Politecnico di Milano
Dipartimento di Elettronica e Informazione
Dottorato di Ricerca in Ingegneria dell'Informazione

A Reliability-aware Design Methodology for Embedded Systems on Multi-FPGA Platforms

Doctoral Dissertation of:
Chiara Sandionigi
738757

Advisor:
Prof. Cristiana Bolchini
Tutor:
Prof. Donatella Sciuto
Supervisor of the Doctoral Program:
Prof. Carlo Ettore Fiorini

2011 – XXIV

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32 I-20133 — Milano

A mia mamma

Ringraziamenti

Tre anni di dottorato sembrano ora essere passati molto velocemente, forse troppo velocemente. Qualche tempo fa stavo discutendo una tesi che segnava la fine del mio periodo da studentessa di ingegneria, e ora ho appena concluso una nuova tesi che segnerá la fine di questa intensa e bellissima esperienza al Politecnico. Tra questi due momenti, che a me sembrano essere cosí vicini tra loro, c'é stata una vita vissuta a Milano, c'é stato un anno trascorso in Olanda tra la vita a Leiden e il lavoro in ESA, ci sono stati i viaggi, le conferenze, le riunioni e gli eventi della PhDEI association, e cosí tante esperienze da non poter riassumere in queste poche righe. É stato un percorso professionale e personale che non avrei potuto compiere senza il sostegno di alcune persone, a cui vanno i miei ringraziamenti.

La prima persona che ringrazio é Cristiana, relatrice di questa tesi. Noi non ci siamo propriamente scelte per questo percorso di dottorato, ma credo che difficilmente avrei potuto avere advisor migliore. Lei é sempre stata presente, anche quando ero in Olanda, dandomi consigli e indicazioni su come portare avanti il mio lavoro. Questa tesi non sarebbe stata possibile senza il suo aiuto.

Un grazie per i preziosi consigli va anche ad Antonio. Forse a volte con altri ha risposto con aria di sufficienza (scusa Antonio, ma non potevo non scriverlo :)), ma con me Antonio é sempre stato molto disponibile e la sua esperienza mi é stata di enorme aiuto. Inoltre lui é stato parte di un ufficio che mi dispiacerá veramente molto lasciare, insieme a Christian, Daniele, Luigi, Marco e PRG.

Un ringraziamento va poi ad ESA, e in particolare alla Sezione di Microelettronica, senza il cui supporto questo dottorato non sarebbe stato possibile. Grazie per avermi dato la possibilitá di lavorare un anno in ESTEC, e in generale di vivere un anno in Olanda, e grazie per il sostegno, in particolare grazie a Luca e a David per il loro aiuto. Allo stesso modo ringrazio Donatella e Marco, senza i quali questa esperienza per me non sarebbe neanche iniziata. Il dottorato per me é stato un'esperienza importante, una scelta che rifarei senza esitazioni, quindi grazie infinite per avermi dato questa opportunitá.

Al Politecnico ho potuto conoscere e lavorare con persone che spero faranno sempre in qualche modo parte della mia vita, dovunque andró

in futuro. Tra queste persone c'é Danilo, mio relatore di minore e amico. Lavorare con lui é stato istruttivo e divertente allo stesso tempo, e spero tanto di poter ripetere questa esperienza in futuro.

Infine, parlando del Politecnico, non posso dimenticare la PhDEI association, e in particolare la mia "mamma mora" e la mia "mamma lampone", Annalisa e Rossella. Grazie per i sorrisi e grazie perché so che ci saranno sempre nonostante la lontananza.

Ed eccomi ora arrivata alla vita al di fuori delle mura del Poli e ai ringraziamenti per i miei migliori amici. Un grazie di cuore va a Stefania, mia migliore amica, che mi é accanto sempre e comunque. Spero che lei sappia quanto é importante per me e quanto io apprezzi tutto ció che fa. Un grazie va poi a Michele, sempre pronto ad ascoltare (e a tollerare che gli invada casa a qualsiasi ora :)). E un grazie va a Fabrizio, che é in qualche modo entrato nella mia vita solo di recente, ma mi ha ascoltato e supportato (o sopportato) molto durante questi ultimi mesi di dottorato.

Un grazie di cuore é poi naturalmente per Alessio, mio migliore amico e ormai da tempo confidente. So che per noi lontananza vuol dire solo avere piú cose da raccontarci quando ci rivedremo.

Infine, "last but not least", un ringraziamento va alla mia famiglia e in particolare alla persona a cui é dedicata questa tesi, mia mamma, che da sempre mi supporta qualunque siano le mie scelte.

Grazie a tutti per questi tre importanti e intensi anni che hanno portato a questa tesi.

Abstract

In the last decades, Static RAM (SRAM) Field Programmable Gate Arrays (FPGAs) have become an attractive technology for the electronics of embedded systems, for both fast prototyping and final production. Their most attractive feature is the flexibility, related to the opportunity of re-programming (or reconfiguring) the device in a few clock cycles and on-line, while the device is operating. This allows the system implemented on the FPGA to be updated/upgraded, also remotely.

SRAM-based FPGAs are currently employed in many applicative domains and their use is being investigated also for mission-critical applications, such as the space scenario, the reference settings for this thesis. In fact, in the space environment, such devices are an attractive technology for the possibility of remote update/upgrade, coping with the difficulty of system maintenance. Nevertheless, the harsh environmental conditions and the long lifetime required for space systems prevent the straightforward adoption of SRAM-based FPGAs, due to their susceptibility to faults. In fact, this same environment is particularly critical for static RAM technology, susceptible to physical phenomena that cause both transient and permanent faults. Thus, appropriate tolerance and recovery techniques are adopted; in general, fault tolerance techniques are mostly based on spatial redundancy, whereas recovery ones exploit the FPGAs' reconfiguration capability, allowing to cope with the occurrence of faults by re-programming the faulty parts.

Indeed, hardened systems may require many resources due to their size and complexity, hence a single device may not suffice in terms of available resources, and multi-FPGA solutions are taken into account and investigated. Furthermore, the availability of more devices on multi-FPGA platforms can be exploited to increase reliability as the overall functionality is spread over multiple devices, possibly also using redundant implementations.

The research proposed in this PhD thesis fits in the described scenario. The aim is the definition of a reliability-aware methodology for the design of embedded systems on multi-FPGA platforms. The designed system must be able to detect faults occurrence globally and autonomously, in order to recover or to mitigate the effects of the faults. Two categories of faults are identified, based on their impact on the device elements;

i) *non-recoverable* faults, that are those that cause a permanent problem, making the portion of the fabric unusable, and ii) *recoverable* faults, transient problems that can be fixed without causing a lasting effect.

While some aspects can be taken from previous solutions available in literature, several open issues exist. In fact, reliability techniques have been widely addressed in the case of systems based on a single FPGA, but the problem extended to multi-FPGA platforms has been rarely taken into account. Indeed, the single FPGA approaches can not be straightforwardly adopted for the multi-FPGA scenario, that poses several challenges, as, for instance, the partitioning of the system among the available devices. Furthermore, in the previous contributions, only recoverable faults are usually targeted, whereas non-recoverable ones are rarely analyzed. Thus, no complete design methodology handling all the peculiar issues of the considered scenario has been proposed yet, a gap we aim at filling with our work.

Not only we define the methodology in its relevant aspects, but we also design and develop a framework for supporting the designer in its application. The final system thus exposes reliability properties and increases its overall lifetime and availability.

Riassunto

Negli ultimi decenni, i dispositivi FPGA (Field Programmable Gate Array) basati su SRAM (Static RAM) sono diventati una tecnologia di interesse per l'elettronica dei sistemi dedicati, sia per la prototipazione veloce che per la produzione finale. La loro caratteristica di maggiore interesse é la flessibilitá, relativa all'opportunitá di riprogrammare (o riconfigurare) il dispositivo in pochi cicli di clock e on-line, mentre il dispositivo é operativo. Questo permette di aggiornare/migliorare il sistema implementato sulla FPGA, anche remotamente.

Le FPGA basate su SRAM sono attualmente impiegate in molti domini applicativi e il loro uso viene vagliato anche per applicazioni critiche, come lo scenario delle missioni spaziali, di riferimento per questa tesi. Infatti, nell'ambiente spazio, tali dispositivi sono una tecnologia di interesse per la possibilitá di aggiornamento/miglioramento remoto, che fa fronte alla difficoltá di manutenzione del sistema. Tuttavia, le difficili condizioni ambientali e il lungo tempo di vita richiesto per i sistemi per lo spazio ostacolano l'immediata adozione delle FPGA basate su SRAM, a causa della loro sensibilitá ai guasti. Infatti, questo stesso ambiente é particolarmente critico per la tecnologia RAM statica, sensibile a fenomeni fisici che causano guasti sia transitori che permanenti. Di conseguenza, vengono adottate appropriate tecnologie di tolleranza e recupero; in generale, le tecniche di tolleranza sono per lo piú basate sulla ridondanza spaziale, mentre quelle di recupero sfruttano la capacitá di riconfigurazione delle FPGA, permettendo di far fronte al verificarsi dei guasti riprogrammando le parti guaste.

Invero, i sistemi irrobustiti possono richiedere molte risorse a causa della loro dimensione e complessitá, quindi un singolo dispositivo puó non essere sufficiente in termini di risorse disponibili, e le soluzioni multi-FPGA vengono prese in considerazione e valutate. Inoltre, la disponibilitá di piú dispositivi sulle piattaforme multi-FPGA puó essere sfruttata per aumentare l'affidabilitá poiché la funzionalitá globale é distribuita su piú dispositivi, eventualmente anche usando implementazioni ridondanti.

La ricerca proposta in questa tesi di dottorato si inserisce nello scenario descritto. L'obiettivo é la definizione di una metodologia orientata all'affidabilitá per il progetto di sistemi dedicati su piattaforme multi-FPGA. Il sistema progettato deve essere in grado di identificare il verificarsi di

guasti globalmente e autonomamente, per recuperare o mitigare gli effetti dei guasti. Vengono identificate due categorie di guasti, in base al loro impatto sugli elementi del dispositivo: i) guasti *non recuperabili*, che sono quelli che causano un problema permanente, rendendo inutilizzabile la porzione del dispositivo, e ii) guasti *recuperabili*, problemi transitori che possono essere risolti senza causare un effetto persistente.

Nonostante alcuni aspetti possano essere presi da soluzioni precedenti disponibili in letteratura, esistono molti punti aperti. Infatti, le tecniche di affidabilitá sono state ampiamente affrontate nel caso di sistemi basate su una singola FPGA, ma il problema esteso alle piattaforme multi-FPGA é stato raramente preso in considerazione. Invero, gli approcci per singola FPGA non possono essere direttamente adottati per lo scenario multi-FPGA, che pone molte sfide, come, per esempio, il partizionamento del sistema tra i dispositivi disponibili. Inoltre, nei contributi precedenti, sono di solito trattati solo i guasti recuperabili, mentre quelli non recuperabili sono raramente analizzati. Di conseguenza, non é ancora stata proposta una metodologia di progetto completa che gestisca tutti i problemi peculiari dello scenario considerato, un vuoto che puntiamo a colmare con il nostro lavoro.

Non solo definiamo la metodologia nei suoi aspetti rilevanti, ma inoltre progettiamo e sviluppiamo una struttura che supporti il progettista nella sua applicazione. Di conseguenza il sistema finale espone proprietá di affidabilitá e aumenta il suo tempo di vita globale e la sua disponibilitá.

Contents

List of Figures

List of Tables

1. Introduction

In the last decades, Field Programmable Gate Arrays (FPGAs) have established themselves as target technology for both fast prototyping and final production of embedded systems. Among the many different families, Static RAM (SRAM) FPGAs are the most used devices, since they offer various advantages with respect to other technologies. In particular, the most attractive feature of SRAM-based FPGAs is their flexibility, related to the opportunity of re-programming (or reconfiguring) the device in a few clock cycles and on-line, while the device is operating. This allows the system implemented on the FPGA to be updated/upgraded, also remotely. For this reason, such devices are currently employed in many applicative domains and their use is being investigated also for mission-critical applications, for example the space ones, where the direct maintenance of the system is a difficult task.

Nevertheless, SRAM-based FPGAs are more susceptible to faults than alternative solutions (for example, Application Specific Integrated Circuits, ASICs), thus they can not be straightforward adopted for mission-critical applications unless reliability techniques are applied. In particular for space applications, considered as target scenario of the thesis, reliability is a strict requirement due to the harsh environmental conditions and the difficulty of system maintenance. The widespread diffusion of this kind of devices has led to the investigation and definition of design techniques and methodologies for hardening and recovering FPGA-based systems, with the final aim of exploiting such systems also in mission-critical scenarios. In general, hardening techniques are mostly based on spatial redundancy, whereas recovery ones exploit the FPGAs' reconfiguration capability, allowing to cope with the occurrence of faults by re-programming the faulty parts.

Indeed, as hardened systems require many resources due to their size and complexity, a single FPGA may not suffice in terms of available resources, and multi-FPGA solutions start being taken into account and investigated. Furthermore, the availability of more devices on multi-FPGA platforms could be exploited to implement a distributed engine devoted to fault management. In fact, the use of multiple FPGAs provide an increased reliability as the overall functionality is spread over multiple devices, possibly also using redundant implementations.

While fault detection and masking techniques have been widely addressed in the case of systems based on a single FPGA, the problem extended to multi-FPGA platforms has been rarely taken into account. The multi-FPGA scenario arises various issues, e.g. the partitioning of the system among the available devices, and the single FPGA approaches can not be straightforwardly adopted. In literature, no complete design methodology handling all the peculiar issues of the considered scenario has been proposed yet, a gap we aim at filling with our work. In this thesis, we define a complete methodology for designing reliable embedded systems on multi-FPGA platforms. In the following, the proposed research is introduced. First, we will provide an overview of the research and will highlight the main contributions of our work, presenting also the list of publications where the various aspects of the research have been presented.

1.1. Research overview and statement of originality

The thesis proposes a reliability-aware methodology for designing embedded systems on multi-FPGA platforms, with the final aim of exploiting commercial SRAM-based FPGAs for mission-critical applications. The idea is to achieve fault tolerance against faults by exploiting the reconfigurable properties of the devices. Two categories of faults are identified, based on their impact on the device elements, such that a physical damage occurs or not; i) *non-recoverable* faults, that are those that cause a permanent problem, making the portion of the fabric unusable, and ii) *recoverable* faults, transient problems that can be fixed without causing a lasting effect. Recoverable faults can be mitigated by reconfiguring the system (and possibly only the faulty sub-system portion) with the same configuration used before fault occurrence, whereas non-recoverable faults, being characterized by a destructive effect, lead to the necessity of relocating the functionality to a non-faulty region of the device. The designed system must be able to detect the occurrence of faults globally and autonomously, in order to recover or to mitigate their effects. Thus, the methodology allows the overall system to continue working even if faults occur, increasing both system reliability and lifetime.

The main innovative contributions raised by this thesis are summarized as follows:

- *Definition of an overall reliable multi-FPGA system with distributed control architecture.* Rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover

from faults, we have envisioned a distributed solution, where each FPGA on the platform is in charge of monitoring and, in case of fault, reconfiguring the other devices. The aim is to achieve a higher level of reliability in the overall system, trying to avoid the single point of failure characterizing the centralized solution and requiring to be implemented onto a particular device (e.g., an ASIC or an antifuse-based FPGA).

- *Management of both recoverable and non-recoverable faults.* In literature, only faults recoverable by reconfiguring the FPGA are usually targeted, whereas faults physically damaging the device are rarely analyzed. Nevertheless, also faults of the latter type have become relevant in the embedded systems design in the recent past, due to the increasingly smaller device and wire dimensions and higher operational temperatures, increasing the need to take them into account next to the other faults. We propose a classification strategy and its companion algorithm for the discrimination of faults in FPGAs based on their impact on the device elements and the consequent possibility to recover from them.

- *Design of the engine in charge of implementing the suitable recovery strategy based on the type of fault.* We propose the design of the controller in charge of managing the fault recovery of the multi-FPGA platform, contributing to the creation of the reliable system. More precisely, we introduce a reliability-aware Reconfiguration Controller, aimed at performing the envisioned fault classification and managing the reconfiguration process of the faulty parts of the architecture to mitigate fault effects.

- *Definition of a complete flow for designing autonomous fault tolerant systems on multi-FPGA platforms.* In literature, methodologies for the design of autonomous fault tolerant systems on multi-FPGA platforms have not been proposed yet. We aim at filling this gap by introducing a complete flow for designing such reliable systems. Furthermore, we describe the developed prototype framework, that implements the flow and automates, as much as possible, the design, hardening, and implementation of the envisioned systems.

The innovative points are covered throughout the thesis, that is structured as follows. Chapter 2 presents the motivations of the proposed work and introduces the background elements useful to set the basis for understanding the rest of the thesis. Chapter 3 describes the proposed

reliability-aware design methodology, and Chapter 4 introduces the related design flow and prototype framework. Chapter 5 presents the classification strategy and its companion algorithm for the discrimination of faults into recoverable and non-recoverable. Chapter 6 introduces the design of the reliability-aware Reconfiguration Controller, aimed at performing the fault classification and managing the reconfiguration process. Chapter 7 discusses the methodology evaluation, performed by implementing a real case study. Finally, Chapter 8 closes the presentation of this research, drawing some conclusions and giving some possible future research directions.

1.2. Publications

The various aspects of the research presented in this thesis have been published in international conference proceedings and international journals. The list of papers is the following.

DFT'10 Cristiana Bolchini, David Merodio Codinachs, Luca Fossati, Antonio Miele, Chiara Sandionigi, *A reliable reconfiguration controller for fault-tolerant embedded systems on multi-FPGA platforms*, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2010, pp. 191-199

The paper presents the first proposal of the controller in charge of managing the fault tolerance of multi-FPGA platform. The raised innovative points are the identification of a distributed control architecture, allowing the avoidance of single points of failure, and the definition of an overall reliable multi-FPGA system.

ESL'10 Cristiana Bolchini, Chiara Sandionigi, *Fault classification for SRAM-based FPGAs in the space environment for fault mitigation*, IEEE Embedded Systems Letters, Volume 2, 2010, pp. 107-110

The letter describes the first proposal of fault classification algorithm to discriminate between recoverable and non-recoverable faults occurring in SRAM-based FPGAs. By considering space applications, the controller definition starts from a characterization of the radiation effects and aging mechanisms.

IOLTS'11 Cristiana Bolchini, David Merodio Codinachs, Luca Fossati, Chiara Sandionigi, *A reliable fault classifier for dependable systems on SRAM-based FPGAs*, 17th IEEE International On-Line Testing Symposium, 2011, pp. 105-110

The paper presents an enhanced and formally evaluated fault classification algorithm. With respect to previous approaches, included the one described in the previous letter, the main contributions of the work are the formal definition of the parameters characterizing the algorithm, the evaluation of the conditions for correct fault classification, and the investigation of the reliable implementation of the classifier.

DFT'11 Cristiana Bolchini, Chiara Sandionigi, *A reliability-aware partitioner for multi-FPGA platforms*, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2011, Accepted, To appear

The paper presents a partitioning approach for reliable systems on multi-FPGA platforms. With respect to literature, where the partitioning problem is solved by heuristic algorithms based on non-accurate models, we have proposed a partitioner that identifies the global optimal solution in an acceptable execution time and can be integrated in an overall reliability-aware design flow.

From a wider point of view, part of the research here presented has also been exploited in the single FPGA scenario, when smaller systems are considered, leading to the following publications.

TC'10 Cristiana Bolchini, Antonio Miele, Chiara Sandionigi, *A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs*, IEEE Transactions on Computers, 2010, Accepted, To appear

The paper presents a methodology for the implementation of systems on single SRAM-based FPGAs with soft error mitigation properties. This is the first proposal of complete design flow for realizing reliable FPGA-based systems. The approach has been taken into account and extended for the scenario envisioned in the thesis; more precisely, multi-FPGA platforms have been considered for the implementation of systems able to cope with both recoverable and non-recoverable faults.

ETS'10 Cristiana Bolchini, Antonio Miele, Chiara Sandionigi, Niccoló Battezzati, Luca Sterpone, Massimo Violante, *An integrated flow for the design of hardened circuits on SRAM-based FPGAs*, 15th IEEE European Test Symposium, 2010, pp. 214-219

In this paper, the design flow presented in the previous one has been enhanced. The framework integrates strategies independently

designed to tackle the problem of recoverable faults. Also this approach has been considered and extended for the envisioned scenario.

FPL'11 Cristiana Bolchini, Antonio Miele, Chiara Sandionigi, *Automated resource-aware floorplanning of reconfigurable areas in partially-reconfigurable FPGA systems*, 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 532-538

The paper presents a floorplanner for FPGA-based systems. The proposed floorplanner is based on an accurate model of the devices and takes into account all the elements characterizing them, namely the constraints imposed by the peculiar structure of the fabric and the reconfiguration capabilities.

Finally, the following paper has been submitted to international conference.

DATE'12 Cristiana Bolchini, Antonio Miele, Chiara Sandionigi, *Increasing autonomous fault-tolerant FPGA-based systems' lifetime*, Submitted to Design, Automation and Test in Europe 2012

The paper proposes a methodology for the design of autonomous fault-tolerant FPGA systems, with the final objective of increasing the system's lifetime and availability. The main contribution of the work is the design space exploration, that identifies the most convenient with the final aim of maximizing the number of tolerated non-recoverable faults.

2. Motivations and working scenario

Our research proposes a methodology for designing reliable embedded systems on multi-FPGA platforms, with the final objective of increasing the system's lifetime in critical scenarios. This chapter presents the motivations of the proposed work and introduces the background elements useful to set the basis for understanding the rest of the thesis.

The chapter is structured as follows: Section 2.1 discusses the motivations leading to the proposed thesis. Section 2.2 introduces the fault model that has been adopted as a reference for the definition of the reliability-aware design methodology. Section 2.3 refines the model by referring to the working scenario, namely the space environment, considered for its stringent reliability requirements due to the harsh environmental conditions, the long lifetime of space missions, and the difficult of system maintenance. Finally, Section 2.4 draws the chapter summary.

2.1. Motivations

During the last decades, FPGAs have been adopted as a target technology for both the fast prototyping and final production of embedded systems. Among the many different FPGA families, SRAM-based ones are the most used platforms, due to various advantages they present with respect to alternative solutions. Compared to traditional microprocessors, they reveal higher computational capability, lower power consumption, and lower cost [2]. Moreover, thanks to the ability of changing their behavior over time (feature often referred to as "reconfigurability"), they provide an increased flexibility with respect to alternative hardware solutions, like Application Specific Integrated Circuits (ASICs) and antifuse-based FPGAs, that can be programmed only once. In fact, SRAM-based FPGAs can be reprogrammed in a few clock cycles and on-line, while the device is operating. The advent of such dynamically reconfigurable hardware into the embedded systems domain allows for exploiting the performance of hardware along with the adaptability of software; reconfigurable devices may achieve performance similar to hardware while being as flexible as software, given that they can change the function

they perform. As such, reconfigurable devices are a first class choice for combining both adaptability and performance. Devices similar to SRAM-based FPGAs are Flash-based FPGAs, enabling device reconfiguration through the use of non-volatile configuration memory; nevertheless, such technology presents lower throughput, smaller amount of logic, and higher costs. For these reasons, SRAM-based FPGAs are currently employed in many applicative domains.

Given their peculiarities, SRAM-based FPGAs could be an attractive technology not only for the standard commercial market, but also in mission- or safety-critical applications; for instance, the space environment, considered as a possible application scenario for this work, is particularly critical for the harsh operating conditions and the not affordable maintenance. Nevertheless, such devices are more susceptible to faults with respect to traditional solutions (e.g., ASICs) [3], thus fault mitigation and recovery techniques are paramount to guaranteeing that the system will work correctly. For this class of devices, in general, fault detection/tolerance strategies are mostly based on spatial redundancy, whereas recovery ones exploit the device's reconfiguration capabilities by performing scrubbing, readback or even partial reconfiguration of the faulty parts [4]. When a fault is detected, the affected functionality is re-created on the FPGA, thus restoring the device's overall processing capabilities. Traditionally, the major source of concerns has been mainly represented faults causing errors in the memories devoted to the application data and configuration. These errors can be easily fixed by rewriting the correct information. Nevertheless, also faults physically damaging the device have become relevant for digital systems in the recent past; in fact, as highlighted in 2003 by the International Road Map of Semiconductors [5], such faults will have a higher impact due to the increasingly smaller device and wire dimensions and higher operational temperatures, raising the need to take them into account next to the other faults.

Another aspect gaining importance is the limited amount of functionalities FPGAs can accommodate. Indeed, as the size and complexity of the systems being designed increase (in particular after the application of fault tolerance techniques), a single FPGA may not suffice in terms of available resources, and multi-FPGA solutions start being taken into account and investigated (e.g., [6, 7, 8, 9]). Furthermore, the availability of more devices on multi-FPGA platforms could be exploited to implement a distributed engine devoted to fault management.

While fault detection and masking techniques have been widely addressed in the case of systems based on a single FPGA (e.g., [10, 11, 4]), the problem extended to multi-FPGA platforms has been rarely taken

into account. Moreover, only faults recoverable by reconfiguring the device are usually targeted ([4, 12, 11]), whereas faults physically damaging the device are rarely analyzed. The approach in [1] presents a design flow for implementing digital systems on single SRAM-based FPGAs with soft error mitigation properties. While it could be adapted to multi-FPGA platforms and extended to faults with destructive effect, we claim that an ad-hoc solution for this scenario would consistently improve the system's performance and its reliability. In literature, when considering multi-FPGA platforms, the available devices are usually exploited to host replicas of the main system, as in [8], where three FPGAs are used to apply the classical Triple Modular Redundancy (TMR) technique on the whole circuit. Each FPGA hosts the same configuration and a controller, implemented on an external radiation-hardened ASIC, acts as a TMR voter. Indeed, this is a different scenario with respect to the one we envision; we aim at better exploiting the devices potential and providing a scalable solution, independent of the number of FPGAs used. In addition, the controller needs to be implemented on particular ASIC technology, eliminating most of the advantages of having SRAM-based FPGAs in the system, namely flexibility and relatively low cost. To conclude, no complete design methodology handling all the peculiar issues of the considered scenario has been proposed yet, a gap we aim at filling with our work.

In this thesis, we define a complete methodology for designing reliable embedded systems on multi-FPGA platforms. The final objective is the exploitation of commercial SRAM-based FPGAs for mission-critical applications, realizing reliable systems that autonomously cope with both faults recoverable by reconfiguration and faults physically damaging the device. In the next section, we introduce the adopted fault model, that is at the basis of the proposed methodology.

2.2. Fault model

When designing fault mitigation strategies, the *single fault assumption* is traditionally considered. We adopt such assumption because realistic for the considered scenario. It implies that i) faults occur one at a time and ii) the time between the occurrence of two subsequent faults is long enough to allow the detection of the first fault before the second one occurs. Indeed, the fault only produces an observable effect, an error, if i) the fault occurs in a used resource and ii) the applied input (sequence) is such that a difference in the data/behavior is caused with respect to the fault-free situation. Thus, since it may happen that not all

available resources are actually used, the observability of a fault is related to the probability of the fault to hit a used resource. Furthermore, it is necessary that the adopted detection mechanism identifies the fault situation with short latency between the fault occurrence and the error detection.

It is worth noting that the single fault assumption prevents dealing with faults accumulation, possibly leading to the failure of fault tolerance techniques due to masking/biasing effects. In general, though, since the proposed system is divided into areas, as described in Chapter 3, fault mitigation techniques can also deal with multiple faults, provided they occur in independent portions of the entire system. A full extension to multiple faults management will be considered in future work, although for some sets of multiple failures the proposed approach holds.

When considering SRAM-based FPGAs, it is possible to identify two types of memory that can be subject to faults; the one storing the application data being processed and the one storing the configuration bitstream, that defines the functionality performed by the reconfigurable fabric. If the data memory is corrupted, an erroneous value is produced, whereas, when a fault corrupts a configuration memory element, it modifies the programmed functionality. Configuration memory accounts to more than 95% of the fraction of the device sensitive to faults [12], thus an erroneous functionality is the most common produced effect. Nevertheless, we aim at covering faults affecting both kinds of memories. To cope with erroneous values, a reset of the application can be performed or feedback loops to propagate the correct values to the registers can be added, whereas, to restore erroneous functionalities, the configuration must be re-written.

In the next section, we provide a more detailed characterization of faults by referring to the considered working scenario, namely the space environment.

2.3. Working scenario: the space environment

The working scenario of this thesis is the space environment, selected for the stringent reliability requirements of space applications due to the harsh environmental conditions and the difficulty of system maintenance. In fact, the massive presence of radiations possibly causes glitches in the system elaboration, and maintainability operations are very difficult when considering that the communication with Earth is limited in both availability and bandwidth, necessary to remotely cope with unpredicted failure situations. Two problems have been identified as main causes of

faults in space; radiations and device aging. In the following, they are analyzed in relation to the selected platform, SRAM-based FPGAs, by identifying their main effects and characteristics. Finally, the adopted fault classification is presented.

2.3.1. Radiation effects

Radiations are the most common cause of faults in space. They consist of different particles, mainly electrons, protons and heavy ions, originated from various sources, e.g. the sun, novas, and supernovas. Their energy levels depend on the source and the orbit; in Van Allen belts region, radiations' electrons and protons have energies up to tens and hundreds of MeV, respectively, whereas solar cosmic rays have energies up to GeV, and the galactic cosmic rays up to TeV [13]. Radiations, when hitting an electronic device, can cause a problem mainly observable in memory elements implemented with static cells. The effects can be divided into two main groups [14]:

- *Single Event Effect* (SEE), that is a measurable effect resulting from the deposition of energy from a single ionizing particle strike, and
- *Total Ionizing Dose* (TID) effect, that is a cumulative long term ionizing damage mostly due to protons and electrons.

SEEs can take many forms. They are distinguished into non-destructive or destructive, based on their effect; non-destructive SEEs can be recovered by resetting or reconfiguring the device, whereas destructive SEEs have a persistent effect even after a reset or a reconfiguration since the device is permanently damaged by the radiation. Non-destructive SEEs due to radiations are commonly called Single Event Upsets (SEUs), modeled as bit flips. Particular types of SEUs are Single Event Disturbs (SEDs), momentary voltage excursions at nodes, and Single Event Functional Interrupts (SEFIs), interrupts leading to temporary non-functionality of the affected device by involving power-on-reset, Select map port or JTAG port. SEDs are treated as general SEUs, whereas SEFIs are not taken into account in this thesis since they are negligible in occurrence [15]. Other SEEs have the potential to destroy the device. Destructive SEEs are Single Event Latchups (SELs), energy from a charged particle leading to an excessive supply power. They are not taken into account in this work as test reports on SRAM-based FPGAs reveal that no SEL was observed during the experiments, up to the maximum tested Linear Energy Transfer of tens of $MeV \cdot cm^2/mg$ [16, 17], that is sufficient for the considered applications.

Also TID effects have the potential to destroy the device. TID defines the total sum of radiations hitting the device. The unit of measurement in the International System of Units is the Gray (Gy), defined by the amount of 1 Joule of energy deposited per kilogram, however the deprecated unit rad (radiation absorbed dose) is still frequently used [14].

2.3.2. Aging effects

Device aging is the other analyzed cause, an important aspect for long-lasting space missions, where system maintenance or substitution is difficult. Aging causes the following faults with destructive effect [18, 19]:

- *Time Dependent Dielectric Breakdown* (TDDB), that is a breakdown caused by charge flow through the oxide,
- *Electromigration* (EM), that is a development of voids in metal lines due to heavy current densities over a period of time,
- *Hot-Carrier Effects* (HCE), that are interface traps affecting the I-V characteristics of the transistors, and
- *Negative Bias Temperature Instability* (NBTI), that is the degradation dependent on the time a PMOS transistor is stressed in the circuit.

In the field of FPGAs, these faults are just being recently addressed, also due to their rareness.

2.3.3. Adopted fault classification

We identify two categories of faults based on the possibility to recover from them by reconfiguration; *recoverable* and *non-recoverable* faults. Recoverable faults can be mitigated by reconfiguring the system, and possibly only the faulty sub-system portion, with the same configuration used before fault occurrence. Non-recoverable faults permanently compromise part or all of the device, such that further use of the corrupted portion of the device must be avoided and the logic hosted must be moved in a different location. The identification of the type of fault occurred on the device is thus fundamental to apply the suitable recovery strategy.

To summarize, recoverable faults are the ones caused by radiations without a destructive effect, whereas non-recoverable faults are those caused by radiations with destructive effect, radiations' accumulation and device aging. Figure 2.1 shows such fault classification based on the analyzed causes and effects.

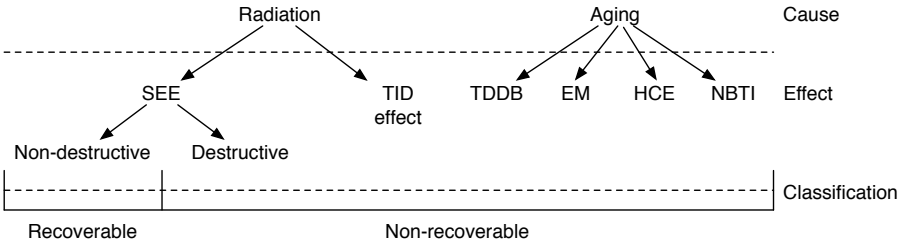


Figure 2.1.: Cause-based fault classification in the space environment.

2.4. Chapter summary

This chapter has presented the motivations that lead to the proposed work, and has introduced the background elements useful to set the basis for understanding the rest of the thesis. The adopted fault model has been described, and faults have been characterized also by referring to the selected working scenario, namely the space environment. The adopted single fault assumption and the two identified categories of faults, recoverable and non-recoverable, constitute the basis of the proposed design methodology, described in the next chapter.

3. Proposed reliability-aware design methodology

The wide use of FPGAs in critical scenarios, such as long-term space missions, leads to the necessity of design methodologies for the realization of systems able to autonomously cope with the occurrence of both recoverable and non-recoverable faults. Indeed, the scenario is even more complex when systems of considerable size are taken into account, requiring the use of several FPGAs at once. However, no complete design methodology handling all the peculiar issues has been proposed yet, a gap we aim at filling with our work. More precisely, in this chapter, we introduce a methodology for the design of autonomous fault tolerant systems implemented on multi SRAM-based FPGAs, with the final objective of increasing the system's lifetime and availability.

The chapter is structured as follows: Section 3.1 presents the proposed design methodology. Section 3.2 describes the devised reliable system and Section 3.3 shows the behavior of the proposed system in various fault scenarios. Finally, Section 3.4 draws the chapter summary.

3.1. Reliability-aware design methodology

The proposed reliability-aware methodology realizes autonomous fault tolerant systems implemented on multi-FPGA platforms, with the final objective of exploiting commercial SRAM-based FPGAs for mission-critical applications. While some aspects can be taken from previous solutions available in literature, several open issues for the proposed methodology exist and have been investigated. They can be summarized as follows:

- *Reliability-aware architecture definition.* First of all, it is necessary to establish how to exploit the available SRAM-based FPGAs to build the autonomous fault tolerant system. The architecture implementing the system must be defined; the communication between FPGAs for data exchanging and configuration must be identified and the controller solution must be selected.

- *Reliability-aware circuit hardening and distribution.* The nominal application circuit must be hardened and distributed among the FPGAs available on the platform. A set of metrics guiding in the definition of the reliable solution must be selected, and a strategy for identifying the most promising solution must be defined.

- *Fault classification strategy and Reconfiguration Controller design.* It is necessary to define a policy to classify the faults occurred in the system as recoverable or non-recoverable, based on the fact that the portion of the device is not actually compromised and thus will not properly work. Furthermore, it is necessary to design the engine in charge of implementing the recovery strategy. The controller must be able to monitor a parametric number of error signals, generated by a not known a-priori number of other FPGAs in the architecture, and, based on the identified type of fault, it must perform the suitable recovery action.

The proposed methodology aims to cope with these open points. It is composed of the following main activities, each of them trying to fix one of the described issues, as shown in Figure 3.1:

- *Architecture definition.* This activity defines and characterizes the autonomous fault tolerant system. It identifies the platform topology; given the available FPGAs, it establishes how they are connected to each other for communication and configuration. Moreover, it defines the controller solution, that could be centralized on a single device or distributed among the available FPGAs.

- *Design space exploration.* This is the core of the proposed methodology; this activity is devoted to the hardening of the nominal circuit on the multi-FPGA platform. As the name suggests, we are not just pursuing *a* working solution, rather the one offering the most convenient trade-off with respect to the designer's selected metrics. The activity is composed of three main steps: i) *hardening*, that selects and applies fault detection/tolerance techniques, ii) *partitioning*, that distributes the obtained reliable circuit among the available FPGAs, and iii) *floorplanning*, that positions each sub-circuit within the related device. A design space exploration is performed to identify the most promising solution according to a selected set of metrics.

- *Fault management definition.* This activity identifies how to classify the faults and cope with their occurrence. It defines the con-

troller engine, implementing the fault classification and recovery strategies.

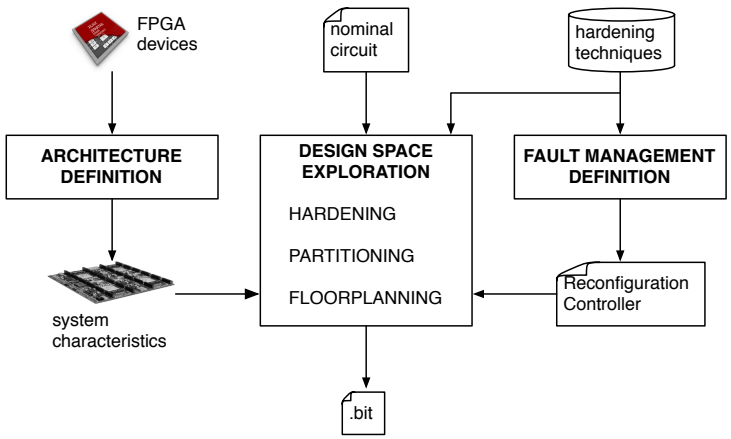


Figure 3.1.: Proposed reliability-aware design methodology.

The next section deals with the first activity of the methodology, defining and characterizing the proposed architecture.

3.2. Proposed autonomous fault tolerant system

The devised platform is composed of multiple SRAM-based FPGAs connected to each other in a mesh topology, i.e., each FPGA is connected to more than one FPGA and they can all connect to each other via multiple hops [20]. In the general situation, the complete nominal, not hardened, circuit implementing the actual application is distributed onto the platform. When fault tolerance properties are required, a circuit hardening approach based on a hybrid strategy is adopted; the circuit is hardened by means of reliability techniques exploiting, for example, space redundancy, and a reconfiguration of the devices is used to mitigate fault effects (e.g., as proposed in [11]). In this perspective, each FPGA hosts

- a hardened portion of the entire circuit, organized in *independently recoverable areas* (IRAs [1]) that detect, mask/tolerate, and signal the occurrence of a fault, and

- a *Reconfiguration Controller*, that is the engine in charge of monitoring the error signals to trigger, when needed, the reconfiguration of the faulty part of the circuit.

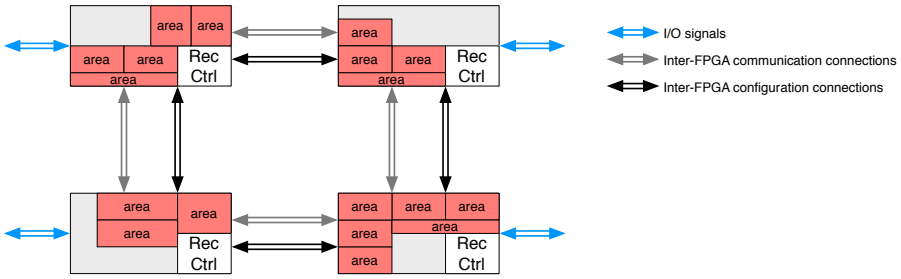


Figure 3.2.: Proposed system architecture.

An overview of the system architecture is shown in Figure 3.2.

The circuit is designed to provide fault detection and localization information, by generating error signals to allow the detection of faults. Fault detection and tolerance techniques, e.g., by means of spatial redundancy, are used to identify the occurrence of the fault and to localize the corrupted portion of the FPGA, the independently recoverable area. Thus, we envision the circuit to be partitioned into areas, each one reconfigurable independently by the remaining part of the device upon fault occurrence, as shown in Figure 3.3. In order to avoid that an area is much more sensible than the others to faults, we assume that the overall distribution of faults is uniform for all areas in the FPGA, which means that all areas should be similar in terms of size and used resources. As for the fault-error relation, it is also assumed that it does not sensibly change throughout the entire circuit/device.

The areas' error signals are received by the Reconfiguration Controller hosted on the neighbor FPGA. The controller discriminates between recoverable and non-recoverable faults and, based on the identified type, performs the suitable recovery action. Should the fault be considered as recoverable, the device is partially reconfigured by reloading the bitstream portion related to the faulty area, otherwise the faulty area becomes tagged as "unusable" and is relocated to a spare region reserved on purpose on the FPGA. Indeed, before performing relocation, the spare region is checked by means of software structural test.

An overview of the described system is reported in Figure 3.4, where the global hardened multi-FPGA architecture is depicted (Figure 3.4(a)) together with the detail of the reconfiguration approach (Figure 3.4(b)). In case of a fault of an area on $\texttt{FPGA}_{i+1}$, the Reconfiguration Controller hosted on $\texttt{FPGA}_i$ detects the anomaly, as shown in Figure 3.4 (a). The fault is classified as recoverable or non-recoverable and, based on the type of fault, the Reconfiguration Controller on $\texttt{FPGA}_i$ performs the reconfiguration using the correct bitstream, as shown in Figure 3.4 (b); if the

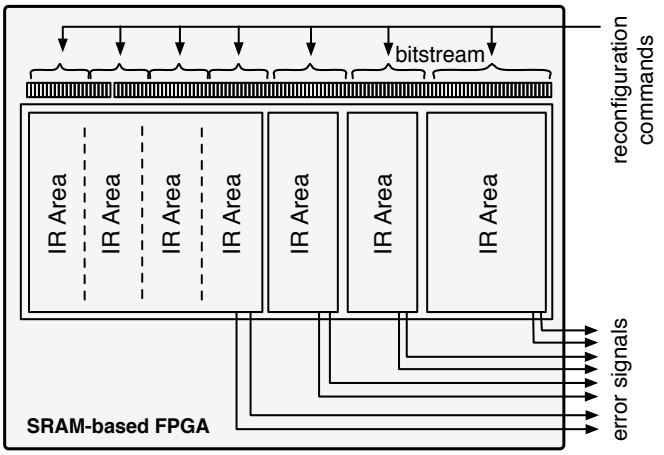


Figure 3.3.: Detail of the hardened circuit, composed of independently recoverable areas.

fault is recoverable, the functionality is restored by performing partial reconfiguration, otherwise a relocation of the functionality is performed, moving the functionality on a non-faulty device region.

Indeed, it is necessary to prevent erroneous reconfigurations by detecting faults affecting the controller, hence providing a reliable implementation of the engine. When the Reconfiguration Controller signals the presence of a fault in itself, it is handled as an area and reconfigured by another controller hosted onto a different FPGA. If implemented onto a single FPGA platform, the system can work only until an error is not detected in the Reconfiguration Controller. Multi-FPGA platforms are taken into account also for this reason (besides the large amount of available resources) and the Reconfiguration Controller has been distributed on the available devices.

Thus, rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover from faults, we have envisioned a distributed solution, where the Reconfiguration Controller hosted on an FPGA is in charge of reconfiguring the areas hosted on a neighbor FPGA. The aim is to achieve a higher level of reliability in the overall system, trying to avoid the single point of failure characterizing the centralized solution and requiring to be implemented onto a particular device (e.g., an ASIC or an antifuse-based FPGA, as in [8]). In the next section, the behavior of the proposed reliable system in various fault scenarios is presented.

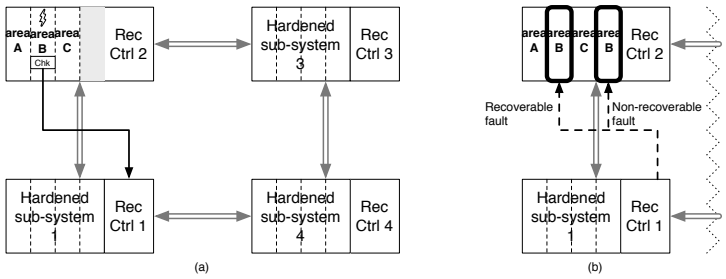


Figure 3.4.: Target scenario: (a) hardened multi-FPGA architecture and (b) proposed fault management strategy.

3.3. Additional fault scenarios

The proposed methodology implements a reliable system on multi-FPGA platform with the final objective of increasing its lifetime and availability. As described in the previous chapter, in designing the system, the single fault assumption is adopted, implying that faults occur one at a time and the time between the occurrence of two subsequent faults is long enough to allow the detection of the first fault before the second one occurs. However, we hereby consider other possible fault scenarios and discuss how the proposed methodology behaves.

- *Multiple faults.* The proposed methodology can also deal with multiple faults, provided they do not lead to the failure of fault detection techniques or they occur in independent portions of the entire system. We envision two scenarios, where multiple faults occur i) on the same single FPGA or ii) on different devices. In the former case, the overall system would be able to cope with this situation, and the designed controller would manage the concurrent faults in different areas. Nevertheless, it is worth noting that the proposed Reconfiguration Controller and the implemented fault classification algorithm, described in the next chapters, have been designed based on the single fault assumption. Therefore, we expect tolerable glitches in the classification possibly leading to delays in tolerating a non-recoverable fault, as discussed in the upcoming chapter. In the latter case, faults occurred on different FPGAs are recovered by the controllers in charge of monitoring the devices, preventing a failure of the entire system. A failure could occur when no recovery action can be performed due to faults contemporarily affecting all controllers; indeed, this is a rare, negligible situation.

- *Failure of the reconfiguration process.* An error can occur during the reconfiguration process, triggered by the occurrence of a fault in the monitored FPGA. This can happen because of i) a fault in the Reconfiguration Controller or ii) a fault in the inter-FPGA communication. In the former case, the reconfiguration is blocked as soon as an erroneous behavior of the Reconfiguration Controller is observed and the fault is recovered by the controller on the neighbor FPGA. In the latter case, no recovery action can be performed for the faulty FPGA and the system gradually degrades.

To conclude, although the proposed methodology is based on the traditionally adopted single fault assumption, the obtained reliable system can also deal with other more complex and not frequent fault scenarios.

3.4. Chapter summary

In this chapter, we have introduced the methodology we devised for the design of autonomous fault-tolerant FPGA-based systems. We exploit multi-FPGA platforms to build reliable systems able to autonomously cope with both recoverable and non-recoverable faults, with the final objective of increasing the system's lifetime. Differently from the past approaches, rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover from faults, we have envisioned a distributed controller solution. Although the methodology considers the traditionally adopted single fault assumption, the obtained solution can also deal with other, infrequent fault scenarios.

In the rest of the thesis, the methodology's activities are described in detail. Figure 3.5 provides an overview of the methodology, with reference to the chapters and publications where the main topics are dealt with (refer to Chapter 1 for the list of publications). The architecture definition activity has already been presented in Section 3.2. The next chapter introduces the proposed design flow for obtaining the reliable system.

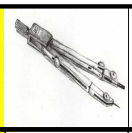
			Chapter	Publication
Architecture definition		Autonomous fault tolerant system	3	[DFT'10]
Design space exploration		Hardening	4	[DATE'12]
		Partitioning		[DFT'11]
		Floorplanning		[FPL'11]
Fault management definition		Fault classification algorithm	5	[ESL'10], [IOLTS'11]
		Reconfiguration Controller	6	[DFT'10]

Figure 3.5.: Methodology overview with reference to the chapters and the publications where the main topics are dealt with.

4. Design flow and framework

The methodology introduced in the previous chapter aims at realizing systems able to autonomously cope with the occurrence of both recoverable and non-recoverable faults. An autonomous fault tolerant multi-FPGA system has been proposed, and all the aspects to deal with for its implementation have been described. In this chapter, a flow for designing such fault tolerant systems is introduced, as well as the developed prototype framework implementing the flow, in order to automate as much as possible the design, hardening, and implementation.

The chapter is structured as follows: Section 4.1 presents the proposed design flow. Section 4.2 describes the multi-FPGA platform model adopted in the flow's activities and the subsequent sections focus on the flow's design space exploration tasks; Section 4.3 presents the preliminary partitioning of the nominal circuit among the available devices, Section 4.4 describes the circuit hardening, Section 4.5 reports the validation of the obtained solution, and Section 4.6 describes the positioning of the circuit areas within each device. Section 4.7 presents the prototype framework implementing the design flow. Section 4.8 proposes an alternative design flow, building the fault tolerant system and identifying the suitable platform based on the number of non-recoverable faults that must be tolerated, whereas the flow previously described works on a given platform. Finally, Section 4.9 draws the chapter summary.

4.1. Design flow

The proposed design flow identifies a suitable reliable implementation of the considered circuit and builds the overall fault tolerant system presented in the previous chapter, as shown in Figure 4.1.

The nominal application circuit, modeled in terms of a structural description containing a set of components interconnected with each other, is taken in input by the *synthesis and parsing* tasks. The synthesis activity derives the implementation costs of each component. The parsing activity builds a more agile representation of the circuit based on graph, annotated with components' costs and characteristics, derived by the preliminary analysis activity.

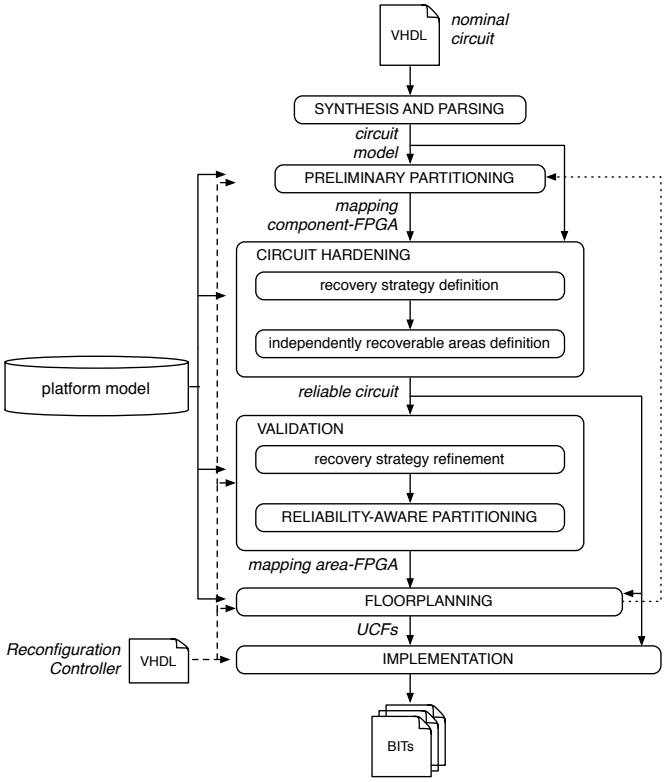


Figure 4.1.: Proposed design flow.

The costs and the model defined by analyzing the nominal circuit are taken in input by the *preliminary partitioning* task, that distributes the circuit components among the FPGAs available on the platform, by achieving a uniform distribution and minimizing inter-FPGA communication. The output of the partitioning problem is a mapping, specifying for each component its hosting on one of the devices.

For each FPGA, the *circuit hardening* task performs the application of fault tolerance techniques to the sub-circuit hosted on the device, creating the independently recoverable areas. The task is, in turn, composed of two activities; recovery strategy definition and independently recoverable areas definition. The *recovery strategy definition* activity identifies the reliable system's parameters involved in the recovery strategy from non-recoverable faults. For each FPGA available on the platforms, it defines i) an estimate of the maximum number of non-recoverable faults the system can autonomously recover from without any external action, and ii) the maximum number of independently recoverable areas that can be hosted on the device. Based on these parameters, the *independently*

recoverable areas definition activity partitions the circuit components into groups, each one hardened and mapped on independently recoverable areas (IRAs [1]). Hardening is, at present, achieved by means of Triple Modular Redundancy (TMR). However, it is possible to take into account other techniques.

A *validation* task aims at improving the obtained solution. It is, in turn, composed of two activities; recovery strategy refinement and reliability-aware partitioning. Considering the obtained hardened circuit, the *recovery strategy refinement* activity can provide a more accurate valuation of the number of tolerated non-recoverable faults. This value is taken in input by the *reliability-aware partitioning* task, that re-distributes the independently recoverable areas among the available FPGAs by taking into account also possible recovery actions needed during the system's lifetime. It reserves a suitable spare region for relocations in case of faults physically damaging the device. The output of this partitioning problem is a mapping, specifying for each area its hosting on one of the devices.

The *floorplanning* task positions the independently recoverable areas inside the related FPGA. It identifies a suitable placement for the hardened circuit and the recovery actions.

Finally, the *implementation* phase integrates the Reconfiguration Controllers, one for each FPGA, with the reliable circuit, building the proposed overall reliable system. In this phase, all the bitstreams of the identified solution are generated.

In the following, more details about the tasks composing the design space exploration are provided. Indeed, in order to define a feasible solution, an accurate model of the multi-FPGA platform must be adopted. In the next section, the proposed model is introduced.

4.2. Multi-FPGA platform model

We consider a platform composed of identical FPGAs, $\mathtt{d} \in \mathtt{D}$, where $\mathtt{D}$ is the set of devices. Given the assumption of having identical devices (together with the assumption of having a uniform distribution of the system on the board, guaranteed by the partitioning approach), in case a faulty device becomes unavailable, another FPGA could be possibly exploited to substitute it. Nevertheless, the proposed model can be easily extended to represent different devices. The device information made available to the flow must specify both the existing resources and the reconfiguration properties of the considered FPGA.

FPGAs are characterized by several types of discrete and distributed

resources, spread across the device. They are composed of three main building blocks and have additional resources directly embedded on the die. The main building blocks are i) Configurable Logic Blocks (CLBs), composed of slices, ii) Input/Output blocks (IOBs), and iii) routing resources. By opportunely configuring them, the design functionality is achieved. Additional resources are, for instance, RAM blocks (BRAMs), DSPs and multipliers. They allow designers to take advantage of specific hardware without the need to implement it using the CLBs, with an improvement in performance. Each FPGA is characterized by the number of resources hosted on it, $\texttt{dev\_res}_r$, $\texttt{r} \in \texttt{R = \{slice, bram, dsp\}}$, in terms of slices, BRAMs, and DSPs. Xilinx FPGAs are considered for this model, although it is worth noting that `R` can be easily modified or extended to consider other types of resources. The various types of resources are organized in columns distributed either uniformly or not. FPGA families, even the less recent ones (e. g., Xilinx Spartan-3 [21] or Virtex-II [22]), have heterogeneous resources. In addition more recent families, such as Xilinx Virtex-4 [23] and Virtex-5 [24], adopt a not uniform distribution. For each resource type, we extract the characterizing parameters, namely the number of columns, the number of resources within each column, and the position of each column. Since all devices share a common matrix-like structure, the FPGA is shaped as a grid of slices with columns constituted by different types of resources.

Another important feature of FPGAs' architecture is their reconfiguration capabilities. In particular, it is important to identify the reconfiguration dimension, that can be mono-dimensional (1D) or bi-dimensional (2D). In 1D reconfiguration, the part of resources affected by the process, called reconfigurable area, always spans the whole height of the device ($\texttt{height}_{dev}$, expressed in terms of slices), exploiting a column organization, while 2D reconfiguration allows the specification of the reconfigurable area in a 2D space, organized in columns and rows. In both cases the smallest addressable configurable unit is the frame, a vertical line of resources, and the reconfiguration process can span the whole width of the device ($\texttt{width}_{dev}$, expressed in terms of slices). The distinction between the two reconfiguration schemes is based on the number of rows of frames; the 1D case is characterized by a single row of frames, while the 2D case supports multiple rows. Less recent FPGA families, such as Xilinx Virtex-II, offer 1D reconfiguration, while more recent ones, such as Virtex-4 and Virtex-5, support the 2D scheme.

A schematic FPGA structure is shown in Figure 4.2. The proposed model defines an accurate specification of the FPGAs characteristics, allowing the design flow's activities to identify a feasible candidate solution for the reliable system.

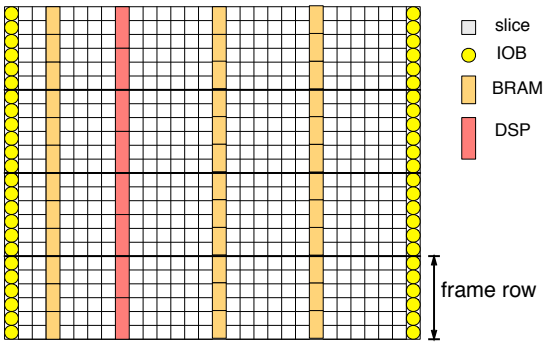


Figure 4.2.: A schematic FPGA structure.

The inter-FPGA communication is implemented through dedicated wires. We consider $\texttt{dev\_wires}_{d_1,d_2}$, $\texttt{d}_1$,$\texttt{d}_2 \in \texttt{D}$, that is the number of wires between devices $\texttt{d}_1$ and $\texttt{d}_2$, and `tot_wires`, that is the number of total external wires on the platform. The platform topology is modeled by specifying the devices' adjacency and the communication paths between non adjacent devices; $\texttt{direct\_comm}_{d_1,d_2}$, $\texttt{d}_1$,$\texttt{d}_2 \in \texttt{D}$, equals to 1 if devices $\texttt{d}_1$ and $\texttt{d}_2$ are adjacent, 0 otherwise, and $\texttt{non\_direct\_comm}_{d_1,d_2,d_3}$, $\texttt{d}_1$,$\texttt{d}_2$,$\texttt{d}_3 \in \texttt{D}$, equals to 1 if device $\texttt{d}_1$ allows the communication between devices $\texttt{d}_2$ and $\texttt{d}_3$, 0 otherwise. In this model, topologies implemented over dedicated wires (e.g., mesh or complete-graph topologies) are taken into account, and the communication paths between non-adjacent devices are defined by the designer.

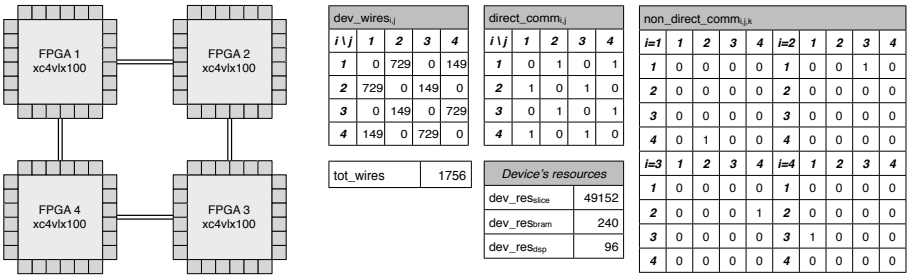
Table 4.1 reports the parameters of the proposed model and Figure 4.3 summarizes them by characterizing a commercial multi-FPGA platform, in particular a Synopsis HAPS-34 board [25]. It is worth noting that the communication paths between non-adjacent devices are defined by the designer; for example, FPGA 2 allows the communication from FPGA 1 to FPGA 3, whereas FPGA 4 allows the communication from FPGA 3 to FPGA 1. The proposed platform model is used by the design flow's tasks, described in the following.

4.3. Preliminary partitioning

The circuit components are taken in input by the partitioning task, that distributes them among the available FPGAs. Each component is characterized by its resource requirements, $\texttt{comp}_{c,r}$, $\texttt{c} \in \texttt{C}$, $\texttt{r} \in \texttt{R}$, that is the number of resources of type `r` required by component `c`. The communication between the components is modeled by specifying the number of

Table 4.1.: Multi-FPGA platform parameters.

Parameter	Description
D	Set of devices
R	Set of resources
$\texttt{dev\_res}_r$	Number of resources of type r hosted on the FPGA
$\texttt{height}_{dev}$	Height of the FPGA in terms of slices
$\texttt{width}_{dev}$	Width of the FPGA in terms of slices
$\texttt{dev\_wires}_{d_1,d_2}$	Number of wires between FPGAs d_1 and d_2
tot_wires	Number of total external wires on the multi-FPGA platform
$\texttt{direct\_comm}_{d_1,d_2}$	Binary parameter equal to 1 if FPGAs d_1 and d_2 are adjacent
$\texttt{non\_direct\_comm}_{d_1,d_2,d_3}$	Binary parameter equal to 1 if d_1 allows the communication between d_2 and d_3



dev_wires$_{i,j}$

i \ j	1	2	3	4
1	0	729	0	149
2	729	0	149	0
3	0	149	0	729
4	149	0	729	0

tot_wires	1756

direct_comm$_{i,j}$

i \ j	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

Device's resources	
$\texttt{dev\_res}_{slice}$	49152
$\texttt{dev\_res}_{bram}$	240
$\texttt{dev\_res}_{dsp}$	96

non_direct_comm$_{i,j,k}$

i=1	1	2	3	4	i=2	1	2	3	4
1	0	0	0	0	1	0	0	1	0
2	0	0	0	0	2	0	0	0	0
3	0	0	0	0	3	0	0	0	0
4	0	1	0	0	4	0	0	0	0
i=3	**1**	**2**	**3**	**4**	**i=4**	**1**	**2**	**3**	**4**
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	2	0	0	0	0
3	0	0	0	0	3	1	0	0	0
4	0	0	0	0	4	0	0	0	0

Figure 4.3.: Multi-FPGA platform model.

wires connecting each pair of components ($\texttt{comp\_wires}_{c_1,c_2}$, $c_1,c_2 \in C$) and the throughput between components ($\texttt{comp\_comm}_{c_1,c_2}$, $c_1,c_2 \in C$). Finally, the designer can specify his/her requirements about the placement of the components; $\texttt{same\_dev}_{c_1,c_2}$, $c_1,c_2 \in C$, equals to 1 if components c_1 and c_2 must be placed on the same device, 0 otherwise. In this way, the designer can constrain the placement of components characterized by critical communication, for example components exchanging data frequently. The proposed partitioner considers the designer's requirements as constraints, but it can be easily modified to handle them as preferences. Note that we do not model connections shared between components, since every wire is considered as a dedicated connection. Hence, we overestimate the number of wires, but in this way we also increase the weight of the connections used by more components. However, the designer can require that componets sharing connections are placed on the same FPGA.

The problem of partitioning a circuit onto a multi-FPGA platform is a multi-objective problem, since various metrics are considered when distributing the circuit among the devices. We consider the following set of metrics:

- *Distribution uniformity.* When considering circuit partitioning, it is generally desirable to divide the circuit into portions of roughly

equal sizes. This is even more important when reliability is taken into account, to have balanced ratio between used and spare fabric.

- *Number of external wires.* The number of inter-FPGA connections must be minimized both for the performance degradation with respect to intra-FPGA wires and for the limited number of I/O pins reserved for inter-FPGA communication.

- *External communication.* For the performance degradation characterizing inter-FPGA connections, components with highest communication requirements should be placed on the same device.

The output of the partitioning problem is a mapping, specifying for each component its hosting on one of the devices. This is modeled by the decision variable $\mathtt{x}_{c,d}$, that equals to 1 if component `c` is hosted on device `d`, 0 otherwise. Furthermore, to account for communication, we introduce the binary variables $\mathtt{y}_{c1,d1,c2,d2}$, equal to 1 if component `c1` is hosted on device `d1` and component `c2` is hosted on device `d2`.

The partitioning problem includes constraints associated with components placement, system distribution, external wires, and external communication, listed in Table 4.5.

Components placement constraints. The partitioner must place each component on a device, as defined by Constraint C1. Moreover, the designer's requirements related to the placement of componets on the same device must be fulfilled. Constraint C5 allows to meet these requirements by exploiting the variables $\mathtt{y}_{c1,d1,c2,d2}$, that identify the placement of each couple of components. Constraints C2, C3, and C4 define the valid values of the variables $\mathtt{y}_{c1,d1,c2,d2}$, by referring to the variables $\mathtt{x}_{c,d}$; Constraints C2 and C3 define the minimum bound of $\mathtt{y}_{c1,d1,c2,d2}$ (if $\mathtt{x}_{c1,d1}$ or $\mathtt{x}_{c2,d2}$ equal to 0, then $\mathtt{y}_{c1,d1,c2,d2}$ equals to 0), whereas Constraint C4 identifies the maximum one (if both $\mathtt{x}_{c1,d1}$ and $\mathtt{x}_{c2,d2}$ equal to 1, then $\mathtt{y}_{c1,d1,c2,d2}$ equals to 1).

System distribution constraints. The system must be distributed uniformly among the available devices. The resources occupation on each device, computed as the sum of the resources required by the Reconfiguration Controller and the components hosted on the device, as defined by Constraint C6, can not be greater than the available resources, as constrained by C7. The distribution of the system among the available devices is uniform when the gap between the maximum and minimum resources occupations among the devices is null. The maximum and minimum occupations of each type of resources are constrained by C8 and C9, and the difference between them defines the gap, as computed by Constraint C10.

External wires constraints. The external wires are the connections between components hosted on different devices, by considering both adjacent and non adjacent devices. The number of external wires between each couple of devices is computed by Constraint 11, by summing the external connections between the components hosted on the devices. This number is exploited to compute the number of external wires for the communication between adjacent and non-adjacent devices, as defined by Constraints C12 and C13, respectively. The number of external wires required for inter-FPGA communication can not be higher than the available wires between each couple of devices, as constrained by C14, and for the overall platform it is computed by Constraint C15.

External communication constraints. The computation of the external communication exploits the computation of the external communication of each component on each device, defined by Constraint C16, and the external communication between each couple of devices, defined by Constraint C17. The external communication of each device is computed by Constraint C18, by summing the external communication of the components hosted on the device and the external communication between non-adjacent devices through the considered one. Finally, the total external communication on the overall platform is computed by Constraint C19, by summing the external communication of each device.

The partitioning problem includes only binary and continuous variables and linear constraints, hence it is a MILP problem. It is a multi-objective problem, with the following objective function:

$$min(w_{gap} \cdot \sum_{r \in R} \frac{Gap_r}{dev\_res_r} + w_{wires} \cdot \frac{Wires}{tot\_wires} + w_{comm} \cdot \frac{Comm}{\sum_{c1 \in C} \sum_{c2 \in C} comm_{c1,c2}})$$

where w_{gap}, w_{wires}, and w_{comm} are weights assigned by the designer.

4.4. Circuit hardening

This section describes the strategy adopted for hardening the circuit. The traditional fault tolerance technique TMR is coupled with the FPGA dynamic reconfiguration property to achieve error mitigation capability. This is applied to single components or group of components of the circuit and mapped on areas that, in case of fault, are recoverable by reconfiguration independently from the others. Indeed, the higher the number of areas the finer the fault detection granularity, but, at the same time, a memory with higher capacity to store the recovery bitstreams is required.

Table 4.2.: Preliminary partitioner constraints.

Components placement	
C1	$\sum_{d \in D} x_{c,d} = 1, \forall c \in C$
C2	$y_{c1,d1,c2,d2} \leq x_{c1,d1}, \forall c1, c2 \in C, d1, d2 \in D$
C3	$y_{c1,d1,c2,d2} \leq x_{c2,d2}, \forall c1, c2 \in C, d1, d2 \in D$
C4	$y_{c1,d1,c2,d2} \geq x_{c1,d1} + x_{c2,d2} - 1, \forall c1, c2 \in C, d1, d2 \in D$
C5	$\sum_{d \in D} y_{c1,d,c2,d} \geq same\_dev_{c1,c2}, \forall c1, c2 \in C$
System distribution	
C6	$dev\_occupation_{d,r} = \sum_{c \in C} comp_{c,r} \cdot x_{c,d} + rec_r, \forall d \in D, \forall r \in R$
C7	$dev\_occupation_{d,r} \leq dev\_res_{d,r}, \forall d \in D, \forall r \in R$
C8	$max\_dev\_occupation_r \geq dev\_occupation_{d,r}, \forall d \in D, \forall r \in R$
C9	$min\_dev\_occupation_r \leq dev\_occupation_{d,r}, \forall d \in D, \forall r \in R$
C10	$Gap_r = max\_dev\_occupation_r - min\_dev\_occupation_r, \forall r \in R$
External wires	
C11	$d\_to\_d\_wires_{d1,d2} = \sum_{c1 \in C} \sum_{c2 \in C} comp\_wires_{c1,c2} \cdot y_{c1,d1,c2,d2}, \forall d1, d2 \in D$
C12	$dev\_dir\_wires_{d1,d2} = d\_to\_d\_wires_{d1,d2} \cdot direct\_comm_{d1,d2}, \forall d1, d2 \in D$
C13	$dev\_non\_dir\_wires_{d1,d2} =$
	$\sum_{d \in D} (d\_to\_d\_wires_{d1,d} \cdot non\_direct\_comm_{d2,d1,d} + d\_to\_d\_wires_{d,d2} \cdot non\_direct\_comm_{d1,d,d2}) +$
	$\sum_{d3 \in D} \sum_{d4 \in D} d\_to\_d\_wires_{d3,d4} \cdot non\_direct\_comm_{d1,d3,d4} \cdot non\_direct\_comm_{d2,d3,d4}, \forall d1, d2 \in D$
C14	$dev\_dir\_wires_{d1,d2} + dev\_dir\_wires_{d2,d1} + dev\_non\_dir\_wires_{d1,d2} +$
	$dev\_non\_dir\_wires_{d2,d1} \leq dev\_wires_{d1,d2}, \forall d1, d2 \in D$
C15	$Wires = \sum_{d1 \in D} \sum_{d2 \in D} dev\_dir\_wires_{d1,d2} + dev\_non\_dir\_wires_{d1,d2}$
External communication	
C16	$ext\_comp\_comm_{c,d} = \sum_{c1 \in C} comp\_comm_{c,c1} \cdot x_{c,d} - comp\_comm_{c,c1} \cdot y_{c,d,c1,d}, \forall c \in C, \forall d \in D$
C17	$dev\_to\_dev\_comm_{d1,d2} = \sum_{c1 \in C} \sum_{c2 \in C} comp\_comm_{c1,c2} \cdot y_{c1,d1,c2,d2}, \forall d1, d2 \in D$
C18	$dev\_comm_d = \sum_{c \in C} ext\_comp\_comm_{c,d} + \sum_{d1 \in D} \sum_{d2 \in D} dev\_to\_dev\_comm_{d1,d2} \cdot non\_direct\_comm_{d,d1,d2}, \forall d \in D$
C19	$Comm = \sum_{d \in D} dev\_comm_d$

The circuit hardening task defines the independently recoverable areas taking into account these issues. It performs a design space exploration composed of two main activities, described in the following.

4.4.1. Recovery strategy definition

The first activity performed when hardening the circuit defines the parameters constraining the definition of the independently recoverable areas on the FPGAs. Such parameters are related to the adopted recovery strategy, described in the following.

When reconfiguring to cope with non-recoverable faults, a new configuration that replaces the faulty region in the FPGA with a spare one is loaded. The strategies for defining the recovery configurations can be classified into two categories:

- *On-line bitstream computation*, that dynamically creates the recovery configurations, as in [26] and [27], and
- *Off-line bitstream computation*, that creates the alternative configurations during the design phase, as in [28] and [29].

The former strategy means that only one bitstream is saved in memory and that, if needed, it is modified at runtime to assign the functionalities to the correct areas. Partial dynamic reconfiguration and bitstream relocation provided by Xilinx FPGAs [30] are used for defining on-line approaches. These approaches do not address the reliability issues related to the system's static region defined by the Xilinx design flow, that contains the global communication infrastructure and, possibly, the internal Reconfiguration Controller. Actually, a fault affecting the static region cannot be recovered and would affect the overall system. Moreover, the strategy would make the Reconfiguration Controller heavily dependent on the actual FPGA and it would consistently increase its complexity. For these reasons we have not considered on-line bitstream computation in our approach.

However, the other strategy, off-line bitstream computation, requires the storage of all the bitstreams that might be needed during the system's lifetime, thus a large memory is required. In fact, the approach consists of computing all the bitstreams that might be needed before deploying the system and, at runtime, only the correct bitstream is loaded. For each sequence of occurred non-recoverable faults and for each independently recoverable area the circuit is divided into, a bitstream must be stored in memory (refer to Figure 4.4). Indeed, it is necessary to maximize i) the number of non-recoverable faults the system can autonomously recover from without any external action, to improve the system's lifetime, and ii) the number of areas, both to allow fault detection at finer granularity and to reduce the average partial scrubbing time required for coping with recoverable faults. Also, it is required not to exceed the capacity of the bitstream memory. Thus, the following parameters related to the relocation strategy are defined:

- `#faults`, that is the number of non-recoverable faults for each FPGA the system can autonomously cope with, and
- `max_areas`, that is the maximum number of independently recoverable areas the sub-circuits on the FPGAs are divided into.

To compute `#faults`, a hardened version of the circuit is taken into account, partitioning each component in a different area and applying TMR. Then, the maximum number of tolerated non-recoverable faults for each device `d` ($\texttt{max\_faults}_d$) is the number of times the greatest area hosted on the device can be moved onto a not used, fault-free region. `#faults` is the minimum value among the identified numbers of tolerated non-recoverable faults:

$$\#faults = \min\left(max\_faults_d, \forall d \in D\right)$$

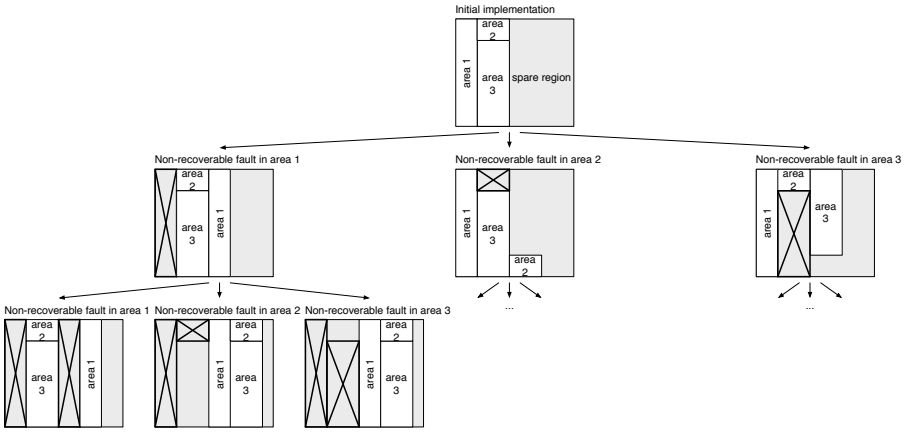


Figure 4.4.: The adopted recovery strategy for non-recoverable faults.

When considering multiple subsequent non-recoverable faults, a specific bitstream is required for each sequence of occurred faults. Therefore, assuming a given number of independently recoverable areas $\#\texttt{areas}_d$, equal to the number of components hosted on FPGA `d`, to tolerate `#faults` the following number of alternative bitstreams for each device has to be generated:

$$\#bitstreams_d = \sum_{i=0}^{\#faults} \#areas_d^i, \forall d \in D$$

It is worth noting that $\#\texttt{bitstreams}_d$ can be very high also for low values of `#faults`, hence too a large memory should be required. Thus, the number of areas `max_areas` is set by fulfilling the memory constraint:

$$max\_areas = \max\left(\#areas \mid \sum_{i=0}^{\#faults} \#areas^i \leq \frac{memory\_size}{bitstream\_size}\right)$$

being `memory_size` and `bitstream_size` the size of the memory and of the bitstream for the selected device, respectively. `max_areas` is considered when defining the independently recoverable areas, as described in the following.

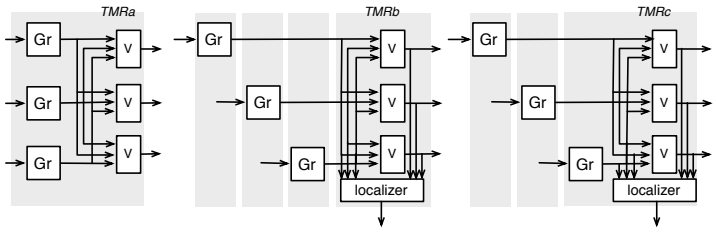


Figure 4.5.: TMR technique: different application schemas.

4.4.2. Independently recoverable areas definition

By considering the parameter `max_areas` defined in the previous activity, the independently recoverable areas composing the hardened circuit, described in Chapter 3, are defined. For each FPGA on the platform, the components of the sub-circuit hosted on the device are grouped and a hardening technique is applied on each identified group. We select TMR as fault tolerance technique and, borrowing from [1], we envision different strategies of applying it. Four different strategies of TMR application can be adopted on each group, containing one or more components, as shown in Figure 4.5:

- TMR on 1 area, mapping the whole system (replicas and voter) on a single area,
- TMR on 2 areas, mapping two replicas on an area and the third replica and the voter on another area,
- TMR on 3 areas, mapping each replica on a different area and the voter with one of the replicas, and
- TMR on 4 areas, mapping each replica and the voter on four different areas.

Each of these strategies is a hardening technique, that identifies the independently recoverable areas where the groups of components are mapped. Indeed, also other hardening techniques can be considered as well.

We recall that the circuit is modeled as a set of components, $\mathtt{c} \in \mathtt{C}$, interconnected with each other by wires; $\mathtt{comp\_wires}_{c_1,c_2}$ is the number of wires between components $\mathtt{c}_1$ and $\mathtt{c}_2$, and $\mathtt{comp\_out}_c$ is the number of output wires of component $\mathtt{c}$. Each component is characterized by its resource requirements, $\mathtt{comp\_res}_{c,r}$, that is the number of resources of type

r required by component c. The activity partitions the nominal circuit in groups, $g \in G$, where G is the set of groups, and applies a hardening technique, $t \in T$, where T is the set of techniques, to each of them. Each group is mapped on one or more areas, $a \in A$, where A is the set of areas. Thus, each area of the hardened circuit hosts one or more components of the nominal circuit and constitutes an independently recoverable area. The parameters used in the circuit hardening are reported in Table 4.3.

Table 4.3.: Parameters used in the circuit hardening.

Parameter	Description
C	Set of components
G	Set of groups
T	Set of hardening techniques
A	Set of areas
R	Set of resources
$\texttt{comp\_res}_{c,r}$	Number of resources of type r required by component c
$\texttt{comp\_wires}_{c_1,c_2}$	Number of wires between components c_1 and c_2
$\texttt{comp\_out}_{c}$	Number of output wires of component c

The definition of the independently recoverable areas hosted on each device has been modeled as a MILP problem, whose inputs are the nominal circuit and the parameter max_areas identified by the previous activity. The output is the hardened version of the circuit, described in terms of a mapping between the components and the hardened groups. This is modeled by the decision variables $x_{c,g,t}$, that equals to 1 if component c belongs to group g where technique t is applied, 0 otherwise. Moreover, to account for the number of identified groups and the communication between groups, the following binary variables are introduced; $y_{g,t}$, that equals to 1 if group g where technique t is applied hosts at least a component, and $z_{c_1c_2,g,t}$, that equals to 1 if components c_1 and c_2 are assigned to group g where technique t is applied. When solving the proposed problem, the following metrics have been considered:

- *Distribution uniformity*, that is generally desirable to divide the circuit into portions of roughly equal sizes, and is even more important when reliability is taken into account, to have balanced areas reserved for relocations.
- *Number of areas*, that must be maximized both to allow fault detection at finer granularity and to reduce the average partial scrubbing time required for coping with recoverable faults.
- *Number of wires between groups*, that must be minimized for reducing the resource requirement of the necessary TMR voters.

The problem of hardening the nominal circuit and partitioning it in independently recoverable areas includes constraints associated with groups definition, wires between groups, and resources distribution, listed in Table 4.5.

Groups definition constraints. The hardening process must assign each component to a group, as defined by Constraint C1. Moreover, each group hosting at least a component must be hardened by means of a technique. Constraint C4 allows to meet this requirement by exploiting the variables $\mathtt{y}_{g,t}$, that identify the groups hosting at least a component. Constraints C2 and C3 define the valid values of the variables $\mathtt{y}_{g,t}$, by identifying the maximum bound and the minimum one, respectively. Finally, the number of areas allocated for mapping the groups cannot be higher than `max_areas`, required by the relocation strategy, or lower than `min_areas`, possibly used to constrain the fault detection granularity, as defined by Constraints C6 and C7, respectively. Constraint C5 computes the number of allocated areas by taking into account the number of groups and the number of areas required by the applied technique ($\#\mathtt{areas\_req}_t$).

Wires between groups constraints. The number of connections between groups is identified by computing the number of wires between components hosted on different groups, as defined by Constraint C11. The variables $\mathtt{z}_{c_1,c_2,g,t}$ are exploited. Constraints C8 - C10 define the valid values of the variables $\mathtt{z}_{c_1,c_2,g,t}$, by identifying the maximum bound (Constraints C8 and C9) and the minimum one (Constraint C10).

Resources distribution constraints. The resource requirement of each area is computed by considering the occupation of the components hosted on it and the occupation of the voter, as defined by Constraint C12. To compute the occupation of the components hosted on the area, the parameters $\mathtt{w\_comp}_{t,a,r}$ are exploited, defining the number of components required by the applied technique on the area. To compute the occupation of the voter, depending on the number of voted wires, the parameters $\mathtt{w\_voter}_{t,a,r}$ are exploited, indicating the multiplication factor for wires. Indeed, the resource requirements of the areas can not exceed the resource availability on the FPGA, as defined by Constraint C13. When dedicated resources (e.g., BRAMs, DSPs) are not sufficient, it is possible to convert them in slices; such conversion is made for all the replicas of a module to have areas characterized by uniform distribution. Constraints C14 and C15 identify the area with the minimum resource requirement and the area with the maximum one, respectively. It is worth noting how the area with the minimum resource requirement is defined by considering only the areas hosting components; we exploit the parameter `M`, a enough big value that allows to avoid wrong values of $\mathtt{min}_r$ when con-

sidering areas without components. Finally, the gap between the largest area and the smallest one, $\texttt{gap}_r$, $\texttt{r} \in \texttt{R}$, needed to define the uniform distribution, is identified by Constraint C16.

Table 4.4.: Independently recoverable areas definition constraints.

Groups definition	
C1	$\sum_{g \in G} \sum_{t \in T} x_{c,g,t} = 1, \forall c \in C$
C2	$y_{g,t} \leq \sum_{c \in C} x_{c,g,t}, \forall g \in G, t \in T$
C3	$y_{g,t} \geq x_{c,g,t}, \forall c \in C, g \in G, t \in T$
C4	$\sum_{t \in T} y_{g,t} \leq 1, \forall g \in G$
C5	$\#areas = \sum_{g \in G} \sum_{t \in T} y_{g,t} \cdot \#areas\_req_t$
C6	$\#areas \leq max\_areas$
C7	$\#areas \geq min\_areas$
Wires between groups	
C8	$z_{c_1,c_2,g,t} \leq x_{c_1,g,t}, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C9	$z_{c_1,c_2,g,t} \leq x_{c_2,g,t}, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C10	$z_{c_1,c_2,g,t} \geq x_{c_1,g,t} + x_{c_2,g,t} - 1, \forall c_1 \in C, c_2 \in C, g \in G, t \in T$
C11	$group\_wires_{g,t} = \sum_{c_1 \in C} \sum_{c_2 \in C} comp\_wires_{c_1,c_2} \cdot (x_{c_1,g,t} - z_{c_1,c_2,g,t}) + \sum_{c \in C} comp\_out_c \cdot x_{c,g,t}, \forall g \in G, t \in T$
Resources distribution	
C12	$group\_res_{g,t,a,r} = \sum_{c \in C} comp\_res_{c,r} \cdot x_{c,g,t} \cdot w\_comp_{t,a,r} + group\_wires_{g,t} \cdot w\_voter_{t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C13	$\sum_{g \in G} \sum_{t \in T} \sum_{a \in A} group\_res_{g,t,a,r} \leq dev_r, \forall r \in R$
C14	$min_r \leq M \cdot (1 - y_{g,t} \cdot req_{t,a}) + group\_res_{g,t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C15	$max_r \geq group\_res_{g,t,a,r}, \forall g \in G, t \in T, a \in A, r \in R$
C16	$gap_r = max_r - min_r, \forall r \in R$

The objective function adopted in this problem is the following:

$$min(w_{res} \cdot \sum_{r \in R} \frac{gap_r}{dev_r} + w_{areas} \cdot (1 - \frac{\#areas}{max\_areas}) + w_{wires} \cdot \frac{\sum_{g \in G} \sum_{t \in T} comp\_wires_{g,t}}{\sum_{c_1 \in C} \sum_{c_2 \in C} comp\_wires_{c_1,c_2} + \sum_{c \in C} comp\_out_c})$$

being $\texttt{w}_{res}$, $\texttt{w}_{areas}$, and $\texttt{w}_{wires}$ designer-assigned weights. It is worth noting that the sum of the weights must equal to 1. An optimal solution is identified, achieving a circuit able to offer correct functionality even in presence of non-recoverable faults.

4.5. Validation

The validation task aims at improving the obtained reliable solution. It is composed of two activities, described in the following.

4.5.1. Recovery strategy refinement

The previous task has defined the independently recoverable areas hosted on each FPGA, taking into account the number of non-recoverable faults

tolerated on each device (`#faults`). Indeed, `#faults` is a preliminary estimate, evaluated by considering a hardening of the sub-circuits at component level. The recovery strategy refinement activity re-evaluates the number of tolerated non-recoverable faults, by taking into account the hardened circuit defined by the previous activity.

For each FPGA `d`, the maximum number of tolerated non-recoverable faults ($\texttt{max\_faults}_d$) is computed by considering the number of times the greatest independently recoverable area hosted on the device can be relocated on the spare region. Then, the constraint imposed by the memory capacity is taken into account and the actual number of non-recoverable faults tolerated by the system on each FPGA is defined as follows:

$$\#faults = \min\left(\#faults_d \mid \sum_{i=0}^{\#faults_d} \#areas_d^i \leq \frac{memory\_size}{bitstream\_size}, \forall d \in D\right)$$

where $\#\texttt{areas}_d$ is the number of independently recoverable areas hosted on FPGA `d` and $\#\texttt{faults}_d \leq \texttt{max\_faults}_d$ is the tolerated number of faults.

This value is considered by the subsequent reliability-aware partitioning activity, that re-distributes the circuit among the available FPGAs, also taking into account the recovery actions.

4.5.2. Reliability-aware partitioning

The independently recoverable areas defined in the previous task are taken in input by the reliability-aware partitioning task, that distributes them among the available FPGAs, with the aim of possibly improving the obtained solution. The reliability-aware partitioner is an extension of the one presented in Section 4.3, considering also the possible recovery actions needed during the system's lifetime.

In literature, the problem of partitioning hardware circuits has been widely investigated and various approaches have been proposed. Some contributions [31, 32, 33] partition design specifications at the behavioral level, providing limited details about the systems' implementation, which for FPGA platforms are quite relevant. Other approaches [34, 35] take into account only the routing between FPGAs, without considering the structure of the reconfigurable fabric, peculiar and heterogeneous in the most recent FPGA families. Therefore, those works adopting either a simplified homogeneous model of the resources (e.g., [36, 37, 38]) or a heterogeneous one but with a uniform distribution (e.g., [39]) can only be used in specific application scenarios, since with the detailed model of up-to-date devices would be inaccurate and would possibly lead to

unfeasible solutions. Finally, some contributions seek a cover of the circuit in terms of used devices (e.g., [40, 41, 42, 43, 44, 45, 46]), however these approaches do not model the platform's topology. Again, the approaches proposed in literature could be used as hints for a preliminary exploration of the solution space, but not as final methods to be introduced within an automatic design flow, which should always produces a feasible solution. Moreover, none of the approaches actually deals with the specifics of hardened systems, which have additional requirements in terms of resources and placement. Finally, while most of the contributions in literature present algorithms based on heuristics, we propose a MILP model that identifies the global optimal solution in an acceptable execution time. It is worth noting that, even if the approach has been designed to handle hardened systems, it is possible to use it also for nominal systems, with no fault management features.

The partitioner takes as input the independently recoverable areas composing the reliable circuit. Each area is characterized by its resource requirements, $\texttt{area}_{a,r}$, $\texttt{a} \in \texttt{A}$, $\texttt{r} \in \texttt{R}$, that is the number of resources of type r required by area a. The communication between the areas is modeled by specifying the number of wires connecting each pair of areas ($\texttt{areas\_wires}_{a_1,a_2}$, $\texttt{a}_1,\texttt{a}_2 \in \texttt{A}$) and the throughput between areas ($\texttt{areas\_comm}_{a_1,a_2}$, $\texttt{a}_1,\texttt{a}_2 \in \texttt{A}$). Finally, the designer can specify his/her requirements about the placement of the areas; $\texttt{same\_dev}_{a_1,a_2}$, $\texttt{a}_1,\texttt{a}_2 \in \texttt{A}$, equals to 1 if areas $\texttt{a}_1$ and $\texttt{a}_2$ must be placed on the same device, 0 otherwise.

When distributing the reliable circuit among the devices, the same set of metrics of the preliminary partitioner are considered, namely the distribution uniformity, the number of external wires, and the external communication. The output of the partitioning problem is a mapping, specifying for each area its hosting on one of the devices (it will be the floorplanner, in the subsequent step, to position it inside the device). This is modeled by the decision variable $\texttt{x}_{a,d}$, that equals to 1 if area a is hosted on device d, 0 otherwise. Furthermore, to account for communication and the spare areas, the following binary variables are introduced:

- $\texttt{y}_{a1,d1,a2,d2}$, equals to 1 if area a1 is hosted on device d1 and area a2 is hosted on device d2;
- $\texttt{z}_{a,r,d}$, equals to 1 if area a requires the maximum number of resources of type r on device d.

The partitioning problem includes constraints associated with areas placement, system distribution, external wires, and external communication, listed in Table 4.5.

Areas placement constraints. The partitioner must place each area on a device, as defined by Constraint C1. Moreover, the designer's requirements related to the placement of areas on the same device must be fulfilled. Constraint C5 allows to meet these requirements by exploiting the variables $\mathtt{y}_{a1,d1,a2,d2}$, that identify the placement of each couple of areas. Constraints C2, C3, and C4 define the valid values of the variables $\mathtt{y}_{a1,d1,a2,d2}$, by referring to the variables $\mathtt{x}_{a,d}$; Constraints C2 and C3 define the minimum bound of $\mathtt{y}_{a1,d1,a2,d2}$ (if $\mathtt{x}_{a1,d1}$ or $\mathtt{x}_{a2,d2}$ equal to 0, then $\mathtt{y}_{a1,d1,a2,d2}$ equals to 0), whereas Constraint C4 identifies the maximum one (if both $\mathtt{x}_{a1,d1}$ and $\mathtt{x}_{a2,d2}$ equal to 1, then $\mathtt{y}_{a1,d1,a2,d2}$ equals to 1).

System distribution constraints. The system must be distributed uniformly among the available devices, by considering both the areas hosting the system components and the spare ones. The spare area(s) must include enough resources to host the greatest area among the ones placed on the device. Constraints C6 and C7 define the variables $\mathtt{z}_{a,r,d}$, necessary to identify the greatest area for each type of resources on each device; Constraint C6 defines that $\mathtt{z}_{a,r,d}$ equals to 0 if the area is not hosted on the device, and Constraint C7 defines that there is one greatest area for each type of resources on each device. The spare area is identified by Constraints C8 and C9; Constraint C8 defines that the spare area must contain enough resources for relocating every area hosted on the device, and Constraint C9 sets the resources requirements of the spare area to equal the ones of the greatest area on the device for each type of resources. A number of spare areas equal to the number of non-recoverable faults specified by the designer must be reserved on each device. The spare areas concur in the computation of the resources occupation, together with the resources required by the Reconfiguration Controller and the areas hosted on the device, as computed by Constraint C10. The resources occupation on each device can not be greater than the available resources, as constrained by C11. The distribution of the system among the available devices is uniform when the gap between the maximum and minimum resources occupations among the devices is null. The maximum and minimum occupations of each type of resources are constrained by C12 and C13, and the difference between them defines the gap, as computed by Constraint C14.

External wires constraints. The external wires are the connections between areas hosted on different devices, by considering both adjacent and non adjacent devices. The number of external wires between each couple of devices is computed by Constraint 15, by summing the external connections between the areas hosted on the devices. This number is exploited to compute the number of external wires for the communication between adjacent and non-adjacent devices, as defined by Constraints

Table 4.5.: Reliability-aware partitioner constraints.

Areas placement	
C1	$\sum_{d \in D} x_{a,d} = 1, \forall a \in A$
C2	$y_{a1,d1,a2,d2} \leq x_{a1,d1}, \forall a1, a2 \in A, d1, d2 \in D$
C3	$y_{a1,d1,a2,d2} \leq x_{a2,d2}, \forall a1, a2 \in A, d1, d2 \in D$
C4	$y_{a1,d1,a2,d2} \geq x_{a1,d1} + x_{a2,d2} - 1, \forall a1, a2 \in A, d1, d2 \in D$
C5	$\sum_{d \in D} y_{a1,d,a2,d} \geq same\_dev_{a1,a2}, \forall a1, a2 \in A$
System distribution	
C6	$z_{a,r,d} \leq x_{a,d}, \forall a \in A, \forall r \in R, \forall d \in D$
C7	$\sum_{a \in A} z_{a,r,d} \leq 1, \forall r \in R, \forall d \in D$
C8	$spare\_res_{d,r} \geq area_{a,r} \cdot x_{a,d}, \forall d \in D, \forall r \in R, \forall a \in A$
C9	$spare\_res_{d,r} = \sum_{a \in A} area_{a,r} \cdot z_{a,r,d}, \forall d \in D, \forall r \in R$
C10	$dev\_occupation_{d,r} = \sum_{a \in A} area_{a,r} \cdot x_{a,d} + rec_r + max\_faults \cdot spare\_res_{d,r}, \forall d \in D, \forall r \in R$
C11	$dev\_occupation_{d,r} \leq dev\_res_{d,r}, \forall d \in D, \forall r \in R$
C12	$max\_dev\_occupation_r \geq dev\_occupation_{d,r}, \forall d \in D, \forall r \in R$
C13	$min\_dev\_occupation_r \leq dev\_occupation_{d,r}, \forall d \in D, \forall r \in R$
C14	$Gap_r = max\_dev\_occupation_r - min\_dev\_occupation_r, \forall r \in R$
External wires	
C15	$d\_to\_d\_wires_{d1,d2} = \sum_{a1 \in A} \sum_{a2 \in A} areas\_wires_{a1,a2} \cdot y_{a1,d1,a2,d2}, \forall d1, d2 \in D$
C16	$dev\_dir\_wires_{d1,d2} = d\_to\_d\_wires_{d1,d2} \cdot direct\_comm_{d1,d2}, \forall d1, d2 \in D$
C17	$dev\_non\_dir\_wires_{d1,d2} =$ $\sum_{d \in D} (d\_to\_d\_wires_{d1,d} \cdot non\_direct\_comm_{d2,d1,d} + d\_to\_d\_wires_{d,d2} \cdot non\_direct\_comm_{d1,d,d2}) +$ $\sum_{d3 \in D} \sum_{d4 \in D} d\_to\_d\_wires_{d3,d4} \cdot non\_direct\_comm_{d1,d3,d4} \cdot non\_direct\_comm_{d2,d3,d4}, \forall d1, d2 \in D$
C18	$dev\_dir\_wires_{d1,d2} + dev\_dir\_wires_{d2,d1} + dev\_non\_dir\_wires_{d1,d2} +$ $dev\_non\_dir\_wires_{d2,d1} \leq dev\_wires_{d1,d2}, \forall d1, d2 \in D$
C19	$Wires = \sum_{d1 \in D} \sum_{d2 \in D} dev\_dir\_wires_{d1,d2} + dev\_non\_dir\_wires_{d1,d2}$
External communication	
C20	$ext\_area\_comm_{a,d} = \sum_{a1 \in A} areas\_comm_{a,a1} \cdot x_{a,d} - area\_comm_{a,a1} \cdot y_{a,d,a1,d}, \forall a \in A, \forall d \in D$
C21	$dev\_to\_dev\_comm_{d1,d2} = \sum_{a1 \in A} \sum_{a2 \in A} areas\_comm_{a1,a2} \cdot y_{a1,d1,a2,d2}, \forall d1, d2 \in D$
C22	$dev\_comm_d = \sum_{a \in A} ext\_area\_comm_{a,d} + \sum_{d1 \in D} \sum_{d2 \in D} dev\_to\_dev\_comm_{d1,d2} \cdot non\_direct\_comm_{d,d1,d2}, \forall d \in D$
C23	$Comm = \sum_{d \in D} dev\_comm_d$

C16 and C17, respectively. The number of external wires required for inter-FPGA communication can not be higher than the available wires between each couple of devices, as constrained by C18, and for the overall platform it is computed by Constraint C19.

External communication constraints. The computation of the external communication exploits the computation of the external communication of each area on each device, defined by Constraint C20, and the external communication between each couple of devices, defined by Constraint C21. The external communication of each device is computed by Constraint C22, by summing the external communication of the areas hosted on the device and the external communication between non-adjacent devices through the considered one. Finally, the total external communication on the overall platform is computed by Constraint C23, by summing the external communication of each device.

The partitioning problem includes only binary and continuous variables and linear constraints, hence it is a MILP problem. It is a multi-

objective problem, with the following objective function:

$$min(w_{gap} \cdot \sum_{r \in R} \frac{Gap_r}{dev\_res_r} + w_{wires} \cdot \frac{Wires}{tot\_wires} + w_{comm} \cdot \frac{Comm}{\sum_{a1 \in A} \sum_{a2 \in A} comm_{a1,a2}})$$

where $\mathtt{w}_{gap}$, $\mathtt{w}_{wires}$, and $\mathtt{w}_{comm}$ are weights assigned by the designer.

4.6. Floorplanning

The floorplanning task finds a suitable placement of the hardened areas on the FPGA fabric (refer to Table 4.1 for the parameters characterizing the device). Indeed, also the Reconfiguration Controller is considered as an area to be positioned on the device. The areas are seen as rectangles of specified height and width, identified by coordinates [($\mathtt{x}_{BL}$,$\mathtt{y}_{BL}$),($\mathtt{x}_{TR}$,$\mathtt{y}_{TR}$)], being the bottom left corner and the top right corner, respectively. To obtain a feasible implementation, they must satisfy the following constraints:

- Each area is contained in the device:

$$0 \leq x_{BLa} < x_{TRa} < width_{dev}, \ \forall a \in A$$

$$0 \leq y_{BLa} < y_{TRa} < height_{dev}, \ \forall a \in A$$

- Different areas do not overlap with each other:

$$a_1 \cap a_2 = \oslash, \ \forall a_1, a_2 \in A$$

- Each area satisfies the resources requirements:

$$area_{a,r} \leq area\_res_{a,r}, \ \forall a \in A, \ \forall r \in R$$

  where $\mathtt{area}_{a,r}$ is the number of resources of type `r` required by components implemented in area `a` and $\mathtt{area\_res}_{a,r}$ is the number of resources of type `r` contained in area `a`.

These areas, being reconfigurable, are subject to reconfiguration constraints too. The FPGAs reconfiguration capability is exploited by identifying the number of rows of frames (`num_frame_rows`) and the number of frames per column for each kind of resource, allowing to accurately compute the reconfiguration overhead. The distinction between 1D and 2D reconfiguration is modeled by imposing constraints on the definition of the reconfigurable areas:

- 1D. The height of the reconfigurable area must be equal to the height of the device (each frame spans the whole height of the device):

$$y_{BLa} = 0,\ \forall a \in A$$

$$y_{TRa} = height_{dev} - 1,\ \forall a \in A$$

- 2D. The height of the reconfigurable area can be multiple of the height of the row of frames ($\texttt{height}_{frame\_row}$):

$$y_{BLa} = \alpha \cdot height_{frame\_row},\ \forall a \in A$$

$$y_{TRa} = \beta \cdot height_{frame\_row} - 1,\ \forall a \in A$$

  where α, β are integers $\in (0, \texttt{num\_frame\_rows}]$.

- 1D and 2D. The width of the reconfigurable area must be multiple of two slices columns:

$$x_{BLa} = \gamma \cdot 2,\ \forall a \in A$$

$$x_{TRa} = \delta \cdot 2 - 1,\ \forall a \in A$$

  where γ, δ are integers $\in [0, \frac{width_{dev}}{2}]$.

Figure 4.6 shows these rules, presenting the configuration of a module with (a) 1D and (b) 2D schemas.

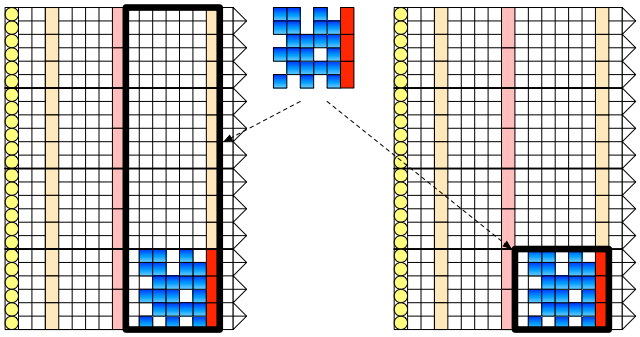


Figure 4.6.: Comparison between 1D and 2D reconfiguration schemas.

An example of reconfigurable area definition according to the introduced model is shown in Figure 4.6, that presents (a) a correct specification, and two incorrect ones, namely (b) an area that does not contain the required resources and (c) an area that does not satisfy the reconfiguration constraints.

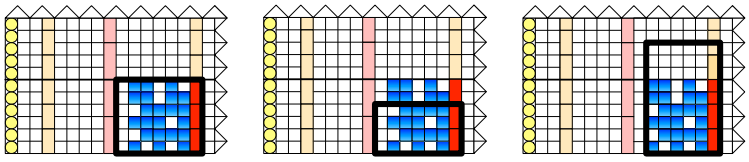


Figure 4.7.: Reconfigurable area definition: (a) correct specification, (b) area that does not contain the required resources, and (c) area that does not satisfy the reconfiguration constraints.

Besides defining the initial placement, the complete set of recovery replacements for each possible sequence of area failure must be considered (see Figure 4.4). The process is automated by means of a resource-aware floorplanner supporting 2D reconfiguration constraints necessary to enable partial scrubbing. The adopted floorplanner stems from classical approaches, using an enriched sequence pair representation and exploiting the simulated annealing engine customized to cope with the specific scenario. The initial floorplanner, presented in [47], has been extended to perform forbidden region avoidance. First, the initial placement is generated. Then, for each possible single non-recoverable fault, another placement is found by avoiding the faulty region. The process continues by following the order of the recovery actions from non-recoverable faults as shown in Figure 4.4.

The floorplanning task produces the placement constraints for each area on the related FPGA. The hardened circuit is placed on the multi-FPGA platforms and, integrating the necessary Reconfiguration Controllers (one for each FPGA), the proposed reliable system is implemented.

4.7. Prototype framework

The proposed design flow is implemented by a prototype framework, developed to support the designer in the realization of the reliable system. In the following, the implementation of the flow's main tasks (namely partitioning, hardening, and floorplanning) is described.

The circuit partitioning, both the preliminary and the reliability-aware one, is based on MILP optimization models, described in Sections 4.3 and 4.5.2, respectively. Both models have been implemented in AMPL. To solve the optimization problems, we used the IBM ILOG CPLEX

12.1 tool, which implements a parallel branch and cut procedure [48]. The physical system supporting the partitioner execution is based on VMWare ESXi 4.0, running on an Intel Nehalem dual socket quad-core system with 32 GB of RAM. CPLEX is hosted on a Virtual Machine (VM) running Ubuntu 11.04 Linux. The VM has four physical cores dedicated to its execution with guaranteed performance and 8GB of memory reserved. The partitioner performance has been measured and evaluated in terms of execution time, as reported in Appendix A, showing that, despite being the partitioning problem NP-complete [49], the proposed models run in acceptable time.

The CPLEX tool has been exploited also to solve the optimization model for the independently recoverable areas definition task, described in Section 4.4. Also this optimization model has been implemented in AMPL.

Finally, the floorplanning task has been developed in a C++ prototype. This floorplanner has been empirically evaluated by means of the experimental campaign reported in [47].

By performing the tasks above, the developed prototype framework allows to identify the solution offering the most convenient trade-off with respect to the designer's requirements.

4.8. Design flow for fixed number of non-recoverable faults

The design flow described in the previous sections builds the envisioned fault tolerant system on a multi-FPGA platform specified by the designer. Alternatively, it is possible to define the reliable system based on the number of non-recoverable faults that must be tolerated; after predicting the number of non-recoverable faults expected during the mission lifetime, the minimum number of FPGAs required to implement the hardened circuit and the Reconfiguration Controllers is identified. A design flow targeted at this second scenario is required.

The proposed flow for designing the fault tolerant system when the number of non-recoverable faults is fixed is shown in Figure 4.8. With respect to the previous flow, the required inputs and the preliminary partitioning task change. Whereas the previous flow takes in input the model of the multi-FPGA platform, this second approach requires the model of the single FPGAs composing the platform and the number of non-recoverable faults to be tolerated. The circuit components are taken in input by the reliability-aware iterative partitioning task, that distributes them among the necessary FPGAs. A hardened version of the

circuit is taken into account, where TMR is applied on each component. By starting from two FPGAs, the partitioner distributes the hardened components among the devices and increases the number of required FPGAs as far as it identifies the suitable spare region for tolerating the non-recoverable faults. The subsequent tasks are the ones of the previous flow. The two approaches are compared in Chapter 7 by implementing a real case study.

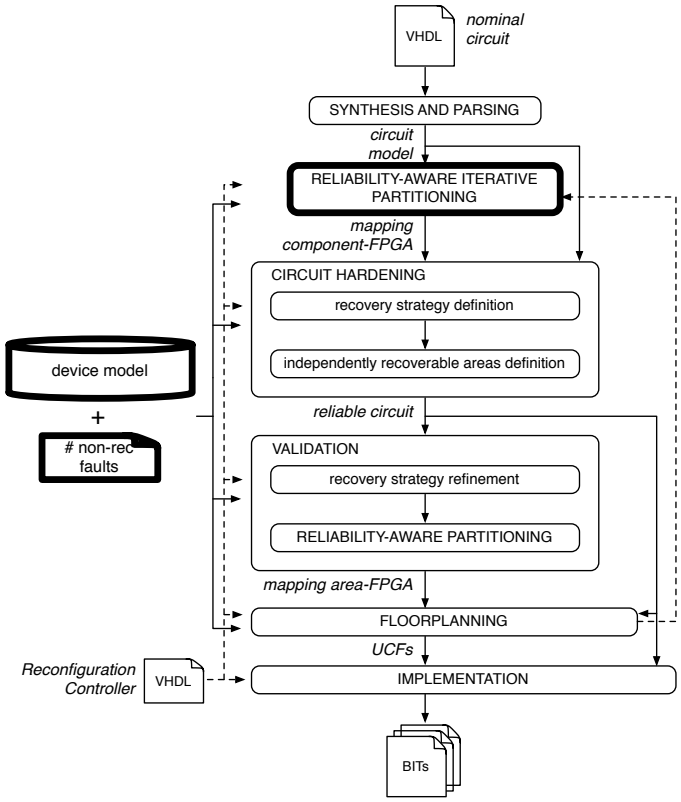


Figure 4.8.: Design flow for fixed number of non-recoverable faults.

4.9. Chapter summary

In this chapter, we have introduced a flow for designing fault tolerant systems according to the methodology proposed in the thesis. A prototype framework implementing the flow has been developed, in order to automate as much as possible the implementation of such systems. All the various implementation issues characterizing modern FPGAs have been taken into account and an accurate model of the multi-FPGA plat-

form has been adopted. The proposed design flow produces a feasible solution implementing an autonomous fault tolerant system.

5. Fault classification

When FPGAs, in particular SRAM-based ones, are exploited for mission-critical applications, fault tolerance and recovery techniques must be applied, but first it is necessary to understand the faults themselves in order to apply the suitable recovery action. In fact, some faults can be recovered by reconfiguring the device (and possibly only the faulty part) with the same configuration loaded before the fault occurrence, while others, characterized by a destructive effect, lead to the necessity of relocating the functionality loaded onto the faulty part of the device. This chapter presents a classification strategy and its companion algorithm for the discrimination of faults in FPGAs based on their impact on the device elements and the consequent possibility to recover from them, so classifying faults into two macro categories; *recoverable* and *non-recoverable* ones. Once classification has been carried out, the suitable fault recovery strategy is applied, with the final aim of enabling the exploitation of SRAM-based FPGAs for mission-critical applications, such as the ones in the space environment. While the idea behind the classification strategy is straightforward and intuitive, the definition of the parameters at the basis of such an approach is the real challenge and complex point. The main contributions of our work are thus the formal definition of the parameters characterizing the algorithm and the evaluation of the conditions for correct fault classification.

The chapter is structured as follows: Section 5.1 discusses the related work. Sections 5.2 and 5.3 describe two preliminary proposals of fault classification algorithms, presented in [50] and [51], respectively. Section 5.4 describes the proposed algorithm and Section 5.5 discusses its evaluation. Finally, Section 5.6 draws the chapter summary.

5.1. Related work

Few works in literature deal with the problem of identifying faults that physically damage the device.

An approach dealing with the discrimination of faults in FPGAs is presented in [52]. The authors propose the use of a timer; if two errors are revealed in the same position in a time interval smaller than a predefined threshold, it is assumed that they are related to the presence of a

fault that has physically damaged the device. The main drawback of the approach is that it is based on the knowledge of the Mean Time Between Failure (MTBF), that is highly variable, being dependent on the specific operating conditions.

The work presented in [53] pursues an objective similar to ours, by proposing a mechanism for the diagnosis of hard faults in microprocessors. The approach diagnoses hard faults at the Field Deconfigurable Unit (FDU) level of granularity. When an error is detected in an instruction executed by the microprocessor, an error counter for every FDU used by that instruction is incremented and, when a counter exceeds a threshold, the related FDU is identified as affected by a hard fault, and is deconfigured and no longer used. As long as transient errors do not lead the error counters to exceed the threshold, the counters are cleared periodically with a frequency depending on the expected transient error rates, or when a deconfiguration is activated. Moreover, the threshold is chosen not to be *too small* and is related to the specific FDU usage. Although the overall solution is presented, no indication is available on how to derive the parameters' value.

Indeed, we deem fundamental to be able to define the values of the parameters to distinguish between recoverable and non-recoverable faults, in a systematic way based on the application scenario. Therefore, although the existing proposals are meaningful in terms of the presented strategies, they cannot be adopted. In particular, we focus our attention on how to derive such parameters, in a more general way, also determining them for our environmental space-mission context. More precisely, with respect to the previous contributions, the proposed work formally defines the parameters characterizing the algorithm and evaluates the conditions for correct fault classification, presenting a robustness analysis, in order to see how the overall behavior may be affected by possible impreciseness.

5.2. First preliminary proposal

A first preliminary proposal of fault classification algorithm has been presented in [50]. We propose a classification of the fault affecting a portion of the FPGA based on the *absolute and relative frequency* of the detected faults. The rationale is that, if an area is affected by a fault more frequently than the others, a relocation should take place to prevent further use of the area.

The defined strategy keeps track of both the number of faults occurring in each area (application areas and controller) and of their his-

tory, recording the locations of the most recent observed faults. To support this strategy, the Fault Classifier has n separate counters, to count the number of faults per area, and a *trend buffer* to track the most recent areas affected by the faults. The dimension of the buffer is determined by the application scenario, in such a way that the higher the expected SEU frequency, the deeper the buffer. It is worth noting that the adopted methodology for system hardening produces solutions with a limited number of application areas (see Chapter 3), thus the number of counters is small and the buffer has a manageable size. The foreseen system lifetime is used to estimate a k threshold, together with the SEU incidence. A combination of absolute and relative frequencies of the area corruption contributes to determine which kind of reconfiguration to apply. More in detail, the fault classification process considers i) the faulty area presence in the buffer, ii) the absolute value of the counter related to the faulty area with respect to k, and iii) the relative value of the counter with respect to the other counters, as reported in the flow diagram in Figure 5.1.

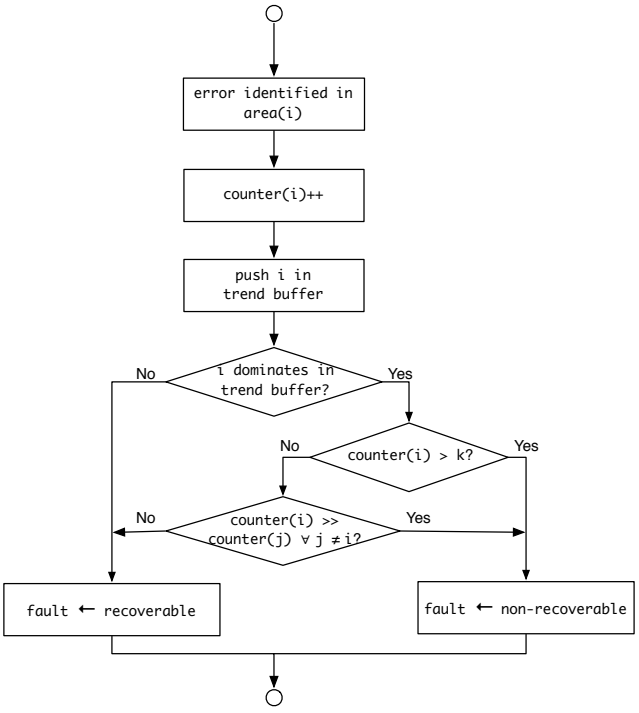


Figure 5.1.: First proposal of fault classification algorithm.

The first preliminary proposal of fault classification algorithm is rough and not evaluated, since the focus of [50] is on the design of the overall

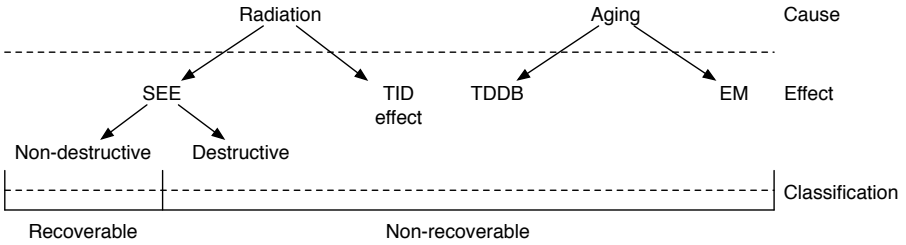


Figure 5.2.: Cause-based fault classification in the space environment.

engine managing fault tolerance. Thus, in this approach, a formal definition and accurate study of the parameters characterizing the algorithm is not provided.

5.3. Second preliminary proposal

A detailed and refined version of the fault classification algorithm has been presented in [51]. The algorithm is based on a characterization of the radiation effects and aging mechanisms in the space environment, discussed in the upcoming subsection.

5.3.1. Fault quantification

Based on the fault characterization in the space environment presented in Chapter 2, we hereafter propose a fault quantification, identifying the parameters involved in the classification algorithm. Recoverable faults are the ones caused by radiations without a destructive effect, whereas non-recoverable faults are those caused by radiations with destructive effect, radiations' accumulation and device aging. Figure 5.2 shows such fault classification based on the analyzed causes and effects. It is worth noting that the HCE and NBTI aging phenomena were not considered in this second preliminary proposal, but the approach can be easily extended to consider such faults as well.

Recoverable faults, caused by radiations, are characterized by a frequency that depends by the device's cross section (i.e., the fraction of the device sensitive to radiations) and the target environment, whose conditions heavily influence the radiations rate. The cross section is distinguished into static and dynamic; the former one is the total sensitive fraction of the device, while the latter one is the operational fraction. A radiation in an unused FPGA resource typically does not affect a circuit's proper operation, consequently the fraction of faults due to radiations

causing the circuit failures is directly proportional to the specific sensitive subset of the FPGA static cross section. The second aspect influencing the recoverable faults rate is the destined environment, characterized by radiations rate that heavily depends by space environmental conditions. By taking into account the average SEU rate forecast into the environment (λ_{SEU}), the following formula has been proposed in [12] to calculate the rate of the faults due to radiations that are non destructive, which we identify as *recoverable faults rate*:

$$\lambda_{rec} = \frac{\text{Dynamic cross section}}{\text{Static cross section}} \cdot \lambda_{SEU}$$

Based on this parameter, the Mean Time To Failure (MTTF) for recoverable faults can be estimated as:

$$MTTF_{rec} = \int_0^{\text{lifetime}} t \cdot f_{lifetime}(t)\, dt = \frac{1}{\lambda_{rec}}$$

being $f_{lifetime}(t)$ the exponential probability density function generally assumed to apply to electronics hardware. Consequently, the estimated number of recoverable faults is:

$$\#faults_{rec} = lifetime \cdot \lambda_{rec}$$

The other faults due to radiations are non-recoverable ones, and they are destructive SEEs and TID effects. Their failure rate is related to the amount of dose and does not obey to constant failure rates over time. SEL occurrence has not been taken into account since test reports on SRAM-based FPGAs reveal that no SEL was observed during the experiments up to the maximum tested Linear Energy Transfer of tens of $MeVcm^2/mg$ ([16], [17]), and no other data is available. TID effects occurrence is calculated by considering the device's TID and computing the predicted dose rate δ as follows:

$$MTTF_{TID} = \frac{TID}{\delta}$$

Consequently, the number of non-recoverable faults due to TID effects is the following:

$$\#faults_{TID} = \frac{lifetime}{MTTF_{TID}}$$

The last class of non-recoverable faults includes those due to device aging: TDDB and EM. The MTTFs due to such faults are estimated by using the equations proposed in [19]. The MTTF due to TDDB is strongly dependent on the thickness and area of the oxide, the gate voltage, the

temperature and the leakage current through the gate; it is estimated by exploiting the MTTF empirically evaluated for an older technology ($MTTF_{TDDB0}$) characterized by oxide thickness $t_{ox0} = 10nm$ and with an applied gate voltage $V_0 = 7V$:

$$MTTF_{TDDB} = \frac{MTTF_{TDDB0}}{10^{\frac{t_{ox0}-t_{ox}}{0.22}}(\frac{V_0}{V})^{a-bT}e^{\frac{X+\frac{Y}{T}+ZT}{kT}}}$$

where X, Y, Z, a and b are constants obtained from empirically fitted numbers for scaling. Thus, it is possible to estimate the expected number of non-recoverable faults due to TDDB:

$$\#faults_{TDDB} = \frac{lifetime}{MTTF_{TDDB}}$$

EM causes device aging involving interconnections, and its MTTF depends primarily on the current density J and the wire length L as follows:

$$MTTF_{EM} = \frac{J^{-n}e^{\frac{E}{kT}}}{L}$$

where n, E and k are constants. The expected number of non-recoverable faults due to EM can be estimated as:

$$\#faults_{EM} = \frac{lifetime}{MTTF_{EM}}$$

In conclusion, the foreseen number of non-recoverable faults is computed as follows:

$$\#faults_{nrec} = \#faults_{TID} + \#faults_{TDDB} + \#faults_{EM}$$

and the total number of faults occurring during the system's lifetime is computed as follows:

$$\#faults_{tot} = \#faults_{rec} + \#faults_{nrec}$$

5.3.2. Proposed classification algorithm

The presented quantification of faults affecting the FPGAs in the space environment is used as the basis for the classification algorithm. The proposed algorithm classifies the fault affecting a portion of the FPGA as recoverable or non-recoverable based on the relative and absolute frequency of faults, as in the first preliminary proposal, and also the estimated scenario.

The defined strategy keeps track of the history of faults, recording the locations of the most recently observed faults in a trend buffer, and of the number of faults occurring in the payload sub-system, keeping three separate counters, one for each type of non-recoverable fault (TID, TDDB, and EM). The rationale is that, if an area is being affected by a fault more frequently than the others, the algorithm verifies whether the relevant frequency is due to radiations accumulation or device aging. If the estimated MTTF matches the present scenario, the fault is classified as non-recoverable, recoverable otherwise. For each fault detection, if an area is being affected more than the others, a discrimination is made, based on the number of recoverable faults that are estimated to occur before a non-recoverable fault (derived from the computed MTTFs). Thus, each counter acts actually as a sort of timer. Such hypotheses are supported by the frequencies identified for recoverable and non-recoverable faults' occurrence. The proposed algorithm is described in the flow diagram in Figure 5.3.

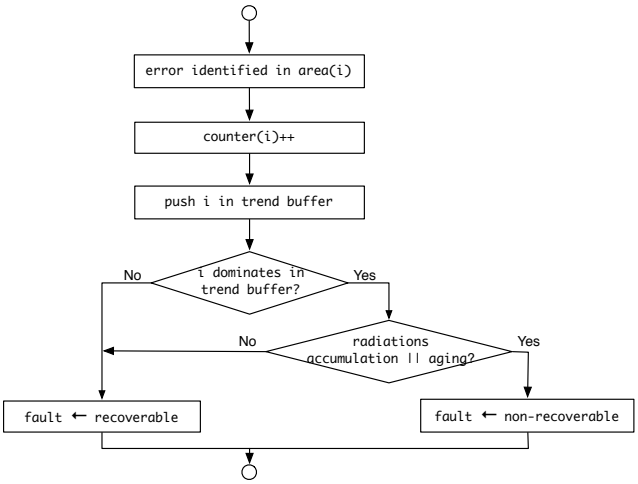


Figure 5.3.: Second proposal of fault classification algorithm.

The size of the trend buffer is related to the target environment such that the higher the expected fault rate the deeper the buffer, and is based on a defined observation period t:

$$l = \lambda_{SEU} \cdot t$$

selecting t such that $l >> m$. The first control performed by the algorithm verifies if the fault identified in $area_i$ has been registered more frequently than faults in other areas during the last observations. The

frequency in the trend buffer is:

$$Presence_i > \alpha \cdot Presence_j, \forall j \neq i$$

where α is a constant chosen in the interval (1,l). The second control performed by the algorithm discriminates between faults due to radiations accumulation or device aging. The former aspect is estimated as follows:

$$Counter_{TID} > Threshold_{TID} = \frac{\#faults_{tot}}{\#faults_{TID}}$$

while device aging is estimated as follows:

$$Counter_{TDDB} > Threshold_{TDDB} = \frac{\#faults_{tot}}{\#faults_{TDDB}} \; || \;$$

$$Counter_{EM} > Threshold_{EM} = \frac{\#faults_{tot}}{\#faults_{EM}}$$

Once the operating scenario is characterized, it is possible to derive the parameters driving the fault classification. We have developed a prototype and have evaluated it by simulating the system behavior in a case study, as reported in the next subsection.

5.3.3. Algorithm evaluation

An experimental session has been performed by considering a circuit characterized by 2206 slices, implemented on a Xilinx Virtex-II XC2V1000 and equally distributed into 5 different areas. A ten-years LEO mission at 560 km, 35 degree inclination is envisioned. An observation period $t = 40$ days has been selected, consequently setting the buffer length $l = 11$, and a trend buffer constant $\alpha = 3$. The experimental setup parameters, reported in Table 5.1, have been then computed with SPENVIS [54], an interface for models of the space environment and its effects, and by referring to [19].

The fault classification algorithm has been implemented and evaluated in a preliminary software version. When a fault occurs in the payload system, the fault classification algorithm, activated by the fault signal, determines the fault type and triggers the recovery strategy. The Mean Time To Repair (MTTR), corresponding to the time necessary to classify the fault and to reconfigure/relocate the faulty area, is neglected since it is much smaller than $MTTF_{rec}$. Recoverable faults are simulated by generating a fault in a payload system's area randomly chosen at various time instants, while a non-recoverable fault is simulated by continuously

Table 5.1.: Experimental setup parameters for the second preliminary algorithm evaluation.

Input parameters		**Calculated parameters**	
Radiation-related parameters			
Stat.cross sec.	$0.15 \cdot 10^{-8} cm^2$	λ_{rec}	0.285 SEU/day
Dyn. cross sec.	$0.01 \cdot 10^{-8} cm^2$	$MTTF_{rec}$	3.5 days
λ_{SEU}	4.276 SEU/day	$\#faults_{rec}$	1040
δ	86.4 rad/day	$MTTF_{TID}$	2314.8 days
TID	200 krad	$\#faults_{TID}$	1.5768
		$Threshold_{TID}$	666.50482
Aging-related parameters			
T	629.77 K	$MTTF_{TDDB}$	476 days
t_{ox}	1.2 nm	$\#faults_{TDDB}$	7.668
V	3 V	$Threshold_{TDDB}$	137
		$MTTF_{EM}$	2190 days
		$\#faults_{EM}$	1.7
		$Threshold_{EM}$	618.2028

activating a fault in the chosen faulty area from a selected time instant. In all cases the algorithm correctly classifies the fault.

We also evaluated the ability to correctly discriminate faults even when the $MTTF_{rec}$ has been erroneously estimated, to test the robustness of the approach. In these situations, the classification is not as prompt as in the optimal case, and the number of times the fault is detected and preliminary considered recoverable differs from the expected one. However, the relative dominance of the area being detected as faulty produces its effects and the fault is recognized as non-recoverable. For instance, by referring to Figure 5.4, with an estimated $MTTF_{rec} = 3.5$ days, should the real scenario be characterized by a $MTTF_{rec} = 5$ days, 44 activations instead of 8 are required to enable the classification to tag the fault as non-recoverable.

However, it can be noted that, also for this algorithm, the parameteres characterizing it (e.g., the buffer length) have not been formally defined and evaluated. In the following, we present the proposed fault classification algorithm, clearing the limitations of the previous solutions.

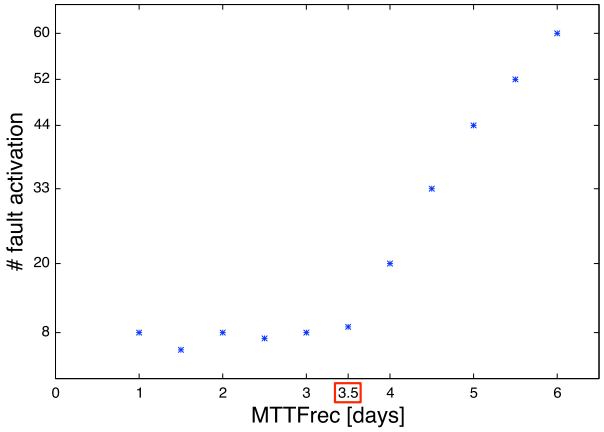


Figure 5.4.: Second preliminary algorithm robustness evaluation.

5.4. Fault Classifier specification

The fault classification algorithm is executed each time an error signal from the monitored areas shows the presence of a fault. The monitoring of the signals is continuous and the classification begins when the error is detected. The proposed classification is based on the analysis of the fault's frequency. An area is considered affected by a non-recoverable fault when it has been "recorded" as faulty during the last K subsequent observations. The behavior of the Fault Classifier implementing the proposed algorithm is shown by the flow diagram in Figure 5.5.

The challenge with this approach is selecting K; a too small K would lead to erroneously consider most faults as non-recoverable, while a too large K would cause not to recognize non-recoverable faults as such. In order to properly assist the designer of the reliable system in dimensioning K, we introduce the following elements: i) $P_{mis_{r-nr}}$ is the accepted probability of classifying a recoverable fault as non-recoverable, and ii) $P_{mis_{nr-r}}$ is the accepted probability of classifying a non-recoverable fault as recoverable. The value of K can be determined by using the above introduced probabilities of a mistake in the classification as thresholds in relation to the events that actually cause the algorithm to fail in classifying the fault. It is worth noting that, if the first kind of mistake is made, a relocation of the area deemed as faulty is performed, discarding an actually still healthy portion of the FPGA. In the second scenario, instead, useless reconfigurations are performed without a real benefit, thus incurring in a waste of time and effort, and in the accumulation of multiple faults.

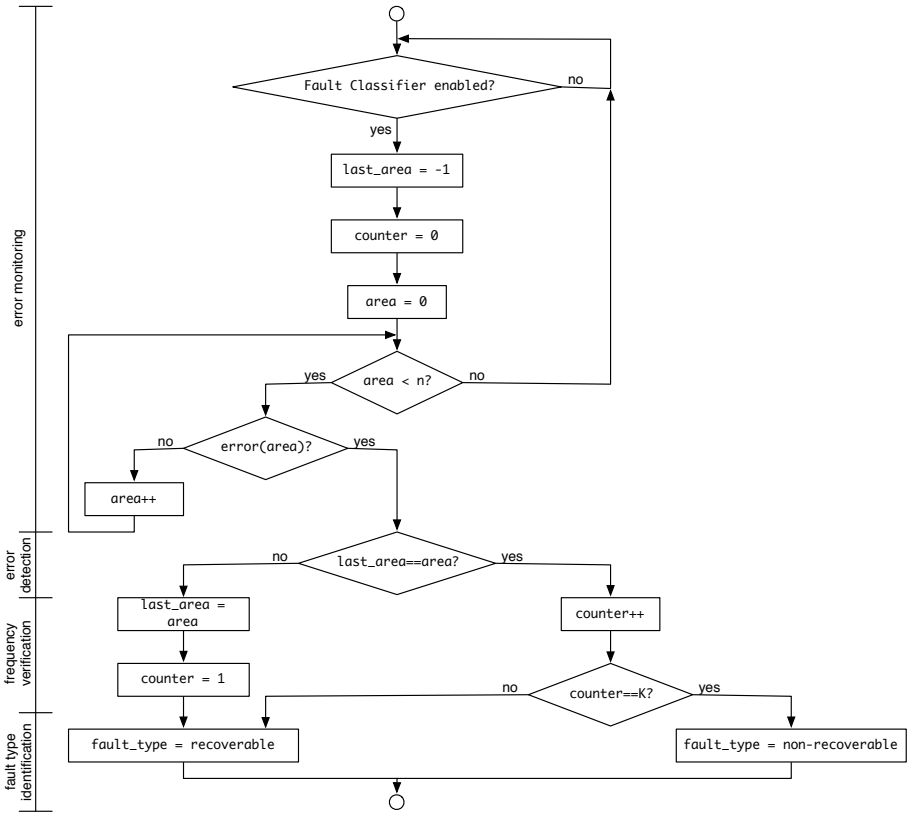


Figure 5.5.: Fault Classifier behavior

Recoverable faults are mistaken for non-recoverable ones whenever K subsequent recoverable faults occur all in the same area, and such event occurs with a probability $P_{area \times K}$ equal to:

$$P_{area \times K} = \left(\frac{1}{n}\right)^K \tag{5.1}$$

Therefore, to keep the probability of a mistake within the desired threshold, we obtain:

$$K \geq \lceil log_{\frac{1}{n}} P_{mis_{r-nr}} \rceil \tag{5.2}$$

In the definition of K, also the other kind of classification mistake must be taken into account. A fault is mistaken as recoverable if a non-recoverable one is not characterized by K subsequent detections because a recoverable fault occurs within the sequence. As a result, as shown in Figure 5.6, to avoid this situation, the time necessary to classify a non-recoverable fault must be shorter than the time elapsing between

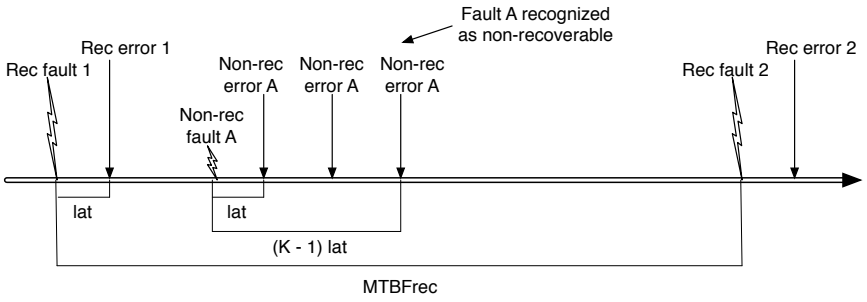


Figure 5.6.: Non-recoverable fault classification

two recoverable faults, defined as $MTBF_{rec}$. This must be avoided as it leads to the violation of the single-error hypothesis on which this work is based. The probability of this event occurring is related to the latency lat, that is the time for a given non-recoverable fault to produce an error. Since for the first $(K-1)$ times the system tries to recover by reconfiguring the area assuming it to be a recoverable error, $(K-1) \cdot lat$ indicates the time necessary to recognize a non-recoverable fault, where lat is associated with that specific fault. Nevertheless, as stated in Chapter 3, we assume a homogeneous implementation of the system onto the device, such that this parameter can be assumed as unique for all faults, using an average value. The fault, not yet classified as non-recoverable, is hereafter called *not-recovered*. By considering this scenario, the probability that a recoverable fault occurs before a previous, non-recoverable one is identified as such, is the following:

$$P_{lat} = \frac{(K-1) \cdot lat}{MTBF_{rec}} \tag{5.3}$$

Based on this equation, the threshold K must satisfy the following condition:

$$K \leq \frac{MTBF_{rec}}{lat} P_{mis_{nr-r}} + 1 \tag{5.4}$$

By selecting a value of K which satisfies both Equation 5.2 and Equation 5.4, we obtain an algorithm that correctly identifies non-recoverable faults with desired level of accuracy. In general, though, latency is neglected since the relation between fault and error is of orders of magnitude smaller than fault occurrence, hence Equation 5.4 could be neglected and no upper bound for K could be identified. However, the latency is a positive value that actually depends on the specific implementation, so an upper bound for K must be considered. We define the

optimum value of K as follows:

$$K = \lceil log_{\frac{1}{n}} P_{mis_{r-nr}} \rceil \tag{5.5}$$

The choice of considering the errors' frequency for the classification is supported by the assumption that, when an area is affected by a non-recoverable fault, errors in that area are signaled with a higher frequency than errors due to recoverable faults. In fact, when an error is observed, a reconfiguration is triggered and, if the fault is recoverable, no error is then detected for the subsequent time window corresponding to the $MTBF_{rec}$, otherwise the reconfiguration produces a benefit only until an input causes the fault to be observed again (an error in the same area is detected after latency lat). This situation occurs because, once we observe an error due to a recoverable fault, we immediately reconfigure the system, correcting the fault, and the next error will be seen when the next recoverable fault happens (i.e., when the system is again hit by a particle causing a SEU). Concerning errors due to non-recoverable faults, instead, they are taken care of only once recognized, which happens with a certain latency after they are first encountered. This means that, once the first of such errors is seen, the next one will be observed as soon as the fault propagates again to the system's outputs. More rigorously, the frequency of a recoverable fault depends on its occurrence and its observability, whereas a not-recovered fault (i.e., a non-recoverable fault that has not been classified as such yet) is already present in the system and its frequency depends only on the latency related to the fault-error relation. As stated, this value is of orders of magnitude smaller than the $MTBF_{rec}$, even for harsh environments as the space one. As a consequence, we derive that the $MTBF_{rec}$ is bigger than the time elapsing between observations of the same non-recoverable fault, hence we can expect a number of not-recovered fault observations, here identified as K, before a recoverable fault occurs. Since the MTTR is equal for both recoverable and not-recovered faults, the frequencies of recoverable and not-recovered faults can be compared based only on their MTTFs, hence justifying the assumption underlying to the algorithm's design.

5.5. Algorithm evaluation

This section describes the evaluation of the proposed algorithm, by presenting the experimental setup and results in Sections 5.5.1 and 5.5.2, respectively. Finally, the algorithm is compared with the related work and the preliminary versions in Section 5.5.3.

5.5.1. Experimental setup

The algorithm has been tested by simulating the behavior of four systems in the space environment, where recoverable faults are caused by radiations without destructive effect (*SEU*), and non-recoverable faults are caused by radiations' accumulation (*TID*) and device aging (*TDDB* and *EM*). Ten-year long missions at three different orbits have been envisioned: Low-Earth Orbit (LEO), Polar orbit, and Medium Earth Orbit (MEO). We assume five years of Solar Minimum and five of Solar Maximum (characterized by recoverable faults rates $\lambda_{rec_{min}}$ and $\lambda_{rec_{max}}$, respectively), with two solar flares lasting one week and one hour ($\lambda_{rec_{week}}$ and $\lambda_{rec_{day}}$, respectively). The frequency λ_{rec} predicted for each system has been computed as $P_r \cdot \lambda_{SEU}$, where P_r is the probability that the fault hits a used resource, and λ_{SEU} is the forecast SEU rate. The values of P_r and λ_{SEU}, reported in Tables 5.2 and 5.3, respectively, are computed by using the data provided in [12]. The values of λ_{rec} are reported in Table 5.4. For each experimental session, also an average value $\lambda_{rec_{avg}}$ between the various conditions have been computed. Such data is the basis for the execution of 12 experimental conditions: three different orbits for each of the four considered systems.

Table 5.2.: Probabilities P_r in the payload systems.

Payload	Dyn. cross sec. $[cm^2]$ [12]	P_r
Multiplier	$8.3 \cdot 10^{-9}$	$6.5 \cdot 10^{-2}$
Counter	$3.5 \cdot 10^{-9}$	$2.7 \cdot 10^{-2}$
Synthetic	$3.2 \cdot 10^{-9}$	$2.5 \cdot 10^{-2}$
DSP kernel	$8.9 \cdot 10^{-9}$	$6.9 \cdot 10^{-2}$

Table 5.3.: Orbits' λ_{SEU}.

Orbit	$\lambda_{SEU_{min}}$ [SEU/day]	$\lambda_{SEU_{max}}$ [SEU/day]	$\lambda_{SEU_{week}}$ [SEU/day]	$\lambda_{SEU_{day}}$ [SEU/day]
LEO	0.7	0.4	0.4	0.4
Polar	1.6	1.3	26.4	91.2
MEO	1	11	88.8	312

Also prediction of non-recoverable faults' rates has been performed to identify a rough timeline for simulating non-recoverable faults. A rough

Table 5.4.: Predicted λ_{rec} at the considered environmental conditions.

	Orbit	$\lambda_{rec_{min}}$ [SEU/day]	$\lambda_{rec_{max}}$ [SEU/day]	$\lambda_{rec_{week}}$ [SEU/day]	$\lambda_{rec_{day}}$ [SEU/day]
Multiplier					
exp1	LEO	$4.5 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$	$2.6 \cdot 10^{-2}$
exp2	Polar	$10.4 \cdot 10^{-2}$	$8.4 \cdot 10^{-2}$	$171.6 \cdot 10^{-2}$	$592.8 \cdot 10^{-2}$
exp3	MEO	$6.5 \cdot 10^{-2}$	$71.5 \cdot 10^{-2}$	$577.2 \cdot 10^{-2}$	$2028.0 \cdot 10^{-2}$
Counter					
exp4	LEO	$1.9 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$
exp5	Polar	$4.3 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$	$71.3 \cdot 10^{-2}$	$246.2 \cdot 10^{-2}$
exp6	MEO	$2.7 \cdot 10^{-2}$	$29.7 \cdot 10^{-2}$	$239.8 \cdot 10^{-2}$	$842.4 \cdot 10^{-2}$
Synthetic					
exp7	LEO	$1.7 \cdot 10^{-2}$	10^{-2}	10^{-2}	10^{-2}
exp8	Polar	$4.0 \cdot 10^{-2}$	$3.2 \cdot 10^{-2}$	$66.0 \cdot 10^{-2}$	$228.0 \cdot 10^{-2}$
exp9	MEO	$2.5 \cdot 10^{-2}$	$27.5 \cdot 10^{-2}$	$222.0 \cdot 10^{-2}$	$780.0 \cdot 10^{-2}$
DSP kernel					
exp10	LEO	$4.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$
exp11	Polar	$11.0 \cdot 10^{-2}$	$9.0 \cdot 10^{-2}$	$182.2 \cdot 10^{-2}$	$629.3 \cdot 10^{-2}$
exp12	MEO	$6.9 \cdot 10^{-2}$	$75.9 \cdot 10^{-2}$	$612.7 \cdot 10^{-2}$	$2152.8 \cdot 10^{-2}$

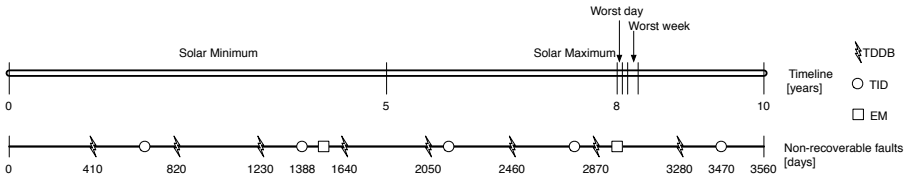


Figure 5.7.: Non-recoverable faults' timeline.

prediction is sufficient since the algorithm accuracy in identifying non-recoverable faults does not depend on their time occurrence. The rate of non-recoverable faults due to TID effect is computed by considering a predicted dose rate of 86.4 rad/day [55] and, by taking into account a Xilinx FPGA XCV1000, a TID of 60 krad [56]; we obtain $MTBF_{TID}$ = 694 days. Finally, we have considered the MTBFs due to device aging computed in [19], i.e., $MTBF_{TDDB}$ = 410 days and $MTBF_{EM}$ = 1460 days. The timeline of the predicted non-recoverable faults is shown in Figure 5.7.

5.5.2. Experimental results

The algorithm has been implemented in a software version for the evaluation. A SystemC module implements the algorithm, activated by signals simulating faults in the system. Recoverable faults are simulated by generating an error in an area randomly chosen. A non-recoverable fault is simulated by activating a fault in an area and by keeping the signal active until it is classified as non-recoverable. As this holds for all simulated areas and for both recoverable and non-recoverable faults, the validity of the experimental results is not compromised. Note that for recoverable faults, the MTTR has been neglected since it is much smaller than the MTTF, being the former in the units of milliseconds and the latter of days. In fact, the typical configuration time is around one microsecond per bit [57] and, in case of fault, only the faulty area must be reconfigured. The experimental sessions consisted of measuring the algorithm robustness and evaluating the condition for correct fault classification.

In the first experimental session, we varied K according to Equation 5.5. In a first evaluation phase, in addition to varying the experimental conditions according to Table 5.4, we analyzed the impact of the number of areas, by using $n = \{3, 5, 10, 15, 20, 30\}$, yielding a total of 72 experiments. The accepted probability of performing an erroneous classification of the recoverable faults has been set to $P_{mis_{r-nr}} = 0.005$, which means that, by varying n, we obtained different thresholds K. For each experimental condition and for each value of n, we considered the rate of recoverable faults mistakenly classified ($R_{mis_{r-nr}}$) as the fraction between the number of mis-classifications and the number of injected recoverable faults. Table 5.5 presents the results by reporting the average value of $R_{mis_{r-nr}}$ between the 12 conditions. It can be noted that, by decreasing K, more errors are performed by marking recoverable faults as non-recoverable. This is shown by the increasing value of $R_{mis_{r-nr}}$. However, $R_{mis_{r-nr}} \simeq P_{mis_{r-nr}}$ when the algorithm operates in a safe region, which means for sufficiently high values of K. In a second evaluation phase, we have set n to a fixed value ($n = 5$) and have varied $P_{mis_{r-nr}}$. Table 5.6 reports the average value of $R_{mis_{r-nr}}$. Results are not reported for $P_{mis_{r-nr}} > 0.1$ since we would have $K < 1$, completely hindering a proper classification. It can be noted that, also by varying $P_{mis_{r-nr}}$, $R_{mis_{r-nr}} \simeq P_{mis_{r-nr}}$.

In the second experimental session, we evaluated the condition for correct fault classification by computing the maximum latency according to Equation 5.3, as follows:

$$lat \leq \frac{P_{mis_{nr-r}} \cdot MTBF_{rec}}{K - 1} \tag{5.6}$$

Table 5.5.: Average $R_{mis_{r-nr}}$ by setting $P_{mis_{r-nr}}$=0.005

	n=3 (K=5)	n=5 (K=4)	n=10 (K=3)	n=15 (K=2)	n=20 (K=2)	n=30 (K=2)
exp1	0	0	0.018	0.055	0.046	0.055
exp2	0.018	0.006	0	0.043	0.036	0.061
exp3	0.004	0.004	0.013	0.046	0.034	0.029
exp4	0	0	0	0.102	0.051	0.051
exp5	0.008	0.008	0.008	0.074	0.041	0.057
exp6	0	0.012	0	0.036	0.036	0.024
exp7	0	0	0	0.139	0.028	0.083
exp8	0.009	0.009	0.017	0.053	0.044	0.044
exp9	0	0	0	0.065	0.026	0.052
exp10	0	0	0.016	0.049	0.05	0.049
exp11	0.012	0.006	0.024	0.067	0.048	0.054
exp12	0.008	0.004	0.008	0.057	0.036	0.028
avg $R_{mis_{r-nr}}$	0.005	0.004	0.104	0.065	0.04	0.049

Table 5.6.: Average $R_{mis_{r-nr}}$ by setting n=5

	$P_{mis_{r-nr}}$=0.001 (K=5)	$P_{mis_{r-nr}}$=0.004 (K=4)	$P_{mis_{r-nr}}$=0.01 (K=3)	$P_{mis_{r-nr}}$=0.1 (K=2)
exp1	0	0	0.064	0.156
exp2	0	0.006	0.018	0.165
exp3	0	0.004	0.034	0.143
exp4	0	0	0	0.154
exp5	0	0.008	0.033	0.164
exp6	0	0.012	0.024	0.167
exp7	0	0	0	0.167
exp8	0	0.009	0.053	0.21
exp9	0	0	0.026	0.208
exp10	0	0	0.049	0.164
exp11	0	0.006	0.054	0.164
exp12	0	0.004	0.036	0.17
avg $R_{mis_{r-nr}}$	0	0.003	0.032	0.169

This condition should guide the designer when defining the system, characterized by a fault-error relation respecting the maximum latency. For the values of K estimated in the previous experimental session, we computed the latency by setting sufficiently low values of the misclassification error probability, $P_{mis_{nr-r}} = \{0.005, 0.01, 0.05, 0.1\}$, and computing $MTBF_{rec} = \frac{1}{\lambda_{rec}}$, where λ_{rec} is the worst value between the environmental conditions. Table 5.7 presents the results by reporting the average value of *lat* in the 12 conditions. Results show fault classification succeeds even with *lat* up to the order of hours (with a few exceptions in the range of tens of minutes), which is a very high value considering a typical fault-error relation, thus confirming the algorithm rationale. Moreover, it should be taken into account that we considered very strict values for $MTBF_{rec}$, by considering the worst case between the various environmental conditions. Hence, the latency required for correct fault classification is even higher than the one computed. Finally, it should be noticed that the latency increases when raising the value of $P_{mis_{nr-r}}$, that can be higher than the selected $P_{mis_{r-nr}}$ since mistaking a fault as recoverable is less serious than mistaking it as non-recoverable. In fact, in the latter case, we discard an actually still healthy portion of the FPGA. In all cases, the latency is in the order of hours also at high orbits, where faults are more frequent. In most cases, several hours are allowed for fault detection and recovery.

5.5.3. Comparison with related work

The proposed algorithm has been compared with the related work and the preliminary versions. No comparison with the first preliminary version of the algorithm has been performed since no indication for setting the parameters characterizing the algorithm has been provided.

The approach in [52] can be considered equivalent to ours when K is set to 2, if we do not fix a MTBF (not so easily predictable). Such value of K is suitable when the number of areas is high enough (greater than 10, according to our approach), otherwise it is possible to fall into a mis-classification of a recoverable fault as non-recoverable.

The algorithm in [53], targeted for microprocessors, is quite similar to ours. Both approaches count the number of faults affecting the areas and identify hard faults when a predefined threshold is exceeded. We verified that by setting the period for clearing the counters as MTBF of recoverable faults and the threshold as K, the algorithm in [53] has the same classification performance as ours. However, our approach only requires the use of one counter of length K, whereas, in [53], it is necessary to set as many counters as the areas, whose number can be very high; counters

Table 5.7.: Average latency [hour]

	$K=2$	$K=3$	$K=4$	$K=5$
exp1	73.85	36.92	24.61	18.46
exp2	24.61	12.31	8.21	6.15
exp3	6.05	3.03	2.02	1.51
exp4	177.78	88.89	59.26	44.44
exp5	59.26	29.63	19.75	14.81
exp6	14.57	7.29	4.86	3.64
exp7	192.00	96.00	64.00	48.00
exp8	64.00	32.00	21.33	16.00
exp9	15.74	7.87	5.25	3.93
exp10	69.57	34.78	23.19	17.39
exp11	23.19	11.59	7.73	5.80
exp12	5.70	2.85	1.90	1.43
$P_{mis_{nr-r}}$=0.005	1.54	0.77	0.51	0.39
$P_{mis_{nr-r}}$=0.01	3.09	1.54	1.03	0.77
$P_{mis_{nr-r}}$=0.05	15.45	7.72	5.15	3.86
$P_{mis_{nr-r}}$=0.1	30.90	15.45	10.30	7.72

can considerably impact the area occupation of the Fault Classifier both for their number and for the hardening process. Moreover, the structure of our algorithm enables a formal and rigorous study of the effect of each parameter.

Finally, we have taken into account the second preliminary version of the algorithm. We have considered that a fault occurs relatively more frequently than the others when only the faulty area is registered in a buffer of length K. Moreover, we have computed the thresholds related to non-recoverable faults to analyze their absolute frequency, by considering the values reported in the previous section. We have evaluated such algorithm by considering the number of fault observations required to classify the fault as non-recoverable and by comparing it with K, as shown in Figure 5.8. Results show that, in the second preliminary version, the number of fault observations to recognize a non-recoverable fault can be very high, whereas in the proposed solution is fixed.

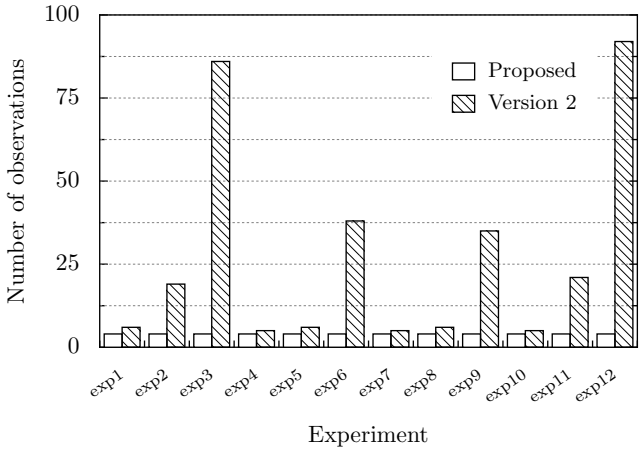


Figure 5.8.: Comparison of the number of fault observations to recognize a non-recoverable fault in the proposed solution and the second preliminary version of the algorithm.

5.6. Chapter summary

In this chapter, an algorithm for the distinction between recoverable and non-recoverable faults has been designed. With respect to the other works proposed in literature, we formally define the parameters characterizing the fault classification algorithm and present rigorous mechanisms for dimensioning them based on the desired classification accuracy. Two preliminary versions of the algorithm have been proposed and a final solution has been accurately evaluated. Experimental results show that the proposed algorithm recognizes non-recoverable faults with desired level of accuracy and prove the effectiveness of the algorithm also with respect to related work. The algorithm has been designed and tested for space applications, being space the target scenario of the thesis, but the work is suitable for any application that must tolerate both recoverable and non-recoverable faults. The proposed algorithm is implemented by the Fault Classifier module of the Reconfiguration Controller, presented in the next chapter.

6. Reconfiguration Controller design

This chapter proposes the design of the controller in charge of managing the fault recovery of the multi-FPGA platform, contributing to the creation of the reliable system. We here introduce a reliability-aware Reconfiguration Controller, aimed at performing the fault classification presented in the previous chapter and managing the reconfiguration process of the faulty parts of the architecture to mitigate fault effects.

The Reconfiguration Controller is the main component responsible for the active fault mitigation process. It continuously monitors the error signals from the neighbor FPGA and, upon error detection, triggers a reconfiguration, based on the classification of the fault nature (recoverable vs. non-recoverable). Should the fault be considered as recoverable, the FPGA is partially reconfigured by reloading the bitstream portion related to the faulty area, otherwise the faulty area is relocated to a spare region by loading a new bitstream. Such recovery action by means of reconfiguration requires few seconds, and even few milliseconds in case of recoverable fault, being the typical configuration time around one microsecond per bit [57]. It is worth noting that the proposed approach is suitable for stateless applications, since the application must be stopped in case of non-recoverable fault; alternative solutions can be investigated as future work.

Indeed, it is necessary to prevent erroneous reconfigurations by detecting also faults affecting the Reconfiguration Controller itself. Hence, fault detection capability is required for the Reconfiguration Controller, also. More precisely, the controller itself can be seen as an area that, in case of fault, is recovered or relocated based on the type of occurred fault.

The chapter is organized as follows: Section 6.1 describes the related work on controller design. Section 6.2 introduces a software implementation developed to validate the designed controller with respect to its requirements, and Section 6.3 proposes a preliminary hardware reliability-aware implementation. Section 6.4 presents the design of the proposed solution and Section 6.5 describes the hardening of the module. Finally, Section 6.6 draws final considerations.

6.1. Related work

To the best of our knowledge, the engine closer to the one proposed and described in this chapter has been presented by Honeywell in [8]. It consists of a controller for a system composed of three FPGAs, each one hosting the same implemented system and thus with the same configuration. The controller, implemented on an external radiation-hardened ASIC, acts as a TMR voter. Upon error detection, it reconfigures the corrupted FPGA and re-synchronizes it with the two remaining FPGAs. Central in Honeywell's paper, as in our work, is the controller module, necessary for an appropriate management of the overall system functionality and responsible for detecting faults and correcting them by reconfiguring the FPGAs' functionalities. This solution has the intrinsic problem of applying device-level TMR, that generally leads to an inefficient exploitation of the multi-FPGA resources and computing capabilities. A high amount of resources and power is also spent during the reconfiguration of the whole faulty FPGA, something that could be limited by enabling the reconfiguration of the faulty portion of the device. By applying TMR at a finer granularity level, we better exploit the devices potential and we provide a scalable solution, independent of the number of FPGAs used. In addition, Honeywell's controller needs to be implemented on specific ASIC technology, eliminating most of the advantages of having FPGAs in the system, namely flexibility and relatively low cost. Finally, the solution can cope with recoverable faults but does not address non-recoverable ones.

Literature reports a few controller designs, however most of them focus on dynamic partial self-reconfiguration for single FPGA platforms, without taking reliability issues into account [58, 59, 60, 61]. More precisely, the authors of [58] propose a solution composed of a microprocessor, that manages the reconfiguration process, and of a controller for the Internal Configuration Access Port (ICAP) interface [62, 23], that enables on-chip dynamic partial reconfiguration. The microprocessor sends the reconfiguration request to the ICAP controller through a Device Control Register bus, and the ICAP controller accesses a memory to fetch the bitstream for reconfiguration by performing Direct Memory Access (DMA) via a Processor Local Bus (PLB). Besides introducing such architecture for dynamic partial reconfiguration, the authors introduce a method to calculate the expected reconfiguration throughput and latency. The use of DMA and burst transfers allows the presented reconfiguration controller to work at a speed that is close to the theoretical maximum achievable throughput.

A preliminary version of this reconfiguration controller exclusively de-

signed for the Xilinx Virtex-II Pro [62] FPGA is presented in [59]. In both versions, particular relevance is given to the ICAP controller, constituting an essential aspect in the achievement of the obtained performance. In fact, compared to an alternative realization using a state-of-the-art solution, namely the OPB HWICAP IP-Core [63], a speed-up by a factor of 20 has been achieved by the older version, and a speed-up by a factor of 58 has been obtained by the new one.

A customized version of this reconfiguration controller is shown in [60], where the authors describe a framework used to create the most suitable controller according to the reconfiguration scenario where it will be used. A set of metrics has been defined to describe the reconfiguration scenario and to set the following parameters of the reconfiguration controller: i) the bus interface of the ICAP controller, by allowing to choose between the PLB and the On-chip Peripheral Bus (OPB), such that the problem of having the ICAP controller on a bus and the memory on another one no longer exists; ii) the implementation type, in terms of slices or BRAMs, of the memory used to store the bitstream inside the ICAP controller, in order to set the resources requirement; finally, iii) the size of the internal memory, by allowing to find a trade-off between resources requirement and reconfiguration throughput. Thus, the attention has been oriented towards the speed-up of the reconfiguration process, as in the previous two reconfiguration controllers, and its simplification.

Another type of controller, implemented by a stand alone IP-Core, is presented in [61]. This custom soft core, called Parallel Configuration Access Port (PCAP), performs partial dynamic self-reconfiguration through the SelectMAP port, by using a partial bitstream stored in a BlockRAM memory within the FPGA itself. The proposed approach stores partial bitstreams in the on-chip memory and reads them from there under the control of PCAP. In this case, the reconfiguration process is accomplished without the ICAP controller, available only in some FPGA families, and without a microprocessor managing the process.

While the analyzed solutions do not take into account the system's fault tolerance and concentrate only on a single FPGA, they are the first proposals of efficient reconfiguration controllers and, as such, they were carefully taken into account while defining the presented solution, possibly adopting the advantageous aspects they propose. However, our work deals with the problem of multi-FPGA platforms' reliability, therefore the attention is on the effects of faults and their mitigation rather than on the overall performance. Hence, we propose a fault management strategy that exploits the reconfiguration capability of the devices to cope with the occurrence of both recoverable and non-recoverable faults.

The component implementing the strategy, namely the Reconfigura-

tion Controller, has been designed, by identifying its functionality and features. Innovative points of the presented solution are:

- distributed, low cost Reconfiguration Controller, eliminating the need to deploy it on ad-hoc device,
- management of both recoverable and non-recoverable faults, and
- scalable use of multiple FPGAs to build a reconfigurable reliable system.

A preliminary software implementation has been developed to validate the designed controller with respect to its requirements, and two hardware reliability-aware solutions have been proposed, as described in the next sections.

6.2. Software implementation

Both software and hardware implementations of the Reconfiguration Controller have been taken into account, to identify the most convenient solution, based also on the available FPGA platform. A preliminary software implementation has been developed and deployed for a board hosting a microprocessor, to refine the controller behavior and to verify the external reconfiguration aspect, triggered by the error detection.

The software-based system implementing the Reconfiguration Controller is shown in Figure 6.1. The Reconfiguration Controller is implemented by a MicroBlaze, that monitors the error signals from the other FPGA(s) through a General Purpose I/O (GPIO) component. In case of fault of a monitored FPGA, it takes the correct bitstream stored in an SRAM memory and performs the reconfiguration of the faulty portion of the system through a JTAG component, that communicates with the other FPGA through the configuration link. The bitstreams are stored in the SRAM memory through UART RS232 and the communication between all components is performed through OPB bus. The code executed by the MicroBlaze is stored in a BRAM memory that communicates with the processor through Local Memory Bus.

The design and prototype implementation of the Reconfiguration Controller have been defined using as a preliminary platform two Spartan-3A FPGAs. One FPGA hosts the Reconfiguration Controller and the other one the system with fault detection properties. When a fault occurs in the monitored system, the Reconfiguration Controller performs the reconfiguration of the system. The two FPGAs are connected through two physical links, one for configuration and one for communication. The

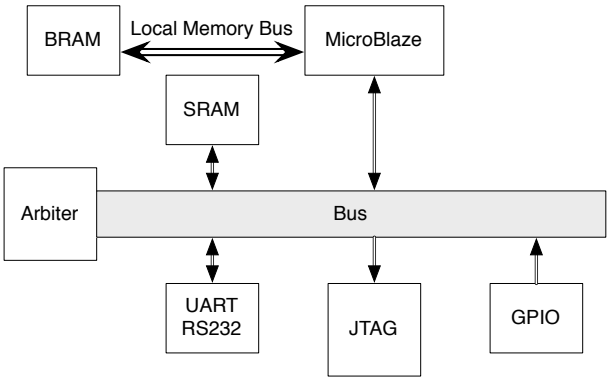


Figure 6.1.: First prototype of Reconfiguration Controller.

configuration link is obtained through the use of a cable attached to the A2 Expansion Connector of the Reconfiguration Controller board and the JTAG of the monitored system board, while the communication link uses a cable attached to the A2 Expansion Connectors of the two boards, as shown in Figure 6.2.

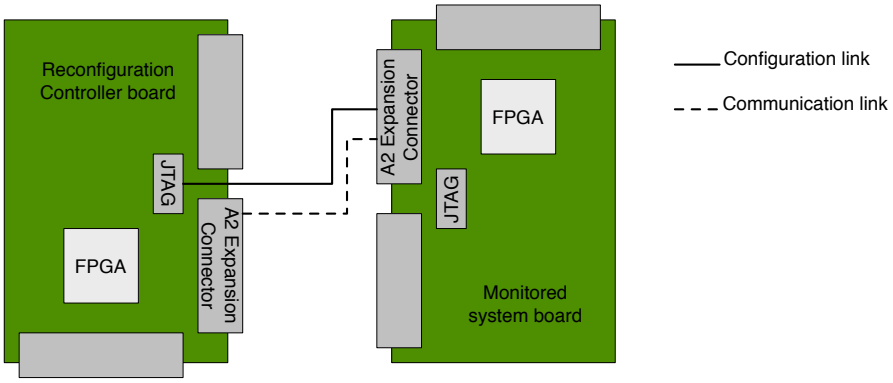


Figure 6.2.: Software-based implemented prototype platform.

The system monitored by the Reconfiguration Controller consists of a counter on which the TMR is applied, as shown in Figure 6.3. The counter computation is shown on the 7 segment display hosted on the board, and the outputs of the 3 counters are sent to a voter. The voter has been modified to support the fault generation, simulated by pressing a button on the board. The error signals are sent to the Reconfiguration Controller FPGA through the communication link.

When the Reconfiguration Controller detects an error, it performs the

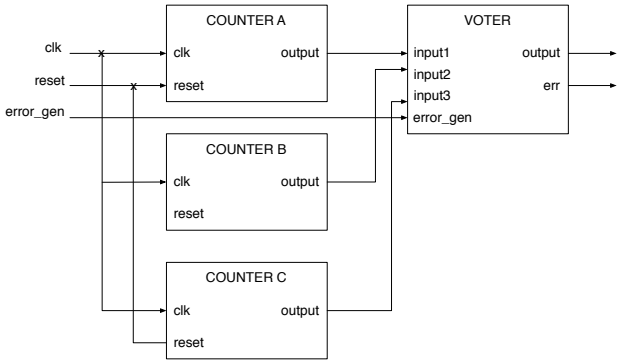


Figure 6.3.: Reliable system.

reconfiguration of the other FPGA by using the appropriate bitstream. The reconfiguration with different bitstreams has been tested by creating two versions of the counter; in the first version (bitstream 1) the count is from 0 to 9, while in the second one (bitstream 2) the count considers only even numbers. Bitstream 1 is loaded when fault occurs while 0, 1, 4 or 6 are displayed on the second board, otherwise bitstream 2 is loaded.

Two are the main limitations characterizing a software implementation, namely the sharing of the microprocessor with the application being normally executed, and the hardening against faults. As far as the former aspect is concerned, should the available microprocessor be exploited to run the nominal application (otherwise the platform would not host a microprocessor, to begin with), the fault mitigation management application would have to be executed periodically, during idle time, thus introducing a latency in fault detection and management. Considering the latter aspect, the prototype did not expose reliability features, since hardening by means of software-based techniques is expensive in terms of overhead (code and performance degradation) as well as it only allows for a partially reliable system [64]. A possible solution to achieve the necessary reliability consists in adopting a fault tolerant IP, such as Leon2-FT [65]. However, the final cost is quite prohibitive, with an occupation of available resources on an xc2v3000 FPGA requiring about 24% registers, 37% Block RAMs and 79% LUTs. For these reasons, a hardware solution, whose design is presented in the next sections, has been preferred.

6.3. Preliminary proposal

In our preliminary proposal, presented in [50], the fault detection capability required for the Reconfiguration Controller is guaranteed by means of readback; each Reconfiguration Controller of the system periodically accesses the neighbor FPGA to perform the partial readback of the configuration memory corresponding to the controller, to detect, and eventually repair, faults. Thus, the main tasks of the controller are: i) continuously monitoring the neighbor FPGA's error signals from the application areas, ii) periodically monitoring the controller hosted on the neighbor FPGA, and iii) upon error detection in one of the areas, application ones or controller, trigger a reconfiguration, based on the classification of the fault nature. The flow diagram shown in Figure 6.4 reports the behavior of the Reconfiguration Controller.

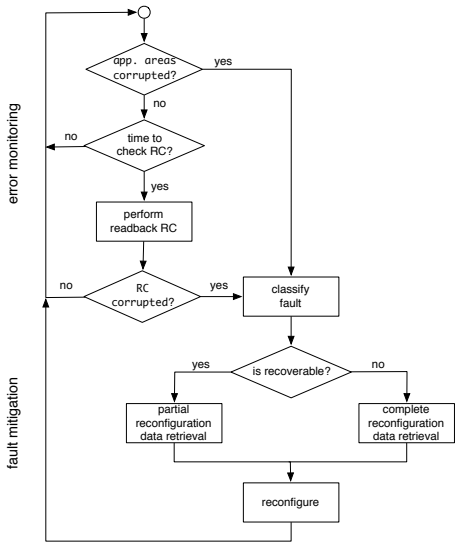


Figure 6.4.: Reconfiguration Controller flow diagram.

In order to carry out such tasks, the following modules have been identified as fundamental "building blocks": *Reconfiguration Interface*, in charge of performing the physical implementation of the reconfiguration process, providing access to the FPGA configuration memory interface, and *Manager*, to control the Reconfiguration Interface based on error detection and classification policies, determining what bitstream shall be used to reconfigure the platform and mitigate (recover or relocate) the effect of the fault. The Manager is, in turn, composed of the following elements: i) a *Readback Module*, for the neighbor controller verification,

ii) a *Fault Classifier*, to determine whether the detected fault can be considered recoverable or non-recoverable, and iii) a *Bitstream Module*, that retrieves the data and information to perform the necessary reconfiguration. Note how the Reconfiguration Interface is a bottleneck for the overall system reliability as a fault in it could trigger undesired reconfiguration, possibly corrupting the functionalities of the neighbor FPGA. Hence, even if there is no explicit self-analysis of the absence of faults in the Reconfiguration Controller, the controller itself is designed to avoid such undesired reconfigurations; when the reconfiguration request is asserted, the error signals are checked to detect if an error has actually been raised, otherwise the reconfiguration is blocked. An overview of the resulting Reconfiguration Controller structure is presented in Figure 6.5, and the detailed presentation of the Manager module is proposed in the next subsections.

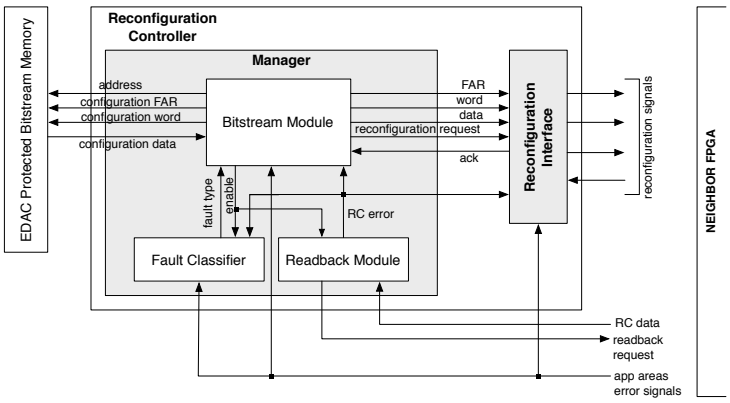


Figure 6.5.: Preliminary Reconfiguration Controller block diagram.

6.3.1. Readback Module

While the hardened application system organized in areas, as described in Chapter 3, is designed to be self-checking, i.e., it incorporates logic to detect the presence of a fault causing an error in its functionalities, this is not true for the Reconfiguration Controller itself. As explained, the Reconfiguration Controller is designed to behave correctly even if a fault occurs in the portion of the FPGA hosting it. However it is the neighbor Reconfiguration Controller's task, through the Readback Module, to determine the presence of such a fault.

We adopt the following strategy. The presence of a fault in the Reconfiguration Controller of $\texttt{FPGA}_{i+1}$ is determined by the Reconfiguration Controller of $\texttt{FPGA}_i$, which periodically reads the configuration data and

compares it with a golden model stored in a protected memory. Such operation is performed every $\texttt{T}_{rb}$ time instants, estimated on the MTBF of the adopted fault model in the envisioned application scenario, mimicking the methodologies presented in [4]. Furthermore, an `enable` signal, coming from the Bitstream Module, is provided in input, to block the readback activity while the neighbor FPGA is being reconfigured to correct a previously detected error.

The output of the comparison, $\texttt{error}_{RC_{i+1}}$, is encoded with the classical Two-Rail Code (TRC) [66], to provide an adequate level of protection within the Reconfiguration Controller module, hence preventing internal faults from triggering unnecessary/incorrect reconfigurations of the monitored neighbor controller $\texttt{RC}_{i+1}$. Such output signals are sent to the Fault Classifier, in charge of classifying the fault as recoverable or non-recoverable.

6.3.2. Fault Classifier

The Fault Classifier module receives the error signals from the Readback Module ($\texttt{error}_{RC_i}$) and from the monitored `n` application areas ($\texttt{error}_{app\ area_1}$, $\cdots$, $\texttt{error}_{app\ area_n}$) of $\texttt{FPGA}_{i+1}$, and, when an error is detected, it classifies the fault as recoverable or non-recoverable. In detail, the input of the module is a `(n+1)` $\times$ `2` bits signals, since each error signal is encoded with the TRC. Furthermore, as in the Readback Module, an `enable` signal, coming from the Bitstream Module, is provided to specify when the error signals are to be considered valid, or not. In fact, as soon as an error is detected, that fault is classified but other error signals should not be taken into account, otherwise the same fault may be accounted for more than once. Fault classification must be disabled in the time frame going from fault detection to the moment fault recovery has been completed.

In this preliminary proposal of controller, the Fault Classifier implements the algorithm described in Chapter 5. The output of the Fault Classifier module feeds the Bitstream Module with the information about having detected or not a fault, and on the derived nature, recoverable or non-recoverable, to trigger a partial or complete reconfiguration of the neighbor FPGA. The output configurations are `01` (no error), `00` (recoverable fault) and `11` (non-recoverable fault).

6.3.3. Bitstream Module

Upon error detection and based on the fault classification, the Bitstream Module disables fault classification and triggers a reconfiguration action,

accessing the protected memory to retrieve the appropriate configuration data. With such information, it programs the Reconfiguration Interface to carry out the FPGA (partial) reconfiguration to recover from the fault. The bitstreams are organized in memory as follows: the first bitstream consists of the current configuration of the monitored FPGA and is used for partial reconfiguration in case of a recoverable fault. The other bitstreams are used to recover from non-recoverable faults, each one of them appropriate for the detected faulty area.

To implement such functionality, inputs to the Bitstream Module are the `fault type`, `application areas error`, and `RC error` signals. When necessary, the Bitstream Module interacts with the memory to retrieve the desired (partial) bitstream, `configuration data`, to be forwarded to the Reconfiguration Interface, together with a `reconfiguration request`, in terms of `FAR`, `word`, and `data` (i.e., the information necessary to perform the reconfiguration). An acknowledge signal, `ack`, is used to increase the Reconfiguration Controller reliability, by having an additional flag to check that reconfiguration has been performed correctly. When the `ack` is received it means that fault recovery is complete and fault classification can be re-enabled (through `enable` signal), thus restoring the system to its nominal behavior.

6.3.4. Solution evaluation: limits

The designed Reconfiguration Controller is the first proposal of controller managing both recoverable and non-recoverable faults. It is a distributed, low cost engine, eliminating the need to deploy it on ad-hoc device and allowing scalable use of multiple FPGAs to build a reconfigurable reliable system.

In this first proposal, the reliability of the controller is guaranteed by means of readback, that is an expensive operation and subject to errors. Thus, we investigated the reliable implementation of the engine, as described in the next section.

6.4. Proposed solution

The design and prototype implementation of the proposed Reconfiguration Controller have been defined using as a preliminary platform the one presented in Section 6.2, where each FPGA is connected to the other through the physical links for configuration and communication. The recovery bitstreams are stored for each device in an XCF02S serial configuration Flash PROM, that, besides configuration data, can possibly store additional non-volatile data. Each FPGA hosts both the

Reconfiguration Controller and part of the system with fault detection properties; the application system is partitioned and distributed among the available devices by exploiting the partitioner presented in the next chapter.

The module implementing the proposed Reconfiguration Controller is composed of the following elements, as shown in Figure 6.6:

- *Fault Classifier*, that implements the algorithm proposed for fault classification, as presented in Chapter 5,
- *Bitstream Address Calculator*, that identifies the memory position of the bitstream for recovering from the occurred fault,
- *Manager*, that moves the configuration data from memory to the interface for reconfiguration,
- *Bitstream Module*, that retrieves the bitstream from memory, and
- *Reconfiguration Interface*, that performs the physical implementation of the reconfiguration process.

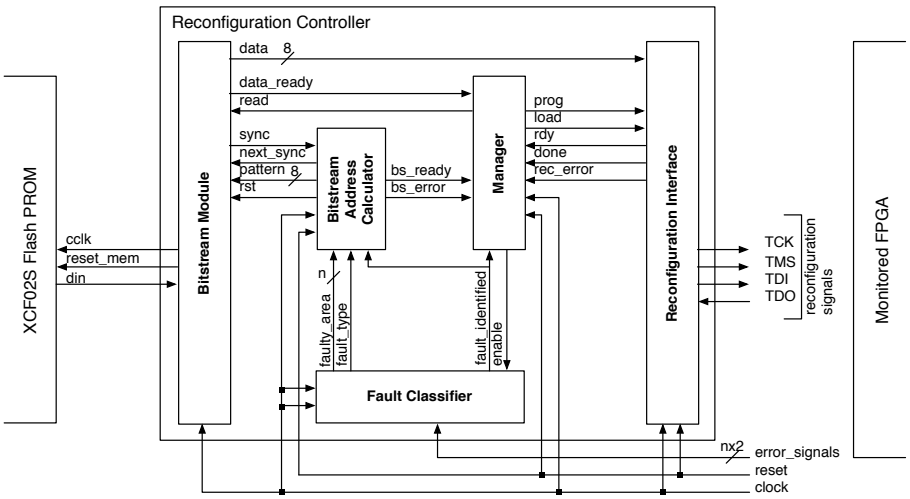


Figure 6.6.: Proposed Reconfiguration Controller block diagram.

6.4.1. Fault Classifier

The module implementing the fault classification algorithm receives in input the `error_signals` from the areas (application areas or controller)

and an `enable` signal specifying when the error signals are to be considered valid (the Fault Classifier is disabled during the recovery phase). We assume that on the error signals the areas adopt TRC, whereas the enable signal, generated within the Reconfiguration Controller, is not encoded since the presence of faults in the overall controller is revealed by an error signal generated by the controller itself. The Fault Classifier generates three outputs: `fault_identified` to signal whether a fault has been detected, `fault_type` to specify whether it is recoverable or not, and `faulty_area` to specify the area where the fault has occurred. The entire information is then used to trigger the appropriate reconfiguration/relocation strategy, managed by the module in charge of executing the re-programming of the neighbor FPGA. The input signals are used, together with the following local information, to classify the fault upon detection of an error condition on the input signals from the monitored areas: `last_ira` to specify the area identified as faulty the last time an error has been detected, `error_counter` to identify the number of consecutive errors detected on the area specified in the above mentioned register, and the parameter K to specify the threshold for classifying a fault as non recoverable. The resulting structure of the nominal Fault Classifier module is reported in Figure 6.7. It is possible to identify the internal signals used to characterize the situation, `error_detected` (when an area is identified as faulty), `same_area` (whether the area identified as faulty is the same as the last detected area), and `max_count` (whether the number of consecutive fault detections in the same area has reached the pre-defined threshold K). These signals are used to define the control of the entire module, described in terms of the FSM reported in Figure 6.8. The FSM generates the internal clock and reset signals (namely `clk` and `rst`) used to control the module evolution, and the (re)setting of the instantiated registers. A VHDL description of the module has been developed for simulation and synthesis purposes. Figure 6.9 reports the waveforms of the module behavior following the detection of a non-recoverable fault.

6.4.2. Bitstream Address Calculator

The Bitstream Address Calculator is in charge of identifying the memory position of the configuration data for fault recovery. When the `fault_identified` signal is asserted, the module locates the suitable recovery bitstream based on `fault_type` and `faulty_area` information. As soon as the memory position has been identified, the `bs_ready` signal is asserted. When the number of non-recoverable faults exceeds the supported threshold, the recovery bitstream can not be located; in this case,

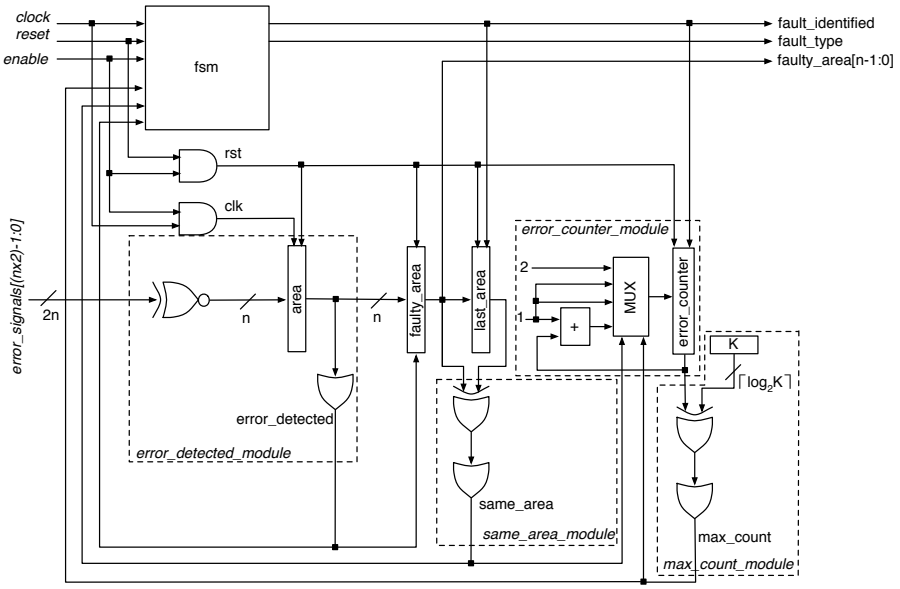


Figure 6.7.: Fault Classifier structure.

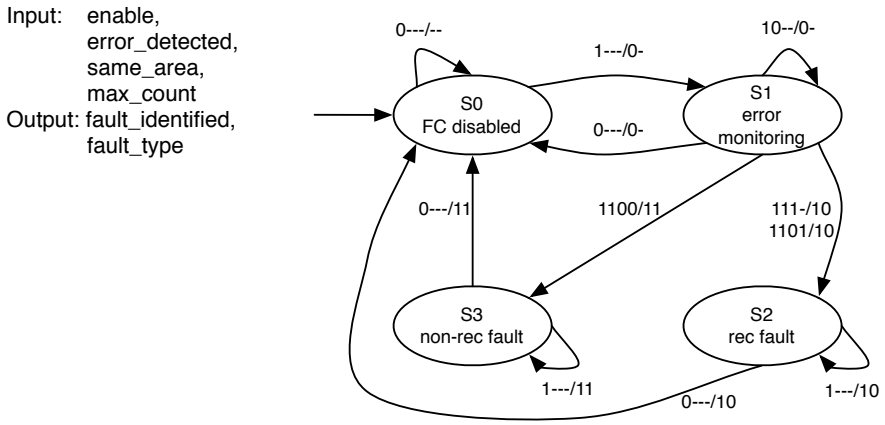


Figure 6.8.: Fault Classifier FSM.

the Bitstream Address Calculator signals an error by asserting `bs_error`.

6.4.3. Manager

The Manager is the module that, upon error detection, moves the bitstream retrieved by the Bitstream Module to the Reconfiguration Interface. When the Fault Classifier arises the `fault_identified` signal, the Manager disables the classifier module through the `enable` signal, indi-

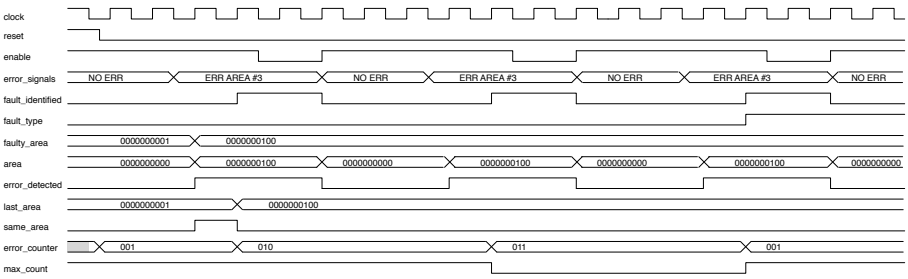


Figure 6.9.: Fault Classifier behavior when monitoring 10 areas and a non-recoverable fault occurs on area #3.

cating that the error signals are not to be considered valid during the recovery phase, and waits for the assertion of the `bs_ready` signal, specifying when the bitstream is ready to be retrieved. If the bitstream for recovering from the occurred fault can not be located, the `bs_error` signal is asserted by the Bitstream Address Calculator. Otherwise, as soon as the configuration data can be read, the module asserts the `prog` signal to begin the reconfiguration process and starts the bitstream transfer from the Bitstream Module to the Reconfiguration Interface. It manages the data move between the two modules by using the `read` and `data_ready` signals of the Bitstream Module and the `load` and `rdy` signals of the Reconfiguration Interface. When the transfer is completed, it de-asserts the `prog` signal and waits for the assertion of the `done` signal. If an error occurs during the reconfiguration process, the `rec_error` signal is asserted by the Reconfiguration Interface.

The Manager's behavior is described in terms of the FSM reported in Figure 6.10. The error signals from the Bitstream Address Calculator and the Reconfiguration Interface are not shown for sake of representation clearness. When the `bs_error` or `rec_error` signals are asserted, a transition from the current state to an error one is performed. Successive transitions from the error state are to itself. The error state indicates that the occurred fault cannot be recovered and the whole system is labeled as faulty.

6.4.4. Bitstream Module

The Bitstream Module is in charge of retrieving the recovery bitstream from memory. Based on the position information provided by the Bitstream Address Calculator and the reading request from the Manager, it retrieves the configuration data and sends them to the Reconfiguration Interface.

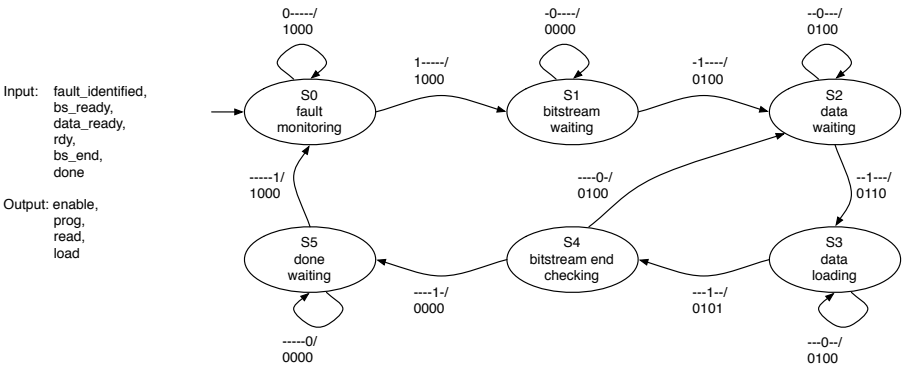


Figure 6.10.: Manager FSM.

To support the adopted prototype platform, where the recovery bitstreams are stored in a Xilinx configuration PROM, we exploited the PROM reader module presented in [67]. The recovery configuration data are stored in PROM as user-defined data separated by synchronization patterns, used by the module to locate the requested bitstream. This module can be replaced by another one accessing a different memory.

6.4.5. Reconfiguration Interface

The Reconfiguration Interface performs the physical implementation of the recovery process, by providing access to the JTAG interface, used in the prototype platform for managing reconfiguration. It handles the JTAG functions needed for programming. The module is controlled by the Manager as described in Section 6.4.3 and receives the configuration data from the Bitstream Module. It controls the JTAG interface signals to the configuration interface. Indeed, this module can be easily substituted to manage different configuration interfaces.

We recall that, in case of recoverable fault, the recovery action requires few milliseconds, whereas, in case of reallocation, few seconds are required to identify the suitable recovery bitstream and to perform reconfiguration.

6.5. Reconfiguration Controller hardening

The reliability of the Reconfiguration Controller must be guaranteed to block erroneous reconfigurations of the monitored FPGA and to include the module into an overall reliable system. Thus, it is necessary to detect faults affecting any portion of the controller and to guarantee the cor-

rectness of the reconfiguration process. In the following, the hardening of each component of the controller is analyzed and discussed. Particular attention has been devoted to the Fault Classifier, for which different hardened implementations have been analyzed, for the criticality of the classification task.

6.5.1. Fault Classifier hardening

Fault detection is required for the Fault Classifier and it can be considered sufficient since the engine is able to work correctly even when it is reset after reconfiguration, because of the robustness of the algorithm's parameters. In fact, a single fault in any one of the parameters being used by the classification algorithm can only slightly impact on the overall process. Should a fault occur, it will be detected and a reset (and reconfiguration) of the controller would only cause a minor delay in classifying a fault as not-recoverable. In case of recoverable faults, the most frequent ones, a loss of information has no negative effects. Otherwise, in the rare, worst case where a non-recoverable fault has occurred and has not been identified yet as such in the payload system, when the Fault Classifier is reset, the fault's observations required for correct fault classification are at most `2K-1` instead of `K` (we recall that `K` is the threshold for classifying a fault as non-recoverable). This is an acceptable value if considering that `K` assumes only small values and the latency estimated for the fault-error relation, now reduced since the threshold is increased, is high enough to give a wide margin for reduction. Hence, even the most critical registers for discriminating between faults, i.e., the ones related to the `error_counter` and the `last_ira`, when reset, do not compromise seriously the correctness of the classification. Therefore, we deemed sufficient to guarantee the detection of faults in this portion of the entire architecture. Nevertheless, for the sake of completeness, implementations achieving fault tolerance properties have been investigated. In particular, our analysis took into account the following hardened solutions: i) an implementation that applies Duplication With Comparison (DWC) on the module, ii) a Self-Checking (SC) implementation based on the application of error detection codes, iii) a Self-Checking implementation coupled with TMR applied to the most critical registers (SC+), and iv) an implementation where TMR is applied to the entire system with a fine granularity, by following the approach offered by Xilinx's TMRTool (X-TMR) [68].

Duplication With Comparison: DWC

In this implementation, two replicas of the Fault Classifier receive the same inputs and their outputs are compared by Two-Rail Code Checkers (TRCCs) for mismatch. The design is straightforward and no specific issues arise in the implementation of the solution, provided the hierarchy is guaranteed during the synthesis process. The tree of TRCCs is applied to the module's primary outputs, amounting to a total of `n+2` lines, being `n` the number of areas in the monitored FPGA. Such implementation is shown in Figure 6.11.

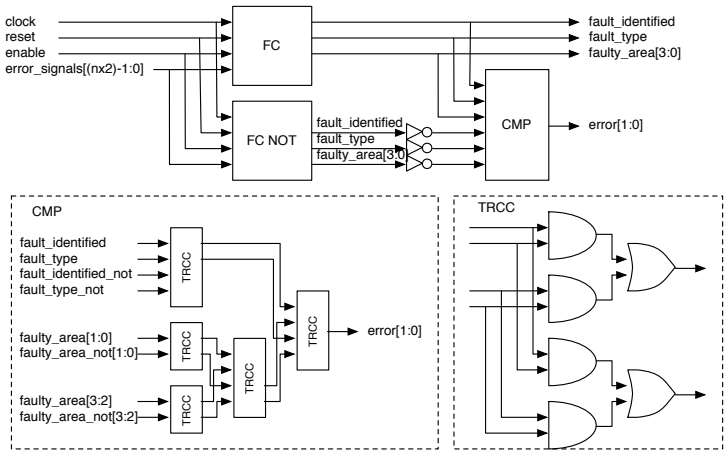


Figure 6.11.: Fault Classifier with DWC.

Self-Checking via Error Detecting Codes: SC

For the SC implementation, three classes of elements that constitute point of failures have been identified: i) the states of the FSM, ii) the internal variables of the datapath, and iii) the outputs. The analysis of the number of states of the control FSM and the values stored in the other registers allowed us to select the 1-hot code, to protect all the registers of the module, thus adopting a very well known (and thus easily implementable) encoding. In particular, since one of the monitored areas may be faulty at most, based on the adopted working hypotheses, the content of the `faulty_area` and `last_area` (from Figure 6.7) naturally exploits a 1-hot encoding. A similar analysis holds for register `area`, buffering the input error signals; eventually, it might contain an all 0s configuration, when no fault is detected on the monitored area. Therefore, by concatenating the complement of the `error_detected` signal with `area`, an immediate, cost-free 1-hot encoding is obtained. For example, when

monitoring 3 areas, if no fault is detected, the resulting code is `0001` since register `area` contains all 0s (no error is signalled) and the complement of `error_detected` is 1. On the other hand, the content of the register is `0010`, `0100` or `1000` when area #1, #2 or #3 is detected as faulty, respectively. Therefore, the 1-hot code is a natural choice for the specific register. For the other internal variables (state register and `error_counter`), since the number of states is limited, the code is not too expensive (the typical limitation of this code). The primary outputs are single lines (`fault_identified` and `fault_type`), encoded with a Two-Rail Code, whereas `faulty_area` is the encoded content of the register. The adoption of this code is particularly interesting, especially since the code is usually recognized and supported also by automatic synthesis tools, which usually have very little open configuration options, especially when targeting FPGA platforms, where the designer's control of the final implementation is hard to achieve. The SC implementation of the Fault Classifier is shown in Figure 6.12 and consists of the functional module generating encoded data and the necessary 1-hot/TRC checkers.

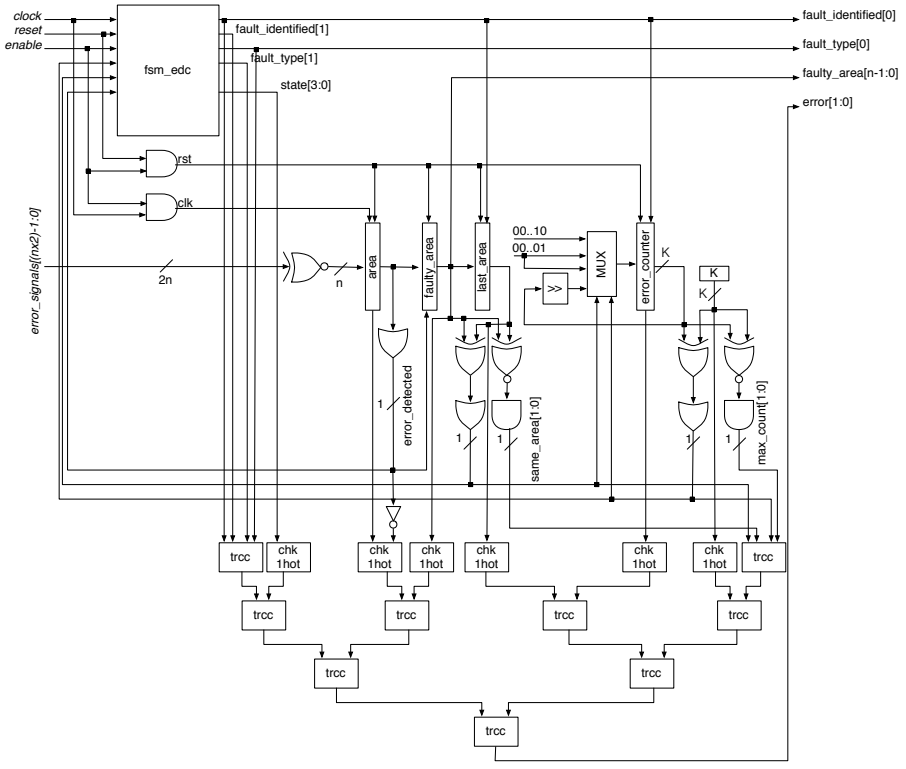


Figure 6.12.: SC Fault Classifier.

SC and TMR: SC+

A first fault tolerant implementation has been defined by protecting the most critical registers with TMR, using a feedback loop to propagate the correct value to the protected registers, thus avoiding any glitch in the stored values, especially useful for the case of non-recoverable faults. The resulting structure is reported in Figure 6.13, where the triplicated registers can be seen, as well as the additional voters, which not only perform a majority vote but also signal the presence of a mismatch on the inputs, for fault detection.

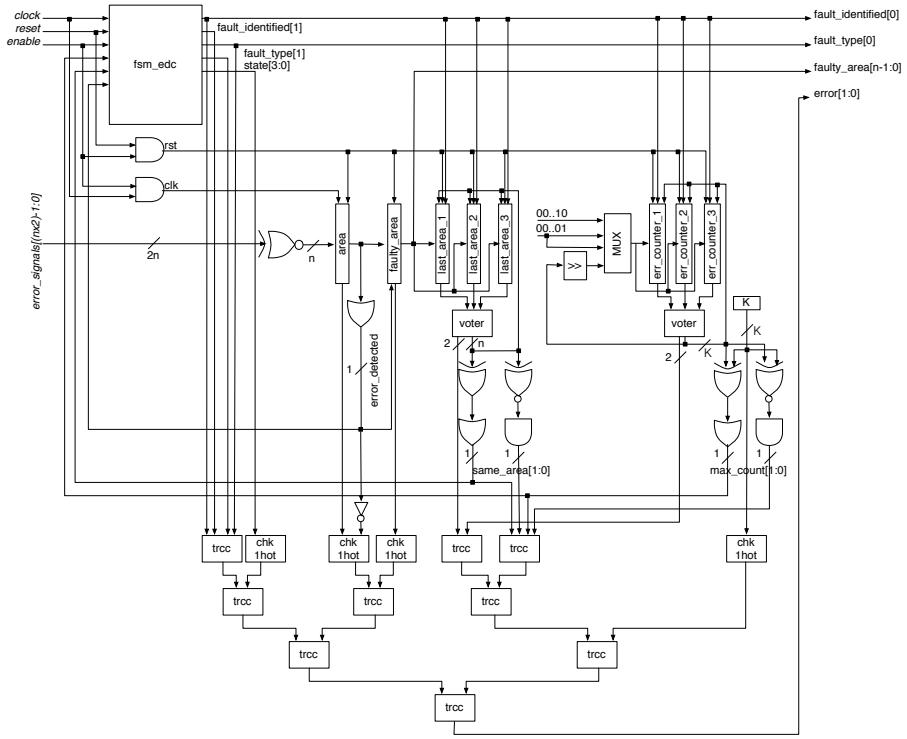


Figure 6.13.: SC Fault Classifier with TMR on the critical registers.

X-TMR

As a final alternative, we applied fault tolerance to the overall module with a fine granularity. We implemented the solution achieved by using X-TMR, that applies TMR to the entire system with a fine granularity. Unlike traditional TMR, X-TMR triplicates i) all inputs, including clocks and throughput logic, ii) feedback logic, and iii) all ouputs. It inserts

majority voters on feedback paths, and minority voters on outputs to detect and disable incorrect output paths. Such solution is the one usually adopted for reconfigurable FPGAs in high-radiation environments.

Cost analysis

The proposed implementations are compared in Table 6.1 with respect to their overheads, in terms of area occupation (number of used slices and flip flops). The selected device is a Xilinx Virtex-II XC2V1000. Each implementation has been analyzed by varying `K` and `n` as discussed in Chapter 5, to derive a trend in the implementation costs.

As expected, as the number of monitored areas `n` increases, the size of the Fault Classifier increases, in all nominal and hardened versions. The most expensive solution is the one obtained by using X-TMR, entailing overheads around 370%, as expected. For the other implementations, the incidence of the fault detection/tolerance added functionality is within the foreseen, typical bounds, keeping the final hardened implementation within acceptable costs, that is less than 2% of the entire FPGA resources. In particular, overheads range from 40% and about 100%, based on the different solutions, allowing the designer to select the solutions s/he deems more interesting.

Table 6.1.: Area occupation (slices and FFs – in parenthesis) of the Fault Classifier implementations and related overhead.

Solution	K=2 (n=15)		K=3 (n=10)		K=4 (n=5)		K=5 (n=3)	
	area occ.	over.	area occ.	over.	area occ.	over.	area occ.	over.
Nominal	86 (52)	-	60 (36)	-	40 (22)	-	29 (16)	-
DWC	176 (104)	105%	120 (72)	100%	81 (44)	102%	58 (32)	100%
SC	120 (53)	40%	96 (39)	60%	57 (25)	42%	47 (20)	62%
SC+	133 (53)	55%	107 (39)	78%	67 (25)	67%	58 (20)	100%
X-TMR	406 (201)	372%	283 (138)	372%	186 (81)	365%	141 (57)	386%

The SC implementation allows us to have a competitive solution, characterized by acceptable costs and benefits in terms of reliability. It has been compared to the nominal solution in Table 6.2, in terms of area occupation of the modules composing the Fault Classifier (see Figure 6.7). Not all modules entail an increment in cost, in particular `error_detected_module` and the registers `faulty_area` and `last_area`, that do not require modifications in the SC version. Alternative solutions, with acceptable but higher costs, do not entail significative benefits in terms of additional reliability.

Table 6.2.: Area occupation (slices and FFs – in parenthesis) of the Fault Classifier modules for the nominal version and the SC one.

Module	K=2 (n=15)		K=3 (n=10)		K=4 (n=5)		K=5 (n=3)	
	Nom.	SC	Nom.	SC	Nom.	SC	Nom.	SC
fsm	9 (4)	14 (6)	9 (4)	14 (6)	9 (4)	14 (6)	9 (4)	14 (6)
error_detected_module	27 (15)	27 (15)	18 (10)	18 (10)	13 (10)	13 (10)	8 (6)	8 (6)
faulty_area	15 (15)	15 (15)	10 (10)	10 (10)	5 (5)	5 (5)	3 (3)	3 (3)
last_area	15 (15)	15 (15)	10 (10)	10 (10)	5 (5)	5 (5)	3 (3)	3 (3)
same_area_module	4 (0)	8 (0)	3 (0)	4 (0)	2 (0)	2 (0)	1 (0)	2 (0)
error_counter_module	5 (3)	3 (2)	5 (3)	5 (3)	5 (3)	6 (4)	5 (3)	8 (5)
max_count_module	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	1 (0)	2 (0)

6.5.2. Other components hardening

The hardening of the other modules composing the Reconfiguration Controller requires the introduction of fault detection and tolerance properties. For the Reconfiguration Interface, fault tolerance is required since it is necessary to guarantee the correctness of the reconfiguration. For the other components, i.e., Manager, Bitstream Address Calculator, and Bitstream Module, fault detection suffices since by detecting erroneous situations, it is possible to block erroneous reconfigurations. Thus, the selected hardening strategy applies TMR on the Reconfiguration Interface and DWC on the other components. The costs of the hardened implementation are reported in Table 6.3. As for the Fault Classifier's cost analysis, the selected device is a Xilinx Virtex-II XC2V1000.

Table 6.3.: Area occupation of the Reconfiguration Controller components' hardened implementations.

Component	Slices	FFs
Reconfiguration Interface	909	723
Manager	26	10
Bitstream Address Calculator	28	20
Bitstream Module	142	112

6.6. Chapter summary

The chapter has presented the design of the Reconfiguration Controller, i.e., the module in charge of managing the fault recovery of the multi-FPGA platform. We introduced a controller performing the following main tasks: i) continuous check of the error signals from the monitored FPGA, ii) classification of fault into recoverable or non-recoverable, and iii) management of the reconfiguration process to mitigate fault effects. Since the module will be hosted on an SRAM-based FPGA, it has been designed to be reliable itself, to prevent erroneous reconfigurations by identifying an erroneous behavior of the controller as soon as it is observable. A preliminary software implementation has been developed to refine the controller behavior. Due to the main limitations characterizing a software implementation, a hardware implementation has been preferred, and two reliability-aware solutions have been proposed.

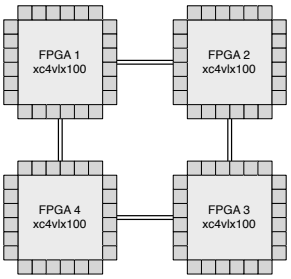
7. Methodology evaluation

The reliability-aware design methodology presented in the thesis has been evaluated by implementing a real case study. The design flow described in Chapter 4 has been followed, and the developed prototype framework has allowed to automate, as much as possible, the system implementation. Different solutions have been explored, and the one offering the most convenient trade-off with respect to the designer's selected metrics has been identified.

The chapter is structured as follows: Section 7.1 describes the selected case study and Sections from 7.2 to 7.5 present the experimental results for the various tasks of the design flow. Sections 7.6 and 7.7 compare the results with the ones obtained by implementing an alternative version of the proposed design flow and the flow for fixed number of non-recoverable faults, respectively. Finally, Section 7.8 draws the chapter summary.

7.1. Case study

The multi-FPGA platform selected for the evaluation of the design methodology is a commercial board, a Synopsis HAPS-34 [25]. It is composed of four Xilinx FPGAs xc4vlx100, interconnected to each other in a mesh topology through connections for communication and configuration. Figure 7.1 shows the platform model, together with the parameters characterizing it. We recall that the platform model is composed of identical FPGAs, $\mathtt{d}_i \in \mathtt{D}$, where `D` is the set of devices. Each FPGA is characterized by the resources hosted on it, $\mathtt{dev\_res}_r$, $\mathtt{r} \in \mathtt{R} = \{\mathtt{slice}, \mathtt{bram}, \mathtt{dsp}\}$, that is the number of resources in terms of slices, BRAMs, and DSPs, respectively. The FPGA is shaped as a grid of slices with columns constituted by different types of resources. The inter-FPGA communication is performed by dedicated wires; $\mathtt{dev\_wires}_{d_1,d_2}$ is the number of wires between devices $\mathtt{d}_1$ and $\mathtt{d}_2$, and `tot_wires` is the number of total external wires on the platform. The platform topology is modeled by specifying the devices' adjacency and the communication paths between non adjacent devices; $\mathtt{direct\_comm}_{d_1,d_2}$ equals to 1 if devices $\mathtt{d}_1$ and $\mathtt{d}_2$ are adjacent, 0 otherwise, and $\mathtt{non\_direct\_comm}_{d_1,d_2,d_3}$ equals to 1 if device $\mathtt{d}_1$ allows the communication between devices $\mathtt{d}_2$ and $\mathtt{d}_3$, 0 otherwise.



$dev\_wires_{i,j}$

i \ j	1	2	3	4
1	0	729	0	149
2	729	0	149	0
3	0	149	0	729
4	149	0	729	0

tot_wires	1756

$direct\_comm_{i,j}$

i \ j	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

Device's resources	
dev_res_{slice}	49152
dev_res_{bram}	240
dev_res_{dsp}	96

$non\_direct\_comm_{i,j,k}$

i=1	1	2	3	4	i=2	1	2	3	4
1	0	0	0	0	1	0	0	1	0
2	0	0	0	0	2	0	0	0	0
3	0	0	0	0	3	0	0	0	0
4	0	1	0	0	4	0	0	0	0
i=3	**1**	**2**	**3**	**4**	**i=4**	**1**	**2**	**3**	**4**
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	2	0	0	0	0
3	0	0	0	0	3	1	0	0	0
4	0	0	0	0	4	0	0	0	0

Figure 7.1.: Multi-FPGA platform model.

The selected case study considers a circuit that performs image edge detection, to be used by an overall control system for people/objects recognition. It detects the image edges and converts the image to transfer it to different devices. In a first phase, the edges are detected and overlapped on the image. Then, the obtained raw bitmap image is encoded into a JPEG compliant coded bitstream, selected as standard format. Finally, as color conversion is necessary when transferring data between devices that use different color models, the image is converted to another color system. In Figure 7.2(a), the nominal circuit structure is shown, annotated with the requirements $<\texttt{area\_wires}_{a_1,a_2}, \texttt{area\_comm}_{a_1,a_2}>$ for the communication between modules; the throughput $\texttt{area\_comm}_{a_1,a_2}$ is identified by the number of bits processed at a time. The resource requirements of the circuit are reported in Table 7.1, by considering the implementation onto a Xilinx FPGA xc4vlx100 device. It is worth noting that a hardened implementation of the circuit, for example by means of the classical TMR technique, could not be hosted onto a single FPGA for lack of resources, hence multiple FPGAs should be exploited.

By considering the selected multi-FPGA platform and circuit, the proposed reliability-aware design flow, shown in Figure 7.3, has been applied, as described in the following.

7.2. Preliminary partitioning

The circuit components have been distributed among the devices available on the selected multi-FPGA platform.

We recall that the partitioning is a multi-objective problem, considering as metrics the distribution uniformity and the minimization of the external communication both in terms of wires and throughput. The

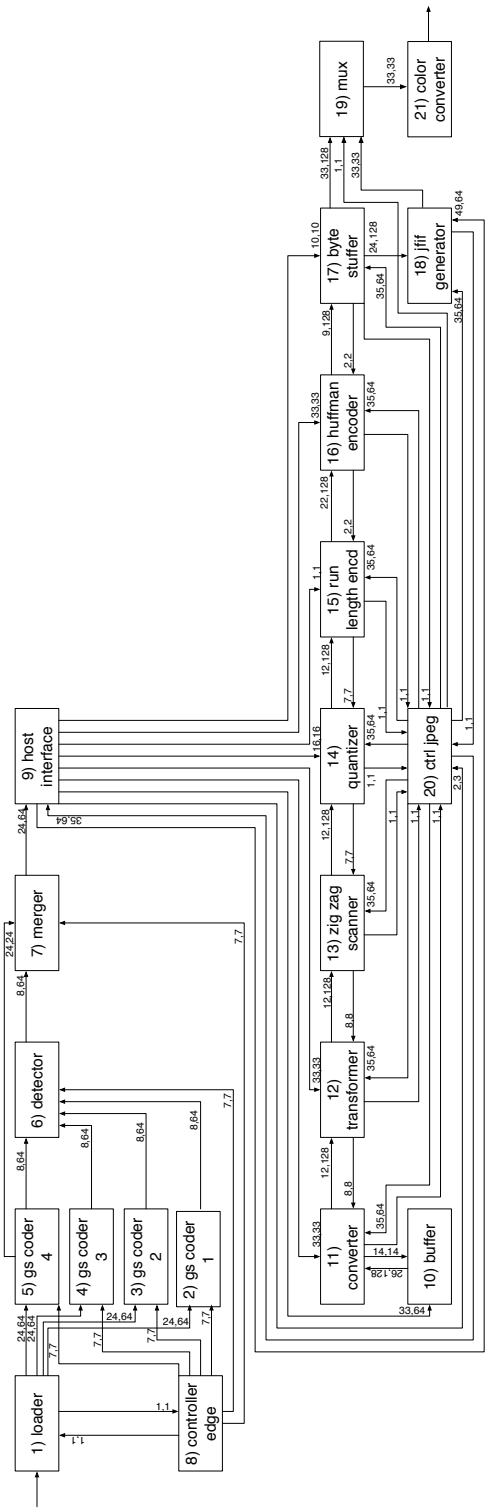


Figure 7.2.: Case study circuit.

Table 7.1.: Resource requirements of the case study circuit.

#	Component	#Slices	#BRAMs	#DSPs
1	loader	2341	12	-
2	gs coder 1	130	-	-
3	gs coder 2	135	-	-
4	gs coder 3	120	-	-
5	gs coder 4	105	-	-
6	detector	41	-	-
7	merger	17	-	-
8	controller edge	12	-	-
9	host interface	160	-	-
10	buffer	2212	12	-
11	converter	3512	-	-
12	transformer	2046	-	-
13	zig zag scanner	46	2	-
14	quantizer	76	3	1
15	run length encoder	266	2	-
16	huffman encoder	3560	-	-
17	byte stuffer	63	-	-
18	jfif generator	89	1	-
19	mux	19	-	-
20	controller jpeg	200	-	-
21	color converter	2435	-	-
Total		*17585*	*32*	*1*

adopted objective function is the following:

$$min(w_{gap} \cdot \sum_{r \in R} \frac{Gap_r}{dev\_res_r} + w_{wires} \cdot \frac{Wires}{tot\_wires} + w_{comm} \cdot \frac{Comm}{\sum_{c1 \in C} \sum_{c2 \in C} comm_{c1,c2}})$$

In the preliminary partitioning task, we aim at achieving a solution uniformly distributed among the available FPGAs, to have balanced ratio between used and spare fabric for applying the hardening techniques, thus preferring the distribution uniformity with respect to the other metrics. A tuning of the weights for the objective function has been performed, as reported in Appendix B, and the following weights have been adopted: $\mathtt{w}_{gap} = 0.7$, $\mathtt{w}_{wires} = 0.1$, and $\mathtt{w}_{comm} = 0.2$.

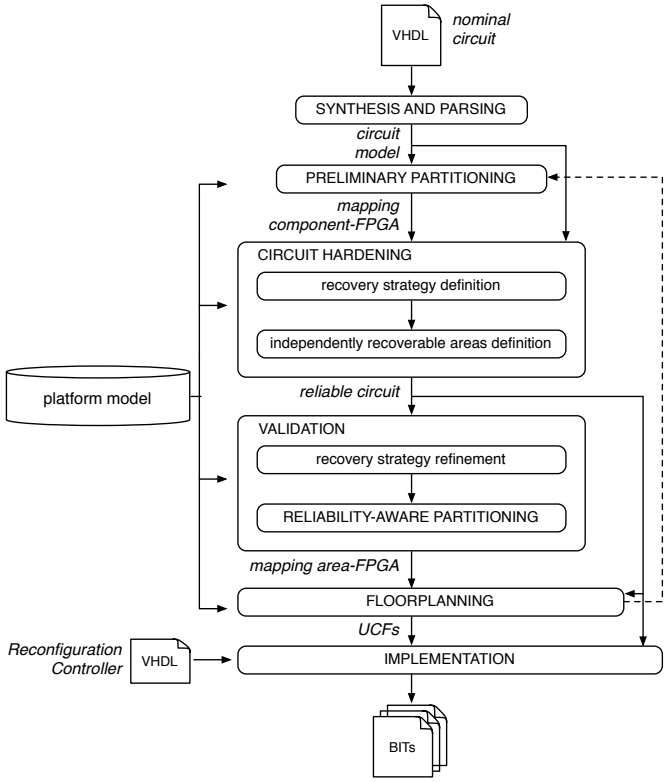


Figure 7.3.: Proposed design flow.

The output of the partitioning is shown in Figure 7.4, together with the parameters characterizing the identified solution. The proposed approach achieves a uniform distribution of the circuit on the platform, also taking into account the external communication both in terms of wires and throughput. The partitioner's execution time for distributing the case study circuit among the four available devices is 33 s.

7.3. Circuit hardening

After partitioning the circuit among the available FPGAs, each subcircuit is hardened by applying TMR to single components or groups of components, defining independently recoverable areas as described in the following.

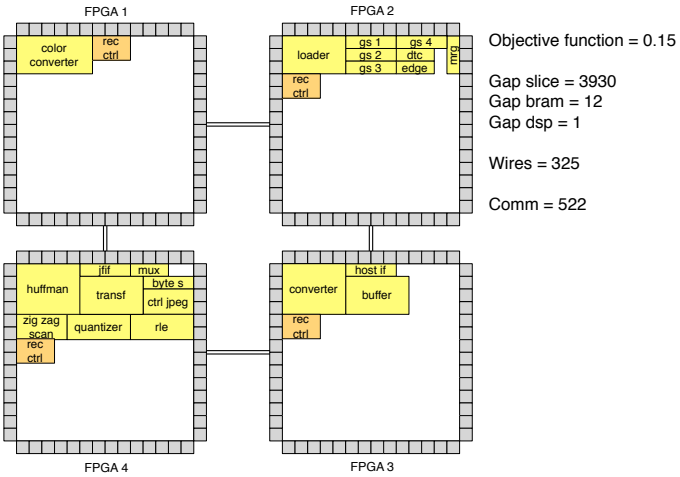


Figure 7.4.: Case study circuit partitioning on the selected multi-FPGA platform.

7.3.1. Recovery strategy definition

In the first activity of the circuit hardening task, the number of non-recoverable faults tolerated by the system on each FPGA (`#faults`) and the maximum number of independently recoverable areas each sub-circuit can be divided into (`max_areas`) are computed. We hereafter recall how to define these parameters.

To compute `#faults`, a hardened version of the circuit is taken into account, by partitioning each component in a different area and by applying the TMR hardening technique. Table 7.2 reports the resource requirements of the hardened components. The maximum number of tolerated non-recoverable faults for each FPGA is the number of times the greatest area hosted on the device can be moved onto a not used, fault-free region; Table 7.3 reports the parameters $\texttt{max\_faults}_d$ computed for each FPGA `d`. Thus, `#faults`, being the minimum value among the identified numbers of tolerated non-recoverable faults, is 2.

To fulfill the memory constraint, a maximum number of independently recoverable areas `max_areas` the sub-circuits can be divided into must be defined; in fact, a too large memory should be required to store a recovery bitstream for each component. `max_areas` is set as follows:

$$max\_areas = \max\left(\#areas \mid \sum_{i=0}^{\#faults} \#areas^i \leq \frac{memory\_size}{bitstream\_size}\right)$$

being `memory_size` and `bitstream_size` the size of the memory and of the bitstream for the selected device, respectively. The size of the

Table 7.2.: Resource requirements of the case study circuit's hardened components.

Component	#Slices	#BRAMs	#DSPs
loader	7411	36	-
gs coder 1	422	-	-
gs coder 2	437	-	-
gs coder 3	392	-	-
gs coder 4	347	-	-
detector	155	-	-
merger	147	-	-
controller edge	220	-	-
host interface	1320	-	-
buffer	6740	36	-
converter	10640	-	-
transformer	6222	-	-
zig zag scanner	222	6	-
quantizer	308	9	3
run length encoder	918	6	-
huffman encoder	10728	-	-
byte stuffer	333	-	-
jfif generator	403	3	-
mux	189	-	-
controller jpeg	1864	-	-
color converter	7437	-	-
Total	*17585*	*32*	*1*

Table 7.3.: Definition of the parameters $\texttt{max\_faults}_d$ for each FPGA `d`.

Parameter	FPGA 1	FPGA 2	FPGA 3	FPGA 4
$\texttt{max\_faults}_d$	5	5	2	2

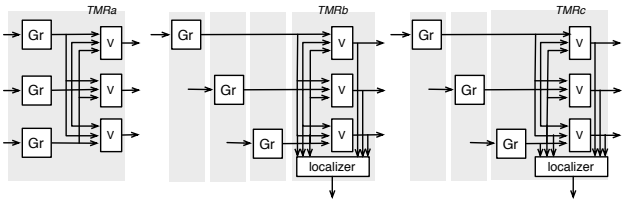


Figure 7.5.: TMR technique: different application schemas [1].

memory selected for storing the recovery bitstreams is 256 MB and the size of the bitstream for the considered FPGA (xc4vlx100) is 3.66 MB. Thus, `max_areas` equals to 7.

7.3.2. Independently recoverable areas definition

By considering the parameter `max_areas` defined in the previous activity, for each FPGA on the platform, the components of the sub-circuit hosted on the device are grouped and TMR is applied on each identified group. We recall that four different strategies of TMR application can be adopted on each group, containing one or more components, as shown in Figure 7.5:

- TMR on 1 area, mapping the whole system (replicas and voter) on a single area,
- TMR on 2 areas, mapping two replicas on an area and the third replica and the voter on another area,
- TMR on 3 areas, mapping each replica on a different area and the voter with one of the replicas, and
- TMR on 4 areas, mapping each replica and the voter on four different areas.

We have implemented the optimization model presented in Chapter 4, defining the independently recoverable areas composing the hardened circuit. Table 7.4 reports, for each FPGA, the identified groups, together with the contained components, the independently recoverable areas of each group, and the resource requirements of each area. Based on the evaluation presented in Appendix D, the following weights have been adopted for hardening the sub-circuits: $\mathtt{w}_{res} = 0.6$, $\mathtt{w}_{areas} = 0.3$, and $\mathtt{w}_{wires} = 0.1$. Thus, the distribution uniformity is achieved, also taking

Table 7.4.: Definition of the independently recoverable areas for each FPGA.

FPGA 1					
Group	Components	Area	#Slices	#BRAMs	#DSPs
1	color converter	1	2435	-	-
		2	2435	-	-
		3	2567	-	-
FPGA 2					
Group	Components	Area	#Slices	#BRAMs	#DSPs
2	loader, gs coder 1, 2, 3, 4,	1	2901	12	-
	detector, merger	2	2901	12	
	controller edge	3	2997	12	
FPGA 3					
Group	Components	Area	#Slices	#BRAMs	#DSPs
3	host interface, buffer	1	2372	12	-
		2	2372	12	-
		3	3184	12	-
4	converter	1	3512	-	-
		2	3512	-	-
		3	3620	-	-
FPGA 4					
Group	Components	Area	#Slices	#BRAMs	#DSPs
5	transformer, zig zag scanner,	1	2805	8	1
	quantizer, run length encoder	2	2805	8	1
	byte stuffer, jfif generator,	3	3345	8	1
	mux, controller jpeg				
6	huffman	1	3560	-	-
		2	3560	-	-
		3	3608	-	-

into account the maximization of the number of areas, as required to apply the fault classification algorithm. Moreover, the minimization of the external wires is considered, to avoid solutions with groups where components are not connected to each other. Figure 7.6 shows the identified 6 groups, each one composed of one or more independently recoverable areas, with a total of 18 areas. The average execution time for hardening each sub-circuit is 3 s.

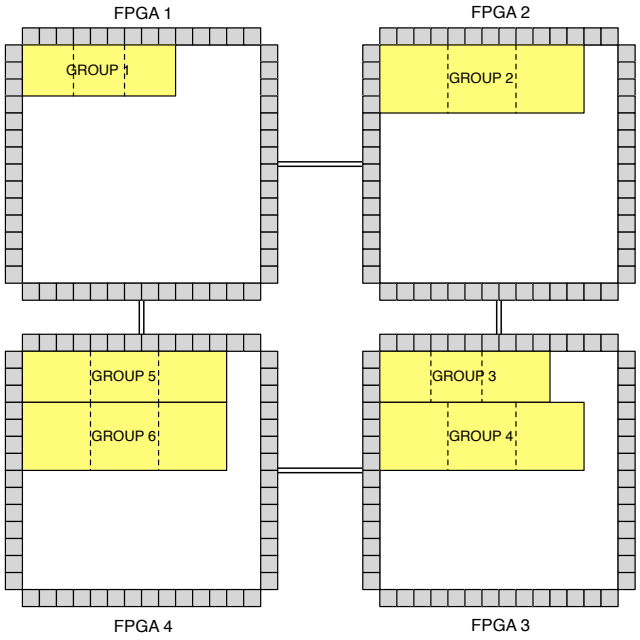


Figure 7.6.: Groups composing the hardened circuit.

7.4. Validation

The validation task aims at improving the obtained reliable circuit. The two activities composing the task are described in the following.

7.4.1. Recovery strategy refinement

By taking into account the defined hardened circuit, the number of tolerated non-recoverable faults is re-evaluated.

For each FPGA `d`, the maximum number of tolerated non-recoverable faults ($\texttt{max\_faults}_d$) is computed by considering the number of times the greatest independently recoverable area hosted on the device can be relocated on the spare region. Then, the constraint imposed by the memory capacity is taken into account and the actual number of non-recoverable faults tolerated by the system on each FPGA is defined as follows:

$$\#faults = \min\left(\#faults_d \mid \sum_{i=0}^{\#faults_d} \#areas_d^i \leq \frac{memory\_size}{bitstream\_size}, \forall d \in D\right)$$

where $\#\texttt{areas}_d$ is the number of independently recoverable areas hosted on FPGA `d` and $\#\texttt{faults}_d \leq \texttt{max\_faults}_d$ is the tolerated number of faults.

Table 7.5 reports the values of $\texttt{max\_faults}_d$ and $\#\texttt{faults}_d$ for each device. For the selected case study, the identified number of tolerated non-recoverable faults is $\#\texttt{faults} = 2$, confirming the preliminary evaluation.

Table 7.5.: Definition of the parameters $\texttt{max\_faults}_d$ and $\#\texttt{faults}_d$ for each FPGA $\texttt{d}$.

Parameter	FPGA 1	FPGA 2	FPGA 3	FPGA 4
$\texttt{max\_faults}_d$	13	14	8	8
$\#\texttt{faults}_d$	3	3	2	2

7.4.2. Reliability-aware partitioning

By considering the hardened circuit defined by the previous task, the independently recoverable areas are re-distributed among the FPGAs, with the objective of obtaining a possible better partitioning. Indeed, the areas belonging to the same group are constrained to be hosted on the same FPGA, by opportunely setting the parameters $\texttt{same\_dev}_{c_1,c_2}$, $\texttt{c}_1$, $\texttt{c}_2 \in \texttt{C}$, expressing whether components $\texttt{c}_1$ and $\texttt{c}_2$ must be on the same device. We implemented the reliability-aware partitioner based on the optimization model presented in Chapter 4. The partitioner is an extension of the preliminary one, considering also the possible recovery actions.

Since in the preliminary partitioning task we have privileged the distribution uniformity with respect to the other metrics, here we have preferred the minimization of the external communication, by considering the following weights: $\texttt{w}_{gap} = 0$, $\texttt{w}_{wires} = 0.5$, and $\texttt{w}_{comm} = 0.5$. The obtained partitioning is shown in Figure 7.7, together with the parameters characterizing it. The solution exploits only three FPGAs, sufficient for hosting the system and reserving the spare region for the recovery actions. The partitioner's execution time for identifying the solution is 6 s.

When considering also the distribution uniformity, the same solution obtained by the preliminary partitioner is achieved. The selected weights are the following: $\texttt{w}_{gap} = 0.3$, $\texttt{w}_{wires} = 0.4$, and $\texttt{w}_{comm} = 0.3$. As shown in Figure 7.8, that reports the solution together with the parameters characterizing it, all the devices available on the platform are exploited when also the distribution uniformity is taken into account. For this

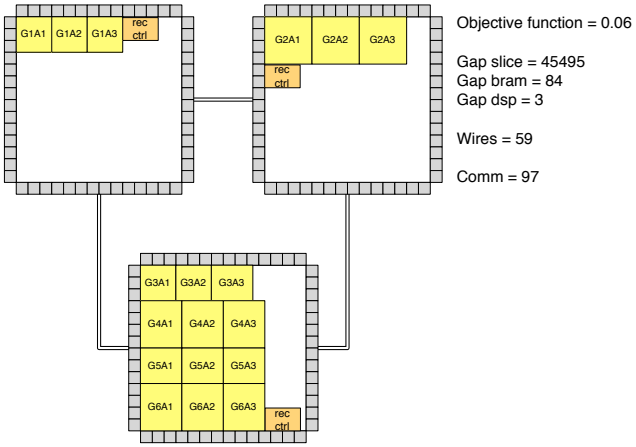


Figure 7.7.: Hardened circuit partitioning on the selected multi-FPGA platform when privileging the minimization of the external communication.

solution, taken into account for the subsequent step of the design flow, the partitioner's execution time is 12 s.

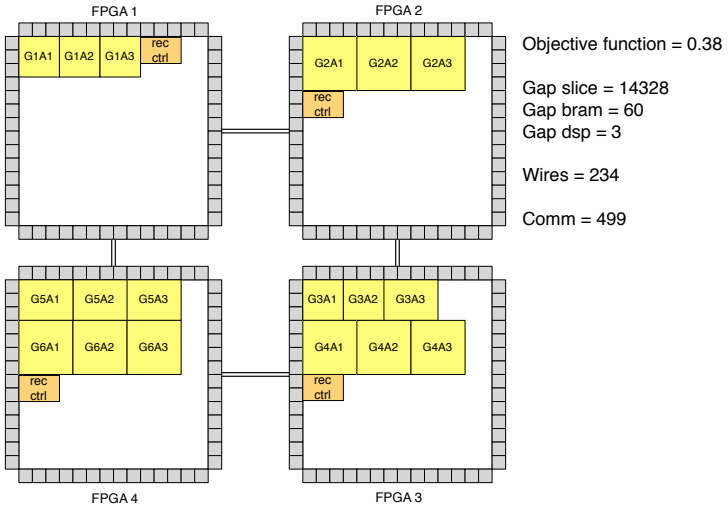


Figure 7.8.: Hardened circuit partitioning on the selected multi-FPGA platform when considering a trade-off of the metrics.

7.5. Floorplanning

The floorplanning activity positions each hardened sub-circuit within the assigned FPGA. Each FPGA has been modeled as a grid of 49152 slices (384 rows and 128 columns), with five columns of BRAMs and one of DSPs, according to the adopted FPGA device.

Figure 7.9 shows the floorplan of each sub-circuit. This task confirms the feasibility of the identified solution and produces the positioning constraints for implementing the system on the multi-FPGA platform.

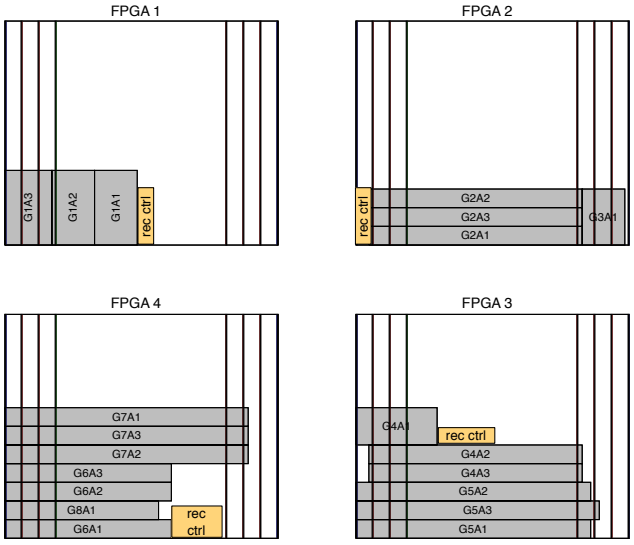


Figure 7.9.: Floorplan of each hardened sub-circuit on the related FPGA.

Finally, the defined hardened circuit is integrated with the Reconfiguration Controllers (one for each FPGA), creating the envisioned reliable system. The design flow has required almost one minute for performing the various explorations, namely partitioning, independently recoverable areas definition, and floorplanning. It produces the configurations for implementing the system on the FPGAs and the recovery bitstreams.

7.6. Alternative design flow

The design flow proposed in the thesis defines the reliable solution by performing a design space exploration composed of activities in the following order: i) preliminary partitioning, ii) hardening, iii) reliability-aware partitioning, iv) validation, and v) floorplanning. It can be noted that the hardening activity can be performed before the partitioning one,

thus an alternative design flow can be considered. In this section, the alternative flow shown in Figure 7.10 is implemented and the obtained solution is compared with the one of the proposed flow.

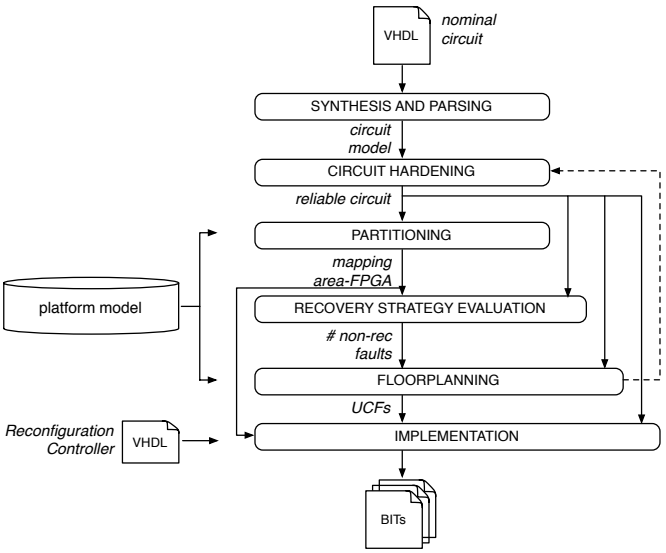


Figure 7.10.: Alternative design flow.

The first activity of the alternative design space exploration performs the hardening of the circuit. Since the proposed hardening optimization model supports circuit with up to 10 components, as reported in Appendix C, the circuit has been analyzed to reduce the number of components to be hardened. The preliminary grouping shown in Figure 7.11 has been identified, based on the components' resource requirement and interconnections. A minimum number of 12 areas is required, by considering a minimum of 3 areas for each FPGA to apply the fault classification algorithm. The hardening activity has been performed by adopting the following weights for the objective function of the optimization model: $\mathtt{w}_{res} = 0.8$, $\mathtt{w}_{areas} = 0$, and $\mathtt{w}_{wires} = 0.2$; the distribution uniformity is achieved, also taking into account the minimization of the external wires. Table 7.6 reports the hardened groups identified by the activity, together with the contained components, the independently recoverable areas of each group, and the resource requirements of each area. The solution is composed of 6 groups, each one composed of 3 independently recoverable areas, with a total of 18 areas.

The obtained hardened circuit has been distributed among the available FPGAs. We aimed at achieving distribution uniformity, that has been preferred with respect to the other metrics, to have balanced ra-

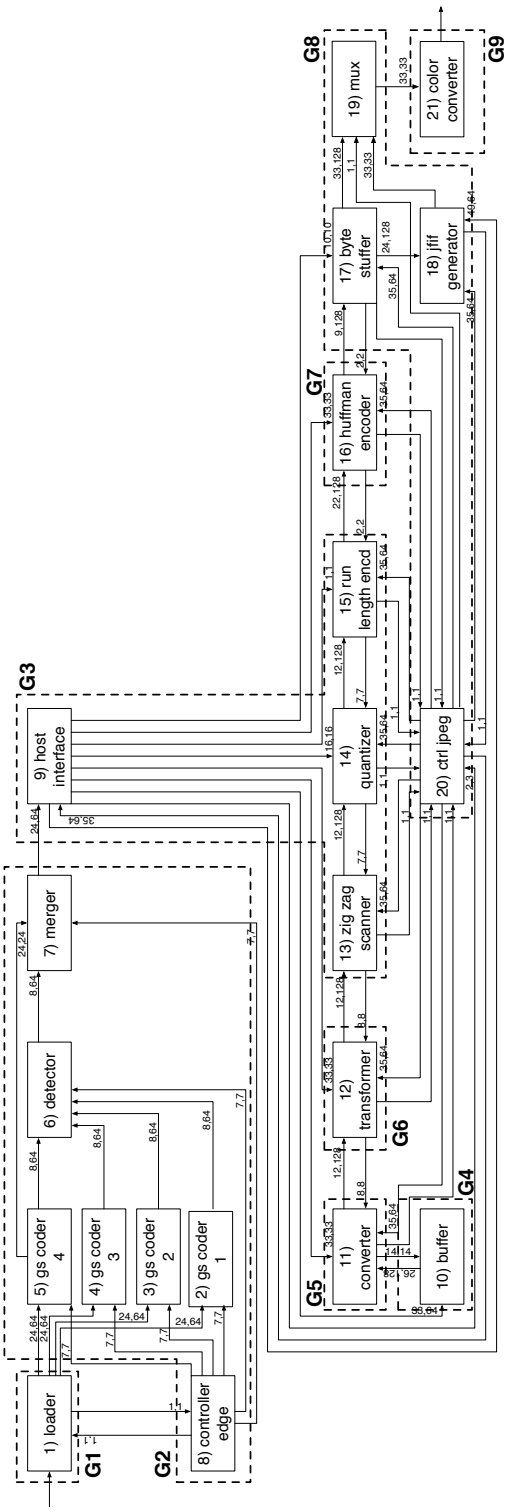


Figure 7.11.: Preliminary grouping on case study circuit.

Table 7.6.: Alternative design flow: Definition of the independently recoverable areas.

Group	Components	Area	#Slices	#BRAMs	#DSPs
1	G1, G2	1	2901	12	-
		2	2901	12	-
		3	2997	12	-
2	G3, G6, G8	1	2965	8	1
		2	2965	8	1
		3	3813	8	1
3	G4	1	2212	12	-
		2	2212	12	-
		3	2316	12	-
4	G5	1	3512	-	-
		2	3512	-	-
		3	3620	-	-
5	G7	1	3560	-	-
		2	3560	-	-
		3	3608	-	-
6	G9	1	2435	-	-
		2	2435	-	-
		3	2567	-	-

tio between used and spare fabric for recovery actions. The following weights have been adopted for the objective function of the optimization model: $\mathtt{w}_{gap} = 0.8$, $\mathtt{w}_{wires} = 0.1$, and $\mathtt{w}_{comm} = 0.1$. The output of the partitioning is shown in Figure 7.12, together with the parameters characterizing the identified solution. Only three FPGAs are exploited and no gap smaller than 19527 in terms of slices can be achieved due to the availability of external wires, which constrains the distribution of the components.

For each FPGA, the number of tolerated non-recoverable faults has been evaluated. Table 7.7 reports the values of i) $\mathtt{max\_faults}_d$, computed by considering the number of times the greatest independently recoverable area hosted on the device can be relocated on the spare region, and ii) $\#\mathtt{faults}_d$, computed by taking into account the constraint imposed by the memory capacity. The identified number of non-recoverable

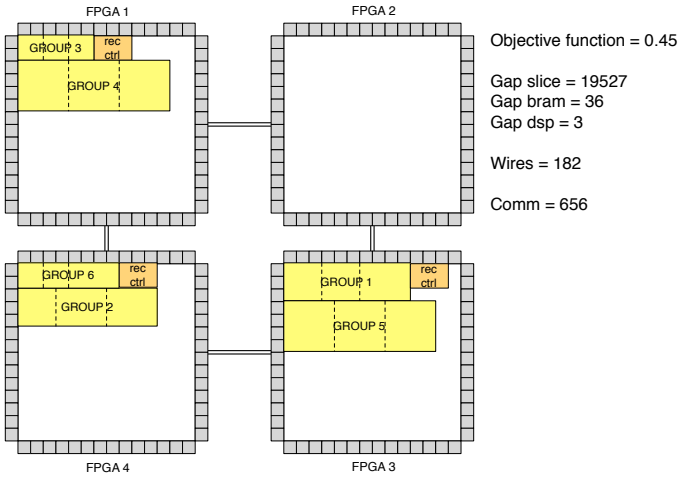


Figure 7.12.: Partitioning of hardened circuit in alternative design flow.

faults tolerated by the system is $\#\texttt{faults} = 2$.

Table 7.7.: Definition of the parameters $\texttt{max\_faults}_d$ and $\#\texttt{faults}_d$ for each FPGA $\texttt{d}$ in alternative design flow.

Parameter	FPGA 1	FPGA 2	FPGA 3	FPGA 4
$\texttt{max\_faults}_d$	8	-	8	8
$\#\texttt{faults}_d$	2	-	2	2

Finally, the floorplanning activity has positioned each sub-circuit within the assigned FPGA, as shown in Figure 7.13.

It can be noted that, by following the alternative design flow, we can not exploit all the devices available on the platform. In fact, the availability of inter-FPGA connections on the platform constrains the distribution of the hardened areas, indeed characterized by higher communication requirements with respect to single nominal components. Moreover, it is worth noting that the execution of the hardening activity before the partitioning one does not allow to define the number of areas (and consequently the error detection granularity) based on the evaluation of the recovery actions; the hardening activity in the proposed design flow can be guided by the recovery strategy definition. Thus, we deem that the proposed design flow can achieve better solutions than the alternative one.

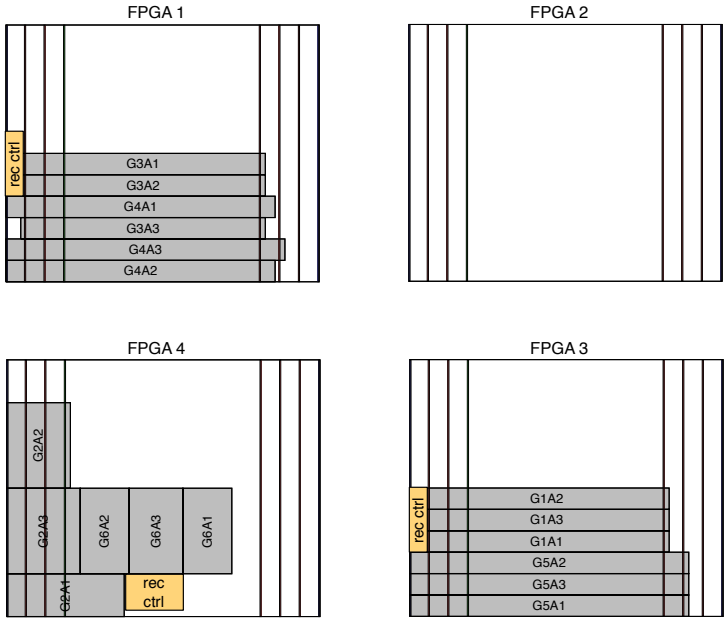


Figure 7.13.: Floorplan of each hardened sub-circuit on the related FPGA in alternative design flow.

7.7. Design flow for fixed number of non-recoverable faults

Throughout this chapter, the proposed design methodology has been evaluated by building the envisioned fault tolerant system on a multi-FPGA platform specified by the designer. Alternatively, it is possible to define the reliable system based on the number of non-recoverable faults to be tolerated, by following the design flow shown in Figure 7.14. In this section, we follow this alternative flow, by considering two non-recoverable faults to be tolerated on each FPGA.

The circuit components are taken in input by the reliability-aware iterative partitioning task, that distributes them among the necessary FPGAs. We recall that the partitioner takes into account a hardened version of the circuit, where TMR is applied at component-level, and, by starting from two FPGAs, increases the number of required devices as far as it identifies the suitable spare region for tolerating the non-recoverable faults. The partitioning of the selected case study is shown in Figure 7.15. Two FPGAs are sufficient to tolerate two non-recoverable faults on each device.

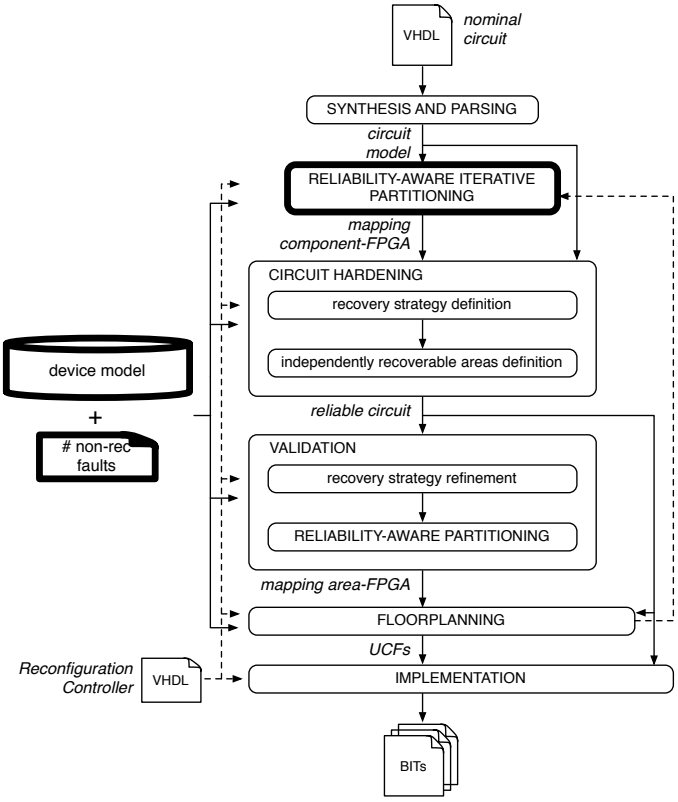


Figure 7.14.: Design flow for fixed number of non-recoverable faults.

7.8. Chapter summary

In this chapter, we have evaluated the reliability-aware design methodology presented in the thesis by implementing a real case study onto a commercial multi-FPGA platform. The design flow introduced in Chapter 4 has been followed, and the developed prototype framework has allowed to automate, as much as possible, the system implementation. Different solutions have been explored and compared, by opportunely setting the parameters of the exploited optimization models. The performed experimental session has allowed to evaluate and refine each task of the flow, in particular by tuning and analyzing the parameters. The experimental results, reported for each task, show that the proposed methodology identifies a reliable solution offering the most convenient trade-off with respect to the designer's selected metrics.

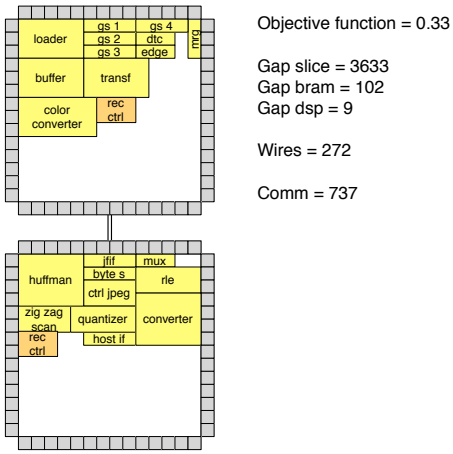


Figure 7.15.: Reliability-aware partitioning of the case study circuit when following the design flow for fixed number of non-recoverable faults.

8. Conclusions and future research directions

The research presented in this thesis has proposed a complete methodology for the design of reliable embedded systems on multi-FPGA platforms. The final aim of the work has been the exploitation of commercial SRAM-based FPGAs for mission-critical applications for systems of considerable size, requiring the use of more than a single FPGA device. In particular, we have considered space applications, characterized by strict reliability requirements due to the harsh environmental conditions and the difficulty of system maintenance.

The proposed methodology leads the designer in the realization of a multi-FPGA system able to detect and cope with the occurrence of faults globally and autonomously, thus extending its lifetime and availability. A circuit hardening approach based on a hybrid strategy has been adopted; the circuit is hardened by means of traditional reliability techniques, e.g. exploiting space redundancy, and a reconfiguration of the FPGAs is used to mitigate fault effects. We have identified two categories of faults, based on the possibility to recover from them by reconfiguration: *Recoverable faults*, that can be mitigated by reconfiguring the system (and possibly only the faulty sub-system portion) with the same configuration used before fault occurrence, and *non-recoverable faults*, that are caused by a destructive effect and lead to the necessity of relocating the functionality to a non-faulty region of the device. In the envisioned reliable system, each FPGA hosts i) a hardened portion of the entire circuit, organized in independently recoverable areas that detect, mask/tolerate, and signal the occurrence of a fault, and ii) a Reconfiguration Controller, in charge of monitoring the error signals and, when needed, performing the suitable reconfiguration of the faulty part based on the type of fault. The obtained autonomous fault tolerant system can continue working even if faults occur, thus increasing its lifetime. It is worth noting that the solution we pursue is not a generic one, rather it is identified by means of a design space exploration, to evaluate different trade-offs, meeting the designer's interests and constraints.

The main innovative contributions provided by this thesis are summarized as follows:

- *Definition of a reliable multi-FPGA system with distributed control architecture.* Rather than making each FPGA an independent fault tolerant sub-system, able to locally detect and recover from faults, we have proposed a distributed solution, where each FPGA hosts a hardened portion of the system and a Reconfiguration Controller. The aim is to achieve a higher level of reliability in the overall system, trying to avoid the single point of failure characterizing the centralized solution, that needs to be implemented onto a particular device (e.g., an ASIC or an antifuse-based FPGA).

- *Management of both recoverable and non-recoverable faults.* In literature, only recoverable faults are usually targeted, whereas non-recoverable ones are rarely analyzed and taken into account. The reliable system obtained by following the proposed methodology is able to discriminate between the two type of faults and cope with them.

- *Design of the Reconfiguration Controller.* We have proposed the design of the engine in charge of performing the envisioned fault classification and managing the reconfiguration process. The reliability of the Reconfiguration Controller has been guaranteed to block erroneous reconfigurations of the monitored FPGA and to include the module into the overall reliable system.

- *Definition of a complete flow for designing autonomous fault tolerant systems on multi-FPGA platforms.* To our knowledge, in literature there are no methodologies for the design of reliable systems on multi-FPGA platforms. We introduced a complete flow for designing such systems, and developed a prototype framework, by implementing the flow and automating, as much as possible, the design, hardening, and implementation of the system.

The work that has been presented in this thesis is the first proposal of a reliability-aware design methodology for embedded systems on multi-FPGA platforms. Thus, there are directions for future work aimed at its improvement and refinement. They are summarized in the following.

- *Experimental evaluation of the system reliability.* A methodological evaluation of the proposed hardened system has been carried out and discussed. Nevertheless, an empirical evaluation can be performed by means of a fault injection campaign, to experimentally analyze the reliability achieved by the proposed system.

- *System resynchronization after a reconfiguration.* In the thesis, the problem of resynchronizing the system after reconfiguration has

not been coped with. The approach is suitable for stateless applications and it can be extended by implementing contributions proposed in literature (e.g., [69]). A future work consists in investigating the resynchronization issue. In the final envisioned system, after realizing partial reconfiguration to cope with the occurrence of a recoverable fault, the recovered area is synchronized with the correct ones.

- *Definition of a reliable reconfiguration protocol.* Throughout the thesis, we have assumed that the reconfiguration process is performed successfully. This is supported by the consideration that reconfiguration is executed only on-demand, to cope with a detected error, hence, by taking into account a typical fault frequency, the occurrence of a new fault is unlikely. Nevertheless, for the sake of system reliability, it is necessary to guarantee that when reprogramming the device, either to reconfigure with the same bitstream or to relocate a functionality, no new fault causes a problem leading the system into an unknown state.

- *Definition of recovery data storage.* As stated in Chapter 4, a specific recovery bitstream is required for each sequence of occurred non-recoverable faults. Indeed, the number of bitstreams can be very high also for tolerating few non-recoverable faults, hence too a large memory should be required to store the configuration data. Thus, strategies of data compression must be considered for minimizing the occupation of recovery bitstreams in memory.

Besides the identified future directions, the proposed work can be refined by reviewing the design flow to exploit lower level or runtime information in the various steps.

To conclude, this research has tackled several issues for the design of reliable multi-FPGA systems, achieving the proposal of a whole methodology. This has opened new research issues that can be pursued to refine the overall approach.

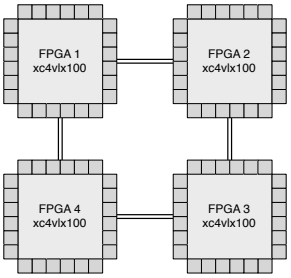
A. Partitioner performance evaluation

This appendix reports the evaluation of the performance achieved by the proposed partitioning approach, presented in Chapter 4. We show that, despite being the partitioning problem NP-complete [49], our approach, based on Mixed Integer Linear Programming (MILP) optimization models, runs in acceptable time. As described in Chapter 4, two models have been proposed, for the preliminary partitioning and for the reliability-aware one. Since the preliminary partitioner can be considered a simplification of the other one, not taking into account the possible recovery actions, we hereafter consider the reliability-aware partitioner for the performance evaluation.

We recall that, to solve the optimization problems, we used the IBM ILOG CPLEX 12.1 tool, which implements a parallel branch and cut procedure [48]. The physical system supporting the partitioner execution is based on VMWare ESXi 4.0, running on an Intel Nehalem dual socket quad-core system with 32 GB of RAM. CPLEX is hosted on a Virtual Machine (VM) running Ubuntu 11.04 Linux. The VM has four physical cores dedicated to its execution with guaranteed performance and 8GB of memory reserved.

In order to show the partitioner's performance, an extensive experimental analysis has been performed, by considering a large set of instances. The partitioner's parameters have been varied as shown in Table A.1. It is worth noting that $same\_dev_{a_1,a_2}$ can take only values 0 or 1, and the sum of the objective function's weights must equal to 1. Platforms with up to 8 devices connected in a mesh topology and characterized by the resources reported in Figure A.1, showing a Synopsis HAPS-34 commercial board [25], have been considered. Circuits include up to 30 areas. The other parameters of the partitioner were randomly generated by assuming a uniform distribution in the reported intervals.

In order to evaluate the performance of the proposed optimization model, the average optimization time required for instances of variable size is shown in Table A.2; each value reported in the table corresponds to the average time over 10 randomly generated instances. A total number of 150 instances have been considered. For the most of the consid-



$dev\_wires_{i,j}$

i \ j	1	2	3	4
1	0	729	0	149
2	729	0	149	0
3	0	149	0	729
4	149	0	729	0

tot_wires	1756

$direct\_comm_{i,j}$

i \ j	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

Device's resources	
dev_res_{slice}	49152
dev_res_{bram}	240
dev_res_{dsp}	96

$non\_direct\_comm_{i,j,k}$

i=1	1	2	3	4	i=2	1	2	3	4
1	0	0	0	0	1	0	0	1	0
2	0	0	0	0	2	0	0	0	0
3	0	0	0	0	3	0	0	0	0
4	0	1	0	0	4	0	0	0	0
i=3	**1**	**2**	**3**	**4**	**i=4**	**1**	**2**	**3**	**4**
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	2	0	0	0	0
3	0	0	0	0	3	1	0	0	0
4	0	0	0	0	4	0	0	0	0

Figure A.1.: Multi-FPGA platform model.

ered problem instances, the overall execution time to identify the optimal solution is around few seconds or few minutes, and for instances of maximum size (which are larger than nowadays real applications), the execution time is around 3 minutes. Thus, the proposed model runs in acceptable time, despite being the problem NP-complete.

Table A.1.: Values of partitioner's parameters.

Parameter	Description	Value
$\|D\|$	Set of devices	4, 6, 8
$\|A\|$	Set of areas	5, 10, 15, 20, 30
num_faults	Number of tolerated non-recoverable faults	[1, 4]
$area\_res_{a,r=slice}$	Number of slices of area a	[50, 2000]
$area\_res_{a,r=bram}$	Number of BRAMs of area a	[0, 10]
$area\_res_{a,r=dsp}$	Number of DSPs of area a	[0, 5]
$area\_wires_{a_i,a_j}$	Number of wires between areas a_i and a_j	[0, 30] if $i \neq j$, 0 otherwise
$area\_comm_{a_i,a_j}$	Throughput between areas a_i and a_j	[0, 60] if $i \neq j$, 0 otherwise
$same\_dev_{a_i,a_j}$	Placement of areas a_i and a_j on the same device	{0, 1}
w_{gap}	Weight for the distribution uniformity	(0,1)
w_{pins}	Weight for the minimization of the number of external wires	(0,1)
w_{comm}	Weight for the minimization of the external throughput	(0,1)

Table A.2.: Partitioner's execution times [s].

$\|A\|\backslash\|D\|$	4	6	8
5	0.33	1.98	2.00
10	1.49	4.46	9.12
15	3.66	10.46	20.00
20	7.63	27.01	67.69
30	22.25	83.70	197.79

B. Weights tuning for the partitioning task

When distributing a circuit on a multi-FPGA platform by exploiting the partitioner presented in Chapter 4, the suitable weights of the objective function must be set. More precisely, $\mathtt{w}_{gap}$, $\mathtt{w}_{wires}$, and $\mathtt{w}_{comm}$ must be opportunely defined, expressing the designer's preferences about the achievement of distribution uniformity and minimization of external communication. We recall that $\mathtt{w}_{gap}$ is related to the distribution uniformity, $\mathtt{w}_{wires}$ to the number of used external wires, and $\mathtt{w}_{comm}$ to the throughput for external communication. Hereafter, we perform a tuning of the weights, by considering the case study presented in Chapter 7. We have evaluated three scenarios, where we privilege the distribution uniformity, the minimization of external communication, and both them contemporarily.

Table D.1 reports the parameters characterizing the solutions obtained when focusing on an improvement of the distribution uniformity with respect to the other metrics. More precisely, for each solution, the following parameters are reported; the selected weights ($\mathtt{w}_{gap}$, $\mathtt{w}_{wires}$, $\mathtt{w}_{comm}$), the maximum gap of resources occupation among the devices, the number of required external wires, and the throughput for external communication. We have evaluated various solutions, obtained by varying the weight $\mathtt{w}_{gap}$ in the interval $[0.6, 1]$. It can be noted that, for $\mathtt{w}_{gap}$ greater than 0.7, the solutions do not entail a significative improvement in distribution uniformity. No gap smaller than 2840 in terms of slices can be achieved due to the availability of external wires, which constrains the distribution of the components. The best solution achieving distribution uniformity, also taking into account the minimization of the external communication both in terms of wires and throughput, is obtained by setting the following weights: $\mathtt{w}_{gap} = 0.7$, $\mathtt{w}_{wires} = 0.2$, and $\mathtt{w}_{comm} = 0.1$. The obtained solution is shown in Figure B.1. It can be noted that, in the achieved partitioning, FPGA 4 hosts two components that do not communicate each other. Thus, for the selected case study, we set the weights $\mathtt{w}_{gap} = 0.7$, $\mathtt{w}_{wires} = 0.1$, and $\mathtt{w}_{comm} = 0.2$, obtaining the solution shown in Figure B.2. These are the weights selected for the design flow's preliminary partitioning task, that aims at achieving a solution

uniformly distributed among the available FPGAs.

Table B.2 reports the parameters characterizing the solutions obtained when privileging the minimization of external communication. It can be noted that, for $\mathtt{w}_{gap}$ less than 0.2, the whole circuit is assigned to a single FPGA.

Finally, Table D.4 reports the parameters characterizing the solutions obtained when both the distribution uniformity and the minimization of external communication are taken into account almost in equal measure. It is worth noting that, for $\mathtt{w}_{gap} \leq 0.3$, the minimization of external communication is privileged, whereas, for $\mathtt{w}_{gap} \geq 0.4$, the distribution uniformity is preferred.

Table B.1.: Partitioning solutions when privileging the distribution uniformity.

Weights			Gap			Wires	Comm
w_{gap}	w_{wires}	w_{comm}	slice	bram	dsp		
1	0	0	2840	8	1	521	1513
0.9	0.05	0.05	2840	9	1	373	1024
0.8	0.1	0.1	2840	10	1	321	950
0.7	0.2	0.1	2840	12	1	289	879
0.7	0.1	0.2	3930	12	1	325	522
0.6	0.2	0.2	4073	12	1	251	554

Table B.2.: Partitioning solutions when privileging the minimization of external communication.

Weights			Gap			Wires	Comm
w_{gap}	w_{wires}	w_{comm}	slice	bram	dsp		
0	0	1	17585	32	1	0	0
0	1	0	17585	32	1	0	0
0.1	0.4	0.5	17585	32	1	0	0
0.2	0.4	0.4	14684	20	1	24	64

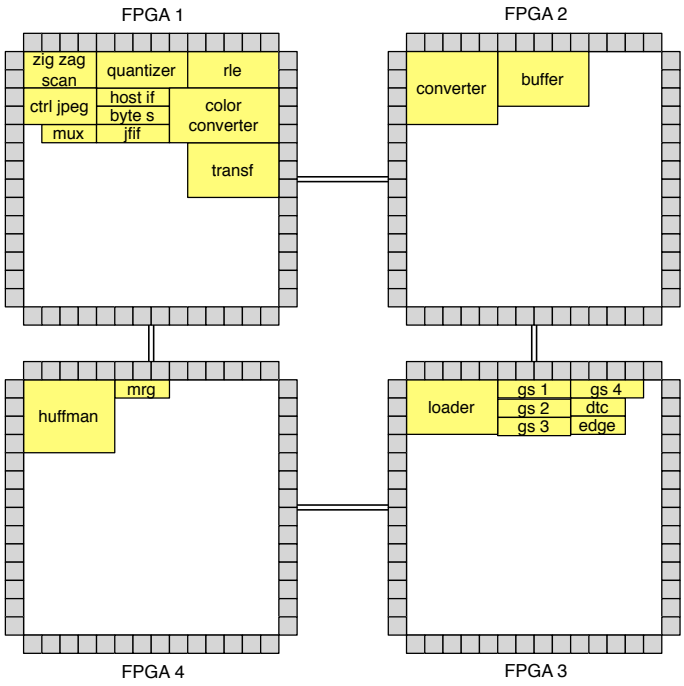


Figure B.1.: Case study circuit partitioning on the selected multi-FPGA platform by setting $w_{gap} = 0.7$, $w_{wires} = 0.2$, and $w_{comm} = 0.1$.

Table B.3.: Partitioning solutions when considering both the distribution uniformity and the minimization of external communication.

Weights			Gap			Wires	Comm
w_{gap}	w_{wires}	w_{comm}	slice	bram	dsp		
0.3	0.4	0.3	14684	20	1	24	64
0.4	0.3	0.3	4073	12	1	251	554
0.5	0.2	0.3	4073	12	1	251	554
0.5	0.3	0.2	4073	12	1	251	554

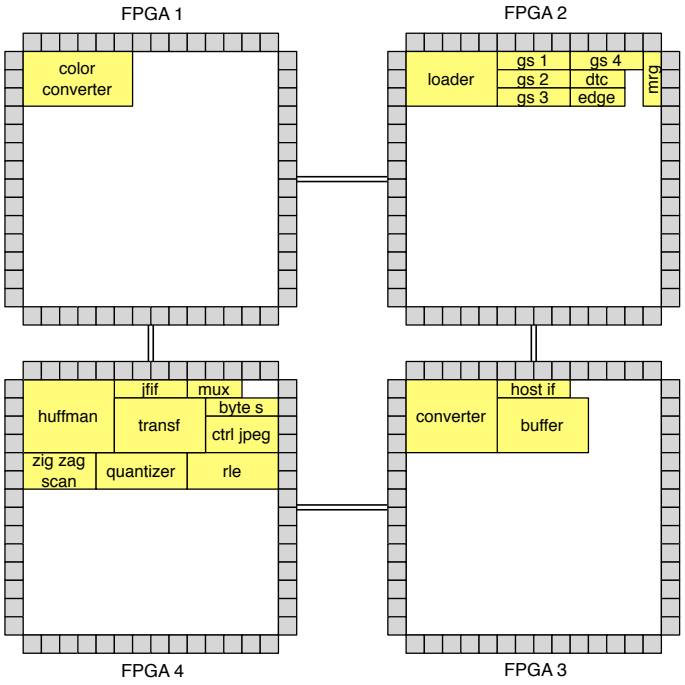


Figure B.2.: Case study circuit partitioning on the selected multi-FPGA platform by setting $\mathtt{w}_{gap} = 0.7$, $\mathtt{w}_{wires} = 0.1$, and $\mathtt{w}_{comm} = 0.2$.

C. Hardening approach performance evaluation

This appendix reports the analysis of the performance achieved by the proposed hardening approach, presented in Chapter 4. The optimization model's execution time has been evaluated by performing an extensive experimental analysis.

We recall that, to solve the optimization problems, we used the IBM ILOG CPLEX 12.1 tool, which implements a parallel branch and cut procedure [48]. The physical system supporting the partitioner execution is based on VMWare ESXi 4.0, running on an Intel Nehalem dual socket quad-core system with 32 GB of RAM. CPLEX is hosted on a Virtual Machine (VM) running Ubuntu 11.04 Linux. The VM has four physical cores dedicated to its execution with guaranteed performance and 8GB of memory reserved.

The performed experimental analysis has considered various instances, obtained by varying the model's parameters. Circuits with up to 15 components have been taken into account. The components are assigned to groups, whose cardinality is the minimum between the number of components ($|C|$) and the number of areas the circuit can be divided into (max_areas). The parameter max_areas gets the following values: i) $\lceil |C|/2 \rceil$, to consider an average case where the maximum number of areas is less than the number of components, ii) $|C|$, when the number of areas is equal to the number of components, and iii) $|C| \cdot 4$, to consider the worst case assigning each component to a group where TMR on 4 areas is applied. The other parameters characterizing the circuit were randomly generated by assuming a uniform distribution in the intervals reported in Table C.1.

In order to evaluate the performance of the proposed optimization model, the average optimization time required for instances of variable size is shown in Table C.2; each value reported in the table corresponds to the average time over 10 randomly generated instances. A total number of 150 instances have been considered. When no execution time is reported, out-of-memory errors have occurred for one or more of the random instances. Thus, the proposed model is suitable when considering circuits with up to 10 components, characterizing most of nowadays ap-

plications or obtained after partitioning the circuit among the available devices. For larger circuits, heuristic algorithms should be applied.

Table C.1.: Values of partitioner's parameters.

Parameter	Description	Value
$\texttt{comp\_res}_{c,slice}$	Number of slices required by component `c`	[0,1000]
$\texttt{comp\_res}_{c,bram}$	Number of BRAMs required by component `c`	[0,10]
$\texttt{comp\_res}_{c,dsp}$	Number of DSPs required by component `c`	[0,5]
$\texttt{comp\_wires}_{c_1,c_2}$	Number of wires between components $\texttt{c}_1$ and $\texttt{c}_2$	[0,32]
$\texttt{comp\_out}_{c}$	Number of output wires of component `c`	[0,32]

Table C.2.: Hardening approach's execution times [s].

$\lvert C\rvert \backslash max\_areas$	$\lceil \lvert C\rvert /2 \rceil$	$\lvert C\rvert$	$\lvert C\rvert \cdot 4$
5	0.07	0.49	1.58
10	5.15	988.31	1549.10
11	7.02	820.21	-
12	13.42	1692.97	-
15	23228.80	-	-

D. Weights tuning for the hardening task

When defining the independently recoverable areas composing the hardened circuit, the optimization model presented in Chapter 4 is exploited. Indeed, the suitable weights of the objective function must be set. More precisely, $\mathtt{w}_{res}$, $\mathtt{w}_{areas}$, and $\mathtt{w}_{wires}$ must be opportunely defined, expressing the designer's preferences. We recall that $\mathtt{w}_{res}$ is related to the achievement of distribution uniformity, $\mathtt{w}_{areas}$ to the maximization of the number of independently recoverable areas, and $\mathtt{w}_{wires}$ to the minimization of the number of wires between groups. Hereafter, we perform a tuning of the weights by taking into account the case study presented in Chapter 7, in particular the sub-circuit performing edge detection, that is one of the sub-circuits considered for the hardening task in the methodology evaluation. As in Chapter 7, we set $\mathtt{max\_areas} = 7$. We have evaluated four scenarios, where we privilege i) the achievement of distribution uniformity, ii) the maximization of the number of areas, iii) the minimization of the number of wires between groups, and iv) a trade-off of the metrics.

Table D.1 reports the parameters characterizing the solutions obtained when focusing on an improvement of the distribution uniformity with respect to the other metrics. More precisely, for each solution, the following parameters are reported; the selected weights ($\mathtt{w}_{res}$, $\mathtt{w}_{areas}$, $\mathtt{w}_{wires}$), the maximum gap of resources occupation among the areas, the number of areas, and the number of wires between groups. We have evaluated various solutions, obtained by varying the weight $\mathtt{w}_{res}$ in the interval $[0.5, 1]$. When only the distribution uniformity is taken into account ($\mathtt{w}_{res} = 1$), the circuit's components are grouped in a single area. For $\mathtt{w}_{res}$ included in the interval $[0.6, 0.9]$, the same solution achieving a trade-off between the distribution uniformity and the number of areas is obtained. In the following, other solutions obtained by setting $\mathtt{w}_{res} \leq 0.4$ are evaluated.

Table D.2 reports the parameters characterizing the solutions obtained when privileging the maximization of the number of independently recoverable areas, by varying the weight $\mathtt{w}_{areas}$ in the interval $[0.5, 1]$. For $\mathtt{w}_{areas} \geq 0.7$, the identified number of areas equals to $\mathtt{max\_areas}$, hence the maximization is achieved.

Table D.3 reports the parameters characterizing the solutions obtained

when privileging the minimization of the number of wires between groups. By varying the weight w_{wires} in the interval $[0.5, 1]$, the minimization of external wires is always achieved.

Finally, Table D.4 reports the parameters characterizing the solutions obtained when a trade-off between the metrics is required. In all cases, the same solution is achieved.

To conclude, the following weights must be set to privilege the related metric: $w_{res} \geq 0.6$ for distribution uniformity (without considering the solution obtained by setting $w_{res} = 1$, identifying a single area), $w_{areas} \geq 0.7$ for the maximization of the number of areas, and $w_{wires} \geq 0.4$ for the minimization of the number of wires. The presented results should help the designer in adopting the suitable weights according to his/her requirements.

Table D.1.: Hardened solutions when privileging the distribution uniformity.

Weights			Gap			#areas	#wires
w_{res}	w_{areas}	w_{wires}	slice	bram	dsp		
1	0	0	0	0	0	1	24
0.9	0.1	0	96	0	0	3	24
0.9	0	0.1	0	0	0	1	24
0.8	0.1	0.1	96	0	0	3	24
0.7	0.2	0.1	96	0	0	3	24
0.7	0.1	0.2	96	0	0	3	24
0.6	0.2	0.2	96	0	0	3	24
0.5	0.3	0.2	96	0	0	3	24
0.5	0.2	0.3	96	0	0	3	24
0.5	0.4	0.1	1953	12	0	7	122

Table D.2.: Hardened solutions when privileging the maximization of the number of areas.

Weights			Gap			#areas	#wires
w_{res}	w_{areas}	w_{wires}	slice	bram	dsp		
0	1	0	2603	12	0	7	148
0.1	0.9	0	1801	12	0	7	218
0	0.9	0.1	2781	12	0	7	63
0.1	0.8	0.1	2631	12	0	7	63
0.3	0.7	0	1801	12	0	7	218
0.2	0.7	0.1	2631	12	0	7	63
0.1	0.7	0.2	2631	12	0	7	63
0.2	0.6	0.2	2631	12	0	7	63
0.3	0.6	0.1	2631	12	0	7	63
0.4	0.6	0	96	0	0	3	24
0.3	0.5	0.2	96	0	0	3	24
0.2	0.5	0.3	2631	12	0	7	63
0.4	0.5	0.1	1953	12	0	7	122

Table D.3.: Hardened solutions when privileging the minimization of wires between groups.

Weights			Gap			#areas	#wires
w_{res}	w_{areas}	w_{wires}	slice	bram	dsp		
0	0	1	5802	24	0	2	24
0.1	0	0.9	0	0	0	1	24
0	0.1	0.9	2901	12	0	4	24
0.1	0.1	0.8	96	0	0	3	24
0.2	0.1	0.7	96	0	0	3	24
0.1	0.2	0.7	96	0	0	3	24
0.2	0.2	0.6	96	0	0	3	24
0.3	0.2	0.5	96	0	0	3	24
0.2	0.3	0.5	96	0	0	3	24

Table D.4.: Hardened solutions with a trade-off between the metrics.

Weights			Gap			#areas	#wires
w_{res}	w_{areas}	w_{wires}	slice	bram	dsp		
0.4	0.3	0.3	96	0	0	3	24
0.3	0.4	0.3	96	0	0	3	24
0.3	0.3	0.4	96	0	0	3	24

Bibliography

[1] C. Bolchini, A. Miele, and C. Sandionigi. A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs. *Transactions on Computers*, 2010.

[2] T. Kuwahara. *FPGA-based Reconfigurable On-board Computing Systems for Space Applications*. PhD thesis, Universitat Stuttgart, Institute of Space Systems, 2009.

[3] R. Katz, K. LaBel, J. J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift. Radiation effectcs on current field programmable technologies. *IEEE Trans. Nuclear Science*, 6(44):1945–1956, 1997.

[4] C. Carmichael, M. Caffrey, and A. Salazar. Correcting Single-Event Upsets Through Virtex Partial Configuration. Technical Report XAPP216, Xilinx Inc., June 2000.

[5] ITRS. International Technology Roadmap for Semiconductors. http://www.itrs.net/links/2010itrs/home2010.htm.

[6] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and G. R. Sechi. Fault-Tolerant Voting Mechanism and Recovery Scheme for TMR FPGA-Based Systems. In *Proc. IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Systems*, pages 233–240, 1998.

[7] F. Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis. Designing Fault-Tolerant Techniques for SRAM-Based FPGAs. *IEEE Design and Test of Computers*, 21(6):552–562, 2004.

[8] G. L. Smith and L. de la Torre. Techniques to enable FPGA based reconfigurable fault tolerant space computing. In *Proc. IEEE Aerospace Conference*, page 11 pp., 2006.

[9] V. Rana, M. Santambrogio, D. Sciuto, B. Kettelhoit, M. Köster, M. Porrmann, and U. Rückert. Partial Dynamic Reconfiguration in a Multi-FPGA Clustered Architecture Based on Linux. In *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, pages 1–8, 2007.

[10] P. K. Samudrala, J. Ramos, and S. Katkoori. Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Trans. Nuclear Science*, 51(5):2957–2969, October 2004.

[11] C. Bolchini and A. Miele. Design Space Exploration for the Design of Reliable SRAM-based FPGA Systems. In *Proc. IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems*, pages 332–340, 2008.

[12] K. S. Morgan. *SEU-induced Persistent Error Propagation in FPGAs*. PhD thesis, Brigham Young University, 2006.

[13] J. L. Barth, C. S. Dyer, and E. G. Stassinopoulos. Space, Atmospheric, and Terrestrial Radiation Environments. *IEEE Trans. Nuclear Science*, 50(3), 2003.

[14] ECSS. Methods for the calculation of radiation received and its effects, and a policy for design margins. Technical Report ECSS-E-ST-10-12C, European Cooperation for Space Standardization, November 2008.

[15] C. Carmichael and C. W. Tseng. Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory. Technical Report XAPP1088, Xilinx, October 2009.

[16] C. Poivey, M. Berg, S. Stansberry, M. Friendlich, H. Kim, D. Petrick, and K. LaBel. Heavy ion SEE test of Virtex4 FPGA XC4VFX60 from Xilinx. Technical Report T021607XC4VFX60, Muniz Engineering Inc., University of Southern California, NASA-GSFC, June 2007.

[17] D. Petrick, W. Powell, J. W. Howard Jr., and K. A. LaBel. Virtex-II Pro SEE Test Methods and Results. In *Military and Aerospace Applications of Programmable Devices and Technologies Conf.*, 2004.

[18] S. Srinivasan, P. Mangalagiri, Y. Xie, N. Vijaykrishnan, and K. Sarpatwari. FLAW: FPGA Lifetime AWareness. In *Proc. Design Automation Conf.*, 2006.

[19] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari. Toward Increasing FPGA Lifetime. *IEEE Trans. Dependable and Secure Computing*, 5(2):115 – 127, 2008.

[20] S. Hauck, G. Borriello, and C. Ebeling. Mesh routing topologies for multi-fpga systems. In *IEEE Proc. Int. Conf. Computer Design: VLSI in Computer & Processors*, pages 170–177, 1994.

[21] Xilinx. Spartan-3 FPGA Family Data Sheet. Technical Report DS099, December 2009.

[22] Xilinx. Virtex-II Platform FPGA: Complete Data Sheet. Technical Report DS031, March 2005.

[23] Xilinx. Virtex-4 FPGA Configuration User Guide. Technical Report UG071, August 2010.

[24] Xilinx. Virtex-5 FPGA Configuration User Guide. Technical Report UG191, June 2009.

[25] Synplicity. HAPS-34. http://www.synopsys.com/home.aspx.

[26] David P. Montminy, Rusty O. Baldwin, Paul D. Williams, and Barry E. Mullins. Using relocatable bitstreams for fault tolerance. In *Proc. of NASA/ESA Conf. on Adaptive Hardware and Systems*, pages 701–708, 2007.

[27] X. Iturbe, K. Benkrid, T. Arslan, I. Martinez, M. Azkarate, and M.D. Santambrogio. A Roadmap for Autonomous Fault-Tolerant Systems. In *Proc. Conf. Design and Architectures for Signal and Image Processing*, pages 311–321, 2010.

[28] S. Mitra, W.-J. Huang, N.R. Saxena, S.-Y. Yu, and E.J. McCluskey. Reconfigurable Architecture for Autonomous Self-Repair. *IEEE Design & Test of Comp.*, 21:228–240, May 2004.

[29] R. Noji, S. Fujie, Y. Yoshikawa, H. Ichihara, and T. Inoue. An FPGA-based fail-soft system with adaptive reconfiguration. In *Proc. Intl. On-Line Testing Symp.*, pages 127–132, 2010.

[30] Xilinx Inc. http://www.xilinx.com.

[31] Frank Vahid. I/O and performance tradeoffs with the FunctionBus during multi-FPGA partitioning. In *Proceedings of the 1997 ACM Fifth International Symposium on Field-Programmable Gate Arrays*, pages 27–34, 1997.

[32] Iyad Ouaiss, Sriram Govindarajan, Vinoo Srinivasan, Meenakshi Kaul, and Ranga Vemuri. An integrated partitioning and synthesis system for dynamically reconfigurable multi-fpga architectures. In

In Proceedings of Parallel and Distributed Processing, pages 31–36, 1998.

[33] Preetham Lakshmikanthan, Sriram Govindarajan, Vinoo Srinivasan, and Ranga Vemuri. Behavioral partitioning with synthesis for multi-fpga architectures under interconnect, area, and latency constraints. In *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 924–931, 2000.

[34] Kalapi Roy and Carl Sechen. A timing driven n-way chip and multichip partitioner. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 240–247, 1993.

[35] S. Hauck and G. Borriello. Logic partition orderings for multi-fpga systems. In *Field-Programmable Gate Arrays, 1995. FPGA '95. Proceedings of the Third International ACM Symposium on*, pages 32 – 38, 1995.

[36] Kalapi Roy-Neogi and Carl Sechen. Multiple FPGA partitioning with performance optimization. In *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays*, pages 146–152, 1995.

[37] J.I. Hidalgo, J. Lanchares, and R. Hermida. Partitioning and placement for multi-FPGA systems using genetic algorithms. In *Euromicro Conference, 2000. Proceedings of the 26th*, volume 1, pages 204 –211 vol.1, 2000.

[38] J.I. Hidalgo, R. Baraglia, R. Perego, J. Lanchares, and F. Tirado. A parallel compact genetic algorithm for multi-fpga partitioning. In *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on*, pages 113 –120, 2001.

[39] Nam Sung Woo and Jaeseok Kim. An efficient method of partitioning circuits for multiple-FPGA implementation. In *Proceedings of the 30th international Design Automation Conference*, pages 202–207, 1993.

[40] Roman Kužnar, Franc Brglez, and Krzysztof Kozminski. Cost minimization of partitions into multiple devices. In *Proceedings of the 30th international Design Automation Conference*, pages 315–320, 1993.

[41] Nan-Chi Chou, Lung-Tien Liu, Chung-Kuan Cheng, Wei-Jin Dai, and Rodney Lindelof. Circuit partitioning for huge logic emula-

tion systems. In *Proceedings of the 31st annual Design Automation Conference*, pages 244–249, 1994.

[42] Roman Kužnar, Franc Brglez, and Baldomir Zajc. Multi-way netlist partitioning into heterogeneous FPGAs and minimization of total device cost and interconnect. In *Proceedings of the 31st annual Design Automation Conference*, pages 238–243, 1994.

[43] Dennis J.-H. Huang and Andrew B. Kahng. Multi-way system partitioning into a single type or multiple types of FPGAs. In *Proceedings of the 1995 ACM Third International Symposium on Field-Programmable Gate Arrays*, pages 140–145, 1995.

[44] Prashant Sawkar and Donald Thomas. Multi-way partitioning for minimum delay for look-up table based FPGAs. In *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pages 201–210, 1995.

[45] Wen-Jong Fang and Allen C.-H. Wu. Performance-driven multi-FPGA partitioning using functional clustering and replication. In *Proceedings of the 35th annual Design Automation Conference*, pages 283–286, 1998.

[46] Helena Krupnova and Gabriele Saucier. Iterative improvement based multi-way netlist partitioning for FPGAs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 1999.

[47] C. Bolchini, A. Miele, and C. Sandionigi. Automated resource-aware floorplanning of reconfigurable areas in partially-reconfigurable FPGA systems. In *Int. Conf. on Field Programmable Logic and Applications*, 2011, To appear.

[48] L. Wolsey. *Integer Programming.* John Wiley and Sons, 1998.

[49] Sadiq M. Sait and Habib Youssef. *VLSI physical design automation.* World Scientific Publishing Co. Pte. Ltd., 2004.

[50] C. Bolchini, L. Fossati, D. Merodio Codinachs, A. Miele, and C. Sandionigi. A Reliable Reconfiguration Controller for Fault-Tolerant Embedded Systems on Multi-FPGA Platforms. In *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2010.

[51] C. Bolchini and C. Sandionigi. Fault Classification for SRAM-based FPGAs in the Space Environment for Fault Mitigation. *IEEE Embedded Systems Letters*, 2010.

[52] Salvatore Pontarelli, Marco Ottavi, Vamsi Vankamamidi, Gian Carlo Cardarilli, Fabrizio Lombardi, and Adelio Salsano. Analysis and evaluations of reliability of reconfigurable fpgas. *Journal of Electronic Testing*, 24(1-3):105–116, 2008.

[53] F. A. Bower, D. J. Sorin, and S. Ozev. A Mechanism for Online Diagnosis of Hard Faults in Microprocessors. In *Int. Symp. on Microarchitecture*, pages 197 – 208, 2005.

[54] European Space Agency. SPENVIS. http://www.spenvis.oma.be/.

[55] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth. Harsh Environments: Space Radiation Environment, Effects and Mitigation. *Johns Hopkins APL Technical Digest*, 28(1), 2008.

[56] V. Bocci, M. Carletti, G. Chiodi, E. Gennari, E. Petrolo, A. Salamon, R. Vari, and S. Veneziano. Radiation test and application of FPGAs in the ATLAS Level 1 Trigger. In *Workshop on Electronics for LHC Experiments*, 2001.

[57] P. Alfke. Xilinx FPGAs: A Technical Overview for the First-Time User. Technical Report XAPP097, Xilinx, December 1998.

[58] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker. A multi-platform controller allowing for maximum Dynamic Partial Reconfiguration throughput. In *Proc. Int. Conf. Field Programmable Logic and Applications*, pages 535–538, 2008.

[59] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele. A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration. In *Proc. IEEE Int. Parallel and Distributed Processing Symp.*, pages 1–7, 2007.

[60] A. Cuoccio, P. R. Grassi, V. Rana, M. D. Santambrogio, and D. Sciuto. A Generation Flow for Self-Reconfiguration Controllers Customization. In *IEEE Int. Workshop Electronic Design, Test and Applications*, pages 279–284, 2008.

[61] S. Bayar and A. Yurdakul. Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP). In *Proc. HiPEAC Workshop on Reconfigurable Computing*, 2008.

[62] Xilinx. Virtex-II Pro and Virtex-II Pro X FPGA User Guide. Technical Report UG012, November 2007.

[63] Xilinx. OPB HWICAP Product Specification. Technical Report DS280, March 2004.

[64] M. Sonza Reorda, M. Violante, C. Meinhardt, and R. Reis. A low-cost SEE mitigation solution for soft-processors embedded in systems on programmable chips. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 352–357, 2009.

[65] ESA Microelectronics. LEON2-FT. Website, 2009. `http://www.esa.int/TEC/Microelectronics/SEMUD70CYTE_0.html`.

[66] D. Nikolos. Self-testing embedded two-rail checkers. *J. Electronic Testing, Theory and Applications*, 12(1-2):69–79, 1998.

[67] Xilinx Inc. Reading User Data from Configuration PROMs. Technical Report XAPP694, Xilinx Inc., November 2007.

[68] Xilinx Inc. *Xilinx TMRTool*, 2006.

[69] Conrado Pilotto, José Rodrigo Azambuja, and Fernanda Lima Kastensmidt. Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications. In *Proceedings of the 21st annual symposium on Integrated circuits and system design*, pages 199–204. ACM, 2008.