DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Error Injection Study for a SpaceFibre In-Orbit Demonstrator

CARLOS ABAD GARCÍA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Error Injection Study for a SpaceFibre In-Orbit Demonstrator

Master Thesis

Carlos Abad García

Examiner: Johnny Öberg
University Supervisor: Kalle Ngo
Company Supervisor: Manuel Sánchez Renedo

A thesis presented for the degree of
Master of Science in ICT Innovation - Embedded Systems

Electronics and Embedded Systems - KTH Royal Institute of Technology
Digital Electronics Group - Thales Alenia Space España
September 2020

Error Injection Study for a SpaceFibre In-Orbit Demonstrator

Master Thesis - KTH Royal Institute of Technology

Carlos Abad García

Abstract

The space electronics sector is shifting towards the New-Space paradigm, in which traditional space-qualified and expensive components and payloads are replaced by commercial off-the-shelf (COTS) alternatives. This change in mentality is accompanied by the development of inexpensive cubesats, lowering the entry barrie in terms of cost, enabling an increase in scientific research in space. However, also well-established and resourceful spacecraft manufacturers are adopting this trend that allows them to become more competitive in the market.

Following this trend, Thales Alenia Space is developing R&D activities using COTS components. One example is the SpaceFibre In-Orbit Demonstrator, a digital board integrated in a cubesat payload that aims to test two Intellectual Property blocks implementing the new ECSS standard for high-speed onboard communication.

This thesis presents the necessary steps that were taken to integrate the firmware for the demonstrator's Field-Programmable Gate Array (FPGA) that constitutes the main processing and control unit for the board. The activity is centered around the development of a Leon3 System-on-Chip in VHDL used to manage the components in the board and test the SpaceFibre technology.

Moreover, it also addresses the main problem of using COTS components in the space environment: their sensitivity to radiation, that, for a FPGA results in Single-Event Upsets causing the implementation to malfunction, and a potential failure of the mission if they are not addressed. To accomplish the task, a SEU-emulation methodology based in partial reconfiguration and integrating the state of the art techniques is elaborated and applied to test the reliability of the SpaceFibre technology.

Finally, results show that the mean time between failures of the SpaceFibre Intellectual Property Block using a COTS FPGA is of 170 days for Low Earth Orbit (LEO) and 2278 days for Geostationary Orbit (GEO) if configuration memory scrubbing is included in the design, enabling its usage in short LEO missions for data transmission. Moreover, tailored mitigation techniques based on the information gathered from applying the proposed methodology are presented to improve the figures.

Error Injection Study for a SpaceFibre In-Orbit Demonstrator

Master Thesis - KTH Royal Institute of Technology

Carlos Abad García

Sammanfattning

Rymdsektorn börjar luta mot "the New-Space paradigm", i vilken traditionella och dyra rymd-kvalificerade komponenter och laster börjar bytas ut kommerciella-från-hyllan (eng. Commercial-off-the-Shelf - COTS) alternativ. Denna förändring i mentalitet ackompanjeras av utvecklingen av billiga CubeSats som sänker entré-kostnaden för vetenskaplig forskning i rymden. Även väletablerade och resursstarka rymdfarkost-tillverkare har anammat denna trend vilket låter dem bli mer konkurrenskraftiga på marknaden.

För att följa trenden så utför Thales Alenia Space R&D utecklingsaktiviteter med COTS komponenter. Ett exempel är SpaceFibre In-Orbit Demonstrator, a digitalt kort integrerat i en CubeSat payload som ämnar testa två s.k. Intellectual Property (IP) konstruktioner som implementerar den nya ECSS standarde för hög-hastighets kommunikation ombord.

Denna avhandling presenterar de nödvändiga stegen för att integrera firmware för demonstratorns programmerbara FPGA-krets (eng. Field-Programmable Gate Array - FPGA) som fungerar som kortets huvudsakliga beräknings- och styrenheten. Aktiviteten är centrerad kring utvecklingen av ett Leon3 System-on-Chip i VHDL för att hantera och managera komponenterna på kortet och testa SpaceFibre-teknologin.

Vidare adresserar den också huvudproblemet med att använda COTS-komponenter i rymdmiljö: deras känslighet för strålning, vilket i en FPGA kan resultera i s.k. Single-Event-Upsets, vilket orsakar fel i implementeringen och ett potentiellt misslyckande av uppdraget om de inte adresseras. För att åstadkomma detta, utarbetas och appliceras en SEU-emuleringsmetodik baserad på partiell rekonfigurering för att testa tillförlitligheten hos SpaceFibre-tekniken.

Slutligen visar resultaten att den genomsnittliga tiden mellan fel (eng. Mean-Time Between Failure - MTBF) för SpaceFibre IP blocken i en COTS FPGA är 170 dagar för låg omloppsbana och 2278 dagar för Geostationär omloppsbana om scrubbing-tekniker implementeras. Skräddarsydda mitigations-tekniker, baserade på den insamlade informationen av tillämpningen av den föreslagna metoden, föreslås för att förbättra siffrorna.

Acknowledgments

I would like to thank my family and friends for staying close and their support, my colleagues at Thales Alenia Space in Spain, especially Manu, for their help, Johnny Öberg and Kalle Ngo from KTH for the opportunity to carry out this work, and all my friends from Stockholm and Berlin for the great times we had. Thank you.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

This document requires readers to be familiar with the following terms and abbreviations.

FPGA	Field Programmable Gate Array
IOD	In Orbit Demonstrator
RISC	Reduced Instruction Set Computer
AMBA	Advanced Microcontroller Bus Architecture
SoC	System on Chip
GPL	General Public License
DMIPS	Dhrystone Million Instructions per Second
CFI	Common Flash Memory Interface
SRAM	Static Random Access Memory
RAM	Random Access Memory
CMOS	Complementary Meta-Oxide-Semiconductor
DSP	Digital Signal Processing
CLB	Configurable Logic Block
LUT	Look-up Table
SerDes	Serializer-Deserializer
LET	Linear Energy Transfer
TID	Total Ionizing Dose
SEE	Single Event Effect
SEL	Single Event Latch-up
SEB	Single Event Burnout
SEGR	Single Event Gate Rupture
SEU	Single Event Upset
SBU	Single Bit Upset
MBU	Multi Bit Upset
SECDEC	Single Error Correction Double Error Code
ECC	Error-Correcting Code
CRC	Cyclic Redundancy Checks

SEM	Soft Error Mitigation
BER	Bit Error Rate
CRAM	Configuration Memory RAM
DUT	Device Under Test
FSM	Finite State Machine
PC	Personal Computer
FMC	FPGA Mezzanine Card
CRC	Cyclic Redundancy Check
QoS	Quality of Service
DVF	Design Vulnerability Factor
CI	Confidence Interval
CIR	Relative Confidence Interval
MTBF	Mean Time Between Failures
ASIC	Application-Specific Integrated Circuit
COTS	Commercial off-the-shelf
IP	Intellectual propery
AXI	Advanced eXtensible Interface
AHB	AMBA High-performance Bus
APB	Advanced Peripheral Bus
GPL	GNU General Public License
HDL	Hardware Description Language
DPR	Dynamic Partial Reconfiguration
BRAM	Block Random Access Memory
SET	Single-Event Transient

Chapter 1

Introduction

Nowadays, the space sector plays a major role in scientific experiments, with ambitious observation programmes that monitor not only land and sea, but also integrate complex instruments such as multispectral cameras and synthetic aperture radars, or even more complex sensors that allow to gather valuable information about our environment. A good example is that less than one year ago, in Sevilla (Spain), the European Space Agency (ESA) comitted to the biggest ever budget to be spent in missions to come, that will allow to improve the understanding of both our planet and the universe.

From the engineering point of view, assembling these missions is not only an extraordinary effort, which requires a collective endeavor by multiple teams around the world, but also poses the challenge of developing the necessary technologies. Apart from the instruments themselves, the elements surrounding them also play a key role in these missions. High performance processing and communications onboard the payloads are necessary, since "raw" instrument data is usually too heavy to be sent back to earth directly. There are two key technologies that allow to overcome this challenge:

- **Field-Programmable Gate Arrays**: FPGAs are programmable integrated circuits that allow to create almost any digital circuit by using Lookup Tables (LUTs) that implement logic functions and programmable interconnections. They offer a processing performance to power consumption ratio of about one to two orders of magnitude higher than processors, and, even though they are not as efficient or powerful as application-specific integrated circuits (ASICs), FPGAs do not incur in the extremely high non-recurrent engineering costs derived from the verification and fabrication of an ASIC. For that reason, FPGAs are usually the chosen platform when high computing power is required and not a significant amount of units will be manufactured. Along the years, several radiation-hardened FPGAs specific for the space environment have been developed. However, they are either less performant than their non radiation-hardened counterparts—due to containing less logic, since a significant part of the silicon is used to protect the functioning transistors—or significantly more expensive. That is why, with the "new-space" paradigm, commercial off-the-shelf (COTS) components are starting to be used in shorter less critical. The protection to the space environment is no longer provided by the components themselves, but by additional conceptual layers of protection implemented at both system level and within the processing elements. This paradigm has not reached to the most critical scientific missions yet, but it is a matter of time, with the performance gap between COTS and space-qualified components, as well as the

biggest FPGA manufacturer—Xilinx—using a COTS non-hardened FPGA as their latest space-grade FPGA: the RT Kintex Ultrascale, which is essentially a Kintex Ultrascale KU060 with a ceramic flatpack packaging.

- **High-speed onboard communications**: with the huge data rate created by the aforementioned instruments, as well as the processing rate of the multiple computation units and storage elements, the need for board-to-board communication able to transfer data at gigabits per second is a common requirement in different space missions. This demand is partially covered by the latest high-speed serial transceiver technologies, that use serializer-deserializer circuits—sometimes several of them in parallel lanes—allowing to achieve the demanded speed. However, there was no network standard designed for its usage in space until the recent development of SpaceFibre, which integrates a network architecture specifically designed for space, and with the same application interface as SpaceWire—a current standard for onboard communications at lower speeds supported by ESA, NASA and JAXA among others. SpaceFibre provides Quality of Service (QoS) mechanisms over the physical layer, allowing for flow-control and error recovery and management among other more sophisticated capabilities at an extremely high data-rate. The standard was published in May 2019, by the University of Dundee under ESA funding, and some Intellectual Property (IP) blocks that implement the protocol have been developed already.

1.1 Motivation

This work is framed in a collaboration between Thales Alenia Space Spain and a commercial space company motivated by these two factors. The aim of the project is the development of an experimental digital card, the SpaceFibre In-Orbit Demonstrator (IOD) that integrates a current generation COTS FPGA based on Static Random-Access Memory (SRAM), the kind that is expected to be adopted in the space sector. The FPGA contains two SpaceFibre IPs developed by Cobham Gaisler/ESA and STAR-Dundee integrated in a System-on-Chip (SoC) based on the LEON processor that manages all the elements in the board needed to test and validate the SpaceFibre technology as well as the FPGA. All stakeholders involved have great interest into testing and accumulating hours of flight into the different components, constituting a relevant research and development activity.

Apart from testing the components in-flight with the demonstrator, with the new-space paradigm and the shift to COTS components, it is of capital relevance to characterize their behavior when subject to radiation before flight, in accelerated campaigns that emulate the effects of the space environment and allow to gather data expected for the mission. For particular components, this is done by performing costly accelerated radiation campaigns, in which a device is exposed to a controlled radiation source, while being monitored for failures. However, as latest campaigns show, for FPGAs, there is a noticeable difference on the failure rates depending on the architecture of the circuit subject to radiation. Apart from the countless implementations that an FPGA allows due to its programmable nature, the need to test the mitigation techniques used for a particular mission make the usage of radiation campaigns impractical, since one separate campaign for each of them would be necessary.

Hence, a research effort is being carried out to emulate the effects of radiation by injecting SEUs in the device to substitute radiation campaigns. It is a relatively new field, and still has not been applied as a validation step in a real implementation, but its potential in saving costs is immense, and valuable contribution can be made by designing a methodology that gathers the state-of-the art techniques and applying it to a relevant implementation.

1.2 Purpose and goals

There are several goals to be achieved for this work:

1. Development of the firmware for the SpaceFibre IOD implemented in the demonstrator board integrating both IPs with the LEON3 SoC and high-speed transceivers
2. Validation of the firmware and support for the integration of the application software in charge of running the recursive tests for the different components
3. Performance of a literature review of the state of the art in SEU emulation
4. Development of a state of the art framework and methodology for SEU emulation. A key step for achieving this goal is to locate a particular design within the FPGA configuration memory
5. Application of the developed methodology to the SpaceFibre IP to obtain results, interpret them and obtain reliability metrics as well as propose recommendations to reduce its SEU vulnerability

The purpose is to contribute to the creation of the SpaceFibre IOD with COTS components, as well as to show the effectiveness of SEU emulation by creating and applying a state of the art injection techniques to the SpaceFibre IP, which will also contribute to the development of the SpaceFibre technology itself and the adoption of COTS components in the space sector.

1.3 Research Questions

The aim then, is to answer to the following research questions:

- How to integrate the SpaceFibre technology in a LEON3 SoC?
- Is the combination of SpaceFibre with a COTS FPGA ready for a commercial mission?

1.4 Research Methodology

To answer these questions, this project follows an inductive approach to carry out the firmware and integrate it into the demonstrator board as well as for the development of the injection methodology. Both elements were carefully designed from the theoretical grounds and calculations, following a qualitative research methodology [5]. However, the

second part of the project, that constitutes the injection campaign and interpretation of the results is inherently experimental and deductive. For this part, the designed methodology is applied in different experiments, collecting a large dataset and interpreting the results from the statistical point of view, using a quantitative methodology.

Chapter 2

Background

This chapter introduces the previous research and literature that are the foundations of this work, as well as the necessary concepts to understand it.

2.1 AMBA bus, GRLIB and LEON3

One of the main contributions from this thesis is the implementation of the firmware used for the SpaceFibre IOD board developed in Thales Alenia Space. The firmware, implemented in a COTS Static Random Access Memory (SRAM) FPGA is based on the LEON3 processor used as a soft-core to execute the control software, along with several peripheral controllers from the GRLIB IP library that allow the board to perform its desired functionality. The SoC uses the Advanced Microcontroller Bus Architecture (AMBA) 2.0 bus to connect to all components of the System on Chip.

2.1.1 AMBA bus

The AMBA bus is an open standard developed by ARM Ltd. for the interconnection of on-chip units in a SoC. It is currently the de-facto standard interface used for the development of hardware IPs in the space sector. There are several interfaces in the AMBA specification, where the Advanced Extensible Interface (AXI), AMBA High-performance Bus (AHB) and Advanced Peripheral Bus (APB) are the most widely used. This thesis uses the two latter for the interconnection of the SoC elements. Both buses are connected to a controller and arbiter that maps the control and status registers of the peripherals to the processor's memory space, with the memory controller also connected to the AHB bus. Both buses are technology-independent and designed for portability. Since they are sometimes implemented in FPGAs, where high impedance internal buffers are not available, they are implemented in the form of a multiplexed bus: each master drives the bus signals, and the output of the current bus master is selected by the multiplexers and sent to the slaves input. Similarly, the output of the active slave is selected and sent to the masters as shown in figure 2.1.

The AHB bus is a high-performance multi-master bus that allows for burst reads and writes. It is used for peripherals that need to frequently transmit and receive larger amounts of data, such as the memory controller —managing the transactions between the processor and the SRAM and flash PROM memories—. The bus is multi-master and multi-slave, controlled by a global arbiter. The arbiter receives the address information

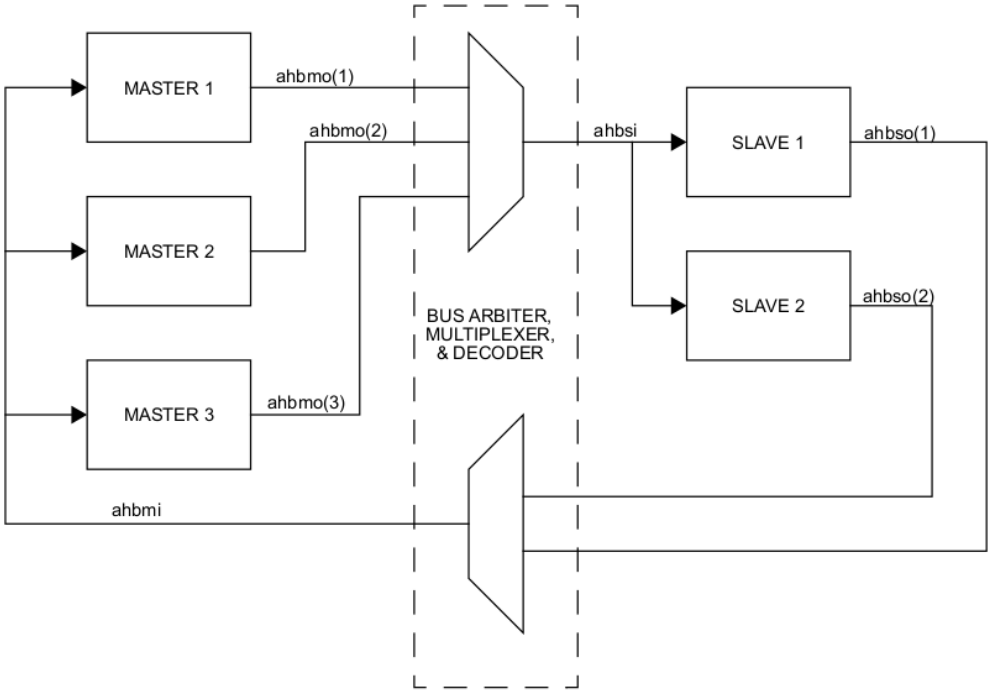

Figure 2.1: AMBA bus master and slave multiplexing [1]

needed to communicate with each master and, by means of a *grant* signal, allows one master at a time to have control over the bus. Similarly, the slaves provide addressing information, and have one dedicated signal to access them while they are selected.

The APB bus, on the other hand, is controlled through an AHB-to-APB bridge, which is an AHB slave and the only APB master in the bus. It is a low-cost bus—in terms of logic— without arbiter used for low-bandwidth and simpler peripherals, such as timers, serial ports and interrupt controllers.

2.1.2 LEON3

The LEON3 processor [6] was designed and developed by Cobham Gaisler A.B., a Swedish company founded by Jiri Gaisler, the creator of the Leon architecture. It is based on the SPARC V8, a Reduced Instruction Set Computer (RISC) architecture, and offered in GRLIB under a GNU General Public License (GPL) as synthesizable VHDL. It uses an AMBA AHB 2.0 interface, which is focused towards the development of SoC designs. Besides, it offers advanced features such as a 7-stage pipeline, configurable caches, debugging support, high performance —1.4 Dhrystone Million Instructions per Second (DMIPS) for each MHz— and it is highly configurable through means of VHDL generics.

2.1.3 GRLIB

One of the key factors for the success of the LEON3 adoption in the space industry is that, along with it, Cobham Gaisler provides an extensive open-source IP library. All IPs come with an AMBA 2.0 compliant interface, and can communicate with a LEON3-based SoC seamlessly thanks to the included AHB and APB controllers. With the exception of

the SpaceFibre integrated in this work, the rest of the components used to form the SoC that composes the firmware for the flight board are directly integrated from GRLIB and freely available.

In order to integrate a GRLIB IP into a design, it is enough to include the necessary library into the compilation of the Hardware Description Language (HDL), and to declare the IP as a component and connect the AMBA signals at the top level of the design. Customization is completely managed by generics. The IPs are also easily managed from the software running on the LEON3, since their status and control registers are memory-mapped in the processor's address space. Instructions to install and use the library, as well as documentation on how the AMBA bus works are detailed in [1]. Every IP available under GPL license is documented in [7].

2.1.4 Additional tools

Besides GRLIB, Cobham Gaisler provides several tools to facilitate the development of SoCs. The one that was used the most is GRMON2 [8], a debugging tool that communicates the debug-support unit of the processor, providing useful services such as obtaining system plug-and-play information, displaying memory contents for the whole address-space and writing to memory in the whole address-space. This allows manual set-up of peripherals and preliminary testing without the need for any kind of software. Likewise, support for Common Flash Memory Interface (CFI) commands is included, facilitating to erase and program the flash rom used to boot the processor. Similarly, it also includes the capability to load software code to the RAM memory and execute it along with the GDB tool for debugging.

Finally, a cross-compiler to generate LEON3 binaries as bare-metal software —i.e., that runs without an operating system— (Bare-C Cross-Compiler) [9] and boot images(MKPROM2) [10] facilitated the work carried out in this thesis.

2.2 Radiation upsets on SRAM-based FPGAs

The data-processing rate and flexibility that modern space missions require is causing an increase in the usage of onboard Field Programmable Gate Arrays (FPGAs). They provide the necessary flexibility, allowing for the implementation of soft-processors and providing the re-programmable capabilities of the logic fabric. Besides, their development cost is notably lower than using ASICs because of the low volume of production inherent to space industry. All onboard components are exposed to the harsh space environment. That includes strong vibrations, a wide span of temperatures and the impact of ionizing particles. The latter is the one in which this work will focus. This section provides a brief introduction to FPGA technologies and the impacts of radiation in such devices, including previous research in emulating Single Event Upsets (SEUs).

This document focuses on SRAM-based FPGAs, particularly in the Kintex-Ultrascale KU060 FPGA from Xilinx, since the injection testing of the SpaceFibre is done on this device to provide the capabilities needed to implement high-throughput applications. Besides, Xilinx has recently released the space-grade (radiation-tolerant) version of the same

architecture [11]. Therefore, it is expected to be the reference FPGA for future high-throughput applications. Later in this section, the architecture of this FPGA will be introduced as well as the most relevant results from previous radiation campaigns.

2.2.1 FPGA technologies

FPGAs are electronic circuits that implement a matrix of programmable logic, in the form of Look-up Tables (LUTs) that implement logic functions, and programmable interconnection blocks conceived to be programmed by the user (not during manufacture) to apply the desired function. Depending on the technology used to implement the programmable logic, there are three major technologies of FPGAs:

- **Anti-fuse FPGAs:** use multiplex-based logic, and need the smallest switch interconnection size. They incur the lowest power consumption and area as well. Despite being only one-time programmable, they are immune to the effects of radiation, and still commonly used in really harsh environments, such as nuclear installations and spacecrafts or satellites [12]. However, they usually provide a small amount of equivalent logic gates, thus not being suitable for the most complex implementations and used only for the most safety-critical operations. Some examples are the RTAX series from Microsemi.

- **Flash-based FPGAs:** re-programmable FPGAs for which the configuration memory is implemented in a flash technology. The contents of the LUTs implementing logic functions, as well as the values that dictate how the programmable interconnect is wired, are stored in a flash configuration memory. Inherent to this technology is an almost non-existent susceptibility to single-event effects. On top of that, the flash memory is non-volatile, and they do not need to be re-programmed after a re-boot. On the other hand, due to the technology used, re-programming a flash FPGA is more expensive in terms of power and time, and the technology node used is older and less efficient than their SRAM counterparts. Besides, their tolerance to the Total Ionizing Dose (TID) and Single-Event Latchup (SEL) is also an issue [13]. Some examples are the ProASIC series FPGAs from Microsemi.

- **SRAM-based FPGAs:** re-programmable FPGAs for which the configuration memory is implemented in SRAM technology, more precisely, Complementary metal-oxide-semiconductor (CMOS) Configuration Latches (CCLs) [14]. Even though the SRAM interconnect switches are the most expensive in terms of area [12], the usage of newer CMOS nodes fairly compensates it, allowing for higher scale integration, density and better efficiency than the flash-based FPGAs. Technology advantage makes their usage grow, since they are able to implement far more complex functions than their counterparts. Besides, they are less expensive to program, allowing for dynamic *runtime* partial reconfiguration (DPR) [15] used to modify the functionality of part of the FPGA while it is functioning. On the other hand, since the CMOS node they use is smaller, they have proven to be more susceptible to the effects of radiation, as will be detailed in section 2.2.6. Furthermore, the volatile nature of SRAM technology cause the necessity of storing the programming bitstream in an external flash memory, introducing potential security breaches to the bitstream that are usually overcome with encryption.

2.2.2 KU060 FPGA structure

This section introduces the basic elements of the KU060 architecture, the SRAM FPGA used for the injection study. It is relevant to understand the elements that it's composed of to better understand the effects that radiation may cause.

The Ultrascale Architecture is focused in high-troughput applications, using a 20 nm technology. Particularly, the Kintex Ultrascale architecture offers an optimized performance per watt [16], including high-throughput Serializer-Deserializer (SerDes), dedicated Digital Signal Processing (DSP) blocks, block ram memory, clock resources and Configurable Logic Blocks (CLBs), the latter being the main building-block for general-purpose logic implementation. Each CLB, grouped physically in a slice, is formed by eight 6-input LUT (each of them configurable as two 5-input LUT), 16 flip-flops, distributed memory and shift-register logic (allowing to implement distributed RAM in the SLICEM slice) as well as wide multiplexers to connect the elements of the CLB [14].

The elements within the FPGA are organized in columns. The most common structure is the one shown in figure 2.4, with one slice on each side of a routing switchbox, used to interconnect the different elements of the FPGA. One clock region is a tile formed by several columns of the same resource that span 60 rows of CLB columns, 24 DSP columns and 12 BRAM columns [17]. Clock regions are divided by horizontal and vertical clock routing and distribution tracks. Besides, the FPGAs also include a core column—integrating the System Monitor, configuration and PCIe blocks—as well as dedicated I/O banks, including high-performance GTH transceivers, I/O logic management and global clock buffers. The device view offered by Vivado, showing 4 clock regions in the KU060 device is depicted in figure 2.3

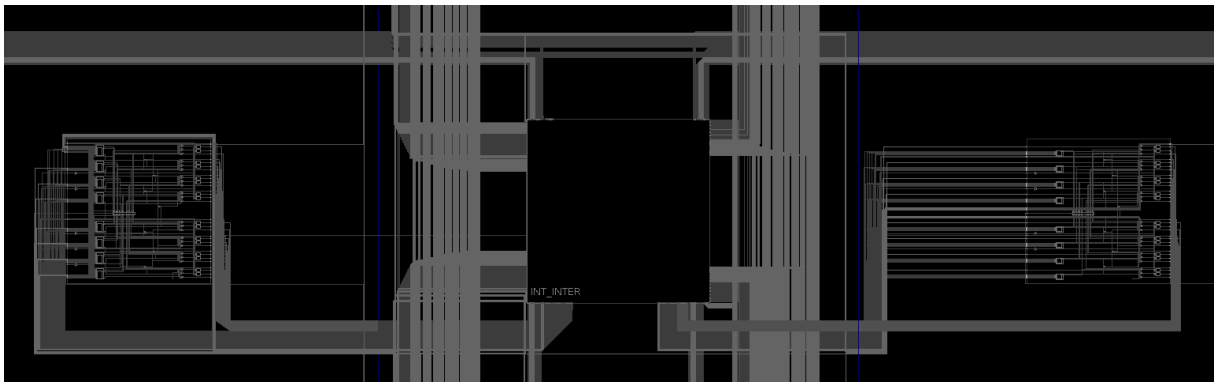

Figure 2.2: Two CLBs and one routing switchbox of the KU060 FPGA

2.2.3 FPGA configuration

The configuration of the FPGA for flash and SRAM FPGAs is stored in the configuration memory (CRAM) of the FPGA. It contains the logic tables for all LUTs, routing resources, interconnects, shift-registers, and flip-flops inside the FPGA, and is stored in a distributed fashion inside configuration latches or the distributed memory elements themselves [18]. The other type of memory conforming a memory space in the FPGA is the Block RAM (BRAM) that provides the main accessible memory. It is utilized to store

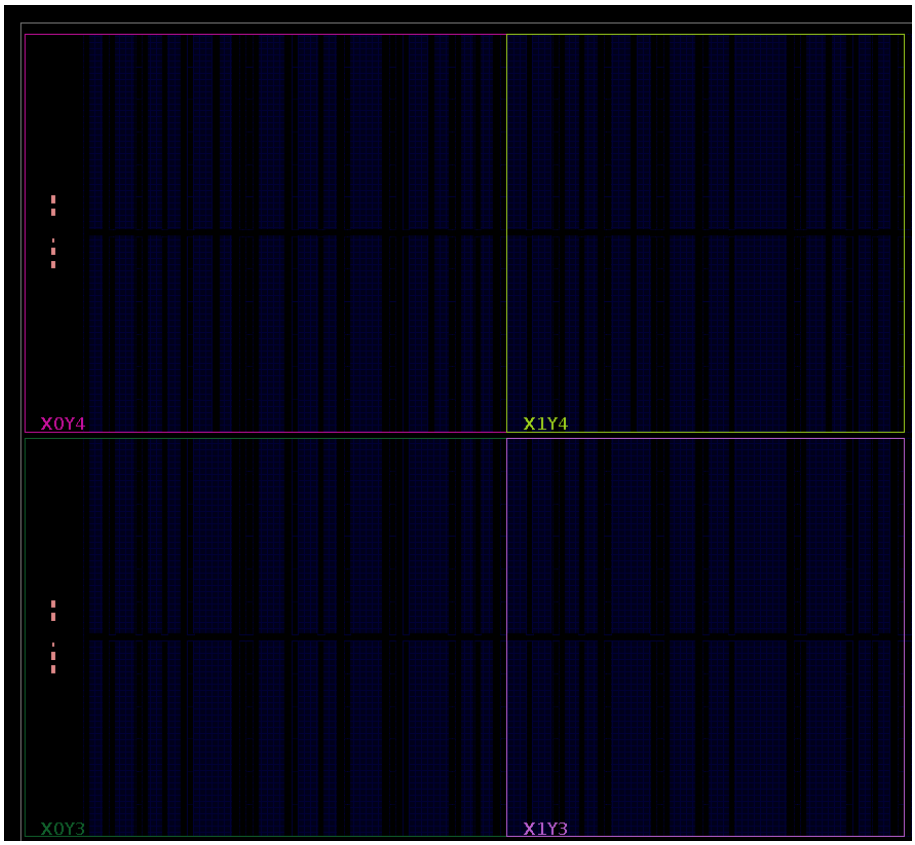

Figure 2.3: Device view of the KU060 showing clock regions and columns within

user data, and is stored in RAM blocks with several customization parameters.

The types of FPGAs mentioned before are configured using a bitstream, that is, the binary file that contains the configuration contents, along with all the necessary commands used to load it into the configuration memory of the device. For volatile SRAM FPGAs, support circuitry is included externally to launch the configuration automatically when the board is booted. For all Xilinx devices, including the targeted FPGA for this work, the minimum addressable unit of configuration is a frame, which in the Ultrascale technology is composed of 123 words of 32 configuration bits. The total amount of frames composing the configuration memory is 37651 [19].

Each configuration bit affects the functionality implemented by the FPGA in some way. Thus, if any value of configuration is changed, its functionality is modified. In order to obtain the configuration bits that are utilized by the circuit, Xilinx offers the essential bits technology. The essential bits are defined as the ones used to define the hardware in the FPGA, and can be automatically reported during the elaboration of the bitstream in the form of a mask file: the ".ebd" file. This file has the same number of bits—in the form of characters, one representing each bit—as the configuration memory, excluding the contents of the BRAM memories. Besides, the dynamic soft values stored in shift-registers and flip-flops are never marked as essential, since their values may change during execution. Thus, the ebd file marks the configuration bits that belong to the design and must remain static for the correct functioning of the circuit. For the sake of simplicity, these bits are denominated configuration bits for the rest of this work.

Unlike the BRAM and registered values, configuration bits are not easily accessible for the user. When the circuit is functioning, there are three ways to access configuration memory: through JTAG, which is extremely slow, through the SelectMAP port, that needs an external device connected to the selectMAP pins to manage the communication, and the Internal Configuration Access Port (ICAP), which can be accessed by the internal logic. Apart from the JTAG connectivity, which is used to program the FPGA in most evaluation boards in the initial stages of a design by using a PC, the most used interfaces are SelectMAP for programming from a flash memory, or using external scrubbing and ICAP for partial reconfiguration and internal scrubbing applications. Regardless of the speed and internal or external nature of the ports, similar commands are issued to read and write configuration memory through them. A major difference is that, when using the ICAP port, all dynamic values stored in BRAM and registers are masked [19], so their values cannot be read. Besides, in the ebd file, these bits will never appear as essential. For more information on the Ultrascale family configuration, the reader is referred to [16].

Frame addressing

There are two ways to locate a particular frame within the configuration memory in Xilinx FPGAs: the physical address, which is related to the position in the FPGA, and the linear address related to the position in the configuration memory. The physical address is composed by the following fields [19]:
Where the row address refers to the row in clock regions in which the frame is located,

Bits	25:23	22:17	16:7	6:0
Content	Block type	Row address	Column address	Minor address

Table 2.1: Physical frame address format

from bottom to top and the column address refers to the major column formed by the same resource type, ordered from left to right. The minor address is used to select a frame within a major column. More details are provided in section 4.3.1.

The linear address is composed translating the position of the frame in the bitstream according to the configuration memory increasing sequentially. Its format is shown in the following table:

Bits	28-12	11-5	4-0
Content	Frame linear address	Word address	Bit

Table 2.2: Linear address format in the KU060 device

SEM IP

The Soft Error Mitigation (SEM) IP [19] is provided by Xilinx to help the user take advantage of the protection mechanisms integrated in Xilinx FPGAs for configuration memory: Error Correcting Codes (ECC) and Cyclic Redundancy Check (CRC) codes. They allow the IP to perform internal configuration scrubbing to correct errors, for example, caused by radiation. It uses two primitives:

- **ICAP primitive:** it automatically manages the ICAP access and commands needed to read and write configuration frames. The commands needed to write and read the configuration through ICAP are not documented, and no support is provided for other usage than integrated with the SEM IP.

- **FRAME ECC primitive:** manages the automatic checking of the ECC and CRC codes integrated in the readback CRC mechanism.

Even though little detail is given on how the primitives function or the ports they include, the general mechanism by which the corrections are performed is known: each configuration frame contains an embedded ECC code able to correct up to 4-bit errors inside a frame. Unlike previous generations of the SEM IP, such as for the 7th generation of Xilinx FPGAs, the ECC code used is not just a Single-Error Correction Double-Error Detection (SECDEC) code, but a more elaborated ECC-based algorithm whose capabilities are not clearly specified. When a frame is read through the ICAP, the ECC check is automatically done by the FRAME ECC. If the location of the error is detected, it is automatically corrected. Besides, it supports an additional classification capability when used in combination with an external memory that contains the .ebd file to indicate whether the affected bits were essential or not. This capability is not used in this work, since the injected bits are all essential.

There may be failures that are undetected or not corrected properly by using ECC, especially if the error affects the ECC information itself, or it is a multiple bit error inside one frame that the ECC does not cover. To detect those cases, the whole configuration memory is covered by a CRC. The SEM IP repeatedly reads the configuration memory, calculating the CRC of the read frames one by one until the last frame is read. The calculated CRC is then compared to the stored golden-CRC, which was previously calculated, before programming the FPGA. If they differ, the FRAME ECC and SEM IP flag a CRC error. However, those bits are not corrected by the SEM IP, and require further action by the user.

To test the scrubbing mechanism of the SEM IP, it includes the capability to inject an error in a configuration bit selected by the user. To do so, it reads the frame in which the bit is located, flips its value and writes it back through the ICAP. Besides, a command interface to connect the IP to a UART and control it externally is included. For more details about the SEM IP, the reader is referred to [19].

2.2.4 Upset classification and occurrence

The space environment, without the protection of the magnetic field from the earth, is extremely harsh. Apart from extreme temperatures and void, electronics have to deal with the undesirable effects produced by radiation. Radiation varies in energy depending on the elements that produce it, ranging from protons and neutrons to light and heavy ions coming from the sun, trapped in the earth magnetic field, or outside the solar system in the form of Galactic Cosmic Rays [20]. Its effects depend on two main factors:

- **The Linear Energy Transfer (LET):** depends on the particle producing the radiation. It is defined as the amount of energy deposited as it passes through a semiconductor, $\frac{dE}{dX}(MeV/mg/cm^2)$ [21]

- **The Fluence:** defined as the number of particles per square centimeter that irradiate the silicon [21].

The most significant effects that radiation produces can be summarized as:

- **Total Ionizing Dose (TID):** it is the amount of ionizing radiation that a particular device can stand until a functional failure, threshold shift, leakage current or timing change is produced [20]. Measured in *krads*, it is usually calculated for every mission, and then components and shielding are chosen so that they will withstand the expected dose.

- **Single Event Effects (SEEs):** they are caused by single particles that pass through the semiconductor material, inducing charge in the substrate that can create a current or voltage spike that may produce diverse effects[20]:

  - Hard (destructive errors): when the particle produces a spike that creates a low-impedance path between power rails (latch-up), or a transistor source, a short-circuit is created, leading to the potential destruction of the device. These errors are denominated Single-event latch-up (SEL) or Single-event burnout (SEB). Besides, if the gate oxide is hit by a high-energy ion, it may break in the so-called Single-event Gate rupture (SEGR) [2]. All the hard errors are mostly destructive errors, with the exception of SEL, that may be recovered with a power cycle.
  - Soft errors: alterations in a logical value in the device caused by a charged particle impacting a routing line, interconnection or memory cell. If the combinatorial path of a device is affected, the event is called Single-event Transient (SET). When a SET propagates through the circuit and reaches a memory element, or directly affects it, it changes the internal state and alter the data stored in that element. This event is a Single-Event Upset (SEU). Depending on the amount of bits affected, they are divided between Single-bit Upsets (SBUs) or Multi-bit Upsets (MBUs) [2]. Some authors [22] distinguish between MBUs caused by a single particle, and coincident SEUs that effectively act as a MBU if they accumulate.

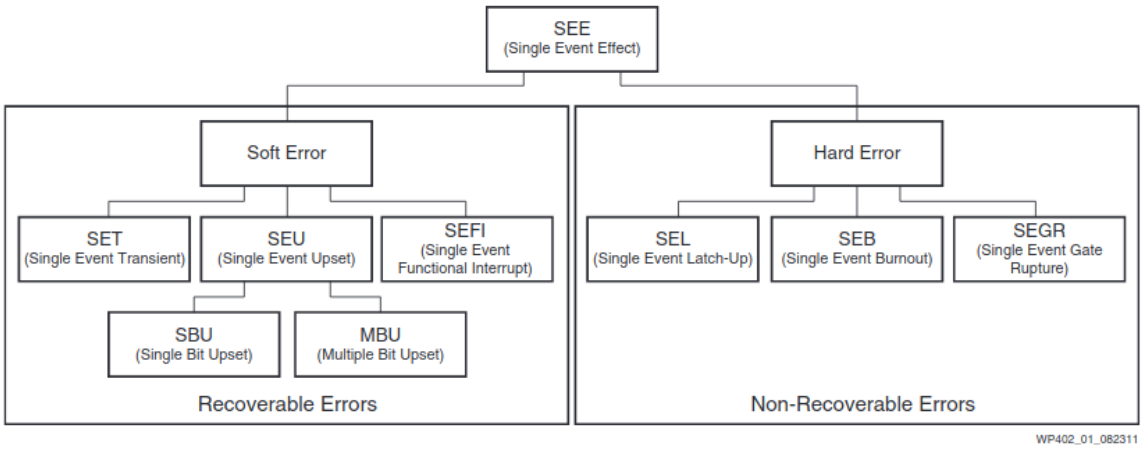

Figure 2.4: Types of Single-Event Effects. [2] p. 4

From a more practical point of view, a SEE is a fault in the circuit that, if propagated in the form of a SEU can cause the circuit to enter an erroneous state. Moreover, if this erroneous state alters the nominal behavior of the circuit and is propagated to an output, it is said that it caused a failure [23].

As CMOS technology is shrinking, new devices operate with reduced electrical margins, and gate oxide thickness is diminishing. As a consequence, the tolerance to SEL and TID is improving [20]. However, the SEU sensitivity is increasing, to the extent that they need to be taken into account for devices operating on the ground and not subject to harsh environments [24]. Hence, this work focuses in the latter, as SEUs are the most likely cause of failure due to radiation, as is also proven by the accelerated testing campaigns introduced below.

2.2.5 Radiation campaigns

In order to obtain the radiation sensitivity figures for a particular device, radiation campaigns are performed. FPGAs are exposed to a controlled source of radiation in terms of LET and fluence. Specialized installations such as Cyclotrons producing a range of particles such as neutrons, protons and heavy-ions are used for this purpose [25].

In those campaigns, the devices are loaded with test-implementations such as counters, shift-registers, and combinatorial elements such as multipliers and adders [26] [27]. However, as complexity of the design increases, more complex designs are starting to be used in the FPGA to effectively reflect a more realistic scenario. Some examples are the usage of the Soft Error Mitigation core (SEM IP) design from Xilinx, the Multi-Gigabit transceiver and Phase-Locking Loops (PLLs) in [28] and soft-processors in [25].

Different techniques are used to characterize the errors and effects of radiation. As mentioned before, SEUs are the most common and main target of radiation campaigns. Besides, SELs and other destructive events are also observed and recorded. Originally, a whole device read-back of both configuration and block memory data was commonly used, which was then compared to the original bitstream [26], [27] [29] [4]. However, this technique is extremely slow, since it requires disabling the clocks of the circuit—effectively freezing its state and masking potential errors—during the readback process, that lasts in the order of seconds [25]. This procedure allows to obtain"snapshots" of the SEU occurrence in the configuration memory and to count the amount of occurrences. For the BRAM, usually a pre-defined pattern is loaded and then read back within the bitstream. Additionally, the power rails are monitored with current monitors to observe potentially destructive effects [27]. On top of that, the increase of current is also an effective metric for the SEU rate that configuration memory is subject to, since it creates spurious consumption sources by altering the device configuration and functionality.

More recently, new monitoring techniques are being used to obtain a finer-grain monitoring in the occurrence of SEUs and shifting the focus towards observation of the functional behavior of the implementations subject to radiation. In [25], [30], a low-cost digital tester composed by an FPGA implements an FSM that provides input stimuli to the DUT-FPGA and checks the outputs and response of the functional units inside it. This enables monitoring events in the order of ns. The purpose of these implementations is

not to gather information about the vulnerability of the design itself, but to create DUT designs that have large and replicable logic structures, with state-spaces easily traversable that will produce observable SEUs. In [28], the DUT designs integrated self-checking features with status values that would be reported to an external monitor, and also remote monitoring is used to analyze the SEM-IP in [31].

Once results are collected, they are usually summarized in terms of the SEU cross-section (σ), which is obtained by dividing the total number of upsets by the fluence used in the test. It is usually reported in cm^2/bit. Also the occurrence of SEL for a range of LETs is reported, and sometimes the TID results. All that information is later used to evaluate the suitability for a particular device integrating certain technology for a given mission.

It is important to consider that the probability of failure is both device-dependent—obtained in radiation campaigns—and design-dependent, thus, both need to be taken into account. In section 2.2.7, how to tackle the second factor will be approached.

2.2.6 Upset occurrence for the KU060 FPGA

The Radiation-tolerant Kintex UltraScale FPGA XQRKU060 [11], is the UltraScale KU060 device tested and qualified by Xilinx for it usage in space. With its release, Xilinx published results from radiation campaigns performed for its qualification in [32]. According to the results, it stands a TID of 100 $Krad$, and is immune to SEL until a LET of 80 $Mev - cm^2/mg$. On the other hand, it has an expected SEU occurrence of 1.0e-8 upsets per bit per day for the configuration memory, and 2.7e-8 upsets per bit per day for the embedded block ram memory. Besides, a Single-Event Upset affecting a critical section of CRAM—such as a control state-machine— that causes loss of functionality, denominated Single-Event Functional Interrupt (SEFI) happens as often as 4.5e-4 upsets/device/day for Geostationary orbits.

Several additional and independent radiation campaigns have been performed in the regular KU060 FPGA, and their results are here summarized:

NASA reports their results in [25], [30]. The obtained cross-section is 1.24e-09 cm^2/bit for a LET of 20.4 $Mev \cdot cm^2/mg$, and 2.3e-15 cm^2/bit for proton testing. They also report the Mean-Fluence to Failure (MFTF) in $particles/cm^2$, which is the inverse of the cross section, and compare the Triple-Modular-Redundancy version of a counter synthesized in the KU060 to the same design—without TMR—in the radiation-hardened Virtex5QV, the current standard for high-throughput implementations. The results show that the UltraScale FPGA had a lower MFTF. Hence, being more vulnerable for the same LET, although the results were in the same order of magnitude, it shows potential for the non-hardened alternative. Besides, they compute the reliability of operation for a year from the MFTF and the expected particle flux per year. The results are shown for a MicroBlaze processor without cache, the same processor with cache and a simple counter, offering a reliability of 0.1, 0.7 and 0.87 for a fluence of 2.5e05 $particles/cm^2$. These facts highlight the relevance of developing mitigation techniques built in the logic, and the need to perform a careful analysis of each design individually, since their vulnerability to radiation varies greatly even for similar circuits.

In [4], the cross section for different LETs is presented, and the one for protons is estimated to be 1.87e-15 cm^2/bit for configuration memory, and 4.74e-15 cm^2/bit for SRAM memory. They extrapolate these results to In-Orbit rates using the CREME96 model, summarized in table 2.3. In [29], neutron and proton cross-sections are obtained, with results of 2.6e-15 for configuration memory (CRAM) and 4.5e-15 for BRAM in the case of neutrons, and 2.5e-15 and 4.3e-15 for protons, all values expressed in cm^2/bit.

	Configuration Memory	Block RAM
per bit, per day	7.54e-09	2.48e-08
per device, per day	7.19e-01	6.26e-01

Table 2.3: In-Orbit SEU rate for CREME96 [4]

In [31], the cross-section of the SEM IP in the KU060 is analyzed by exposing the device to a proton flux while the SEM IP is running in the background, correcting any eventual SBU detected. The cross-section is recorded when an uncorrectable error is produced. The obtained cross-section is 5.91e-11 cm^2/bit for CRAM. Besides, additional data analysis is performed to show that only 0.07% of the SEUs cause MBUs, with 99.93% causing SBUs in CRAM. Additionally, the authors state that no MBUs in BRAMs were recorded in their testing. This fact confirms the statement presented in [33]: to enhance the protective capabilities of SECDEC codes protecting BRAM and CRAM memory words in Xilinx FPGAs, the memory frames are physically interleaved.

In [28], the SEM IP is included in testing under proton irradiation, showing that it improves CRAM cross-section by a factor of 1.77, from 2.05e-15 to 3.63e-15 cm^2/bit. On the other hand, the usage of a SECDEC ECC code in BRAM drastically improved BRAM cross-section by a factor of 3272, from 2.52e-15 to 8.25e-12. These results back the ones presented in the previous paragraph, showing that the protection of BRAM is noticeably easier using ECC codes thanks to the low rate of MBUs when compared to CRAM. It is important to note that these results are based on proton irradiation, more applicable to Low-Earth Orbit (LEO). As a conclusion, the authors present LEO In-Orbit SEU rates (including previous results for heavy ions) for a device with 75% utilization excluding the Multi-Gigabit Transceivers. The results are 0.021 upsets/day, which improves to 0.001 upsets/day with memories using ECC and 0.00086 upsets/day using the SEM IP, rates that are acceptable for short LEO missions. However, the usage of SEM IP shall still be studied, since in [27], the only high-current events observed were related to a $ScrubSEFI$ that would happen when configuration engine and critical registers such as the Frame Address Register (FAR) used for scrubbing and configuration readback were affected by a SEU. These events are only caused while scrubbing through configuration memory, as SEM does, and do not affect FPGA configuration. In the article, the authors present an alternative scrubbing method that would minimize such events.

To conclude, results presented in this section demonstrate three main facts:

- SEUs are the main source of failure due to radiation, and determine the expected reliability of a device operating in space. Few MBUs, SEFIs, SELs or destructive events have been registered, and are not expected in a regular mission.
- Apart from obtaining the expected SEU incidence for a specific device, it is essential to analyze the vulnerability of a particular design to SEU, since the final error rates

will vary greatly depending on both factors. This work aims to design a feasible methodology to do so for any particular implementation, without having to perform a radiation campaign for each mission.

- While BRAM is easier to protect with an ECC or TMR scheme, effectively reducing the error rate by several orders of magnitude, the CRAM is harder to protect. With the current solution offered by Xilinx SEM IP, it is not reliable enough for longer or more critical missions. The need to analyze the effectivity of other mitigation techniques, such as distributed TMR, recommended by NASA. The methodology presented in chapter 4 can also be applied with this goal in mind.

2.2.7 Previous work in SEU injection

Due to the high cost and effort required to perform a radiation campaign, several techniques for SEU emulation have been developed that allow to emulate the occurrence of a SEU by introducing an undesired change in a value, either in the user memory, registers or configuration memory. These campaigns allow to obtain a metric of the design sensitivity by counting the amount of configuration bits in a design that, when affected by an SEU, cause a functional failure. They are usually called *critical bits*, and are extremely useful to compute the reliability of a design. The sensitivity to SEU of a design is summarized by the Design Vulnerability Factor (DVF), which is obtained by dividing the amount of critical bits by the total amount of configuration bits used by the design. Multiple approaches to SEU injection have been presented in recent research:

- **Injection circuits (*saboteurs*):** introducing modifications in the component library of a particular vendor, or using certain commercial tools that modify an initial netlist, these circuits allow a fast and effective injection for user memory [34]. These techniques are oriented towards user memory only, not acting in configuration memory and being more suitable for ASICs. Besides, they need either specific tools or a great design effort, and add overhead to the logic in the circuit.

- **Simulation-based injection:** with this technique, errors are simulated rather than emulated [35]. Hence, the time required to inject enough errors makes this approach impractical as design complexity increases.

- **Reconfiguration-based injection:** based either on total or partial reconfiguration. A configuration frame is read, modified by flipping one or more bits to emulate a SEU and written back to the configuration memory. The approach based on **total** reconfiguration allows for more control of the internal states of the circuit and access both the configuration memory and the user memory through a total readback using JTAG [36]. However, that operation is noticeably slow, requiring a reported time of 916 seconds for 3000 bit-flips (around 3 bits flipped per second). For that reason, injection based in **partial** reconfiguration has become the most popular alternative [37], especially since the Internal Configuration Access Port (ICAP) was introduced by Xilinx, allowing for a faster reconfiguration. This work focuses on this last mechanism for SEU injection, allowing injection of a whole design in a reasonable time, with an estimated time per bit of less than 5 ms per injection [38].

The usage of partial reconfiguration for error injection can use any of the configuration ports provided in Xilinx FPGAs: external ports, i.e., the JTAG configuration port, the

SelectMAP configuration port and the ICAP. The most suitable choice is ICAP, due to the speed achievable compared to its counterparts [38]. The main limitation when using partial reconfiguration to inject errors is that they are only meant to act in configuration memory: as explained in [38], the information stored in a FF is not modifiable after the initial programming operation using the configuration memory unless the sequential logic is reset. Thus, a careful process such as the one detailed in [38] is needed to manage the individual resets. Another alternative is presented in [21], using injection through partial reconfiguration, and a structural approach, this is, adding injection inputs manually to all flip-flops in the design using a parser and the design netlist.

Even though the data that can be collected is limited by the fact that no injections can be performed in BRAM memory and flip-flops, injection by partial reconfiguration is the current mechanism used to compute device sensitivity to SEU. The main reason for this is that it is the most practical mechanism when approaching complex circuits, which does not require any specific tool or manipulation of the circuit, is being able to assess its sensitivity as-is, without any additional logic that may alter the results. The obtained sensitivity will be lower than the real one for devices that have a high utilization of BRAM memory, but, even in that case, the SEU injection is accurate when used to obtain the relative sensitivity that results of implementing mitigation techniques in the circuit, and can be used as a tool to assess the effectivity of these techniques [39].

Apart from the method used to inject errors, it is interesting to reflect on the different mechanisms used to check for functional failures. These partially depend on the type of circuit tested:

- **Golden-model:** consists in running two instances of the same circuit in parallel. One is the Device Under Test (DUT), in which the injection will be carried out, and another one is the *golden reference* or *model*. This approach is commonly used combined with a set of test-vectors that are used as the inputs of both circuits. Then, the outputs are compared, and, if they differ, an error occurred in the DUT. Its main advantage is that it is not hard to implement, provided that an automatic tool is used to generate the test vectors with a good internal-state coverage and that the visibility of errors cannot be affected by the error itself. As a disadvantage, it only allows a limited understanding of the errors, since they can only be detected in the output. Hence, it works properly for circuits that perform a bounded calculation, but not for the case of a module implementing a communication protocol that could mask a significant amount of failures. It is the approach followed in [40], [21], [41], [42].

- **Stand-alone with internal monitoring:** consists in instantiating just one DUT, and monitoring internal signals or registers and outputs to gather more information about potential failure modes. It is a suitable approach for circuits that include Quality-of-service (QoS) mechanisms already integrated that allow for error detection, or if an automatic mechanism to detect failures by examining the internal registers of the circuit is feasible and simpler to implement than duplicating it. Its main advantages are the reduction in the amount of logic needed, allowing for testing bigger DUTs and less intrusive logic that will have less impact on the results. Besides, it is able to detect failures that would be masked in the output in circuits that include error correction or QoS mechanisms. On top of that, as explained at

the end of this section, it is the most suitable approach for verification purposes. [23]. Its main drawback is that, potentially, some internal errors could be masked by the errors themselves. This risk can be minimized by continuously recording the monitored signals and not allowing errors to be introduced in the data generating and checking blocks, which will still be able to detect the most relevant functional errors. This approach will be the one used in this work, and was also used in [43].

- **Mixed approach:** finally, combining both techniques described above is also possible. It provides the most reliable mechanism when it comes to detecting any error or deviation from the expected behavior, allowing for high detection granularity. However, it comes at a very high logic cost, allowing to test DUTs of a constrained size, and altering the occupation of the device in a significant way, which can also introduce differences in the behavior of the circuit against SEUs when compared to a stand-alone implementation. Besides, although uncommon, SEFI events at FPGA level may still mask some errors and cause failure in both DUT and golden model by affecting, for example, the global clock routing. This is the approach used in [39].

Once the mechanism for error injection and correction has been obtained, in literature there are two main approaches to compute a design's sensitivity:

- Following the same approach used in radiation campaigns: injecting errors in random locations of the configuration memory. This approach, used in [39], [42] allows to obtain the average amount of injections needed for the device to fail. It is effective when used with a model of the expected SEUs, and its main advantage is that it does not need the data from radiation campaigns to estimate the reliability of a particular circuit. However, this method is not comprehensive, in the sense that it will leave configuration bits without analyzing them, so potential vulnerability points will remain unknown.

- Systematically injecting errors in all locations of configuration memory that belong to the DUT. This approach is used in [21], [43] and [40]. This method allows for a comprehensive analysis of all potential vulnerabilities related to configuration memory for any particular implementation, and for a more precise calculation of the DVF. Its main drawback is that it requires data from radiation campaigns performed in the same device to draw conclusions about the absolute reliability that can be expected for a particular mission, and that might be to slow in bigger designs.

Although several differences have been presented, all the works analyzed have a similar methodology in common. The usage of partial reconfiguration to inject errors in configuration memory is done either through the SEM IP from Xilinx [21, 40] or through means of a custom reconfiguration controller [41, 39, 42]. The injection consists in flipping one bit, emulating an SEU, except for the case of [41], which is the only work that focuses on MBUs. The reason of focusing on SBUs is that those are the most common events—as shown in section 2.2.6. After the error is injected, the circuit is kept running until it performs a particular operation and a set of outputs are available. For the sake of velocity,when errors are observed in the outputs they are corrected by writing back the original word to configuration memory. Only in [39] and [40] a full device reboot was performed to ensure the error correction.

2.3 Space Fibre

This work is closely related with the SpaceFibre technology. In this section, a short introduction to its basic concepts is shown. For the sake of brevity, most details and several capabilities of SpaceFibre are omitted. Hence, the reader is referred to the ECSS standard [3] for more information. SpaceFibre is a high speed serial link based network standard defined for on board spacecraft usage. SpaceFibre specifies the physical layer for both electrical and fibre optic medium. In the electrical medium, which is the one used in this work, Gigabit serial transceivers are used to serialize the data and transmit it. In particular, GTH transceivers from the KU060 FPGAs are used for the injection campaign.

2.3.1 Gigabit serial transceivers

The main component that SpaceFibre relies on is a transceiver capable to achieve line rates in the order of gigabits per second. These transceivers are becoming increasingly used onboard spacecrafts to transmit the huge data generated by the latest generation of instruments and sensors. Originally, they were not designed with the space environment in mind. Hence, additional protocols that ensure QoS such as SpaceFibre are needed. The basis of this transceivers lies in a SerDes element, which transmits a data word bit by bit at an extremely high rate. To do so, the SerDes uses a fast reference clock—125 MHz in this work—and several Phase-Locked Loop (PLL) circuits that are able to generate clocks whose frequency is a multiple of the reference clock and whose phase varies in a controlled manner, and transmit the data aligned to the flanks of these faster clocks, reaching line rates in the order of the gigabits per second. Only two differential lines are used: the TX and the RX line. At the rates used by gigabit transceivers, transmitting the reference clock in a separate line is not practical. However, from the values transmitted through the lines, the phase of the reference clock can be aligned at the far-end, to be able to appropriately sample the input values at the rigth moment. These values then deserialized and provided with a lower clock rate in parallel words. Apart from a reference clock and high-performance PLLs to align it, there are other elements that allow the successful transmission of data at such a high rate. Two of the most relevant, and whose usage is required by SpaceFibre are listed here:

- **Differential signaling:** the GTH transceiver uses LVDS in current-mode, which drives two parallel lines that encode the transmitted bits in the differences in voltage in the line. The lines are terminated with a resistor to maintain a constant value of current flowing between receiver and transmitter. Therefore, the generated magnetic fields cancel each other, significantly reducing the emissions. Besides, the noise affects both voltage values equally, and its effect is cancelled when obtaining the data encoded by the difference in voltages.

- **8b/10b encoding:** in order to be able to recover the clock from the data transmitted in the line, the voltage should resemble a typical digital clock as much as possible. Hence, data is encoded with 8b/10b aiming to minimize the consecutive bits that have the same voltage, maximizing its variation. To do so, the encoder maps every 8-bit symbol to a 10-bit symbol, defining two alternative symbol mappings for each 8-bit value. The symbols are selected to ensure that no more than 5 consecutive bits have the same value to facilitate clock recovery, and that the total

difference of 1s and 0s transmitted is lower or equal than two, to ensure DC balance, which minimizes noise and bit errors. Besides, the remaining values are used to define special control characters, the *comma* character being the most relevant one. It is a character that cannot be obtained by partially concatenating any other characters in the serially received bitstream. Hence, when detected, it marks a byte boundary used to align the received bytes correctly.

2.3.2 Space Fibre modules

As a network protocol or standard, SpaceFibre is divided into layers of abstraction that implement certain functionality as described by the protocol description in [3]. A SpaceFibre link is composed by one or more physical connections, called "lanes". SpaceFibre is capable of transmitting data in parallel through different lanes to increase its throughput. However, this work focuses in the single-lane mode with a throughput of 2.5 Gbps, which is enough for most applications. The layers for a single lane are the following: application, network, data link, lane and physical. They are shown in figure 2.5 The application layer is defined by the implemented system. Only its interface with the network layer is defined, establishing how data and broadcast words shall be transmitted to SpaceFibre. The physical layer is composed of the SerDes, drivers, cables and connectors, and the IP that implements the 8b/10b encoder and the synchronization and clock correction buffers according to the standard. The network layer—describing the routing and transmission between nodes of packets and broadcast messages—is not fully implemented in the SpaceFibre IP port, which is designed to receive and send data over a single link. The only functionality implemented is the conversion of packets and broadcast messages to N-chars and fills used in the data link layer. Therefore, this section introduces the data link and lane layers, which are the most relevant for this work. The SpaceFibre IP implementation divides the functionality of the data link layer in the virtual channel layer and retry layer:

- **Virtual channel layer:** a SpaceFibre virtual channel is an independent channel that can carry information across a single link in parallel with other independent, information carrying channels [3]. It is equivalent to a SpaceWire channel or link (terms with equal meaning in the SpaceWire protocol), with the term "virtual" used to highlight that a SpaceFibre link can contain several multiplexed SpaceWire channels used by different applications to transmit and receive data. One of the aims of SpaceFibre is to be compatible with SpaceWire. Hence, the virtual channels in SpaceFibre follow the same packet format as the channel in a SpaceWire link. The virtual channel layer is in charge of managing the medium access control, deciding which virtual channel transmits data depending on the allocated bandwidth and the amount of received flow-control tokens. The flow-control tokens indicate the amount of free space in the receiving buffer of the far-end, to ensure that no overflow and data loss occurs—. Other additional capabilities are the capability to define scheduling and precedence between virtual channels. The input and output buffers, one for each virtual channel, are located in this layer. On top of that, since several virtual channels may share a physical link, a virtual-channel de-multiplexer is used in the receiver part to drive the received data to the appropriate buffer. Data to transmit is encapsulated in frames, and passed to the lower layer. Finally, next to this layer, the broadcast layer is implemented, managing broadcast information that is passed directly to the lower layer.

- **Retry layer:** the retry layer is in charge of implementing the error recovery mechanisms of SpaceFibre. It receives data and broadcast information from the virtual channel layer, and passes the received data and Flow-Control Tokens to it. On the other hand, it exchanges data and control words with the lane layer. In this layer, the CRC field from each frame is checked, as well as the sequence in which the frames are received. This order is stored in a field inside each frame. If both checks are successful, an acknowledgment is sent to the other end, and the frame is written to the virtual channel buffer waiting for the application to read it. Else, a retry operation is initiated, asking the other end to send the frame again, until a maximum number of attempts. If a successful reception is not achieved in the determined number, it is notified to the lane layer to reset it. In the sending side, it implements an error recovery buffer to keep the frames stored until their acknowledgment, and the necessary logic to re-transmit them if an error occurred.

- **Lane layer:** the main purpose of the lane layer is to establish and maintain communication across a SpaceFibre lane [3]. It manages and detects the control words used for that matter, which are the initialization, signal loss and stand-by words. The control is integrated in the lane or initialization FSM, in charge of resetting and establishing the connection with the other end exchanging configuration information and performing a complex handshake. It abstracts all the lane control, sending to the retry layer the state of the connectivity. The lane layer interfaces directly with the physical layer, enabling the drivers when ready to transmit, and passing the corresponding symbols. Besides, it receives the parallel data de-serialized and decoded from the physical layer. Finally, when the 8b/10b decoder in the physical layer receives a wrong or unexpected symbol, or lane connectivity or synchronization cannot be reached, an RXERR word is passed to the lane layer, which contains a counter that can be used to measure the Bit Error Rate (BER) of the channel.

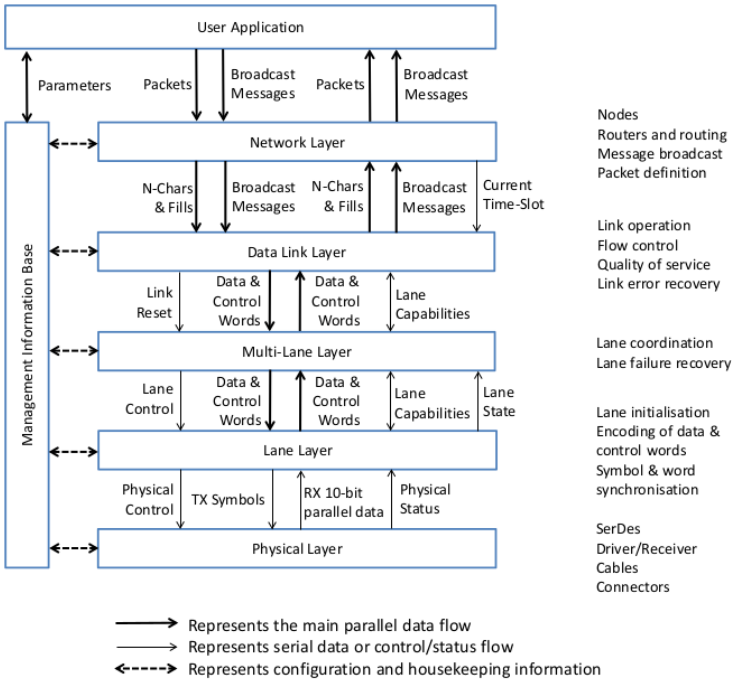

Figure 2.5: SpaceFibre Protocol Stack. [3] p. 36

Chapter 3

Firmware development for the flight demonstrator

The SpaceFibre In-Orbit Demonstrator is a digital board developed by Thales Alenia Space whose main purpose is to control two IP blocks that implement the SpaceFibre protocol, connected to data generators and checkers running recursive tests to validate them in orbit. One IP was originally developed by Cobham Gaisler AB for ESA, and the other one by STAR-Dundee Ltd. This chapter will introduce the SoC that was developed for the demonstrator board. The contribution from this work was the VHDL firmware for both SRAM and Flash FPGAs integrated within the board, as well as the support to develop the software test routines and validation of their functionality.

At the time of writing this thesis, the board is being prepared and integrated in a CubeSat project from a commercial space company for its launch later this year. The purpose of the demonstrator is to launch two different SpaceFibre IP cores and repeatedly perform test routines to validate its behavior in-orbit.

3.1 Board features and baseline

The board used to develop the demonstrator is shown in figure 3.1. The main elements of the board are: two COTs FPGAs, one SRAM-based and another FLASH-based; the SpaceFibre E-SATA connectors, routed to the SerDes transceivers of the SRAM FPGA; the programming SRAM memory for the FPGA firmware; ADCs connected to the power rail for continuous current monitoring; and the PC-104 connector that includes the UART and GPIO pins used to interface with the rest of the payload of the cubesat. Since there are no other SpaceFibre nodes in the payload, the SpaceFibre data connectors are routed in loopback, so that the two SpaceFibre ports will receive the same data they sent, and one port will act as both ends of the link. The ADCs are radiation-hardened, and monitor the FPGA power rails, in case radiation upsets induce high current events in the FPGA.

The flash FPGA is used to connect and multiplex signals from the ADC and PC-104 connector and drive them into the SRAM FPGA. Apart from that, all the firmware that was developed was for the SRAM FPGA, containing the LEON3 SoC with the SpaceFibre IPs. The SoC is used to run the necessary software to perform test runs, housekeeping monitoring for both IPs and reporting the telemetries them to the OBC.

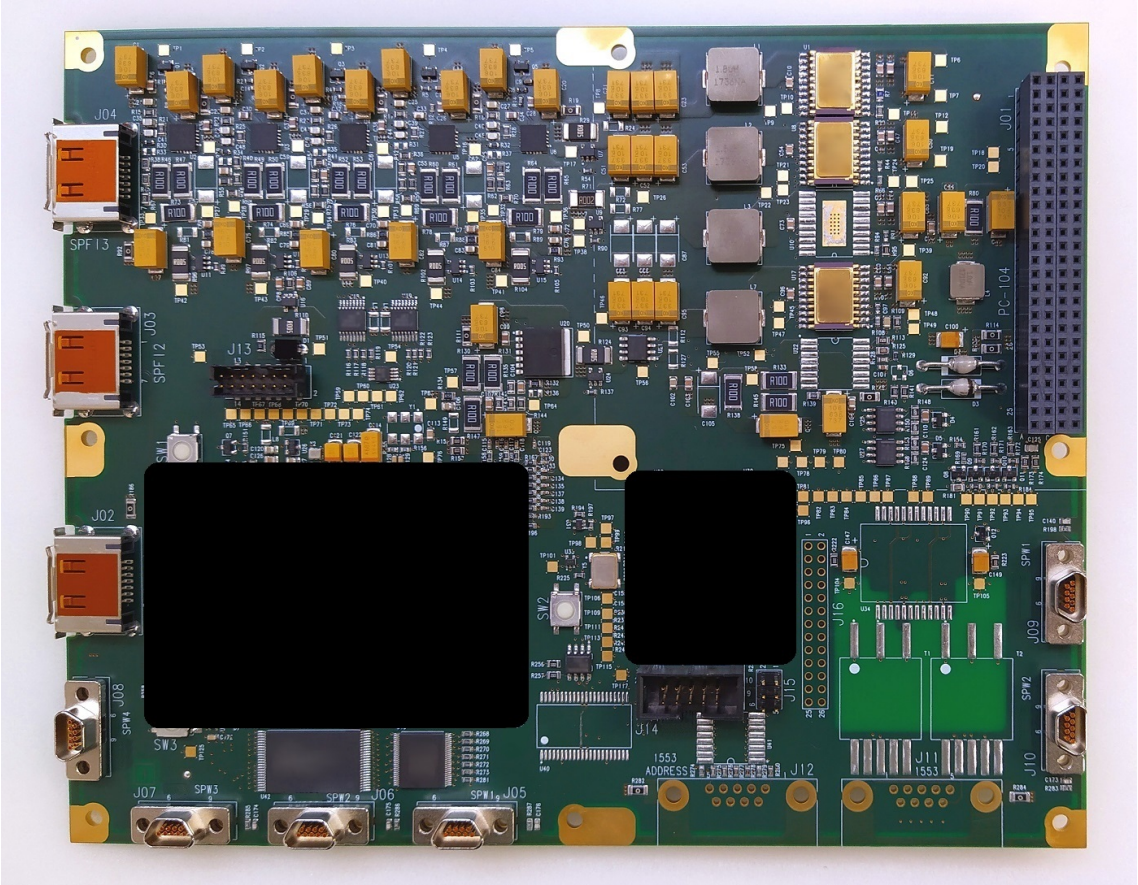

Figure 3.1: SpaceFibre In-Orbit Demonstrator board

3.2 Developments carried out

The developed System on Chip is presented in figure 3.2. The central elements are the LEON3 processor connected to 4 AMBA buses:

- **The main AHB bus:** this high-performance bus is connecting the elements with the highest bandwidth. In this case, those elements are the memory controller for both SRAM and ROM containing the user memory and the boot image for the software respectively, the Debug Support Uit, and the three controllers for the APB buses.

- **The peripheral APB bus:** the slower APB bus is used to manage peripherals that do not require a high-bandwidth, such as low-speed communication devices (UART and SPI) and managing control signals (interrupt controller and GPIO).

- **The SpaceFibre APB buses:** each SpaceFibre IP uses the generated user clock provided from the SerDes transceiver. This is due to the transceiver sampling and providing the parallel data at its input using the user clock, generated by dividing the main reference serial clock. Since the line rates for both transceivers are different, the generated user clocks are also different, hence the need for two separate APB buses that work at different clock frequency. The SpaceFibre APB buses are controlled from an AHB to APB bridge, that converts the signals from an AHB RMAP to APB signals. There is no need for a full APB controller since the bus is only connected

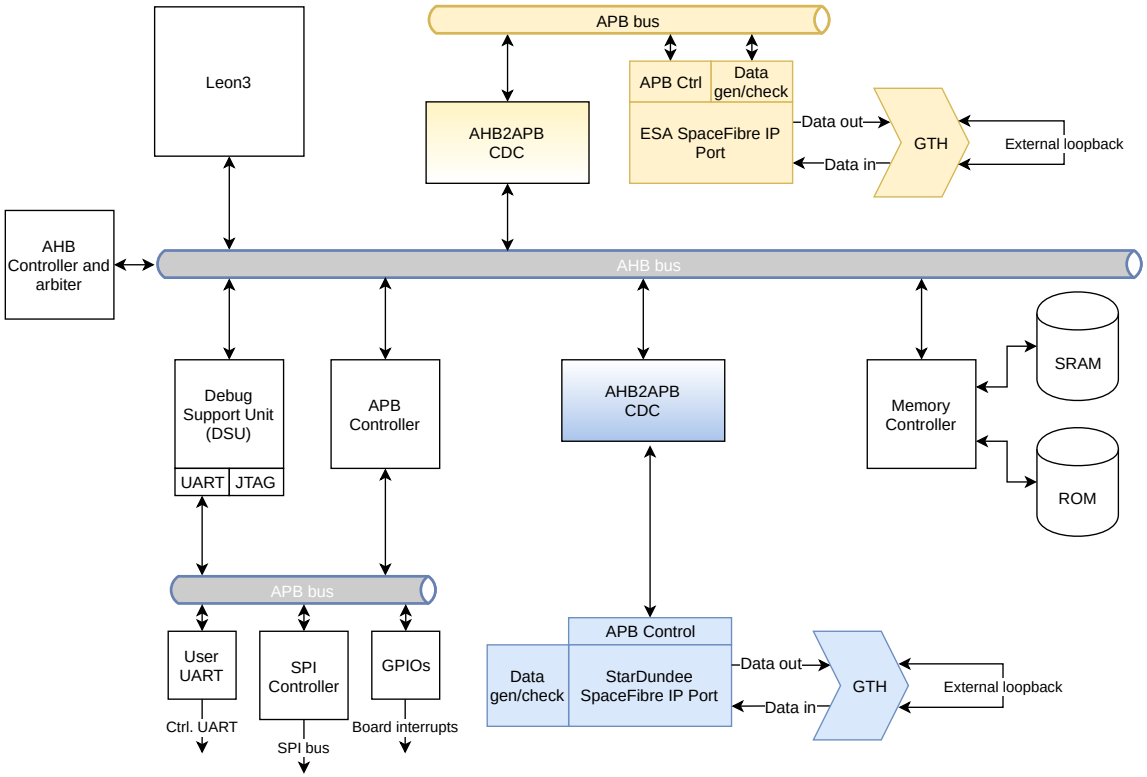

Figure 3.2: developed SoC

to one data source and sink. However, since the system frequency used for the SoC is different from both user clock frequencies, an appropriate clock-domain-crossing must be implemented inside this module. For that, handshaking along with double flip-flops to avoid metastability were included in the AHB to APB interface.

The SoC was implemented following the instantiation of each GRLIB component, and interconnecting them to the appropriate bus along with providing an index, address and mask compatible with the state-space of each peripheral. To facilitate the process, one of the baseline SoCs provided by Gaisler in the GRLIB was used as an example on how to put together the different components in the SoC. It is important to comment that, to facilitate configuration of the SoC, usually all cores are configured by using VHDL generics, that can be modified in the file *config.vhd* file. Apart from including the GRLIB elements shown in figure 3.2, after each important piece was introduced in the SoC, a validation in the board was performed to make sure that the behavior was correct. Since the detailed technical characteristics and specifications of the SpaceFibre IPs and hardware components such as ADCs and memories cannot be disclosed due to commercial

reasons, a short and general summary of the activities is presented here.

3.2.1 DSU validation

The first step was to instantiate the LEON3 system with the AHB bus and the APB bus along with all elements of the Debug Support Unit (DSU). A first bitstream was generated to check the capability of the FPGA to be programmed from the flash memory, as well as setting up the debug communication via UART connected to a PC running GRMON, which would allow to verify all elements within the SoC. To set-up the bitstream generation for programming from a flash device, a tutorial can be found in [44]. To validate this step, the ROM flash for the SRAM FPGA was programmed with the bitstream, and GRMON launched connected to the UART with the following command written in the windows command line:
`grmon -uart \.\\comport`
Once it recognizes the LEON3 processor with the DSU, a summary of the elements included in the SoC is displayed upon connection. More detailed information, including the address ranges and the obtained sizes of connected memory can be obtained by issuing the `info sys` command in the GRMON prompt.

3.2.2 Memory controllers validation

Before being able to run code in the LEON3 processor, it was necessary to instantiate the memory controller with the appropriate address space and word size parameters. Like most peripherals, its configuration can be accessed in runtime. However, since memory should be correctly configured at implementation time, an input port to the memory is provided where configuration parameters can be written as VHDL constants. Once integrated into a new bitstream, both SRAM and ROM memories could be validated. For the SRAM memory, the BCC compiler was used to compile the dhrystone benchmark for a LEON3 SoC, which would also verify that the LEON3 processor worked correctly. The command used to compile with bcc from the Windows command line is:
`./sparc-gaisler-elf-gcc dhry.c -o dhry.elf -lm -mcpu=leon3 -msoft-float`
The added flags are important to obtain correct behavior. The binary file was be loaded in the SRAM memory from GRMON using the `load` command. Then, using the `run` command, the testbench was executed. After that, to verify the ROM memory and controller, a boot image was created from the binary file using MKPROM2 with the following command:
`mkprom2 -leon3 -rmw -ramsize 8192 -ramdwidth 16 -ramws 1 -romsize 8192 -romwidth 16 -romws 5 -msoft-float dhry.elf`
It is important to specify correctly all memory parameters and waitstates. The waitstates are used to introduce no-operation (NOP) instructions to wait for a memory access in case the memory is not fast enough to provide data in time. Before programming the ROM memory, a preliminary check on its communication with the processor shall be done by sending the command `flash` in the GRMON prompt, that will use the Common Flash Memory Interface (CFI) if supported by the ROM memory to send the characteristics of the memory. If the information is retrieved correctly, the flash memory can be erased with `flash erase all` and then programmed with the boot image with `flash program dhry.prom`, both commands issued in the GRMON prompt.

3.2.3 SPI controller and multiplexing validation

The next step was to set-up the SPI controller to communicate with the ADCs connected to the power rails. After adding the SPI controller connected to the APB bus in the SoC, it shall be configured in runtime as stated in GRLIP IP catalog [7], so that each SPI transaction is compatible with the waveforms as specified in the ADCs datasheet. To configure the ADC, GRMON's DMA was used to write to the configuration registers of the SPI controller in the address offsets shown in the GRLIB IP catalog. Then, reads to the different channels of the ADC were issued following the same procedure by writing the address of the SPI channel in the SPI transmit data register, and the received data was checked in the SPI received data register. To validate the measurements from the SPI, a multimeter was used to measure the voltages at the input pins of the ADC, and that value compared to the one read from GRMON.

3.2.4 SerDes transceivers validation

Before integrating the SpaceFibre IP cores, the loopback routing used in the board to emulate both ends of the SpaceFibre link with a single port, and reference clock from an independent oscillator used to reach 2.5 Gbps were tested. Simple designs configured to work with a reference clock of 125 MHz and set-up for the high-speed transceivers were integrated in the SRAM FPGA. The design includes several status signals that indicate that the transceiver's Phase Locked Loops (PLLs) were correctly locked to the reference clock and are ready to send and receive data. Those status signals were mapped to test-points in the board, and checked that after a reset the transceiver would initialize correctly and be ready to transmit data at the desired rate.

3.2.5 ESA SpaceFibre IP integration and validation

The first SpaceFibre IP to be integrated was ESA's. It includes the necessary elements to be connected to the AHB bus using the AHB to APB bridge with clock-domain crossing. The control interface and data generator and checker already included the APB interface. Hence, these two elements that use different indexes for the APB communication were connected to the AHB to APB bridge and to the SpaceFibre IP itself. The IP was then connected to the SerDes transceiver, appropriately configured to provide the necessary interfaces with the IP as well as use the data rate for which the IP is configured (2.5 Gbps on this case, with a reference clock of 125 MHz for an input data width of 32 bits, and configured to generate 8b/10b output). The clock used for all the SpaceFibre IP blocks shall be the output user clock from the transceiver, since it is the clock that the data is aligned to. The transceiver must be set up to be reset either from the main board reset or from a software reset coming from the AHB bus. The SpaceFibre IP reset must be issued from the transceiver, since it can only exit the reset state when the transceiver has been successfully locked to the reference and is providing a stable output clock signal. Hence, the IP was connected to the *reset_done* signal from the transceiver.

When it comes to its validation, it was firstly done manually by using GRMON to write to the configuration registers of the IP, that are detailed in its user manual. Details about the registers cannot be shared in this report. Hence, the validation is summarized as enabling the IP, waiting until the lane is set-up after a sucessful handshake between transceiver and receiver and enabling the data generator/checker, starting to transmit

data. Status signals are checked for the correct lane state and absence of errors.

After the IP was verified and integrated, the driver written in the C language and provided by ESA was adapted to the address space of the SoC, compiled and a repetitive test that would check for errors and deviations in the normal operation as well as a check for the transmitted bandwidth was run to confirm the correct behavior of the IP. This code would be the baseline used to develop the final software of the demonstrator.

3.2.6 STAR-Dundee SpaceFibre IP integration and validation

After the successful integration of ESA's IP, the last step in the process was the integration of STAR-Dundee's SpaceFibre IP. The IP was provided in the form of an encrypted netlist, without access to the VHDL code. Besides, a control block to manage the reset of the transceiver was provided. The reset block is connected to the board reset, and, similarly to ESA's block, it will wait for a stable clock input to the IP to release the reset. The control interface of the IP had to be adapted to manage the APB transactions and addressing to have access to all configuration and status parameters, which was done with the help of STAR-Dundee. Besides, the AHB to APB bridge had to be modified to fully comply with the AMBA 2.0 standard, since, originally, the signal *psel* was only compliant to the AMBA3.0 standard. The IP also integrated a data generator and checker internally. After that, the transceiver was instantiated and configured to work at a line rate of 1.25 Gbps, this time with input data of 40 bits, since the 8b/10b coder was already integrated into STAR-Dundee's IP. The IP was connected to the transceiver, and the APB interface to the AHB to APB bridge which, in turn, was connected directly to the AHB bus.

Once finished the development of the firmware, the last validation was performed following the same steps as for ESA's IP: starting with GRMON manually setting-up the parameters for the core, and following with the final software validation using C code to configure the transceiver and set it up to transmit and check test-data, gathering all metrics of errors or potential problems.

3.2.7 Final validation

A final validation with the Thales software responsible for the project was performed, running all final routines and emulating the commands that would be sent from the OBC to the board during the mission. It was also relevant to use the oscilloscope to inspect the signals and transactions, to make sure that the electrical levels and timings were according to the specification.

As a final remark, the demonstrator did not implement any SEU prevention or mitigation techniques. However, it is re-programmed every time a test is run, clearing all possible errors caused by radiation effects. Besides, one of the goals of the demonstrator is to observe potential SEU occurence in LEO for the SRAM FPGA, and tests can fail without causing any harm to the system.

Chapter 4

SEU injection in the SpaceFibre IP core

This chapter presents the methodology elaborated to perform a SEU injection campaign in the SpaceFibre IP port. The applied methodology is described in detail, since it provides valuable guidelines for future injection campaigns, and the author proposes its usage as an additional step in the verification phase, allowing to assess the reliability of a particular design for a determined mission, and observe the failure modes of the circuit in order to propose and evaluate different mitigation techniques. It is applicable to any implementation which includes a soft-processor—a majority currently in the sector, and does not introduce a significant amount of additional logic in the design.

4.1 Design choices

Before introducing the methodology, the design choices that were taken during its elaboration are presented:

- The SEM IP was chosen as the injection engine, since it is a convenient and already tested tool for injection through partial reconfiguration that does not imply any drawback compared with custom solutions if the injection speed is not critical.

- For the correction, the automatic mechanism of the SEM IP was chosen. It uses the ECC code calculated for each frame combined with the CRC for the whole CRAM to detect errors. Even though writing back the corrected bit through the ICAP is possible and slightly faster, the automatic mode was chosen to observe potential functional failures in the SEM IP that would risk the correctness of the experiment and that would not be caught by the alternative. In the results section, these failures are explained in more detail.

- The integrated LEON3 processor was selected to monitor the internal signals of the SpaceFibre IP through the APB interface. This alternative was preferred over the development of a custom and faster hardware-FSM, since, due to the amount of signals to monitor, it incurred a high cost in logic occupation that would not be used for the mission and would introduce additional vulnerabilities. Besides, it reduces the cost of developing the injection campaign, since having to design a custom interface for every circuit to be tested would be too costly. Moreover, the

usage of the APB bus as an interface with the DUT is fast enough to monitor the change in any of the signals, with a transaction latency of just 4 clock cycles for each status register, or less than 1.8 μs for all the monitored signals.

- The UART interface of the SEM IP was chosen for controlling the injection and correction. Again, avoiding a custom internal interface with the LEON3 processor has the same advantages discussed above. Besides, having an external monitor for the injection allows automatically rebooting and reprogramming the FPGA upon the occurrence of uncorrectable errors or functional failures of the SEM IP, hence completely automatizing the process and removing the need for human intervention. On top of that, the UART was also chosen as the interface with the internal supervisor, the LEON3 processor, and the test controller, the PC, allowing for a simple synchronization between both of them.

- ESA's SpaceFibre IP was selected for the injection. Due to time constraints, only one injection campaign was planned, and ESA's IP was chosen over StarDundee's because the full VHDL source was available, allowing for a distinction on the different layers of the IP, and performing separate runs into each of them. Besides, ESA's IP was available from the beginning of the project, allowing to start the integration and development of the drivers faster when compared to the alternative.

- Python was the programming language of choice for the test controller. It was chosen over the alternatives for the simplicity of managing the serial port—with the PySerial library [45]—and processing the data managing different files, while providing more than enough performance to control the experiment in a timely manner.

- Due to accessibility limitations to the demonstrator board related with Covid-19, the design was ported to the SSDP board developed by Thales Alenia Space. This board is commercially available through Cobham Gaisler as the GR-XCKU board, and includes a KU060 FPGA, flash, RAM and ROM memory with very similar characteristics to the the demonstrator board. However, the loopback e-SATA connector used in the demonstrator is not present in the Thales board. To substitute it, a loopback FPGA Mezzanine Card (FMC) was connected to loop-back the high-performance pins from the FMC interface to the SpaceFibre lane. Besides, the FMC card provides a 125 MHz oscillator that was used as the reference clock for the SpaceFibre IP.

- To detect failures in the IP upon fault injection, a continuous inspection of the status registers of the IP was done. Apart for the general reasons justified in section 2.2.7, the other main reason to choose this mechanism with respect to using two IPs—one of them as a golden-model—was the difficulty to synchronize two SpaceFibre IPs functioning identically, specially upon the occurrence of any failure, which would be impractical.

4.1.1 Limitations

There are three main limitations to this methodology:

- It can only be applied to Xilinx FPGAs due to the usage of elements inherent to their architecture

- It cannot act on registers or BRAM memories, only in the rest of the configuration memory not composed of storage elements
- It can only be used with designs that integrate either a soft or hard processor, since it is the element used to inspect the signals for the occurrence of failures

4.2 Architecture of the injection experiment

There are 4 main components of the architecture that need to be present in this methodology:

1. The devide under test: SpaceFibre IP port.
2. The internal supervisor: LEON3 processor.
3. The test controller: a PC.
4. The injector: Xilinx SEM IP.

This section details the functionality and purpose of each element from the architecture presented in figure 4.1.

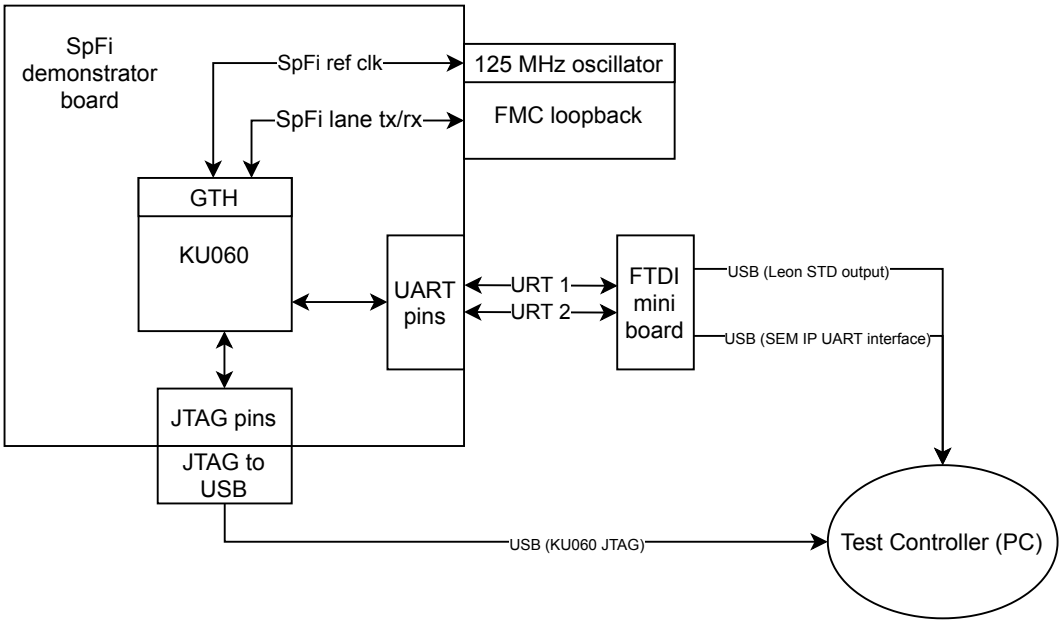

Figure 4.1: Hardware Architecture for the injection experiments

4.2.1 DUT

As mentioned before, the Gaisler/ESA's SpaceFibre IP Port was chosen to perform the injection. It is composed by the IP core itself, along with one control APB block used to read the status signals and write the control signals of the IP and a data generator and checker, also including the APB interface to monitor eventual errors that may not be addressed by the QoS measures of the IP. As explained in section 3.2.5, managing the reset from the APB interface allows to recover the state of the IP after an error is injected

and proceeding to the next one, making each injection independent from the previous one, since all *soft values* and registers are reset.

The IP is configured to work with two simultaneous Virtual Channels. The first virtual channel would produce packets of 1000 words of 32 bits, and the second virtual channel would produce packets of 5000 words of 31 bits. All that information is sliced in frames in the lane layer, with length of 64 words of 32 bits. Both of them are set to use a bandwith of 47% each, with the remaining 6% reserved for broadcast messages.

4.2.2 Internal supervisor

The LEON3 processor working at 50 MHz is used to control all APB signals of the core. Having the combination of a soft-core plus any AMBA-based peripheral, the standard in the space sector, allows to easily monitor all internal signals in a fast and convenient way. Thus, the LEON3 is continuously running a bare-metal C code that includes:

- A UART driver to communicate with the test controller. It uses the UART code provided with the BCC compiler, adapted to implement a simple protocol in which every transaction waits for an ACK, the character 'a' sent from the test controller, before continuing the execution. This simple protocol implements a sofware barrier, allowing to synchronize both threads running in the LEON3 and in the test-controller respectively.

- A SpaceFibre driver to communicate with the DUT. Adapted from the code provided by ESA, it implements functions to read and write to any status or control register, as well as initialization and reset routines to set up the lane connectivity and configuration.

- The FSM controller used during the injection process. It monitors all status signals, looks for deviations from the nominal values and synchronizes with the injection thread to collect all failures, report them and prepare the SpaceFibre IP for the next injection. It is presented with more detail in section 4.4.

4.2.3 Test controller

A Personal Computer (PC) running Python code acts as the test controller. It supervises the experiment, commanding both the internal supervisor and the injector and coordinating them. Besides, it is programmed to obtain information about the status of the internal supervisor and injector and automatically reboot and reprogram the FPGA in case any of them fail during the injection. On top of that, it stores the results from every injection into a log file for further processing. Thus, the Python code is continuously running during the experiment, being the main component of the software architecture. It is presented with more detail in section 4.4.

The test controller is connected to the board with a USB cable that is plugged into the USB interface of a miniboard containing an FTDI chip, which is soldered to transmit two UART connections with one TX and one RX pin each through the USB cable. Another USB cable is connected to the KU060 USB JTAG device. It allows to reboot the FPGA.

4.2.4 Injector

As mentioned before, Xilinx SEM IP is used to inject faults in the configuration memory emulating SEUs. It uses partial reconfiguration, writing values to the configuration memory through the ICAP interface in idle mode, and looking for errors and automatically correcting them in observation mode. It is capable of addressing multiple single-bit errors, as long as they are located in different frames, and to detect the presence of an error in an unknown position thanks to the CRC mechanism. It is configured in the mitigation and testing mode, which allows to inject and repair errors, using both ECC and CRC. The IP was generated to run at the system clock frequency (50 MHz), with the ICAP and FRAME ECC placed in the example design and with error classification disabled. since the injections will be done in the addresses of the essential bits, the classification mode would only be useful for random injection. The example design was generated with the Vivado tool, and the VHDL sources included in the firmware, instantiating the top level of the example design in the SoC connected to the system clock. The UART interface was connected to the board UART pins.

During the experiment, the automatic reports generated in the observation state when attempting a correction are used to gather information about whether the injection and correction were successful, or if there was any anomaly, such as two or more errors appearing in one injection. The test controller is configured to recognize the different reports offered by the SEM IP upon any circumstance. These example reports are extremely useful, containing the information of the amount of errors corrected, their location, and whether an injection was successfully performed or not, indicated by the transition from the idle state to the injection state. The example reports can be found in appendix D of [19].

4.3 Testing framework

Before proceeding with the injection, there is a number of steps that need to be performed: preparing the injection addresses for the test, the amount of time the DUT will run between each injection and the recovery mechanism. These topics are presented in this section.

4.3.1 Addresses for the injections

As explained in section 2.2.3, the configuration memory is stored in a distributed manner inside the FPGA, programming the functionality of the CLBs, BRAM memories, DSPs and routing interconnections, among others, within the FPGA. It is organized in frames arranged in rows and columns of the same configurable resource, with several frames per column, forming clock regions inside the FPGA. To perform the injection, the frame addresses, and words and bits within those frames, that belong to the DUT must be obtained.

For that matter, Xilinx offers the essential bits technology [19] that provides, along with the bitstream, a mask that indicates whether that configuration bit is used in the design or not, stored in the ".ebd" file. To generate the *ebd* file, the following line must be added to the constraints file: `set_property BITSTREAM.SEU.ESSENTIALBITS YES [current_design]`. The mapping from the file to the configuration memory is straight-

forward: each line of the file represents a configuration word, and each character in a line a configuration bit, the line starting with the LSB and finishing with the MSB. For the Ultrascale technology, each frame is composed of 123 words. The first frame, which corresponds to the first 123 lines, is a dummy frame, and then each frame is located according to their linear address. This is, starting from the frame with linear address 0, increasing one by one without interruption until the last frame of the configuration memory. Then, the problem is reduced to locating the bits within the *ebd* file that belong to the DUT in which the injection is to be performed.

To do so, the first step is to constrain the DUT in a known region of the device. That can be done using the Xilinx Vivado tool, opening the synthesized design in the device view—after synthesis has been successfully completed— and clicking the button for creating a new *pblock* in the toolbar. In this work, the maximum granularity obtained to translate a region of the device to its location in the configuration memory was one configuration column within a clock region, i.e., a column that comprises the height of a full clock region, and contains several frames. Each column is composed of one kind of configuration blocks [19]: CLBs, BRAM blocks, DSPs, etc. Therefore, the DUT must be constrained in a *pblock* that contains one or more complete columns. An example of a clock region with two *pblocks* is shown in figure 4.2. For that case, the pblock in the left contains 6 columns of CLBs and one of BRAM blocks—in purple— and the pblock in the right is formed by 3 CLB columns and one BRAM column.

For the case of this work, two designs were analyzed with the injection: one in which each of the main layers of the SpaceFibre protocol—lane, broadcast, interface, virtual channel and retry—was contained in a separated *pblock*, used to analyze the vulnerability of each layer, and another one in which the whole IP was contained within a single *pblock*. After drawing the *pblocks*, entities from the design's netlist can be assigned to them by left-clicking on them and selecting "floorplanning → assign to *pblock*". Before saving the design with the *pblocks*, it is important to check that each of them has enough resources to implement the logic inside. It can be confirmed by clicking on each of them, and checking in the "Pblock Properties" tab the physical resource estimates given by Vivado.

Once the synthesized design is saved, one option must be set in Vivado before generating the bitstream. From the toolbar, navigating to "Tools → settings → project settings → bitstream, the box "logic_location_file" shall be checked. This file contains the location of the distributed memory elements—shift registers, BRAMs and regular registers—that are masked by Vivado during a configuration readback, but can be accessed with the readback capture through JTAG. In this work, it is used to help obtain the physical address related to a particular *pblock*.

After the bitstream has been generated, there are two ways of obtaining the addresses of the columns in which the logic was implemented:

- **Using the implemented design:** the location of a particular column and row to obtain the physical address can be visualized by opening the *device view* in the *implemented design*. The row directly corresponds to the Y coordinate of the clock region, shown directly in this view. The column address can be obtained by counting the major columns formed by blocks of the same function, excluding the columns of laguna tiles, the *terminal columns*— whose tile name begins with "INT_TERM"—the DSP columns, the gap columns—whose name starts with "FSR_GAP"—and the

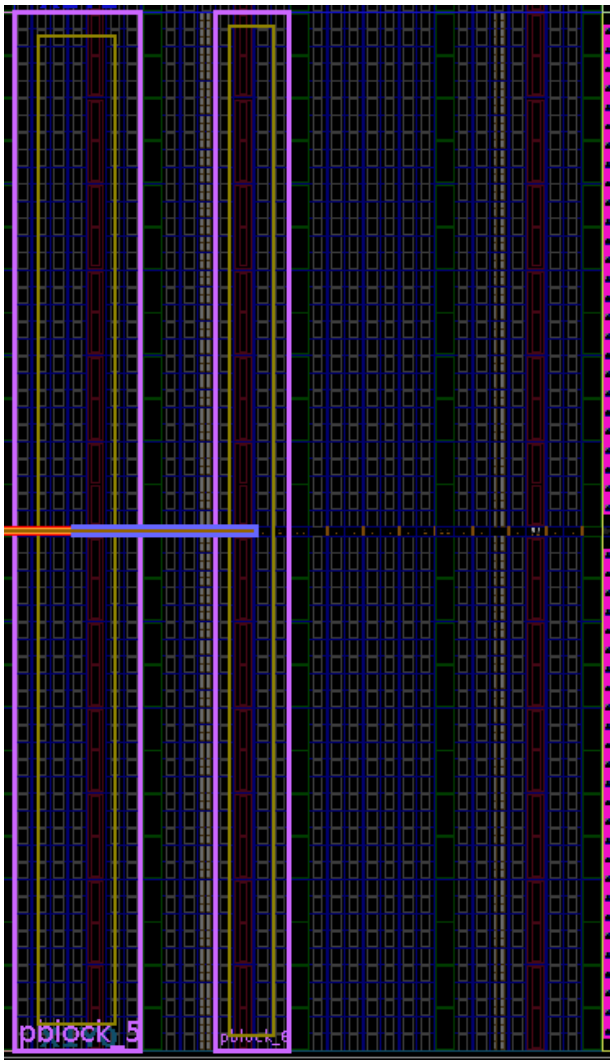

Figure 4.2: Pblocks drawn inside a clock region

gaps that can be appreciated visually, and whose name starts with "CFRM_CBRK". Thus, starting from 0 and counting upwards to the right, row by row, excluding the aforementioned columns it is possible to obtain the column address range for any implementation. Besides, using the device view from the implemented design is recommended to inspect the interconnection blocks that each *pblock* contains, key elements that affect the injection and are a cause of failure within the configuration memory but are not visible in the synthesized design when drawing the *pblocks*. In some cases, such as for the interface layer shown in figure 4.4, the interconnect elements that contain routed signals within the layer—and hence, should be used in the injection—are not included in the *pblock*. For this case, some additional inspection is recommended to identify the columns that should be included in the injection list, and not just including the *pblock* columns.

- **Using the logic location file:** the method presented above can be tedious and more prone to failure for bigger designs that span a large range of configuration columns. In order to help as a guide locating the row and column address of a particular element in the design, the logic location (".ll") file can be used. It only covers memory elements within a design, but it is enough to locate a sub-module

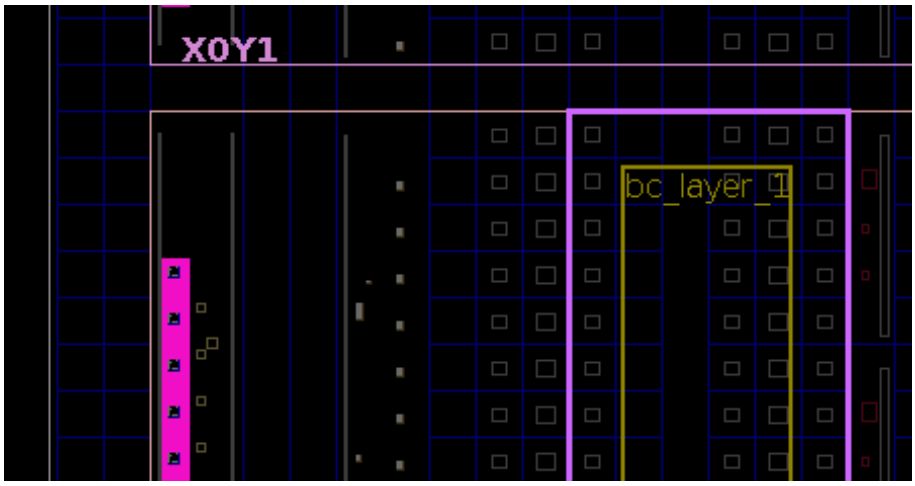

Figure 4.3: Pblocks that contains the broadcast layer, in columns 2 to 6

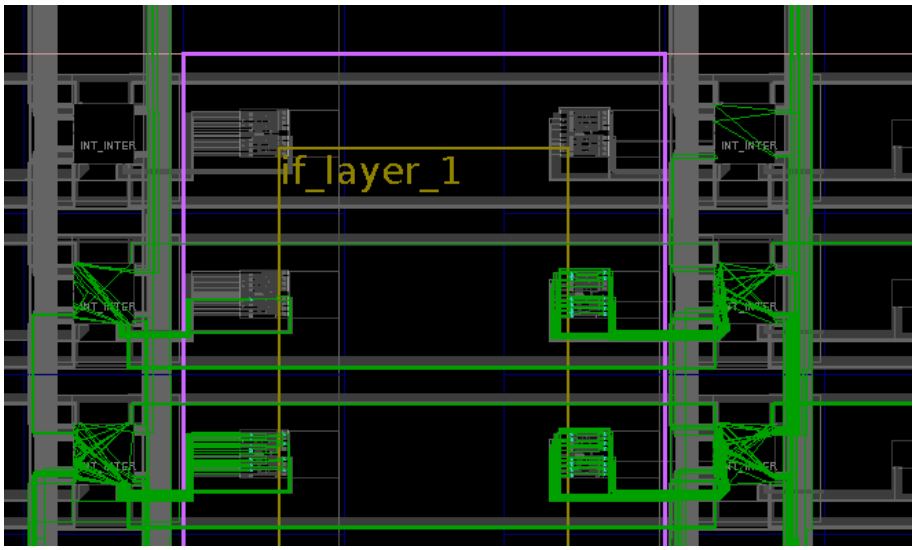

Figure 4.4: Interconnect and CLB columns in the interface layer

that contains some sequential elements. To do so, the name that the sub-module has in the netlist can be used to locate it in the logic location file. That can easily be done by using the integrated search of any text editor. After that, as explained in the header of the file, the physical frame address is located in the third column of the file, in hexadecimal format. From that address, the row and column can be extracted, and used as an indication to where the sub-module is located within the device. With that information, using the device view of the implemented design, the rest of the columns contained in the sub-module can be obtained by manually counting from the reference column obtained. In general, the logic location file will allow to locate most of the CLB and BRAM columns for any design.

Once the set of columns containing the logic for the DUT has been determined, the final step is to obtain the addresses of the frames related to them. Obtaining the physical address from the row and column within the FPGA is straightforward, with the exception of the bits that indicate the frame within a column, since not all columns have a fixed number of frames. Besides, the *ebd* file contains the essential bits located according to their linear address, with the physical address not being specified. Then, a translation

between physical and linear addresses solves both issues at the same time, since the linear address is incremented frame by frame, with no missing values. To do so, again the SEM IP was utilized. It includes a very useful command to translate a linear address to its corresponding physical address. By sending the character 'T' along with a space, and the linear address in the same format as specified in table 4.4.1, the IP will answer with the physical address for that parcitular frame. Thus, writing a script that repeats the process for the 37651 frames that are contained in the configuration memory of the KU060 FPGA, and recording the answer allows to obtain a table that contains both addresses for the same frame.

Finally, with the range of columns where a particular sub-module or design is implemented within the FPGA, and the translation table between linear and physical address, the process of obtaining the essential bits inside a design that belong to the DUT can be automatized. The first step is to translate the row and column to a range of physical frame addresses. Then, with this range, and using the translation table from physical to linear frame address, a list of all linear frame addresses of interest is elaborated. This list can be used to locate the frames inside the essential bits mask, and obtain the essential bits for all the frames that belong to the DUT. Finally, with the frame address and including the word and bit index, the linear configuration address for each essential bit is written to a file in the form of a list of addresses in which injection should be performed.

4.3.2 Time between each injection

A value that should be carefully chosen—especially when the experiment is done in a circuit that does not perform a bounded operation, but a continuous one—is the amount of time in which the circuit is functioning after the injection. This is the amount of time in which the error can propagate and cause potential failures before being corrected. To calculate it, the size of both packets and frames, as well as the line rate were used.

The SpaceFibre Port IP is configured with a frame size of 64x32 bits, that are in charge of encapsulating and sending the content of packets whose biggest size is 5000x32 bits. The line rate used is 2.5 Gbps. Taking into account the 8b/10b encoding, it leaves an effective rate of 2 Gbps. Leaving a time of 1 ms between injections leads to 2 million bits being transmitted, which corresponds to almost a thousand SpaceFibre frames containing more than 12 complete SpaceFibre packets.

The IP is configured for a data bandwidth of 94% splitted in two virtual channels. The remaining bandwidth is used for broadcast messages to control and maintain the connectivity. Since only a 6% of time broadcast information is transmitted, the initially estimated time of 1 ms, corresponding to 12 packets of 5000 bits is not enough. The case in which only 12 packets of 5000 bits occupy the lane and no broadcast packets were sent was possible. Thus, the time selected was 5 ms, statistically ensuring that at least three broadcast packets would be sent, for a total minimum of more than 62 packets transmitted, and almost 5000 frames. This time allows to ensure that all functional states are reached for each injection, maximizing the probability of a configuration error being detected as a functional failure.

The time between injections fixed in this section is the absolute minimum. As mentioned

in section 4.4, the circuit will be kept running while new failures are being produced, thus, in many cases, it lets configuration errors propagate at least one polling loop more than the 5 ms, continuing in case any new error appears during that polling loop, which allows to showcase the behavior of the circuit upon a cascaded error scenario, in which previous configuration errors cause the circuit to enter failure-recovery states—such as a re-transmission—in which, in turn, new errors can be produced. To validate the approach, several independent injection runs were performed, in which the IP would be in different states between each of them, but the amount of failures produced was similar. Therefore, the configuration errors propagated to the states causing almost the same failures in all different runs. The results from these independent runs are presented in chapter 5, and used to estimate the confidence interval of the amount of failures for the complete injection of a module.

4.4 Software architecture

In this section the software architecture is presented. It is composed by two main threads running simultaneously in the LEON3 processor and the test controller. Since both threads work as FSMs with synchronization through barriers, the architecture is presented as an FSM diagram with both threads in parallel. It is then straightforward to convert the diagram to code. Besides, a few indications on how to implement the serial communication drivers are given.

4.4.1 Software FSM

The FSMs are shown in figure 4.5. The left FSM corresponds to the thread running on the test controller, and the right one to the internal supervisor. It can be divided into three main steps:

Initialization

First of all, the FPGA is rebooted to ensure a correct and known state. Then, the serial links are set up in the test controller. The code automatically recognizes the logical serial ports in which the UART connection is mapped by scanning the names shown by the OS. Then, the PySerial API is configured with the parameters shown above, and prepared to transmit data.

In the LEON3 processor, after the reprogramming is done, the UART driver is set up and enabled. Then, the IP is reset, initializing in order the GTH transceivers and the DUT , once the reference clock is locked. After that, the data generator and checker is configured through the APB interface, and the IP is enabled to establish lane connectivity.

Once both threads are configured to transmit, an initial synchronization that consists of a handshake with double acknowledgment to make sure that both threads are ready to start the experiment is done.

Injection

In this phase, the test controller send the injection command to the SEM IP. It is done by writing an 'N' character, followed by a space and the linear address of the bit to inject,

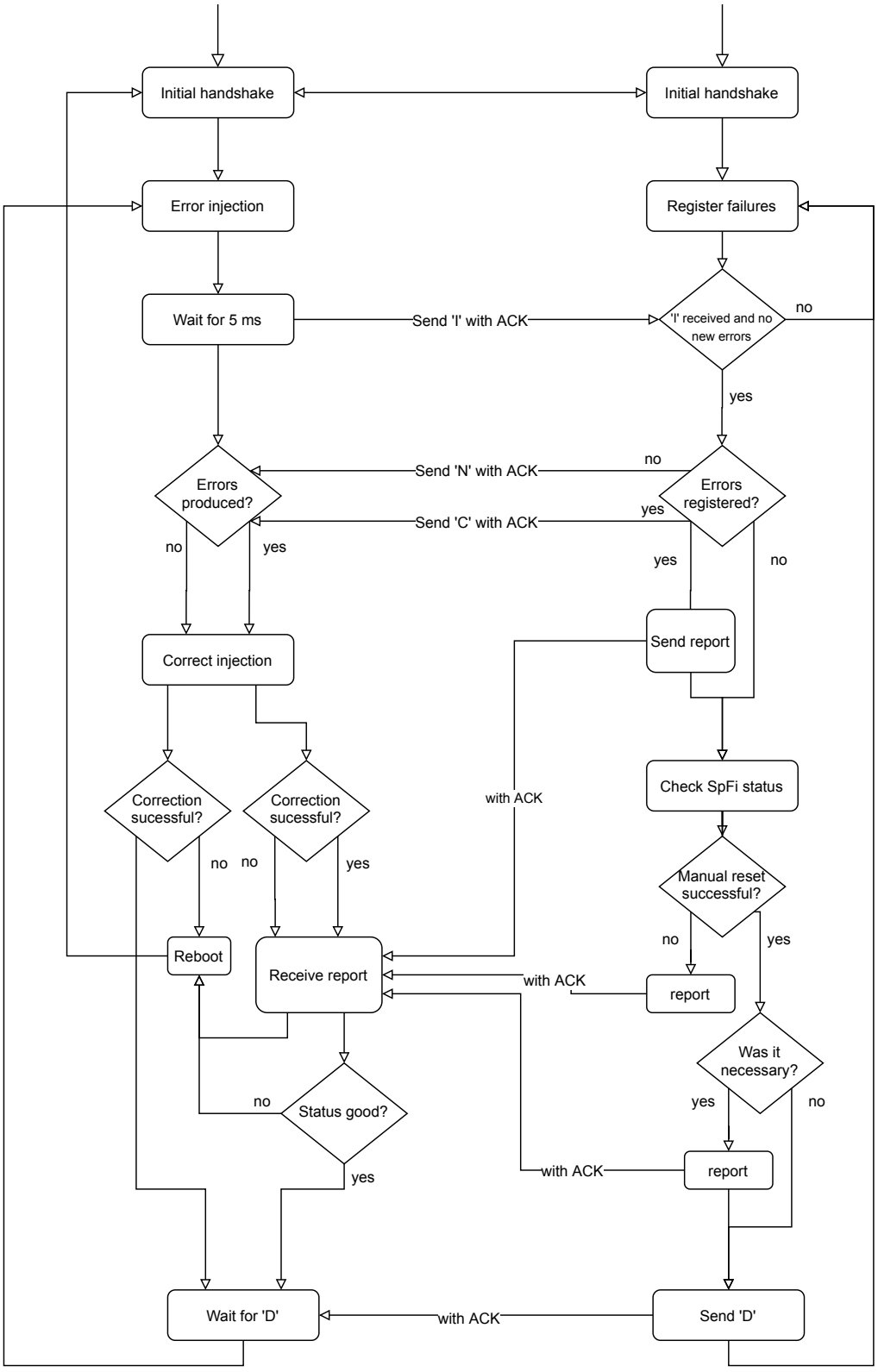

Figure 4.5: Software Architecture for the injection methodology

written in the format specified in table 4.4.1.

Bits	39-36	35-29	28-12	11-5	4-0
Content	0xC	0s	Frame linear address	Word address	Bit

Table 4.1: Linear address format for an injection using the SEM IP in the KU060 device

The frames to inject are written in the same format in a file that contains a list of all configuration bits that belong to the design. Then, the test controller waits for the response of the SEM IP, indicating that the injection was performed successfully by a transition to the injection state and then back to the idle state. After that, the test controller waits for 5 ms before sending an 'I' character to the LEON3 processor, confirming the injection.

In the meantime, the SpaceFibre IP is continuously running in a loop, checking the values for all status registers. If any deviation from nominal behavior or a failure is detected, it is registered in an array of flags, with one position for each potential error. The internal supervisor keeps polling the registers until it has been confirmed that the injection was performed, 5 ms passed after its confirmation **and** there are no new errors registered in the status registers—i.e., if 5 ms passed, but after each new polling loop new errors appear in parameters that had a nominal value, the polling will continue—. This technique prevents the duration of the polling loop to hide potential errors that may appear, making sure that all errors are registered.

Correction, report, status check and recovery

Once the injection phase is over, the LEON3 processor sends a message to the test controller indicating that it exited from the polling loop. If any failure was detected, then a 'C' character is sent and the FSM proceeds to report the failures by sending one by one a short command indicating the occurrence for each of them. If no deviations from nominal behavior were registered, then the 'N' character is sent, and no failures will be reported.

The test controller waits until either an 'N' or 'C' character is received. Then, it commands the SEM IP to transition to the observation state, where it will detect the injected error and automatically attempt to correct it. If any anomaly on the SEM IP's behavior—a double error is found, it does not respond to the command, or an uncorrectable error happens—a reboot is performed after the report has been collected. After that, if an 'N' character was received before, the report is collected, writing to a log file the short codes received for each failure.

Before collecting all failures, an additional check is done in the LEON3 processor, obtaining the value of the lane state. This check is done to make sure that the correction has happened before it, and to gather further information about the impact of the configuration fault that was injected. If the lane state was not active, it means that the IP could not recover the connectivity, even after the correction of the error, and it is notified and logged in the test controller. In any case, after the lane state check, a software reset is attempted, with a re-initialization of the IP. If this re-initialization is not successful, it is also notified, and the FPGA is rebooted and initialized. If the re-initialization is successful, it is indicated by a command. This means that the IP recovered its internal

state and is running with a configuration clear from errors. Then, both threads repeat the injection process.

To reboot the FPGA, Vivado is launched as a sub-process in batch mode from the python logic to run a *tcl* script that reboots the FPGA using the JTAG connection with the following command:

```
vivado -mode batch -nojournal -nolog -notrace -source reboot.tcl
```

4.4.2 Serial Communication

The serial communication between the PC and the LEON3 processor and the SEM IP in the KU060 FPGA use both the same protocol with acknowledgment for each command. Both are set up to run at 115200 bauds per second, a byte size of 8 bits, no parity check and one stop bit. These settings were chosen to maximize performance and minimize latency. The PySerial API is used to write driver functions that manage the communication for both serial interfaces, creating a driver that is used to directly send and receive commands through both channels. Special care is taken to manage the input buffers from each channel, using the API functions that read character by character from the buffer and regular-expressions implemented to recognize the different commands and reports. Using this implementation allows to also reduce the latency to the minimum, and avoid potential buffer overflow scenarios.

4.5 Evaluated parameters

In this section, the status parameters from the SpaceFibre Port IP that are collected during the injection are presented along with their severity. Except the re-initialization parameters, all status parameters are monitored in the polling loop during the injection. It is important to remark that only the occurrence in the deviation from the nominal value of these parameters is collected, but not the amount of times this happens. It is expected that, when the configuration memory is affected and the logic operation of any circuit is changed, the occurrence of these failures will be recurrent unless they are corrected. The goal is to compute the amount of configuration bits sensitive to an error in the configuration memory, and to characterize the kinds of failures that a configuration bit will cause.

4.5.1 Severity of the failures

Before describing the parameters, it is useful to classify the severity of the failure detected by each of them. Each parameter was assigned a severity depending on whether a deviation was expected under nominal behavior, or if recovery from the failure was contemplated in the SpaceFibre protocol. Besides, three other levels of severity were given to failures to recover lane connectivity **after** the configuration error was corrected. When analyzing the results, each essential bit causing a failure will be assigned the severity of the deviation with the biggest severity caused by the error in that particular address. The levels of severity for this work are then:

- **Severity 0:** failures expected during nominal operation, related mostly with potential noise in the channel. Completely transparent to the rest of the system.

- **Severity 1:** failures not expected during nominal operation. However, the SpaceFibre protocol contemplates recovery measures to correct them in a transparent manner for the application. They may affect the functionality on system level if not addressed immediately, since they may be recurrent and cause the recovery mechanisms to fail.

- **Severity 2:** failures not expected during nominal operation, not transparent to the application layer or without a recovery mechanism. These failures are critical for the operation and the functionality of the device, and will propagate into other failures in more elements of the system, potentially causing harmful failures at higher levels. The reception of erroneous data in the application level, or a fatal protocol error overriding data in the buffers are examples of failures categorized with this severity.

- **Severity 3:** related to the state of the circuit upon error correction, a failure of severity 3 is caused when the circuit is not able to recover the connectivity even after the correction, causing a need for an explicit reset of the IP and the transceiver managed from the system level. This is, either from the software (internal supervisor) or external to the FPGA. It is recoverable if, and only if, this scenario is contemplated in the system application.

- **Severity 4:** this level was originally not contemplated. However, after the first injection runs, it arose as highly critical. The scenario in which a manual reset from higher a level, i.e., application software, is not necessary but, if issued, or upon other eventual error that causes the lane to disconnect or reset, will cause a severity 5 failure, requiring the reboot of the whole FPGA to recover, is categorized with this severity. This scenario was found when, after an error was injected and corrected, the IP was reset to begin the next injection in a safe state, but this reset would fail and the DUT would not reach a functional state unless an FPGA reboot was issued.

- **Severity 5:** the most critical failure, happens when the SpaceFibre IP is not able to recover functionality by any other mechanism than rebooting and reprogramming the FPGA. In a mission scenario, this would mean a forced interruption of the system-level functionality, which could be extremely harmful.

4.5.2 Application layer

These parameters are obtained by the application layer, checking the outputs from the SpaceFibre IP and comparing them to the inputs in the data generator and checker block:

- **Error Counter:** number of received words (32-bit words for this configuration) that differ from the value sent by the data generator. For this test, a 32-bit counter generates the values to be sent through each virtual channel and broadcast channel, and, on the reception side, the values are checked comparing them to the previous received value and ensuring that they are consecutive. Else, one message was lost or modified, and the error counter is increased. An error counter different from zero is a failure with severity 2.

- **Error End of Packet Counter:** number of packets that were interrupted during transmission—for example, by a lane disconnection is marked by the reception of an Error End of Packet (EEP) word. An EEP counter bigger than zero is a failure with severity 1.

- **Late Frame Counter:** number of frames that were unusually delayed, or a retry was needed to transmit them due to a link error. If they are a broadcast frame—with some control information—they should be ignored by the application. A late frame counter bigger than zero is a failure with severity 1.

4.5.3 Virtual channel layer

The Virtual channel layer is in charge of managing the flow control for each channel, in relation to the allocated bandwidth and the reception of Flow-Control Tokens (FCTs) that indicate the amount of free space in the reception buffer of the far-end:

- **Bandwidth Overuse:** a virtual channel is using more bandwidth than allocated. Severity 0.
- **Bandwidth Under-use:** the maximum idle time in the channel is reached, and thus, the channel is not currently transmitting data. Severity 1.
- **Input buffer overflow:** fatal protocol error that is caused when data was received with a full buffer. It indicates an error in the management of the FCT tokens. Severity 2.

4.5.4 Retry layer

The retry layer is in charge of addressing CRC errors by keeping the frames in a buffer until they are acknowledged and re-sending them if necessary:

- **CRC-16 error:** frames arriving with more than one erroneous bit through the lane. Severity 1.
- **CRC-8 error:** frames arriving with a single-bit error through the lane. Severity 0.
- **Frame error:** invalid frame received. E.g., a different type of frame from the one indicated with the start of frame word. Severity 1.
- **Sequence error:** after checking the sequence numbers, data arrives to the receiver out of order or repeated. Severity 0.
- **Retry counter overflow:** too many errors were produced, and the counter for the amount of retry attempts overflows. The link is reset automatically after the overflow. Severity 1.
- **Protocol error:** an ACK or NACK were received with an inconsistent sequence counter, resulting in a link reset. Severity 1.

4.5.5 Lane layer

The lane layer is in charge of managing the connectivity from both ends of the link, including establishing it and re-initializing it upon signal loss. During initialization, several configuration parameters such as the lane capabilities and polarity are exchanged to ensure a correct communication:

- **Far-end link reset:** the far-end has reset the link, and sent lane capabilities, causing the near-end to reset the link initialization FSM. Severity 1.
- **Far-end standby:** the far-end sends stand-by words, causing the lane state to move to standby. Severity 1.
- **Timeout:** set when the link initialization FSM fails to reach connectivity and the process timed out. Severity 1.
- **Far-end loss:** loss of signal words received, sent by the far-end after losing connectivity with the near-end. Triggers a new handshake and initialization of the link. Severity 1.
- **RXERR words received:** the lane layer automatically substitutes words that are not recognized by the IP—for example, because their 8b/10b encoding or disparity is not valid, or a signal loss, standby or init word are received unexpectedly—by the RXERR words. Most of the time, their occurrence is related to noise in the channel, and can be used to estimate the Bit Error Rate (BER). The IP also integrates a counter of RXERR words received, and when it overflows, a reset of the lane FSM is triggered. Severity 0.
- **Lane state:** the state for the lane FSM—also called initialization FSM—is an indicator of the current state of the IP, and whether it established connectivity with the far-end successfully. Since the injection and polling loop are issued after the lane is initialized and in active state, any deviation from this state is considered a failure with severity 1.

Chapter 5

Results and discussion

In this chapter, results for the different injection runs are presented, reasoning about the potential causes and consequences of the obtained vulnerabilities, as well as about measures to overcome them. For each experiment, results on the amount of configuration bits causing a failure, the amount of bits segregated by their severity and the DVF are presented.

First, the implementation results in terms of resources and configuration bits are presented along with a statistical analysis to validate the rest of the results. After that, results from every significant layer of the SpaceFibre IP are shown, to finalize with the results from the whole SpaceFibre IP, as well as several comments on limitations shown by the SEM IP during the analysis.

5.1 Implementation results in the KU060

First of all, it is obvious that the vulnerability of a particular design depends on the amount of logic that it uses, because the more configuration memory used, the most likely an SEU is to affect it. Therefore, as first results, occupation of the SpaceFibre Port IP, and each individual layer in which an injection study is carried out, are reported. It is important to highlight that these reports only include the part of the design that is being injected, with the APB clock-domain-crossing, GTH transceiver logic and data generator and checkers being omitted. Results of implementing the Port in the Kintex KU060 FPGA, divided by functionality layer or block and for the whole IP are presented in table 5.1.

Module	CLUTs	MLUTs	FFs	RAMB18	RAMB36	Configuration Bits
Broadcast Layer	55	0	28	0	0	11543
Interface Layer	752	24	574	0	0	180487
Lane Layer	496	0	182	0	0	101246
Retry Layer	1146	16	820	1	2	244887
VC Channel	2439	0	1848	16	0	601725
SpFi IP Port	3105	40	2172	6	1	719108

Table 5.1: Resource utilization in the KU060 FPGA for each layer of the SpaceFibre IP Port

5.2 Results for the lane layer

5.2.1 Statistical validation

Before showing the results of the work, a short statistical validation will be presented. For that, a series of 10 runs repeating the same injection procedure with identical addresses and DUT was performed. Ideally, 100 or more runs would provide a more solid result. However, due to the time-constrains in the project only 10 could be performed, which was decided were enough given the similarity of the results among different runs. The mean and variance during the 10 runs are extracted for each parameter measured during the injection to monitor the deviations from the expected behavior. Finally, the absolute and relative confidence intervals are presented for each parameter.

This series of injection runs were not performed for the whole SpaceFibre IP due to time constraints. Instead, the Lane Layer section of the IP was chosen, since it counts with enough resource count and diversity to be representative—with 101,246 configuration bits—while being small enough to allow 10 injection runs to complete in a short amount of time.

The first step is to obtain the amount of occurrences in deviation of the nominal behavior for each parameter in each run. Since these were the first set of injection runs performed, bandwidth overuse and underuse were not measured at the time, and they are omitted. After that, the average value (μ) for each parameter is computed and use to calculate the standard deviation (σ) using its expression, where N is the number of runs:

$$\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \mu)^2} \tag{5.1}$$

By inspection of the values that a parameter takes for different runs and their deviations from the average, results for each parameter were modeled as a gaussian distribution around their mean. Then, the 95% confidence interval (CI) can be obtained as:

$$(\bar{x} - z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}}, \bar{x} + z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}}) \tag{5.2}$$

Where $z_{\alpha/2} = 1.96$ [46]. Finally, also the relative confidence interval (CIR) is obtained by dividing the confidence interval by the average value for each parameter. Results are presented in table 5.2.

Parameter	1	2	3	4	5	6	7	8	9	10	Average	σ	CI	CIR
Sequence error	8196	8177	8173	8190	8170	8154	8131	8163	8191	8203	8186	9.51	±13.18	±0.16%
CRC-8 error	6050	6032	6053	6029	6002	6043	6025	6029	6061	6068	6041	9.0	±12.47	±0.21%
CRC-16 error	3141	3154	3159	3113	3112	3143	3117	3105	3135	3145	3147	6.49	±8.98	±0.29%
Frame error	2751	2778	2752	2747	2753	2741	2737	2774	2776	2754	2764	13.49	±18.69	±0.68%
Link reset	9	10	10	16	10	12	10	7	15	15	9	0.71	±0.98	±10.89%
Protocol error	34	31	30	37	37	34	34	30	32	32	32	1.58	±2.19	±6.85%
Channel error	743	742	727	735	701	728	706	724	690	721	742	0.71	±0.98	±0.13%
Signal lost	371	369	368	371	352	363	364	358	353	368	370	1.0	±1.39	±0.37%
Lane not active	428	427	428	424	431	419	421	428	428	421	427	0.71	±0.98	±0.23%
Stand-by	127	129	131	131	128	128	130	126	132	131	128	1.0	±1.38	±1.08%
Init. timeout	114	117	110	111	117	119	110	115	106	104	115	1.58	±2.19	±1.91%
Input overflow VC1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Input overflow VC2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Error received VC 1	197	185	198	188	201	197	191	202	193	188	191	6.0	±8.32	±4.35%
Error received VC 2	198	191	190	182	198	199	202	200	190	191	194	3.54	±4.9	±2.52%
Error received BC	46	41	44	45	40	41	44	44	44	46	43	2.54	±3.53	±8.22%
Interrupted packet VC 1	30	26	28	36	33	32	27	23	37	34	28	2.0	±2.77	±9.89%
Interrupted packet VC 2	28	27	26	38	28	35	32	31	33	33	27	0.71	±0.98	±3.62%
Late broadcast frame	3939	3947	3929	3943	3917	3883	3952	3940	3965	3949	3943	4.0	±5.54	±0.14%
Too many CRC errors	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Manual reset needed	17	22	18	17	19	6	9	21	14	10	19	2.54	±3.53	±18.59%
Manual reset failed	6	8	7	8	6	5	7	8	8	7	7	1.0	±1.39	±19.79%
Manual reset would fail	269	244	294	257	262	264	282	276	264	244	256	12.49	±17.31	±6.76%

Table 5.2: Parameter measurements for 10 injections in the Lane Layer and its average, standard deviation and confidence interval

Severity	Amount of bits	% Essential bits
0	3523	3.48%
1	6056	5.98%
2	227	0.22%
3	11	0.01%
4	269	0.27%
5	6	0.006%

Table 5.3: Amount of critical bits distributed by severity for the lane layer

5.2.2 Other results

One of the goals of the injection analysis is to derive the DVF (see section 2.2.7). For that, first, the amount of critical bits must be obtained. The critical bits are defined as those bits that, when affected by an SEU, will cause a functional failure. As introduced in section 4.5, for this study, a functional failure will be a deviation in a parameter with severity of 1 or more, since it is not expected under nominal operation. It is important to note that the values for each parameter presented in table 5.2 count the amount of configuration bits in which a particular parameter experiences a deviation throughout the injection. Still, it is common that more than 1 parameter is affected by an error in the same configuration bit. Hence, the critical bits are not obtained by summing values from the aforementioned table, but from counting the amount of bits in the injection that have at least one deviation in a parameter with severity 1 or higher. The obtained amount of critical bits in the lane layer are 6572. The amount of essential bits—configuration bits that belong to the Lane Layer— is 101246. Then, the DVF can be obtained as:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{6572}{101246} = 0.065 = 6.5\% \tag{5.3}$$

Those critical bits can be divided by the severity of the worst failure that a SEU would cause if it affected that particular bit. Results for the first run are shown in table 5.3

5.2.3 Discussion of the results for the lane layer

Statistical validation

The results for the confidence interval show that the values obtained for just an injection run accurately reflect the critical bits, with a low variance and small 95% confidence interval, indicating that most of the values of successive runs would have low dispersion. This result allows to just use one injection run for the rest of the layers, and take the obtained values as statistically significant, saving a substantial amount of time for the injection campaign.

Having almost the same amount of errors being presented in the status interface for 10 different injections shows that, no matter the state of the circuit when the error was injected—the injection time for each bit and the particular state of the circuit for that time is unpredictable given the time variability introduced by the UART injection through the SEM core—running during 5 ms is enough to propagate an error through the circuit and showing in the status register all errors that a particular SEU would cause.

Injection results

Taking as an example the first run, results presented are the amount of bits in which the deviation of the parameter in the column "parameter" differed from its nominal value. The lane layer is in charge of managing the connection at Lane layer. It contains the logic that manages the different connection states: from the initial handshake to establishing a connection to managing an interruption in the connectivity and re-connecting or changing to the stand-by state. It obtains data to transmit from the virtual channel layer and detects lane-level control words used to keep the physical link synchronized. In a fault-free case, the lane should remain in active state after initialization. Thus, a significant amount of critical bits cause failures related to the lane state FSM—lane not active, link reset, signal lost, stand-by and initialization timeout. This is due to the fact that the logic of the lane FSM is affected by the injection. This undesired interruption in the communication causes data errors in a remarkable amount of cases—detected by the retry layer as CRC, sequence and frame errors—. However, most of them are recovered thanks to the retry capabilities, and indicated by the high occurrence of late frames along with the aforementioned errors for the same injection, and only in a reduced number of injections they were propagated to the application level (the data checker) as interrupted frames or data errors.

These results highlight the relevance of inspecting internal signals and error counters rather than just the output of a circuit to check for failures during the injection process. For the case of the lane layer, most of the errors would have been masked by the retry layer, hiding a faulty behavior of the circuit to the application level. This also shows that the SpaceFibre port is capable of effectively handling errors that may occur in the channel, as well as a sudden connection loss.

It is important to stress that only 19 bits caused the need for a manual reset—coming from the application controlling the port, not managed by the port itself—and only in 7 bits would this manual reset fail, meaning that reprogramming the FPGA is the only mechanism to recover the functionality. Moreover, the amount of injections in which a manual reset would fail—i.e., it was not needed to recover functionality, but when attempting to reset the SpaceFibre Port for the next error injection, the lane would not be able to recover connectivity—was 256. Again, the only way to recover the functionality of the circuit was to reboot the FPGA.

Finally, thanks to the approach followed, dividing the critical bits by their severity allows for a more accurate view of the vulnerability of a particular layer. For the case of the lane layer, it can be seen that there is a relatively low vulnerability to high-severity failures, with just around 500 critical bits causing a failure that SpaceFibre is unable to address—i.e. severity 2 or greater—while most of the critical bits will produce failures with a severity of 1, which the QoS capabilities of SpaceFibre will address. However, as mentioned before, if configuration errors are not corrected with scrubbing, errors with less severity interrupting the communication may compromise the system. Nevertheless, scrubbing alone should be enough to protect this layer effectively.

5.3 Results for the retry layer

The results for the injection in the retry layer are covered in table 5.4.

Parameter	Value
Sequence error	19081
CRC-8 error	6418
CRC-16 error	7723
Frame error	4440
Link reset	4
Protocol error	1962
Channel error	2056
Signal lost	1611
Lane not active	1312
Stand-by	6
Init. timeout	40
Input overflow VC1	9
Input overflow VC2	32
Error received VC 1	7024
Error received VC 2	7040
Error received BC	6243
Interrupted packet VC 1	65
Interrupted packet VC 2	61
Late broadcast frame	13323
Too many CRC errors	6273
Bandwidth Overuse VC0	3
Bandwidth Overuse VC1	3
Bandiwdth Underuse VC0	2
Bandiwdth Underuse VC1	2
Manual reset needed	17
Manual reset failed	0
Manual reset would fail	3

Table 5.4: Parameter measurements for the injections in the retry layer

5.3.1 Other results

The amount of critical bits with severity 1 or higher obtained for the retry layer are 28804, and then, the DVF is:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{28804}{244887} = 0.1176 = 11.76\% \tag{5.4}$$

Those critical bits can be divided by the severity of the worst failure that a SEU would cause if it affected that particular bit. Results for the first run are shown in table 5.5

5.3.2 Discussion

The retry layer is in charge of detecting CRC and frame errors and managing the retransmission of the corresponding packets to mask the occurrence of errors to higher layers such as the virtual channel layer and the application layer. Compared to the lane layer, it is more vulnerable to errors in configuration memory, as shown by its higher DVF. This is due to the fact that an error in the logic, in charge of managing the error

Severity	Amount of bits	% Essential bits
0	7194	2.94%
1	16488	6.73%
2	12296	5.02%
3	17	0.07%
4	3	0.01%
5	0	0%

Table 5.5: Amount of critical bits distributed by severity for the retry layer

recovery in the link, can easily cause an undesired or wrong re-transmission, in turn causing additional errors. Besides, it fails to prevent the errors from reaching the application layer. As it can be seen, with a similar amount of injections in which CRC errors occurred as compared with the lane layer, critical bits that would cause these errors to arrive to the application layer were around 30 times higher.

On the other hand, the failures related to the lane FSM and reset mechanism are less common than in the case of the lane layer. Even though the critical bits in which the lane was not in the active state were more than in the lane layer, most of them were due to protocol errors—inconsistent acks or nacks coming from the faulty recovery layer or data errors—that cause the lane to be reset. However, no events in which the manual reset of the lane would fail were recorded.

It can be concluded then, that this layer is highly vulnerable to errors in configuration memory, with an SEU in this layer causing errors to go undetected and arrive to the application layer, since all error detection and correction mechanisms fail. An interesting failure mode that shows its vulnerability is that the retry buffer overflows—a very critical error that should happen when there are too many CRC errors to handle—with a similar occurrence in CRC errors compared to the lane layer, where this overflow does not happen since the error handling is not affected by the injection.

This is also reflected by the severity of the critical bits, with a great amount of critical bits having a severity of 2 or higher. Nonetheless, most of them are constituted by data errors that arrive to the application layer and not by errors in the lane FSM as was the case for the lane layer.

5.4 Results for the virtual channel layer

For the injection campaign, the IP was configured to handle two lanes simultaneously. Results for the injection to this layer are reported in table 5.6.

5.4.1 Other results

The amount of critical bits with severity 1 or higher obtained for the virtual channel layer are 18914. Then, the DVF is:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{18914}{601725} = 0.0314 = 3.14\% \tag{5.5}$$

Parameter	Value
Sequence error	3821
CRC-8 error	215
CRC-16 error	886
Frame error	1911
Link reset	3
Protocol error	10
Channel error	2700
Signal lost	2673
Lane not active	2296
Stand-by	0
Init. timeout	0
Input overflow VC1	94
Input overflow VC2	138
Error received VC 1	13544
Error received VC 2	13855
Error received BC	9786
Interrupted packet VC 1	6651
Interrupted packet VC 2	490
Late broadcast frame	80
Too many CRC errors	104
Bandwidth Overuse VC0	114
Bandwidth Overuse VC1	298
Bandiwdth Underuse VC0	2
Bandiwdth Underuse VC1	1
Manual reset needed	16
Manual reset failed	0
Manual reset would fail	0

Table 5.6: Parameter measurements for the injections in the virtual channel layer

Those critical bits can be divided by the worst severity of the failure that would be caused if a SEU affected that particular bit. Results are in table 5.7

5.4.2 Discussion of the results

The virtual channel layer is in charge of managing the buffers and flow-control for all different virtual channels. Hence, the main source of critical errors is related with the failure of the flow control, which leads to an overflow of the input and output data buffers of the virtual channels. This is a critical failure, since data that has not been sent to the TX buffer or received in the RX buffer is overwritten by new data, causing its loss. This is reflected by the huge amount of configuration bits that cause the reception of wrong data in both virtual channels—Error received VC 1 and 2, flagged by the data checker at the application level—and, simultaneously, triggering the lane reset after the communication protocol fails due to the data in the virtual channel having been overwritten. Besides, it is also reflected with the bandwidth overuse for both virtual channels caused by the failure of the flow-control mechanism. On the other hand, no errors that would make rebooting

Severity	Amount of bits	% Essential bits
0	6703	1.11%
1	2721	0.4%
2	16177	2.66%
3	16	0%
4	0	0%
5	0	0%

Table 5.7: Amount of critical bits distributed by severity for the virtual channel layer

the FPGA registered, neither any error when resetting the core manually.

As a conclusion, even though the vulnerability of the virtual channel layer is, in proportion to its size of the FPGA's used resources, small, a failure in one of the critical bits is likely to end-up causing errors that cannot be addressed by the retry layer, since that layer lies "under" the virtual channel layer and therefore, those errors will reach the final application, with the SpaceFibre communication channel failing to deliver correct data with potential harmful effects. This fact is again indicated by the big amount of critical bits that, when affected, would cause a failure with severity 2 or greater. In absolute terms, it introduces a similar vulnerability to the IP with respect to the retry layer.

5.5 Results for the interface layer

Results for the injection to this layer are reported in table 5.8.

5.5.1 Other results

The amount of critical bits with severity 1 or higher obtained for the interface layer are 29239, and then, the DVF is:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{29239}{180487} = 0.1628 = 16.28\% \tag{5.6}$$

Those critical bits can be divided by the worst severity of the failure that would be caused if a SEU affected that particular bit. Results are in table 5.9

5.5.2 Discussion of the results

The interface layer lies under the lane layer, and is connected directly to the high-speed transceiver. It is in charge of decoding the 8b/10b words, word synchronization and includes an elastic buffer to fix clock alignment and skew. As the results show, most failures produced from the injected errors in the configuration memory can be fixed by the upper layers of SpaceFibre. The most common failure was related to invalid values received from the channel, most likely caused by an error with word synchronization. Besides, other errors such as frame, sequence and CRC errors were mostly corrected, resulting only in a few configuration errors arriving to the data checker with wrong values.

However, the loss of word synchronization also generated more relevant failures: since an error in this layer affects all kinds of received data, it caused the biggest occurrence of

Parameter	Value
Sequence error	25850
CRC-8 error	12489
CRC-16 error	7706
Frame error	17705
Link reset	0
Protocol error	35
Channel error	20589
Signal lost	15764
Lane not active	19840
Stand-by	75
Init. timeout	6331
Input overflow VC1	0
Input overflow VC2	0
Error received VC 1	463
Error received VC 2	464
Error received BC	122
Interrupted packet VC 1	108
Interrupted packet VC 2	95
Late broadcast frame	18447
Too many CRC errors	14647
Bandwidth Overuse VC0	0
Bandwidth Overuse VC1	0
Bandiwdth Underuse VC0	0
Bandiwdth Underuse VC1	0
Manual reset needed	120
Manual reset failed	36
Manual reset would fail	27

Table 5.8: Parameter measurements for the injections in the interface layer

CRC errors detected in the retry layer, creating overflows in the error recovery buffer, triggering the reset of the lane in as many as 20000 different configuration bits during the injection. Since they can be handled by the SpaceFibre QoS mechanisms, most of these failures are classified as severity 1 failures. Again, if failures are addressed with scrubbing, this is not a highly vulnerable layer, since most failures can be automatically recovered. Nonetheless, the interface layer is the most sensitive layer in absolute terms with respect to failures of severity 3 and 5.

5.6 Results for the broadcast layer

The broadcast layer is the smallest analyzed layer, and also the least vulnerable to SEUs. The injection process produced only a few errors, being only significant the arrival with errors of broadcast messages, occurring in 35 configuration bits.

Severity	Amount of bits	% Essential bits
0	1166	0.64%
1	28481	15.7%
2	611	0.34%
3	84	0.047%
4	27	0.015%
5	36	0.019%

Table 5.9: Amount of critical bits distributed by severity for the interface layer

Severity	Amount of bits	% Essential bits
0	23	0.19%
1	4	0.035%
2	38	0.33%
3	0	0%
4	0	0%
5	0	0%

Table 5.10: Amount of critical bits divided by severity for the broadcast layer

The amount of critical bits obtained for the broadcast layer are 42, and then, the DVF is:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{42}{11543} = 0.0036 = 0.36\% \tag{5.7}$$

Those critical bits can be divided by the worst severity of the failure that would be caused if a SEU affected that particular bit. Results are in table 5.10

5.7 Results for the SpaceFibre IP core

Finally, an injection run for the top level of the SpaceFibre IP core was performed. This includes all previously shown layers, as well as all the additional glue logic and control integrated in the top level. The errors produced by SEUs for the injection of the IP are shown in table 5.11.

5.7.1 Vulnerability factor

The amount of critical bits with severity 1 or higher obtained for the SpaceFibre IP are 87258, and then, the DVF is:

$$DVF = \frac{NB_{critical}}{NB_{essential}} = \frac{87258}{719108} = 0.1213 = 12.13\% \tag{5.8}$$

Those critical bits can be divided by the worst severity of the failure that would be caused if a SEU affected that particular bit. Results are in table 5.12

5.7.2 Discussion

The most important factor to take into account when interpreting the data is that, unlike for all previous runs, this time the IP is not forced to be separated in the FPGA layout,

Parameter	Value
Sequence error	63963
CRC-8 error	27137
CRC-16 error	28936
Frame error	23170
Link reset	14
Protocol error	2386
Channel error	19945
Signal lost	15497
Lane not active	14735
Stand-by	223
Init. timeout	4739
Input overflow VC1	51
Input overflow VC2	119
Error received VC 1	11270
Error received VC 2	10919
Error received BC	8595
Interrupted packet VC 1	251
Interrupted packet VC 2	346
Late broadcast frame	45947
Too many CRC errors	24984
Bandwidth Overuse VC0	26053
Bandwidth Overuse VC1	26219
Bandiwdth Underuse VC0	0
Bandiwdth Underuse VC1	0
Manual reset needed	172
Manual reset failed	28
Manual reset would fail	26

Table 5.11: Parameter measurements for the injections in the SpaceFibre IP

but implemented inside a region that contains the whole logic, as done in a regular design without restrictions. This allows for an optimization from the synthesis and place and route tools, producing a significantly smaller circuit.

It is easy to see that the vulnerability of the whole IP offers results that are close to the sum of vulnerabilities for each layer, after taking into account certain reductions in the total number due to the optimization. This optimization also causes a drastic reduction in the configuration bits used by the design, which, with a comparable amount of failures as in the separated design, accounts for a slightly higher DVF value than expected. Apart from that, no new information can be extracted about the failure modes and their cause than the ones introduced individually for each layer. This highlights the usefulness of performing separate injections, since they allow to obtain more detailed information about the different failure modes accountable for each layer, and design a more effective mitigation strategy. On the other hand, the overall injection in the implemented design *as-is* allows to derive more accurate results for the expected reliability of the design during any projected mission. These two later points are discussed in the section 5.9.

Severity	Amount of bits	% Essential bits
0	39192	5.45%
1	63545	8.84%
2	23465	3.26%
3	144	0.02%
4	26	0.003%
5	28	0.003%

Table 5.12: Amount of critical bits divided by severity for the SpaceFibre IP

Finally, according to Xilinx, between 2 to 10% of configuration memory bits are critical to the function of a circuit. The slightly higher figure obtained (12%) is attributed to the fact that the methodology followed allows to observe more failures than most common methodologies. As an example, if only the observable failures were inspected, the resulting DVF would have been 3.29%. In order to obtain a more accurate result from external observation, a careful bandwidth measurement would have been needed, deriving the masked failures as reductions in the effective bandwidth due to the re-transmission, but without obtaining any information about its cause.

5.8 Special events and limitations when using the SEM IP

During the injection experiments performed as part of the work, three anomalies in the SEM IP were observed and are presented in this section, along with hypotheses about their cause.

5.8.1 ECC bits

The usage of the SEM IP has one additional limitation that is not listed in the user manual. Apart from not being able to inject into BRAM memories and registers—they are masked for both read and write operations into configuration memory—this work found out that the ECC bits embedded into every frame of configuration memory, and used to detect and correct errors were also masked for write operations from the SEM IP. This was detected by analyzing a list of the frame addresses in which the injected errors were not corrected, i.e., no error was detected by the ECC or CRC mechanisms after an injection to that particular bit—.

To make sure that these events were not functional failures of the SEM IP, manual queries to the locations in which errors were not corrected were performed both before and after injection. Queries showed that these errors were not being detected and corrected by the ECC mechanism because they were not being injected. Then, by obtaining the word address within the frame for all bits in which errors had not been injected, it was found out that they belong to the ECC words within the frame, that, in the Ultrascale technology are located in words 60, 61 and 62 if the assumption that the errors not injected belong to these words. Xilinx states that the ECC words for the Ultrascale family are located in words 61 and 122 in [47] and 60 and 61 in [19]. Hence, it is not clear that all the errors

not injected are ECC bits.

The other hypothesis is that they belong to distributed memory elements or FFs. However, the location in the bitstream of those elements is detailed in the logic location (.ll) file. After comparing the locations of both storage elements and essential bits file there was no overlapping. Therefore, the most suitable hypothesis is that those locations in words 60, 61 and 62 are actually ECC bits, since they are marked as essential bits in all frames, and, for some reason, Xilinx decided not to disclose its location.

Finally, ECC bits are correctly marked as essential in all frames in spite of the logic of the circuit not being changed if any of them is affected by a SEU for the following reason: if an error happens in the ECC bits, and the user relies on them to perform scrubbing, or is using the SEM IP for that purpose, it will appear disguised as a single or multiple bit error inside a frame, in the words and bits that, if flipped, would change the ECC to that value. Consequently, the user should combine the CRC error detection with the ECC correction and detect this case if, after correcting several errors detected as an ECC error, the CRC error remains. For this work, ECC errors were not counted as critical, since no scrubbing is implemented in the demonstrator and it is user-dependent. However, they should be accounted for if scrubbing is used. The amount of ECC bits marked was of 11435 out of 719108 (1.5%) for the whole IP.

5.8.2 Uncorrectable errors

Sometimes during the injection, the SEM IP would report multiple errors in a location that did not correspond to where the injection had been performed. Besides, it would get stuck in this undesired state, repeatedly attempting to correct errors in different locations and automatically going back to the idle state. These events are reported in [19] as uncorrectable errors, occurring in around 0.3% of the errors injected. However, no further explanation is given for its cause.

The fact that the errors appear to the SEM IP as errors in completely different frames indicates that, most likely, they affected some logic external to the SEM IP, but necessary for its correct behavior, such as a global routing or interconnection. When observing the location of the injections that cause these double errors, all of them happened in the same frames, even when injecting in completely different runs, and even in different bitstreams—for both the IP with and without separation by layers. The linear addresses for the frames that would cause these errors are: 78, 102, 103, 258, 259, 332, 333, 570, 571, 1270, 1271, 1352, 1353, 1508, 1509, 1820 and 1821. Those linear addresses correspond to the physical addresses 0x134, 0x206, 0x207, 0x506, 0x507, 0x686, 0x687, 0xb06, 0xb07, 0x1886, 0x1887, 0x1a06, 0x1d06, 0x1d07, 0x2306 and 0x2307.

As it can be seen, frame addresses present certain regularity, always grouped in pairs. When it comes to the particular bit and word, it appears that this event could happen in any word within those frames, but always in bits 2, 3, 7, 21, 22 or 25.

No further indication on the cause for these errors was found. As indicated in [19], after one of these events, the FPGA was re-programmed, and the injection continued normally. It is of interest to show the amount of times this event was present during the injection,

being of 726 during the injection of the whole IP, which accounts for 0.1% of the injections. The proportion is maintained for all injections performed, and this figure lies in the same order of magnitude as indicated by Xilinx.

Depending on the real cause and mission, these bits should be considered critical or not. It is important to remark that none of these errors caused errors to the SpaceFibre IP logic, and only the SEM IP was affected by them. Then, if the mission includes the usage of SEM IP or scrubbing through the configuration engine, it is most likely that an SEU in these addresses would cause the protection mechanism to fail, hence being classified as critical bits with the maximum severity. Otherwise, if no scrubbing or reconfiguration are used, and these events did not cause failures in other parts of the system, then they should not be taken into account as vulnerable configuration bits.

5.8.3 Two errors caused by a single injection

During the injection process, a particular event was observed in which a single error injection, when attempting to correct it, would appear to the SEM IP as two errors in different configuration words. This behavior was already observed in [40] and attributed to the possibility of a slice register not being masked, since it would not appear listed as a memory element. When this unmasked memory element is modified, the change in configuration is captured by the SEM IP as a configuration error. However, this explanation seems unlikely, since it would mean that there is a bug in the masking capability of the SEM IP, and this kind of event should arise—notified as an error—whenever the value of that register is changed by the user logic.

This event was also recorded as many as 215 times happening during the injection for the whole IP. As it was the case for the uncorrectable errors, they all happened in frames 102, 103, 258, 259, 332, 333, 570 and 571, again with no correlation between the words it would occur on, but also affecting most often the same bits mentioned before.

The most likely explanation for this event is that it has the same nature as the uncorrectable errors, and a SEFI-like event in the SEM IP or configuration engine provoked by an injection to a global clocking route causes them. Nonetheless, unlike uncorrectable errors, these two errors can be corrected by SEM IP because they occur in different configuration words. There are two mechanisms that could be used for testing the hypothesis: either reverse-engineering the bitstream of the UltraScale architecture and figuring out the elements that are related to the frame addresses in which these event occur, or performing the injection and correction following another method that does not involve partial reconfiguration or the SEM IP to discard them as the cause of the failure. Both approaches are outside the scope of this work.

5.9 Reliability of the SpaceFibre IP

Once data has been collected, and a figure for the DVF obtained, it can be used to calculate the overall reliability that is expected from the IP. As stated in [25], the probability of

failure in a particular design due to SEU can be written as follows:

$$P(f) = P(f_{configuration}) + P(f_{BRAM}) + P(f_{FFs}) + P(f_{SEFI}) \tag{5.9}$$

With the results from this work, only the first term can be obtained. Data from memories is easily protected by using ECC codes at a relatively low cost [19]. As for the probability of a SEFI error happening, it is mostly architecture-dependent, and no data for its likelihood was found. However, for most designs, configuration memory is the biggest contributor to functional failures when affected by an SEU [19], and, for the case of SpaceFibre, this is especially true. As the injection shows, most data "soft" errors can be corrected, without user intervention, with the QoS and recovery mechanisms integrated in the IP. Hence, only with applying TMR to the reception buffers (above the retry layer) and critical registers as well as using safe state machines, the probability of failure can be approximated by the probability of a failure caused by an error in the configuration memory.

As shown in [32], the upset occurences for both LEO and GEO for the KU060 FPGA are:

- LEO: 2.4e-07 upsets/bit/day
- GEO: 1.8e-08 upsets/bit/day

Using these data, the expected failure rate (μ) for the SpaceFibre IP is:

$$\textit{Failure occurence} = \textit{Upset occurence} \cdot \textit{Number of essential bits} \cdot DVF \tag{5.10}$$

And, thus, the failure rates for both orbits are:

- LEO: $\mu = 2.4e-07 \cdot 719108 \cdot 0.1213 = 0.0209$ failures/day
- GEO: $\mu = 1.8e-08 \cdot 719108 \cdot 0.1213 = 0.00157$ failures/day

From that, the Mean Time Between Failures (MTBF) can be obtained:

- LEO: $MTBF = 1/\mu = 1/0.0209 =$ **47.77** days
- GEO: $MTBF = 1/\mu = 1/0.00157 =$ **636.9** days

Therefore, it is expected that after around 48 days of mission, an error in configuration memory would affect the behavior of the circuit. If no corrective measure is implemented, it is highly likely that this failure will persist in time, interrupting the communication through SpaceFibre.

The classification of failures according to their severity allows to estimate the MTBF for the case of some preventive or corrective measures being implemented. It is clear that in a mission using COTS FPGA, a scrubber (e.g., SEM IP) should be used. The reduced failure rate allows to assume that, in practice, all SEU-induced errors in configuration memory can be corrected by the SEM IP. Taking into account that failures with a severity of 1, if not persistent (that is, if they are corrected shortly after their occurrence, as done during the injection), can be recovered and will not be presented in the application level, only errors with severity 2 or higher, as well as uncorrectable errors should be counted as critical errors. This approximation can be done because temporary errors with lower

severity will only delay the information reaching the application layer, but will not affect to its correctness. The new DVF can be calculated as:

$$DVF = \frac{NB_{critical^2} + NB_{uncorrectable}}{NB_{essential}} = \frac{23663 + 726}{719108} = 0.0339 \tag{5.11}$$

Then, the failure rates and MTBF would be:

- LEO: $\mu = 2.4e-07 \cdot 719108 \cdot 0.0339 = 0.0059$ failures/day. $MTBF = 1/0.0059 =$ **170.92** days
- GEO: $\mu = 1.8e-08 \cdot 719108 \cdot 0.0339 = 4.38e-04$ failures/day. $MTBF = 1/4.38e-04 =$ **2278.9** days

Those results refer to the failure rates if correctness is needed in the data transmitted through the channel. However, if the SpaceFibre link is used to transmit just raw data from an instrument, these failures are not critical, and can be addressed by further data processing layers. Besides, if scheduled resets of the IP are performed between data transactions, which may be feasible depending on the overall payload architecture, the MTBF figure would improve slightly, since failures with severity 3 would not contribute to the overall reliability. Therefore, a careful management of the lane reset and timeout from the software layers are recommended.

To improve the reliability further, enabling its usage to transmit critical control data in a longer mission, including a layer of light forward-error correction (FEC) code over the data transmitted, as done in [48], would allow to drastically reduce the probability of failure, discarding data errors. This measure would reduce the critical bits that cause severity 2 errors. Additionally, the usage of distributed TMR in, at least, the most vulnerable layers, such as the retry layer or virtual channel layer, would significantly improve the reliability of the circuit. If it is not possible to schedule resets of the IP regularly, also the implementation of distributed TMR is recommended for the interface layer, to prevent potential failures of severities 3 and 5.

Concerning the availability of the system, which is a metric that is more useful in a practical scenario in which recovery measures are implemented, drawing a figure is out of the scope of this work since it depends on the higher level architecture of the payload and data acquisition strategy. In [13], a scheme to obtain the availability of an FPGA implementation is presented, and could be used to calculate the availability for a particular payload architecture and mission. However, it is clear that with a scrubber and periodic reset that allows to completely recover the state of the IP in a short lapse, the figure would be really close to a 100%, hence making its usage suitable for transmitting large amounts of data.

Chapter 6

Conclusion

In this work, the firmware for the SpaceFibre in-orbit demonstrator was developed. The demonstrator integrates a COTS SRAM FPGA, on a board with all the necessary peripherals to carry out the testing of the SpaceFibre Port IP as well as the COTS FPGA itself when launched into LEO. The developed firmware enabled the successful integration and control of all elements within the board necessary to carry out the tests in orbit. At the time of writing this report, it is has already been integrated in the 3U cubesat that will host the experiment, to be launched later this year.

This mission is relevant to provide insights of both SpaceFibre and COTS FPGAs, which are expected to become the de-facto standard for onboard high-data rate communications and for missions requiring great processing power respectively. Apart from the relevance of the mission, this work allowed Thales Alenia Space to gather practical know-how and guidelines on how to integrate and work with SpaceFibre.

Apart from the firmware, an injection campaign methodology for testing the reliability of any implementation including a processor in a Xilinx FPGA under the occurrence of configuration errors caused by SEUs was developed. The injection was performed in the KU060 FPGA, expected to become the next standard in space FPGAs for high-throughput applications. After performing a detailed literature research, the methodology was designed following the state-of-the-art techniques and applied to the SpaceFibre IP, taking into account the particularities of testing an IP that implements a communications protocol. Results were extracted and processed, obtaining the reliability of the IP as well as extracting information about its vulnerabilities. Since SpaceFibre is a technology that will be repeatedly used for many missions to come, it is of great relevance to draw figures of its reliability and improve the SpaceFibre IP to increase its tolerance to SEUs, enhancing its usage in COTS FPGAs. There are several important advantages of the methodology followed that prove really useful when applied to the SpaceFibre IP, and that show their value when applied to other modules as well:

- The usage of internal signals and parameters to monitor them from deviations on their nominal values offers a more complete and accurate figure on how vulnerable to SEU the design is. If only the correctness of the outputs of the circuit was observed, as done for other injection campaigns in the literature, a great deal of potentially harmful failures would have been missed, and only a reduction in effective bandwidth (if monitored) would have been observed.
- Performing separate injections to the different layers allows to gather more focused

and detailed information about the cause of the different vulnerabilities and propose mitigation techniques for each case.

- Separating the errors according to their severity divides them depending on the necessary techniques that shall be used for their mitigation, and offers more information on the potential of such mitigation.

- Obtaining the design vulnerability factor allows to draw direct reliability figures if the expected SEU incidence is known. This is extremely useful when combined with data from the architectural vulnerability factor (vulnerability of a particular device to radiation), since the reliability of the design for any mission can be obtained from this data.

These advantages allow to draw several conclusions:

- When used in a non-hardened FPGA, SpaceFibre requires additional measurements to ensure its reliability. As shown in section 5.9, protecting the SpaceFibre IP Port with TMR'd memories and registers, as well as the usage of Xilinx SEM IP is enough to obtain a good MTBF, that guarantees a good availability figure. However, if the SpaceFibre link is used in a critical or longer mission, the methodology allows to enhance the protection of the different layers in a more efficient manner.

- Analyzing the failure modes for different layers, the recommendations are as follows:

  - Lane layer: the critical failures observed in the lane layer are related mostly to the initialization FSM. Hence, protecting the FSM logic would be enough to drastically increase its reliability. Data failures caused by this lane can be temporarily addressed by the retry layer until the failure is corrected by scrubbing.
  - Retry layer: as a layer which is highly vulnerable to configuration memory errors, but small in size, it should be protected with distributed TMR in its totality.
  - Virtual channel layer: since this layer is above the retry layer, data errors will not be corrected if caused by a configuration error in this layer. A good measure of protection would be provided by using FEC codification as an "outer-code" in the transmission, and also by protecting the logic and registers in charge of implementing flow-control.
  - Interface layer: with the presence of scrubbing, it is not a critical layer, since the upper layers will address the errors caused by loss of synchronization. Nevertheless, it shall be protected by distributed TMR if the SpaceFibre operation cannot be interrupted by resetting the IP regularly.
  - Broadcast layer: it is the least vulnerable layer. Hence, no additional protection needs to be implemented for a short mission.

Therefore, the presented modifications shall be added to the port in missions in which the correctness of the data transmitted through SpaceFibre is critical. That would enable the integration of such a powerful technology in missions following the "new-space" paradigm, with cheaper components and smaller qualification effort but great capacities, leading to a more competitive position in the market.

Coming back to the research questions posed in the introduction, as well as the objectives set for the project, the answers are as follows:

- The necessary steps for the integration of the SpaceFibre technology in a LEON3 SoC have been presented in chapter 3.
- SpaceFibre is ready for its use in a COTS FPGA if it is combined with scrubbing and periodic scheduled resets of the IP. Besides, techniques to reduce the sensitivity to radiation have been proposed, optimizing the additional logic cost leveraging the designed methodology.

6.1 Contributions

The contribution to the topic of error injection done by this methodology is:

- To the best of the author's knowledge, it is the first error injection campaign published for the Ultrascale technology, providing guidelines to locate the design in configuration memory for error injection with the granularity of one configuration column for this new architecture.
- To the best of the author's knowledge, it is the first error injection campaign performed in a SpaceFibre port, allowing to gather knowledge about the sensitivity and vulnerabilities of this future technology. Besides, it presents a methodology to perform error injection that can be utilized as a verification step, since the stand-alone approach followed minimizes the amount of additional logic. It is important to remark that, for verification purposes, a circuit cannot be altered, and all additional logic introduced must stay afterwards. Hence, this methodology is applicable for all blocks that are part of a SoC.
- Apart from [40], it is the only work that uses the automatic correction mode of the SEM IP to correct errors, allowing to monitor SEFIs in the configuration engine of the FPGA that would produce wrong results if it is not re-programmed.
- To the best of the author's knowledge, it is the first error injection campaign to present a finer-grain analysis, not only monitoring functional failures, but including other metrics and potentially harmful side-effects of deviation from nominal values in status parameters.
- Apart from [39], it is the only work to monitor signals other than the outputs, and to use a non-benchmark complex circuit that does not just perform a bounded operation, but runs continuously while the injection is performed.

  It is relevant to highlight that, apart from [43] and [39], this is the only one of the analyzed works that performs an error injection campaign in a device that is not an educational example or benchmark.

6.2 Future work

Finally, there are two key aspects that were out of the scope of this work, but would be interesting to address in future research to overcome its limitations:

- Including registers and BRAM memories in the injection strategy: using some techniques mentioned in the literature, it should be possible to include these structures in the injection analysis. BRAM memories are usually protected by EDAC or TMR, and the effects of an SEU can be mostly reduced to data errors. However, it would be interesting to take registers into account, especially those that are used in the control logic, to obtain a more accurate representation of the global reliability. It is important to remark that the reliability figures presented in this work represent only the failures produced by SEUs in the configuration memory. Hence,

- Implementing the proposed protection recommendations: including the measurements shown to improve the SEU tolerance and applying the methodology again to re-assess the improvement in the DVF would validate the methodology and improve the SpaceFibre IP Port.

- Expanding the analysis to a system-level scenario, including a more complex architecture and use-case able to obtain availability figures.

Bibliography

[1] C. G. AB, "GRLIB IP Library User's Manual. Nov 2019, Version 2019.4." [Online]. Retrieved 23/01/20.

[2] D. White, "Considerations surrounding single event effects in fpgas, asics, and processors,"

[3] E. C. for Space Standarization, "ECSS-E-ST-50-11C – SpaceFibre – Very high-speed serial link (15 May 2019)." [Online]. Retrieved 10/02/20.

[4] D. S. Lee, G. R. Allen, G. Swift, M. Cannon, M. Wirthlin, J. S. George, R. Koga, and K. Huey, "Single-event characterization of the 20 nm xilinx kintex ultrascale field-programmable gate array under heavy ion irradiation," in *2015 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–6, IEEE, 2015.

[5] A. Håkansson, "Portal of research methods and methodologies for research projects and degree projects," in *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*, pp. 67–73, CSREA Press USA, 2013.

[6] C. G. AB, "LEON3 Multiprocessing CPU Core Product Sheet." [Online]. Retrieved 23/01/20.

[7] C. G. AB, "GRLIB IP Core User's Manual. Nov 2019, Version 2019.4." [Online]. Retrieved 23/01/20.

[8] C. G. AB, "GRMON2 User's Manual. Apr 2018, Version 2.0.93." [Online]. Retrieved 23/01/20.

[9] C. G. AB, "Bare-C Cross-Compiler User's Manual. Jan 2020, Version 2.1.1." [Online]. Retrieved 23/01/20.

[10] C. G. AB, "MKPROM2 User's Manual. Jul 2018, Version 2.0.65." [Online]. Retrieved 23/01/20.

[11] Xilinx, "Radiation-tolerant Kintex Ultrascale XQRKU060 FPGA Data Sheet (DS882)." May 2020 [Online]. Retrieved 29/05/20.

[12] J. Wang, "Radiation effects in fpgas," *CERN*, 2003.

[13] F. Siegle, *Fault detection, isolation and recovery schemes for spaceborne reconfigurable FPGA-based systems*. PhD thesis, University of Leicester, 2016.

[14] Xilinx, "Ultrascale Architecture Configurable Logic Block User Guide. February 2017, Version 1.5." [Online]. Retrieved 12/04/20.

[15] R. Roosta, "A comparison of radiation-hard and radiation-tolerant fpgas for space applications," *NASA Electronic Parts and Packaging (NEPP) Program JPL D-31228*, 2004.

[16] Xilinx, "Ultrascale Architecture Configuration User Guide. March 2020, Version 1.12." [Online]. Retrieved 11/04/20.

[17] Xilinx, "Ultrascale Architecture Clocking Resources User Guide. October 2019, Version 1.9." [Online]. Retrieved 20/05/20.

[18] A. M. Keller and M. J. Wirthlin, "Benefits of complementary seu mitigation for the leon3 soft processor on sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 519–528, 2017.

[19] Xilinx, "PG187 - Ultrascale Architecture Soft Error Mitigation Controller v3.1 Logicore IP Product Guide." [Online]. Retrieved 12/02/20.

[20] K. LaBel, "Radiation Effects on Electronics 101: Simple Concepts and New Challenges." [Online]. Retrieved 25/05/20.

[21] M. Mousavi, H. R. Pourshaghaghi, M. Tahghighi, R. Jordans, and H. Corporaal, "A generic methodology to compute design sensitivity to seu in sram-based fpga," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, pp. 221–228, IEEE, 2018.

[22] J. C. Fabero, H. Mecha, F. J. Franco, J. A. Clemente, G. Korkian, S. Rey, B. Cheymol, M. Baylac, G. Hubert, and R. Velazco, "Single event upsets under 14-mev neutrons in a 28-nm sram-based fpga in static mode," *IEEE Transactions on Nuclear Science*, 2020.

[23] I. Villalta, U. Bidarte, J. Gómez-Cornejo, J. Jiménez, and J. Lázaro, "Seu emulation in industrial socs combining microprocessor and fpga," *Reliability Engineering & System Safety*, vol. 170, pp. 53–63, 2018.

[24] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, "A novel seu, mbu and she handling strategy for xilinx virtex-4 fpgas," in *2009 International Conference on Field Programmable Logic and Applications*, pp. 569–573, IEEE.

[25] M. Berg, K. LaBel, and M. Campola, "FPGA Assurance: from Radiation Susceptibility through Trust and Security." NASA Electronics Parts and Packaging (NEPP) Electronics Technology Workshop (ETW), June 2018 [Online]. Retrieved 29/05/20.

[26] D. M. Hiemstra, V. Kirischian, and J. Brelski, "Single event upset characterization of the kintex ultrascale field programmable gate array using proton irradiation," in *2016 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–5, IEEE, 2016.

[27] D. S. Lee, G. Swift, and M. Wirthlin, "An analysis of high-current events observed on xilinx 7-series and ultrascale field-programmable gate arrays," in *2016 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–5, IEEE, 2016.

[28] D. M. Hiemstra and V. Kirischian, "Part ii: Single event upset characterization of the kintex ultrascale field programmable gate array using proton irradiation," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–4, IEEE, 2018.

[29] P. Maillard, M. Hart, J. Barton, P. Jain, and J. Karp, "Neutron, 64 mev proton, thermal neutron and alpha single-event upset characterization of xilinx 20nm ultrascale kintex fpga," in *2015 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–5, IEEE, 2015.

[30] M. Berg, K. LaBel, M. Campola, and J. Pellish, "NASA Electronic Parts and Packaging (NEPP) Program - Update of Single Event Upset FPGA Testing." NASA Electronics Parts and Packaging (NEPP) Electronics Technology Workshop (ETW), June 2017 [Online]. Retrieved 29/05/20.

[31] P. Maillard, M. Hart, J. Barton, P. Chang, M. Welter, R. Le, R. Ismail, and E. Crabill, "Single-event upsets characterization & evaluation of xilinx ultrascale^{TM} soft error mitigation (sem ip) tool," in *2016 IEEE Radiation Effects Data Workshop (REDW)*, pp. 1–4, IEEE, 2016.

[32] Xilinx, "Rt kintex ultrascale fpgas for ultra high throughput and high bandwidth applications," 2020.

[33] M. Wirthlin, D. Lee, G. Swift, and H. Quinn, "A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and secded-protected arrays," *IEEE transactions on nuclear science*, vol. 61, no. 6, pp. 3080–3087, 2014.

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

[35] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Ongil, M. Sonza Reorda, and G. Squillero, "An industrial environment for high-level fault-tolerant structures insertion and validation," vol. 0, pp. 229– 236, 02 2002.

[36] D. de Andrés, J. C. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of vlsi systems," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 4, pp. 422–431, 2008.

[37] F. Serrano, J. A. Clemente, and H. Mecha, "A methodology to emulate single event upsets in flip-flops using fpgas through partial reconfiguration and instrumentation," *IEEE Transactions on Nuclear Science*, vol. 62, no. 4, pp. 1617–1624, 2015.

[38] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 965–970, 2007.

[39] A. M. Keller and M. J. Wirthlin, "Benefits of complementary seu mitigation for the leon3 soft processor on sram-based fpgas," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 519–528, 2016.

[40] S. F. Baig, "Reliability estimation and memory-efficient error mitigation schemes for a self-healing architecture," 2019.

[41] R. Zhang, L. Xiao, J. Li, X. Cao, and C. Qi, "A fault injection platform supporting both seu and multiple seus for sram-based fpga," *IEEE Transactions on Device and Materials Reliability*, vol. 18, no. 4, pp. 599–605, 2018.

[42] A. M. Keller, T. A. Whiting, K. B. Sawyer, and M. J. Wirthlin, "Dynamic seu sensitivity of designs on two 28-nm sram-based fpga architectures," *IEEE Transactions on Nuclear Science*, vol. 65, no. 1, pp. 280–287, 2017.

[43] W. Mansour and R. Velazco, "Seu fault-injection in vhdl-based processors: a case study," *Journal of Electronic Testing*, vol. 29, no. 1, pp. 87–94, 2013.

[44] Digilent, "How to store your SDK project in SPI Flash." [Online]. Retrieved 29/01/20.

[45] C. Liechti, "Pyserial 3.4 - Python Serial Port Extension." [Online]. Retrieved 06/03/20.

[46] L. Sullivan, "Confidence Intervals." [Online]. Retrieved 06/06/20.

[47] Xilinx, "Soft Error Mitigation IP Guidance for testing with error injection." [Online]. Retrieved 23/03/20.

[48] R. Giordano, S. Perrella, D. Barbieri, and V. Izzo, "A radiation-tolerant, multi-gigabit serial link based on fpgas," *IEEE Transactions on Nuclear Science*, 2020.

TRITA-EECS-EX-2020:901