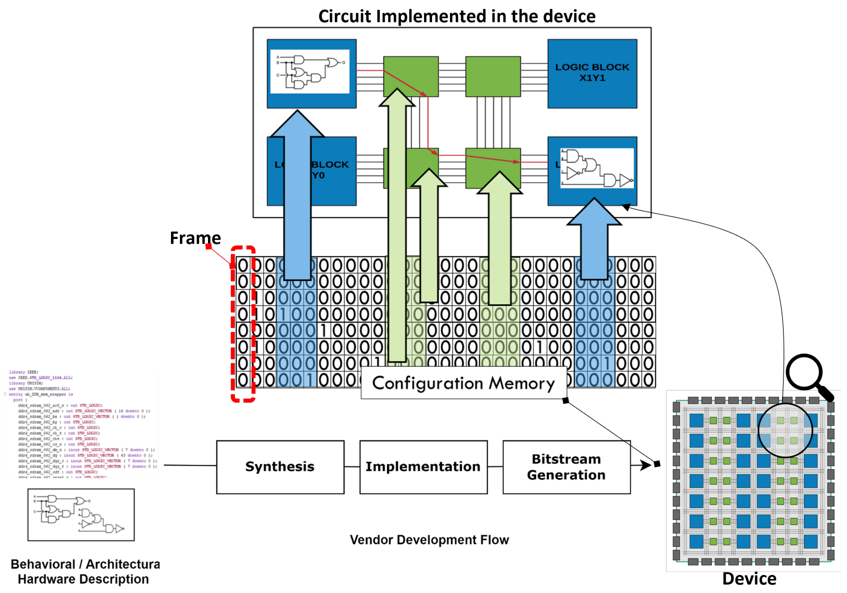


Figure 2.1 Conceptual schema representing the relation between configuration memory and hardware resources.

with certain bits in a configuration memory (CRAM), which are used to configure the components within the tile. The configuration memory is made up of frames, which are the smallest addressable group of bits. The number and the size of frames vary depending on the device size and architecture. While the bitstream file structure is partially revealed by manufacturers, the actual purpose of the specific bits in the configuration data and their connection to hardware resources usually remain largely unknown.

A conceptual representation of the development flow, as well as the relationship between configuration memory and configured hardware resources, is depicted in Figure 2.1.

2.1.1 Reconfigurable Architecture

Reconfigurable Hardware architectures have evolved continuously over the last years, as new technologies and products have been adopted and commercialized. The current section focuses mainly on more recent families of SRAM-based hardware-programmable devices. However, most of the concepts here reported applying to other vendors and families too, with few differences.

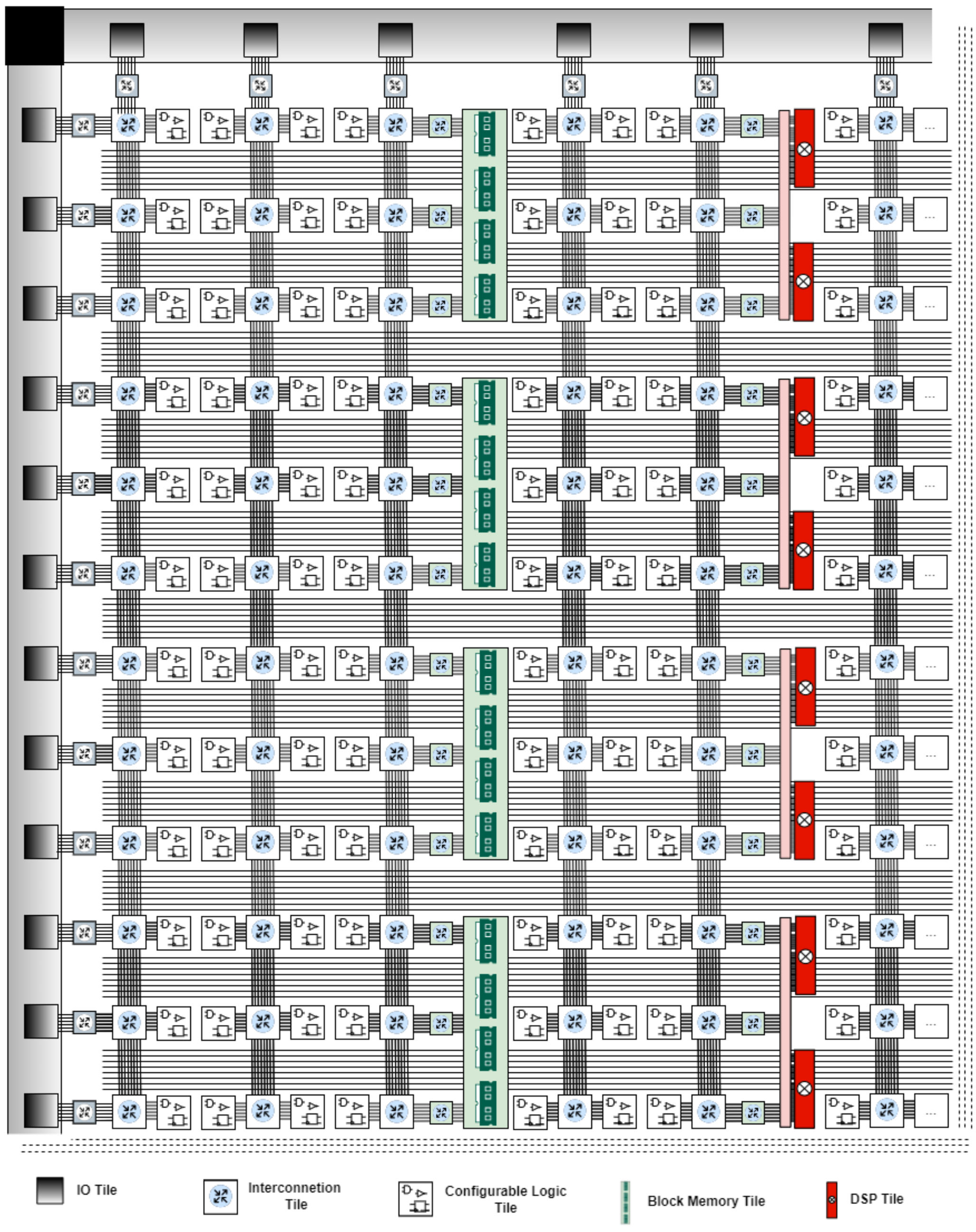


Figure 2.2 Schema of the architecture of an FPGA.

FPGAs are composed of a large number of logic blocks, configurable interconnect, and hardwired blocks. These blocks can be configured for implementing specific logical functions and connected through configurable interconnections. The configurable interconnects allow the logic blocks to be connected together in a variety of ways, allowing the user to create complex logic functions. The general architecture of an FPGA device is depicted in Figure 2.2

In general, reconfigurable hardware architecture consists of a 2-D array of tiles. Tiles are the basic elements of the FPGA architecture. An FPGA is made of different types of tiles, and different tiles types are dedicated to different purposes. Usually, tiles of the same type keep the same characteristic to one as the others within the same families. Each tile type is composed of different programmable elements based on its purpose. For instance, a logic tile includes configurable combinational and sequential logic elements, such as Flip-Flops, Look-Up Tables (LUTs), while interconnection tiles can be configured to route input signals to specific tile pins or to other interconnection tiles. Finally, some tiles are not configurable at all.

FPGAs are organized into a hierarchical structure. Tiles are grouped in Clock Regions which in turn are grouped in Super Logic Regions. Super Logic Regions are single device die slices stacked one on the other in Stacked Silicon Interconnect Devices. Tiles have their internal hierarchy too. A tile can contain zero, one, or more sites, which may contain Basic Elements. A Basic Element is the smallest configurable element in tiles. Within the device, tiles are arranged in columns and rows and tiles belonging to the same column are typical of the same type.

There are many different types of tiles in the FPGA fabric. Even if the internal organization of tiles of the same type is usually shared among devices of the same family, they differ among different families. Additionally, how tiles are arranged in the device differs among devices belonging to the same family too.

Logic Tiles

Logic tiles contain the configurable logic elements within FPGA. Each logic tile contains several programmable hardware resources, such as Flip-Flops, Look-Up Tables, and multiplexers. Flip-flops are used to implement sequential logic, while LUTs store a predefined list of outputs for every combination of inputs, to implement logical functions. Multiplexers are used to route signals internally to the tile to the outputs, using or bypassing flip-flops.

Figure 2.3 depicts the internal structure of the logic tile of a recent programmable hardware device. It includes eight 6-Input LUTs with 2 outputs, that support the possibility to be used to implement 2 either input-independent or input-shared logic functions. Two flip-flops are dedicated to each LUT. Additionally, a carry-chain is available for each logic tile as well as several multiplexers for routing internal signals to the logic tile output pins or flip-flops and carry-chain inputs.

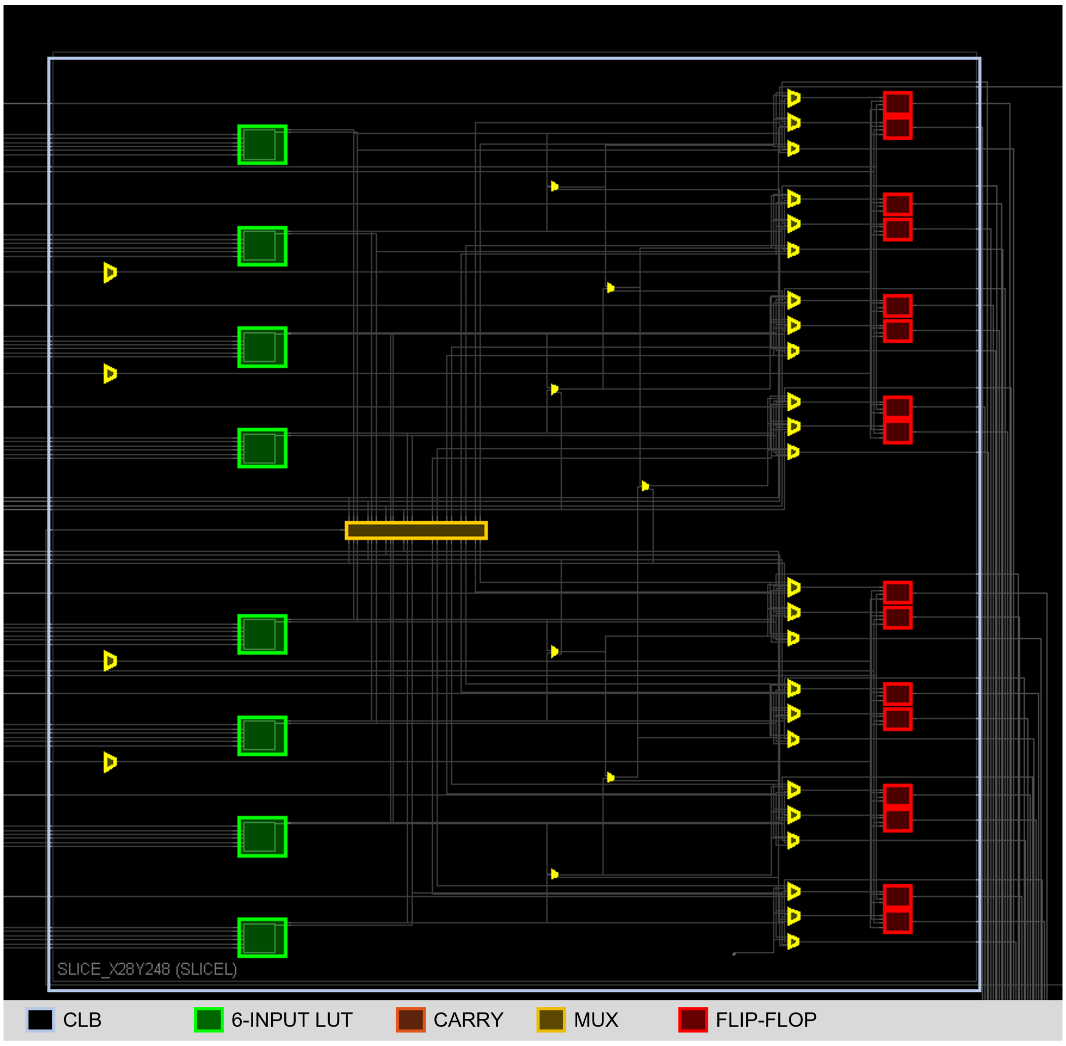


Figure 2.3 The internal structure of logic tile.

Interconnection Matrices

Commonly, each logic tile is associated with an interconnection matrix, referred also as a switch box or switch matrix. The interconnection matrices are used to route signals between different pins, either internals such as pins to connect the input of logic tiles, or external to interface the FPGA with the extern. The connections within switch matrices are typically constructed using pass-transistors or multiplexers. The state of the connections within the switch matrices is controlled using configuration bits, which determine active connections. These programmable interconnections connect two points called junctions or nodes. A junction can be associated with one or more input or output (or both) programmable interconnections. The switch matrices can be used to route signals between different components of a system too, such as between a processor and a memory module or a DSP. In this case, they are called interfacing matrices or interfacing boxes. Some junctions are connected with long lines that span over different tiles connecting switch matrices one with the other, forming the backbone of the device routing infrastructure.

In Figure 2.4 a representation of a switch matrix device is depicted, where the numerous programmable interconnections of the switch matrix, as well as the associated long lines, are highlighted.

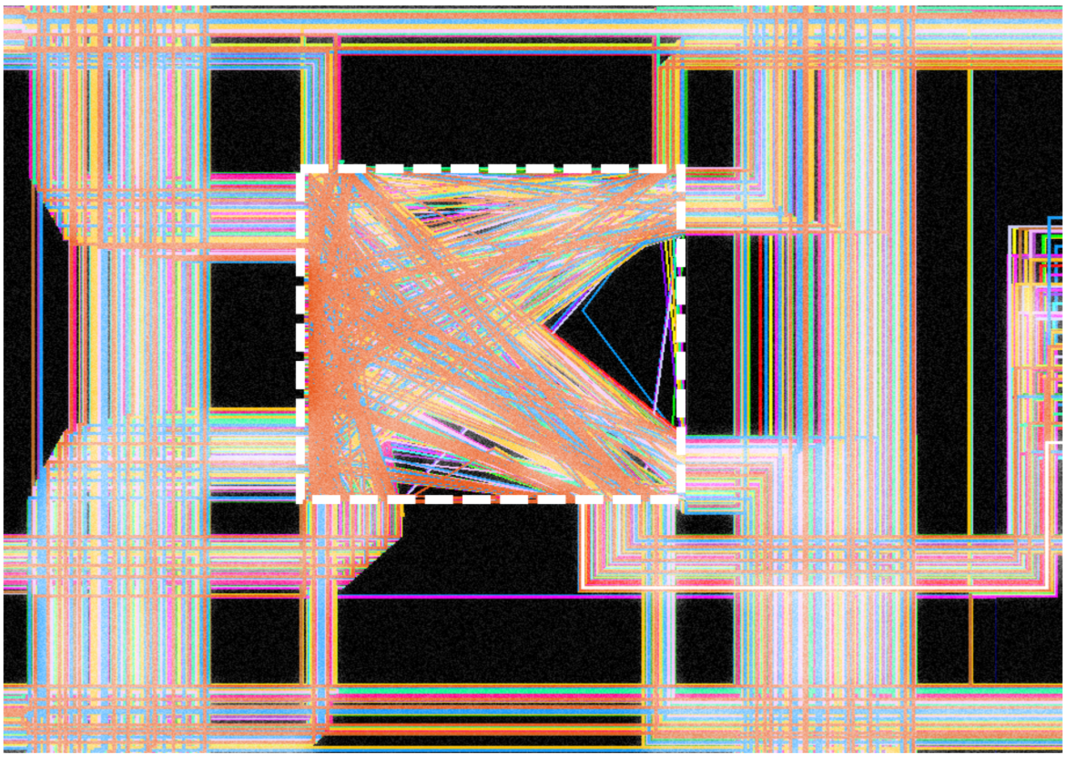


Figure 2.4 A switch matrix of a programmable hardware device with highlighted point-to-point interconnections and long lines.

Figure 2.5 shows the detail of a single junction in a switch matrix and its point-to-point programmable interconnection. The junction in the figure is characterized by both inner and outer connections.

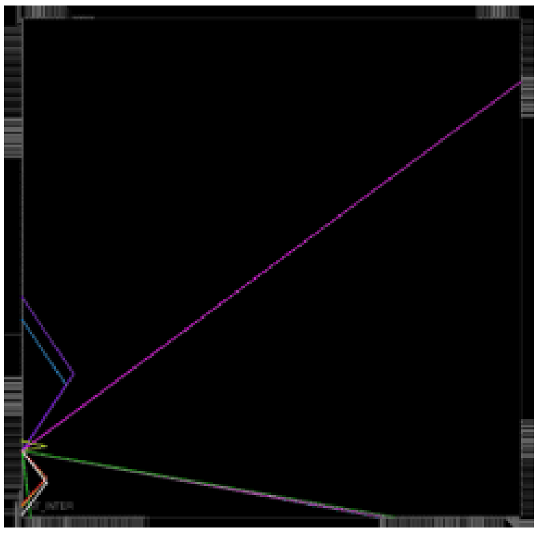


Figure 2.5 Overview of programmable segments associated with a junction in an interconnection box.

The programmbale interconnections are the most numerous programmable elements in an FPGA device and the vast majority of the cells of the CRAM are dedicated to programming routing. A single switch matrix of a modern device is characterized by more than 1,000 junctions and 4,000 PIPs, and a device can include more than hundreds of millions of configurable connections.

Input-Output Blocks (IOBs)

The IOB is a configurable block that is dedicated to interfacing the device with the extern allowing for the control of the device's I/O pins. They are programmable to be used either as input or output and can be programmed to meet the specific needs of the application, using different standards and voltages.

Hardwired Elements

Modern programmable hardware devices include hardwired elements directly in the fabric of the FPGA. They range from simple memories and DSPs to microprocessors and engines dedicated to parallel computation. These systems are usually not configurable. However, it is possible to route signals to their inputs in the same way as with any other element. While less recent families include only simple hardwired circuits such as DSP and BRAMs, more recent devices, include peripherals and multiprocessor systems, that are accessible from the programmable hardware.

2.1.2 All-Programmable Systems-on-Chip

A System-on-Chip (SoC) is an Integrated Circuit that implements a full system in a single chip. As usual for new technologies, SoCs have progressed from very simple architectures based on a limited number of modules, such as a small microcontroller, limited memory, and few peripherals to high-performance systems provided with multiple microprocessors, on-chip memories, accelerators, such as DSPs, GPUs and FPGAs, controllers, network-on-chip and more.

Hardware-Reconfigurable Systems-on-Chip devices are provided with an FPGA-like part integrated with processor systems and memory on the same chip, usually used as hardware accelerators and coprocessors.

The first Zynq device has been released in 2011 by Xilinx, Inc., currently, AMD-Xilinx, which already invented and commercialized the first cost-friendly FPGA in 1985. Zynq presented an innovative architecture based on a new/old paradigm. Indeed, while processor plus accelerator architecture was a well-known architecture, Zynq proposes a performant System-on-chip that provides processor and reconfiguration capability, creating the first device of its kind. Many applications took advantage of this kind of system during the last years.

Reconfigurable Systems-on-chip, and in particular Zynq, are heterogenous systems that allow to easily combine together hardware acceleration and software. A typical architecture on these platforms involves a hardware accelerator implemented into programmable hardware that is used by the processor system to perform the most demanding computations. The interface between the processor system and accelerators is usually implemented using high-performance port and communication protocols. This architecture paved the way for developers toward easy use of high-level synthesis approaches and hardware-software co-design. Nowadays, a board equipped with a system-on-chip embedding one of the most recent programmable hardware technology, general-purpose, and real-time multi-core processors on a unique chip is available for a few hundred dollars.

Chapter 3

Radiation Effects on Electronics

3.1 An Introduction to Radiation Effects

Radiation effects are a major concern when designing and deploying safety-critical systems and applications since they may cause malfunctions or destructive events in electronic devices. This is a major concern for safety critical systems that must remain operational and reliable in extreme conditions. Even if rad-hardened devices are designed to have higher robustness against radiation effects, they are usually characterized by reduced performance and high costs compared to commercial solutions. For this reason, modern COTS (Commercial Off-the-Shelf) components can provide significantly better performance compared to radiation-hardened solutions when used for space applications since they are typically more cost-effective than rad-hardened components. This makes them very appealing for many mission-critical applications. However, the sensitivity of these devices to radiation effects is a major obstacle to the adoption of these devices in safety-critical applications.

Radiation impacts the performance, accuracy, and durability of electronic systems [3]. The interaction of particles with the matter of the electronic device is the origin of many phenomena, including transient, permanent, or even destructive effects. The criticality and frequency of these effects are functions of many factors. The type, energy, and flux of interacting particles, which differ among radiation environments, as well as the device's materials, technology, and architecture, contribute to defining the severity, frequency, and typology of these events.

Due to this, radiation effects are a major concern for many electronic applications, particularly those that operate in radiation-heavy environments such as space missions or particle physic experiment monitoring. These systems must take into account the potential faults that radiation exposure can cause in the device. Even ground-level safety-critical systems cannot ignore the disturbances that terrestrial radiation can cause if they want to meet the high-reliability constraints required in applications such as autonomous vehicles.

Radiations can cause damage to electronic devices influencing the electrical properties of a component or system. This can include changes in the electrical characteristics, such as decreased performance, increased noise, and even complete failure. There are two major families of effects that radiation has on electronics: total dose effects and single event effects. The former is caused by modification of the atoms that compose the semiconductor that become ionized or displaced as a result of extended radiation exposure, while the latter is a transitory effect induced by a single radiation event.

3.1.1 Dose Effects

The dose effects are one of the main factors that limit the operational lifetime of electronics working in radiation environments. Permanent damages caused to electronics by radiation are caused by two different dose effects briefly summarized below.

Total Ionizing Dose is the sum of various physical interactions affecting the device exposed to radiation, which results in a degradation of the device's performance, such as threshold voltage modification leakage and others, that eventually lead to permanent failure. This phenomenon is a result of ionizing radiation.

Differently, non-ionizing radiation causes Displacement Damage. A particle with sufficient energy can displace an atom in the silicon lattice, leading to imperfections that degrade performance changing the electrical properties of the silicon.

3.1.2 Single Event Effects (SEEs)

SEEs include a wide range of effects that can be caused by the interaction between the matter of semiconductor devices and particles. These effects can be either destructive or nondestructive.

In general, SEEs are the result of the charge carriers generated in the silicon from an ionized particle traversing the device. These carriers are subjected to recombination or transport leading to a current, named Single Event Transient, that causes circuit response to this undesired signal.

Destructive SEE is usually caused by a low-impedance path triggered by the ion-generated charge that remains until the device is powered off or eventually permanently damaged. Single Event Lacthup and Single Event Burnout are examples of destructive SEEs.

Single Event Transient (SET)

A SET is a spurious signal that propagates in the same manner as a proper signal in the electronic circuit. A SET can propagate through different paths of the electronic circuit while being broadened or filtered by the logic. In a digital circuit, it will eventually reach a memory element, causing a Single Event Upset if latched or simply not causing any effect. The probability of SET being latched is dependent on many factors such as width, amplitude, and arriving time.

Single Event Upset (SEU)

An SEU is an error generated in a memory element. Even if the underlying mechanism is different for different memory cells, an SEU results in a change in the bi-stable element. It results in data corruption that can affect the system functioning in different ways based on the role of the affected memory cell. It is important to note that the functionality of the circuit is not damaged, so the memory element will continue to work properly after the event.

SEU affecting Programmable Hardware

SRAM memory cells have an essential role in reconfigurable systems. Other than being used as memory elements, that are fundamental in any modern sequential computing system, they are also the basis of reconfigurable hardware. SRAM-based memory can be found in modern hardware-reconfigurable systems-on-chip both as on-chip memory in the processor system or as BRAMs in the programmable logic, but more importantly as the basic block of the configuration memory.

However, SRAM memory is very sensitive to SEEs. This is due to the high density of transistors and their characteristic of being sequential elements, thus capable of capturing radiation-induced transient pulses. The fact that the physical layout for the embedded system is dominated by area and power constraints, leads to a minimum device geometry that exacerbates sensitivity to radiation due to the reduction of the critical charge.

Programmable hardware offers the possibility to implement custom hardware circuits in the fabric. Logic elements and interconnections are programmed by the content of SRAM cells of the configuration memory (CRAM). Each cell of the CRAM is dedicated to programming a specific resource in programmable hardware, thus the whole configuration memory defines the architecture of the hardware that is implemented on the programmable hardware.

Data corruption in the CRAM due to the radiation effect can interfere with how a resource is programmed leading to unexpected behavior and catastrophic failure. The fact that how the content of the CRAM is related to the resources composing the programmable hardware is usually undocumented exacerbates the problem making it hard to propose mitigation methodologies and approaches aware of the possible faults and the sensitivity of a specific circuit.

Additionally, this issue is worsened by the fact that the configuration memory of programmable hardware is often written at boot time and rarely rewritten. As a result, SEUs affecting the CRAM accumulate eventually causing multiple faults that can lead to system failure.

3.2 Radiation Environments

The type of radiation-induced effects affecting the electronic systems depends on the radiation environment to which the electronic device is exposed. In particular, electronic devices operating on Earth or in space are subjected to undesired effects that mine their reliability. However, the sources, effects, and frequency of these events can vary significantly among different environments where the system will operate.

3.2.1 Electronics operating in Space

The space environment is characterized by different sources of radiation. During space missions, electronics are exposed to different fluxes, energies, and types of particles. The characteristics of the orbit, especially Earth's distance, and orbit inclination, heavily impact the shielding effect introduced by Earth's magnetic field. Finally, solar activity is a main factor that determines the frequency of harmful radiation events that can affect electronics operating in the solar system.

As main sources of radiation in the solar systems, Galactic Cosmic Rays (GCR), Solar radiation, and radiation belts are the main cause of both SEEs and permanent effects affecting electronic systems.

Space Radiation Sources

Galactic Cosmic Rays are an isotropic flux of high-energy protons that originated outside the solar system. They mostly consist of protons and include also alpha particles and a few ratios of electrons and heavy ions (1%). GCR flux is mitigated by the shielding effect of the heliosphere that protects the planet from part of incoming GCR flux. About 75% of the particles are shielded by this phenomenon, which allows only particles with the highest energies to penetrate the heliosphere [3].

Solar activity has a strong influence on the solar system's radiation environment. It is characterized by an 11-year cycle, divided into four years of quiescence and seven years of activity. In the years of activity, the number of sunspots increases, influencing and increasing the frequency of solar events. Sun activity is the source of the solar wind, a flux of ionizing particles, composed of electrons, protons, and alpha particles. The energies of these particles range from 0.5 to 10 KeV. The impact of solar wind on electronic components operating in space is limited but has an important role in shielding the effects of Cosmic Rays. Differently, solar flare events and coronal mass ejections are more dangerous events, emitting bursts of radiation rays at very high energies that can seriously impair the functionality of electronic systems [3].

Finally, particles that have been confined in Earth's radiation belts, in particular, protons (about 10 MeV) and electrons (ranging from 1 to 5 MeV), interest altitudes between about 1,200km and 6,000 km (inner belts) and from 13,000 km to 60,000 km (outer belts). In these zones, the radiation exposure is much higher due to the high flux. Additionally, the asymmetry in the magnetic field of Earth results in the named South Atlantic Anomaly, which brings Van Allen's belt closer to the surface of the planet (up to 200 km). Radiation exposure in this area is much higher than elsewhere on earth at this altitude and becomes of significant importance for LEOs missions [3].

Operating Orbits

Operating orbit is another factor that strongly influences the radiation characterizing the environment of specific space applications. Low Earth Orbit (LEO), Medium Earth Orbit (MEO), Geosynchronous (GEO), and High Earth Orbit (HEO) are influenced in different ways by the shielding effect of the magnetic field of Earth and by particles trapped in Van Allen's belt.

Low Earth Orbit is the closest to the Earth, with an orbital period between one and two hours and an altitude lower than 2,000 km. The low altitude ensures several advantages such as faster communication compared to higher orbits. Due to its proximity to the land, LEO can benefit the most from the shielding effects of the magnetic field of Earth against radiation, since it is located under the belts that strongly shield cosmic rays.

Medium Earth Orbit, with an orbital period that can range from 2 to almost 24 hours, is located between 5,000 and 10,000km from Earth's surface. At these altitudes, the shielding effect of Earth is much weaker.

GSOs are adopted mainly by missions that need to minimize the disturbance introduced by Earth's magnetic field, but as a result, they are the most susceptible to radiation effects. They operate at a constant altitude of 35,786 km.

Finally, HEO and interplanetary or deep space missions are heavily exposed to a high flux of high-energy particles.

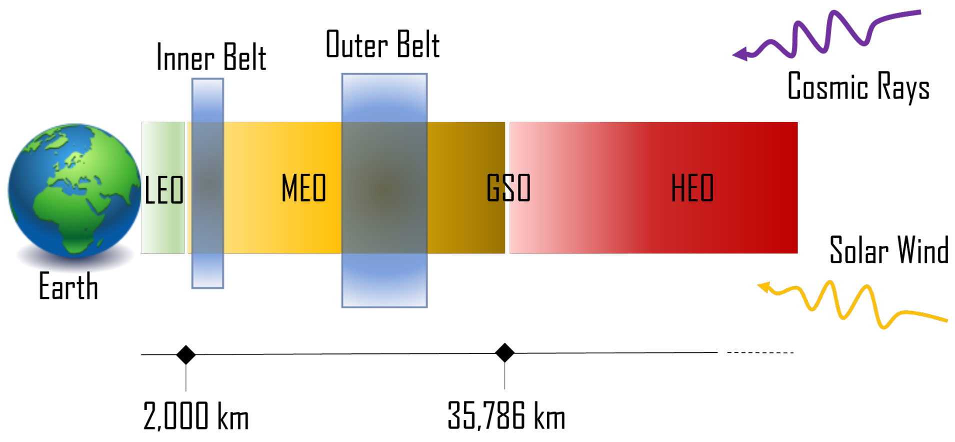


Figure 3.1 Conceptual schema of common Earth orbits for space applications, with the approximated location of the radiation belts

3.2.2 Electronics operating on Earth

On Earth (from sea level to about 22 km of altitude), SEEs are the main concern to electronic systems reliability, since particle flux is too low for causing significant damage due to dose accumulation. High-energy Cosmic-Ray neutron radiation and Low-Energy Cosmic-Ray neutrons are the main sources of radiation-induced faults in this environment together with alpha particles.

Earth Radiation Sources

Alpha particles are generated by the natural decay of radioactive impurities that may be present in the materials used in the production process of electronic devices. At ground level, they are the main source of SEE and they indeed have been the first hint of the existence of soft errors in the 1970s. Positively charged Alpha particles can create disturbances leading to spurious accumulation of charge carriers and so soft errors [3].

High-energy (>1 MeV) cosmic-ray neutron radiation is the cause of radiation showers, cascades of secondary particles resulting from high-energy protons of GCR that interact with the atmosphere. Usually, particles reaching Earth's surface are mostly neutrons and secondarily protons and pions. In this case, neutrons are the main source of soft errors. The flux of neutrons is dependent on three factors (in decreasing order of importance): Altitude, Longitude, and Solar Activity. The neutron flux is much higher at higher altitudes, so neutrons become the main concern for events in avionics since the SEE rate increase with flux and so altitude. Contribution to neutron flux due to latitude is related to the shielding effects of Earth's magnetic fields that are stronger near the equator. Shielding of high energy cosmic rays decreases the frequency of particle showers, so flux is higher near Earth's poles. Solar activity, and in the particular solar wind, provide additional shielding against cosmic rays, decreasing the probability that high energy protons reach the atmosphere generating shower events. So during the 7-year solar activity radiation flux at ground level is lower compared to Sun's inactivity period.

A significant difference when considering neutrons flux compared to proton flux is that neutrons are not charged particles and silicon ionization is the only result of nuclear reactions. Practically, for silicon, a single SEE is expected for every 1,000 to 10,000 neutrons. Due to the same reason, shielding from neutrons is not always viable, since it requires meters of concrete, which is often unfeasible for embedded and avionics applications.

Lastly, thermal neutrons can be captured by Boron-10 isotopes producing both Alpha and Lithium-7 particles that can be sources of SEEs. Since Boron-10 is commonly used in the fabrication of intermetal layers for semiconductors, it has contributed to increasing the SEEs rate[3].

Part II

Physical-Level Radiation Analysis of Reconfigurable SoCs

Chapter 4

Analysis of Single Event Transient

4.1 Overview on Single Event Transient Analysis

A Single Event Transient (SET) results from a particle, such as a proton or a heavy ion, traversing an active node of a circuit. It results in a transient voltage disturbance pulse that propagates in the circuit like normal signals. A SET is characterized by many features such as amplitude, duration, shape, and polarity. The characteristics of the produced SET are a function of many parameters related to the particle, the device, the angle and location of the incident path, and more.

A SET generated within a node of an electronic device will propagate the logic elements of the circuit. During the propagation in the circuit, the SET can be masked, attenuated, or broadened. As a result of the propagation, it can also propagate from the single node where it has been generated to several nodes through different paths. Eventually, it may reach a sampling element. If when it arrives in the proximity of a clock edge to the sampling element with enough amplitude and width, it can be sampled leading to data corruption or an erroneous system state. Due to its transient feature will produce a soft error in digital circuits only if sampled or if it is propagated until the system output.

The miniaturization of new devices, the lowering in operating voltages, and the increase in operating frequency made the issue more severe, increasing the probability of erroneous strong radiation-induced pulses being generated and eventually sampled [4].

While many works focused on SETs propagation and analysis in the application layer for Flash-based FPGA, works addressing this issue for SRAM-based devices are limited. Indeed, due to the fact that Flash-based FPGAs are intrinsically immune to SEUs in CRAM, SET represents one of the main contributions to error for Flash-based FPGAs, which led to a strong interest in their study. Differently, SEU in configuration memory is a significant concern in SRAM-based programmable hardware and is the most investigated soft error for these devices. However, the increase in operating frequency and the continuous transistor miniaturization that is characterizing modern SRAM-based programmable hardware devices are exacerbating SETs effects that cannot be furtherly ignored in these devices.

Additionally, the SET happening during the reconfiguration process for Flash-based FPGA represents a negligible source of errors. Indeed, Flash-based FPGAs do not provide partial or dynamic reconfiguration capability. Differently, SRAM-based programmable devices offer Dynamic Partial Reconfiguration features. This feature offers a number of advantages, such as a lower utilization of resources on the FPGA and a lower consumption of power. In addition, the DPR could be used in several areas, such as fault tolerance systems or systems for self-repair, thereby increasing the flexibility and reliability of the application. For instance, it is also widely exploited for mitigating the accumulation of errors in configuration memory, refreshing the configuration memory with the correct values. The impacts of SETs happening while writing the CRAM, such as during a partial reconfiguration, have not previously been studied in the literature.

4.2 SETs Propagation in the Programmable Logic

4.2.1 State of the Art of the Analysis of SET Propagation in Programmable Devices

Even though Single Event Transients are a fundamental mechanism of the interaction between radiation and microelectronics, analyzing them is a challenging task. In FPGA logic, when a SET introduces an observable error, it usually behaves like a SEUs in user flip-flops. Propagation-induced Pulse Broadening (PIPB) is a mechanism that causes a propagating SET to be broadened (or compressed) while traversing logic gates. For instance, a SET with an initial width of 200 ps can be broadened to a few nanoseconds when it propagates through several logic gates [5]. Since SET sampling is strongly dependent on the width and amplitude of the pulse when it reaches latching elements, it is of crucial importance to estimate SET sensitivity. A representation of this phenomenon is depicted in Figure 4.1.

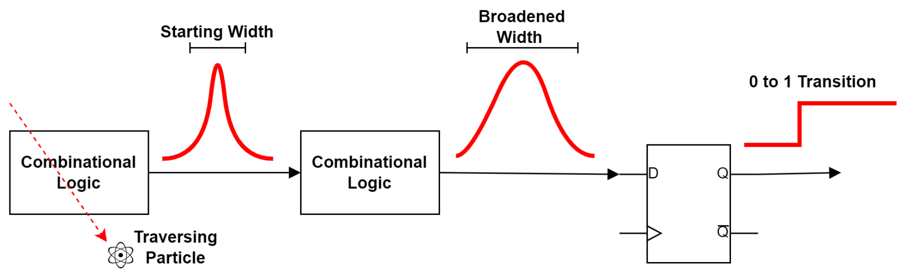


Figure 4.1 Conceptual representation of PIPB effect.

Many research works have been conducted to evaluate the sensitivity and characterize Single Event Transients [6–9]. Most of these studies focus on simulating the radiation-induced pulse and propagating it through the combinational logic without considering pulse broadening or attenuation, which depends on the specific technology. Radiation testing and electrical injection approaches have been proposed to produce more realistic results [10]. Although radiation testing is the most realistic method of testing, it is a demanding approach and requires specialized facilities. Device-external pulse injection and measurement are not recommended due to the high levels of distortion they introduce. Device-internal fault injection is a better option as it provides good control of the pulse characteristics and measurement accuracy. However, it requires modifying the implemented circuit for inserting the fault injection module and it can require high development efforts.

In [11], and further in [12], we proposed a flow for statistically assessing the sensitivity of a netlist to Single Event Transients by evaluating propagation and broadening of transients in the combinational and sequential logic of the device. The analysis of a group of benchmark circuits has been conducted to assess their susceptibility to Single Event Transients (SETs). The results of this evaluation were then compared to the outcomes of fault injection electrical experiments that had been previously conducted on a physical device [13].

4.2.2 A Static Analyzer for PIPB effect

The proposed methodology is based on an architectural model of function generators (i.e., LUT) used in the target SRAM-based FPGA family. The architectural model has been analyzed using 3D physical simulation to provide a realistic characterization of induced Single Event Transients, in terms of amplitude, widths, and source node. Additionally, the electrical characterization of the function generator model has been evaluated using electrical simulations. The characteristics of the Single Event Transient population generated by the physical analysis have been used together with the Function generator electrical model for instrumenting the propagation analyzer.

The main advantage of the proposed methodology is to be generally fast and simple to apply and integrate into the development flow compared to other validation and error detection techniques. Even if the detection of flaws through static analysis is usually not comprehensive due to its intrinsic static nature, it still allows for solving a significant number of issues during the early phases of the development flow in a relatively short time. SET and logic function generator characterization can be performed independently on the netlist under test, making the approach easily extendable to other devices. For this reason, it can be performed once per device or technology, significantly speeding up the evaluation process by reducing iteration time.

4.2.3 Modeling Function Generators and SETs Characteristics

An FPGA is characterized by programmable function generators that can be configured and connected together, as well as combined with sequential elements, for implementing practically any logic function. Implementation details of these elements, such as the physical layout, are usually not publicly available for Xilinx devices. However, a model has been proposed using open libraries resembling the functional description provided by Xilinx. Nevertheless, when it comes to open-source programmable hardware architectures, the availability of detailed information on layout and technology can make analysis even more realistic and trustworthy.

The Function Generator model proposed is depicted in Figure 4.2. We proposed a model for Function Generators based on 6-input LUTs architecture which is adopted in Xilinx Series 7 devices, as it is reported in [1] and [14]. The proposed architecture consists of two 5-inputs LUTs with shared inputs (A1-A5). The outputs of the two LUT are the inputs of a multiplexer controlled by a sixth input (A6). The multiplexer forwards one of the two LUT outputs to the function generator output (O6) accordingly with the selector signal (A6). The secondary output of the Function Generator (O5) is driven by the output of the first LUT (with the output connected to the 0 port of the multiplexer).

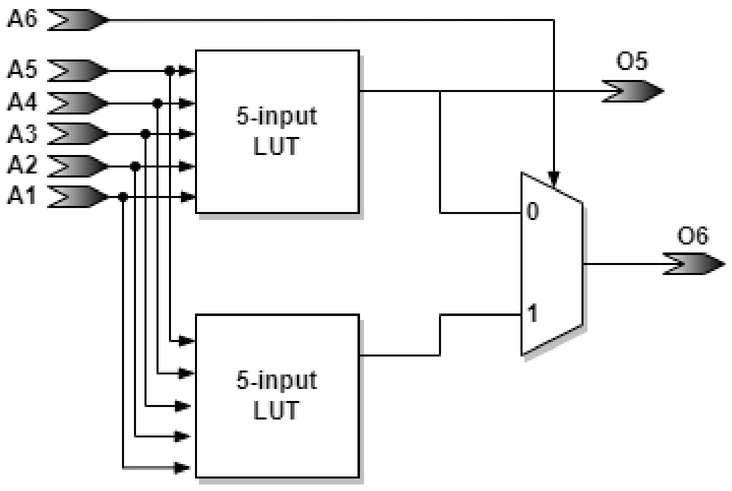


Figure 4.2 Conceptual schema of proposed Function Generator Model

We assumed the 5-input LUTs composing the function Generator to be implemented by 5 stages of the multiplexer, as represented in Figure 4.3. Functions are implemented by writing the truth table of the LUTs in the configuration memory cells associated with the Function Generator instance.

It has been shown in [13] and [15], that LUTs have different effects on the SET pulses, based on specific logic function implemented, output line considered, and pulse characteristics. In order to obtain a detailed characterization of the Function Generator's electrical behavior to be integrated into the propagation analysis algorithm, we performed an electrical simulation of the model.

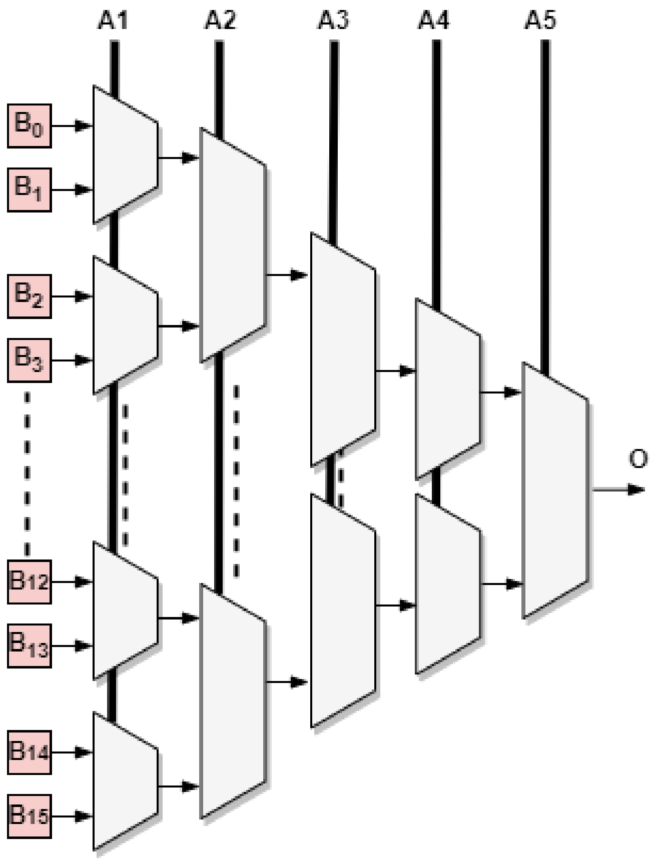


Figure 4.3 Conceptual schema of proposed LUT architecture used in the Function Generator Model

In order to perform physical and electrical simulations of the Function Generator, a physical layout for the 5-input LUT is needed. Therefore, a physical implementation for the LUT has been implemented using an open-source gate library with the same technology as the target device (i.e. 28nm CMOS), using the following cell types: 2-input MUX, input buffer, and 2SRAM cell. The generated layout is illustrated in Figure 4.4, and information about total cells and area are reported in Table 4.1

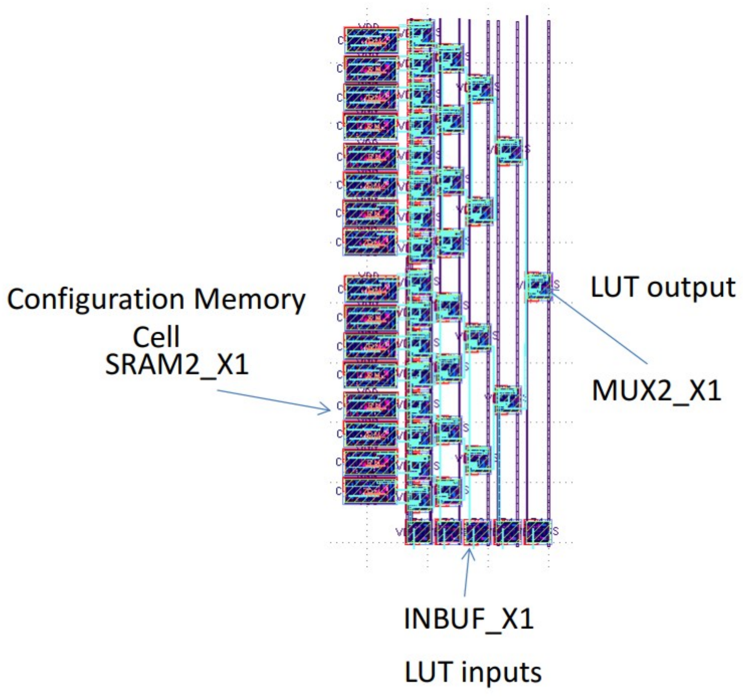


Figure 4.4 GDS layout of 28 nm CMOS LUT.

Table 4.1 Cells information for proposed LUT Physical Layout.

Cell	Number [#]	Area [μm^2]
SRAM2 X1	16	29.97
MUX2 X1	31	79.68
INBUF X1	5	4.17

The proposed layout geometry has been then used both in the physical simulation aiming to predict generated SET characteristics and in the electrical simulation to obtain a realistic characterization induced by function generators.

Single Event Transient and PIPB effect Characterization

The characteristics, such as amplitude, width, and generating node, of the SETs that can affect the device are crucial for evaluating circuit sensitivity. Indeed, they strongly affect both the propagation-induced effects on the pulse, such as compression or broadening, and as a consequence the probability of the pulse being sampled becoming a soft error.

For this reason, the analysis includes a Single Event Transient characterization. It is based on RadRay, the physical simulation tool presented in [16]. It takes into account the parameters of a specific radiation environment and considers how radiation particles interact with the device under test, simulating particles traversing the device. In the simulation, parameters such as incident angles, LET, and type of particle are considered for predicting SET pulse characteristics, such as amplitude, widths, and node when it has been generated. The tool provides a report of generated SETs characteristics. The purpose of this analysis is to provide a comprehensive characterization of the Single Event Transient pulses that can be generated in the technology.

The proposed layout geometry for 5-input LUT, including cells, their routing, and the routing between cells and VCC and GND rails, has been provided to the tool for 3D physical simulation in order to obtain a realistic SET characterization to be used during static analysis.

The analysis involved six heavy ions commonly used in radiation testing: Carbon, Neon, Aluminium, Chromium, Nickel, and Xenon. 1000 particles have been evaluated for each element, generating an equal number of SETs. Weakest SETs (< 0.4 V) have been filtered based on the technology sensitivity of the target device while the remaining pulses have been categorized by duration. Distribution is reported in Figure 4.5

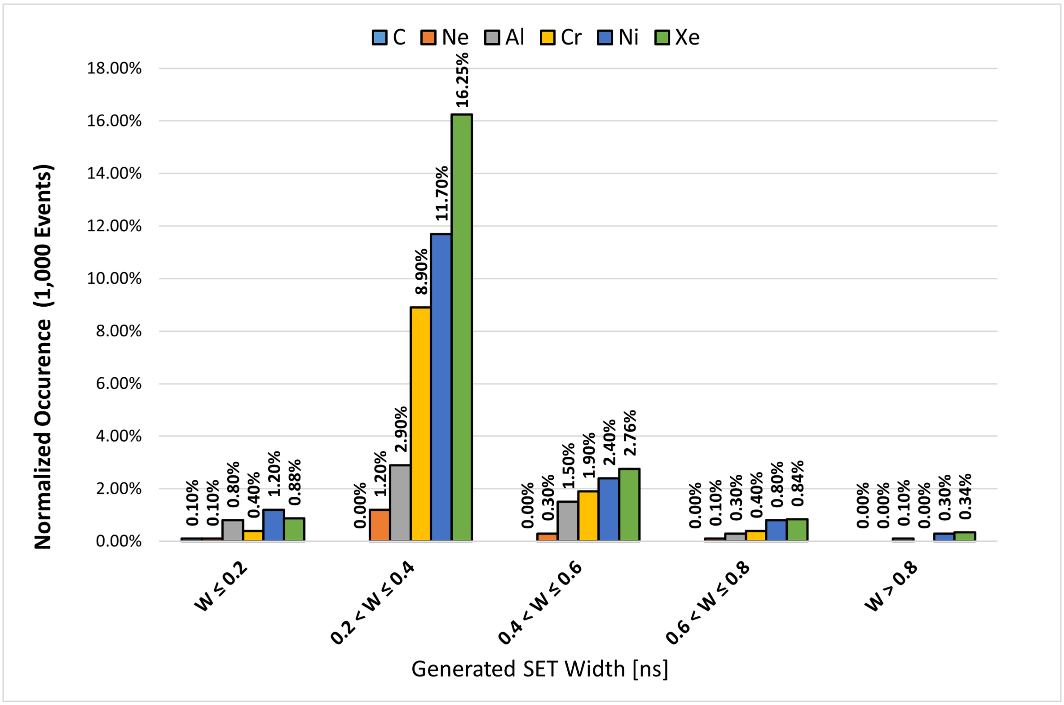


Figure 4.5 SET Characterization resulting from physical simulation of 1,000 particles for each heavy ion.

Most of the critical SETs (i.e., with amplitude higher than 0.4 V) are in the range of 200 ps and 600 ps. For this reason, we performed an electrical simulation for evaluating PIPB effect introduced by the Function Generator for these groups of SETs. Characterization has been conducted for different logic functions, considering pulse width, amplitude, and generating node. Electrical simulation behavior has been further polished using experimental data obtained through the method reported in [13].

4.2.4 Analyzing PIPB on Placed-and-Routed Netlists

The PREDA framework has been developed to model the logical and physical architectures of a routed design. PREDA is coded in Python and interfaces with Vivado, and Xilinx commercial design suite, through the Tcl language, and can be extended to other programs that provide post-layout design information.

PREDA can build a graph structure from the post-implemented design, exploited later in propagation static analysis. The graph includes characteristics of the logic cells such as primitive type and location. Additionally, PREDA can analyze the graph structure to build a subgraph describing the logic cone of a flip-flop in the circuit. It is further used for evaluating the SET propagation since when SETs will reach a sampling element they will be filtered or sampled becoming an error, anyway stopping its propagation. Thus SETs impacting a memory element can be evaluated by analyzing the logic bordered by the sequential element itself and another memory element.

To study the impact of LUTs on SETs propagation within the circuit, we developed a SET propagation analyzer, APES (Analyzer of PIPB effect on SETs) that has been integrated with PREDA. With APES, it is possible to evaluate how many SETs, injected with specific locations and widths, can be broadened when they propagate through distinct LUTs along a specific design route. The method aggregates the various contributions made by the LUTs while it propagates along the path after considering the width of the SET and the physical pins of the LUTs via which they propagate.

APES can evaluate what and how much memory elements are impacted by SETs happening in a specific logic node, as well as what is the worst broadening effects that may affect SETs that reach a memory element.

The worst PIPB affecting a memory element is evaluated by computing as SETs propagate through the levels of logic gates in the logic cone. The worst PIPB affecting each logic cell is calculated in an iteration if PIPB affecting its parent cells has been computed. The process is repeated until the worst PIPB effect on the flip-flop input is identified. A conceptual schema of the algorithm is represented in Figure 4.6.

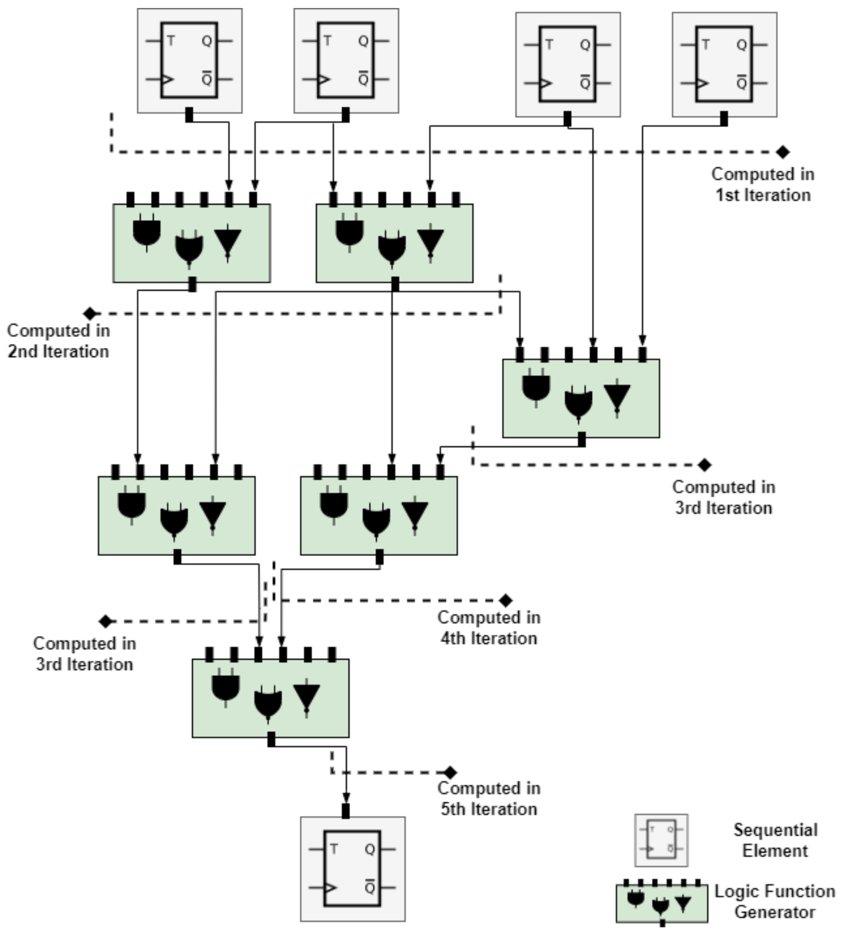

T Q
Q̄
Computed in 1st Iteration
Computed in 2nd Iteration
Computed in 3rd Iteration
Computed in 4th Iteration
Computed in 3rd Iteration
Computed in 5th Iteration
Sequential Element
Logic Function Generator

Figure 4.6 Conceptual schema of SET propagation procedure.

4.2.5 Validating PIPB Analysis on Benchmark Circuits

For validating the proposed approach, four benchmarks have been selected from the ITC'99 benchmark collection [17]. Selected circuits were B05, B12, B14, and B15. The overview of the post-layout characteristics of the four benchmark circuits implemented on a Xilinx Kintex7 is reported in Table 4.2.

Table 4.2 Implementation Details for the Benchmark Designs.

Circuits	Flip-Flop Cells [#]	Logic Cells [#]	Max. Logic Cone Depth [levels]	Avg. Logic Cone Depth [levels]
B05	34	91	6	3.2
B12	119	251	6	4.1
B14	215	1,071	14	11.1
B15	416	1,390	14	9.2

The proposed evaluation workflow resumed in Figure i4.7 has then been applied to the four benchmarks for evaluating SET sensitivity. SET and Function Generator characterization, already presented in the previous subsections, have been provided to the framework.

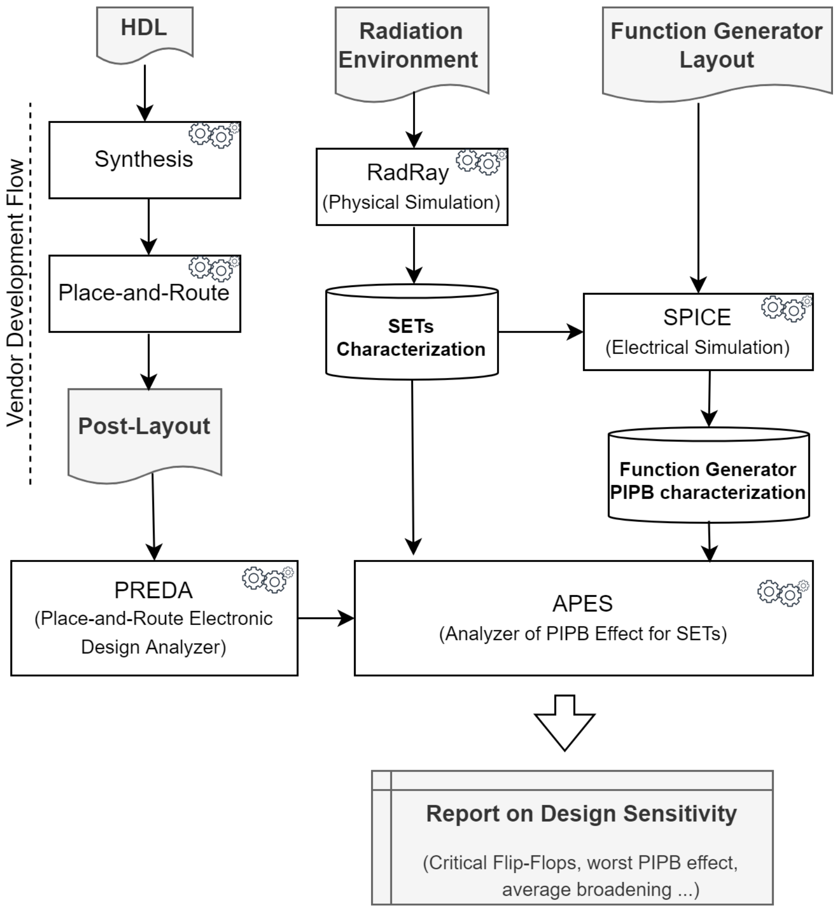

HDL
Radiation Environment
Function Generator Layout
Vendor Development Flow
Synthesis
RadRay (Physical Simulation)
Place-and-Route
SETs Characterization
SPICE (Electrical Simulation)
Post-Layout
Function Generator PIPB characterization
PREDA (Place-and-Route Electronic Design Analyzer)
APES (Analyzer of PIPB Effect for SETs)
Report on Design Sensitivity
(Critical Flip-Flops, worst PIPB effect, average broadening ...)

Figure 4.7 Workflow for evaluating PIPB effect.

The B14 and B15 circuits are characterized by higher complexity and longer combinational logic paths. The results obtained by the proposed static analysis method have been compared with the electrical fault injection approach applied to B12 and B15 designs [13]. The comparison is reported in Table 4.3, resulting in coherent results.

The framework allowed us to statically evaluate prediction for all the Flip-flops within benchmark circuits. The resulting categorization of the flip-flops of the benchmark circuit is reported in Figure 4.8

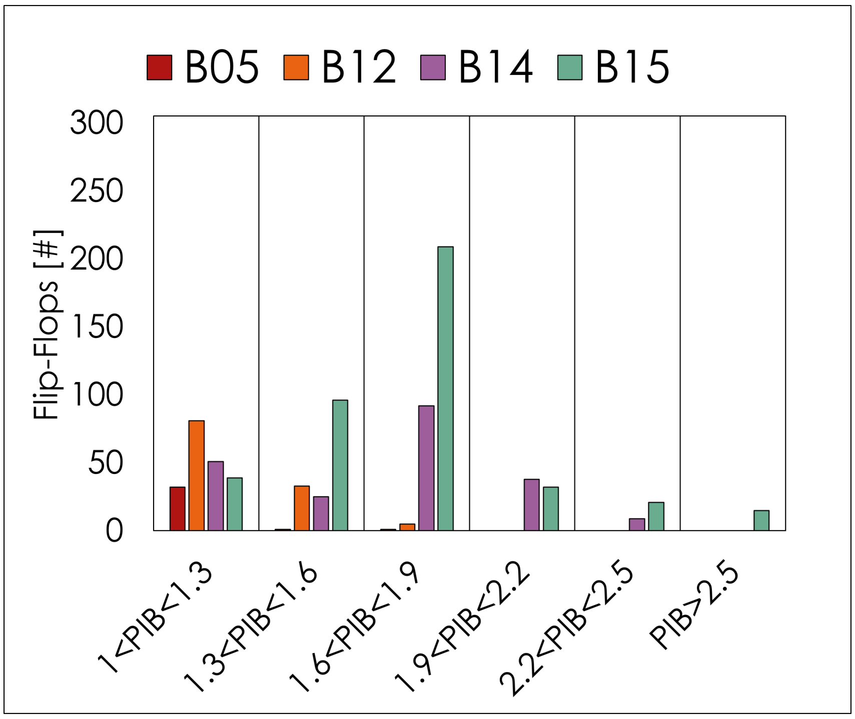


Figure 4.8 Sensitivity of the flip-flops in the analyzed benchmark designs.

Even if the prediction could result in a sensitivity assessment slightly less accurate than the one provided by internal fault injection, it should be considered that, after the characterization which is required only once per device, the framework is able to analyze the benchmark circuits in minutes. Differently, the development of an internal fault injection infrastructure usually requires significant efforts for implementing a generator and meter for each flip-flop in the design.

Table 4.3 Max PIPB effect predicted

Circuit	Electrical Fault Injection	APES Analysis
B05	-	1.60
B12	2	1.86
B14	-	2.24
B15	3	3.12

A fast categorization of sensitivity for flip-flops that can be integrated directly into the design flow offers a valuable resource for identifying flip-flops more impacted by the long SET, allowing the designer to apply fine-grained mitigation techniques to the most sensitive elements.

4.3 Effects of SETs during FPGA Reconfiguration

4.3.1 State-of-the-Art on the Evaluation of SETs during Hardware Reconfiguration

Partial Reconfiguration is a technique for improving the performance and utilization of SRAM-based FPGA. The basic idea is to use the same hardware resources to deploy two mutually exclusive functions or circuits. During the application development flow, the designer can select part of the fabric to be partially reconfigurable. As a result, a partial bitstream can be downloaded in the configuration memory section associated with that resources, changing the implemented circuit. This operation can happen while the remaining part of the device works normally.

Applying partial reconfiguration, it is possible to improve power consumption and device utilization. Partial reconfiguration can also be exploited for refreshing specifically part of the configuration memory where an error has been detected, both through configuration memory content inspection or application layer detection, improving the reliability and robustness of the application.

Even if Single Event Transients investigation usually focuses on Flash-based FPGA systems and the FPGA user logic, the reconfiguration task of SRAM-based FPGA has the characteristics to be very sensitive to SET. Indeed, Reconfiguration is based on SRAM cells written at high frequency during configuration, making spurious pulses easy to be sampled. If FPGA programming happens in a controlled environment, it is reasonable to consider errors induced by SET during reconfiguration negligible. Anyway, when these systems are used in a radiation environment the number of events and their impact on the application should be investigated.

However, no methodology or analysis is proposed to address or evaluate this issue. For this reason, we proposed in [18] an evaluation flow for assessing SET's impact when SRAM-based FPGAs are reconfigured. A Monte Carlo analysis is used to assess the SET characteristics produced by heavy ions. The analysis predicts parameters such as the amplitude and length of the pulses in relation to the layout characteristics. The SET pulses are then used in an electrical analysis simulation for generating fault models that are then emulated on the actual hardware during reconfiguration to determine the Dynamic Error Rate.

4.3.2 Modeling Reconfiguration Circuitry

Resources on the FPGAs are programmed by the content of the CRAM. The CRAM defines how these resources are used and connected. To load the bitstream into the configuration memory, the control logic makes use of a serpentine shift register where data transit before being stored in the configuration memory [19]. An overview of the architecture involved in downloading configuration data in configuration memory is reported in Figure 4.9

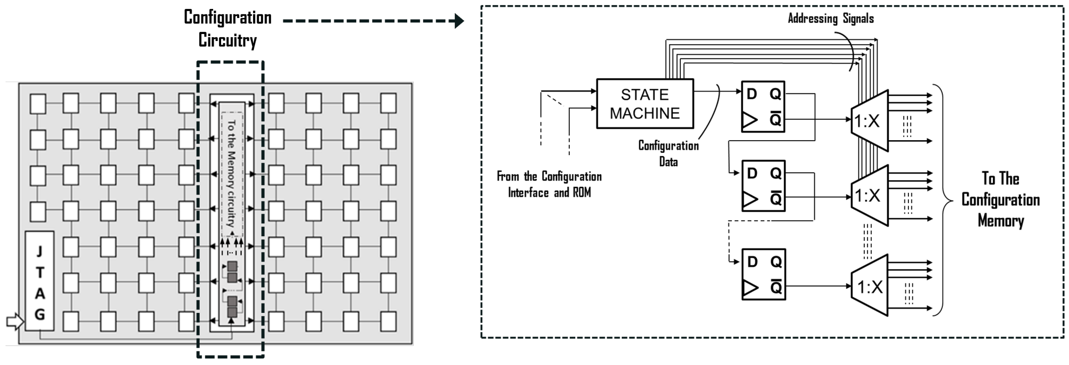


Figure 4.9 An example of Configuration Memory control circuitry.

In order to propose a realistic SET characterization, a model for the serpentine shift register has been proposed based on the technology used in Xilinx Series 7 devices, since no details are provided on his characteristics from the vendor, except for what is reported in [19].

The shift register is a circuit very sensitive to SET for its intrinsic characteristics. Indeed, it is characterized by a high density of sampling elements and high frequency. The proposed physical layout has been reported as GDS-II data, and a visual representation of a part of it, in particular three flip-flops chained together, are reported in Figure 4.10. For developing the model, an open-source technology library has been used [20] resized to 28 nm. A spice model has also been proposed to evaluate the electrical effects of the transient pulses on the circuit and evaluate the fault models for the configuration memory.

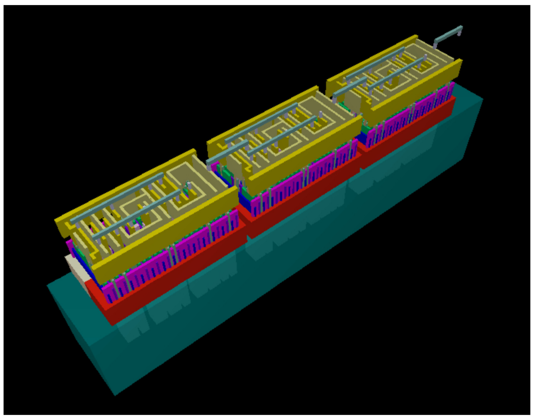


Figure 4.10 A representation, based on GDS-II description, of three 28 nm flip-flops of the shift-register.

4.3.3 Fault Model: From Transient Pulse to Soft Error

To comprehensively model the faults that may affect the content of the frames composing the CRAM due to SETs affecting the shift register, a Monte Carlo radiation particle analysis has been performed based on the RadRay tool [16]. The tools simulate the radiation particles that traverse the electronic component generating the transient pulses. The approach is based on the proposed physical level description of the serpentine shift register. The resulting cross-section is reported in Figure 4.11

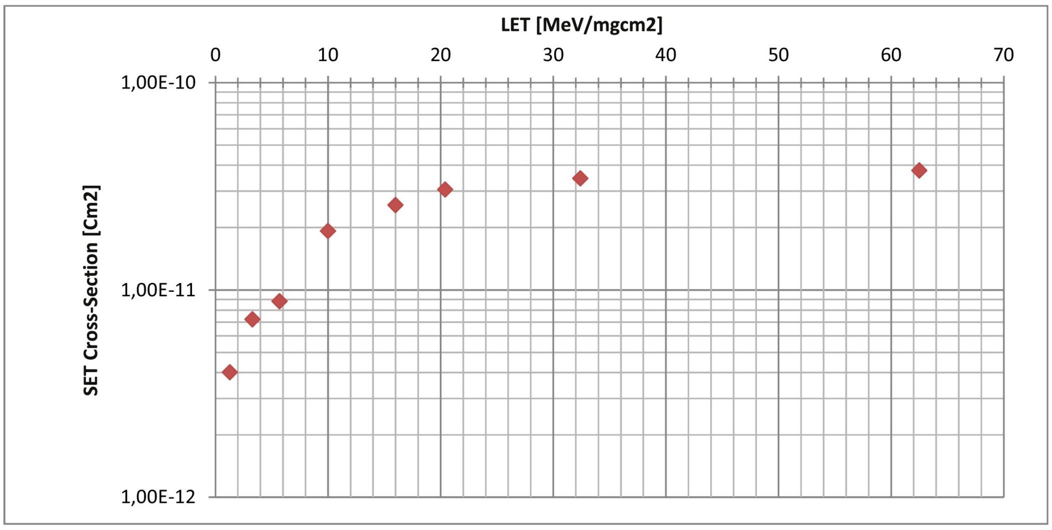


Figure 4.11 SET cross-section of a 28 nm Flip-Flop of the shift-register

The electrical SPICE model of the serpentine shift register has been used for electrical simulation. The SETs generated by the physical analysis are injected during electrical simulation in the shift register model at a random time while the configuration data pass through the circuit. Then the results of the simulation have been analyzed to extract the fault model.

An example of the electrical simulation output, where a 600 ps SET is injected and sampled is illustrated in Figure 4.12

Three different behavior have been observed:

- **Single**: A bit in the frame has been modified. The result is an SEU in a frame of the CRAM.
- **Burst**: A sequence of bits is modified. The result is an MBU affecting up to eight sequential bits of a frame.
- **Multiple**: Multiple bits in the frame are affected, but they are interleaved with unaffected bits.

4.3.4 Evaluating Errors due to SETs during Reconfiguration

The evaluation flow has been applied to a benchmark design. The analysis is based on a SET analysis for the proposed model of the shift register. The effects of

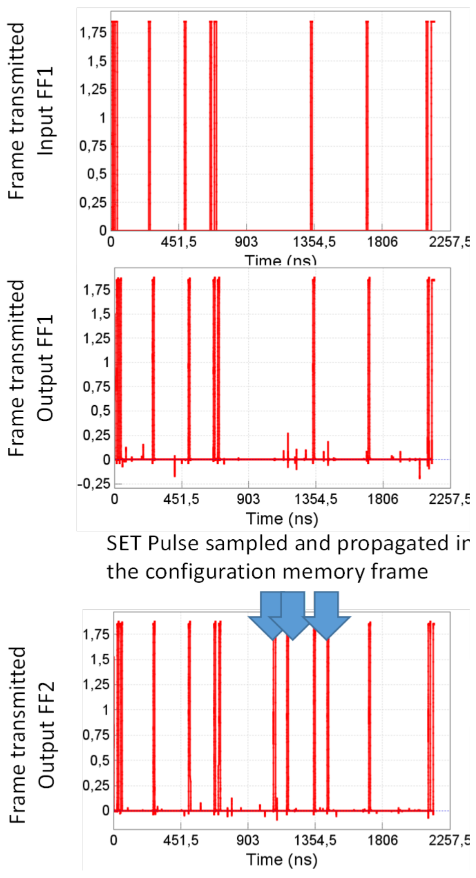


Figure 4.12 SET sampling as reported by the electrical simulation tool.

SET observed have been used as fault models. Faults have been emulated in the configuration memory content through manipulation of the configuration data. Since fault occurs during reconfiguration, the injection of fault is performed at configuration time.

The device under test is a Zynq 7020 SoC. The purpose of the analysis is to assess the dynamic error rate caused by faults that arise during reconfiguration in the given design, and evaluate if SET-induced faults can lead to a significant error rate during the reconfiguration process.

A DRPM has been selected as the design under test. The design comprises a processor system, ARM Cortex-A9, and four reconfigurable modules, consisting of a couple of CORDIC IP Cores and of a couple of BRAMs controllers. An AXI Infrastructure connects the modules to the processor system, which communicates with an experiment manager via a serial connection with a host computer. A test program running on the processor system stimulates the core in programmable logic. A schematized view of the platform is illustrated in Figure 4.13

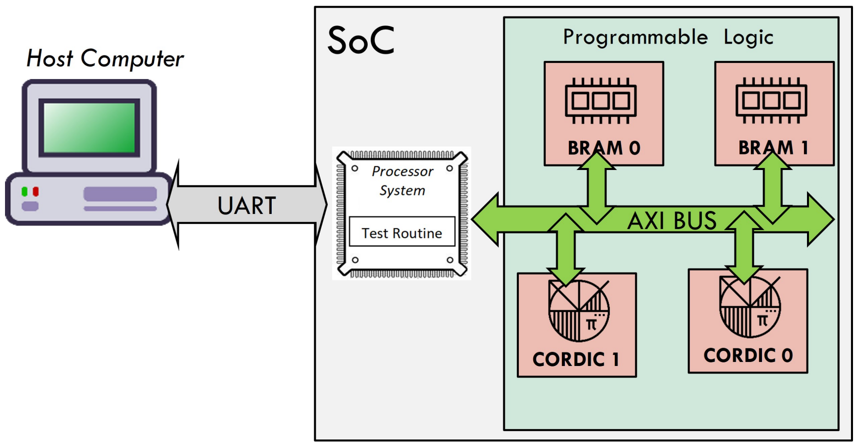


Figure 4.13 DRPM System Architecture

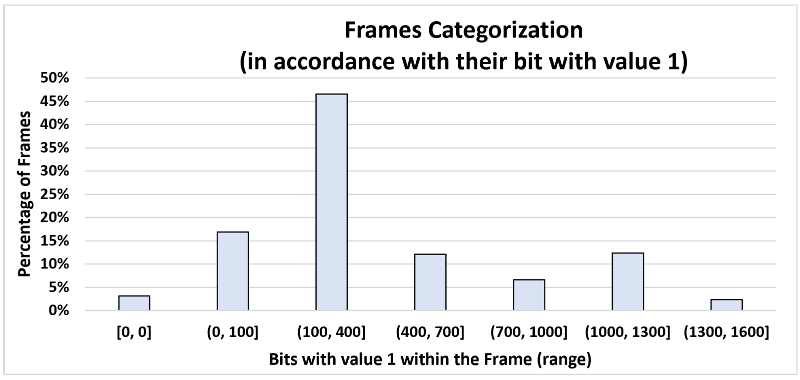
The software routine on the processor evaluates the functionality of the modules of the DRPM with test routines. BRAMs are evaluated via MATS tests, while CORDIC modules compute trigonometric operations. The MATS test implements a sequence of memory read/write operations to detect any malfunctions [21]. The test results are provided to the experimental manager platform through the UART, which checks whether the MATS tests and CORDIC operations were successful or not.

PyXEL [22] has been used as a fault injection and analysis platform for emulating fault models in the configuration memory of the device. A first fault injection has been carried out based on a single-bit fault model. During electrical simulation, Single bit fault model was the most observed fault model among the three.

The injection space has been reduced using PyXEL and is limited to the used parts of the device. A first evaluation campaign has been carried out with 10,000 fault injection experiments. This campaign resulted in a dynamic error rate of 3.86%. Furthermore, the frames with the highest error have been selected to be exhaustively tested, yielding an error rate of up to 15%.

Additional investigation has been carried out considering the characteristics of specific data in frames. Frames have been categorized accordingly with the number of bits having logic value 1. Even if configuration data of used resources are not only encoded using 1 bit, since some data such as the truth table of the LUTs can use also 0 values, it is still a reasonable indication of the use of a frame.

We performed a second fault injection campaign, in which we injected a fixed number of faults into each subset of frames. This enabled us to accurately evaluate the dynamic error rate associated with frames with a specific number of one-bits. As the number of one-bits in a frame increases, the number of faults introduced by random injections decreases, as faults injected on bits with a value of '1' is masked. Indeed, we observed during electrical simulation that transitions from 0 to 1 were much more common. However, the number of one-bits in a frame indicates that more resources have been programmed, which increases the probability that faults in that frame will lead to errors in the output. The dynamic error rate, as well as frame distribution, for each selected category, is reported in Figure 4.14



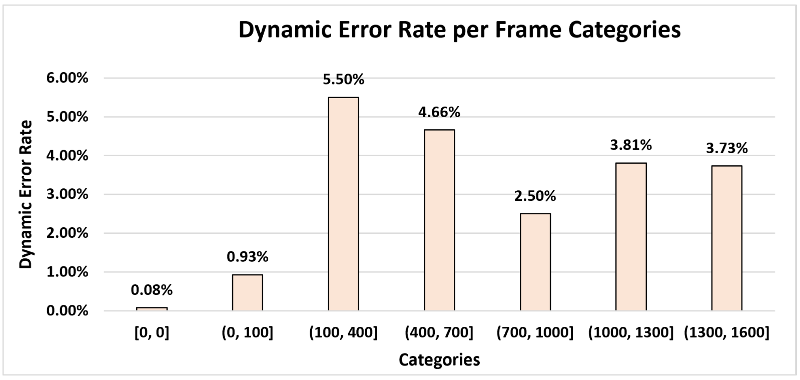


Figure 4.14 Dynamic Error Rate for different categories of frames, frames are divided into categories accordingly to the number of bits of value 1.

The error rate obtained for the design under test is not negligible, particularly when considering that only a single-bit fault model has been tested, while other fault models could be also more critical since they involve more bits in the frame. The dynamic error rate of reconfiguration tasks involving critical frames (i.e., the one with a high utilization) reaches approximately 15% when a single-bit fault model occurs. Critical applications that want to exploit DPRM features, as well as scrubbing techniques. should take into account that SETs during reconfiguration processes can have an impact on reliability.

4.4 Research Advancements on the Analysis of Single Event Transients

We proposed and developed new flows for evaluating the sensitivity of the design to Single Event Transients propagating in the combinational logic and in the configuration circuitry of programmable devices based on the characterization of SET generation by physical simulation and of SET propagation through electrical simulation.

This approach has been adopted in the development of a static flow for the analysis of SETs propagation in the netlist of placed-and-routed FPGA designs. The flow based on the characterization of SETs and their propagation in the function generators of FPGA devices provided results comparable to the ones obtained in radiation test experiments. The PREDA and APES tools have been developed to support the proposed methodology by extracting place-and-route information and performing propagation analysis. The approach represents a valuable methodology for performing preliminary reliability analysis of FPGA design that can ease the evaluation of robustness during development and before accelerated radiation testing.

An analysis of the contribution of SETs during the configuration memory reconfiguration has been proposed for the first time. The analysis based on the electrical model of a configuration circuitry has been carried out through fault injection, emulating the fault model obtained by physical SET generation and electrical simulation.

Chapter 5

Analysis of Single Event Upset

5.1 Overview on Single Event Upset Analysis

An SEU is a transient error that can occur in sequential logic when a particle, such as a proton or a neutron, passes through the device. While the underlying mechanism involves many physical phenomena, it can be modeled as a bit to be flipped from a 0 to a 1 or vice versa, resulting in a logic error at a higher abstraction level. SEUs can be caused by radiation from cosmic rays, nuclear reactors, particle accelerators, and other sources. SEUs can significantly impact an electronic system's operation, as they can cause incorrect operation of the device or even cause the device to fail.

Nowadays, no electronic system is totally immune to SEU, and that is becoming a major concern for systems with high-reliability requirements, such as automotive or space applications. However, for SRAM-based programmable-hardware systems, the problem is even more severe. Indeed, Programmable hardware is configurable in the sense that the content of configuration memory defines the hardware architecture of the system. The configuration memory space is made of millions (even billions for the largest devices) of bits dedicated to programming with millions of logical and sequential elements[2, 1].

The high-transistor density of configuration memory makes it very sensitive to SEUs. Additionally, since configuration memory is rarely written, errors can accumulate in the memory space, leading to architectural errors that can affect the system execution, eventually leading to failure.

Even if mitigation solutions and analysis methodologies for dealing with SEUs in configuration memory have been proposed, it is still a hot topic since device complexity is rising up rapidly, increasing logic resources and on-chip elements such as microprocessors, memories, DSPs, and networks-on-chip. Mitigation techniques such as Triple Modular Redundancy and scrubbing are effective but may introduce significant overheads in area, availability time, power consumption, and development efforts.

Further complicating the question is the absence of detailed information from the vendors on the characteristics of the configuration memory. Indeed, no official notions of how the circuit is encoded in configuration memory are available, limiting the ability of industry and researchers to develop static and dynamic methodologies for analysis, reliability evaluation, or mitigation that could benefit from low-level knowledge of how the circuit can be modified by SEUs occurring in the configuration memory.

When it comes to third-party tools to support FPGA design analysis, they are mostly used to facilitate and automate design placement and routing or partial reconfiguration [23–26]. Some tools have attempted to reverse engineer bitstreams, with only partial success, and most are targeting FPGA families that are three generations or more out of date [27–29]. Additionally, these tools are based on the Xilinx Design Language (XDL) supported by ISE, which is no longer supported in the more recent Vivado Design Suite, the provided vendor tool for development on Xilinx programmable devices. This lack of support for XDL and the lack of APIs for interfacing with the Vivado framework poses an additional challenge to interfacing third-party tools with the vendor's tool in order to perform tasks, extract information, and create custom place-and-route solutions for fault mitigation.

5.2 PyXEL: A framework for Easing Bitstream Analysis and Experiments

PyXEL is a Python-based framework initially created to simplify the analysis of fault effects in FPGA routing. PyXEL has been originally designed to work with Artix-7 FPGAs from Xilinx, but it has been widely extended further to support additional devices, families, and fault models. The current version of PyXEL underlies much of the research work that is presented in this dissertation. The software has developed gradually from the earliest versions to the current one and is still expanding. PyXEL is a Python-based tool created to make it easier to measure the resilience of programmable hardware designs and integration with the vendor's tools for enabling custom implementation and automatized solutions. It offers comprehensive support for the automation of radiation testing and fault injection experiments and includes the ability to visualize, decode and analyze configuration data of programmable hardware devices. PyXEL can also map CRAM to hardware, enabling comprehensive reliability analysis techniques such as fine-grained fault injection, specific fault emulation (e.g. open routing, LUT corruption), and fault localization in radiation testing. Furthermore, PyXEL provides a mechanism for automating and integrating fault injection tasks, bitstream analysis, and custom placement and routing solutions.

Since PyXEL and its use are integral to much of the work presented below, this chapter introduces and explains the capabilities and architecture of the framework and how it has evolved from what was presented in [22].

5.2.1 Bitstream Analysis, Visualization, and Manipulation

PyXEL inherits and extends the basic idea proposed in [30]. PyXEL has been originally designed to assist in the analysis of the organization of FPGAs configuration memory. PyXEL is able to parse the bitstream, identifying configuration memory content, header, and configuration commands.

The software can identify frames and bits of the configuration memory associated with any tile in the device fabric. It supports bitstream manipulation to inject fault models both at the memory-content level, such as Single Event Upsets or Multiple Bit Upsets, and at netlist-level, such as open-fault interconnections or LUT table modifications.

Additionally, it implements a methodology for the visualization of the configuration memory of a given design. Since each frame of the bitstream is associated with a column of hardware elements in the FPGA fabrics, PyXEL provides a visualization feature for the bitstream based on a 2-D bitmap. The 2-D bitmap enables easy visual inspections of the bitstream to correlate sections of the bitstream and parts of the programmable-hardware fabric.

To provide an example, a view of the visualization feature of PyXEL is reported in Figure 5.1. Thanks to the visualization feature, the part of the bitstream associated with an AXI IP Core has been identified. On the left is the view of the circuit as it is provided by Vivado, while on the right is the bitmap provided by PyXEL. The yellow rectangle highlights the part of the design as shown in Vivado with the part of the bitmap associated with it by visual inspection.

Additionally, PyXEL can be used to manipulate the bitstream in order to make PyXEL able to support fault injection tasks on specific elements and perform netlist manipulation.

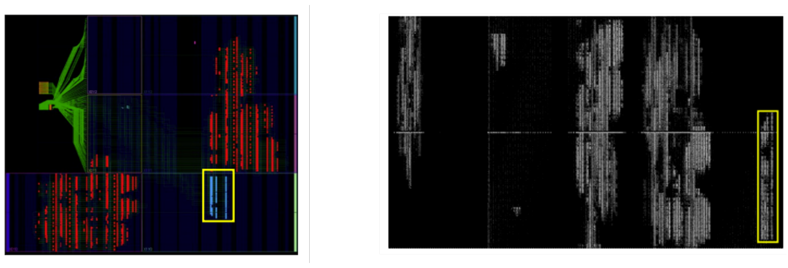


Figure 5.1 Visualization feature offered by PyXEL.

5.2.2 Vivado Integration

One of the issues that have held back development in the analysis and decoding of the bitstream of the latest devices is the migration from the ISE environment to Vivado Design Suite. Vivado is superior in functionality, usability, and performance to ISE but does not support XDL and intermediate format files such as NCD, which were useful resources for working at the layout level. Most of the information that previously was possible to obtain from ISE is still embedded in the Vivado environment but is complex to access since they are embedded in the environment or in files with an undisclosed format. Additionally, Vivado relies on Tcl, which is *de facto* the standard language for commercial CAD tools. However, programming in the Tcl language can be complex and the language is slow even compared to other interpreted languages such as Python, as well as weak in the support for object-oriented programming.

For these reasons, PyXEL has been implemented in Python in order to be easy to learn, extend and integrate into other projects. However, it still needs to rely on Vivado for extracting information, generating bitstream, and reading or creating design checkpoints. Thus, interfacing PyXEL with Vivado is necessary to automatize steps such as extraction of layout and device information, place-and-route solutions, netlist modification, and so on. As a solution, a Tcl server that runs directly in Vivado transparently to the user has been developed. In this way, Pyxel can interface with Vivado, but limits Vivado's tasks to extract the information to be used on the Python side. Additionally, this mechanism made it possible to provide APIs for the user to simplify and automate complex operations, such as routing, analysis, or placement algorithms.

5.2.3 Bitstream Decoding

Bitstream decoding is important to support fault injection and analysis methodology for programmable hardware. Many of PyXEL's features rely on the relation between configuration memory content and resources in the programmable hardware fabric. Bitstream decoding process has been automatized to be easily applied to any FPGA family. It consists of two parts.

Firstly, a coarse mapping is performed that maps any tile to the configuration memory bits associated. This part requires extracting a few pieces of information for a device family using visual inspections or simple analysis of the bitstream. Most of the information is common to all the devices of a family. In particular, the rules of ordering for the clock regions and the width in frames of tiles types are needed. This information is provided to the tile mapper. Additionally, the tile mapper can autonomously extract from Vivado the information on the architecture of the programmable hardware. These notions are enough to build a comprehensive function that maps any tile to the correspondent configuration memory section.

Secondly, the encoding and decoding functions for each tile are a more complex task, due to the unique characteristics of each tile typology. Considering that tiles of the same type are configured in the same way regardless of their position in the FGPA fabric, a mixed methodology was adopted based on the correlation between the bits used to program a tile type in different designs, which was then refined by hand by visual inspection. Currently, the method has been used to decode only the most important tiles, such as the switch boxes and CLBs. The method takes any number of designs as input. It extracts information about the configuration of each tile used from these designs and associates this configuration state with the bitmap related to this tile in the configuration memory using the tile mapper. By correlating all tiles of the same type and their bitmaps with each other, it associates the individual bits of the tile bitmap with the resources used. In this way, it incrementally reduces the number of bits associated with a specific configuration of a specific resource within a tile. The approach is incremental and refines, iteration after iteration, the set of bits associated with a resource with good accuracy. The advantage of the approach is that it is not necessary to create specific designs to decode part of the resources in a tile with good accuracy. Further refinement of the results can still be achieved with designs that use specific resources. The methodology allowed full decoding of CLBs to the vast majority of PIPs in interconnection tiles. Particularly for Ultrascale devices, already with 10 input designs we previously used for other purposes, which the framework analyzes in the order of minutes per design, 99% of the PIPs could be decoded without any user intervention.

The approach for switch matrices decoding is schematized in Figure 5.2

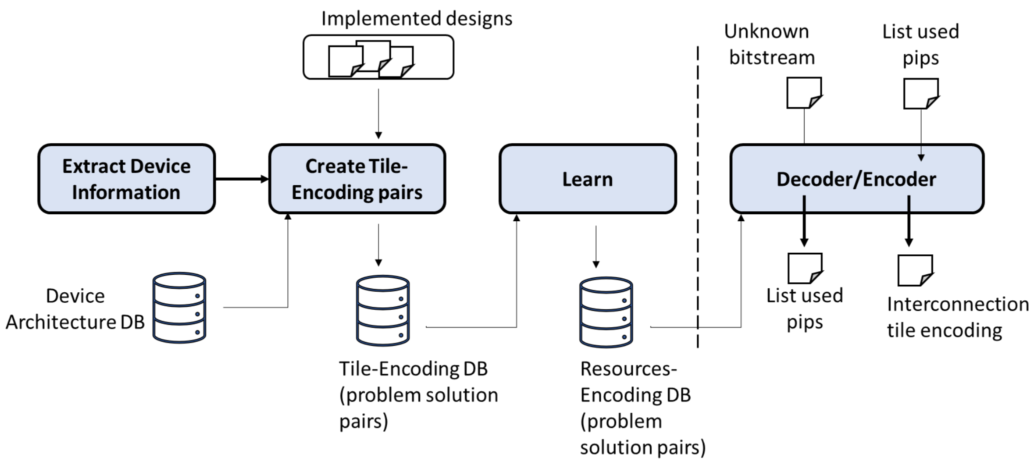

Implemented designs
Unknown bitstream
List used pips
Extract Device Information
Create Tile-Encoding pairs
Learn
Decoder/Encoder
Device Architecture DB
Tile-Encoding DB (problem solution pairs)
Resources-Encoding DB (problem solution pairs)
List used pips
Interconnection tile encoding

Figure 5.2 Bitstream Decoding Flow.

As a validation experiment, two bitstreams were analyzed by PyXEL with the purpose of inferring from the bitstream alone the PIPs used in each tile of the design. The first benchmark was B12, taken from the ITC'99 benchmarks and featured 4,149 PIPs, while the second, which cannot be revealed because it is protected by a non-disclosure agreement, featured 113,332 PIPs. PyXEL was able to predict enabled and disabled PIPs with an accuracy of 99.3%, a recall of 100%, and a precision of 78% for both designs.

5.2.4 Automation Support for Fault Injection and Radiation Testing Experiments

Configuration memory corruption due to radiation effects is one of the most critical issues affecting programmable hardware reliability. Evaluating applications against the SEU fault model in configuration memory is mandatory for programmable hardware devices operating in radiation environments or adopted in mission-critical applications.

However, since the vendors do not provide information about how resources in the programmable hardware fabric are mapped and configurated by bits of the configuration memory, fault injection evaluation approaches are subjected to some limitations. Firstly, the huge number of resources and, consequently, the large memory space of the new devices makes fault injection in random locations useless if evaluating the robustness of single modules is desired. Secondly, when a circuit is implemented on programmable hardware, only a small part (usually empirically estimated as around 10% of the whole configuration memory) produces errors. It is due to the high number of resources in FPGA that stay unused even with high utilization. This results in many bits that can be selected for fault injection but will likely not produce any error since they are not used by the application. Thus, time is wasted in evaluating bits that cannot be a source of errors for the system.

Essential bits tried to overcome this issue, providing a subset of bits of the configuration memory that are more likely to produce errors. Even if essential bits reduce the injection space, they still do not provide any information on which module has been targeted by a fault. Furthermore, essential bits are provided with a file format that is not open-source, making it usable only with SEM IP Core provided by Xilinx. Finally, they provide a coarse-grained selection of used bits based on used tiles instead of basic elements and PIPs, significantly overestimating the subset.

Pyxel offers several features to support the fault injection analysis for programmable hardware.

PyXEL can be used as as a fault generation platform instrumented to inject faults with a high degree of customization. Methods are provided for injecting specific fault models, such as SEUs, and MBUs. The inner knowledge of the relation between FPGA fabric tiles and bitstream structure allows for targeting specific modules implemented in the FPGA fabric. The information on how specific resources, especially PIPs, and LUTs, are programmed enables the emulation of topological faults for routing, such as conflicts and ope fault models or logic function corruption. This feature is valuable also for ASIC prototyping, where specific faults at the hardware level can be quickly emulated. Please note that this approach is partially prevented by modern commercial tools where faulty netlist prevents bitstream generation without the possibility of forcing the process.

As an experiment manager, PyXEL provides methods for fully automatizing experimental flow both during radiation testing and fault injection campaign. In particular, PyXEL can be used for programming both processor systems and programmable logic of Xilinx devices, collecting data from the serial ports, and performing configuration memory readback and error detection. PyXEL also provides an embedded logging mechanism.

5.3 Analysis of Electrical Behaviour of Faulty Interconnections

About 80% of the bits in the configuration memory of modern programmable hardware are used for programming routing resources. Routing resources are especially critical for several reasons. Other than being a significant part of the used resources of circuits mapped on programmable hardware, they can be a source of failures that can propagate along different paths or create interference and conflicts between different modules and nets as a result of SEU in the configuration memory.

FPGAs programmable interconnection structure is based on Programmable Interconnection Points (PIPs). These PIPs connect longlines to create specific paths for signals to be propagated through combinational and sequential elements on the fabric.

Various studies have been conducted on the impact of Single Event Upsets on the FPGAs routing structure and the electrical effects that these faults can produce. Most of the research on this topic are dedicated to simulating the SEU effect on routing or emulating faults in the physical device. Considering that the CRAMs can consist of billions of bits, and test routines for evaluating faults effect can be time-demanding or not comprehensive, these approaches give a general overview of the design reliability while requiring a significant amount of time.

Indeed, due to the lack of information provided by FPGA vendors, it is difficult to accurately map routing resources into bitstreams. This lack of information makes it difficult to inject specific fault configurations into FPGAs and to decode and manipulate bitstreams.

5.3.1 Methodology for Interconnection Faults Analysis

The version of PyXEL used to support this work has been published in [22]. It automates the process of carrying out experiments in FPGAs, making it more research-friendly and reducing the possibility of errors.

In particular, PyXEL offers APIs to control the place-and-route of a circuit within FPGA fabric to select which routing resources to test, generate the placed-and-routed design, and evaluate the electrical effects produced as a Boolean logic function of the input data.

An SEU affecting the configuration memory section associated with a switch matrix can generate many different faulty configurations. Topological modification can be categorized into topological models. Common examples are:

- **Open**: a PIP is deactivated interrupting connection in the net.
- **Conflict**: a PIP is activated connecting two used junctions, as a result, a node is now driven by two drivers.
- **Antenna**: a PIP is activated between an unused and used junction.
- **Bridge**: a PIP is modified driving another junction

The listed fault models are visualized in Figure 5.3. However, this is only a subset of the possible faulty configuration that is variegated due to the high number of allowed configurations.

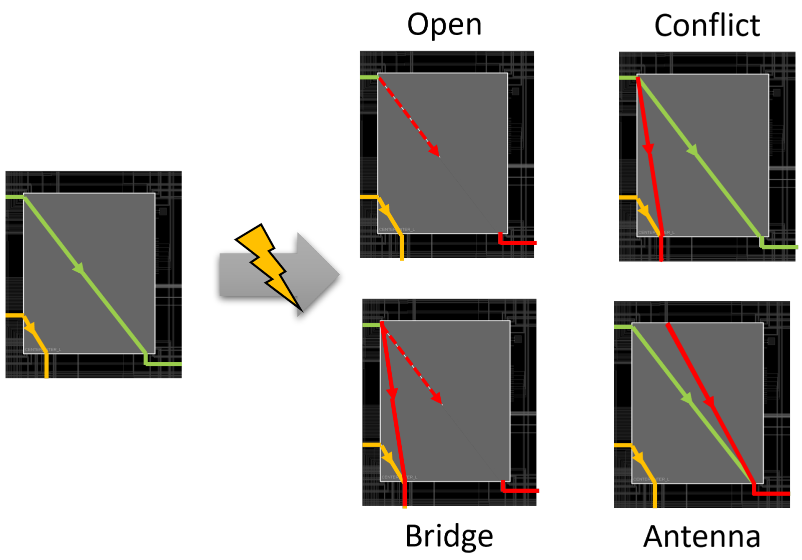


Figure 5.3 Example of possible faulty configurations of the connection that can be caused by SEUs in configuration memory

Previous works were based on XDL methodologies for generating faulty bitstreams. However, we found during our experiment that this approach presents some limitations in the new bitstream generator tool, and produce unreliable configuration file. In order to produce the faulty configuration a mixed approach has been proposed.

Initially, a benchmark design is implemented by using the vendor tool in the FPGA fabric. Through automatized place-and-routing manipulation bitstream files are generated using different PIP elements. These files are then analyzed by PyXEL for extracting bitmaps embedding the information for enabling or disabling specific PIPS.

The benchmark design consists of a UART receiver, a UART transmitter, and two registers. A byte is sent on the serial connection from the experiment manager on the host computer, stored in a register, forwarded to the second register by using the routing and the PIPs on the FPGA fabric, and then transmitted by the UART module to the host computer. PyXEL methods are used to route two nets that will be used for connecting two FF of the registers using specific PIPs under the test of an interconnection matrix. A conceptual schema of the platform, the benchmark circuit, and the evaluation methodology is exposed in Figure 5.4

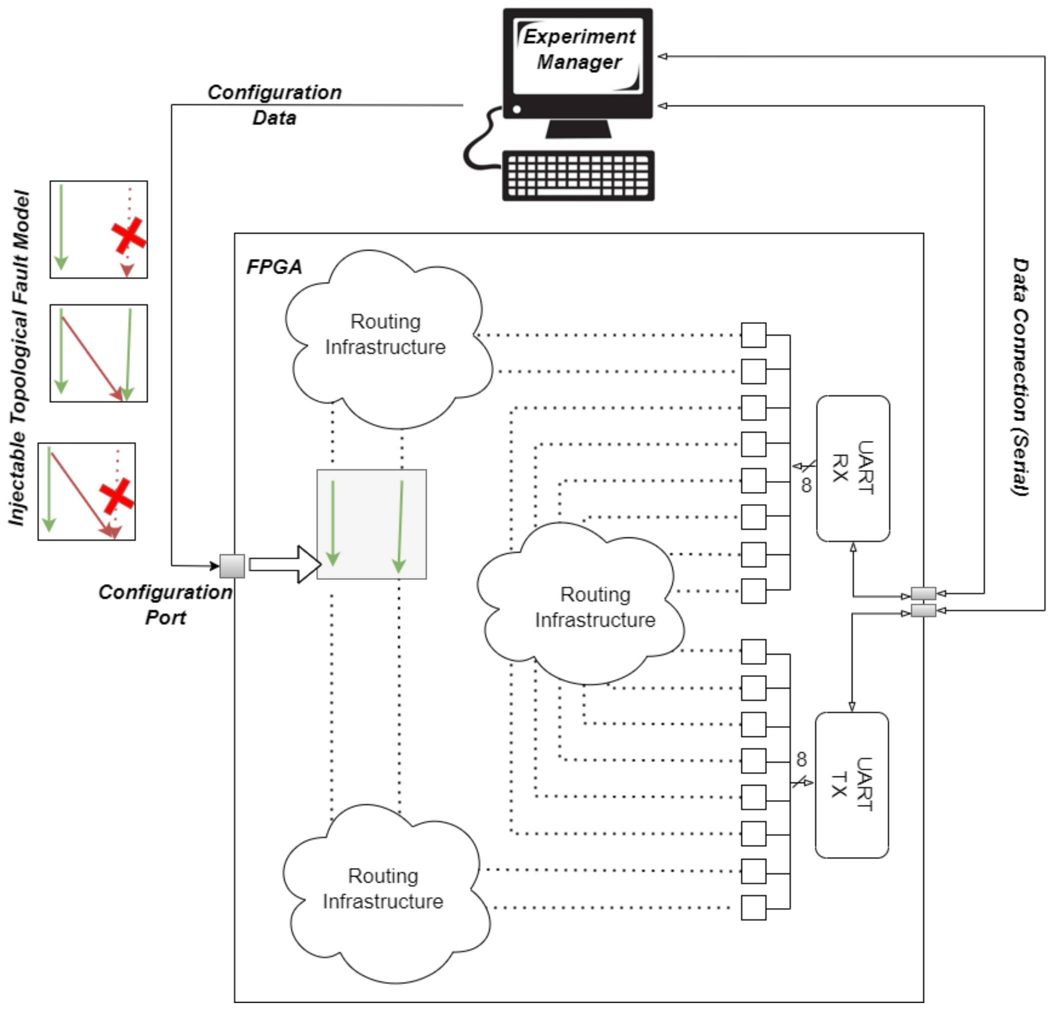


Figure 5.4 Conceptual schema of the system architecture, the design benchmark, and methodology for electrical characterization of faulty programmable interconnections

During each experimental evaluation, we want to modify the circuit programmed by the bitstream in order to produce specific topological faults we want to observe. This is achieved by bitstream manipulation without the support of the vendor toolchain. Indeed, the vendor's tool prevents the generation of faulty configurations during application design.

The proposed approach for manipulating the circuit through bitstream manipulation has been validated by generating two versions of the same valid configuration of the circuit, having no faults and using specific PIPs for routing the nets. The first version has been generated using the vendor CAD tools, while the second one uses PyXEL. We compared the bitstream of PyXEL-generated and Vivado-generated sections of the manipulated interconnection matrix, as well as the result of the benchmark design to check the consistency. We routed the input bits to different output bits successfully.

5.3.2 Faulty Interconnection Electrical Characterization

Experiments are based on the benchmark design. The design is modified at the bitstream level using PyXEL. Indeed, differently from the traditional XDL approaches, Vivado Design Suite does not allow a bitstream for faulty netlist to be generated.

We evaluated different topological fault behavior. Experiments have been automatized using PyXEL. A list of PIP pairs under test is provided to the framework, as well as a description of the faulty configuration to generate. The framework produces a golden design using the desired PIPs and then manipulates the bitstream for inducing the target topological fault models. At this point, it programs the device, stimulates the circuit through the serial port, waits for the outputs, and analyzes the relation between the inputs and the outputs for inferring the logical behavior observed as a result of the topological faults.

Additionally, modifying the circuit through bitstream manipulation means that it is not necessary to implement a new design using Vivado Desing Suite, saving a significant amount of time.

The device selected for the experimental analysis was a Xilinx Kintex-7. Antenna, Bridge, Open, and Conflict fault models have been selected to be evaluated.

The Antenna did not show faulty behavior. Junctions seem to be robust against disturbance introduced by the antenna topological fault model. However, only single antennas have been tested. It is possible that multiple antennas may produce unexpected behavior, but this possibility needs to be investigated further.

A bridge fault has the same behavior as a conflict fault on the multi-driven junction plus an open fault on the junction that is not driven anymore.

Open Faults always have produced a stuck-at-1 bit. This could mean that unused junctions use pull-up resistors.

Finally, the conflict shows interesting results. Indeed, conflict faults have produced different kinds of electrical behaviors. A few hundred PIP pairs have been tested and common patterns relating to electrical behavior and the encoding in the

Table 5.1 Conflict Electrical Characterization Resume

Logical Fault	Ratio [%]
Wired-AND	57.45
Wired-OR	39.71
Driven-By	2.94

bitstream of the PIPs have been found. In particular, the three behavior observed for the junction driven by the two PIPs was:

- **Wired-AND**: The logical value on the output junction is a logical AND between the logical values expected on the junctions of the two PIPs that are driving the output junction.
- **Wired-OR**: The logical value on the output junction is a logical OR between the logical values expected on the junctions of the two PIPs that are driving the output junction.
- **Driven-by**: The junction behaves like it is driven by only one of the two source junctions, named the Predominant Junction.

However driven-by behavior was much rarer than the others two, while AND was slightly more common than OR. Occurrences have been reported in Table 5.1

Additionally, some recurring patterns have been identified in how PIP pairs were encoded in configuration memory and the electrical characterization. It is difficult to identify the occurrence of a Forced-by behavior that occurred rarely. Nevertheless, our research revealed that when two Programmable Interconnect Points (PIPs) are encoded with a bit having the 1 value in the same position of the configuration memory and no Forced-by behavior is present, it behaves as a Wired-AND. Conversely, if the condition is not met, a Wired-OR behavior is observed.

5.4 Research Advancements on the Analysis of Single Event Upsets

The PyXEL tool has been developed for easing configuration memory analysis and understanding, as well as fault injection experiments and campaigns. PyXEL to offer comprehensive reliability analysis techniques, which include fine-grained fault injection, specific fault emulation (such as open routing and LUT corruption), and fault localization in radiation testing. These techniques are not currently available from vendors or third-party tools and assist in solving the mapping between CRAM and hardware. Additionally, using PyXEL, it has been possible to evaluate electrical behaviour of faulty interconnections, that can provide a more accurate model of the faults produced by Single Event Upsets in the configuration memory of programmable hardware.

Chapter 6

Radiation Test Analysis

6.1 Overview on Radiation Test Analysis

Accelerated radiation testing is an important methodology for evaluating the reliability of electronic components and systems. In a much shorter period, it simulates the effects of exposure to radiation that can be experienced in space or other radiation environments. This allows us to quickly identify, evaluate and address potential issues of the system before they are deployed in the field. This is especially important for safety-critical applications that involve the use of costly equipment, such as those used in satellite and space exploration, or are responsible for preventing natural disasters or preserving human life.

Radiation testing presents some unique advantages with respect to other techniques such as simulation or fault injection. Even if at higher fluxes for speeding up evaluation, it emulates the real physical phenomenon on the actual device, thus usually providing more accurate results.

Radiation testing can be adopted for evaluating the sensitivity of systems either to Total Ionizing Dose (TID) or Single Event Effects (SEEs). The tests that will be presented in the following chapter are dedicated to testing SEEs. These tests evaluate how the device under test will operate while it is bombarded with accelerated high-energy particles, such as protons or heavy ions. SEE evaluation can involve either static characterization, which can be used for evaluating the sensitivity of components and modules, for instance, the SEU rate in memory, or dynamic characterization where the system's functionality is evaluated.

However, radiation testing also presents some challenges. Testing is becoming more difficult as devices grow much more complex. It is usually hard to isolate completely which component, especially in the system-level experiments is exposed or failing. Additionally, it requires specialized facilities provided with particle accelerators. This can make device testing costly and slow down time-to-market based on facility availability. Moreover, exposure to accelerated radiation testing can permanently destroy the device's functionality.

Finally, radiation testing requires significant efforts in developing the experimental setup and data acquisition system for ensuring the accuracy of the results and preventing device destructive events.

Due to the high performance characterizing commercial reconfigurable-hardware SoCs, and their appeal for state-of-the-art applications, including in fields such as space exploration and high-energy particle experiments, the evaluation of these systems against ionizing radiation is a hot topic.

The complexity of system-level analysis is further compounded by the integration of various components, such as programmable logic, processors, and memories, onto a single chip. Various studies have been conducted to investigate the characteristics and reliability of SRAM memories, hard and soft microprocessors, and hardware acceleration when exposed to ionizing radiation. However, much of the research work on programmable hardware only focuses on the configuration memory, which is a very sensitive component, leaving in the background, processor systems, system memory, or system-level dynamic evaluation.

6.2 Testing the Zynq On-Chip-Memory with Protons

6.2.1 State of the Art of Radiation Analysis of SoC Memory

In the past decade, system-on-chip solutions integrating into the same chip general-purpose processors and application-dedicated processors have been widely adopted. In particular, hardware-reconfigurable SoC has proven to be an attractive solution since it offers the possibility to implement custom hardware architecture for accelerating the most demanding tasks. However, before these devices can be adopted in mission-critical applications, their reliability must be thoroughly evaluated, as the increasing operating frequency, decreasing operating voltages and transistor size have made them more susceptible to soft errors.

Many studies examined the characterization and radiation tolerance of SRAM memories, processors, as well as SoC. The majority of investigations into the memories of Zynq devices have only focused on the CRAM programming the FPGA, without taking into account the microprocessor and memory. However, SRAM memories are particularly sensitive to radiations such that they are also used as detectors for particles after a comprehensive characterization [31]. Due to the complexity of modern systems, a comprehensive analysis of the characteristics of the memory in these systems is required in order to properly consider the effect of faults on the overall reliability of the system.

Only a few studies have explored the On-Chip Memory (OCM) of SoC under radiation. In [32], the author presents a heavy-ion irradiation of a low-cost Commercial-Off-the-Shelf microcontroller, including an SRAM dynamic analysis, but the analyzed events are limited to SEU cross-section. [33] and [34] studied the memories of a Zynq System-on-Chip, such as Block RAMs, CRAM, and OCM, under heavy-ion and proton radiation, respectively. Finally, [35] presented an attempt to relate OCM errors to other components of the SoC.

The Zynq devices lacked an accurate analysis taking into account the integrated nature of the SoC, the interface and connection of the memory with other components, as well as the usage of the memory by other modules. Such an approach will provide a more accurate assessment of the radiation sensitivity of a memory component and its contribution to the reliability of the system.

To address this problem, we provided a comprehensive evaluation of the events that occurred in the OCM of an ARM Cortex-A9 processor system exposed to proton radiation[36]. The fault resulting from the radiation campaign, as observed from the processor side, has been described and categorized. The advantage of such analysis is to provide a realistic fault model to be adopted in reliability analysis such as fault injection, simulation, and emulation. Such a fault model can be a valuable resource for preliminary robustness analysis or when a radiation test is not a viable solution.

6.2.2 Proton Radiation Testing Setup

At the Paul Scherrer Institute (PSI) Proton Facility, a proton test analysis was conducted by irradiating a Zynq-7020 system-on-chip with energies ranging from 16 to 200 MeV. The goal was a static analysis of the proton-induced SEE in the OCM of a Zynq 7020. A dynamic analysis approach has been adopted for evaluating errors occurring during the use of the OCM memory by the system processor.

A PYNQ-Z2 board has been utilized in this test. It incorporates a Zynq-7020 system-on-chip, which is equipped with an ARM Cortex-A9 MPCore processor and a 256 KB SRAM memory interconnected through a Snoop Control Unit (SCU). The on-board DDR memory, which was not directly exposed to the radiation source, was selected as the storage location for the test program so as to minimize the potential for external errors such as processor halting and code corruption. Caching was disabled to ensure that any events occurring directly in the SRAM memory could be observed through reading and writing of the memory content.

To ensure continuous monitoring of the on-chip SRAM memory, a software routine was implemented to run on the ARM Cortex A9 processor. The software continuously writes new values into the memory, verifying that the value written during the previous test loop has not been corrupted, and verifying that the current value has been written and read correctly.

The system under test was placed in the irradiation room and connected to a host computer in the control room using a USB-screened cable with 3 signal repeaters (1 every 5 meters), for a total of 20 m connection. In the event of an error being detected, the processor of the SoC notifies the host computer that logs the event. Furthermore, the software routine is equipped to identify both SEFI errors, such as a memory cell not being able to be written or read correctly, while the experiment manager is able to detect system halting and send software reset or notify the need for a power reset, to be manually executed from the control room via a power switch.

The board in the irradiation chamber was docked to a flexible support structure. The design was tested with a variety of energies and fluxes, the conditions of which are detailed in Table 6.1

Table 6.1 Proton Test Conditions: Energy, Flux and Fluence

Energy [MeV]	Flux [$cm^{-2} s^{-1}$]	Fluence [cm^{-2}]
16.04	1.89×10^7	2.17×10^{10}
29.31	4.12×10^7	1.70×10^{11}
50.80	4.02×10^7	1.94×10^{11}
69.71	4.11×10^7	6.70×10^{10}
101.34	4.32×10^7	1.86×10^{11}
151.18	4.09×10^7	1.23×10^{10}
200.00	4.14×10^7	1.97×10^{11}

6.2.3 Proton Test Results and Fault Models

We evaluated events affecting the OCM of a Zynq-7020. A detailed classification of the observed events and their respective cross-sections is proposed. The events cross-section and the bits cross-section have been evaluated and the results are reported in Figure 6.1. In the SEE cross-section, each event contributes as a single event to the cross-section computation, while in the bits cross-section the number of faulty bits contributes to the total bits cross-section.

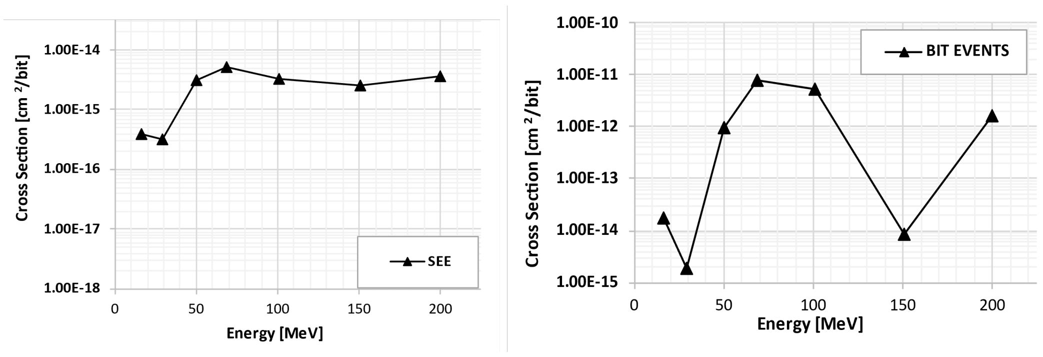


Figure 6.1 SEEs and Bits Cross-Sections for OCM of a Zynq-7020.

A comprehensive assessment of the events observed on the on-chip SRAM memory from the processor system during radiation tests allowed us to propose a set of fault models. These fault models provide a description of the events impacting the on-chip memory from the processor perspective that can be valuable in reliability analyses which can benefit from a precise fault model to perform preliminary analysis.

SEUs have been identified as the most common event. The SEU cross-section is reported in Figure 6.2 for the evaluated energies. The SEU cross-section has been computed for both 0-to-1 and 1-to-0 transitions but no significant differences have been observed

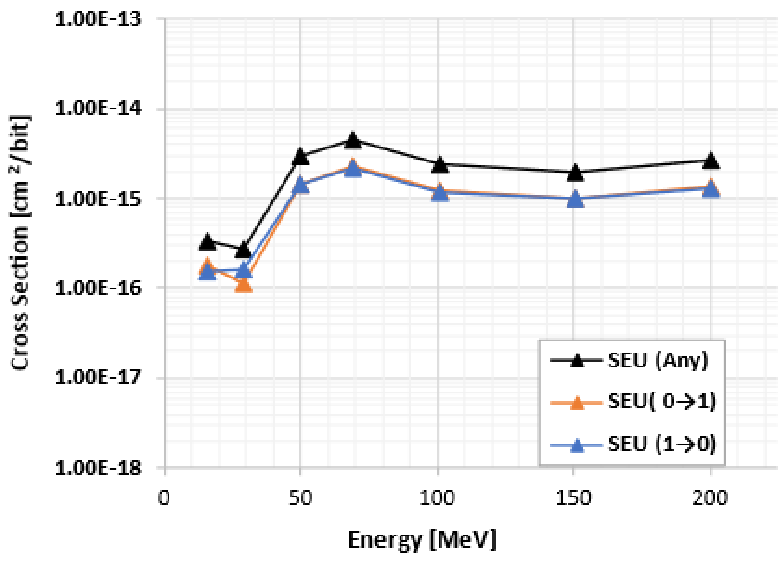


Figure 6.2 SEUs Cross-Section for OCM of Zynq-7020.

SEMU have been observed during the radiation test, with a moderate frequency. They turned out to be more common and characterized by a higher number of bits when testing at higher energies. Again, 0-to-1 and 1-to-0 transitions were evaluated but without noticing significant variation. A pattern was noted that can probably be associated with the physical layout of the memory. In fact, the bit flips associated with the same event always affected the same significant bit in memory words at recurrent logical distances, the distribution of which is shown in Table 6.2, while size distribution is reported in Table 6.3.

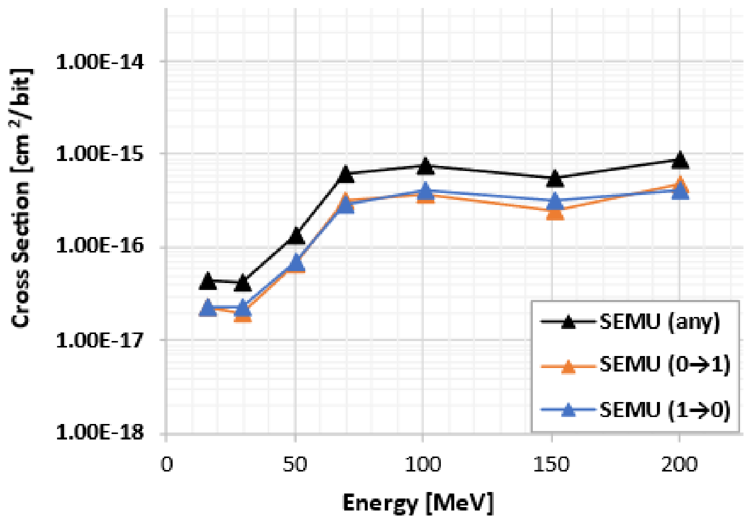


Figure 6.3 SEMUs Cross-Section for OCM of Zynq-7020.

Table 6.2 Normalized Occurrence of Logical Distance of SEMU in OCM.

Logical Distance [bytes]	Occurrence
128	61%
4	12%
124	6%
132	3%
16	1%
256	1%
others	less than 1% (total 16%)

Table 6.3 Normalized Occurrence of SEMU Size.

Size [# bit]	Occurrence
2	64.57%
3	20.15%
4	7.96%
5	3.47%
6	1.28%
7	1.03%
8	0.51%
10	0.51%
11	0.26%
12	0.13%
15	0.13%

Finally, Burst Events were observed to occur at a much lower rate than SEUs and SEMUs. However, the Burst Events had a much more significant effect, affecting numerous memory locations and cells simultaneously. In some cases, this type of event, called Burst Stuck-at, caused the memory to be inaccessible, leading to Single Event Functional Interrupts. In order to return to nominal behavior, a power cycle was required.

Table 6.4 Normalized Occurrence of Memory Locations Number affected by a Single Burst Events.

Number of Affected Locations	Occurrence
Less than 1,000	16%
Between 1,000 and 10,000	62%
More than 10,000	22%

Burst Events occurred with different characteristics. A clear/set event cause the content of a large part of the on-chip memory to be cleared (0 value) or set (1 value), while a burst error causes the corruption of a large number of bits. Finally, Stuck-at events force the results of reading operations to a fixed value. The distribution of different burst events size is further detailed in Table 6.4. Although no burst events were detected at 150 MeV during our experiments, we theorize that these events could still occur at this energy, however, due to the lack of statistical evidence, we are unable to estimate a cross-section. The burst events cross-section is reported in Figure 6.4

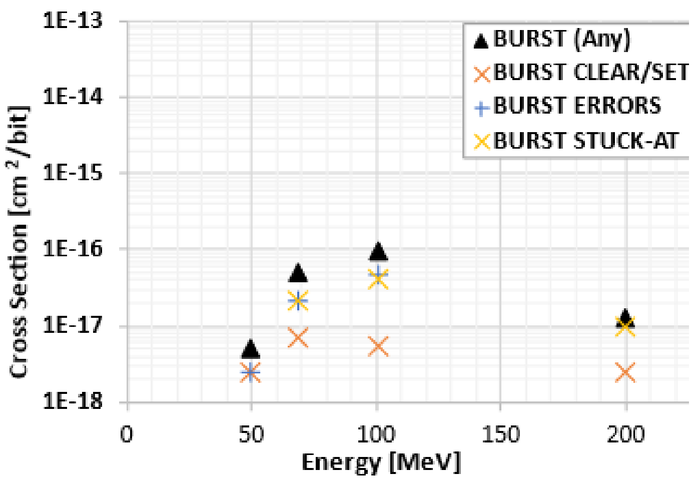


Figure 6.4 Burst Events Cross-Section for OCM of Zynq-7020.

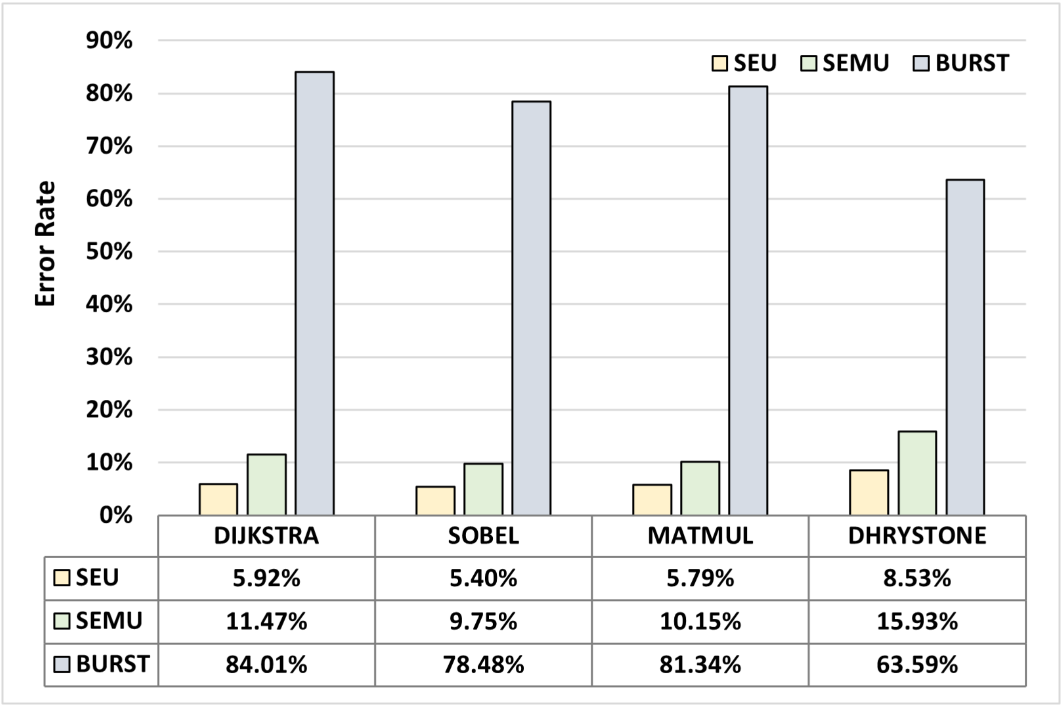
6.2.4 On-Chip Memory Fault Emulation

The identified fault models enable the possibility to evaluate the impact on software applications emulating the fault affecting the OCM from the processor point-of-view. Four bare-metal software programs have been evaluated, each with a distinct fault injection campaign and the proposed fault models previously discussed.

The same system-on-chip has been used in the fault injection campaigns. The ARM core is utilized to execute the software benchmarks, while faults are emulated in the OCM. As an experiment manager and fault injection platform, we used PyXEL. The manager runs on a host computer and orchestrates the fault emulation process and collects results using a serial connection.

The analyzed benchmark MatMul, which multiplies a series of matrices and provides the resulting matrix; Sobel, which applies an edge detection algorithm to a picture and generates the resulting picture; Dijkstra, which uses Dijkstra's algorithm to find the shortest paths between two nodes in a graph and sends the paths and their costs through an output channel; and Dhrystone, a synthetic computing benchmark that performs string processing tasks and sends the results through an output channel.

Each software benchmark has been evaluated against the fault model presented, such as SEU, SEMU, and Burst events. Each campaign consists of 10,000 injections, fault locations have been generated randomly and fault model characteristics are based on the statistical results obtained during radiation testing.



	DIJKSTRA	SOBEL	MATMUL	DHRYSTONE
▢ SEU	5.92%	5.40%	5.79%	8.53%
▢ SEMU	11.47%	9.75%	10.15%	15.93%
▢ BURST	84.01%	78.48%	81.34%	63.59%

Figure 6.5 Overall Error Rate of the Software against Evaluated Fault Models

The error rate of the applications is reported in Figure 6.5 and categorized in Figure 6.6. Even if there is a slight variation in the errors among evaluated software, the behaviors are consistent. Compared to SEU, the errors only marginally rise for SEMUs and experience much higher values for burst events. Interestingly, the ratio of Silent Data Corruption increases only marginally for burst events, in spite of the considerable rise in the number of errors, mainly due to halt errors. Due to the alarming nature of SDCs passing silently, it is important to be aware that burst events mainly lead to halting errors, which, combined with their lower cross-section, will result in a lower rate of silent errors compared to what could have been anticipated for such an impacting event. Nevertheless, they should still be taken into consideration due to their great impact on system availability. Furthermore, the percentage of halt outcomes to which applications appear to be vulnerable for all fault models should prompt designers to consider the necessity of finding solutions to reduce the occurrence of these effects, which, although easy to detect at run time (unlike SDCs), contribute significantly to the system total error rate.

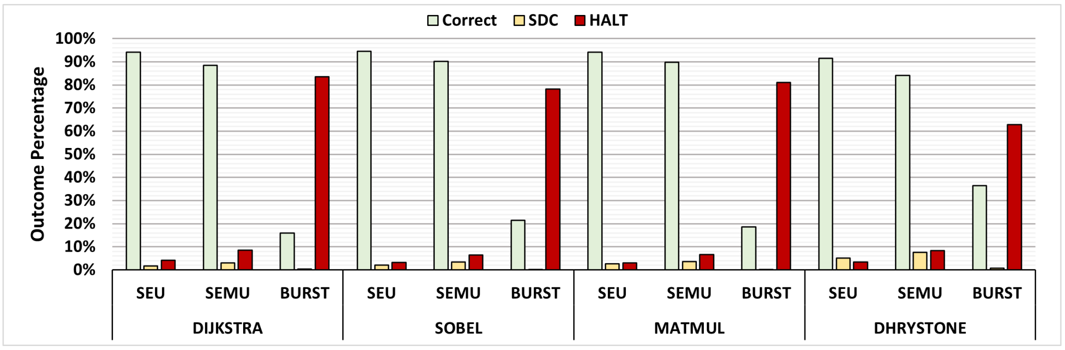


Figure 6.6 Results categorization for different software and fault models

6.3 A CRAM Technology Analysis: CMOS vs FinFET

The release of new programmable hardware devices has followed the common trend of Moore's Law. New families and devices have been released over the years exploiting new technologies, moving toward miniaturization of the transistor size.

In particular, Xilinx SRAM-based programmable hardware migrated from 28 nm CMOS technology adopted in Series 7 devices toward the FinFet multigate technology. Indeed UltraScale and UltraScale+ devices make use of 20 nm, 16 nm, and 7 nm manufacturing technology. Since these transistors are based on different gate structures, they have different electrical characteristics, thus different sensitivity to ionizing radiation. Many factors have a role in radiation sensitivity, including operational voltages and the physical structure of the component.

Many works are dedicated to evaluating and characterizing the FinFet and CMOS transistors against radiations. However, despite the abundance of research that has been conducted to compare the radiation sensitivity of FinFET and CMOS-based ASICs, there has been a lack of evaluation into the radiation sensitivity of reconfigurable FPGAs, which are currently undergoing a technological transformation.

For this reason, we evaluated the radiation sensitivity of two Xilinx FPGAs based on different fabrication technologies: 28 nm CMOS Zynq and 16 nm FinFET UltraScale+. The experimental analysis is based on two proton radiation tests conducted at the PSI facility. The dynamic test was based on evaluating the reliability of a hardware-accelerated multi-core engine implemented on each FPGA. During the test, the static content of the CRAMs has been evaluated as well. The results of the experiments demonstrate that 16 nm FinFET is one order of magnitude less sensitive to Single Event Upsets (SEUs) compared to 28 nm CMOS, but more prone to a critical event, such as Single Event Latch-Up. Additionally, a comprehensive analysis of the occurrence of Single Event Multiple Upsets (SEMUs) for each technology has been conducted [37].

6.3.1 Proton Test Experiment Setup

In order to conduct an accurate comparison, the same benchmark was developed and deployed on two different FPGAs using configuration memories based on the CMOS and FinFet technology. The selected devices are two Zynq SoC.

As the first device, based on CMOS technology, we selected a PYNQ-Z2 Board that embeds a Zynq-7020 based on a 28 nm CMOS programmable logic and a dual-core ARM Cortex-A9 processor. As a FinFet-based device, we used a Xilinx UltraScale+ ZCU104 board that embeds a ZU7ev. ZU7ev is based on a 16 nm FinFET programmable logic and includes a quad-core ARM Cortex-A53 application processor and a dual-core ARM Cortex-R5 processor on the same chip.

A benchmark circuit replicating the hardware computing architecture of an AI-oriented hardware accelerator has been implemented on both platforms. It consists of a parallel computing unit connected with the processor that is present on both SoCs. The chosen computing cores are COordinate Rotation DIgital Computer (CORDIC) IP Cores. The benchmark is based on both the programmable logic and the microprocessor system. The microprocessor system is connected to the Zynq High-Performance ports via a Master Advanced eXtensible Interface (AXI), connected with two different AXI Direct Memory Access (DMA) IP Blocks. Each DMA IP Block can manage multiple channels and transfer data between the 16-core hardware accelerator situated in the programmable logic, resulting in a total of 32 CORDIC cores.

The general schema of the architecture of the application design is depicted in Figure 6.7, while device utilizations are resumed in Table 6.5.

Table 6.5 FPGA utilization for the Benchmark Circuits

Device	**28nm CMOS Zynq (7 Series)**		**16nm FinFet Zynq (US+)**	
Resources	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 (4.39%)

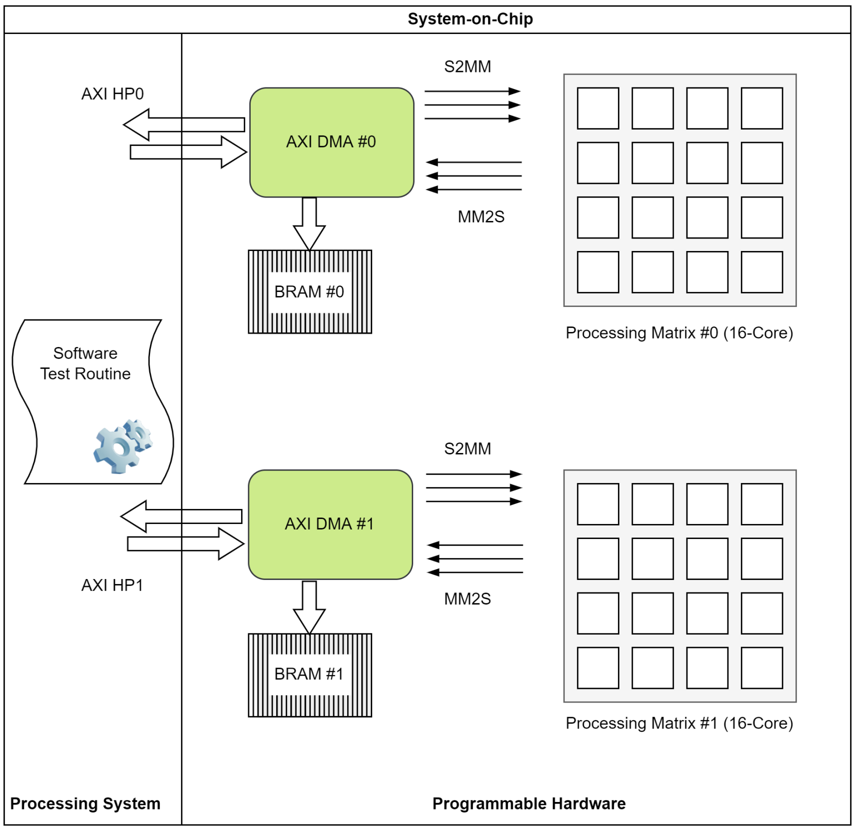


Figure 6.7 Architecture of the application running on the platforms during the proton test.

Two proton-radiation campaigns were conducted at the Paul Scherrer Institute (PSI) Proton Facility in Switzerland. Test conditions used for evaluating both the devices are reported in Table 6.6.

The PyXEL experiment manager on a dedicated host computer in the control room has been used to monitor and configure the devices via a serial link. The framework has been used to acquire the output data of the application benchmark running on the devices under test and a periodic readback of the configuration memory contents. This platform has enabled the calculation of the number of SEU

Table 6.6 Proton Test Conditions: Energy, Flux and Fluence

Energy [MeV]	Flux [$cm^{-2}s^{-1}$]	Fluence [cm^{-2}]
16.04	1.89×10^7	3.20×10^{10}
29.31	4.12×10^7	9.17×10^{10}
50.80	4.02×10^7	6.06×10^{10}
69.71	4.11×10^7	2.12×10^{10}
101.34	4.32×10^7	2.41×10^{10}
151.18	4.09×10^7	1.23×10^{10}

(Single Event Upset) events that have occurred in the configuration memory in real-time by comparing the original (without fault) bitstream and the readback file.

6.3.2 Proton Test Results, Analysis, and Comparison

The SEU cross-section for both 16 nm FinFET and 28 nm CMOS technologies is reported in Figure 6.8. It is noteworthy that for 16 nm FinFET, no SEU events were detected for energies below 50 MeV. Therefore, the cross-section value only applies to energies of 50 MeV or greater. As seen from Figure 6.8, the 16 nm FinFET technology has a lower SEU cross-section and a decreased susceptibility to radiation compared to the 28 nm CMOS technology in terms of events per bit in the configuration memory.

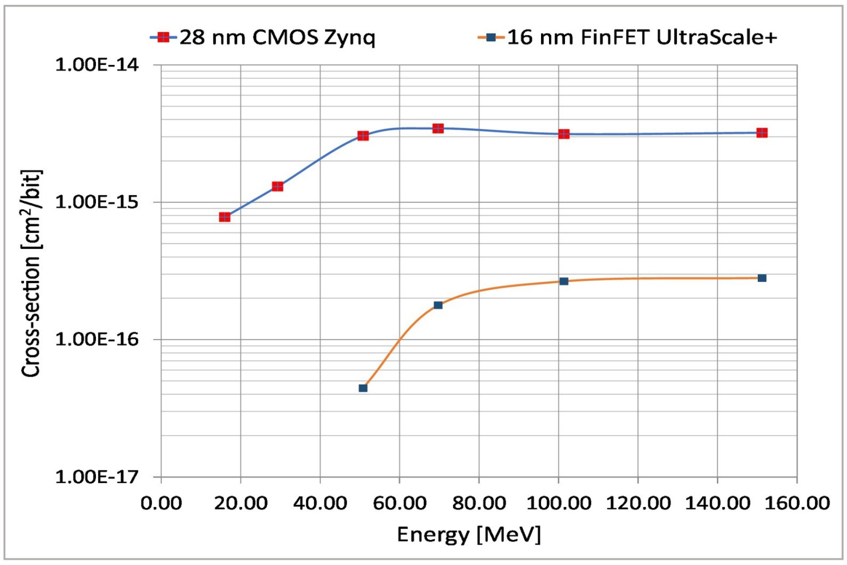


Figure 6.8 Comparison between SEU cross-section of Zynq UltraScale+ and Zynq7.

The readback process of the configuration memory content was conducted with a duration of approximately 5 s and 12 s for the Zynq-7020 and ZU7EV, respectively, over the entire irradiation period. By adjusting the particle flux to detect only a few bit flips in the configuration memory between two consecutive readbacks, it was feasible to identify groups of SEUs with a strong correlation in both time and space. The large size of the configuration memory (over 10^8 for the Zynq-7020 and more than 10^9 for the ZU7EV) and the ability to continuously evaluate the content of the configuration memory allowed us to detect clusters of bits with a high likelihood of being caused by a Single Event Multiple Upsets (SEMUs).

The SEMU patterns found from the configuration memory readback data analysis have been studied. The recurring shapes and sizes resulting from the research are represented in Figure 6.9. Figure 6.10 presents the cross-section for different cluster sizes for both technologies. It is evident that the size and pattern of the clusters vary depending on the technology being studied.

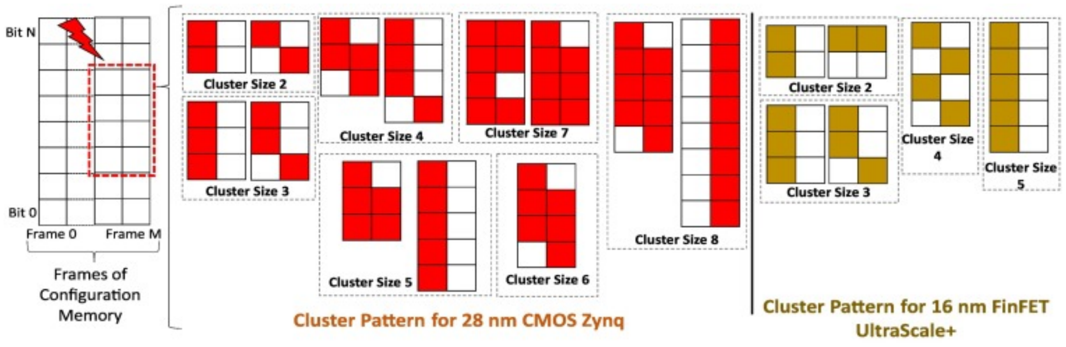


Figure 6.9 Cluster shapes and sizes resulting from proton tests in 28 nm CMOS and 16 nm FinFET.

Single Event Latch-ups (SELs) and Single Event Functional Interrupts (SEFIs) have been observed in the ZU7EV and Zynq-7020, respectively. SELs can cause a high current to flow through the device, potentially resulting in either a loss of functionality or device destruction, depending on the magnitude of the current. A power cycle is necessary to restore the device to its nominal state if the event is not destructive. Differently, SEFIs can halt the system requiring a soft reset or a power cycle to restore the normal function, but it is not destructive for the device. However, if the configuration memory is reset or corrupted, the device will malfunction, and reconfiguring the device is the only way to restore the device to its normal state. So it is essential to consider that even if more resistant to SEUs, the SEL sensitivity of

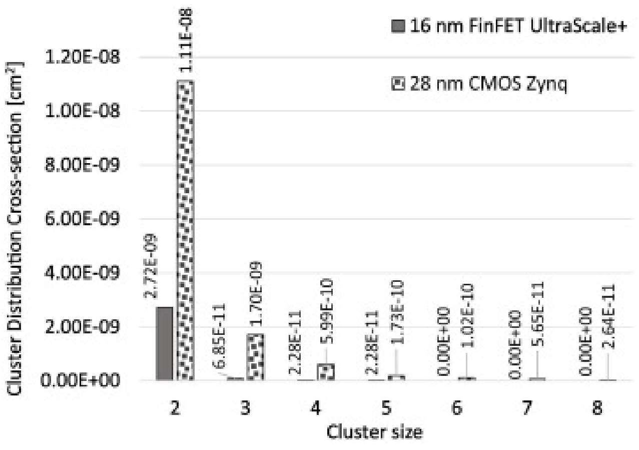


Figure 6.10 Comparison of Cross-Sections for different cluster sizes

the UltraScale device can lead to severe repercussions. In contrast, SEFI that occurs in 7-Series devices is much less concerning.

In detail, during the radiation tests, the ZU7EV and Zynq-7020 recorded up to 18 SEL and 7 SEFI, respectively. Interestingly, we did not detect any SEL events on the Zynq-7020 or any SEFI events on the ZU7EV.

The comparative study of FPGAs manufactured with 16 nm FinFET and 28 nm CMOS technology conducted through Proton radiation testing to determine the sensitivity cross-section of the SRAM cell-based configuration memory has led to interesting considerations. SEUs and SEMUs cross-sections for the two technologies showed how the 28 nm CMOS technology was more sensitive to radiation-induced soft errors in configuration memory than the 16 nm FinFET technology. However, the 16 nm FinFET had a higher SEL rate than the 28 nm CMOS during the radiation test, which could raise numerous concerns for application in radiation environments.

6.4 Research Advancements in Proton Testing of Reconfigurable SoCs

The two proton testing presented in the current chapter produced interesting data on the memories of reconfigurable system-on-chips, considering both on-chip memory and configuration memory.

A set of fault models deriving for 28 nm CMOS on-chip memory of a Zynq-7020 SoC is proposed. Comprehensive statistics and analysis of observed events affecting the on-chip memory as they manifest from the processor side are also provided. Such fault models are valuable to enable realistic fault emulation, fault injection, and simulation analyses useful for preliminary reliability evaluation or when a radiation test is not a viable solution. Indeed, while significant numbers of works are dedicated to the study of memories used by embedded processors, very few works provide an analysis of the SEE effects in the integrated on-chip SRAM of embedded processors taking into account the fault models observed by applications running on the system rather than on SEUs and bit event cross-section only.

An evaluation of the configuration memory of two different technologies has also been carried out. The 28 nm CMOS Zynq and the 16 nm FinFET UltraScale+ have been evaluated in accelerated proton test experiments. A comprehensive analysis of SEMUs has been provided for both devices. Significant differences in the sensitivity of the two devices to faults in configuration memory and Single Event Latch-up have been highlighted.

Part III

Reliability Analysis of Reconfigurable Hardware-Accelerated SoCs

Chapter 7

Evaluating Reliability of Embedded Processor Systems

7.1 Overview on Reliability of Embedded Processor

In recent years, embedded processors have been subject to increasing interest. Embedded devices, pushed by the mobile market, have been able to leverage large investments that have led to increasingly cutting-edge systems. Year after year, computing power has grown in parallel with miniaturization and power consumption reduction, and various application fields have been able to take advantage of the achievements.

In particular, systems-on-chip have become more affordable thanks to improvements in technology processes, successfully integrating more and more systems. Reconfigurable hardware has also taken advantage of this trend. New devices began to pair programmable logic with multi-core systems first and multiprocessor systems later.

The advantages offered by these devices have made them attractive also for safety-critical applications, such as the one typical of the automotive and avionics industries. However, the high number of systems integrated on the same chip made the evaluation of the reliability of these systems a complex task.

In particular, hardware-reconfigurable systems-on-chip include different systems in the chip along with programmable hardware, which results in a device capable of being reconfigured, including either hard or soft microprocessors or even both.

Hard microprocessors are the typical processors implemented in the silicon, thus not hardware reconfigurable, and are characterized by higher performance. Soft microprocessors, on the other hand, are more flexible and can be easily reconfigured. Soft microprocessors are usually designed with a set of instructions that can be changed and adapted to different applications. This makes them ideal for applications that require frequent changes or updates. The hardware architecture of these systems can be easily adapted to new needs. It allows also to the adoption of hardware solutions and mitigation methodologies to improve the fault tolerance of these systems.

Reconfigurable SoCs are becoming increasingly popular in the market as they offer the flexibility to change and adapt to new technologies and applications. These systems are often used in embedded systems, where they offer the ability to quickly and easily reconfigure the system to meet the changing needs of the application.

Additionally, soft processors often provide RTL descriptions of the system, which can be valuable to implement, emulate or simulate the device for having comprehensive reliability evaluation, while it can be more challenging for hard processors that can be subject to trade secrets held by the vendor.

Finally, soft processors and hard processors can be subject to different kinds of faults. Indeed, soft microprocessors are implemented in programmable hardware and inherit all the issues about configuration memory sensitivity typical of configurable systems. However, hard processors are not immune to faults that can happen in memory elements due to external phenomena, such as radiation-induced SEUs and SETs that must be evaluated as well to ensure overall reliable system-on-chips.

7.2 Evaluating Reliability of Hard Processors

7.2.1 State of the Art of Reliability Analysis of Hard Processors

Microprocessors are an appealing platform for high-performance safety-critical applications. They allow simplified development and provide high-flexibility thanks to the software programmability feature. Embedded Commercial-off-the-Shelf (COTS), such as reconfigurable system-on-chip, includes hardwired multi-core microprocessors as an integrated system of modern SoC devices. Combining together hardware and software programmability, these systems can reach even more high-performance goals with moderate costs.

However, for mission-critical applications, radiation-induced errors, such as single event effects (SEEs), can be a significant reliability issue, primarily if they must operate in a radiation environment. Radiation testing is the most accurate technique for evaluating the reliability of such a complex system as a microprocessor. The application under evaluation running on the actual hardware system while exposed to a flux of particles can provide important insight into the system's robustness. However, this methodology is often limited by other factors, such as facility availability, costs, and expertise. Even if it is a usually mandatory step for a mission-critical application, using less demanding techniques such as fault injection or static analysis has its advantages.

The moderate costs, easy setup, as well as the possibility to apply them earlier in the development flow, make analysis based on fault injection. emulation or simulation is mandatory as well for preparing the system for a radiation test experiment. Additionally, full control in terms of injection time and fault location provided by these approaches is valuable for preliminary and mitigation analysis.

Many works proposed reliability analyses of microprocessors based on emulation platforms [38–44]. They focus mainly on the SEU fault model or microarchitectural errors, ignoring more complex fault models such as MBUs or burst events already introduced in 6.2. Additionally, these analyses can be very time-demanding if based on an architectural emulator or simulation. Thus, it is important to propose methodologies that can produce fast reliability analysis while assuring support to commonly observed fault models, such as instance, deriving by static radiation testing analyses of the system under evaluation, but that can also be easily integrated into the development flow.

In [45], we proposed a fault injection environment for analyzing the impact of errors on the functionality of an ARM Cortex-A9 microprocessor. The environment is capable of emulating radiation-induced effects within the SoC targeting memory resources, such as main memory and registers, during the execution of software applications. Unlike most other platforms presented in the literature, it is not limited to evaluating SEU sensitivity but supports more complex fault models, too, such as MBU or word clearing. Such a solution does not require any modification of the software application under evaluation and can run directly on the actual system under test without additional effort or equipment. Please notice that the platform does not operate at the software level, for instance, manipulating variable value, but emulates hardware-induced errors directly on the hardware resource, such as, for instance, in the register content.

7.2.2 Microprocessor Fault Injection Platform

We have developed a fault injection environment based on Python, which is designed to operate within the operating system of the target hardware platform without additional modifications to the application or to the platform. This environment is capable of emulating radiation effects in memories and classifying the effect of the failure by analyzing the behavior of the application and the exceptions generated at the operating system level. The environment is also able to provide the execution status of the application when the fault has been injected, as well as when a failure, such as halting or exception, occurs, thereby allowing us to investigate the cause of the application failure.

The platform is based on two different approaches targeting either the running process or the processor system registers. The platform has been developed in Python, and the reported evaluation flows, as well as the fault injection framework and the application, execute on the target system, that in our implementation, is an ARM Cortex processor system of a Zynq 7020 SoC with a Linux operating system.

Single Event Effects (SEE) can affect the memory resource of the processing system, where the binary of the application, as well as data, is stored. The workflow depicted in Figure 7.1 takes a golden binary file, i.e. the non-faulty executable binary of the software application as an input, and then performs fault injection on the system memory or binary executable. Through this injection, a random bit is selected and modified for emulating the different SEE effects. The injection is repeated based on the injection number selected by the user, thus generating a set of outcomes.

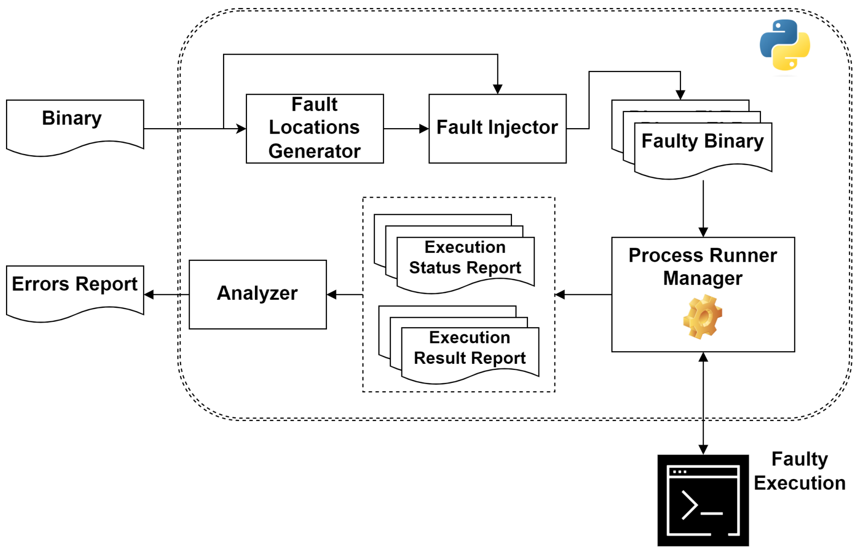


Figure 7.1 Fault Injection Platform for SEE in processor memory

To further enhance the efficiency of the python tool, multi-processing features have been implemented, dedicating each process to generating and executing the faulty process. Additionally, the execution status and exit code of the process are collected, as well as the output values, and classified by the main process in order to observe the effect of SEU on the memory. The outputs generated by executing the faulty processes are evaluated by comparison with the golden result to provide a more detailed investigation of the impact of SEE on the output of the application.

The second analysis workflow is dedicated to emulating the radiation effects within the processor registers. The fault injection environment can mimic the Single Event Effect (SEE) effect during the runtime of the application on the Processing System. This flow, depicted in Figure 7.2 is based on the GNU Project Debugger (GDB) tool, integrated and instrumented within the fault injection environment.

The methodology emulates the SEE effect on the software-accessible registers during the runtime of the application. To do so, a register among software-accessible registers is selected randomly together with a bit among the bits of the selected registers. Our target platform is based on ARMv7 architecture, therefore, the software-accessible registers include the floating-point status and control register (FPCR), the 64 NEON technology registers, the general-purpose registers (R1 to R13), and the three special-purpose registers, i.e. program counter (PC), link register (LR) and stack pointer (SP).

Parallel to fault location selection, the fault injection tool computes a random injection instruction. The injection node represents the injection time. It means the fault will be injected before executing the instruction. If a node is executed multiple times, the injection time includes an additional parameter that identifies after how many times that instruction has been executed the fault will need to be emulated. The information on how many times an instruction is executed in the golden run is extracted during the golden run steps that are executed as the first step from the platforms for generating the application's golden output. Specific instruction is selected in two steps: firstly, an instruction of the application is selected in the source code. Secondly, exploiting GDB features, a random assembly instruction offset starting from the selected LOC baseline is chosen. Please note that this methodology overcomes the limitation of time-based breakpoint selections for applications with short execution times.

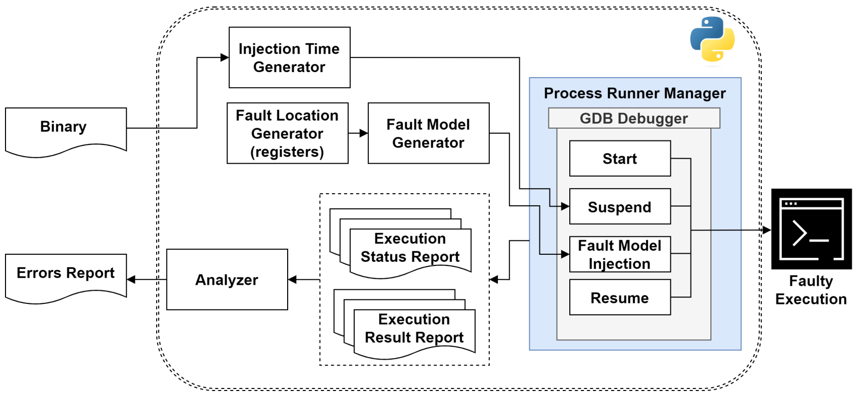


Figure 7.2 Fault Injection Platform for SEE in processor registers

The injection phase is carried out by exploiting the GDB, an open-source debugger that uses low-level system calls to monitor and modify the value of the core resources as well as to control the application execution flow. Specifically, GDB allows the platform to start the execution of the application with the golden binary, interrupting the execution when it reaches the randomly chosen breakpoint node, injecting bitflip by modifying the value of the selected bit of the selected register and resuming the execution of the application until the termination.

7.2.3 Fault Models

A collection of fault models was chosen to be incorporated into the fault injection platform, based on radiation test results, relevant literature, and radiation analysis.

Single Event Upset is very common among radiation-induced effects and thus one of the most analyzed. It is a modification of a memory cell. In our work, we considered that they can happen either in registers or main memory. Obviously, other elements exist in the processor system that can be affected by SEUs, such as the flip-flop of control circuitry. However, they are hard to target also for simulation or microarchitectural simulation environments. Indeed, often the details of microprocessor architecture are not available, so their contribution needs to be evaluated later in the development flow by radiation testing. Nevertheless, they could manifest with different effects on the registers and main memory content. For example, they could produce the clearing of registers. These other fault models can be emulated in order to obtain a more comprehensive analysis of the application's reliability.

Single Event Multiple Upset is the second fault model we considered. With the scaling down of the size of transistors, the distance between adjacent memory cells is significantly reduced. This means that multiple sensitive junctions can be present in close proximity. As a result, a single incident particle can affect multiple active regions leading to multiple upsets by a single event. Radiation tests have shown that this phenomenon can occur in different memory cells as well. We incorporated in our platform the characteristics of this fault model as deriving from a radiation test on a Xilinx Kintex-7 SRAM-based FPGA, that is based on the same technological process of the platform under test. Detail on the heavy-ion radiation experiment, that has been performed in the CERN facility, is presented in [46]. Similar events have also been reported in previous radiation tests with lower energy [47][48]. The most common shapes and sizes of bit clusters are depicted in Figure 7.3, while information on their cross-sections is exposed in Figure 7.4.

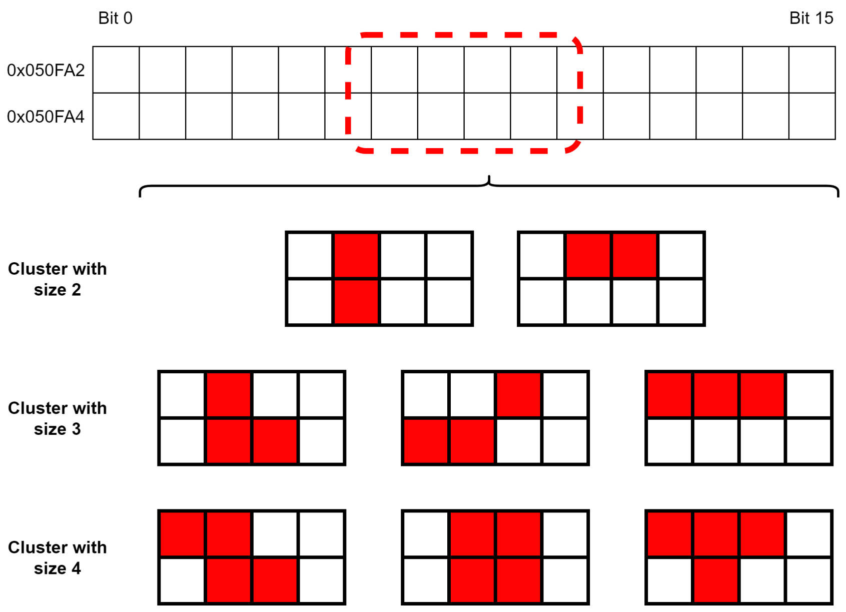


Figure 7.3 Most common shapes of SEMU observed during heavy-ion irradiation

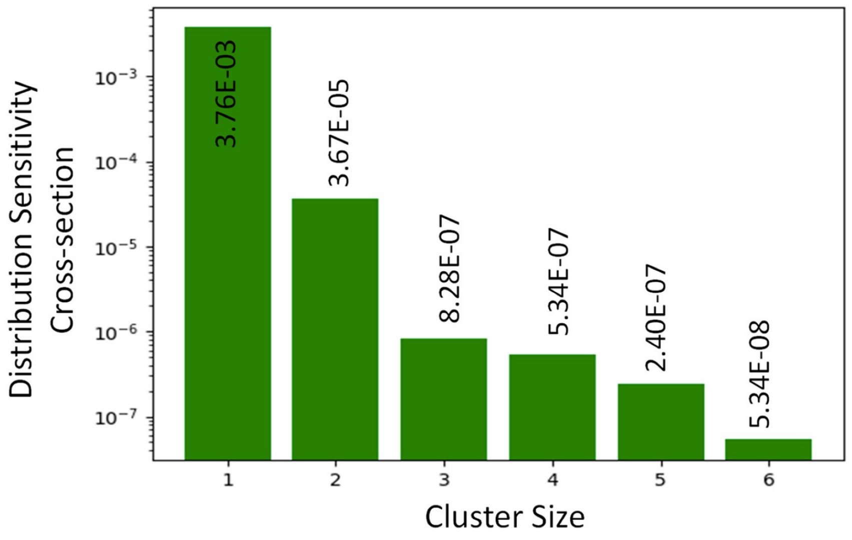


Figure 7.4 Cross-Section for different clusters of radiation-induced bit flip.

Additionally, a clear/preset fault model has been considered. The occurrence of a particle strike can lead to a situation in which the content of one or more memory locations, or the content of a register, is set to 0, known as clear content in memory resources or in a register, respectively. This effect is usually due to a failure in the control logic resources, which is a significant part of the memory responsible for managing and decoding signals from the processor. Additionally, a single event transient (SET) can be produced as a result of a particle incident in the clock signal of the memory element, causing a clock glitch and resulting in an erroneous data value or clear content in memory.

The fault models listed in this subsection are considered to may happen either in the register of the processor or main memory.

7.2.4 Software Benchmark Reliability Evaluation

A reliability analysis based on the developed evaluation flow has been conducted in order to validate the proposed platform and evaluate some common outcome resulting from radiation-induced faults for benchmark software running on an embedded processor. Three software applications have been selected from MiBench benchmark suite. An ARM Cortex-A9 Processor embedded within the Zynq-7020 AP-SoC has been chosen as the hardware platform under test. The three chosen software are the basicmath, bitcount, and FFT applications. The characteristics of the chosen benchmarks are as follows: basicmath performs basic mathematical operations, such as cubic functions solving and angles conversions; bitcount is dedicated to bit manipulation; FFT performs Fast Fourier Transforms, composed of pseudorandom sinusoidal components with variable amplitude and frequency. Four groups of instructions have been identified in each software: control (unconditional and conditional branch), integer, floating-point, and memory (load and store). An overview of the type of instruction used by each software is reported in Figure 7.5

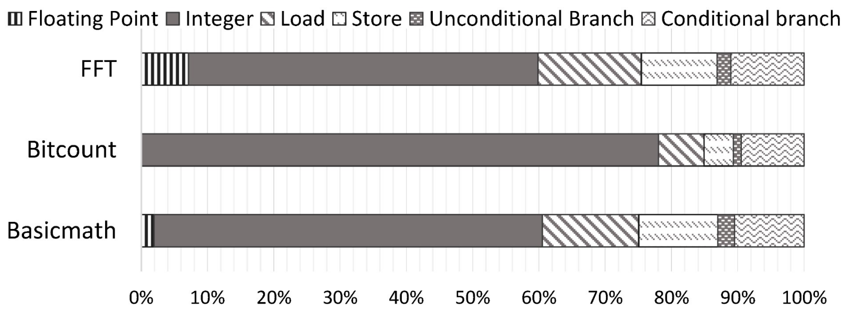


Figure 7.5 Overview of the type of the instruction characterizing the evaluated software applications.

The software is evaluated while running on the target hardware platform, a dual-core ARM Cortex-A9 Processor where Ubuntu 18.04, based on version 5.4 of Linux Kernel, an operating system is running. Both the evaluation platform and the target software execute on the device.

The fault injection platform is responsible for monitoring each process running the application under evaluation. After evaluating the application's execution, any faults that have been injected are classified into various groups according to the software outcome. If the fault does not disrupt the normal operation of the software and the program finishes without any exceptions, and the result matches the expected one, the fault is considered masked. Differently, if the program finishes without any exceptions but the output does not match the expected result, it is labeled as Silent Data Corruption. If the fault model that has been implemented causes the operating system to throw exceptions when the program is running, the program will be terminated before it can finish and the termination status will reflect the exceptions that were generated by the OS and used for classifying the outcome.

After conducting a thorough investigation into the causes of each raised exception, we classified them as follows:

- Segmentation Fault: it occurs when the injected fault modifies the address provided to the instruction operand or modifies the content of registers such as PC and SP or a memory location storing an address.
- Illegal Instruction: it is the result of the injected fault corrupting the OPCODE of instructions;
- Bus Error: it is the result of the process executing the application requesting to access a memory location that is not physically accessible;
- Abort: it occurs when the injected fault leads to the interruption of the execution of the application, not due to an exception risen by the operating system but due to an exception risen by the process executing it;
- Breakpoint Trap; it is the result of the injected fault causing the process to enter the debug state and execute the breakpoint hook instructions;
- Arithmetic Operation Error: it occurs when the injected fault corrupts one of the operands, resulting in the process trying to execute an arithmetic operation that rise an exception, for instance, dividing by zero;

- Hang: as a result of the injected fault the execution of the application does not terminate, for instance, due to an infinite loop or a deadlock situation.

In order to investigate the radiation sensitivity of the selected applications, we conducted different fault injection campaigns using the proposed platform. The campaigns consist of 10,000 fault injections, targeting both the memory and registers of the processing system. Evaluated fault models are SEU, SEMU, clear content (i.e., all the bits of a word are set to 0), and preset content (i.e., all the bits of a word are set to 1).

Figure 7.6 reports the error rates resulting from the fault injection campaigns, while Figures 7.7 and Figure 7.8 report the occurrence of each termination status and risen exception in the total error rate, dedicated to the faults occurring in memory and to the faults happening in registers, respectively. As can be observed, most of the errors are SDC, Using the proposed platform, for performing 10,000 fault injections on memory resources, approximately 10 minutes is required, while this value increases to three hours for the fault injection campaign in register resources.

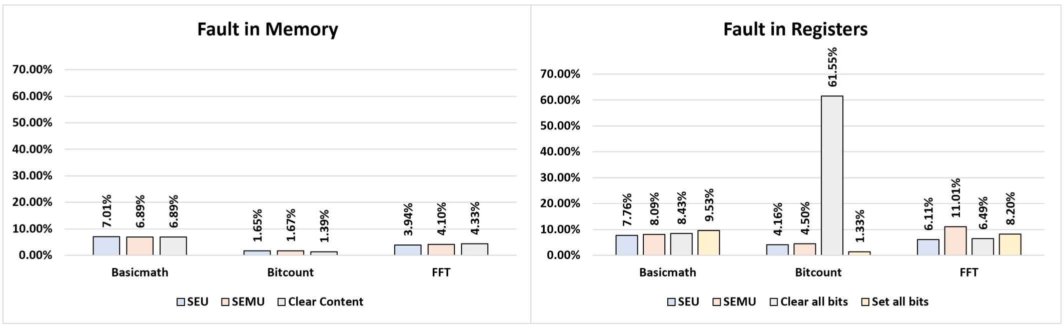


Figure 7.6 Error Rates resulting from fault injection campaign on ARM A9 embedded processor.

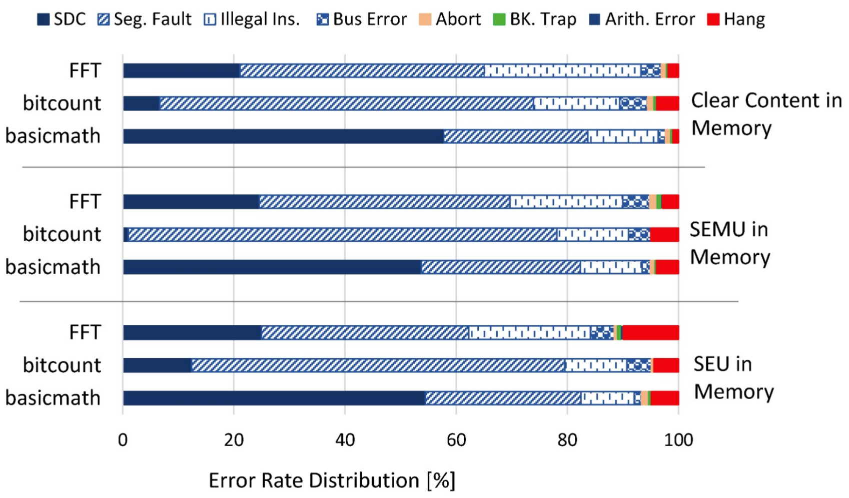


Figure 7.7 Overview of the type of the instruction characterizing the evaluated software applications.

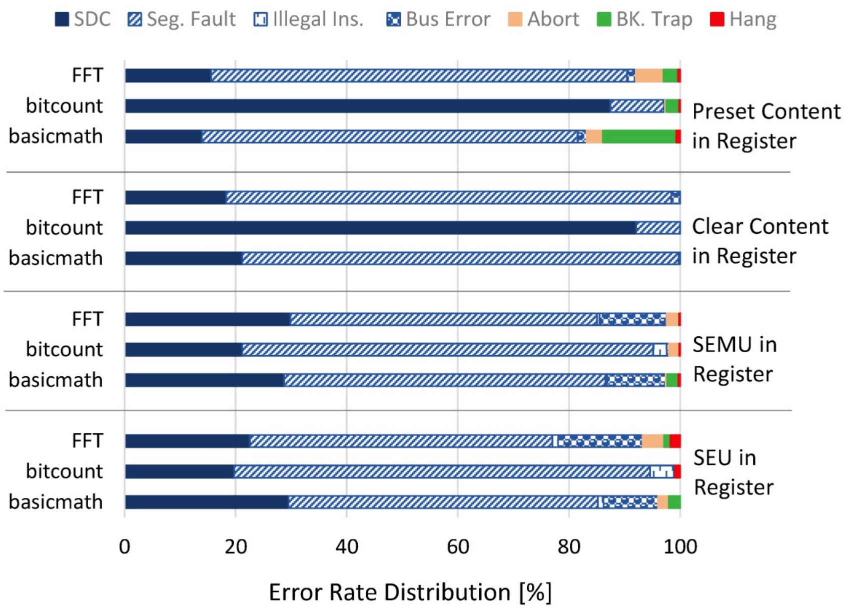


Figure 7.8 Overview of the type of the instruction characterizing the evaluated software applications.

It is important to note different applications are characterized by different assembly instructions, leading to different levels of sensitivity to faults that may occur during execution, accordingly to used registers, memory footprint, and other factors. For instance, if an application is more focused on bit manipulation instructions, then it is more likely to be vulnerable to register injection, as registers are often used throughout the execution and a single single-event upset (SEU) in the registers can have a serious impact on the output. Additionally, this can be used to compare the fault tolerance of different implementations of the same application, by examining how different instructions and data structures can affect the system's resilience to faults.

7.3 Evaluating Software Reliability in Soft Processors

7.3.1 State of the Art of Reliability Analysis of Soft Processors

A soft microprocessor is one of the cores frequently implemented using programmable hardware to support hardware acceleration with software flexibility. Even when the SoC is already equipped with a hard microprocessor, a soft processor can be useful for offloading the computation demands from the hard processor, or for redundancy. Among the available solutions, RISC-V soft microprocessors have become increasingly popular due to their open license and wide community support.

For mission-critical applications that use a soft microprocessor, Single Event Upsets must be taken into account as a possible source of malfunctions. Many strategies have been proposed to improve reliability against SEUs, the most reliable being hardware redundancy. Nevertheless, this approach is expensive in terms of design time, area overhead, and power consumption. Approaches based on software, however, are easier to apply and less demanding. These strategies involve replicating data in memory, executing redundant code, and verifying consistency during execution. Software approaches are not as efficient as hardware redundancy, but they provide a cost-effective way to increase reliability against SEUs.

Soft processors implemented in programmable hardware have an additional vulnerability compared to hard microprocessors due to the configuration memory, which stores the data that programs the circuit implemented on programmable hardware. Configuration memory can be corrupted by Single Event Upsets like the processor memory, but resulting in hardware faults. Software approaches based on replication could be less effective on soft processors than on hard processors due to the architectural faults that are much less common in fixed-hardware processors.

In [49], we proposed a comprehensive evaluation of the impact of soft errors, and in particular SEU in the memories of a RISC-V soft microprocessor implemented in programmable hardware. The experiments were designed to evaluate the effectiveness of hardening-by-replication software techniques against Single Event Upsets in the CRAM and processor memory. The fault injection campaigns revealed that while these techniques marginally improved the robustness against Single Event Upsets targeting processor memory, they had a much more significant impact against hardware faults.

In recent years, a variety of hardware and software techniques have been explored in order to meet the high-reliability requirements of critical systems. One of the first approaches proposed to enhance the fault tolerance of these systems was software hardening-by-replication, which has been found to be less effective than the more complicated hardware techniques like TMR. Despite this, its simple implementation has made it popular; for example, NASA advised the use of software techniques to improve the fault tolerance of mission-critical applications as far back as the 2000s [50].

Previous research has addressed the resilience of software and hardware platforms. In [32], a strategy for identifying soft errors in code and data was suggested, which employed a software replication technique. [33] . Additionally, [34] presented a software method that provided both detection and correction based on data and code replication. [35] evaluated hardware and software techniques through fault injection campaigns against SEUs that affected microprocessors. FPGAs were included in the analysis, though the authors only evaluated faults that impacted storage elements, without taking into account configuration memory elements or other components.

Previous studies have considered the robustness of software and hardware platforms. In [51], a strategy for identifying soft errors in code and data was suggested, which employed a software replication technique. [52] proposed SWIFT, proposed SWIFT, a software fault detection approach based on exploiting instruction-level parallelism through unused instructions. Additionally, [53] presented a software method that provided both detection and correction based on data and code replication. In [54], authors evaluated hardware and software techniques through fault injection campaigns against SEUs that affected microprocessors. FPGAs were included in the analysis, though the authors only evaluated faults that impacted storage elements, without taking into account configuration memory elements or other components.

Software hardening-by-replication is based on the redundancy of data and/or computations. To protect against SEUs, input data may be replicated in the memory and the software may be written so that computations are run multiple times using the same input data (or replicated copies if they are available in memory) of the same inputs. Depending on the desired level of granularity and acceptable overhead, a number of detection and correction checkpoints are inserted into the code. At these checkpoints, the values of the temporary results are compared to one another in order to detect errors in the computations; any erroneous values are then corrected through majority voting either during the checkpoint or on the final output. An overview of this approach is shown in Figure 7.9.

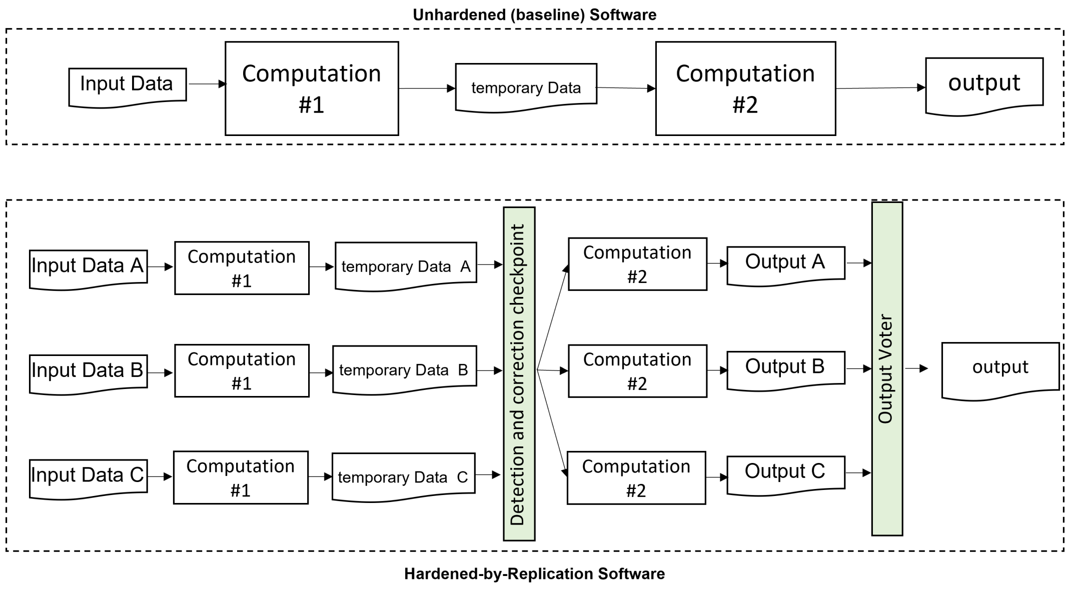


Figure 7.9 Schematic view of a baseline software and its hardened version.

7.3.2 Soft Processor Reliability Analysis Flow

A RISC-V soft processor is being used in the experiment as the basis for reliability evaluations. In order to do this, a set of software applications have been chosen as the benchmark suite. These applications have both an unhardened, or baseline, and a hardened-by-replication version. To carry out the reliability analyses, a fault injection platform has been used. The fault models are being simulated in the CRAM and in the memory of the processor while the software runs on the platform.

Table 7.1 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%)

Hardware Platform

In order to assess reliability, we chose PULPissimo as the microcontroller architecture for our analysis. This single-core platform is the product of a joint venture between ETH Zurich and the University of Bologna known as the PULP project [55]. As the programmable hardware device, we selected a Nexys Video Artix-7. The device utilization in implementing PULPissimo is reported in Table 7.1.

Software Benchmark Applications

Four software applications have been adopted as benchmarks. The applications cover different domains, such as signal and image processing. These are:

- CoreMark: it involves list processing, matrix manipulation, state machine execution, and cyclic redundancy check.
- Dhrystone: it is a performance benchmark focusing on string processing, without the use of any floating-point operation.
- FFT: it implements the Fast Fourier Transform, a widely used technique in signal processing.
- Sobel: it implements the Sobel operator, used for edge detection in image processing.

Both FFT and Sobel are part of the MiBench Benchmark Suite [56].

To evaluate the effectiveness of the hardened software, a hardened version of every software program has been produced. According to [50], we applied single-version software fault tolerance techniques. This required replicating the input data, variables, and functions in the memory, changing the software to perform the repeating of the same operations on the different data copies stored in the memory, and performing detection and correction checkpoints.

Fault Models

Soft microprocessors are particularly susceptible to Single Event Upsets (SEUs) due to their architecture and technology. SEUs manifest as bit flips in the memory content and can affect both the processor memory and the CRAM. This can lead to a range of failures and errors. We evaluated the dependability of applications running on soft processors with regard to SEUs in the memory of the processor and in the CRAM. An SEU in the processor memory can corrupt the data or code segments of the program in that area of memory, which could result in data value corruption or a system crash. Differently, SEUs in configuration memory cause faults in the hardware architecture of the soft processors. For instance, if the ALU is affected, it can lead to errors in the calculations performed by the software applications.

Evaluation Methodology and Platform

Two different fault injection platforms have been adopted for emulating single-event upsets (SEUs) in the main memory and configuration memory of a microprocessor.

A python-based fault injection platform has been developed to automatically emulate SEU in the main memory of the microprocessors by acting directly on the Executable and Linking Format (ELF). This platform performs the fault injection process in the main memory emulating the effect of SEUS, loads the binary code into the memory of the PULPissimo microcontroller, and collects the output of the injected applications. To do this, it is integrated with Open On-Chip Debugger (OpenOCD) and GDB, which communicate with the RISC-V debug module via the JTAG interface. The platform also has a timeout mechanism to handle the halting of the processor due to injected faults and will wait for the results of the software computation on the serial port.

The PyXEL platform has been utilized to emulate SEUs in the configuration memory of the Artix-7 XC7A200T FPGA. This python-based platform is capable of manipulating FPGA bitstreams for fault injection campaigns. By corrupting a bit of the bitstream to be loaded in the configuration memory, PyXEL can efficiently emulate SEUs. Moreover, it automates the steps for configuring the FPGA platform with the bitstream implementing the RISC-V soft microprocessor, resetting the platform in the event of a halt due to fault injection, loading and running software applications on the soft microprocessors, and collecting the results.

7.3.3 SEUs Evaluation of Hardened Software on Soft Processor

We conducted dedicated reliability analyses to assess the advantages and disadvantages of utilizing hardening-by-replication software techniques for applications running on a soft microprocessor. To this end, we evaluated the baseline and hardened software benchmark applications against SEUs both in the soft microprocessor main memory and in the hardware-configurable platform configuration memory. The software benchmarks were run as bare-metal, without any operating system.

The results of the various fault injection experiments have been categorized accordingly with the following categories:

- Correct: The task terminates correctly and the output matches the golden one.
- Silent Data Corruption: The task terminates but the output does not match with the golden one.
- Halt: the soft microprocessor does not complete the task. It can be due to different causes, such as infinite loops, illegal code instructions, or others. It can be generated either by a fault in the binary code or at the hardware architecture level.

We define error rate as the proportion of results that do not conform to the expected behavior, i.e. the percentage of outcomes that are classified as Silent Data Corruption (SDC) or Halting.

We conducted a first fault injection campaign to evaluate the baseline software's reliability against SEUs in memory. For each of the four proposed software benchmarks, 10,000 experiments were executed, with SEU coordinates (i.e., the bit to flip in the binary code) chosen randomly and independently for each experiment.

During a second fault injection campaign, we conducted an analysis of the effects of an SEU in the configuration memory of the FPGA implementing the soft microprocessor. The corruption of the configuration memory content may cause errors in the hardware architecture implemented on the programmable hardware. Because only a subset of the resources is used and programmed, the error rate resulting from SEUs in configuration memory is typically low, since only a limited portion of the configuration memory bits are required for the design. Nevertheless, since these errors can permanently affect the microprocessor operation until the next reconfiguration or power cycle, they are of significant importance in reliability evaluation. This fault injection campaign consists of 10,000 faults, injected singularly and randomly in the configuration memory. Each software benchmark has been evaluated while running on each of the faulty configurations. Since different software utilizes different logic of the soft microprocessor, they will be characterized by different error rates even when running on the same faulty configurations.

A categorization of errors is reported in Table 7.2.

Table 7.2 Error categorization resulting from fault injection campaigns on baseline software.

Fault	SEU in Processor Memory			SEU in Configuration Memory		
Error	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 7.3 Error categorization resulting from fault injection campaigns on hardened software.

Fault	SEU in Processor Memory			SEU in Configuration Memory		
Error	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

A figure of merit of the error rate resulting from these campaigns to evaluate the reliability of the baseline software on the proposed platform is reported in Figure 7.10

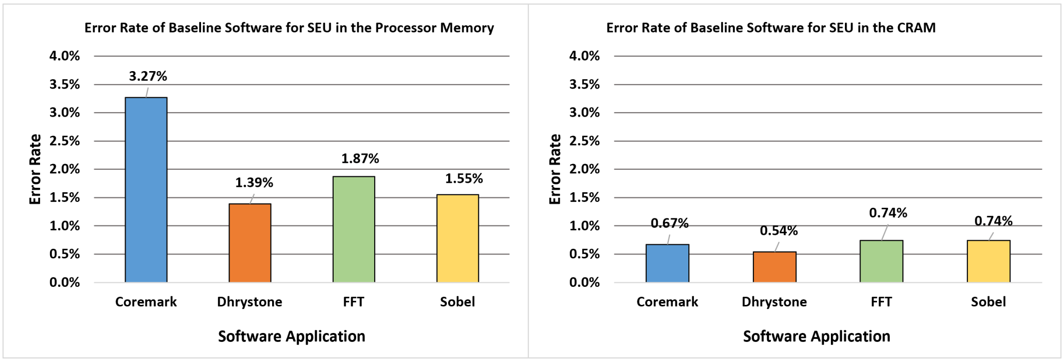


Figure 7.10 Figure of Merit of baseline software reliability against SEUs.

The following two campaigns are dedicated to evaluating the effects of the hardening technique on the software benchmarks. These campaigns are the same as the previous campaigns presented in this subsection but the software version evaluated is the hardened one. The results of these campaigns are reported in Figure 7.11, while the occurrence of different types of errors is resumed in Table 7.3.

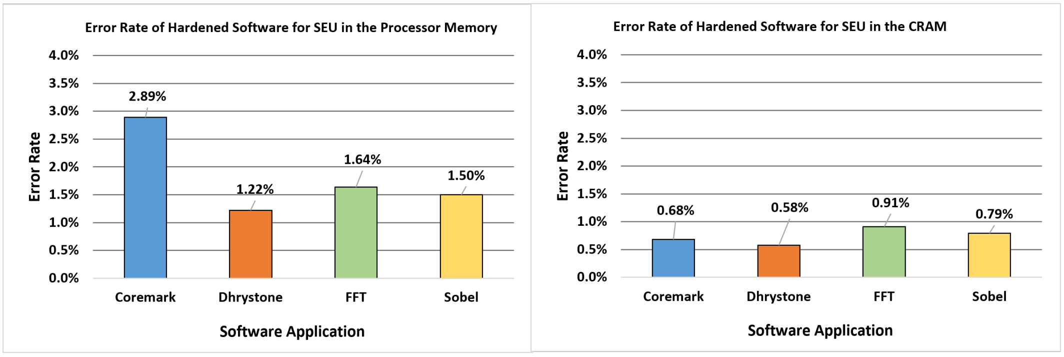


Figure 7.11 Figure of Merit of hardened software reliability against SEUs.

The comparison between the reliability of the baseline and hardened software yielded some interesting results, reported in Figure 7.12. Specifically, software hardening was found to slightly increase the reliability of all applications against SEUs affecting processor memory. However, the same trend was not observed for SEUs in configuration memory, which actually reported a decrease in reliability.

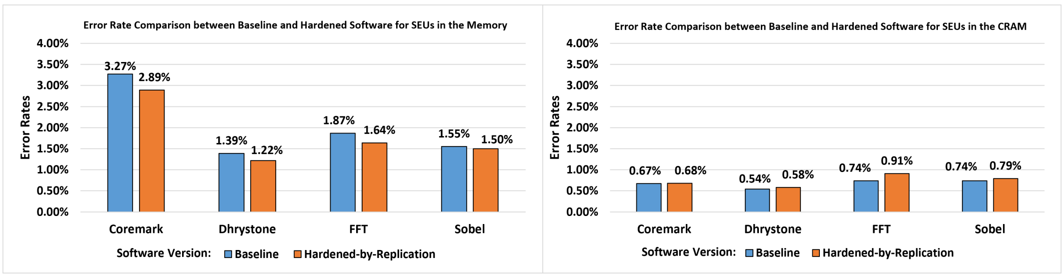


Figure 7.12 Comparison between baseline and hardened software reliability against SEUs.

We theorized two explanations for this. Firstly, software replication techniques are ineffective when errors affect hardware elements of the microprocessors, as performing the same operation twice on the same faulty hardware will likely produce the same erroneous output. There are certain exceptions to this, such as errors generated in the reading of a specific memory cell being able to be corrected by reading replicated data stored in different memory cells. Nevertheless, the mitigation benefits are still reduced in comparison to faults affecting microprocessor memory. Secondly, the introduction of the code for implementing the detection and correction checkpoint could stimulate sections of the electronic circuit that were not used by an unhardened version of the software, which could cause hardware faults that were previously not propagated to the application outputs to now produce a deviation from the software application's nominal behavior.

7.4 Reliability of Applications running in Real-Time Operating System on Soft Processors

7.4.1 State of the Art of Reliability Analysis of Real-Time Applications on Soft Processors

The interest in Real-Time Operating Systems to meet stringent real-time requirements in embedded and safety-critical systems is rising. Among the different available hardware architectures, implementing an RTOS on a soft processor embedded in a programmable device is an efficient and flexible solution for providing programmable hardware with software programmability or supporting a general-purpose processor with a real-time system. However, radiation-induced failures pose a serious threat to the reliability of electronic systems in space applications.

Microblaze is a well-known industry leader in FPGA-based soft processing solutions due to its highly flexible architecture and configuration options, making it an ideal choice for embedded applications. As the task complexity of embedded systems has increased the use of bare-metal systems may not be sufficient, thus Real-Time Operating Systems have emerged as a viable solution for meeting real-time requirements. FreeRTOS is a particularly popular RTOS choice due to its ability to execute multiple tasks in an organized and predictable fashion, with deterministic and predictable task scheduling that allows for hard real-time execution of more critical tasks.

As already reported in the previous Section, it is important to account for the reliability issues that can arise from the exposure of the softcore processors to ionizing radiation, such as SEUs. Unlike hardwired microprocessors, the netlist of soft microprocessors like Microblaze is stored in the CRAM of the FPGA. This memory can be corrupted by SEUs, resulting in micro-architectural faults in the hardware that can propagate to the application layer. Furthermore, additional complexity exists when an operating system, that adds a much more complex software layer, is used.

Studies evaluating the sensitivity of embedded operating systems to SEUs typically involve modifying the original kernel of the embedded operating system, altering the memory used by the OS, or changing the parameters of system calls. For instance, in [57], the vulnerability of FreeRTOS was assessed using a software-based fault injection method that targeted the most relevant variables and data structures of the OS. Additionally, an automatic method for fault injection into program and data memory was proposed in [58], while [59] proposed a detailed analysis and hardening architecture based on lockstep synchronization for FreeRTOS. Nevertheless, these software application-level methods do not take into account the effect of faults occurring at the architectural level of the soft-core processor on the functionality of the operating system.

On the other hand, works such as [60] and [61] focused on the reliability evaluation of soft cores, particularly RISC-V cores, implemented in programmable hardware against radiation effects. However, these analyses do not consider real-time operating systems such as FreeRTOS. Other approaches are based on the simulation of the HDL description of microprocessors [62]. These methods are advantageous in that they allow the injection of upsets into any memory element at any time; however, they are very time-consuming. As for Single Event Multiple Upset (SEMU), which is one of the fault models on which our analysis places particular attention, there is limited literature available, as most of the research, especially works focusing on reconfigurable hardware reliability, is dedicated to SEU evaluation, as it is still the main source of errors for less recent technologies. Nevertheless, due to technology scaling and lower operating voltages, SEMUs are becoming more prominent, partly due to their danger to redundant systems [36][63].

In order to provide a comprehensive evaluation of the effects of radiation-induced architectural faults on the performance of applications running on the Microblaze soft processor with FreeRTOS, we carried out an analysis in [64], and further in [65], considering MBUs and RTOS. The fault model includes different clusters of Multiple Bit Upset (MBU) identified through proton radiation at the Paul Scherrer Institute (PSI) radiation facility. The analysis focused on the impact of these faults on the execution of different software applications considering the full stack programmable hardware, FreeRTOS, and software application. It is important to note that this platform is not designed to target faults at the software level, but rather to focus on hardware faults and their effects on the software running in the operating system. Specifically, the platform emulates SEUs and MBUs by introducing bitflips into the configuration memory of the FPGA, thus creating a hardware-level fault that can propagate to the system's output and result in incorrect behavior.

7.4.2 Multiple Bit Upset Fault Model

A proton radiation test was conducted at the Paul Scherrer Institute (PSI) Proton Facility in Switzerland. A Zynq-7020 device was exposed to proton beams of varying energies, ranging from 29 to 200 MeV. Testing parameters are reported in Table 7.4.

A host computer installed in the control room and connected to the system under test via a serial connection was dedicated to acquiring the content of the configuration memory by readback. The monitoring system was designed to autonomously send a soft reset signal to the device in the irradiation room if the readback system stopped working; however, if this was unsuccessful, a power switch was available in the control room to perform a manual power cycle if needed. The experiment was conducted for 8 hours in December 2022 at the PSI proton facility. To monitor the

Table 7.4 Proton Test Conditions: Energy, Flux and Fluence

Energy [MeV]	Flux [$\text{cm}^{-2}\,\text{s}^{-1}$]	Fluence [cm^{-2}]
29.31	4.12×10^{7}	9.17×10^{10}
50.80	4.02×10^{7}	6.06×10^{10}
69.71	4.11×10^{7}	2.12×10^{10}
101.34	4.32×10^{7}	2.42×10^{10}
151.18	4.09×10^{7}	1.23×10^{10}
200.00	4.14×10^{7}	3.94×10^{10}

configuration memory of the device, a periodic reading of the content was conducted every about 4 seconds. The flux of the particles was adjusted to maintain a few bitflips in the configuration memory during each snapshot in order to identify easily multiple bit upsets.

By analyzing the data of the configuration memory content, we were able to detect the occurrence of Multiple Bit Upsets (MBUs). Due to the large size of the configuration memory (more than 10^8 bits) and the ability to observe snapshots of the configuration memory data with higher frequency, it was feasible to evaluate a cluster of bits with a high likelihood of having been caused by a Single Event Multiple Upsets (SEMUs). The most common cluster shapes are depicted in Figure 7.13 for different cluster sizes.

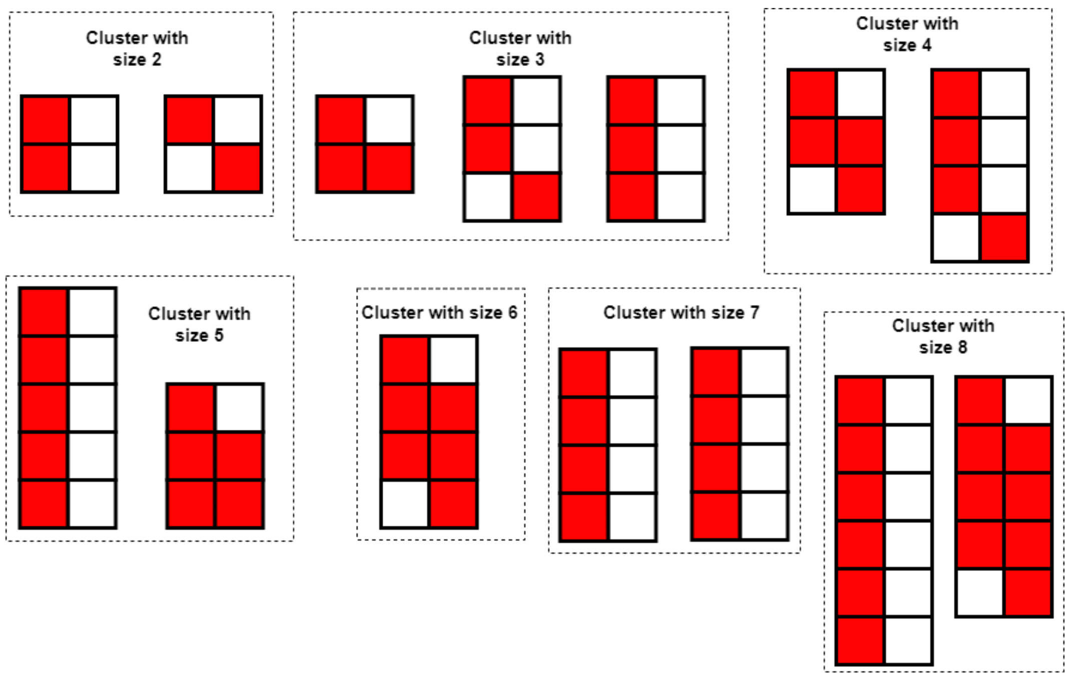


Figure 7.13 Shapes of MBU cluster detected in Zynq7020 configuration memory during proton test

As expected, larger clusters occurred rarely. Figure 7.14 illustrates the cross-section per particle for each cluster size, calculated by dividing the number of clusters of a certain size by the total number of particles that have passed through the device over the course of the entire test. It is evident that, although SEUs are the most common effect, MBUs still represent a significant portion of the observed events, accounting for more than 40% of the detected radiation-induced errors. This suggests that MBUs could have a significant influence on the overall sensitivity of the system to radiation-induced errors. Additionally, MBUs can be a threatening to systems that relies on mitigation techniques combining TMR and scrubbing, for avoiding multiple domain failures.

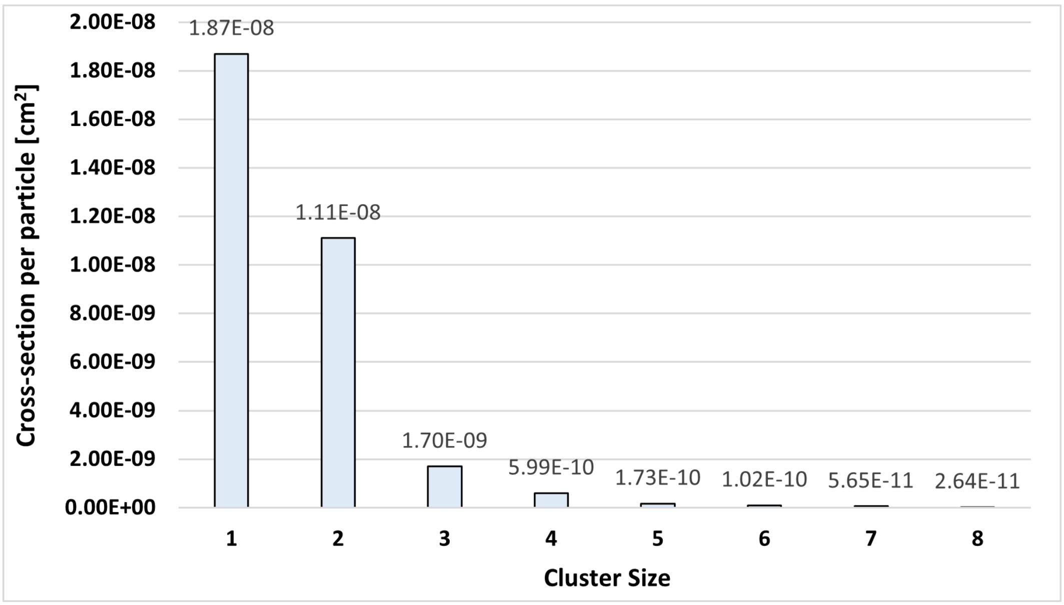


Figure 7.14 Cross-Sections for MBU clusters detected in Zynq7020 configuration memory during proton test

7.4.3 Soft Processor under Test: Device, Hardware, Operating System and Application

The Xilinx 28 nm CMOS Zynq-7020 FPGA was chosen as the target hardware device to port the Microblaze embedded soft processor. This processor is highly configurable, allowing for the selection of a specific set of features required by the design, and is thus suitable for supporting FreeRTOS, a Real-time operating system that enables concurrency among several tasks with different priority levels and supports a preemption mechanism. Additionally, the system is implemented with an exception handler to cope with the standard exception conditions defined by the Microblaze soft-core. A overview of the elements composing the hardware platform used in the reliability analysis is depicted in Figure 7.15, while Table 7.5 reports the used resources in the programmable hardware.

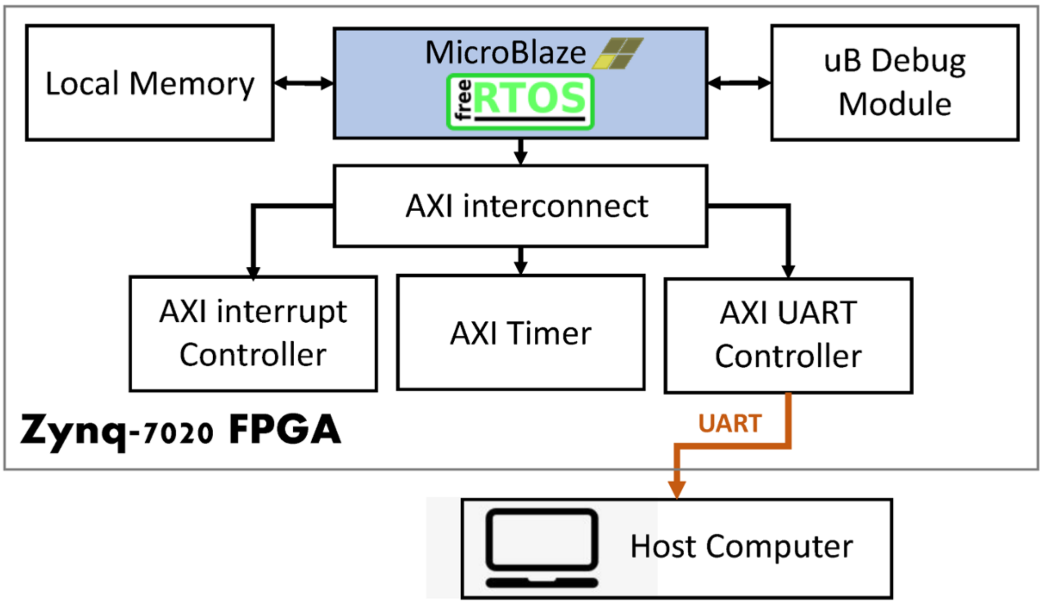


Figure 7.15 Hardware Platform under Evaluation

A set of software applications have been selected to be evaluated, each of which is composed of three tasks with the same priority that run in the FreeRTOs. Each of these tasks is running the same program code, but with different input data, and producing different results. This means that the three tasks share the same binary code in their instruction memory, but are operating on distinct input data, and are sharing the processor's execution time. The three applications are:

- matmul: multiple matrix multiplications between a series of matrices of integers.
- matconv: convolution between large matrices of integers and a different kernels.
- dijkstra: finding of shortest path in graph using the Dijkstra algorithm.

7.4.4 Reliability Analysis for Soft Processor running RTOS

Reliability analysis involved the reported fault model and platform under test. Reliability evaluation is conducted through fault injection campaigns. To support this

Table 7.5 Programmable-hardware utilization of the Microblaze Soft Processor implemented on Zynq-7020

Resources	**Available**	**Used**
Logic Slices	13,300	966 (7.26%)
LUTs	53,200	2,596 (4.88%)
Flip-Flops	106,400	2,668 (2.51%)
BRAMs	140	32 (22.86%)

campaign, the PyXEL platform was instrumented to inject MBU patterns identified from the proton radiation test into the configuration memory of the FPGA. The fault injection has been conducted targeting only the tiles used by the hardware implemented on the programmable logic. This allowed to speed up the evaluation time. Indeed, as reported in Table 7.5, the utilization of the hardware is low and a random fault injection could require a much higher number of faults to obtain statistically significant results. As a reference, a 10,000-fault injection campaign requires approximately 20 hours. To further speed up the experiments, we used multiple fault injection platforms in parallel.

The PyXEL experiment manager, which is running on the host computer, was connected to the device under test via a serial connection. This connection is used to manage experiments on the device, configuring and starting the platform, and collect results. In the event of the processor halting or entering an endless loop due to the injected faults, a timeout mechanism is used to manage the situation.

The fault injection platform is designed to emulate the effects of MBUs by altering the content of the configuration memory. Through the JTAG interface, the faulty configuration data are loaded into the configuration memory. The benchmark software utilizes the serial connection to transmit the results of the computation, which are then logged by the experiment manager alongside the injected fault model and fault location.

The outcomes of the experiments are classified in the following categories:

- Correct: application produced an output that matches the golden one.
- Silent Data Corruption: the task terminates succesfully but the produced output data does not match the golden one.

- Halt: the task cannot complete and the system is unresponsive. It can be due to different causes, such as infinite loops and application timeout.
- Exceptions: an exception in the FreeRTOS (i.e., at the software level) is detected. Since fault injection targets only reconfigurable hardware, it is resulting from a fault affecting Microblaze architecture (i.e., netlist modification due to configuration memory corruption).

Moreover, exceptions are further evaluated and classified as follows:

- FSL_EXCEPTION: an error on the data bus generated the exception.
- UNALIGNED_ACCESS: systems tried to perform access to memory with unsupported unaligned access.
- ILLEGAL_OPCODE: fault caused the system to try to execute an illegal opcode.
- AXI_D_EXCEPTION: data system bus timeout occurred.

Two fault injection experiments have been conducted. The first was conducted based on the distribution of SEUs and MBUs shown in Figure 7.14. The second was dedicated to testing each group of clusters in order to assess the vulnerability and the effect on the application. As discussed above, only a portion of the total configuration memory of the circuit under test was targeted by the fault injection process, in order to reduce the injection area to the resources utilized by the implemented netlist.

We conducted 10,000 fault injections in the first campaign, using the cluster distribution. Figure 7.16 shows the resulting error rate, which is defined as the percentage of results that deviate from the nominal behavior, for each cluster.

The results revealed that the three applications were affected in slight different way by the faults. Specifically, matconv registered the highest number of errors (including all four categories) at 2,179 out of 10,000 injections, followed by matmul and dijkstra with 1028 and 1026 errors respectively.

For all the applications, the vast majority of the errors can be attributed to cluster injection of sizes 1 and 2. It is evident that the distribution of errors closely follows the distribution of the clusters, as Figure 7.14 shows.

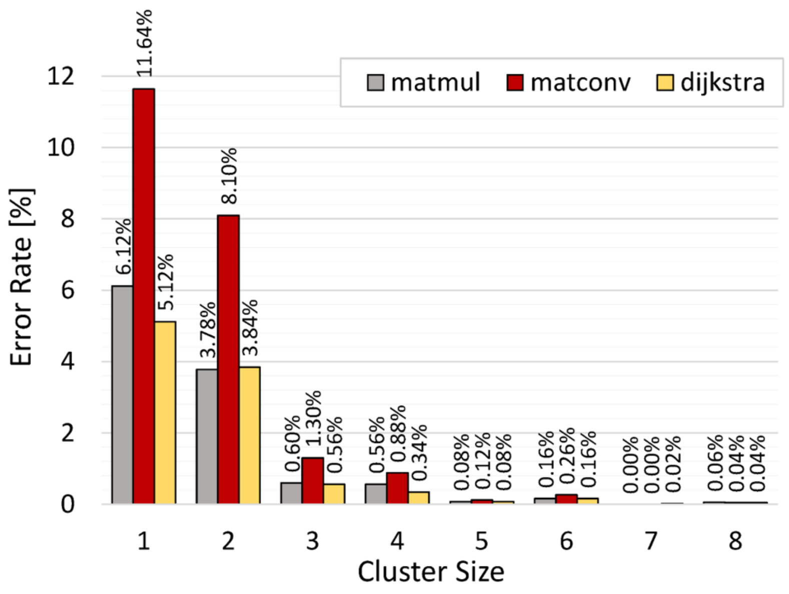


Figure 7.16 Error Rate for software application resulting from MBU fault injection weighted on cluster cross-section.

The second fault campaign has been dedicated to evaluating the effects induced by different clusters of errors. Each cluster characterization is based on a 5000-fault injection campaign, for a total of 40,000 experiments. The results are illustrated in Figure 7.17, where the outcomes have been classified. The Halt label consistently has the highest value, which is likely caused by the malfunction of the communication components in the design.

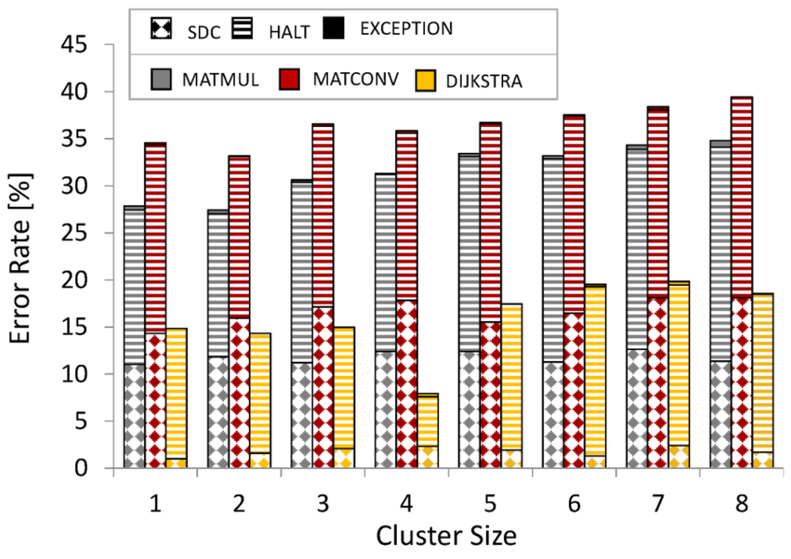


Figure 7.17 Classification of the Effects caused by Cluster with different size.

Finally, a comprehensive classification of the exceptions observed in the two fault injection campaigns was conducted, with the results shown in Figure 7.18. This chart indicates the relative frequency of each exception type. All of the exceptions encountered were related to reading from system memory, except for ILLEGAL_OPCODE, which was the least frequent of the observed errors.

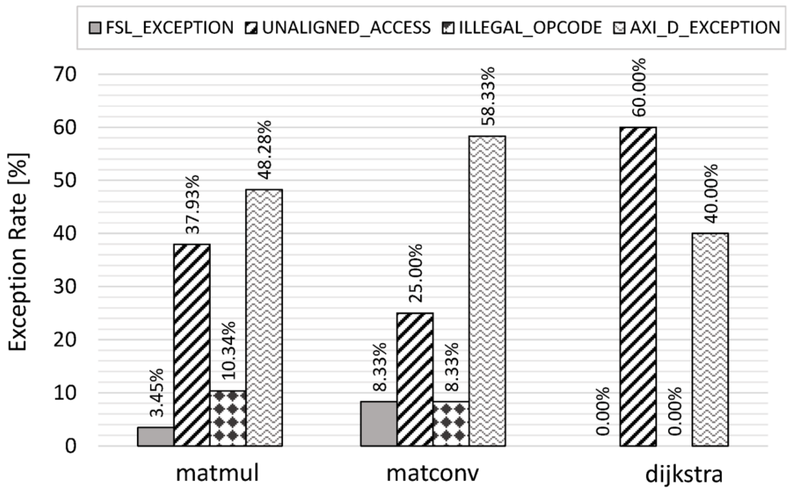


Figure 7.18 Classification of Exceptions observed during the two fault injection campaigns.

In order to gain further insight into the sensitivity of the analyzed software and platform, the mean-time-to-failure (MTTF) and the mean-executions-to-failure

(METF) have been evaluated based on the cross-sections of different cluster sizes at different energies, and the error rate of the applications against the specific clusters resulting from the second fault injection campaign. Specifically, the MTTF, which is the mean time between two faulty outcomes of the application, is reported in Figure 7.19. Additionally, the METF has been computed using the average execution time of the software applications. Generally, it is expected that as the particles move toward higher energies, the failure rate increases, leading to a smaller value for MTTF and METF.

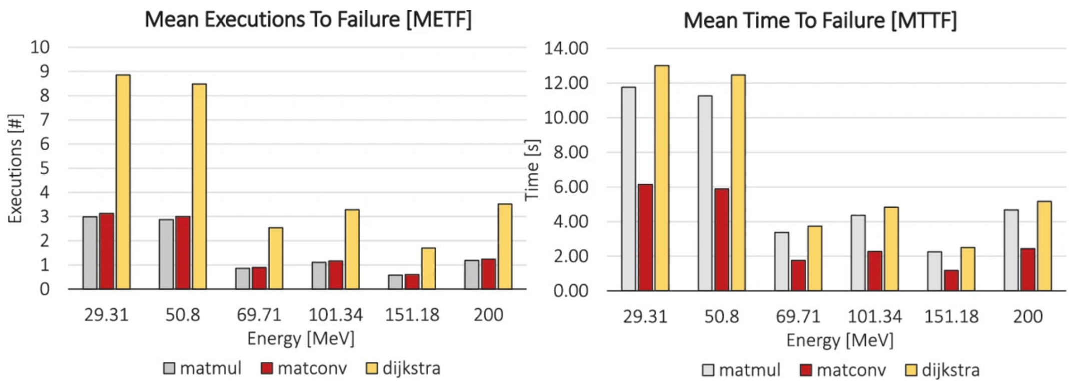


Figure 7.19 METF and MTTF for the applications based on cross-sections and error rate for different energies and fluxes reported in Table 7.4.

7.5 Research Advancements in Reliability Evaluation of Embedded Processors

Novel methodologies, techniques, and tools for analyzing the robustness of hard and soft processor systems have been proposed. The analysis involved both software-level and architectural-level faults. Proposed fault models included faults affecting software and hardware implementation. The analysis produced interesting results that highlighted the criticality of Single and Multiple Event Upsets in the configuration memory producing architectural hardware faults. Preliminary results show how software mechanisms, such as the operating system exception mechanism, can provide support in identifying hardware faults induced by configuration memory corruption.

Chapter 8

Evaluating Reliability of Host-Accelerator Interfacing

8.1 Overview on Reliability of Host-Accelerator Interfacing

Reconfigurable Systems-on-chip has become increasingly popular in various domains due to their feature to integrate a microprocessor and programmable hardware on a single chip. This integration has made it easier for these devices to meet the requirements for high-performance systems, thanks to the possibility to enhance the system with dedicated hardware acceleration implemented on the on-chip programmable hardware, making them attractive even for fields such as avionics, aerospace, and automotive. Indeed, these devices allow designers to move computationally demanding tasks to be executed by the custom hardware. However, to ensure high performance, the communication interface between the host and the accelerator must be proper to prevent it from becoming the system's main bottleneck or point of failure.

The Advanced Microcontroller Bus Architecture (AMBA) is a standard ARM developed to facilitate the interconnection of blocks in a system-on-chip. This standard supports high-performance and high-frequency communication and includes the specification for AXI4 interfaces. Xilinx has adopted AXI4, AXI4-Stream, and AXI4-Lite for IP blocks interfacing and on-chip communication. The AXI modules are essential for designs that employ programmable hardware to improve performance. This module is responsible for mediating the communication between the processor system and the modules implemented in the hardware or the access to shared memory space. Consequently, the IP Cores dedicated to processor-accelerator interfacing are crucial for solutions that use programmable logic in the high-performance domain.

Despite the benefits of the AP-SoCs architecture, when the programmable hardware of these systems is exposed to ionized particles, soft errors can affect the programmable logic of the SoC, leading to incorrect computation or even to the modification of the hardware architecture implemented in the programmable logic. This issue is a major concern for safety-critical systems that must be investigated to ensure that failure of the interfacing module will not compromise the reliability of the whole system.

8.2 Analysis of the AXI Interconnect Module

8.2.1 State of the Art on the Evaluation of Robustness of AXI Interconnection Core

Reconfigurable SoCs are integrated circuits that consist of a processor system and programmable logic which can be customized by the user. The vendor typically provides the AXI Interconnect module for the developers to enable communication between the processor system and the modules implemented on programmable hardware. This module facilitates the exchange of data between the two components, allowing the processor system to access the hardware modules and vice versa.

The AXI Interconnect module is an essential component for designs implemented in programmable logic, acting as a bridge between processor systems and modules that support AXI Interface. The Xilinx IP catalog provides the AXI Interconnect IP Core to enable the implementation of a 1-to-N communication model in programmable logic. This core allows the connection of a single master to multiple slaves, with the master being able to access the slaves transparently according to the interface characteristics of each IP block. In this way, the processor system can demand computationally expensive tasks to the hardware modules (slaves), which can perform different operations in the case of different tasks needing to be accelerated, or the same operation on different data vectors in the case of a highly parallel computing application. The architecture can also be used for the same function on the same data to use mitigation techniques based on replication and compare the results obtained from the hardware modules to detect and eventually correct errors.

However, AXI IP Core is not immune to SEUs in the configuration memory. Several research studies have been conducted to assess the effects of SEUs on the configuration memory of programmable devices. However, only a few have examined the AXI Interconnect module, which is used for communication between the processor system and the programmable logic. In [66] and [67], the authors evaluate the reliability of various AXI interfaces connected to the AXI Interconnect core by accumulating faults in the configuration memory through fault injection. Specifically, they tested the AXI Interconnect along with the AXI-DMA IP Cores as a single module and found that it is a weak link for the design, even with the hardening of the AXI DMA module. Nevertheless, the characteristics of errors and experienced misbehaviors have not been evaluated.

In [68] and further in [69], we investigated the implications of radiation-induced SEUs on the AXI Interconnect module. SEUs are injected into the CRAM of a Zynq-7000 AP-SoC programmable logic, targeting only the specific module under test and the unused resources in its vicinity. The experimental analysis results demonstrate numerous effects on the communication architectures, such as the unavailability and incorrect computation of the modules connected via the AXI Interconnect module.

8.2.2 The Reliability Evaluation Flow for AXI interconnect Core

We propose a reliability evaluation platform and flow for evaluating AXI Interconnect Core against SEU. The platform is based on PyXEL framework to support fault injection in the programmable logic of a Zynq-7000 device. This platform consists of a fault injection framework and a test platform. The structure of the reliability evaluation environment is depicted in Figure 8.1.

Beginning with the bitstream and the netlist generated by the standard FPGA design process, we exploited the PyXELframework to analyze the bitstream structure and identify the subset of the bitstream related to the AXI Interconnect module. This step allows us to reduce the injection space to a subset of the entire space, so that only faults affecting the module being tested can be injected, separating the cross-section

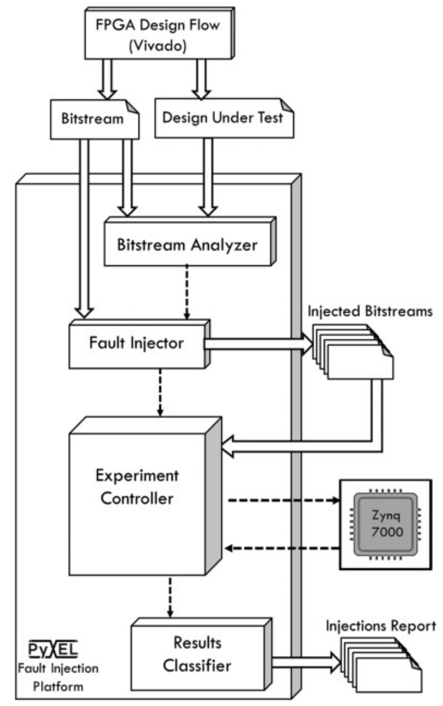


Figure 8.1 A conceptual view of the reliability evaluation environment for AXI Interconnect IP Core

of the AXI Interconnect IP core from the cross-section of the whole design. The chosen subset of bits is limited to the hardware resources allocated to the module under test and the unused resources adjacent to it. The fault injector, instrumented with the information obtained from the bitstream analyzer, can be used to generate a set of faulty bitstreams with specific characteristics. These characteristics can include the subspace of the configuration data where the injection should be done and the type of fault to inject, such as single or multiple bit corruption, or a desired bit transition (set, reset, or flip).

The experiment controller is capable of automatically detecting errors by comparing the obtained results with the expected results, in order to determine if any faults injected into the design have caused any errors. Specifically, the controller configures the programmable logic, launches the software to detect errors in the design being tested, and produces a report indicating the accuracy and availability of the data obtained from the computing units integrated into the hardware and connected to the processor system through the module under test.

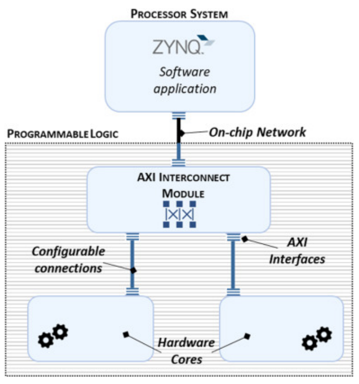


Figure 8.2 A schema of the architecture of the benchmark design

8.2.3 Hardware and Software Benchmark

A benchmark design based on the hardware accelerator and redundant paradigm has been proposed for analyzing the reliability of the AXI Interconnect IP Core. The design implements a dual-with-comparison approach on a replicated core implemented in programmable logic. A test routine executing on the processor system stimulates the hardware accelerators with a randomly generated test vector. The outputs are then compared by the processor system with the expected results, and the execution report is sent to the host computer.

Hardware Platform

The benchmark design features a hardware accelerator duplicated and connected to the processor system with the AXI Interconnect IP Core. The hardware accelerator was developed with Vivado HLS and is capable of computing a non-linear signature from four 32 bit fixed-point parameters (22 bits for the integer part and 10 bit for the decimal part). Moreover, it offers an AXI4-Lite interface that provides access to six memory-mapped registers, four for the inputs, one for the output, and a control register. A schematic of the system under test is provided in Figure 8.2.

The two hardware accelerators and the AXI Interconnect IP Core, which is configured as a single master and multiple slaves, connecting the processor system to both instances of the hardware accelerator, are implemented on the programmable hardware. Furthermore, the system-on-chip is connected to the host computer, which can configure the FPGA through the JTAG interface, initiate the test routine running on the processor system over a serial connection, and receive the report generated by the test routine on the same channel.

Software Test Routine

The test routine running on the processor system is composed of three distinct parts: a preamble, a body, and an epilogue. At the beginning, the preamble initializes the software data structures necessary to use the hardware accelerator IPs and verifies that they are in a correct state. During the body of the routine, each hardware is stimulated with 1000 different inputs. For each input, the result is collected and verified for correctness or to detect if the IP under test hangs. The epilogue of the routine reports the status of the IP cores to the fault injection platform and signals the end of the test routine. Throughout all phases, the processor system reports the status of the test routine and IP cores executions results to the host computer, which stores them for future analysis.

8.2.4 Reliability Analysis for AXI Interconnect IP Core

The platform runs on the host computer and is responsible for managing the experimental workflow. In particular, it controls the generation of the injection locations and injected bitstreams, the download of the faulty bitstreams into the configuration memory, the triggering of the test routine, and the collection of results. All injection steps are automated and executed by the platform without requiring user interaction.

A fault injection campaign has been executed on the Zynq-7000 AP-SoC, using the reliability evaluation environment exposed, to evaluate the errors induced by SEUs on the AXI Interconnect module. We generated 10,000 faulty bitstreams through randomly corrupting a single bit in each of them, allowing for both 0 to 1 and 1 to 0 transitions.

A representation of the described analysis flow is provided in Figure 8.3.

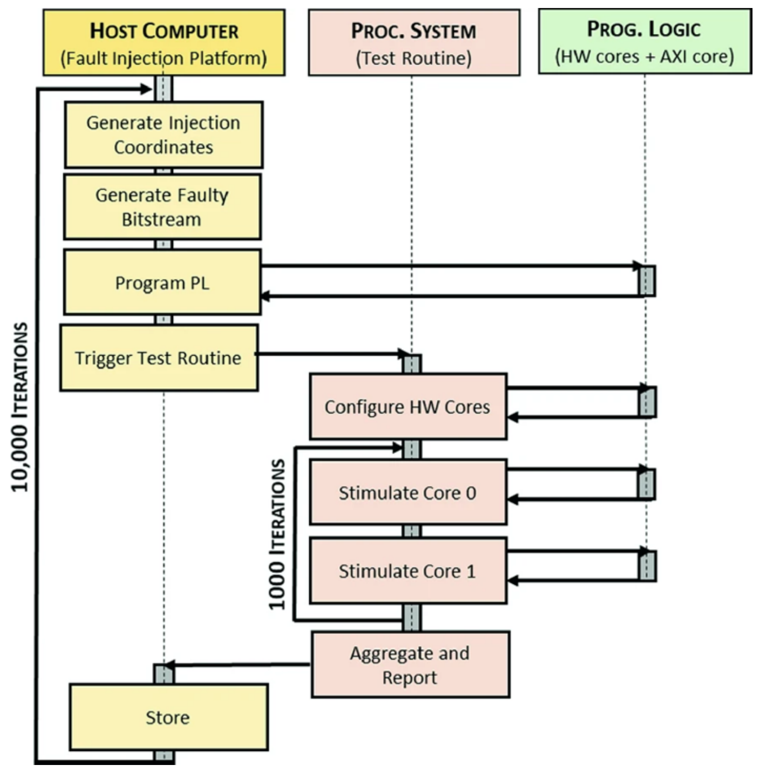


Figure 8.3 A schema of the experimental flow for evaluating errors on the AXI Interconnect IP Core

8.2.5 Results of Reliability Analysis of AXI Interconnect IP Core

The resulting errors have been divided into four categories based on the effects they produced. These categories are: unavailability, silent data corruption, detectable data corruption, and unavailability and data corruption.

Unavailability error is caused when the communication core fails completely, resulting in a loss of data and commands from the master to the slaves or vice versa due to injected faults. Detectable data corruption occurs when the cores which are implementing the same computation and stimulated with the same input vector logic generate different outputs In this case, the master is able to detect the malfunction through comparison. Silent data corruption occurs when the outputs of the modules subjected to the same input return the same faulty output, making it detectable only through comparison with the expected results. Unavailability and data corruption includes a mix of unreachable and faulty cores. There has not been any case of a mix of unreachable and correctly working cores.

Out of the errors identified in the flawed configurations, 50.66% caused the hardware cores in the programmable logic to become completely inaccessible. 4.90% of the errors led to partial unavailability, with one core being unreachable by the master and the communication, while the other core experienced errors including unavailability and data corruption.

The remaining errors were related to data transferred between the master and the slave, with 11.76% generating detectable data corruption and 32.68% leading to silent data corruption. Figure 8.4 illustrates the types of faults and their respective percentages.

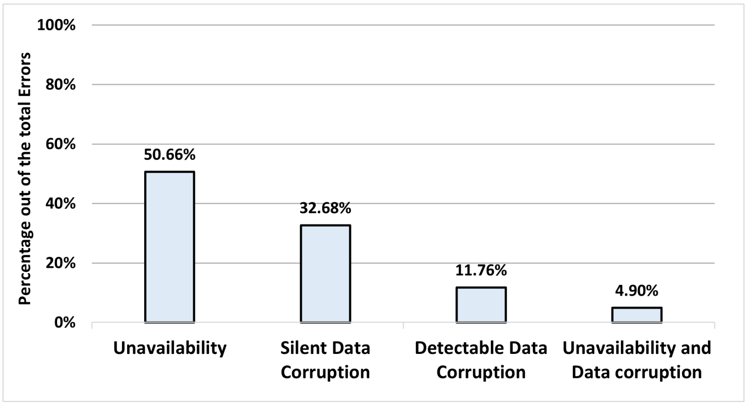


Figure 8.4 Categorization of SEU-induced Errors on AXI Interconnection Core

Further analysis of the categories reporting errors on transferred data has been conducted. It has been found that 78.80% of the errors introduced by the communication core over exchanged data are stuck-at faults. These faults always have an effect on the communication between hardware cores and, particularly, the same bits of the data words are affected for both cores. It is important to note that if the hardware cores are connected to the master system through the same AXI Interconnect, as is usually the case, then stuck-at faults introduced by the AXI Interconnect Module can lead to silent data corruption, even when the cores are duplicated.

8.3 Research Advancements in Reliability Evaluation of AXI Interconnections Module

A comprehensive analysis of the AXI interconnect module has been presented. The role of the AXI interconnection module, in charge of managing data transfer between processor system and hardware-accelerated modules, in producing common mode failures has been analyzed. In particular, the fact that AXI Interconnection has been identified as a single point of failure that can introduce errors affecting replicated modules, leading to silent data corruption.

Chapter 9

Evaluating Reliability of Hardware-Acceleration for Neural Networks

9.1 Overview on Reliability of Hardware-Accelerated Neural Networks

Neural Networks are becoming the state-of-the-art solution for solving a multitude of different and complex problems. They have proven to be suitable for being adopted in different industries such as automotive, computer vision, and healthcare. This has caused an increase in interest in studying the resilience and tolerance of neural networks against faults.

In addition, advances in computer hardware and software have made Deep Neural Networks (DNNs) more accessible and easier to use, making them even more attractive to researchers and developers. Novel DNNs architecture is characterized by a high number of layers and parameters, as well as by a higher computational complexity.

GPUs are generally the best choice for training neural networks, as they offer high-performance and parallel computing capabilities. However, CPUs are often used in inference when the required computational power is not too high. For large-scale applications, specialized hardware accelerators can offer more remarkable performance.

Programmable Hardware has emerged as a contender for high-performance neural network inference tasks. Indeed, programmable hardware systems are particularly attractive as they are highly configurable and can be programmed to meet the specific requirements of a neural network, adapting to the specific architecture. Furthermore, they offer low-latency and high-throughput operations, making them ideal for applications that require real-time processing.

In particular, Hardware-reconfigurable Systems-on-chips (SoCs) are becoming increasingly popular for neural network inference due to their heterogeneous architecture. By using reconfigurable SoCs, neural networks can be easily adapted to different tasks, and integrated with other systems, and the applications can take advantage of hardware accelerators and dedicated processors for computationally demanding tasks while relying on the flexibility and ease of use provided by the software part.

The architecture of DNNs is composed of layers with a few to hundreds of millions of parameters that are used to perform computations such as convolution and pooling. These parameters, inputs, and outputs of each layer are all potential sources of errors in the network. Additionally, the hardware and its fault models can affect the parameters and nodes at the application, topology, or algorithmic level.

However, reliability analysis of deep neural networks typically ignores the actual implementation of the hardware. It is based solely on the corruption of data and parameters of the layers, or topological modifications of the net. Even though the large amount of memory required by neural networks makes storage one of the main sources of error, soft errors in the storage elements are only a subset of the real faults that can affect a device. As a result, the reliability and resilience evaluation of deep neural networks should not overlook errors in data paths and hardware architecture.

9.2 Effects of SEUs on Reconfigurable Accelerated Quantized Neural Network

9.2.1 State of the Art of Reliability Analysis of Quantized Neural Network implemented on Reconfigurable Hardware

In recent years, researchers have demonstrated that Convolutional Neural Networks (CNN) can produce accurate results even when using reduced precision data types. For example, [70] and [71] presented CNN implementations using fixed-point data types; [72] and [73] presented extreme cases of reduced precision, where weights and activations are constrained to single-bit values. The Quantized CNN and Binary Neural Network implementations offer new possibilities for applications where hardware resources are limited, as the redundancy induced by floating point data is removed. Nevertheless, the impact on the reliability of such implementations must be studied, particularly when safety-critical applications are involved.

Due to the fact that the configuration memory containing the configuration data of the design mapped to the device is composed of SRAM cells, which are highly vulnerable to radiation effects, it is important to analyze the system's reliability when an SRAM-based FPGA is used, and possible fault-tolerant strategies must be implemented to meet the reliability requirements, particularly in safety-critical applications.

In [74], we present the results of a reliability analysis of a CNN implementation on a Pynq-Z1 board, as reported in [75], conducted via a fault injection platform that utilizes multiple boards for acceleration. The analysis presents a preliminary evaluation of the effects of Single Event Upsets (SEUs) and MBUs on a Quantized CNN implementation with 1-bit weights and activations, in the configuration memory. Additionally, we observed that the fault injection results for SEUs in the configuration memory have different impacts on correctness and performance, although a more detailed analysis is needed.

9.2.2 Fault Injection Platform

In order to assess the impact of SEU in configuration memory on SRAM-based FPGA, a fault injection platform has been created. The fault injection methodology is based on flipping bits in the configuration memory frame data to emulate SEU in the configuration memory. In order to speed up the reliability evaluation, the platform has been parallelized to utilize multiple boards. Additionally, it supports MBU fault injection in the configuration memory. Figure 9.1 provides an overview of the platform's overall architecture.

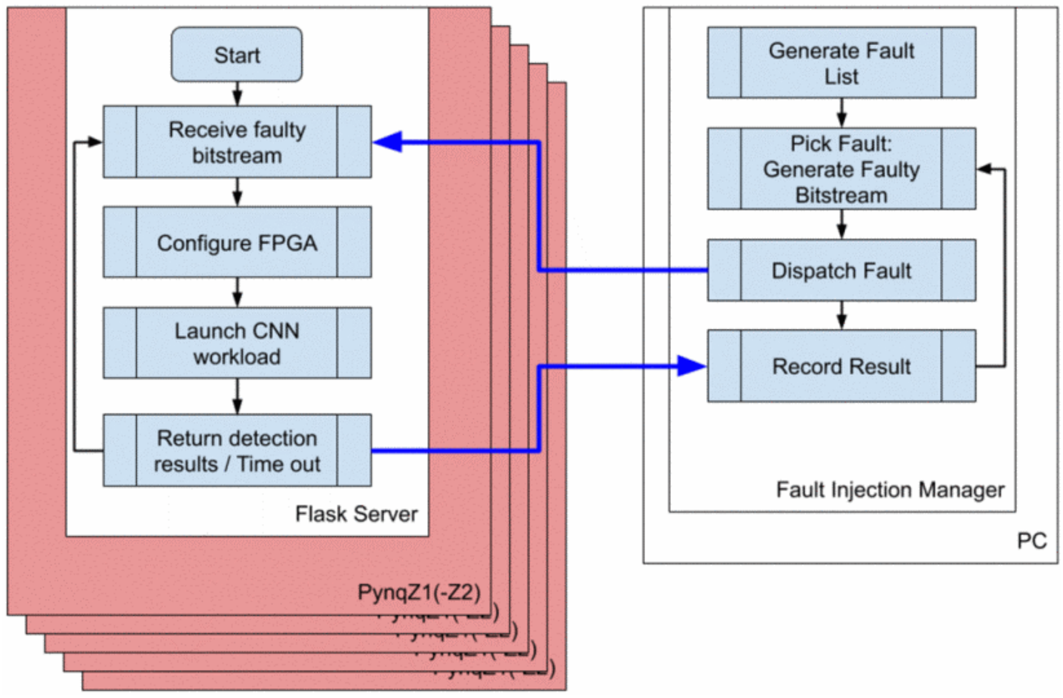


Figure 9.1 [1]Architectural Schema of the Fault Injection Platform

The Pynq-Z1 boards, which are used in the platform, are equipped with a Xilinx Zynq-7000 SRAM-based FPGA device that has an ARM dual-core processor embedded. The Pynq environment provides PetaLinux as its operating system, with a Python environment integrated, making the platform's implementation much easier, especially for managing communication with the host PC. On the Pynq board side, a server has been implemented using Python, enabling it to accept fault injection requests from the host PC. This server performs a variety of tasks upon receiving requests:

- Fault Bitstream Acquisition: the host PC generates a faulty bitstream and sends it to the server on Pynq via a request, as this is more efficient than generating it on Pynq itself.
- FPGA Configuration: in order to carry out the fault injection run with the uploaded faulty bitstream, the Python library provided for using FINN BNN was modified to ensure that the faulty bitstream is downloaded each time an object recognition workload is initiated. This modification enabled the FPGA to be configured correctly, ensuring that the faulty bitstream was always used during the fault injection run.
- Inference: the server runs a new process in order to perform object recognition using the implemented CNN implementation. The same image will be used and loaded from the SD card for all the fault injection runs.
- Result Report: the inference result, even if it failed or timed out, is sent to the host computer.

In order to prevent the server from becoming unresponsive due to a fault injection, the PetaLinux device tree and boot image have been recompiled to enable a watchdog reset-on-timeout feature. In the event of a functional interruption, the watchdog will reset the board after 10 seconds, causing the server to reboot and return to an online state.

A Fault Injection Manager is present on the host PC. It is in charge for:

- Fault Injection Location: depending on the type of fault model chosen, a list of faults is generated and stored.
- Faulty Bitstream Selection: the fault from the list which has not been processed yet is selected, and the bitstream associated is generated. t
- Faulty Run: Once a faulty bitstream has been generated, a Pynq board is checked for availability, and if found, the faulty bitstream is provided along with the fault injection request.
- Result Report: Once the fault injection request has been sent, the thread waits for a response from the server, with a timeout set to five seconds. This time frame is much longer than the expected duration of the object recognition

Table 9.1 Programmable-hardware utilization of the cnvw1a1 network implemented in Zynq-7020

Resources	Available [#]	Used [#]
LUTs	53,200	25,447 (47.83%)
Flip-Flops	106,400	42,090 (39.56%)
BRAMs	140	124 (88.57%)
DSPs	220	24 (10.91%)

workload. If the server does not respond, the thread will report the server as dead and wait for it to be revived through a watchdog reset.

These steps are performed continuously until all the elements in the fault list have been processed.

This fault injection platform enables us to conduct a fault injection campaign involving a large number of faults in a relatively brief period of time, depending on the number of boards employed. In our case, by using 4 boards, the average time for each fault injection run is 1 second.

9.2.3 Reliability Evaluation Analysis of the Binarized Network

The cnvW1A1 network from the CNN implementations presented in [75] utilizes only 1 bit of weight and 1 bit of activation. The hardware utilization of this particular network, when implemented on the Pynq board, is reported in Table 9.1. We then selected the road signs classifier, one of the three classifiers already provided, to recognize road signs from the input image, which was used as the workload executed by the fault injection server on the Pynq board. This workload produced two results: the classification index and the recognition duration.

For estimating the nominal recognition duration, 10,000 runs were conducted with fault-free bitstreams. Indeed, timing can cause the recognition duration to vary within a certain margin. This margin was used to determine whether a run had been impacted by the injected fault, leading to a decrease in performance. Although this margin is not completely accurate, it gives an indication of how the performance of the CNN can be negatively impacted by SEUs in the configuration memory.

The recognition duration in the configuration memory when there are no errors present resulted to be around 1580 μs for most of the runs, while a few of them are slightly slower with a duration of fewer than 1700 μs. However, there are a few cases in which the duration can exceed 1800 μs.

We generated three different fault lists associated with that many fault injection campaigns. They are:

- single bit fault model, random bits selected from Logic Location file generated along with the bitstream;
- single bit fault model, random bits selected in the remaining configuration memory;
- multiple bits fault model, 4 bits randomly selected as a cluster to emulate the MBU as reported in [46];

The results of the fault injections are then categorized based on classification results and timing as follows:

- Masked: the classification result is correct and the inference time is lower than 2 ms;
- Performance Degradation: the classification result is correct, but the inference time is over 2 ms;
- Misclassification: the object the classification result is not correct;
- Timeout: the object classification did not finish within 5 seconds.

Random SEU in Logic Locations

The first fault list has been generated from the Logic Location file, which controls the resources such as latches, FFs, LUTs, and Block RAMs used in the design. More than 500,000 faults have been generated and injected. As a result, 99.30% of the faults have been masked, while in the remaining 0.7% of runs, no Misclassification was observed. Additionally, 99.7% of runs suffered from Performance Degradation which could reach up to more than 21 ms.

It is worth noting that the lack of Misclassification can be attributed to the fact that the bits from the Logic Location file are mainly related to registers and Block RAMs used in the CNN implementation. Furthermore, due to the multilayer nature and the training process of the CNN implementation [75], the likelihood of a corrupted node in the network causing a wrong final classification may be quite low, given that the weights and activations are limited to single bit values in the selected cnvW1A1 network. Nonetheless, the Performance Degradation effects observed in the results are also critical in safety-critical applications, as they could compromise the correct execution of some real-time tasks, as well as a few runs with Timeout.

Random SEU in Configuration Memory

When considering SRAM-based FPGAs, the design implemented and mapped to the FPGA typically involves multiple types of resources, such as registers and Block RAMs specified in the Logic Location file. In addition to logic resources, configuration memory configure also Digital Signal Processing (DSP) units and routing resources that are not present in the logic location file. However, bits in the configuration memory controlling the DSP units, LUTs, and configurable routing resources connecting the nodes in the network could also be affected. Therefore, a second fault injection campaign was conducted with randomly selected bits in the configuration memory, which did not overlap with the Logic Location file.

The results of the fault injection campaign, consisting of more than 14,000 fault injections, revealed that the percentage of not-masked results is significantly higher than the results seen previous fault injection campaign, around 4.29%. Additionally, more than 80% of the not-masked categories are Misclassifications (41.94 %) and Timeout (40.00%), which could be attributed to the corruption of the CNN structure caused by the SEU injected into the configuration memory, while the remaining 18.06% is classified as Performance Degradation. The Recognition Duration of the Misclassification is mostly within the nominal execution time range, making it hard to detect, similar to Silent Data Corruption errors. This suggests that proper fault-tolerant techniques should be employed to improve the reliability of CNN implementation, not just for user registers and Block RAMs, but especially for the others element in the configuration memory to protect the integrity of the CNN.

Random Multiple Bit Upset in Configuration Memory

Due to the scaling of technology, transistors are becoming increasingly smaller and their threshold voltage is decreasing, making them more vulnerable to radiation particles even at lower energy levels. In space environments, the energy of radiation particles that may strike the device could reach the GeV scale, which means that SEU in the configuration memory may not be enough to model the radiation effects of a single particle [46, 36]. Thus, another fault injection campaign was carried out with randomly selected 4-bit clusters following the patterns reported in [46].

The fault injection consisted of more than 14,000 fault injections. Results revealed that some of the Performance Degradation has worsened into Misclassification or Timeout, and the rate of not-masked has increased in comparison to the SEU scenario. This is due to the fact that more bits can be corrupted, while the resiliency against SEU is less effective against MBU.

9.3 Emulating Architectural Faults on Hardware Accelerated Neural Networks

9.3.1 State of the Art of Robustness Analysis of Hardware Accelerated Neural Networks

How much a neural network can be considered robust against faults is mainly related to its inference phase, as the training phase is typically run in a controlled environment. Possible causes of hardware faults include physical manufacturing defects or radiation damage. If such faults occur, and electrical or logical mechanisms do not mask them, they may both reach the system's outputs, causing an erroneous output or even leading to system failure. To study the robustness of neural networks, proposed state-of-the-art approaches can work on various levels of abstraction.

To simulate extreme radiation environments and extended periods of operation, the hardware can be exposed to a flux of particles. This type of testing necessitates the use of highly specialized equipment and is limited in terms of visibility and control, meaning that often multiple components are irradiated and may fail at once.

In cases where radiation testing is not feasible, alternative methods of emulating the system and its potential faults are used in order to assess its reliability. Software-based fault injection techniques are often used in conjunction with radiation testing to evaluate the reliability of neural network systems. This approach involves simulating particular fault models in the application-level model of the network, allowing for experiments to be conducted in a fully controlled environment. However, as the actual hardware is not being taken into account, this method can result in inaccurate evaluations. Software-level simulation is the most abstracted method from the hardware, as it involves an application-level analysis that is not aware of the hardware actually implementing the neural network. It offers many advantages, such as high controllability and inherent simplicity and flexibility, but it also has the limitation of complete abstraction from the hardware. Some faults, such as bit flips in memory cells containing data, can be easily emulated in the software-level model, but errors in hardware microarchitectural components, such as those affecting communication interfaces, interconnection lines, timing, specific computational units, and other elements related to the accelerator hardware architecture, are much harder to emulate.

For achieving a more precise evaluation, a hardware-level simulation that is based on a model of the underlying hardware platform may be conducted. This simulation is typically done at the RT- or Gate-level and involves injecting faults into the simulated hardware architecture. By doing so, the behavior of the hardware components can be accurately represented, making the analysis more reliable. The downside of this method is that simulation-based approaches to analyzing modern and complex architectures can be extremely costly in terms of execution time and computational power, making them unaffordable in many cases. Additionally, they cannot always access the low-level description of the hardware, meaning they are limited to a higher level of abstraction.

The increasing need for neural networks that are highly reliable for mission-critical applications, coupled with the miniaturization of device technology, which makes their hardware accelerators more fragile, has pushed research into the reliability assessment of neural network systems. As a result, several methodologies, approaches, and platforms for reliability analysis have been developed. In [76], the authors present a lightweight framework for empirical analysis based on a software-based approach working at the algorithmic level. It allows users to perform preliminary analyses to gain insight into where to conduct more detailed fault analysis at the microarchitectural level. However, microarchitecture faults themselves are not covered. In [77], an application-level fault injection environment is proposed for evaluating hardware-accelerated systems, which is built on the Tiny-CNN framework. This approach enables faults to be injected into the data path and buffer without considering combinational logic and control logic units. As the RTL implementation of the accelerators was not available, the authors mapped each line of code in the software simulator to the respective hardware element to measure the impact of each fault injection location in terms of the hardware microarchitecture. Additionally, in [78] and [79], the fault injectors are based on a torch-based framework but are implemented at the application level. However, they are often based on application-level software-based fault injection approaches, without considering the underlying hardware platform. In [80], the authors proposed CLASSES a framework to evaluate the reliability of CNN executing on GPUs against SEUs taking into account the hardware platform. Other studies, which do not focus on the application level, mainly utilize radiation testing. For example, [81] analyzes the reliability of NVIDIA's Kepler, Maxwell, and Pascal GPU architectures by using YOLO, Faster R-CNN, and ResNet. This analysis is conducted through both architecture-level fault injection, with NVIDIA SASSIFI fault injector, and radiation testing.

Different from previous approaches, we proposed, in [82] and further in [83], a methodology for evaluating the resilience of neural network systems based on the use of heterogenous System-on-Chip. This approach exploits the reconfigurability of the platform to emulate the target hardware accelerator and inject faults in its microarchitectural hardware elements by manipulating the configuration memory data. The method allows a comprehensive and controlled analysis based on full control over the models and locations of the faults injected into the hardware components, without the need for radiation-based approaches and working at a lower level of abstraction than traditional software-based approaches. Taking into account the microarchitectural faults of the hardware platform, which cannot be done in a software-level simulation approach that is oblivious to the underlying hardware. Furthermore, the use of an actual hardware platform instead of a simulation environment for the analysis yields much faster results than software simulation of the hardware level. Lastly, this approach can be used to perform microarchitectural analysis when radiation tests are not possible or as a pre-test before actual radiation tests.

9.3.2 Hybrid-based Approach to Neural Network Reliability

The proposed methodology is suitable for evaluating the robustness of neural network architectures implemented on FPGA-like platforms. Additionally, it is exploitable to target other hardware architectures by emulating the target hardware using programmable hardware. The methodology relies on heterogeneous devices to perform reliability analyses that are not limited to the high abstraction level of software implementation of the neural network architecture but consider the effects of faults affecting the hardware. The programmable hardware can be used to emulate the hardware architecture of the used hardware accelerator and is also exploited for emulating faults involving the hardware through configuration memory manipulation.

The proposed flow is schematized in Figure 9.2. The hardware accelerator to be emulated is implemented in the programmable hardware. It means that HDL files, and HLS methodology, as ready-to-be-used IPs can be adopted to generate the hardware to evaluate. An experiment manager is instrumented with the target neural network model, as well as the information about the experimental analysis to be performed. This module manages the use of the emulated accelerators in the network model, controls the fault injector, and collects the results.

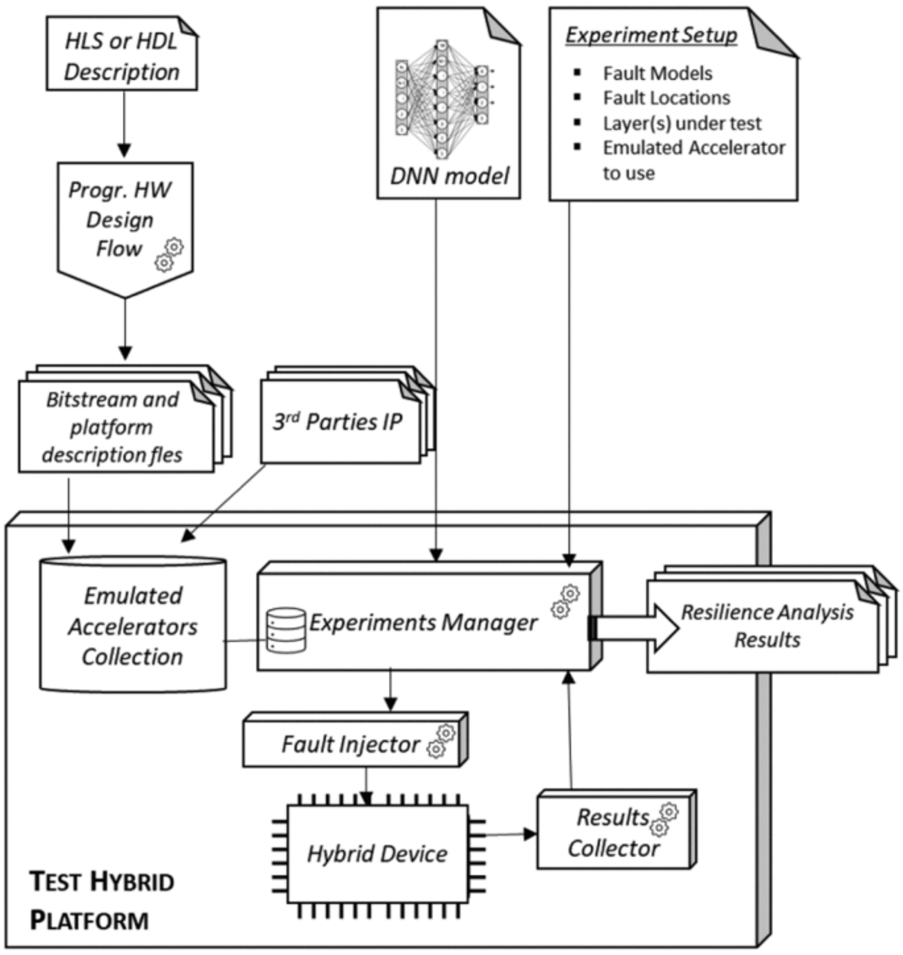


Figure 9.2 Conceptual schema for the proposed approach.

The hybrid devices, due to their programmable logic, provide the opportunity to assess the impact of faults on the hardware in a direct manner. Furthermore, the ability to inject faults into specific hardware components and resources grants a high level of control and visibility. Additionally, emulation-based analysis of hardware not yet produced can be conducted, which provides valuable insight into the resiliency of neural network architectures and hardware accelerators (ASICs and FPGAs in particular) even during the design process. Particularly when the solution for hardware acceleration is programmable logic, the actual hardware accelerator can be implemented on the programmable hardware without the need for emulation of the target accelerator.

The proposed platform enables the manipulation of configuration data of programmable hardware, allowing for the injection of fault models such as stuck-at, conflict, couplings, and others. To do this, specific bits of the bitstream are modified, although the role of each bit is not provided by the vendors. Nonetheless, research works have demonstrated that it is possible to modify the implemented netlist without relying on vendor tools [22, 25, 84].

9.3.3 FireNN Platform

FireNN is the first platform that provides the capability to analyze the resilience of neural networks at both the application and hardware levels by means of emulation. We have selected the Zynq family as the target hybrid platform, in particular, the Zynq-7020. This SoC is equipped with two ARM dual-core Cortex-A9 processors and a programmable logic, which is manufactured using a 28 nm process.

The FireNN platform is composed of two distinct environments: the Machine and the Engine. Figure 4 illustrates the modules and frameworks that make up the FireNN platform. The Machine, which runs on the host computer, provides a means of defining and customizing the neural network model by building on the PyTorch framework [85]. It acts as a controller, is responsible for the overall flow of the experiment, and is capable of communicating with the FireNN Engine on the Zynq platform in order to deploy the emulated accelerator and initiate the fault injection process. An architectural view of the platform is reported in Figure 9.3

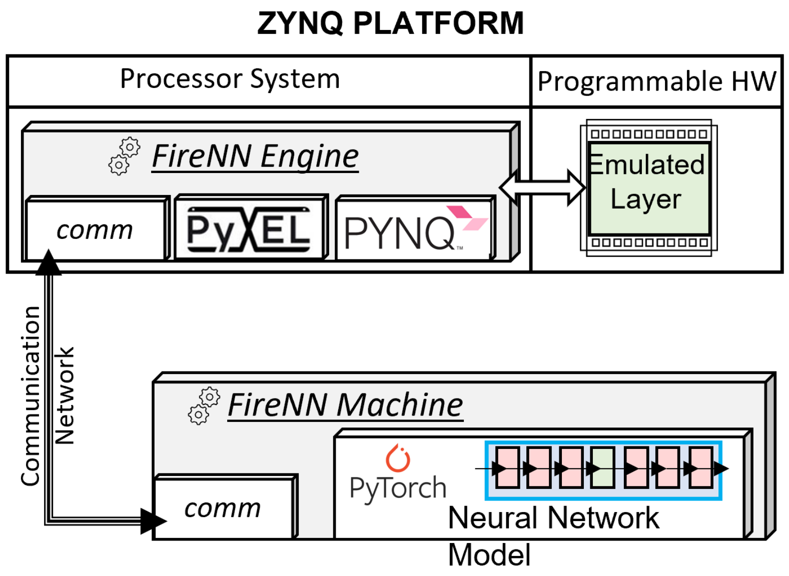


Figure 9.3 Architectural view of the FireNN platform and its modules.

The primary objective of the Machine on the host computer is to transfer the computational demands of the entire neural network away from the Zynq. However, if the performance of the hybrid device is adequate, the Machine and the Engine can also be used as two processes on the same device. The FireNN Machine offers APIs to move the computation of a PyTorch module (i.g. one or more software layers) to the programmable logic. These relocation APIs provided by the Machine mimic the interface provided by PyTorch to move the computation from the CPU to the GPUs. To achieve this, an appropriate hardware-implemented layer is instantiated on the programmable logic to perform the same mathematical operation that the software layer was intended to do. The hardware layer is designed to emulate or implement the target hardware to be evaluated.

The Machine extends the PyTorch model of the network by providing a mechanism that allows a PyTorch module to be replaced with an analogous hardware module running on programmable hardware, thus managing the neural network architecture and experimental execution in a transparent way. The Engine is responsible for instantiating and managing a specific hardware module on the programmable hardware side. To facilitate the relocation of this module, a container called a Shell is encapsulated in the module class of PyTorch. The Shell is then integrated into the neural network model and serves as a mechanism to switch the execution from the original module to the hardware module and back. When a Shell is instantiated on the Machine side, an associated object is instantiated by the Engine in its domain, referred to as a Gear. The Gear is a hardware implementable module emulating the hardware accelerator.

The Shell implements a routine for determining the dimensions of the input and output data. Some PyTorch modules are not specific to the I/O data dimensions, meaning they can accept tensors of any size as input, but the architecture of the network determines the univocal I/O data dimensions (e.g. modules that work with fixed tensor sizes such as fully connected). Therefore, the Shell is necessary to obtain the information on the input and output dimensionality in order to create the hardware module. Additionally, the Shell has methods to easily switch between the original data path using the PyTorch original layer and the new data path using the hardware module implemented on the Zynq. This allows the inference to be executed using the original network or the relocated version with a part of the network implemented in hardware. The deployment, execution, and emulation of faults on the programmable hardware are all handled by the Engine but are triggered by the Machine through the Shell since the Gear is not accessible to the user. The Machine also implements methods for executing the injection of different kinds of fault models at the software level, which does not require hardware emulation. Finally, the Shell has a tracking mechanism for storing the input and output data of the Shell itself to enable post hoc analysis.

The Engine of the Zynq processor system is responsible for managing the programmable hardware and the fault injection process. The Engine's primary functions are the deployment, execution, and fault injection of the Gears. Whenever a Shell is instantiated on the Machine side, the Engine instantiates the corresponding Gear. It takes advantage of the PYNQ project, which is an open-source project supported by Xilinx, to facilitate the exploitation of programmable hardware. The PYNQ project provides the Application Program Interfaces (APIs) for configuring the programmable hardware of the Zynq, as well as the APIs for managing Input/Output (I/O) and communication between the programmable hardware and the processor system [86].

A Gear is a computational module that can be implemented on programmable hardware and emulates a target hardware accelerator. It is composed of three elements: an interface, a driver, and an implementation file (i.e., a bitstream). The interface is common to all Gears and allows the Engine to manage them independently of their particular hardware implementations. The driver provides the APIs for using the specific hardware module through the interface, while the implementation file is the bitstream containing the configuration data for programming the programmable hardware with the hardware module. If necessary, when a Gear object is created, the Shell must transmit the characterizing parameters of the original neural network layer it is replacing (e.g., weights, bias, etc.) during the relocation process.

When a Shell requests a computation to be carried out on a particular Gear, the Engine configures the programmable hardware accordingly, receives the input data, initiates the computation on the hardware, and then sends back the results. Thus, each Gear is linked to a particular computational layer on the software model of the network and simulates a particular hardware accelerator. The development of a Gear can be completed using the customary development flow for programmable hardware (e.g., by creating an HDL or HLS description), or it can be based on IPs provided by external entities.

The Engine is responsible for the process of fault injection, which is enabled by the PyXEL framework. PyXEL handles the manipulation of the configuration bitstream of the programmable hardware, enabling the injection of specific fault models, such as soft errors that affect the truth table of a look-up table (LUT) or the open-interconnection fault model. The Gear common interface is equipped with a configurable timeout mechanism to prevent endless waits due to faults injected during the injection operation. When the Shell requests injection of a specific fault model, a faulty version of the Gear is created and used for the computation. The Engine can be instructed to inject faults into a specific entity (e.g. the AXI Interface), specific resources (e.g. BRAMs interconnections, LUTs truth tables) or induce specific faults (e.g. open interconnections).

For clarity, an example of the evaluation flow of the FireNN platform is reported in Figure 9.4

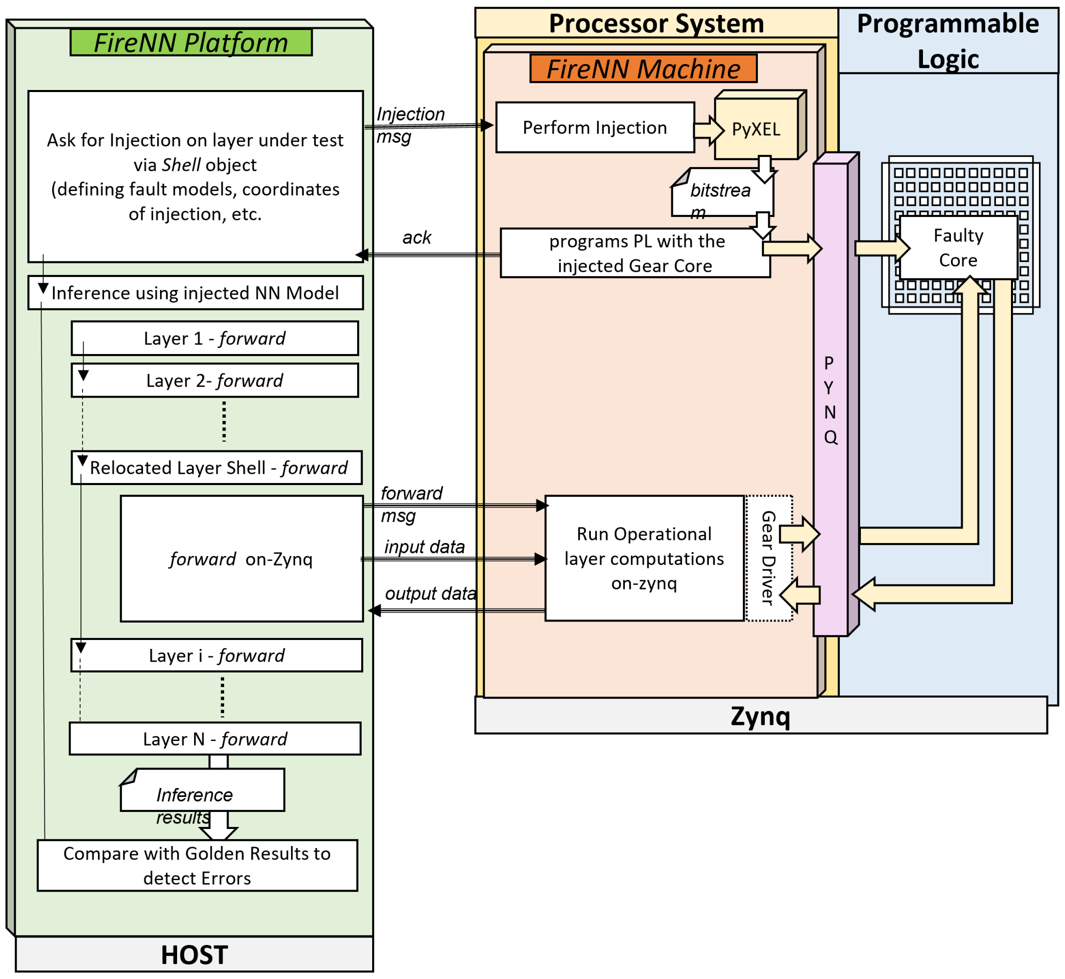

FireNN Platform
Ask for Injection on layer under test via *Shell* object (defining fault models, coordinates of injection, etc.
Inference using injected NN Model
Layer 1 - *forward*
Layer 2- *forward*
Relocated Layer Shell - *forward*
forward on-Zynq
Layer i - *forward*
Layer N - *forward*
Inference results
Compare with Golden Results to detect Errors
HOST
Injection msg
ack
forward msg
input data
output data
Processor System
Programmable Logic
FireNN Machine
Perform Injection
PyXEL
bitstream
programs PL with the injected Gear Core
Faulty Core
PYNQ
Run Operational layer computations on-zynq
Gear Driver
Zynq

Figure 9.4 Example of the evaluation flow of the FireNN platform.

9.3.4 Reliability Analysis of an AlexNet Layer

We used the FireNN platform to evaluate the reliability of a layer of the AlexNet neural network. The 2D-convolution computation executed by the fifth convolutional layer of AlexNet was implemented as a hardware accelerator using Vivado HLS [87]. We evaluated the network layer with fault injection campaigns, using both commonly used software-level analysis and the proposed approach which emulates accelerators and fault models using hybrid devices. We evaluated various parameters such as error rates, failure rates, ratios between failures and errors, timeout events, and distributions of degradation and misclassification errors on an evaluation set.

The experimental analysis in this paper used the version of AlexNet provided by torchvision, a popular deep-learning library [88]. This version of the network was trained on the ImageNet dataset, a collection of 1,000 classes of objects. The architecture of the network consists of multiple layers implementing convolution, pooling, and ReLU operations.

The input to the platform is a 3-dimensional tensor of size 224x224x3 representing an RGB image. The last convolutional layer of the network is selected for the reliability analysis, as it has been identified as the most sensitive of the convolutional layers in [77]. This layer is composed of 590,080 parameters (also referred to as weights), in particular, 256 bias and 256 kernels, each with a size of 256x3x3.

The output of the last fully connected layer of the neural network is an array of values, each of which corresponds to a particular class. This value evaluates the likelihood of the input belonging to that particular class. To normalize and reduce these values to a probabilistic distribution over all the labels, a normalized exponential function (commonly referred to as softmax) is applied. As a result, the final output of the neural network is a list of labels with a corresponding degree of confidence for each one. The label with the highest confidence is the one that is selected as the classification output.

Fault Models

We considered the impact of distinct fault models on the system. Firstly, Single Event Upset affecting weights data have been evaluated at the software level. SEUs are emulated by directly modifying the value of the variables at the bit level, and errors are injected in the value of weights and layer inputs during the runtime of the execution. The purpose of these evaluation campaigns is to compare the results obtained by this analysis, which is commonly adopted in literature, with the results obtained by hardware-based evaluation.

SEUs affecting the configuration memory of programmable hardware must be treated differently, as they can have more serious consequences than other SEUs. In particular, due to the nature of programmable hardware, SEUs affecting the configuration memory can result in a permanent fault that impacts the hardware architecture of the circuit implemented on the programmable hardware, until the hardware is reprogrammed with a new circuit. As a second evaluation, we present a reliability analysis of the convolutional layer based on SEUs in the configuration memory. Depending on which bit is randomly corrupted, various actual faults can occur in the circuit implemented in the hardware. For instance, if a memory cell programming a LUT is affected, it may cause a logical fault, while if the memory cell is related to a programmable interconnection, it may cause an open fault or an antenna fault.

To conclude, the effects of open faults were evaluated exclusively. These faults can be caused by a variety of factors, including fabrication defects, aging, and SEUs (Single Event Upsets) in programmable hardware. To model the effect of open faults in the hardware accelerator architecture, the specific bits of the implemented circuit have been modified for manipulating the implemented netlist in order to emulate the desired fault model in the programmable hardware.

Hardware Accelerator

A hardware-accelerated version of the convolutional layer of the network to serve as a hardware module for the fifth convolutional layer of the AlexNet network has been developed.

Table 9.2 Resources used by the convolutional layer in Programmable Hardware.

Resource	AXI Core [#]	Convolutional Core [#]	Total [#]
Slice LUTs	4,452	31,746	36,198
Slice Registers	5,795	25,476	31,271
Block RAMs	16	67	83
DSP Slices	0	48	48
Muxes	25	87	102

The accelerator was created using Vivado HLS and is capable of computing 2-D multichannel convolution between inputs with dimension 13×13 and a 3×3 kernel, with 256 input channels and 256 output channels. To ensure compatibility with the PyTorch model of the overall network, the data are represented with a 32-bit floating-point representation. Data transfer between the processor system and programmable hardware is achieved through FPGA direct memory access. The IP Core is pipelined and performs convolution using an algorithm based on two buffers and a shifting window, with an AXI4-Lite control register interface. The resources used by the convolutional core and the AXI Core are reported in Table 9.2.

Dataset and Analysis Outcome

A set of 50 images from the ImageNet collection on fauna has been chosen as the evaluation set [89]. The input was preprocessed (i.e. cropped and normalized) to be suitable as inputs of the network, and they represent objects belonging to the set of 1,000 labels used in the training phase. Our goal is to identify if an injected fault will produce a modification in the confidence associated with each label, or eventually cause a change with respect to the original classification. Thus, the original model is initially used to obtain the results of an unfaulty run, which are then compared with the output obtained by a fault-injected model of the network. Any errors detected by this procedure are classified into three groups:

- Misclassification: the classification output has changed due to the injected fault.
- Degradation: the confidences of one or more labels have changed due to the injected fault.
- Timeout: the injected fault prevents the network from completing the classification

We considered misclassification as a critical failure, rather than an error because it drastically altered the expected results. To calculate the failure rate, we determined the number of injections that caused misclassification out of the total. The error rate was determined by counting the number of injections that caused at least one degradation, regardless of whether or not it led to misclassification, out of the total.

The evaluation set is composed of different images with heterogeneous characteristics, which could lead to a mixed outcome as a result of fault injection. to clarify further, a fault could produce different outcomes on different input images. This could include misclassifications, errors, and correct results assigned to different inputs of the evaluation set.

In our analysis, if at least one input has been misclassified, the error is categorized as a misclassification. Degradations follow the same rule. Anyway, we have additionally considered a deviation of less than 10^{-4} in the confidence percentage to be negligible since this could be caused, and undistinguishable by the errors introduced by using the IEEE-754 standard for data representation and computation. Therefore, a deviation of less than 10^{-4} percentage points is not classified as an error.

9.3.5 Results of Software-based Reliability Evaluation of an AlexNet Layer

We conducted two fault injection campaigns at software-level. The first campaign focused on faults affecting the weights and bias of the fifth convolutional layer of the AlexNet model. For the second fault injection campaign, we focused on faults affecting the input and output data of the layer. We used a traditional software fault injection approach to emulate SEUs, which are single-bit flips in the 32-bit floating-point representations of the parameters. The fault injection campaigns were executed using the software-level fault injector embedded in the FireNN platform.

We conducted 10,000 experiments for each fault injection campaign, in which we ran a classification task on the entire evaluation set with a randomly generated fault location (parameter and bit to inject) in each experiment. The first reliability analysis resulted in an error rate of 40.57%, a failure rate of 2.10%, and 5.18% of the detected degradations led to misclassification.

We conducted a second fault injection campaign using the same fault model, but this time affecting the inputs or outputs of the module instead of its parameters. For each experiment, we injected the fault in the same location for all the inputs of the module (e.g. on the same word of the input or output tensor and the same bit). The results showed an error rate of 46.16%, a failure rate of 15.48%, and a ratio of failures to errors of 33.53%.

Even if, as we already discussed, a single misclassified image of the evaluation set was enough for us to classify a fault as causing misclassification, the impact of the fault on the overall evaluation set may differ, since it can affect anywhere from one to all of the images. To gain insight into this, we examined the distribution of faults leading to misclassification over the number of outputs on which the misclassification occurred.

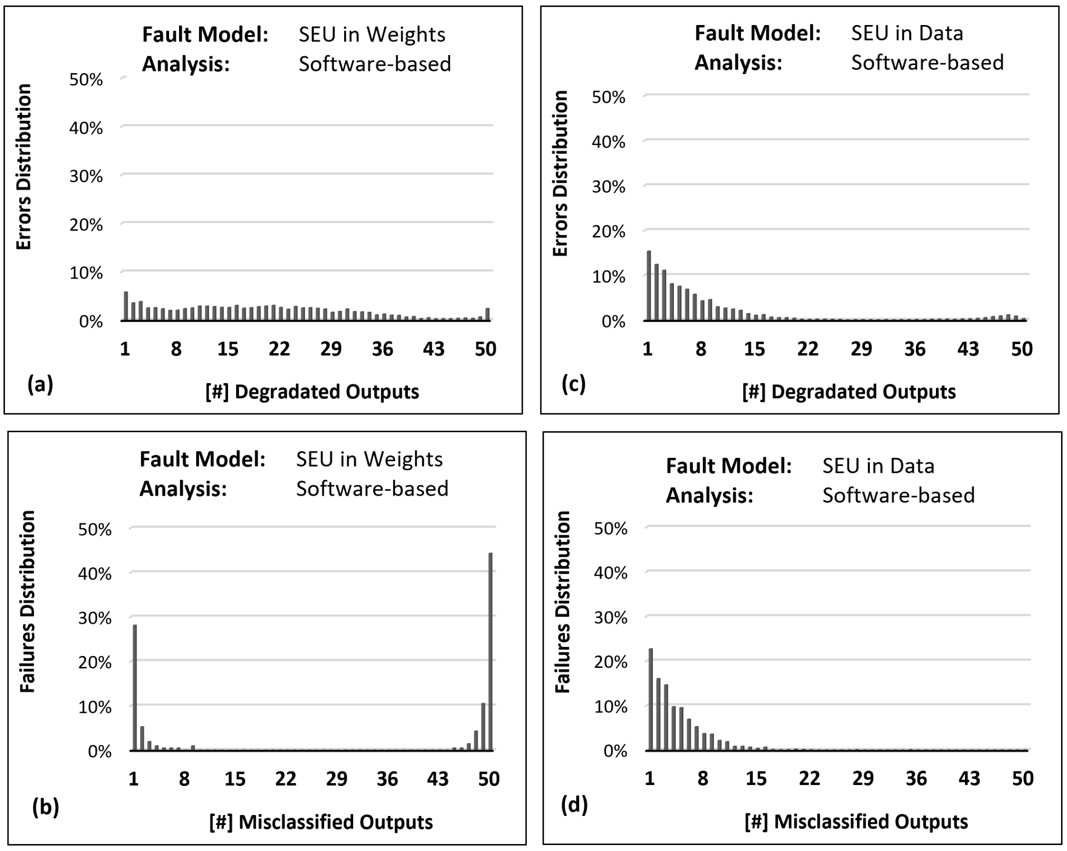


Figure 9.5 Distribution of the degradation (a,c) and misclassification (b,d) outcomes resulting from SEU in the weights (a,b) and SEU in input and output data (c,d) fault models.

Figure 9.5 presented the observed distributions. When evaluating SEU in weights, our findings suggest that faults causing misclassifications often exhibit one of two behaviors. The first is misclassification on a very small number of outputs, which may be due to certain elements of the evaluation set being more sensitive to variations caused by faults than others, and thus misclassifying even with slight deviations from the unfaulty behavior. The second is to affect a large portion of the evaluation set, which could be caused by stimulating highly critical bits of the parameters of the neural network layer. About SEUs in data, the results demonstrate that the probability of detecting multiple errors and failures decreases both for misclassifications and degradations as the multiplicity increases. This is reasonable behavior, since a specific feature extracted by the current and previous layers may be critical for the classification of a specific input or class, but less important for other inputs or classes.

9.3.6 Results of Hybrid-based Reliability Evaluation of an AlexNet Layer

By performing a hybrid fault injection campaign, we are able to analyze the behavior of the platform and address the reliability of the hardware accelerator when it is physically implemented on programmable hardware.

Firstly, we carried out a reliability analysis against SEUs in the configuration memory of a Zynq device. SEUs in configuration memory can induce hardware-architectural faults that may potentially change the structure of the accelerator that has been implemented on the programmable logic. The fault injection campaign targets a hardware-accelerated version of the software convolutional layer that was previously analyzed in the software-based reliability analysis. The campaign consisted of 10,000 fault injection affecting randomly selected bits of the configuration memory.

We found an error rate of 11.05%, a failure rate of 5.12%, and a ratio between failures and errors of 46.33%. Additionally, 0.40% of the injections caused a timeout.

By utilizing the PyXEL framework embedded in FireNN, we gained insight into the randomly selected location for the injection and the error and failure rates associated with them. The results are presented in Table 9.3

Table 9.3 Result of Fault injection campaign of SEUs in CRAM

Resource	Hit (Any)	Hit (Used)	Err. Rate (of Used)	Fail. Rate (of Used)
Routing	4,577	3,584	23.94%	11.43%
LUTs	1,370	1,058	10.49%	3.11%
Block RAMs	493	384	13.54%	7.55%
DSP	432	324	14.81%	7.10%
Flip-Flops	365	286	10.38%	4.89%
Empty	2,337	-	-	-
Others	427	288	9.85%	9.09%

In the second fault injection campaign, we emulated the open-routing fault model. Open faults are the most common error event that occurs in programmable logic devices [90]. From Table 9.3, it can be seen that interconnections are a critical resource for the accelerator under evaluation. To further investigate the issues related to interconnections faults, we randomly injected open-routing faults in the routing of the implemented hardware accelerator. The platform modified the bits related to a specific interconnection to create an open fault in the netlist. This analysis shows how FireNN can be used to inject various and specific fault models in hardware accelerators, either emulated or implemented on programmable hardware.

A fault injection campaign of 10,000 injections was conducted, with the interconnections randomly selected among the programmable routing segments used by design implemented on the hybrid device. This differed from the previous reliability analysis, as it did not hit unused resources. The investigation resulted in an error rate of 59.62% and a failure rate of 40.07%, with a ratio of failures to errors of 67.21%. Furthermore, 2.78% of the fault injections led to timeout events.

Figure 9.6 provides more details on the distribution of detected events for misclassifications and degradations. The results of Figure 9.6(c) suggest that errors induced by open-interconnection fault models are likely to affect all the outputs of the evaluation set. Additionally, the ratio of failures and errors highlights that a very high percentage of them led to misclassification, as shown in 9.6(d) , with a very large part of misclassifications induced by open-interconnection faults affecting a large portion of the outputs.

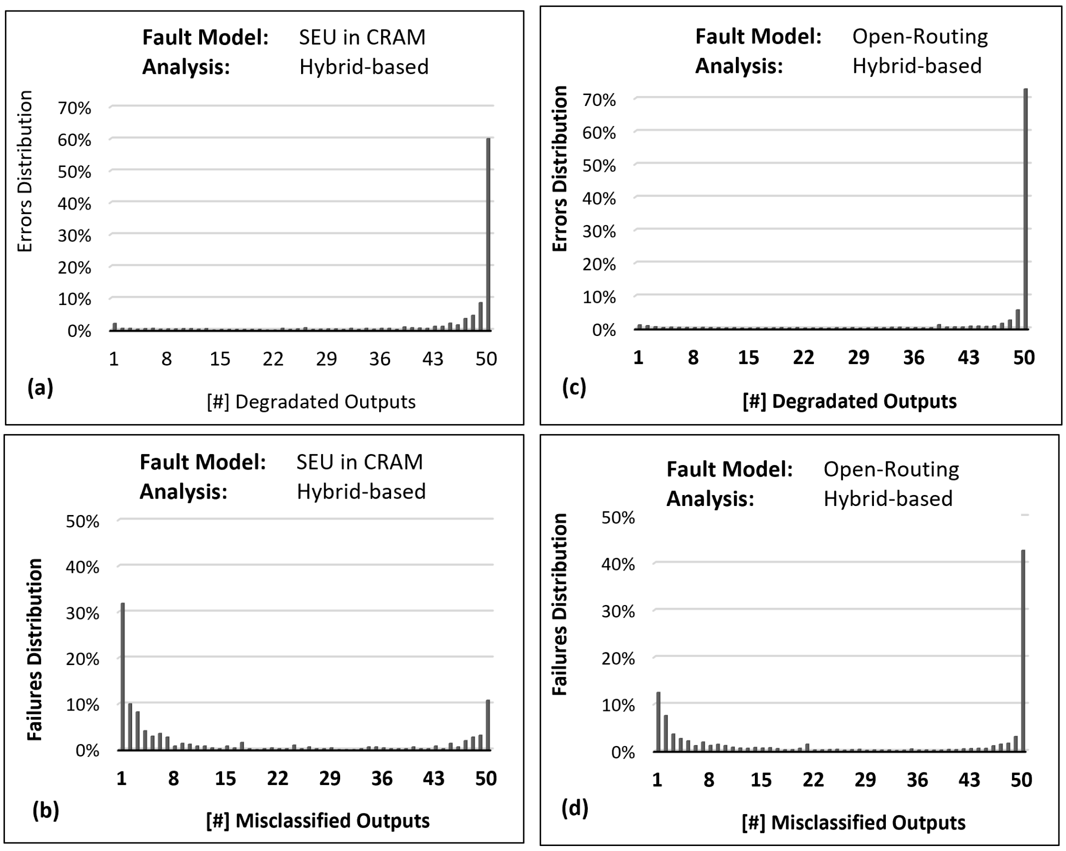


Figure 9.6 Distribution of the degradation (a,c) and misclassification (b,d) categories resulting from emulating SEU in CRAM (a,b) and open-routing fault model (c,d) over the number of outputs experiencing the effect.

FireNN has allowed us to discover that the routing interconnections are the most used resource and the most susceptible to single-event upsets. Through analyzing the error and failure rate associated with this resource, we have gained valuable insight into the hardware domain, which traditional software-based fault injection approaches could not obtain.

9.4 Research Advancements in Reliability Evaluation of Hardware-Acceleration for Neural Networks

Novel methodologies for analyzing errors in hardware-accelerated neural networks have been proposed. After a preliminary analysis evaluating SEU and MBU in CRAM, providing an interesting comparison aiming to highlight how faults in neural network parameters cover only partially the faults that can affect reconfigurable hardware-based accelerators, an evaluation platform based on a reconfigurable system-on-chip has been presented. Based on a novel methodology, the proposed platform exploited reconfigurability for injecting faults in the hardware configured on the programmable logic by manipulating the configuration memory data. The proposed methodology has advantages and characteristics that differentiate it from state-of-the-art. The involvement of a hardware platform in the reliability enables the study of the microarchitectural faults, which are not achievable using software-level simulation approaches usually abstracted from the underlying hardware.

Chapter 10

Conclusions and Future Directions

10.1 Conclusions

This dissertation has proposed and explored methodologies and techniques to enable accurate fault analysis and reliability evaluation in Hardware-Reconfigurable Systems-on-Chip. Through this research, a range of tools and methods have been proposed and developed to allow the analysis, evaluation, and improvement of the reliability of Programmable Hardware-Reconfigurable devices.

Several contributions have been made to the field of hardware-reconfigurable System-on-chips reliability and robustness evaluation against radiation-induced errors. Comprehensive analysis and methodologies involving different types of computational systems, fault models, and techniques have been proposed, showing the potential of these tools and methodologies to analyze critical applications and modules in detail. Methodology and evaluation analysis has been conducted comprehensively, presenting approach and research at different levels, from physical to logical, and using various methods such as fault injection methodologies, accelerated radiation testing, and emulation. With the methodologies and techniques developed in this dissertation, it has been possible to obtain significant insight into the reliability and sensitivity of fault-tolerant systems. It aims to produce substantial steps toward a comprehensive and detailed analysis of the elements composing heterogeneous computational platforms, ensuring more advanced, reliable, and efficient systems.

In particular, methodologies dedicated to evaluating the sensitivity of designs to Single Event Transients (SETs), propagating in the combinational logic and configuration circuitry of programmable devices, have been proposed and supported by developed tools such as PREDA and APES. The analysis of SEUs in configuration memory has also been addressed. Research efforts led to the development of the PyXEl tool to analyze how SEUs in configuration memory affect the design implemented in programmable logic. Additionally, two proton tests were conducted to evaluate the sensitivity of memories of reconfigurable system-on-chips, such as on-chip memory of a Zynq-7020 SoC and 28 nm CMOS and 16 nm FinFET configuration memories. The research has also demonstrated that physical and electrical analysis and radiation test experiments can be used for modeling and evaluating primary radiation-induced effects, such as Single Event Transient and Single Event Upset.

Furthermore, this research has presented methodologies and techniques for evaluating the elements composing a heterogeneous system based on programmable hardware, such as soft and hard microprocessors, interconnection modules, and accelerators, with a particular focus on hardware-accelerated neural network applications, showing the potential of these tools and methodologies to analyze critical applications and modules in detail. Dedicated methodologies and analysis involving actual hardware devices and fault models have been proposed for the various elements composing hardware-reconfigurable systems-on-chip. The robustness of Hard and Soft processing systems has been evaluated considering specific fault models typical of the specific system, such as on-chip memory faults and radiation-induced architectural faults. In particular, radiation-induced architectural faults due to configuration memory corruption highlighted how hardware faults could be propagated from hardware to the software level. It also showed how software mechanisms, such as the operating system exception mechanism, can support identifying hardware faults induced by configuration memory corruption at the software level.

Specific hardware modules implemented in the programmable hardware have been evaluated as well. The criticality of the AXI interconnection core and its contribution to common-mode failure has been highlighted, identifying as it can be a source of silent data corruption even with redundant modules. Finally, a platform for analyzing the reliability of neural networks implemented in programmable hardware has been developed. This research showed how traditional software-level methodologies for assessing neural network reliability cover only partially the faults that can affect reconfigurable hardware-based accelerators, demanding a more comprehensive analysis aware of the hardware on which a model will be implemented.

10.2 Future Directions

The approaches, methodologies, and results presented in this dissertation have explored the potential of Hardware-Reconfigurable devices for fault-tolerant applications and the challenges associated with analyzing fault sensitivity and effects in such systems. The results of this research provide an important foundation for further work in this area and enable many potential directions for future research.

First, further investigations are needed to develop increasingly efficient fault detection, diagnosis, and analysis strategies for Hardware-Reconfigurable SoCs. The efforts in proposing robustness evaluation flow to be applied easily and early in the development flow started in this research need to be extended in order to explore and include more methodologies and approaches, in particular, static analysis approaches, based on a low-level and comprehensive knowledge of the implementation of a design in the configurable hardware of the device, able to predict the point-of-failures and the propagation of faults induced by configuration memory modification in the circuit implemented on programmable hardware are good candidates for a light methodology to be applied early in the design flow.

Second, the deep knowledge of the mapping between faults affecting the configuration memory and the hardware resources and modules implemented on the device developed during this research work can contribute to a more precise and efficient evaluation and enhancement of design robustness based on custom place-and-route solutions that can be early applied evaluated [91].

Finally, more research is needed to develop fault-tolerant systems with high reliability and robustness using Hardware-Reconfigurable SoCs, exploring the potential of using redundant systems and developing new methods for evaluating the reliability and robustness of such systems. In particular, deep and comprehensive fault analysis can be invaluable for increasing the robustness of redundant systems and avoiding cross-domain errors that can significantly compromise the hardening approach based on redundancy [92, 93]. Sensitivity to cross-domain errors can be detected by exploiting the detailed knowledge of the radiation-induced effects on the hardware of implemented designs, which can be reached only through a comprehensive knowledge of how radiation-induced modifications of the configuration memory affect programmed resources.

The knowledge, methodologies, results, and tools developed during this research will be a fundamental basis for future research in these areas.

Bibliography

[1] Xilinx. *7 Series FPGAs Configuration, UG470*. Xilinx.

[2] Xilinx. *UltraScale Architecture Configuration, UG570*. Xilinx.

[3] Robert Baumann and Kirby Kruckmeyer. *Radiation Handbook for Electronics*. Texas Instruments Inc., Texas Instruments, Dallas, Texas, 2020.

[4] Sarah Azimi and Luca Sterpone. Digital design techniques for dependable high performance computing. In *2020 IEEE International Test Conference (ITC)*, pages 1–10, 2020.

[5] V. Ferlet-Cavrois, V. Pouget, D. McMorrow, J. R. Schwank, N. Fel, F. Essely, R. S. Flores, P. Paillet, M. Gaillardin, D. Kobayashi, J. S. Melinger, O. Duhamel, P. E. Dodd, and M. R. Shaneyfelt. Investigation of the propagation induced pulse broadening (pipb) effect on single event transients in soi and bulk inverter chains. *IEEE Transactions on Nuclear Science*, 55(6):2842–2853, 2008.

[6] Alfonso Sánchez-Macián, Pedro Reviriego, Juan Antonio Maestro, and Shanshan Liu. Single event transient tolerant bloom filter implementations. *IEEE Transactions on Computers*, 66(10):1831–1836, 2017.

[7] N. Battezzati, S. Gerardin, A. Manuzzato, D. Merodio, A. Paccagnella, C. Poivey, L. Sterpone, and M. Violante. Methodologies to study frequency-dependent single event effects sensitivity in flash-based fpgas. *IEEE Transactions on Nuclear Science*, 56(6):3534–3541, 2009.

[8] L. Sterpone, N. Battezzati, and V. Ferlet-Cavrois. Analysis of set propagation in flash-based fpgas by means of electrical pulse injection. *IEEE Transactions on Nuclear Science*, 57(4):1820–1826, 2010.

[9] Sana Rezgui, J. J. Wang, Yinming Sun, Brian Cronquist, and John McCollum. Configuration and routing effects on the set propagation in flash-based fpgas. *IEEE Transactions on Nuclear Science*, 55(6):3328–3335, 2008.

[10] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, and T. Turflinger. Digital single event transient trends with technology node scaling. *IEEE Transactions on Nuclear Science*, 53(6):3462–3465, 2006.

[11] Corrado De Sio, Sarah Azimi, and Luca Sterpone. On the evaluation of the pipb effect within sram-based fpgas. In *2019 IEEE European Test Symposium (ETS)*, pages 1–2, 2019.

[12] C. De Sio, S. Azimi, L. Sterpone, and B. Du. Analyzing radiation-induced transient errors on sram-based fpgas by propagation of broadening effect. *IEEE Access*, 7:140182–140189, 2019.

[13] S. Azimi, L. Sterpone, B. Du, and L. Boragno. On the analysis of radiation-induced single event transients on sram-based fpgas. *Microelectronics Reliability*, 88-90:936–940, 2018.

[14] Xilinx. *7 Series FPGAs Configurable Logic Block, UG474*. Xilinx.

[15] Huaguo Liang, Xiumin Xu, Zhengfeng Huang, Cuiyun Jiang, Yingchun Lu, Aibin Yan, Tianming Ni, Yiming Ouyang, and Maoxiang Yi. A methodology for characterization of set propagation in sram-based fpgas. *IEEE Transactions on Nuclear Science*, 63(6):2985–2992, 2016.

[16] L. Sterpone, F. Luoni, S. Azimi, and B. Du. A 3-d simulation-based approach to analyze heavy ions-induced set on digital circuits. *IEEE Transactions on Nuclear Science*, 67(9):2034–2041, 2020.

[17] F. Corno, M.S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *IEEE Design and Test of Computers*, 17(3):44–53, 2000.

[18] C. De Sio, S. Azimi, L. Bozzoli, B. Du, and L. Sterpone. Radiation-induced single event transient effects during the reconfiguration process of sram-based fpgas. *Microelectronics Reliability*, 100-101:113342, 2019.

[19] Prasad Rau, Atul V. Ghia, and Suresh M. Menon. Configuration memory architecture for fpga, U.S. Patent 6222757B1, Apr. 2001.

[20] Carlos H. M. Oliveira, Matheus T. Moreira, Ricardo A. Guazzelli, and Ney L. V. Calazans. Ascend-freepdk45: An open source standard cell library for asynchronous design. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 652–655, 2016.

[21] Ludovica Bozzoli and Luca Sterpone. Mats: An on-line testing approach for reconfigurable embedded memories. In *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2018.

[22] Ludovica Bozzoli, Corrado De Sio, Luca Sterpone, and Cinzia 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.

[23] 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.

[24] Steve Guccione, Delon Levi, and Prasanna Sundararajan. Jbits: Java based interface for reconfigurable computing. In *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, 01 2000.

[25] 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.

[26] 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.

[27] 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.

[28] Zheng Ding, Qiang Wu, Yizhong Zhang, and Linjie Zhu. Deriving an ncd file from an fpga bitstream: Methodology, architecture and evaluation. *Microprocessors and Microsystems*, 37(3):299–312, 2013.

[29] 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.

[30] Ludovica Bozzoli and Luca Sterpone. Comet: a configuration memory tool to analyze, visualize and manipulate fpgas bitstream. In *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, pages 1–4, 2018.

[31] S. Danzeca, G. Spiezia, M. Brugger, L. Dusseau, G. Foucard, R. Garcia Alia, P. Mala, A. Masi, P. Peronnard, J. Soltes, A. Thornton, and L. Viererbl. Qualification and characterization of sram memories used as radiation sensors in the lhc. *IEEE Transactions on Nuclear Science*, 61(6):3458–3465, 2014.

[32] 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.

[33] 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.

[34] 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.

[35] Wei-Tao Yang, Xue-Cheng Du, Yong-Hong Li, Gang He, Chao-Hui nad 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(106):2210–3147, 2021.

[36] Corrado De Sio, Sarah Azimi, Luca Sterpone, and David Merodio Codinachs. Analysis of proton-induced single event effect in the on-chip memory of embedded process. In *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2022.

[37] 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.

[38] Leonardo Passig Horstmann and Antônio Augusto Fröhlich. A fault injection framework for real-time multicore embedded systems. In *2020 X Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8, 2020.

[39] Martin Hiller, Arshad Jhumka, and Neeraj Suri. Propane: An environment for examining the propagation of errors in software. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '02, page 81–85, New York, NY, USA, 2002. Association for Computing Machinery.

[40] Ninghan Tian, Daniel Saab, and Jacob A. Abraham. Esift: Efficient system for error injection. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 201–206, 2018.

[41] Qining Lu, Mostafa Farahani, Jiesheng Wei, Anna Thomas, and Karthik Pattabiraman. Llfi: An intermediate code-level fault injection tool for hardware faults. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 11–16, 2015.

[42] Daniel Oliveira, Vinicius Frattin, Philippe Navaux, Israel Koren, and Paolo Rech. Carol-fi: An efficient fault-injection tool for vulnerability evaluation of modern hpc parallel accelerators. In *Proceedings of the Computing Frontiers Conference*, CF'17, page 295–298, New York, NY, USA, 2017. Association for Computing Machinery.

[43] Felipe Rosa, Fernanda Kastensmidt, Ricardo Reis, and Luciano Ost. A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 211–214, 2015.

[44] R. Velazco, S. Rezgui, and R. Ecoffet. Predicting error rate for microprocessor-based digital architectures through c.e.u. (code emulating upsets) injection. *IEEE Transactions on Nuclear Science*, 47(6):2405–2411, 2000.

[45] Sarah Azimi, Corrado De Sio, Daniele Rizzieri, and Luca Sterpone. Analysis of single event effects on embedded processor. *Electronics*, 10(24), 2021.

[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] Andrew M. Keller, Timothy A. Whiting, Kenneth B. Sawyer, and Michael J. Wirthlin. Dynamic seu sensitivity of designs on two 28-nm sram-based fpga architectures. *IEEE Transactions on Nuclear Science*, 65(1):280–287, 2018.

[48] Matteo Sonza Reorda, Luca Sterpone, and Anees Ullah. An error-detection and self-repairing method for dynamically and partially reconfigurable systems. *IEEE Transactions on Computers*, 66(6):1022–1033, 2017.

[49] Corrado De Sio, Sarah Azimi, Andrea Portaluri, and Luca Sterpone. Seu evaluation of hardened-by-replication software in risc- v soft processor. In *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2021.

[50] Wilfredo Torres-Pomales. *Software Fault Tolerance: A Tutorial*. NASA.

[51] M. Rebaudengo, M. Sonza Reorda, M. Torchiano, and M. Violante. Soft-error detection through software fault-tolerance techniques. In *Proceedings 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (EFT'99)*, pages 210–218, 1999.

[52] 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.

[53] 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.

[54] C. Bolchini, A. Miele, M. Rebaudengo, F. Salice, D. Sciuto, L. Sterpone, and M. Violante. Software and hardware techniques for seu detection in ip processors. *Journal of Electronic Testing*, 24(1):35–44, Jun 2008.

[55] 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. In *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pages 1–3, 2018.

[56] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, 2001.

[57] Dario Mamone, Alberto Bosio, Alessandro Savino, Said Hamdioui, and Maurizio Rebaudengo. On the analysis of real-time operating system reliability in embedded systems. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2020.

[58] I. O. Loskutov, N. D. Kravchenko, V. A. Marfin, P. V. Nekrasov, D. V. Bobrovsky, A. A. Smolin, and A. V. Yanenko. Investigation of operating system influence on single event functional interrupts using fault injection and hardware error detection in arm microcontroller. In *2021 International Siberian Conference on Control and Communications (SIBCON)*, pages 1–4, 2021.

[59] Pablo M. Aviles, Almudena Lindoso, Jose A. Belloch, Mario Garcia-Valderas, Yolanda Morilla, and Luis Entrena. Radiation testing of a multiprocessor macrosynchronized lockstep architecture with freertos. *IEEE Transactions on Nuclear Science*, 69(3):462–469, 2022.

[60] Luis Alberto Aranda, Nils-Johan Wessman, Lucana Santos, Alfonso Sánchez-Macián, Jan Andersson, Roland Weigand, and Juan Antonio Maestro. Analysis of the critical bits of a risc-v processor implemented in an sram-based fpga for space applications. *Electronics*, 9(1), 2020.

[61] Andrew Elbert Wilson and Michael Wirthlin. Neutron radiation testing of fault tolerant risc-v soft processor on xilinx sram-based fpgas. In *2019 IEEE Space Computing Conference (SCC)*, pages 25–32, 2019.

[62] Wassim Mansour and Raoul Velazco. Seu fault-injection in vhdl-based processors: A case study. In *2012 13th Latin American Test Workshop (LATW)*, pages 1–5, 2012.

[63] Chang Cai, Shuai Gao, Peixiong Zhao, Jian Yu, Kai Zhao, Liewei Xu, Dongqing Li, Ze He, Guangwen Yang, Tianqi Liu, and Jie Liu. See sensitivity evaluation for commercial 16 nm sram-fpga. *Electronics*, 8(12), 2019.

[64] Andrea Portaluri, Sarah Azimi, Corrado De Sio, Daniele Rizzieri, and Luca Sterpone. On the reliability of real-time operating system on embedded soft processor for space applications. In Martin Schulz, Carsten Trinitis, Nikela Papadopoulou, and Thilo Pionteck, editors, *Architecture of Computing Systems*, pages 181–193, Cham, 2022. Springer International Publishing.

[65] Sarah Azimi, Corrado De Sio, Andrea Portaluri, Daniele Rizzieri, Eleonora Vacca, Luca Sterpone, and David Merodio Codinachs. Exploring the impact of soft errors on the reliability of real-time embedded operating systems. *Electronics*, 12(1), 2023.

[66] 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.

[67] Fabio Benevenuti and Fernanda Lima Kastensmidt. Analyzing axi streaming interface for hardware acceleration in ap-soc under soft errors. In Nikolaos Voros, Michael Huebner, Georgios Keramidas, Diana Goehringer, Christos Antonopoulos, and Pedro C. Diniz, editors, *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pages 243–254, Cham, 2018. Springer International Publishing.

[68] Corrado De Sio, Sarah Azimi, and Luca Sterpone. On the evaluation of seu effects on axi interconnect within ap-socs. In André Brinkmann, Wolfgang Karl, Stefan Lankes, Sven Tomforde, Thilo Pionteck, and Carsten Trinitis, editors, *Architecture of Computing Systems – ARCS 2020*, pages 215–227, Cham, 2020. Springer International Publishing.

[69] 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.

[70] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, 2014.

[71] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung. X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7510–7514, 2014.

[72] Michel van Lier, Luc Waeijen, and Henk Corporaal. Bitwise neural network acceleration: Opportunities and challenges. In *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–5, 2019.

[73] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing.

[74] Boyang Du, Sarah Azimi, Corrado de Sio, Ludovica Bozzoli, and Luca Sterpone. On the reliability of convolutional neural network implementation on sram-based fpga. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, 2019.

[75] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, page 65–74, New York, NY, USA, 2017. Association for Computing Machinery.

[76] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[77] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[78] Brunno F. Goldstein, Sudarshan Srinivasan, Dipankar Das, Kunal Banerjee, Leandro Santiago, Victor C. Ferreira, Alexandre S. Nery, Sandip Kundu, and Felipe M. G. França. Reliability evaluation of compressed deep learning models. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 1–5, 2020.

[79] Mohamed A. Neggaz, Ihsen Alouani, Pablo R. Lorenzo, and Smail Niar. A reliability study on cnns for critical embedded systems. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 476–479, 2018.

[80] Cristiana Bolchini, Luca Cassano, Antonio Miele, and Alessandro Toschi. Fast and accurate error simulation for cnns against soft errors. *IEEE Transactions on Computers*, pages 1–14, 2022.

[81] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech. Analyzing and increasing the reliability of convolutional neural networks on gpus. *IEEE Transactions on Reliability*, 68(2):663–677, 2019.

[82] Corrado De Sio, Sarah Azimi, and Luca Sterpone. An emulation platform for evaluating the reliability of deep neural networks. In *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–4, 2020.

[83] Corrado De Sio, Sarah Azimi, and Luca Sterpone. Firenn: Neural networks reliability evaluation on hybrid platforms. *IEEE Transactions on Emerging Topics in Computing*, 10(2):549–563, 2022.

[84] 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.

[85] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[86] Pynq. http://www.pynq.io/. Accessed: 2023-01-01.

[87] Xilinx. *High-Level Synthesis, UG902*. Xilinx.

[88] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[89] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[90] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Luca Sterpone. Assess: A simulator of soft errors in the configuration memory of sram-based fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(9):1342–1355, 2014.

[91] Eleonora Vacca, Corrado De Sio, and Sarah Azimi. Layout-oriented radiation effects mitigation in risc-v soft processor. In *Proceedings of the 19th ACM International Conference on Computing Frontiers*, CF '22, page 215–220, New York, NY, USA, 2022. Association for Computing Machinery.

[92] Andrea Portaluri, Corrado De Sio, Sarah Azimi, and Luca Sterpone. A new domains-based isolation design flow for reconfigurable socs. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[93] Andrea Portaluri, Corrado De Sio, Sarah Azimi, and Luca Sterpone. Seu mitigation on sram-based fpgas through domains-based isolation design flow. In *2021 21th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, pages 1–4, 2021.