POLITECNICO DI TORINO

Master Degree Course in Mechatronic Engineering

Study and Development of an Isolation-based Approach for Mitigating Radiation-induced Errors in Reconfigurable Systems-on-Chip

Supervisor

Prof. Luca Sterpone

Candidate

Andrea Portaluri

Co-supervisor

Eng. Corrado De Sio

Dr. Sarah Azimi

Academic Year 2020/2021

Abstract

In the last years, Reconfigurable Systems-on-Chip and SRAM-based Field Programmable Gate Arrays have been widely adopted for mission-critical tasks in aerospace, avionics and automotive, mainly thanks to their flexibility and low costs. Although, one of their main drawbacks is the high susceptibility to radiations-charged particles, both in space and at sea level. Among the several mitigation techniques, Isolation Design Flow obtains promising results in terms of reliability improvement and required time to implement it by acting only at the floorplanning phase. However, when considering numerous blocks and complex systems, implementation of state-of-the-art Isolation Design Flow can lead to a difficult (or even impossible) placement stage.

The main objective pursed by this thesis is the analysis and development of domains-based Isolation Design Flow guidelines to ease the floorplanning phase in the case of hardening-by-replication systems, where the application of the state-of-the-art approach would be otherwise very complex and time-consuming. Thus, a fault injection platform has been built in order to perform experimental campaigns of validation on such a technique. These design rules have been, then, applied to a Triple Modular Redundant benchmark implemented on a Xilinx Zynq-7000 AP-SoC to quantify its effectiveness by means of fault injection campaigns. In addition

to this, fault injection campaigns have also been carried out on the state-of-the-art Isolation Design Flow to compare the results.

Acknowledgements

First, I really want to thank Prof. Luca Sterpone, who guided me through these months of work with patience, reliability and a very friendly attitude. A special thanks goes also to Corrado, Sarah and Ludovica, without whom I would have lost myself over and over again in this journey.

To my parents, Giovanna and Luigi, who allowed me to achieve this goal without a single regret or complain whatsoever. You have been, are and will always be my greatest inspiration.

To my sister Elisa, my grandparents Maria, Stefano, Vittorio and Elena and my whole family for being there when I need the most. I am such a lucky man for having you with me.

To Antonio, Davide, Domenico, Emanuele, Giuseppe (all of them), Greta, Marta, Mirco, Nicole, Luca (all of them), Lucia, Simone and all the others for all the moments we shared and for those we will.

To Chiara, my (girl)friend.

Contents

List of Figures

List of Tables

Abbreviation

ASIC Application Specific Integrated Circuits

AXI Advance Extendible Interface

BRAM Block Random Access Memory

CLB Configurable Logic Block

CORDIC Coordinate Rotational Digital Computer

CM Configuration Memory

DMA Direct Memory Access

DSP Digital Signal Processing

EB Essential Bit

FF Flip-Flop

FPGA Field Programmable Gate Array

IDF Isolation Design Flow

IP Intellectual Property

LUT Look-Up-Table

PIP Programmable-Interconnection-Points

PL Programmable Logic

PS Processing System

R-SoC Reconfigurable System-on-Chips

SEU Single-Event Upset

SRAM Static Random Access Memory

TMR Triple Modular Redundancy

Chapter 1

Introduction

Lately, Reconfigurable Systems-on-Chip (R-SoCs) have witnessed a rapid growth in the number of on-field applications, such as avionics, aerospace, and automotive. The flexibility of re-programmable field transistors and the computational power of a microprocessor on a single device allows the designer to meet very high and strict requirements with relatively low costs, energy, and time-to-market. As a matter of fact, such a technology is now a reasonable alternative to more expensive solutions, like ASICs, both in terms of speed and performance. Concerning the SRAM-based FPGAs, the reconfigurable feature comes from the availability of a configuration memory (CM), where a binary sequence of instructions, called bitstream, is downloaded and stored until a new reconfiguration of the device is loaded. The bitstream is generated by the vendor tools and it is unique for each different model of FPGA. When operating in harsh environments, such a sequence can be corrupted due to the impact of high energy particles that, interacting within the device and releasing their energy, can cause a change of the electric state of a node in the CM or in a logic element, arising the so-called Single Event Upset (SEU) effect [1][2].

An SEU may compromise the output until a new reconfiguration is loaded or in some cases the next power cycle, possibly leading to critical scenarios. Due to technology and voltage scaling, these concerns are no longer circumscribed only to the aerospace field but also at sea level due to secondary particles [3]. Therefore, detecting and mitigating the SEU is becoming one of the main challenges of electronic devices used in critical applications, both is space and ground level.

Through the years, several efficient SEU mitigation techniques for SRAM-based FPGAs have been proposed, such as scrubbing, partial reconfiguration and hardware redundancy [4 - 7]. Although all highly efficient, they often suffer from time and performances overhead or resources over-utilization.

Among these, the Isolation Design Flow (IDF) technique represents a promising alternative to such problems. The fundamental concept of IDF is the physical isolation of modules within the same chip, by means of unused rows and columns of resources and, doing so, preventing the propagation of errors among interconnected blocks. On the other hand, IDF is often followed by issues such as routing congestion and increasing floorplanning complexity when it comes to numerous modules placement (e.g., modular redundancy). Therefore, every so often, implementation of state-of-the-art IDF becomes highly challenging.

In order to prevent the occurrence of such problems, this thesis proposes an innovative sets of design rules to easily implement IDF even in the case of complex redundant systems. Then, analyses on the benefits of such guidelines have been performed, identifying the optimal isolating policy, on the mitigation of radiation-induced SEUs on SRAM-based FPGAs by means of fault injection campaigns. As a study case, a hardware implementation of the CORDIC (COordinate Rotational DIgital Computer) algorithm has been used. The results show that, by isolating a particular set of functions within the logic, an improvement of the reliability of the

redundant modules can be achieved in terms of error rate and occurrence of output data unavailability.

Thesis overview

This work is organized as follows: Chapter 2 gives an overview on the FPGA architecture and design flow, radiation-induced errors and mitigation techniques. Chapter 3 reviews previous related works on SRAM-based FPGAs isolation techniques. The so-called domains-based Isolation Design Flow is then described in Chapter 4. The experiment environment and results are presented in Chapter 5 and 6, respectively. Eventually, Chapter 7 contains conclusions and discussions on further works.

Chapter 2

Background

2.1 R-SoCs and FPGAs

R-SoCs are integrated circuits that exploit the capability of a Processor System (PS) and the flexibility of a Programmable Logic (PL) on a single chip, also known as the combination of a Field Programmable Gate Array (FPGA) and a processor. The FPGA is a matrix-structured integrated circuits with a large number of resources that can be (re)programmed by the designer in order to implement a very wide spectrum of digital circuits and logics. Such a device, together with the computational capability of a microprocessor, has been capable to make its way through mission-critical applications like in satellites and spacecrafts, where the ability of upgrading electronics systems, exploiting the on-line reconfiguration, can avoid permanent failures in the device. Initially used for prototyping, today FPGAs' increasing performances have been made them appetible for many other applications such as aerospace, automotive, military, medical and more. Although the FPGA configuration is strongly vendor dependent, some of the main traits are

common. In particular, this thesis focuses the family of Xilinx Zynq-7000 All Programmable-SoCs. However, the following discussions can be extended to the majority of the off-the-shelf reprogrammable heterogeneous devices without loss of generality.

2.1.1 R-SoC Overview

With reference to the Xilinx Zynq-7000 AP-SoC family, Figure 2.1 shows the block diagram of the device.

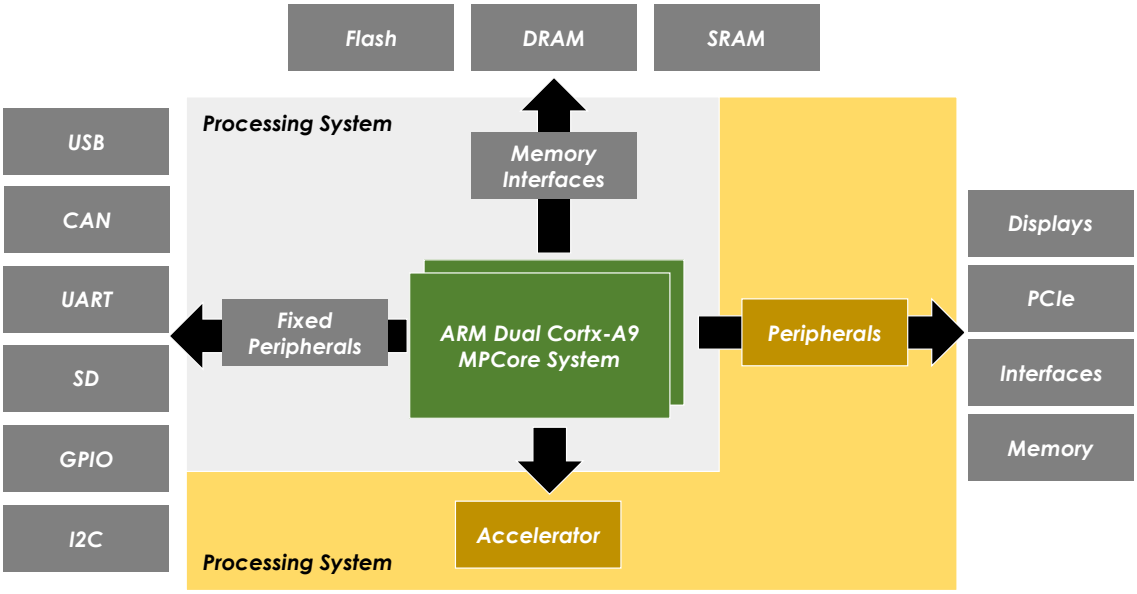

Fig. 2.1: Xilinx Zynq-7000 AP-SoC block scheme

As shown in the picture, it can be clearly observed the coexistence of a PS side together with a programmable logic. The PS includes a microprocessor, which can dialogue directly with the designer by means of I/O peripherals, such as USB, UART and GPIO LEDs and switches. The data transfer between PS and PL, instead, is implemented through AXI (Advance eXtendible Interface) protocol. This

allows the user to develop hardware accelerators in the PL while running software routines on the PS.

2.1.2 PL Architecture

The FPGA architecture is a matrix of tiles comprising of 3 main functional blocks:

- Logic Blocks
- Interconnection Blocks
- Input/Output Blocks

The Logic Blocks are the fundamental bricks for implement any logic function. They include Configurable Logic Block (CLBs, shown in Figure 2.2), Digital Signal Processing blocks (DSPs) and Block Random Access Memories (BRAMs). The CLBs, which are often the most used components within the implemented design, regroup several others sub-blocks such as Multiplexers (MUXs), Look-Up-Tables (LUTs) and Flip-Flops (FFs). The communication between blocks happens by means of the Interconnection Blocks, which consist in Programmable-Interconnection-Points (PIPs, shown in Figure 2.3) and hardwired networks. These can be programmed as well as the Logic Blocks in order to correctly route data. Eventually, Input/Output Blocks (IOB, shown in Figure 2.4) are the elements that allow to interface the PL with external devices and transferring data from/to them.

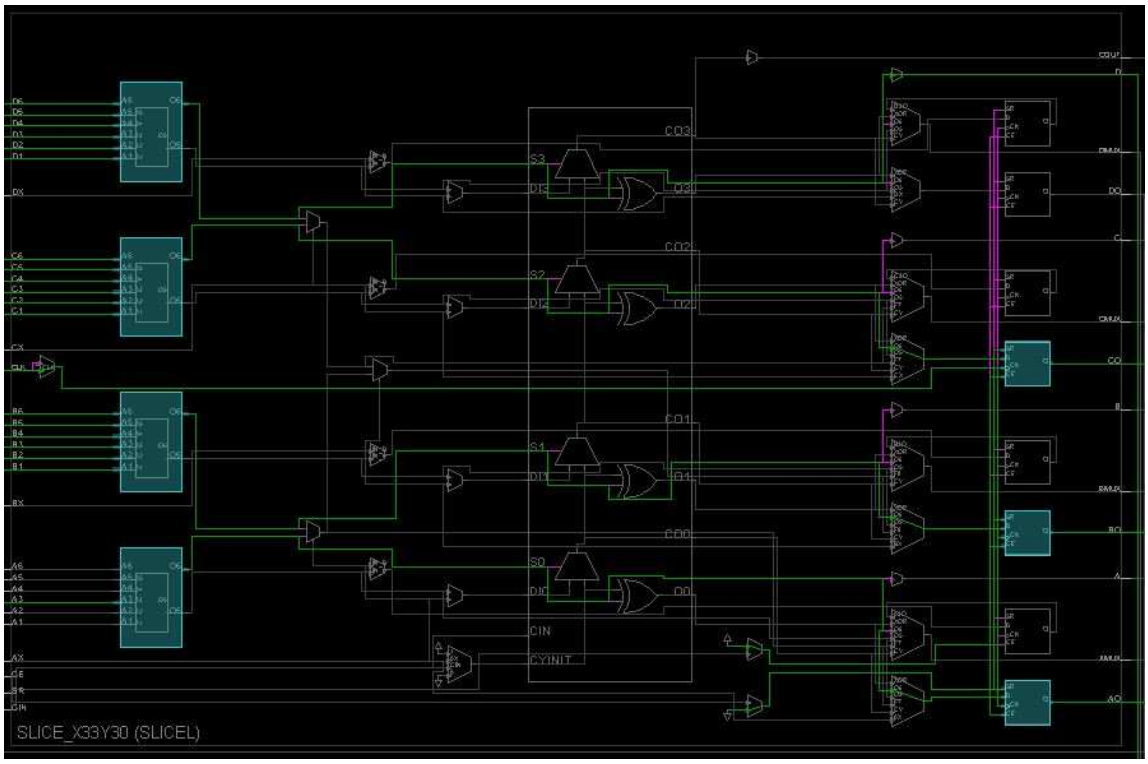

Fig. 2.2: Content of a programmed CLB

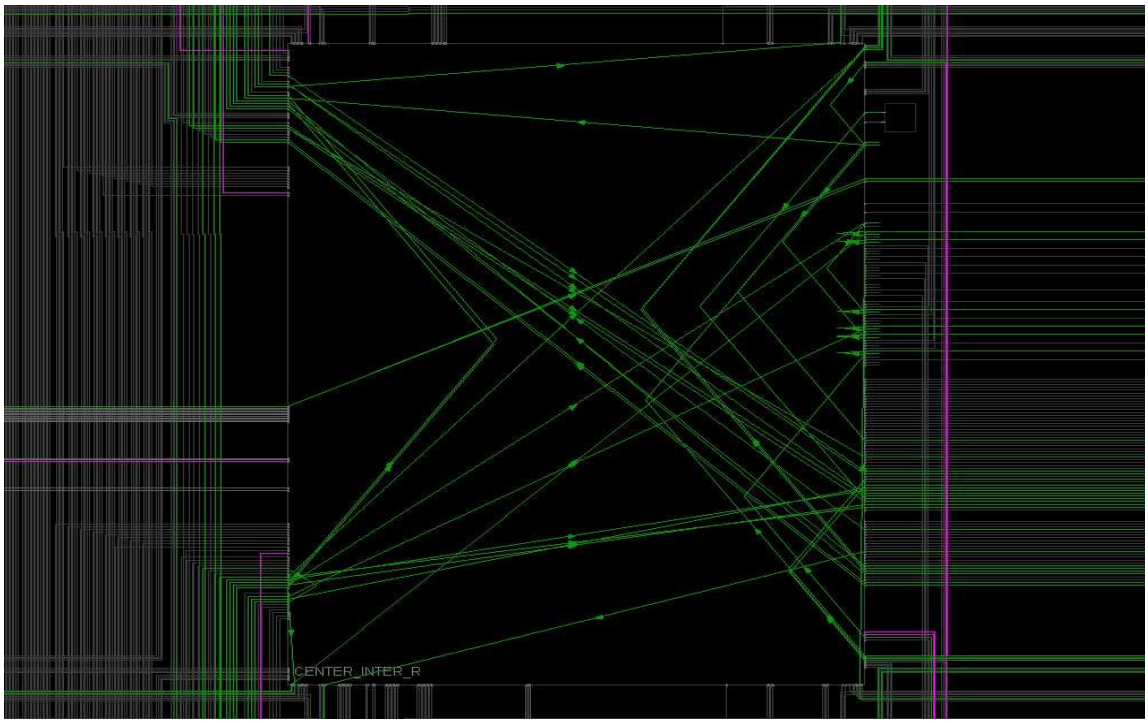

Fig. 2.3: PIP within a programmed interconnect matrix

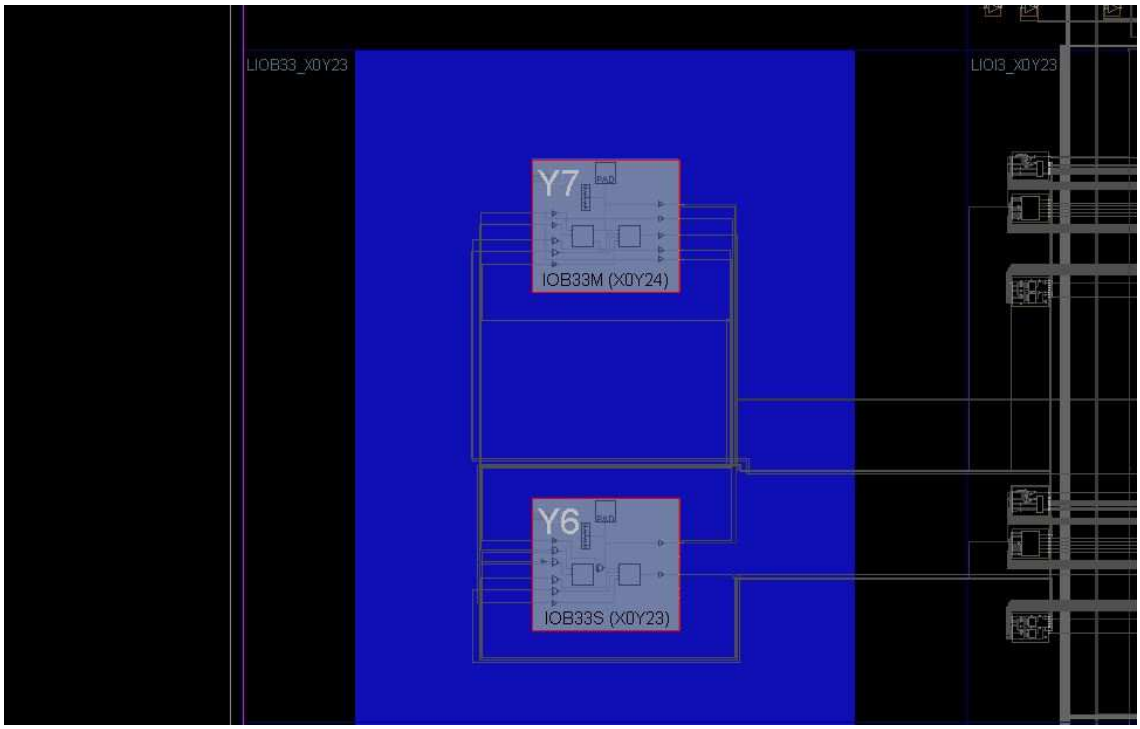

Fig. 2.4: Programmed IOB

All the discussed blocks are configured by means of a set of binary instructions, called bitstream, which is downloaded and stored in the CM of the device. As volatile memory, these data are rebooted whenever a new configuration or a power cycle occurs.

2.1.3 Bitstream and Design Flow

The bitstream is often developed automatically by the vendor tools (e.g., Vivado Design Suite). As a matter of fact, we know very few details about its structure [8]. However, it is possible to identify 3 main sections within it:

- Header
- Configuration data

- Tail

The header contains the information to initialize the configuration such as the mask for the configuration data, clock frequencies and the Cyclic Redundancy Check (CRC). The actual logic is described from the configuration data, where all the required logic elements are programmed according to the design. Eventually, the tail closes the configuration phase with other secondary steps.

The bitstream is generated after several steps, which are shown in Fig. 2.5.

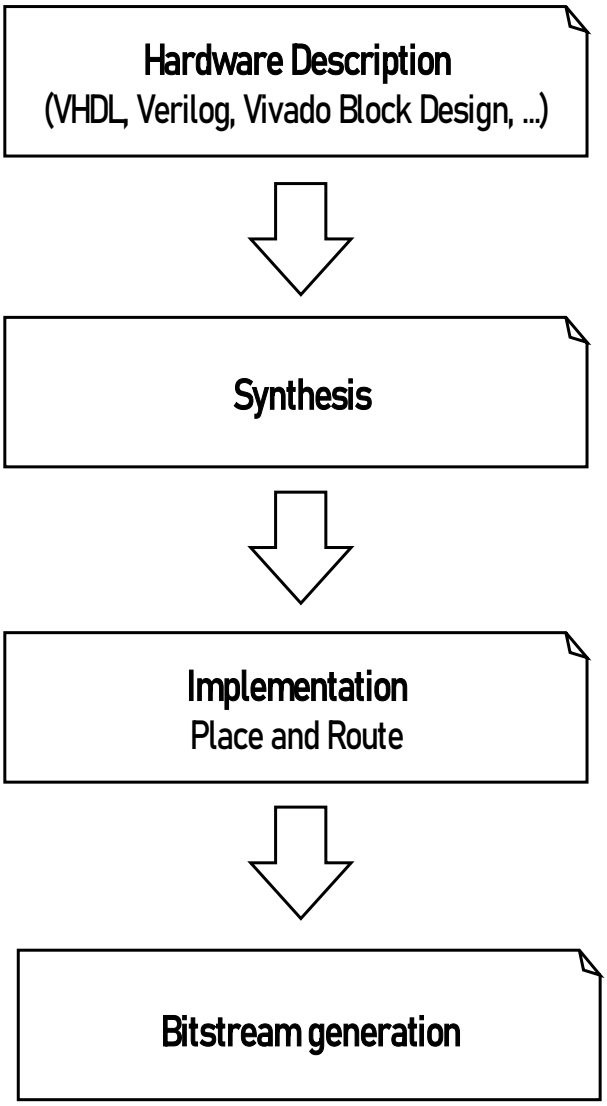

Fig. 2.5: FPGA Design Flow

First, a hardware-level description of the circuit to implement is necessary. This step defines the structure and the behaviour of the circuit. Languages such as VHDL and Verilog are the most used, thanks to their versatility and the wide availability of libraries. Moreover, some CAD design tools vendors (e.g., Xilinx Vivado Design Tool) have developed software add-ons in order to ease the hardware description phase. As instance, Vivado Block Design allows the designer to drag-and-drop already existing blocks, also called Intellectual Properties (IPs), and automatically interface them by means of wizard procedures. Although definitely more user-friendly than the standard HDLs, these approaches lack of flexibility when it comes to complex customized systems. The hardware description is then translated into logic blocks through the synthesis step. Most of the time for the FPGA design flow is spent in this phase and it is the one requiring the highest computational effort.

The mapping of the logic blocks as well as the routing definition takes place in the implementation step. Eventually, the bitstream is generated and downloaded into the device.

2.2 Radiation-induced Errors in FPGAs

As already said before, the main drawback of the FPGA application in mission-critical tasks is its susceptibility to high-energy particles, which are present both in space and at ground level. Due to their interaction with the Silicon surface of SRAM-based devices, unexpected electric reactions can arise within the FPGA, possibly leading to critical scenarios and this is more accentuated due to the FPGA extremely vast requirement of silicon to support reconfigurability. The number of

radiations in space is highly dependent on several factors such as location, altitude and solar events like solar flares and coronal mass ejection, which lead to different upset rates of the device.

Besides long-terms effects, the radiation-induced faults can also cause immediate effects within the device, giving rise to the so-called Single-Event Effects (SEEs). There exist several kinds of SEE such as Single-Event Latchup (SEL), Single-Event Functional Interrupt (SEFI), Single-Event Transient (SET) and Single-Event Upsets (SEU).

The SEL describes the modification of the current flow after the modification of the Silicon structure, which can cause permanent damage of the device itself if not quickly detected. A SEFI is the interruption of the functionality of the system, which requires a power cycle or hard reset to recover.

Among these, the SEUs are one of the most occurring radiation-induced errors in the FPGAs and, for this reason, also the focus of this work [9]. Regardless their origin, the exposure to such ionizing radiations generates electron-hole pairs within the oxide layer of MOS device, developing a disturbance on the threshold voltage and increasing the leaking currents. A disturbance voltage pulse is then generated (a SET) and, if it has a correct timing and amplitude, this evolves in a SEU. Such an event can be modeled as a bit-flip in the CM. If this bit-flip involves a programmed cell in the configuration of the netlist, it can modify the circuit and its functionality [10]. Figure 2.6 shows the dynamics of an SEU in a memory cell.

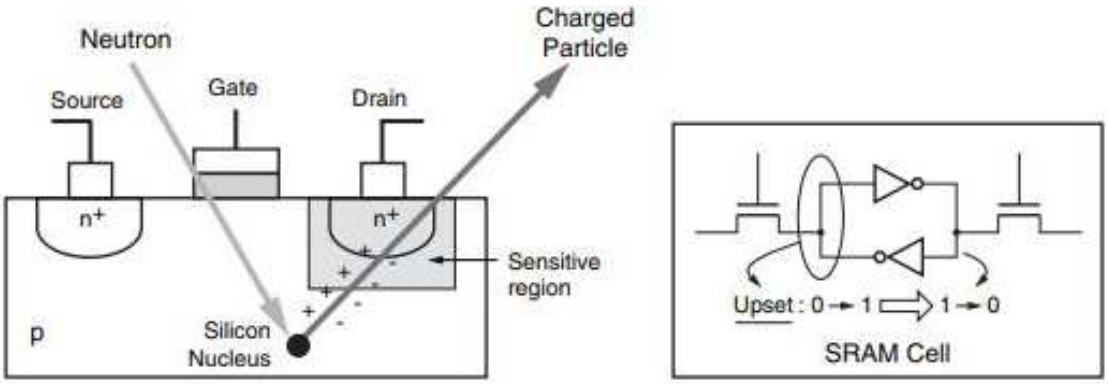

Fig. 2.6: SEUs dynamics in a memory cell

2.3 Methods of Fault Tolerance

Along with the increase in the use of FPGAs and R-SoCs, concerns about effects of radiation-induced faults and how to mitigate them have also become one of the biggest topics of research activities in the last years [11-]. A fault tolerant system is composed of two sub-systems: fault detection and fault recovery. Fault detection must achieve two purposes, which are informing the supervising process that actions must be taken for the system to remain operational in the case of faults and, of course, identify such errors and the defective components, so that a solution can be found. Detection of faults can be categorized in three different broad types:

- Redundant/concurrent error detection
- Off-line test methods/Built-In Self-Test
- Roving test methods

Redundancy is widely adopted as method of fault detection in FPGAs, particularly in the form of *Modular Redundancy* (MR). This method exploits a replication of the logic functions and, when an error occurs, a voter detects a disagreement among the multiple parts of the circuit. The simplest (and, often, most used) form is the

Triple Modular Redundancy (TMR) where each module is triplicated. Concerning the speed of detection, MR allows a very fast response, as it is capable of detecting the fault as soon as it manifests. On the other hand, MR provides no coverage of dormant faults or errors occurring in unused resources of the device. Moreover, the resource overhead is one of the largest among the mitigation techniques: as a matter of fact, the needed logic is ×2-3 more than the standard design. *Concurrent Error Detection* (CED) checks data flows and stores by means of error coding algorithms such as parity and its coverage can be traded-off with the utilized resources.

Off-line detection is another widely-used technique as a means of identifying manufacturing defects in the FPGA. Detection schemes that do not require any external test equipment are referred as *Built-In Self-Test* (BIST). Such circuits are implemented in FPGAs by different test configurations loaded separately to the operating configuration. These configurations comprise a Test Pattern Generator (TPG), the Output Data Evaluator (ODE) and the logic and interconnections to be tested. Thanks to its intrinsic characteristic, BIST methods have no impact on the FPGA during normal operation. As matter of fact, a dedicated test mode must be running to allow BIST to operate, either in start-up process or in response to an error detected by some other means. Thus, no faults detection can be performed when the FPGA is operating. Moreover, higher power consumption with respect to normal operation mode can be registered, together with higher area required. Recent publications achieved better test time and resources usage over the years, although issues like the necessity of testing BIST circuit itself still need to be worked out.

Roving detection exploits run-time reconfiguration to carry out BIST methods on the field. For this purpose, the FPGA is split into equal-sized regions, which are configured to perform self-test while the remain areas are operating the design

functions. These sections are then swapped over time so that the entire array can be tested while fully functional. This technique allows a better resource overhead with respect to the redundant methods although increasing the detection speed (order of a second). Performances are also impacted in two ways. Firstly, moving the test region within the FPGA brings a stretching of the interconnections, resulting in longer signal delays. Secondly, halts are required in order to switch the test regions.

2.3.1 Triple Modular Redundancy

One of the most used approaches for detection and mitigation of soft errors is the hardware modular redundancy. This technique exploits the physical replication of computational domains in order to allow a voting system among the outputs. In the case of Triple Modular Redundancy, the domains are triplicated. Several voting policies can be implemented such as classic algorithms, fuzzy algorithms and minimization algorithms. The simplest and, by far, the most on-field used algorithm is the majority voting system that is capable of mitigating one faulty behavior out of three by selecting the output data replicated by, at least, two domains. Figure 2.7 shows a graphic representation of the TMR with a majority voter. The three domains are fed with the same input data. Domain #0 is supposed to be faulty, hence returning a wrong output product. The majority voter is capable of detecting such a behavior and correct the final output. The major drawback of this technique is the high resource overhead with respect to the non-TMR design. As a matter of fact, almost $\times 3$ more resources are required to correctly implement TMR in a circuit. Concerning the faults coverage, errors that involves more than a single

domain cannot be mitigated, making the voting system unable to perform the correct task.

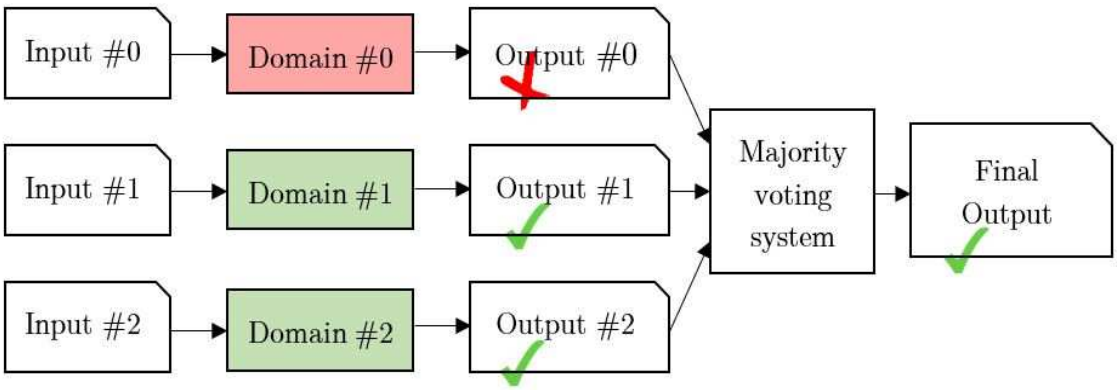

Fig. 2.7: Scheme of a TMR-hardened system

2.3.2 Isolation Design Flow

In addition to the common fault mitigation techniques, several precautions can be applied to harden even more the system reliability. As one of the major FPGA vendors, Xilinx proposed proprietary design guidelines: the *Isolation Design Flow* (IDF) [11]. The state-of-the-art IDF is a set of design rules that aims to physically isolate function within the same chip. Doing so, the designer will be able to avoid propagation of errors between interconnected modules. As a matter of fact, in the case of an occurring SEU, modules might encounter chain failures, compromising the correctness of a task. In order to guarantee the isolation between modules, each module to isolate must have its own hierarchical instance in the hardware description of the netlist (HDL) The isolation is reached through *fences*: rows or columns of unused resources of the device. Width constraints about fences varies according to the primitive type of the cells and the device. The communication

between isolated region is achieved by means of the so-called *trusted routes*. Routes must follow strict rules to be marked as trusted. In details, the routes have to connect one source and one destination only (point-to-point connection) and cross only tiles in the fence separating the two isolated regions that the route is connecting. Figure 2.8 shows a conceptual scheme of isolated regions, fences, and routes. *Route A* is not a trusted route since it passes out the fence region comprised between the two isolated regions that it is connecting. *Route B* is not trusted since it does not realize a point-to-point connection. Due to the need to respect constraints on fences and trusted routes, IDF requires an elaborated floorplanning phase that is only partially supported by the vendor tools. During the floorplanning, the communication between isolated modules can be implemented only between adjacently-placed modules. Moreover, the width of the fence follows some constraints (usually between 3 and 8 tiles). Thus, the manual floorplanning process requested for implementing isolation becomes highly challenging or, in the worst cases, impossible when the number of modules increases. To cope with the increase of complexity, it is possible to group modules under a higher hierarchical level. Therefore, a trade-off between the isolation modules and the complexity and feasibility of the design placement and routing should be considered.

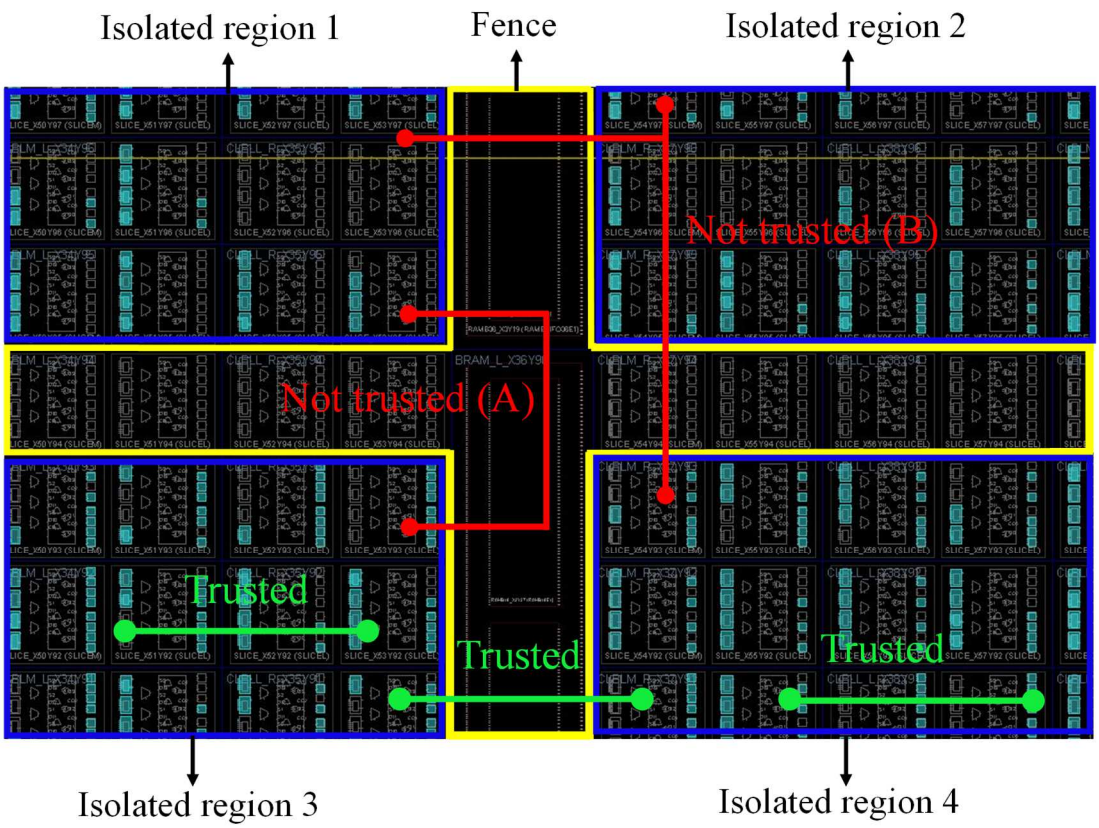

Fig. 2.8: Conceptual scheme of isolated regions with trusted route

Chapter 3

Related Works

This chapter provides an overview of previous work related to the topics this thesis discusses. In particular, works on IDF-based algorithms for faults mitigation, mitigation techniques and analysis on the radiation-induced effects on SRAM-based devices have been carefully evaluated.

So far, few research works have focused on IDF or IDF-based architectures [12-14]. The authors in [12] were the first to propose a technique to use IDF with a partial reconfiguration for Xilinx SRAM-based FPGAs, supporting online module relocation. The approach proposed in [13] suggests a novel method to ease the bitstream relocation in presence of IDF constraints. Eventually, the authors in [14] implement off-chip trusted communication with the partial reconfigured section. However, even though the mentioned methods are all highly effective, the FPGA commercial design tool (e.g., Xilinx Vivado) currently does not support any partial reconfiguration integrated with IDF. Therefore, the main challenges remain the need to interface with external tools and the elevated time needed for implementing the design. As far as the design is concerned, the authors in [15 - 17] lists the most

common design-for-reliability solutions such as hardware redundancy, error-correction coding and configuration scrubbing. Among these, TMR represents one of the most used, effective, and well-known approaches for SEU mitigation [18]. It underwent under several slight modifications over the years like different granularities of replicated modules or being implemented partially. The authors in [19] discuss a fine grain TMR architecture to deal with multiple faults in the architectures. In [20], the feasibility of a partial TMR is proved in order to allow its implementation when not possible otherwise. Effectiveness of TMR is dependent on several factors: in [21], the authors demonstrate how faults in the power supply can affect the replicated modules. In the same way, cross-clock signals have been proved to cause errors in TMR-hardened systems and, thus, needed to be counteracted. The work described in [22] proposes synchronizers to delete the effects of asynchronous sampling. Eventually, [23] discusses the sensitivity of SEU for different routing of TMR replicas of same circuits. However, as far as my knowledge can tell, no work has evaluated the effectiveness of TMR-hardened isolated circuit yet, which is one of the main foci of this thesis.

Several works have analysed the effects of radiations on reprogrammable devices: researches present methodologies to predict or simulate radiation-induced SETs on Flash-based FPGAs [24 – 27] and SRAM-based FPGAs [28 – 30], highlighting how the radiation-induced faults are a very tangible problem in nowadays on-field application.

Chapter 4

Developed Domains-based Isolation Design Flow

4.1 Introduction

This work aims to propose a set of design practices for reducing floorplanning complexity when a replication-based mitigation approach is used. These rules are intended to simplify the floorplanning phase during the isolation design flow, usually delegated to the designers, that quickly explodes in complexity when the state-of-the-art IDF is applied to replicated modules. Indeed, the very high complexity fails to meet the constraints required by IDF for topological reasons, forcing the designer to give up on isolation. The proposed solution reduces the number of blocks to be isolated, coupling together different modules within the same isolated region [31]. However, unaware relaxation of the isolation constraints and module aggregation can lead to nullifying the advantages introduced by IDF. For instance, when coarse-grained TMR is applied, the modules to be hardened are

replicated. The redundant modules are used for performing the same computation independently. Then, the results are compared to detect and correct possible errors through a voter circuit. The following sections describe in details the developed design flow.

4.2 Implementation steps

The steps for integrating domains-based IDF in the traditional FPGA design flow illustrated in Figure 4.1. It consists of the four tasks listed below:

- Pre-synthesis
- Post-synthesis
- Floorplanning
- Post-implementation

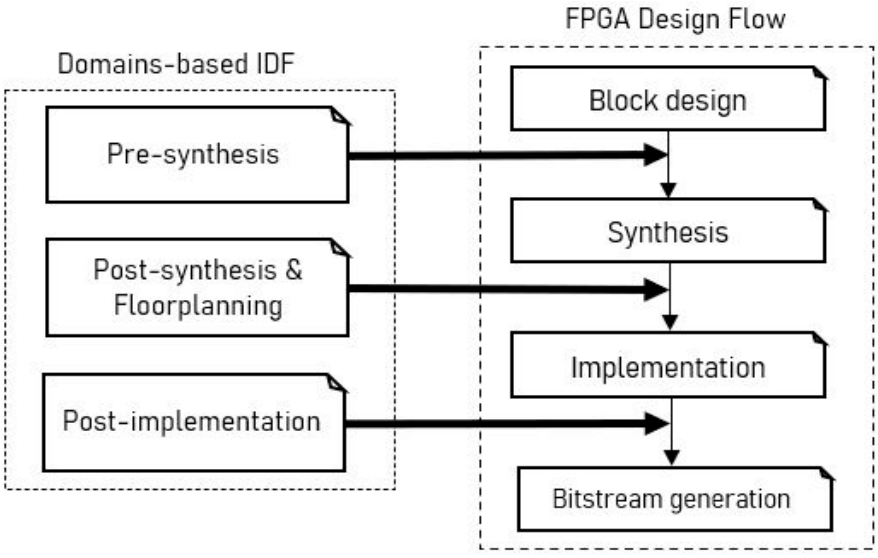

Fig. 4.1.: Domains-based IDF implementation steps

Each of these steps are intended to be performed within the Vivado Environment for Xilinx reprogrammable devices. However, they can be mapped on different CAD tools.

4.2.1 Pre-synthesis

Since errors that will not affect more than a single replicated computational unit are filtered by the voter, the underlying idea is to prioritize the isolation between modules that contribute to two different data domains of the voter. Differently, isolation between modules in the same voter domains (i.e., contributing to the same voter input) can be relaxed. The reduction of the number of the isolated regions will consequently reduce the floorplanning constraints and complexity. In this first phase, the isolated regions are defined. Differently from the state-of-the-art IDF, the modules to include together in the same isolated region must be regrouped in the design hierarchy. It is important to avoid grouping together modules belonging to different domains of the same voter, as explained above. Figure 4.2 shows two possible aggregation policies. The leftmost block scheme represents an inter-domains isolation while the rightmost, the correct one, an intra-domains isolation. Clock signals or other inter-region signals must be declared in the placement constraint file to be, allowed to cross isolated regions.

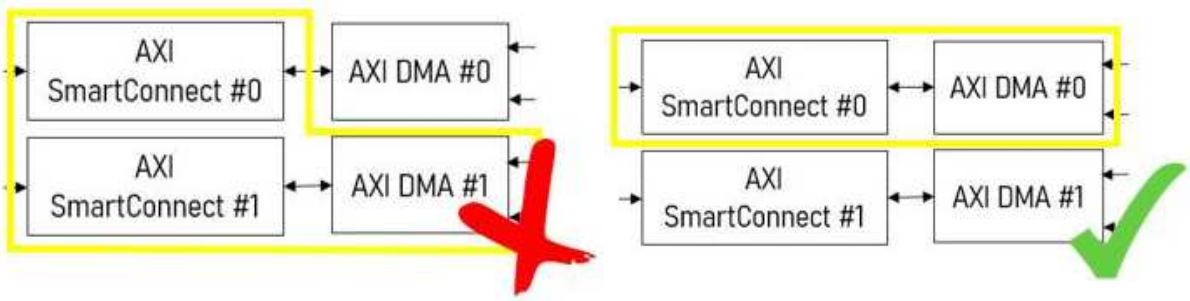

Fig. 4.2: Inter-domains vs. Intra-domains isolation policy

4.2.2 Post-synthesis

The modules previously identified to form an isolated region must be declared as isolated, in order to let the CAD tool know which modules are intended to be isolated. This property can vary according to the tool. As instance, using Vivado 2020 Design Suite, it is called HD_ISOLATED. Doing so, the communication among isolated blocks will be constrained only through the trusted routes. This process is automatically performed by the tool.

4.2.3 Floorplanning

The floorplanning phase is executed manually by instantiating placement blocks (pblocks) (Fig. 4.3). A pblock is a collection of physical resources (e.g., LUTs, PIPs) of the programmable hardware. If the isolation property is correctly checked, the routing and the logic cells of an isolated module will be placed only in the associated pblock. At this stage, the fencing rules must be accurately followed in order to achieve correct isolation. An estimated value of resources needed for the function within the pblock will be reported by the FPGA design tool in order to not run into resource overflow.

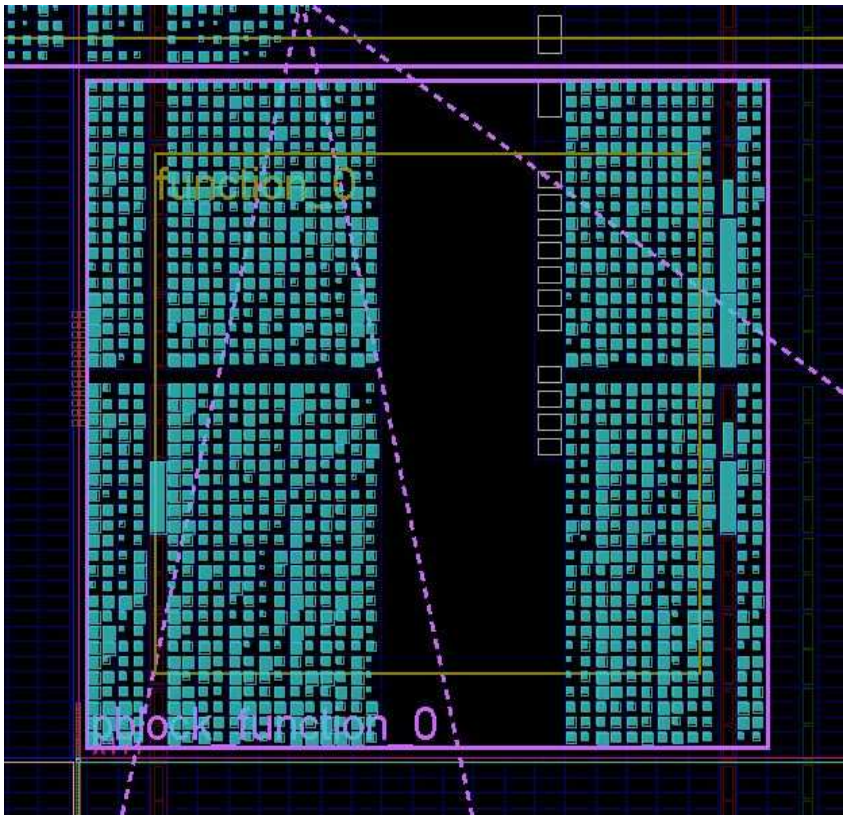

Fig. 4.3: Vivado Implemented View of a pblock

4.2.4 Post-implementation

After the implementation, the Vivado Isolation Verifier (VIV) [32] built-in tool is used to verify the correct implementation of the IDF rules between the isolated blocks. This tool generates a report on possible misplacements of blocks or fences and physical overlay of modules. In case of critical warnings or failed implementation, the user must go back to the Floorplanning phase and correct the highlighted mistakes in the isolation procedure.

Chapter 5

Experimental Environment

5.1 Introduction

For evaluating the benefits introduced by the plain and domain-based IDF, a fault injection environment has been developed. The environment automatizes both the faults generation and faults evaluation tasks, as well as results collection and analysis. For this work, the PyXEL framework has been extended to support Essential Bits, allowing to focus the analysis only on the sensitive bits of the configuration memory.

5.2 Fault Injection Platform

In order to evaluate and compare the reliability of the study cases, an experimental environment has been developed in order to perform fault injection campaigns. The environment runs on a host computer and communicates with the platform

implementing the circuit under test through serial communication. The fault injection mechanism relies on the PyXEL framework [33]. PyXEL, developed by L. Bozzoli *et al.* at Politecnico di Torino, is a Python library for the analysis of faults in FPGA, which allows modifying single or multiple bits in the bitstream to emulate faults. For this work, PyXEL has been extended to focus on a subset of the configuration memory, named Essential Bits (EB). The EBs are a subset of the programmable bits of the specific circuits that are reported by the vendor tool (i.e., Vivado) as bits that if corrupted may lead to errors in the circuit [34]. This classification allows to divide the configuration bits in subsets, as shown in Figure 5.1. The bits that will certainly produce an output error if corrupted are called Critical Bits (CBs) and are a subset of the EBs. They vary in number and coordinates according to the implemented circuit. The details on CBs are strictly encrypted by the vendor.

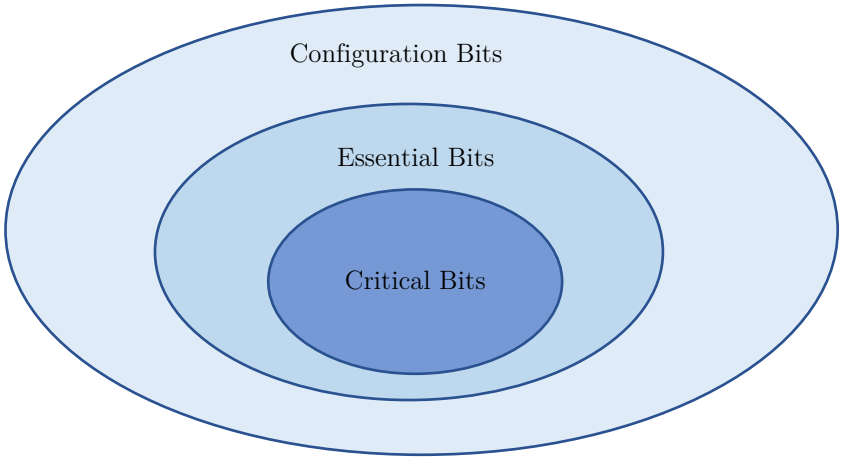

Fig. 5.1: Graphical representation of the Configuration Bits subsets

5.3 Methodology

The fault injection campaigns consist of a collection of single independent trials. For each trial, an essential bit of the circuit under test is corrupted and the effect introduced by the fault is evaluated. The generated faulty bitstream is used for programming the programmable hardware. Then, a software test routine is loaded in the processing system of the R-SoC. The software test routine stimulates the computing modules on the PL. Then, it sends the results to the fault injection platform where they are collected and analysed. All the steps are fully automatized. Figure 5.2 illustrates the steps and modules involved in the fault injection process, together with the implementation of the Domains-based IDF in the classical FPGA design flow.

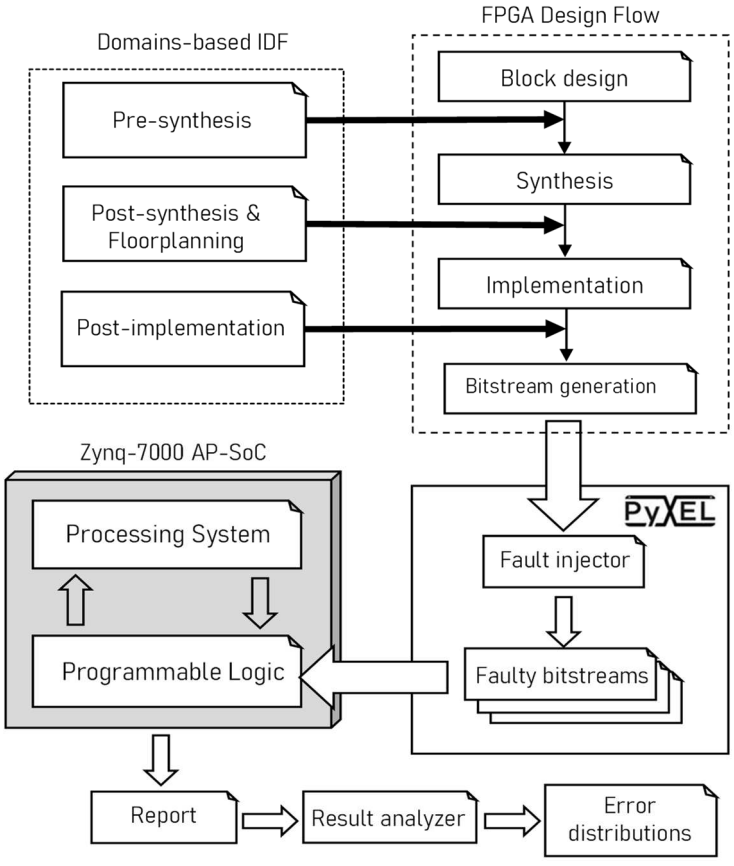

Fig. 5.2: Experimental flow and fault injection platform

Chapter 6

Experimental Results

6.1 Introduction

For evaluating the benefits introduced by traditional and domain-based IDFs, fault injection analyses have been carried out. Fault injection campaigns have been executed using the fault platform reported in section V. The Zynq-7000 AP-SoC has been used as the hardware platform. The evaluated benchmark application is the CORDIC IP provided by the Vivado IP Library. The analysed designs include both plain and TMR-hardened versions implemented with and without traditional and domains-based IDFs. SEU in configuration memory is the fault model emulated during fault injection tasks. The reliability analyses have been compared to quantitively measure benefits introduced by the traditional and domains-based isolation flows.

6.2 Benchmark Application

6.2.1 Programmable Logic

Concerning the benchmark, the CORDIC (COordinate Rotation DIgital Computer) algorithm is the hardware-accelerated core implemented on the programmable logic. Introduced in 1956 by J. Volder, CORDIC is a well-known approach to compute operations such as arithmetical and transcendental functions or coordinates conversions of given input vectors through hardware computations only (i.e., addition/subtraction and bit shift). Figure 6.1 shows a block scheme of the CORDIC algorithm. It has been first used as replacement for the analogic navigation computers in aerospace and within digital filters while today it is still widely implemented in VLSI technology. It has undergone several modifications through the years mainly focused on latency reduction and increasing throughput. Nowadays, CORDIC is often adopted as resource for DSP tasks, robotics and 3D graphics, mainly thanks to its speed and flexibility.

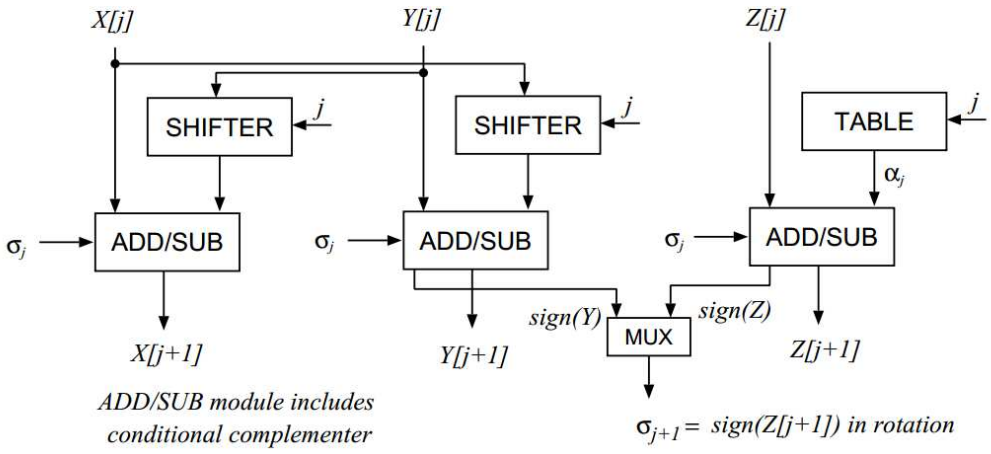

Fig. 6.1: CORDIC block scheme

In details, the CORDIC has been programmed to compute sine and cosine of given input vectors. The algorithm has been implemented on the chip by means of the Xilinx CORDIC IP Core, which is shown in Figure 6.2., and controlled by a software routine running on the processing system. The communication between software routine and hardware modules is implemented through AXI-4 Interconnection Cores. Data transfers are implemented using AXI DMA [35]. The plain benchmark is shown in Figure 6.3. When the software routine is triggered by the fault injection platform running on the host computer, it stimulates the cores on the PL and evaluates if they are working correctly. In detail, the software routine provides a test vector to the CORDIC IP Core, compares the results of the hardware computation with the expected ones and sends the experiment report to the results collector module running on the host computer. A hardened version of the benchmark circuit has been designed using TMR. The CORDIC Core and its communication interfaces have been replicated three times. Each replication can be accessed by the PS through the AXI Interconnect module. The software routine votes the final results based on the output obtained by the three replicas.

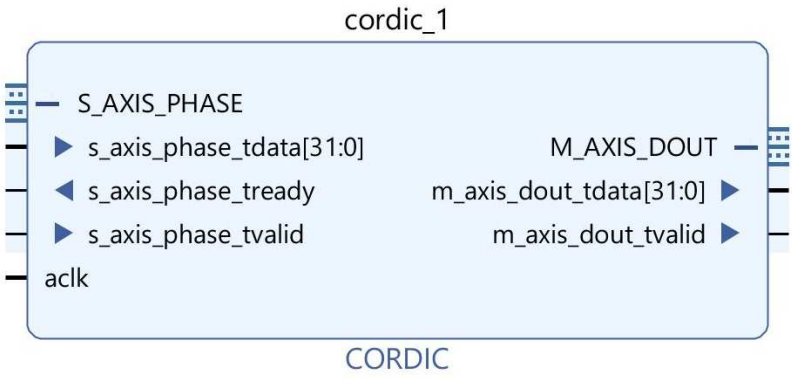

Fig. 6.2: CORDIC IP Core in Vivado Block Design view

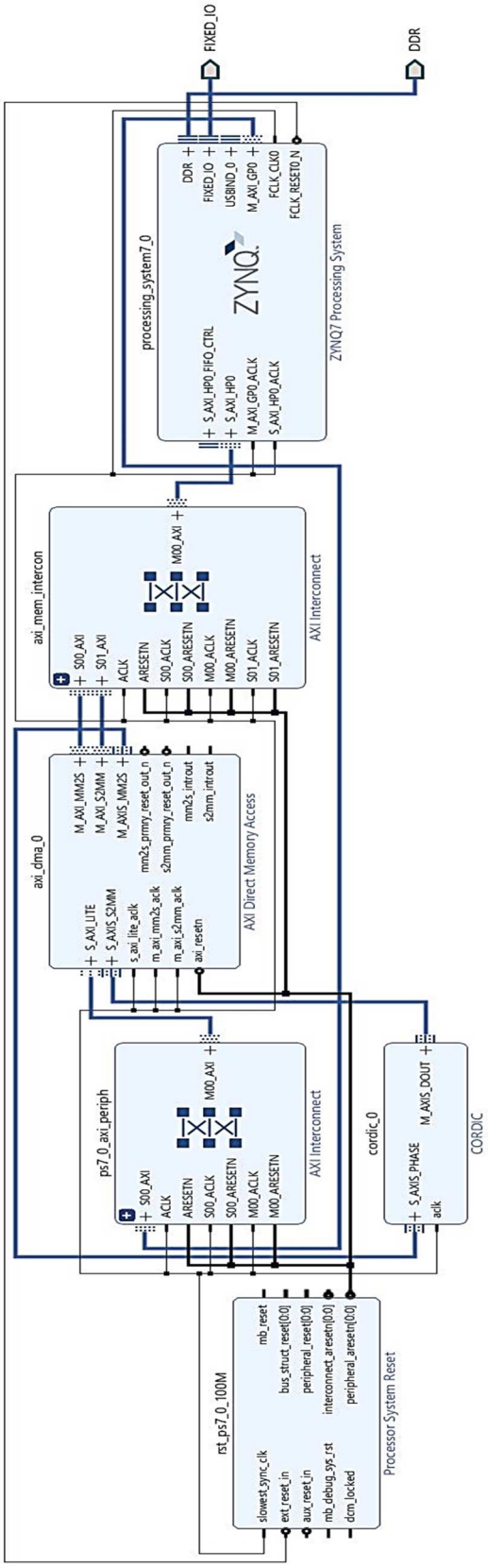

Fig. 6.3: Vivado Block Design View of the plain benchmark

6.2.2 Processing System

Concerning the PS side of the application, a C-based script has been developed. Such a code had two main objectives: first, the initialization of the cores such as the enabling of caches, configuration of the DMA and the PS itself. The second one is the managing of the data transfer to the host PC from the CORDIC (and vice versa) using the DMA. The most note-worthy code slices are now briefly explained:

- *Read/write buffers memory allocation:*

```
#define DDR_BASE_ADDR          0x00100000
#define MEM_BASE_ADDR          (DDR_BASE_ADDR + 0x1000000)
#define TX_BUFFER_BASE         (MEM_BASE_ADDR + 0x00100000)
#define RX_BUFFER_BASE         (MEM_BASE_ADDR + 0x00300000)
#define RX_BUFFER_HIGH         (MEM_BASE_ADDR + 0x004FFFFF)
```

This section of the code declared the memory allocations for the read/write buffers, starting from a base address by adding offsets. These base addresses are automatically defined when the hardware platform is exported, in the header file "xparameters.h".

- *Initialization of PS and DMA:*

```
int main()
{
     init_platform();
     ps7_post_config();
     myDmaConfig =
XAxiDma_LookupConfigBaseAddr(XPAR_AXI_DMA_0_BASEADDR);
     status = XAxiDma_CfgInitialize(&myDma, myDmaConfig);
     if(status != XST_SUCCESS)
     {
          return -1;}
```

The PS and the DMA are initialized in the first lines of the main function. Custom Xilinx functions and classes are defined in the header files “xil_io.h” and “xaxidma.h”.

- *Input vector declaration and conversion:*

```
for (int i=0; i<DIM; i++)
     {
          value = -pi+2*pi*rand()/RAND_MAX;
          TxBufferPtr[i] = inputDataConverter(value);
          #inputCheck[i] = value;
          #outputCheck[i][0] = sin(inputCheck[i]);
          #outputCheck[i][1] = cos(inputCheck[i]);
     }
```

The input vectors (of dimension DIM, user-defined) are generated randomly, then converted by the inputDataConverter() function. This custom function has been developed in order to feed the CORDIC block with two-complement values [36]. Sine and cosine values have also been computed by the PS in order to doublecheck the correctness of the algorithm in the first development stages.

- *Data transferring:*

```
    status =
XAxiDma_SimpleTransfer(&myDma,(UINTPTR)RxBufferPtr,
DIM*sizeof(u32), XAXIDMA_DEVICE_TO_DMA);
    if (status != XST_SUCCESS)
    {
     return XST_FAILURE;
     }

     status =
XAxiDma_SimpleTransfer(&myDma,(UINTPTR)TxBufferPtr,
DIM*sizeof(u32), XAXIDMA_DMA_TO_DEVICE);
     if (status != XST_SUCCESS)
     {
          return XST_FAILURE;}
```

This section manages the data transfer “Device to DMA” and “DMA to Device” in series. It is possible to notice the different memory address in which the function stores the values (RxBufferPtr and TxBufferPtr). The XaxiDma_SimpleTransfer() function exploits the polling transferring as opposite to the interrupt system. This choice has been made for sake of simplicity only.

- *Output vectors and conversion:*

```
for(int i=0; i<DIM; i++)
    {
     if(outputCheck[i][0] -
outputDataConverterSin(RxBufferPtr[i]) <= tol &&
outputCheck[i][1] - outputDataConverterCos(RxBufferPtr[i])
<= tol)
     {
          checkFlag[i] = 1;
     }
     else
          checkFlag[i] = 0;
     printf("%d. sin(%lf) = %lf, cos(%lf) = %lf   %d\n", i,
inputCheck[i], outputDataConverterSin(RxBufferPtr[i]),
inputCheck[i], outputDataConverterCos(RxBufferPtr[i]),
checkFlag[i]);
    }
    printf("\n");
    printf("end\n");
```

This last section involves the print of the output vectors and a conversion from two-complement to binary.

6.3 Experimental Setup

The chosen hardware setup for both the experiments is a Pynq Z2 Evaluation Board of the Xilinx 7000 AP-SoCs family, developed by TUL, shown in Figure 6.4.
This device implements both a reconfigurable hardware and a processing system. In details, the main features used for the purpose are:

- 650MHz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
- Programmable logic equivalent to Artix-7 FPGA
- 630 KB of fast block RAM

The communication between the board and the host computer took place by means of serial connection, mainly due to its integration and ease of use within the Python environment.

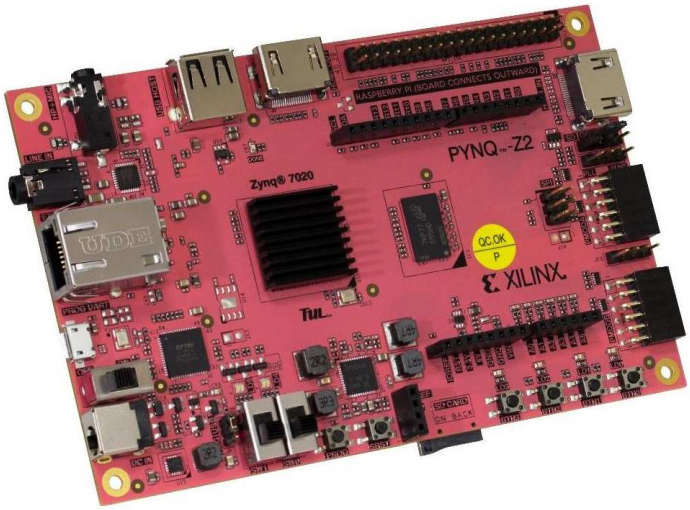

Fig. 6.4: Pynq Z2 Evaluation Board

6.4 Error Classification

The software routine running on the processor system stimulates the cores on the programmable logic. The obtained results are compared with the golden results to detect misbehaviours. If a mitigation approach based on replication is applied, the software running on the processor system runs the computation on each replicated module. The results of the cores are voted and compared to each other to correct or detect errors. The misbehaviours resulting from the fault injection campaigns have been classified into four categories:

- *Data Unavailability* (DU): data unavailability is encountered when it is not possible to receive any results from the PL, usually due to faults affecting the communication modules.
- *Silent Data Corruption (SDC)*: silent data corruption occurs when the results obtained by the PL have errors, but they are detectable only through comparison with the expected results (i.e., there is no cores replication or the voting process elected the wrong result).
- *Recoverable Data Corruption (RDC)*: it occurs when different results are returned by the cores, but the correct results are recovered through voting.
- *Detectable Data Corruption (DDC)*: it occurs when different results are returned by the cores and it is not possible to vote a result (e.g., in a TMR design, two modules return two different results and the third one is unavailable).

6.5 Evaluation of IDF on Plain Benchmark

Two versions of the plain benchmark design (i.e., without TMR) have been implemented using standard design flow and the state-of-the-art IDF. The reliability of the two designs has been evaluated through a fault injection campaign, emulating SEUs in the configuration memory. In the design implemented using state-of-the-art IDF, the block in the higher level of hierarchy (i.e., Zynq PS, AXI DMA, AXI Interconnect, and CORDIC) have been selected to be placed, isolated as required by IDF application notes.

The SEU fault model has been evaluated for each design through two different fault injection campaigns. Each campaign consists of 10,000 fault injections affecting the EB of the benchmark design implemented with and without state-of-the-art IDF. Table I describes the number of EB in each configuration and their percentage over the CM bits.

Table 1: Essential Bits of standard and IDF configurations

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

The campaign resulted in an error rate of 2.97% for the design without IDF and 2.45% for the design implemented using IDF.

The distribution of the errors is reported in Table II and Figure 6.5.

Table 2: Distribution of errors for Standard and IDF designs considering 10,000 injections

Error Type	**Implementation Flow**	
	Standard	**IDF**
DU [#]	103	97
SDC [#]	194	148
Total [#]	297	245

Fig. 6.5: Bar chart distribution of errors for Standard implementation and state-of-the-art IDF designs considering 10,000 injections

Using IDF, the error rate is reduced by 17.51%, acting only on the design placement constraints. The overhead in terms of utilization is reported in Table III. As can be observed, the amount of additional resources is negligible with respect to the available ones. However, IDF design requires about 10% more flip-flops and LUTs compared to the standard one, which can result problematic for more complex circuits.

Table 3: Resources utilization for Standard and IDF designs

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

6.6 Evaluation of Domains-based IDF on TMR

6.6.1 Isolation Policies for Redundant Designs

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

- *Standard (unconstrained) configuration:* this benchmark has been implemented without IDF constraints, thus the modules are not isolated in

this version. The block scheme of the modules at the higher level of the hierarchy is presented in Figure 6.6.

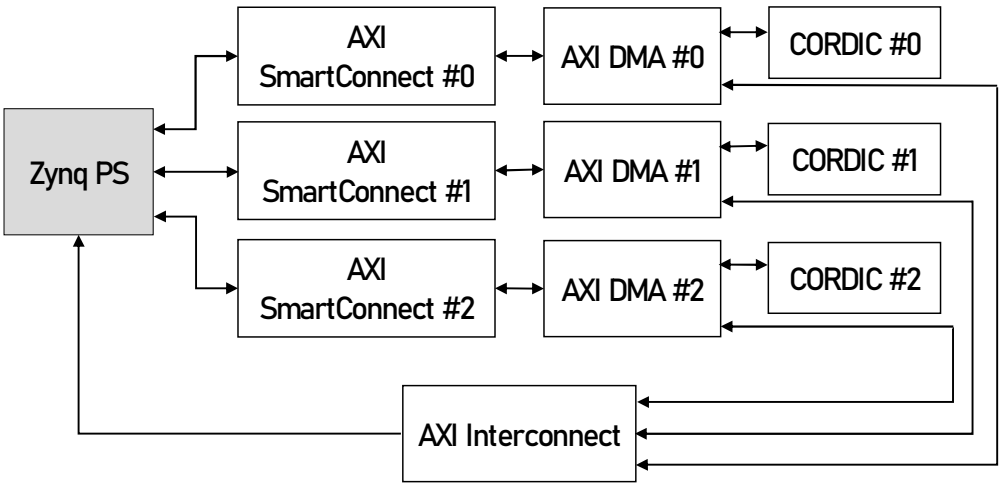

Fig. 6.6: Block scheme of the Standard configuration

- *Domain-based IDF configuration*: this design implements the domains-based IDF. The modules of a domain are grouped together in a block. The blocks are isolated using IDF. A single isolated block is composed of an AXI SmartConnect block, AXI DMA and the CORDIC IP, as represented in Figure 6.7.

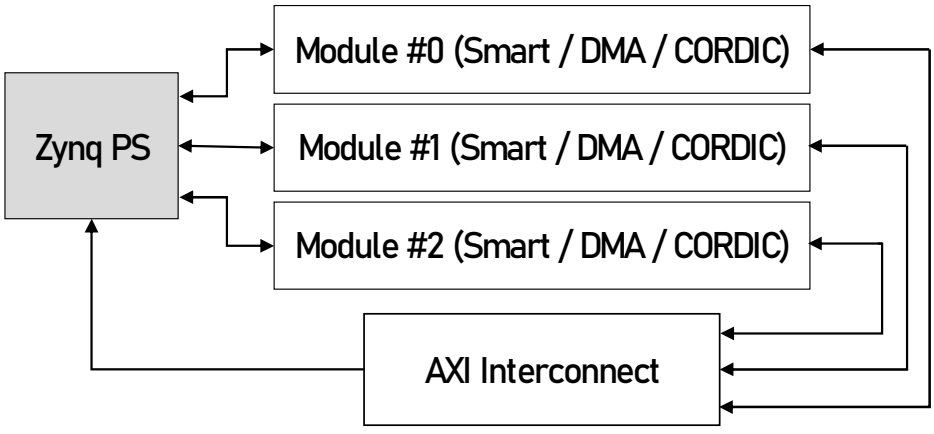

Fig. 6.7: Block scheme of the Domains-based IDF configuration

- *Non-domains-based IDF configuration*: this last isolation patter, represented in Figure 6.8, groups together modules by task. AXI SmartConnect, DMA and CORDIC blocks of the different domains are grouped among them. IDF is applied to these groups. This configuration has been proposed to evaluated the benefits of using the proposed Domains-based aggregation policy with respect to an aggregation policy aiming only to minimize the number of modules to be placed, without taking into account the concept of domains.

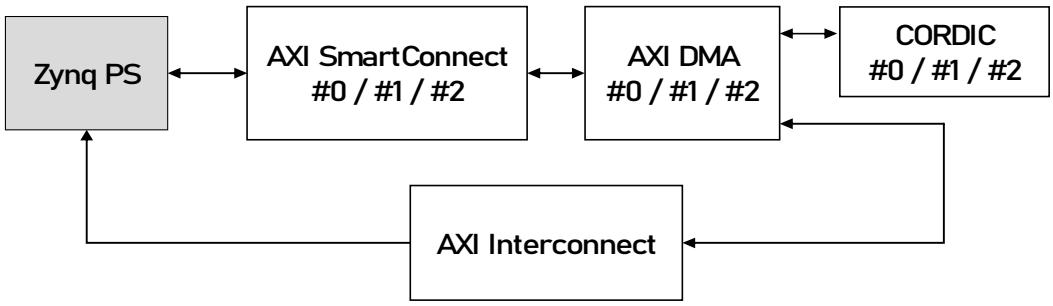

Fig. 6.8: Block scheme of the Non-domains-based IDF configuration

Both of the IDF configurations isolate singularly the AXI Interconnect Module as it is recognized as a weak point of the design [37][38].

6.6.2 Results of the Fault Injections

The fault injection campaigns consist of 10,000 injections emulating SEUs in configuration memory, affecting the EB of different versions of the TMR-hardened benchmark circuit implemented using different design flows. The total number of CM bits and the EB for each design has been reported in Table 4.

Table 4: Essential Bits of Standard TMR, domains-based and non-domains-based configurations

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

Concerning the resource utilization, a slightly higher requirement of logic resources when adopting IDF has been observed, similarly to what was obtained with the plain benchmark. In particular, IDF needs almost 1.5% of LUTs and 2% more FFs than the standard configuration when using IDF, as is reported in Table 5.

Table 5: Resources utilization for standard TMR and IDF designs

Resources	**Implementation Flow**					
	Standard TMR		**Domains-based**		**Non-domains-based**	
	Used [#]	**Utiliz.** [%]	**Used** [#]	**Utiliz.** [%]	**Used** [#]	**Utiliz.** [%]
LUTs	12,878	24.21	13,064	24.56	13,204	24.84
Flip-Flops	17,706	16.64	17,713	16.65	18,037	16.95
Memories	15	10.71	15	10.71	15	10.71

the fault injections have been carried by randomly targeting the EB of the designs. Due to the definition of EB, not all the injections will affect bits programming the used resources of the design. As matter of fact, only some of them will cause an error in the output of the application. This can happen as a result of a fault injected in the used resources or the activation of an unused resource that leads to a conflict.

Considering the three possible configurations, the following results have been observed:

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

The collected data are reported in detail in Table 6. Comparing both of the configurations with the unconstrained design, it can be observed that due to the IDF implementation, the total error rate is slightly dropped with focus on the silent errors (no SDC and 0.09% of the total with the domains-based and non-domains-based configurations, respectively). The domains-based design produced the lowest error rate as well as the highest RDC ratio among the analyzed implementations. Such achievements are also supported by the absence of SDCs, which represent the worst possible behaviour due to their undetectability. The non-domains-based implementation also brought the decrease of SDC with respect to the standard design. However, this configuration appeared to be very sensitive to the DU, which occurred more than three times with respect to the domains-based case. This is likely due to its by-task aggregation, where a fault in one of the communication modules propagates to the communication infrastructure of all the domains. Figure 6.9 compares the error classification resulting from the three implementation

methodologies. Further analysis has been performed on the causes of the Data Unavailability errors affecting the three designs. In particular, it investigated the contribution of the AXI Interconnect module to these errors compared to the other isolated regions of the designs. Using the PyXEL framework, the physical resources affected by the faults resulting in DU errors have been identified. Then, retrieving which module is associated with the physical resource, the DU errors have been grouped in two categories: A*XI Interconnect-fault* (AI-F) and *domain-fault* (D-F). The AI-Fs occur when the fault injection resulting in DU error targets a resource of the AXI Interconnect Module. D-F happens when the bit-flip corrupts a memory cell programming a resource not used by the AXI Interconnect Module. Table 7 and Figure 6.10 report the results of DU categorization.

Table 6: Distribution of errors for Standard TMR, Domains-based and Non-domains-based IDF designs considering 10,000 injections

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

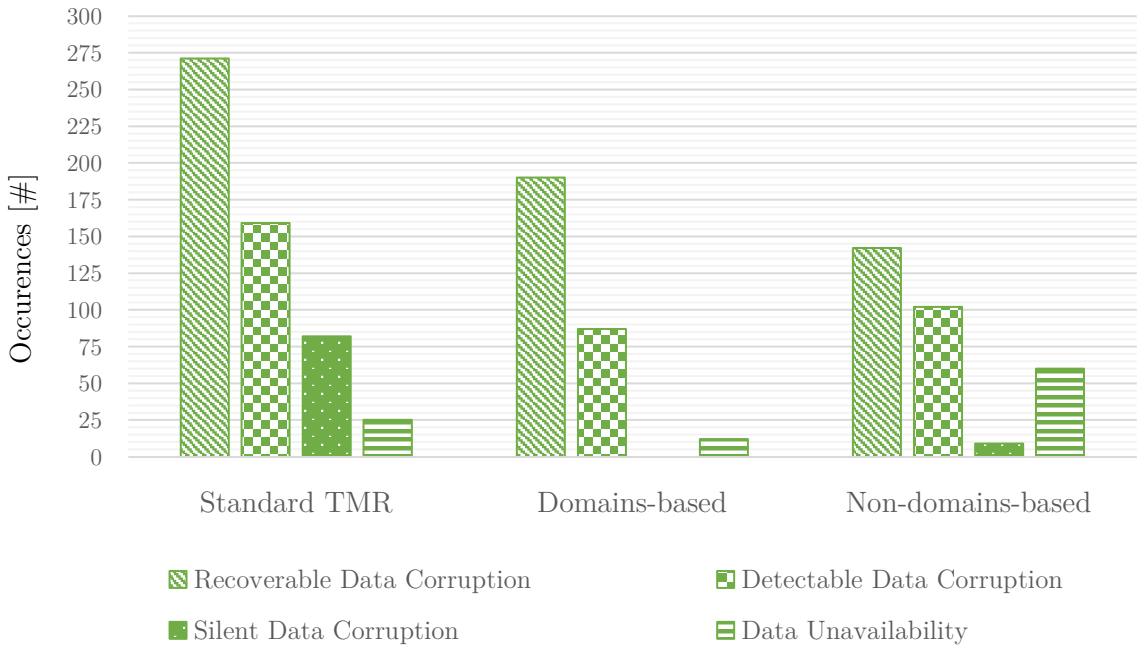

Fig. 6.9: Bar chart distribution of errors for Standard TMR, Domains-based and Non-domains-based IDF designs considering 10,000 injections

Table 7: Data unavailability analysis Standard TMR, Domains-based and Non-domains-based IDF designs

Category	**Implementation Flow**		
	Standard TMR	**Domains-based**	**Non-domains-based**
AI-F [#]	13	11	54
D-F [#]	12	1	6
Total [#]	25	12	60

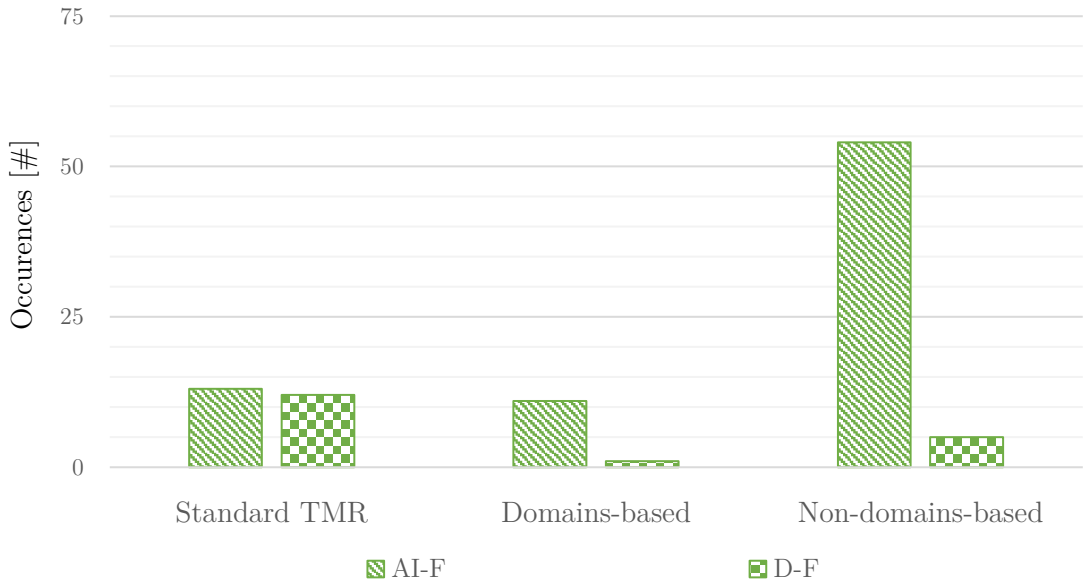

Fig. 6.10: Bar chart distribution of errors for Standard TMR, Domains-based and Non-domains-based IDF designs considering the Unavailability of Data scenarios

The analysis we performed has brought out that the source of data unavailability when IDF is applied is the AXI Interconnect in about 90% of the experiment. Since the normalized values of AI-F for non-domain- and domain-based are comparable, it is likely the high number of DUs observed in non-domain-based IDF design are due to random fault injection produced a higher number of faults in the AXI Interconnect module. However, it is interesting to notice how in standard TMR, where AXI Interconnect is not isolated with respect to the other modules, the contribution of the faults injected in the AXI Interconnection to DU is much lower. It is reasonable to suppose that errors not affecting AXI Interconnect when isolation is not applied, more easily propagate to AXI Interconnect producing DUs. This effect is probably prevented when IDF is applied, making faults in the AXI Interconnect module the main cause of DUs.

Chapter 7

Conclusions and Future Work

This thesis focuses on the effectiveness of IDF for redundant designs implemented on R-SoCs, proposing a set of design guidelines in the case of complex systems unable to implement state-of-the-art IDF, due to topological issues. The proposed Domains-based Isolation Design Flow has been proved capable of mitigating radiation-induced faults for mid-to-high complex designs. Moreover, it has shown an increase in the effectiveness of TMR. Several fault injection campaigns have been carried out on Xilinx Zynq-7000 AP-SoC and the CORDIC algorithm as application benchmark. In particular, Domains-based IDF prevented all Silent Data Corruption errors and increased the recoverable errors by about 33%. Further analysis has been performed on the robustness of such a design technique against unavailability of data.

Future works perspectives include the development of a placement algorithm to automatize the floorplanning process following the domains-based IDF design rules and the application of IDF to SoPC-based computational clusters.

Bibliography

[1] H. Asadi, et al., "Soft Error Susceptibility Analysis of SRAM-Based FPGAs in High-Performance Information Systems," in *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2714-2726, Dec. 2007.

[2] B. Du et al., "Radiation-induced Single Event Transient effects during the reconfiguration process of SRAM-based FPGAs," in *Microelectronics Reliability*, vol. 100-101, Sept. 2019.

[3] R. C. Baumann, Landmarks in terrestrial single-event effects," *Nuclear and Space Radiation Effects Conference*, San Francisco, USA, 2013.

[4] W. Wang, et al., "The research of FPGA reliability based on redundancy methods", in *International Conference on Computer Science and Network Technology*, Harbin, China, 2011, pp. 1608-1611.

[5] Z. Wang, et al., "The reliability and availability analysis of SEU mitigation techniques in SRAM-based FPGAs" in *European Conference on Radiation and Its Effects on Components and Systems*, Brugge, Belgium, 2009, pp. 497-503.

[6] W. Lie and W. Feng-yan, "Dynamic Partial Reconfiguration in FPGAs," in *Third International Symposium on Intelligent Information Technology Application*, Nanchang, China, 2009, pp. 445-448.

[7] S. Azimi and L. Sterpone, "Digital Design Techniques for Dependable High Performance Computing," *2020 IEEE International Test Conference (ITC)*, Washington, DC, USA, 2020, pp. 1-10.

[8] Xilinx, "7 Series FPGAs Configuration – UG470," Xilinx, 2018, pp. 104-107.

[9] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage and T. Turflinger, "Digital Single Event Transient Trends With Technology Node Scaling", *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3462-3465, Dec. 2006.

[10] M. Ceschia et al., "Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs," *in IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2088-2094, Dec. 2003.

[11] E. Hallet, "Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (ISE Tools)," *Application Note: 7 Series FPGAs and Zynq-7000 AP SoC*, 2015.

[12] L. Gantel, et al., "Module relocation in Heterogeneous Reconfigurable Systems-on-Chip using the Xilinx Isolation Design Flow," in *International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, 2012, pp. 1-6.

[13] J. Rettkowski, et al., "RePaBit: Automated generation of relocatable partial bitstreams for Xilinx Zynq FPGAs," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2016, pp. 1-8.

[14] K. Pham, et al., "IPRDF: An Isolated Partial Reconfiguration Design Flow for Xilinx FPGAs," in *IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, Hanoi, Vietnam, 2018, pp. 36-43.

[15] M. Wirthlin, "High-reliability FPGA-based systems: space, high-energy physics, and beyond," in *Proceedings of the IEEE*, 2015.

[16] B. Harikrishna and S. Ravi, "A survey on fault tolerance in FPGAs," *2013 7th International Conference on Intelligent Systems and Control (ISCO)*, 2013, pp. 265-270.

[17] E. Stott, P. Sedcole and P. Y. K. Cheung, "Fault tolerant methods for reliability in FPGAs," *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 415-420, doi: 10.1109/FPL.2008.4629973.

[18] A. Sanchez, et al., "Evaluation of TMR effectiveness for soft error mitigation in SHyLoC compression IP core implemented on Zynq SoC under heavy ion radiation," in *IEEE Internation Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Noordwijk, Netherlands, 2019.

[19] M. Niknahad, O. Sander and J. Becker, "FGTMR - Fine grain redundancy method for reconfigurable architectures under high failure rates", *The 16th North-East Asia Symposium on Nano Information Technology and Reliability*, pp. 186-191, 2011.

[20] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using Partial TMR," *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, 2007, pp. 1-8.

[21] P. Tummeltshammer and A. Steininger, "On the role of the power supply as an entry for common cause faults—An experimental analysis", *2009 12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pp. 152-157, 2009.

[22] Y. Li, B. Nelson and M. Wirthlin, "Synchronization Techniques for Crossing Multiple Clock Domains in FPGA-Based TMR Circuits", *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3506-3514, Dec. 2010.

[23] L. Sterpone and L. Boragno, "Analysis of radiation-induced cross domain errors in TMR architectures on SRAM-based FPGAs," *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017, pp. 174-179

[24] S. Azimi, B. Du, Boyang And L. Sterpone, “On the prediction of radiation-induced SETs in flash-based FPGAs,” *Microelectronics Reliability*. 64, 2016.

[25] S. Azimi, B. Du, L. Sterpone, D. M. Codinachs and L. Cattaneo, "SETA: A CAD Tool for Single Event Transient Analysis and Mitigation on Flash-Based FPGAs," *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2018, pp. 1-52.

[26] L. Sterpone, F. Luoni, S. Azimi and B. Du, "A 3-D Simulation-Based Approach to Analyze Heavy Ions-Induced SET on Digital Circuits," in *IEEE Transactions on Nuclear Science*, vol. 67, no. 9, pp. 2034-2041, Sept. 2020

[27] L. Sterpone and S. Azimi, "Radiation-induced SET on Flash-based FPGAs: Analysis and Filtering Methods," *ARCS 2017; 30th International Conference on Architecture of Computing Systems*, 2017, pp. 1-6.

[28] Azimi, S., L. Sterpone, B. Du and L. Boragno. “On the analysis of radiation-induced Single Event Transients on SRAM-based FPGAs,” *Microelectronic Reliability,* pp. 88-90, 2018.

[29] C. De Sio, S. Azimi, L. Sterpone and B. Du, "Analyzing Radiation-Induced Transient Errors on SRAM-Based FPGAs by Propagation of Broadening Effect," in *IEEE Access*, vol. 7, pp. 140182-140189, 2019.

[30] C. De Sio, S. Azimi, L. Bozzoli, B. Du and L. Sterpone, “Radiation-induced Single Event Transient effects during the reconfiguration process of SRAM-based FPGAs,” *Microelectronics Reliability*, pp. 100-101, 2019.

[31] A. Portaluri, C. De Sio, S. Azimi and L. Sterpone, "A New Domains-based Isolation Design Flow for Reconfigurable SoCs," *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 1-7, 2021.

[32] Xilinx, "Vivado Isolation Verifier," Xilinx, 2020.

[33] L. Bozzoli, et al., "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," in *International Symposium on Rapid System Prototyping (RSP),* Turin, Italy, 2018, pp. 70-75.

[34] Xilinx, "Soft Error Mitigation Controller v4.1 Product Guide," Xilinx, 2018.

[35] Xilinx, "AXI DMA v7.1 LogiCORE IP Product Guide," Xilinx Product Specification, 2019.

[36] Xilinx, "CORDIC v6.0 LogiCORE IP Product Guide," Xilinx Product Guide, 2021.

[37] C. De Sio, et al., " On the analysis of radiation-induced failures in the AXI interconnect module," in *Microelectronics Reliability*, pp. 243-254, 2020.

[38] C. De Sio et al., "On the Evaluation of SEU Effects on AXI Interconnect Within AP-SoCs," in *2020 Architecture of Computing Systems – ARCS*, 2020.