


Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (33^{th} cycle)

New Design Techniques for Dynamic Reconfigurable Architectures

Ludovica Bozzoli

* * * * * *

Supervisor
Prof. Luca Sterpone

Doctoral Examination Committee:
Prof. Eduardo De La Torre, Universidad Politécnica de Madrid
Prof. Bartolomeo Montrucchio, Politecnico di Torino
Prof. Mario Porrmann, *Referee*, Osnabrück University
Prof. Anees Ullah, University of Engineering and Technology Peshawar
Prof. Stephan Wong, *Referee*, Delft University of Technology

Politecnico di Torino
September 28, 2021

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

. . . . . .

Ludovica Bozzoli

Turin, September 28, 2021

Summary

Over the past decades, reconfigurable computing had increasingly gained attention for the high integration, performances, and in-field upgradability it can provide to a computing system. In fact, its main novelty consists of offering on a single platform the benefits of hardware customization coupled with the flexibility typical of software programming.
This has driven the effort in proposing and developing several reconfigurable architectures to efficiently tailor various and increasingly complex applications. Such trend led to the state-of-the-art architecture, consisting of a heterogeneous processing system including on-chip processors coupled with dynamically reconfigurable fabric to implement custom and run-time upgradable cores.
Today, the commercial technology to implement the reconfigurable fabric is represented by the SRAM-based Field Programmable Gate Array (FPGA) and consists of one layer of programmable resources and one layer of configuration, an SRAM memory holding the configuration data defining the behavior of the resource layer. These devices enable to access and modify specific portions of the memory content from the resource layer without stopping the application execution to adapt its functionality to requirements varying over time, to maximize its performance, and to increase its dependability.
This feature, called Dynamic Partial Reconfiguration, has been one of the leading characteristics that drove the rapid growth of this computational paradigm and increasingly widened the deployment of SRAM-based FPGAs in many computational-intensive and high-reliability applications, such as networking, High-Performance Computing, and satellite or high energy physics experiments fault-tolerant electronics.
However, the current SRAM-based FPGA architecture embeds some functionalities that demand to be addressed to fully express the promises of the reconfigurable computing paradigm and that provide opportunities for enhancing reconfigurable systems performance and dependability.
The performance optimizations are related to the relatively long time required to perform run-time reconfiguration in commercial devices that, if not properly managed, can jeopardize gain achieved through to the in-field upgradability.
SRAM-based FPGAs' dependability instead becomes crucial when they operate in

presence of radiation, as in aerospace and high energy physics experiments, due to their configuration memory sensitivity to radiation-induced errors that must be considered and characterized according to the mission environment before deployment to avoid failures.
In this view, the doctoral research presented in this dissertation addresses dynamically and partially reconfigurable architectures' performance through novel techniques for the enhancement of the reconfiguration procedure, and self-reconfigurable systems dependability through the analysis of their radiation sensitivity to radiation-induced effects.
Therefore, the main opportunities and challenges embedded in reconfigurable architectures are firstly introduced and followed by an overview of the state-of-the-art architectural solutions and applications for high-performance and high-reliability reconfigurable platforms.
The main contributions to the optimization of the reconfiguration procedure performances are provided in the **Part 1** of the dissertation.
In detail, the Frame-driven Routing Algorithm (FeDRA) is presented as a novel and generalized approach to optimize within the development process the amount of configuration memory data involved in the reconfiguration of circuits deployed on commercial SRAM-based FPGAs. The proposed approach enables to achieve a consistent reduction of the reconfiguration overhead when compared to solutions obtained with the standard process.
Subsequently, the Reconfigurable Multipotent (ReM) cell, developed as the basic reconfigurable unit for novel architectural solutions oriented to the fast and detailed in-field reconfiguration, is presented. In detail, the proposed reconfigurable cell enables bit-level reconfiguration within a single clock cycle while minimizing the amount of data involved in the procedure thanks to the key feature of reconfiguring itself and the contiguous units.
The **Part 2** of the dissertation instead presents the main contributions to reconfigurable FPGAs dependability that consist of radiation sensitivity analyses centered on the self-reconfiguration controller, which represent the key component enabling in-field reconfiguration.
In detail, a soft-error analysis on different implementations of the self-reconfiguration controller managing the configuration memory access in dynamically and partially reconfigurable applications is presented. This analysis, which has been performed by miming different radiation profiles through detailed fault-injections, allowed to obtain indications about the controllers' dependability and applicability in different radiation environments according to their operational goals.
Accordingly, a self-monitoring setup based on the most suitable self-reconfiguration controller and instrumented for the online and automated radiation analysis of SRAM configuration memory sensitivity is presented. The proposed setup enabled the usage of a Neutron Generator as a radiation source and strongly reduces the time and the cost required by typical radiation testing facilities and approaches.

Acknowledgements

I would like to acknowledge all the people who accompanied me in my doctoral experience aiding, supporting, and motivating me towards the accomplishment of this journey.

Firstly, I would like to thank Prof. Luca Sterpone for giving me the opportunity of taking part in such a fascinating research field and environment and for supporting, encouraging, and advising me all over the years of my Ph.D. strengthening my curiosity and motivation.
Then, I would like to thank all the people in CAD Group and my colleagues for enabling me to pursue my research activity in a truly stimulating and pleasant context. A special thank goes to Sarah Azimi, Corrado de Sio, and Boyang Du for everything I learned working with them and for friendly aiding me in this path.
Furthermore, I would like to thank all the people from other Institutions I had the chance to collaborate with, especially Antonio Tramutola for the support in the research activity carried out with Thales Alenia Space and the people encountered in the training experience at the Los Alamos National Laboratory.

Finally, a special thank goes to Tommaso, my Family, and my Friends for their constant and warm presence and, above all, for the energy they give me and that encourages me every day.

To my Parents...

...who, no matter how far away, always find their way of being close to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Reconfigurable Computing

1.1.1 Origin and Paradigm

The concept of *Reconfigurable Computing* has been suggested for the first time in '60s by Gerald Estrin and his group at the University of California at Los Angeles (UCLA) [1]. The idea was driven by the fact the performances of available computing machines ware not scaling efficiently with the increasing computational demand and complexity of the algorithms to be faced at that time, highlighting the need to explore novel computing architectures moving forward the consolidated Von Neumann paradigm.
Starting from this idea, the first prototype of a reconfigurable platform was proposed by Estrin in [2] as a standard processor tightly coupled with an array of reconfigurable cells, which behavior could be defined by the processor during the execution according to the task effectively required at a specific time.
The key idea behind this reconfigurable machine consisted of coupling on a single computing architecture the flexibility typical of a software program running on standard processors and the high performances typical of dedicated custom hardware exploiting the advantages of both computing approaches.
In fact, specialized hardware circuits, such as Application-Specific Integrated Circuits (ASICs), offer high performance since they are fabricated for a specific task, and thus fully optimized during design and manufacturing to achieve the highest performance in terms of delay, power consumption, and area for the target computation. At the same time, this specialization results in very poor flexibility and implies very high production costs.
On the other side, standard processors and general-purpose programmable systems offer high flexibility enabling the implementation of different software algorithms on the same machine after production and deployment.

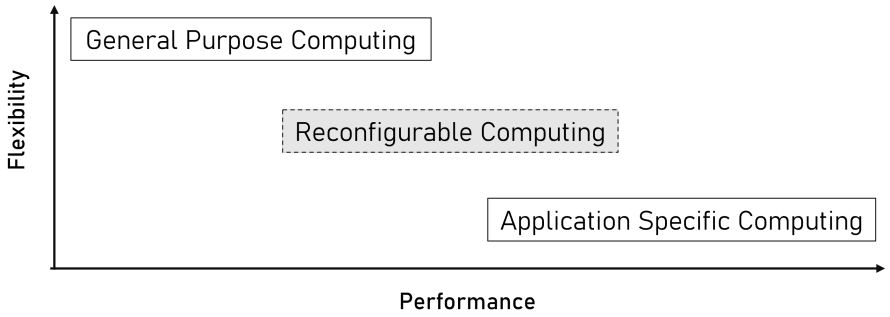

Figure 1.1: Computing Implementations: Flexibility vs. Performance

However, their flexibility and lower costs result in performance losses because even if different algorithms can be easily tailored on them the fixed and general-purpose hardware cannot optimally support all their varying computational requirements.
Therefore, the Reconfigurable Computing paradigm aims at filling the gap between the two classic computing approaches by enabling hardware designs to be customized and optimized while deployed and achieving the in-field flexibility typical of software [3][4].
Accordingly, in 1999, Andre Dèhon and John Wawrzynek defined in [5] reconfigurable architectures as computing machines that enable, simultaneously, customization after fabrication to execute any algorithm, and spatial computation to execute their tasks.
In their view, these two properties are the ones allowing the definition of computing architectures able to adapt and change their functionalities while operating by repurposing their resources according to the needs required at a specific time.
This concept of on-line circuit specialization has been defined as *Run-time* or *Dynamic Reconfiguration*, and has been one of the leading features which have driven the increasing interest from both industries and academia in the development and deployment of reconfigurable architectures.
In fact, the feature of adapting the hardware circuitry according to the needs enables new opportunities for the efficient implementation and optimization of complex and intensive computations: the possibility to dynamically allocate resources to computing tasks at run-time allows the implementation of applications able to adapt to varying conditions and requirements, the deployment of systems requiring more resources than the ones physically available on the target device, and the introduction of error recovery and prevention techniques to avoid application failures.
In general, reconfigurable hardware relies on a two-layers architectural model consisting of one layer of reconfigurable resources and one layer of configuration, a storage memory to hold the configuration data defining the behavior of the reconfigurable hardware layer (Fig. 1.2).

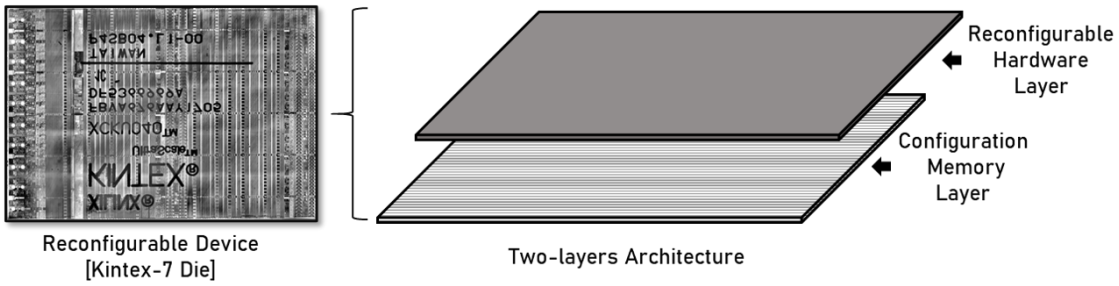

Figure 1.2: Reconfigurable Device's Two Layers Architecture: Reconfigurable Hardware Layer and Configuration Memory Layer

The content of the configuration memory layer representing the circuitry implemented on the hardware layer can be is referred also as *context*. According to the architecture and the purpose, reconfiguration can be applied to the whole hardware context (*Single Context Reconfiguration*, Fig. 1.3a) or selectively to sub-portions of the reconfigurable layer (*Partial Reconfiguration* Fig. 1.3b).

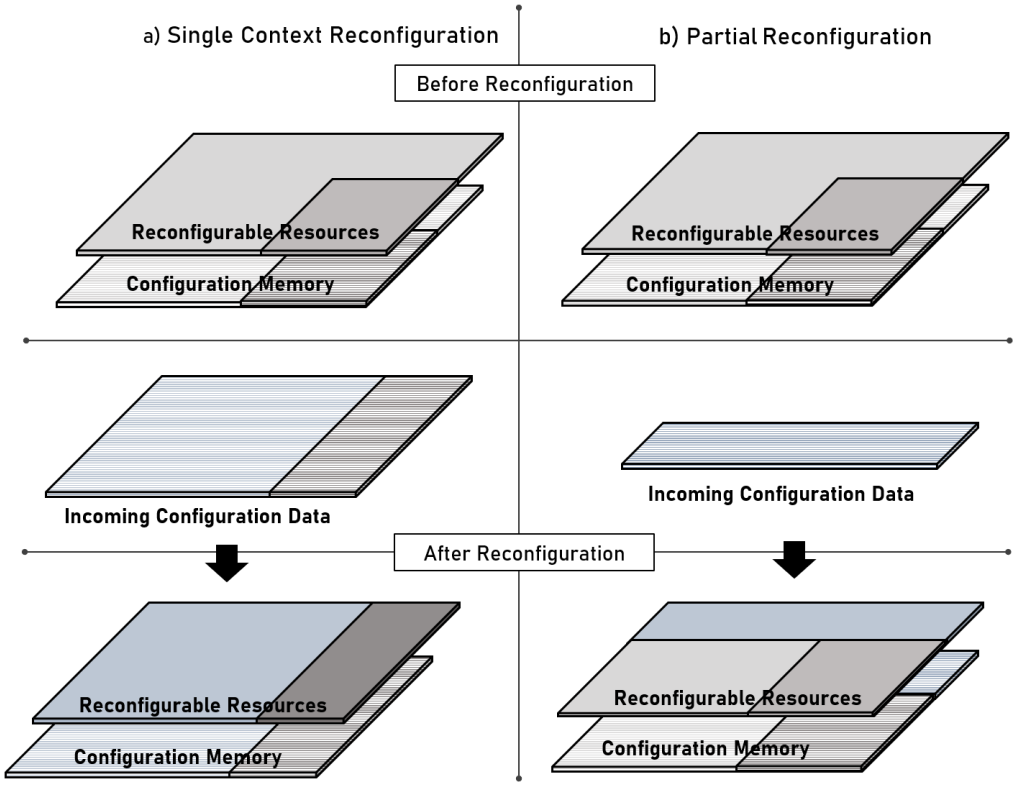

Figure 1.3: a) Single Context Reconfiguration: the Whole Configuration Memory Content is Updated; b)Partial Reconfiguration: Only the Target Portion of the Configuration is Involved in the Procedure

In Single Context Reconfiguration, the full content of the configuration memory is written, and the configuration data involved in the reconfiguration coincide with the full image of the circuit programmed on the reconfigurable resource layer. This kind of reconfiguration is efficient in the case that the whole context deployed on the reconfigurable hardware needs to be modified. Conversely, when just some portions of the context require updates or modifications, the Single Context Reconfiguration is inefficient, as the longer reconfiguration times needed to write the full configuration memory are unjustified since also memory segments that do not require any updates are involved in the procedure.
On the other hand, Partial Reconfiguration enables rewriting only specific portions of the configuration memory, targeting only the data holding the settings for the target portion of the reconfigurable resource layer that requires modifications or updates. This enables to save reconfiguration time, reducing also the risk to affect context portions not involved in the procedure.
Furthermore, as both Single Context and Partial Reconfiguration can be performed offline or at run-time, when Partial Reconfiguration is performed at run-time, it enables the possibility for the rest of the hardware not involved in the procedure to continue its operation while reconfiguration is performed.

Dynamic Partial Reconfiguration

In detail, when reconfiguration is performed only on specific portions of the hardware fabric while the overall application is running, it is called *Dynamic Partial Reconfiguration* (DPR) and it can strongly improve the Reconfigurable Computing paradigm since it truly maximizes its efficiency and flexibility.
In fact, the possibility to address only a specific portion of the configuration memory allows to selectively apply optimizations just to the target area without stopping the execution, requires less data transfer, and reduces the time needed to perform the reconfiguration procedure [6][7].
In general, DPR is supported by hardware fabrics that allow detailed access to specific memory segments, both in reading and writing, without affecting the operation of the parts of the system not subject to reconfiguration and typically can be performed to pursue three main goals: area efficiency, power saving, and reliability [8][9][10].
In detail, thanks to DPR it is possible to time-multiplex the resources available on the fabric, to adapt or update target portions of the application to meet varying requirements, to activate or deactivate computing tasks to tune power consumption according to the required computational effort, and to repair or relocate damaged or critical hardware modules.

1.1.2 Architectural Evolution and Granularity

Although an early prototype of the reconfigurable machine suggested in [2] was realized at UCLA, the Reconfigurable Computing paradigm remains theoretical until the ‘80s.
In those years the earliest devices allowing to be customized after manufacturing appear on the market, while from the ‘90s the production of more sophisticated and capable reconfigurable devices started to rapidly gain commercial attention among the consolidated ASIC and standard processors.
In fact, those years have witnessed a rapid evolution of programmable logic devices (PLDs), logic circuits allowing to define their behavior after the fabrication process one or multiple times programming their configuration settings through dedicated procedures.
The smaller Programmable Array Logic (PAL) and Generic Array Logic (GAL), made of fixed simple logic nodes which could be combined through fuse patterns after fabrication to obtain the wanted Boolean functions, evolved into larger configurable devices, as the Complex Programmable Logic Devices (CPLDs), enabling the post-manufacturing customization of bigger digital circuits.
Eventually, Field Programmable Gate Arrays (FPGAs) reached the market. Early FPGA devices were based on an island-style architecture made programmable functional and connection tiles, allowing the implementation of more complex circuits thanks to their different gate array technology, which relies on programmable Look-up Tables (LUTs).
Providing an higher capacity, flexibility, integration, and in-field programmability versus their predecessors, FPGAs had increasingly become a more attractive solution for logic emulation, prototyping, and eventually, for the implementation of complex applications demanding high computational effort.
Thanks to their growing popularity, FPGA architectures have rapidly evolved over the last decades enhancing their flexibility in the reconfiguration process and implementing more efficient configuration supports and mechanisms.
This trend led to the production of the first devices effectively enabling the possibility to update multiple times, fully or partially, their functionality not only after manufacturing but even after deployment allowing Reconfigurable Computing to emerge as a new and effective solution for the implementation and optimization of complex computational tasks [3].
This paved the way for the study and development of dynamically and partially reconfigurable systems pushing the effort of both academia and industries in the exploration of novel architectures to exploit the concept of reconfigurability for different purposes and with different granularity levels to efficiently tailor on them a wider range of applications, including the ones in the high-performance and mission-critical fields

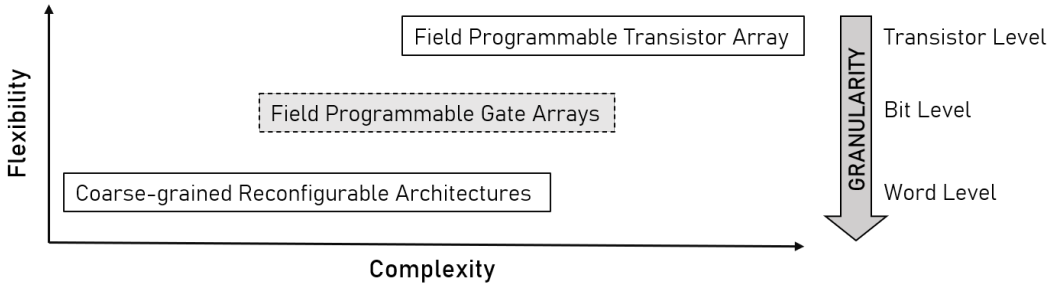

Figure 1.4: Reconfiguration Granularity: Flexibility vs. Complexity

Architecture Granularity

In general, reconfigurable architectures can be distinguished in two macro-categories, *Fine-grained* and *Coarse-grained*, where the reconfiguration granularity is defined according to the size of the data processable by the basic reconfigurable architectural unit [7][11].
According to this definition, FPGAs are fine-grained reconfigurable architectures since their smallest reconfigurable elements operate at the bit level. In general, this kind of architecture presents higher flexibility and configuration capabilities and efficiently supports intensive and irregular bit-level computation, but typically implies higher complexity and area overhead.
For more complex arithmetic and sequential controls, Coarse-grained Reconfigurable Architectures (CGRAs) are preferable since they are based on bigger and more complete functional units which efficiently operate on wider data sizes. On the other hand, the increased granularity size implies a lower flexibility.
At the extreme opposite, Field Programmable Transistor Arrays (FPTAs) have been proposed [12][13]. In fact, FPTA programmability reaches the transistor level enabling the deployment of both analog and digital reconfigurable systems.
However, their deep flexibility pays a huge prize in terms of area overhead and extreme complexity.
Although both CGRAs and FPTAs represent interesting reconfigurable architectures, today FPGAs are the only commercially available devices for the deployment of reconfigurable systems and remain the golden candidates for these applications thanks to the optimal trade-off between complexity and flexibility they provide.

1.1.3 Reconfiguration for Reliability

Reconfigurable FPGAs have gained a high interest for the deployment of systems requiring high reliability, especially in the domain of space and avionic, and more generally for safety-critical applications.

In fact, in addition to the advantages provided in terms of computational capabilities for intensive data processing, the flexibility enabled by the reconfiguration is especially valuable in this context for two main reasons.
Firstly, offline Single Context Reconfiguration enables to strongly reduce the cost and time for application deployment. In fact, the cost of custom hardware development and fabrication is extremely high, and unjustified low and medium-scale productions, like the ones typically required in aerospace and mission-critical domains. Furthermore, fabrication costs highly increase considering that in these domains multiple iterations and modification on the system could be necessary during prototyping and testing to guaranty its readiness to the mission before deployment.
Consequently, the possibility to deploy and test on the same reconfigurable device different versions and updates of the system under development maximize design efficiency and reliability while lowering both fabrication costs and deployment time.
Secondly but most importantly, in-field reconfiguration makes the usage of reconfigurable FPGAs extremely valuable to increase application reliability while deployed. In fact, mission-critical applications as the one in the aerospace field, in addition to strong reliability requirements must face harsh environmental conditions that could variate over time and induce on the system both temporary and permanent misbehaviors.
Therefore, the possibility to rely on both Single Context and Partial Reconfiguration after deployment enables mission-critical systems to adapt to varying conditions and requirements and to embed both error-detection and error-correction capabilities improving their fault tolerance.
Therefore, as will be discussed in Section 1.3.1 the strong improvement reconfiguration can provide to a system in terms of reliability, availability, and extended lifetime coupled with the benefits provided on computational power, area saving, and power-efficiency has made reconfigurable hardware increasingly valuable for the development of high-performance and reliable systems for mission-critical applications. In fact, as it will be further discussed in Section 1.3.3, for mission-critical applications deployed in harsh environment as the space or high energy physics experiments, the possibility to read configuration memory data while the system is performing its operations enables to check their integrity by comparing them with reference data to identify and locate possible corruptions induced by radiations.
On the other side, the possibility to fully or partially update the configuration layer content without interrupting the application execution through dynamic reconfiguration enables the correction of transient errors writing back the golden configuration data or moving critical components from areas of the device identified as permanently damaged.
This is especially true for SRAM-based FPGAs, which technology and architectural characteristics will be presented in Section 1.2. In fact, the flexibility in the access their SRAM configuration memory provides in reading and writing specific

portions or words of the configuration layer, coupled with the presence of both external and internal interfaces to perform such operations at run-time enable the implementation of several error-detection and recovery techniques [10][14].
These features, coupled with the high processing capabilities provided by these devices made them especially attractive for space and mission-critical application deployed radiation environments, such as satellite electronics and high-energy physics (HEP) experiment instrumentation gaining the attention of both research institutions and industries working on these domains [15][16][17].

1.2 SRAM-Based FPGAs

Field-programmable gate arrays play a key role in the evolution of Reconfigurable Computing and currently are the only commercial architectures supporting dynamic and partial reconfiguration.
As mentioned, the first FPGAs appear on the market in the '80s and originally were used mainly for glue logic implementation and for digital design prototyping. Over the past decades, FPGA technology had rapidly evolved becoming an increasingly appealing solution for the implementation of high-performance, compute-intensive, energy-efficient, and dependable applications in a wide range of fields.
In fact, today's FPGA fabric is a highly integrated hybrid platform consisting of advanced Systems on Chip (SoCs) enabling the deployment on the same support of complex applications combining both hardware and software interleaving computations.
FPGAs born as Tile-based modular architectures allowing tailoring on them combinatorial and sequential circuits thanks to an island-style topology made of logic blocks, programmable connections, and input and output ports.
In Figure 1.5 the Island-style FPGA architecture is depicted, showing its main components: the functional blocks, also called Control Logic Block (CLB), the programmable connections blocks, referred to as Switch Matrix (SM) and Input and Output blocks (IOB), relying on terminology used in the devices produced by Xilinx. As will be discussed in Section 2.1.2, different terms are used by other FPGAs vendors to name the equivalents programmable components.
In detail, the CLBs are basic configurable functional elements typically made of Look-up Tables (LUTs) and Flip-Flops (FFs) for the implementation of both logic and sequential nodes of a circuit.
IOBs are the interfaces through which the programmed circuit exchanges data with the outside environment while the SM contains the programmable switches (also called Programmable Interconnect Point, PIP) allowing the connectivity with the fixed wiring resources and implementing the routing infrastructure for the communication among CLBs and IOB.

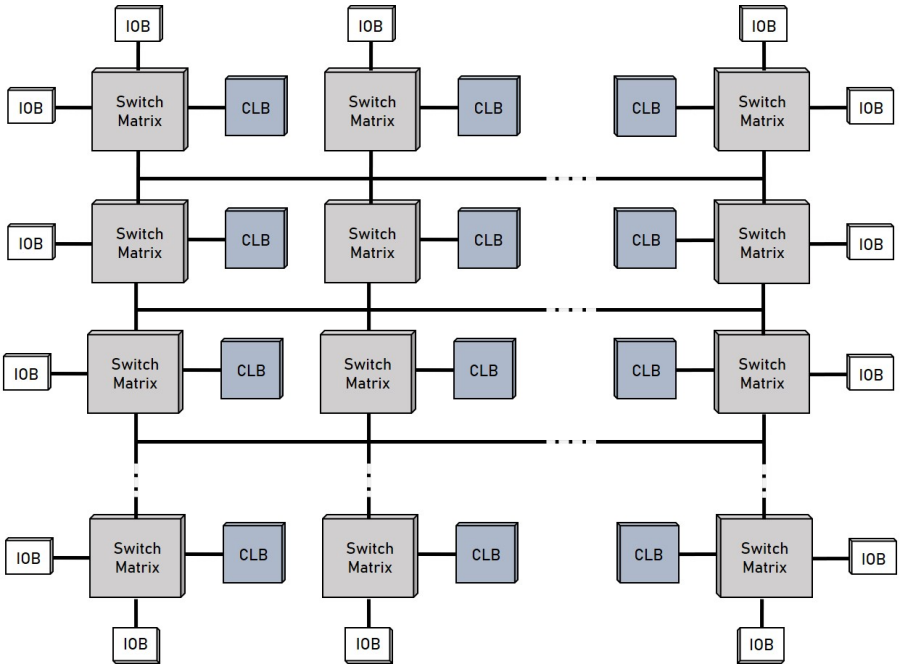

Figure 1.5: High-Level Island-Style FPGA Architecture

The circuit behavior is obtained thanks to the configuration settings stored in the configuration memory layer, which define the functionality of the logic nodes as well as their connectivity requirements among them and with the outside environment.

1.2.1 Configuration Memory

According to the technology used for the implementation of the configuration memory, FPGAs present different characteristics in terms of programmability and reconfigurability.
Today, three main categories of FPGAs are present on the market: Antifuse, Flash-based, and SRAM-based FPGAs.
FPGAs using Antifuse technology are one-time programmable as their configuration settings can be defined one single time after manufacturing through a specific electronic procedure and remain permanent for the whole product life-cycle.
FPGAs relying on both SRAM and Flash technology are instead multiple-time programmable since their configuration settings can be updated an unlimited amount of time after production.
On the other hand, Flash-based and SRAM-based FPGAs differ from the point of view of the reconfigurability as their main distinction resides in the volatility of their configuration memory.

As flash non-volatile memory retains data without power supply, Flash-based FPGAs have the advantage of not requiring to be reprogrammed at each power-up and, consequently, they do not need to rely on external memories for holding configuration data.

SRAM-based Configuration Memory

SRAM-based FPGAs rely on a volatile Static Random Access Memory (SRAM) storing the configuration settings for the reconfigurable layer. From one side, this requires the configuration memory to be refreshed at each power-up relying an external memory for the safe hold of their configuration settings. On the other hand, it enables unlimited and fast accesses both in reading and writing to different memory sections, with the additional benefit of allowing this procedure at run-time.
In detail, SRAM-based FPGAs enable the run-time access and modification at the memory segment granularity.
In current devices the smallest addressable memory segments, typically called frames, are long bit words and their link with the reconfigurable resources is regular but complex, as each one of them partially configure reconfigurable resources.
However, the periodicity and detailed accessibility of the configuration memory layer with respect the programmable layer allows to perform in-field modifications of the functionality of specific resources and components.
Thus, the easy accessibility and volatility of SRAM memory have been the key features that enable dynamic and partial reconfiguration on SRAM-based FPGAs, making them the only commercially available devices supporting the deployment of dynamically and partially reconfigurable applications [16][18].

1.2.2 Current Heterogeneous Fabric

Over time the FPGA architecture has evolved from the original island-based fabric made only by functional and connection tiles to the current heterogeneous architecture where these components interleave with more complex on-chip units.
The trend of the main FPGA vendors (e.g., Xilinx and former Altera, now Intel) follows the direction of providing increasingly integrated systems producing advanced System-on-Chip devices (SoCs) that enable the deployment of complex applications [18].
Such increased heterogeneity provided by vendors has been driven by the increasing demand for more complex and performance functionalities of today's applications becoming too resource-consuming or inefficient when tailored on distributed logic [19].
In fact, due to the increased demand for more complex and powerful arithmetic components, like adders and multipliers, vendors introduced hardwired components dedicated to Digital Signal Processing (DSPs) capable to efficiently support

the implementation of enhanced arithmetic and long-word logic units for massive computations [20]. Furthermore, dedicated hardwired Blocks of RAM (BRAMs) for user memories have been introduced to optimize the deployment of large user memories and FIFO modules, as the demand for bigger and faster memories components not efficiently implementable on distributed LUTs increases [21].
As the deployment of mixed applications that require the presence of software executions interleaved with custom hardware calculations rises, on-chip standard processors have been integrated with the programmable logic. This minimizes the usage of resources consuming soft-cores implemented in distributed logic as well as the latency required for the communications between the custom functionality on the reconfigurable fabric and external microprocessors [22].
Additionally, as the applications implementing dynamic and partial reconfiguration increase, recent FPGAs families embed dedicated interfaces (as the Xilinx Internal Configuration Access Port, ICAP [23]) allowing the interaction between the reconfigurable hardware and configuration memory layers to read or write configuration data of specific portions of the design and resources.
These interfaces can be managed by dedicated controllers on the application layer, either from programmed circuitry or from the on-chip microprocessor, and are the key components enabling the possibility to perform the dynamic and partial reconfiguration in commercial devices.
The current SRAM-based FPGA fabric is summarized in Figure 1.6, reporting the reconfigurable basic tiles interleaving with the hardwired components: DSPs units, BRAMs, the on-chip processing system, and the ICAP.

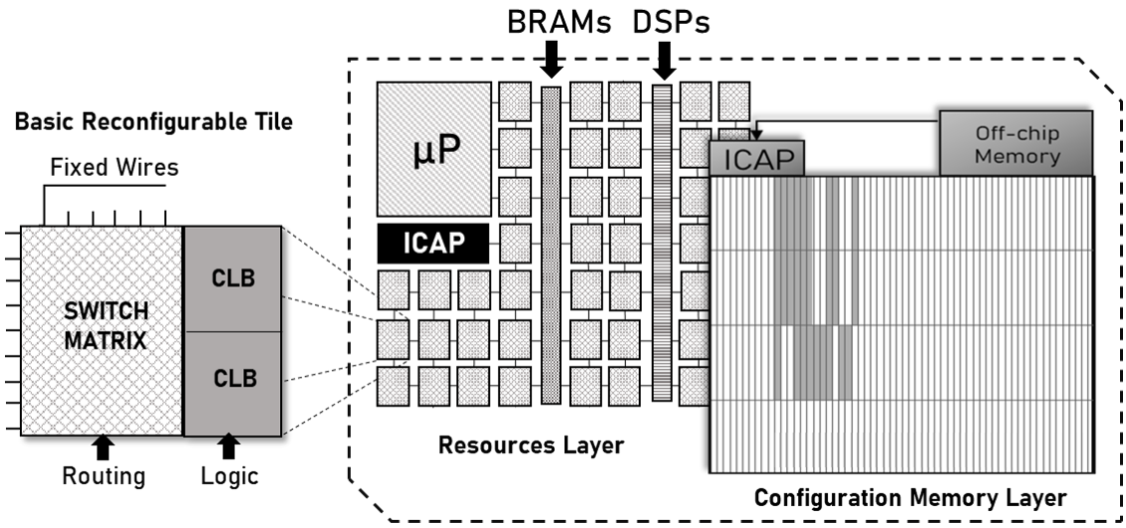

Figure 1.6: Current FPGA Heterogeneous Architecture

1.3 Opportunities and Challenges

Thanks to their high flexibility, resource density, energy efficiency, and in-field upgradability reconfigurable SRAM-based FPGAs are increasingly used for tailoring on them high-performance and reliability-oriented tasks in many fields, from computationally intensive networking and cryptography applications to mission-critical and aerospace systems.
At the same time, current FPGAs still present some architectural and technological aspects which require to be addressed to maximize the potential embedded in the Reconfigurable Computing paradigm.
In fact, the dynamically reconfigurable system performances are often bounded by the efficiency of the configuration procedure, which still requires relatively long times to be performed being strongly dependent on the amount of configuration data involved in the procedure and on the configuration mechanism used.
Furthermore, despite the advantages SRAM-based FPGAs provides in terms of computational and error-recovery capabilities for mission critical applications in harsh environments, such as satellite electronics and high energy physics experiments, when they are deployed in presence of radiation the sensitivity of SRAM cell technology to radiation-induced transient effects should be properly characterized and hardening techniques need to be accordingly introduced to prevent application failures.

1.3.1 High-performance and Reliable Applications

Today the usage of reconfigurable SRAM-based FPGAs in computationally intensive and mission-critical applications is increasingly growing in many fields, ranging from complex digital signal processing and arithmetic to networking and distributed computing [3][18].
In fact, today reconfigurable hybrid architectures allow exploiting the reconfigurable hardware to accelerate custom computations too intensive for full software executions and the added the possibility to perform in-field upgrades provides further gains in terms of performance, power consumption, and resource usage.
In fact, distributed FPGA networks are increasingly and successfully deployed in large-scale data centers accelerating image and pattern recognition, security, and cryptography applications while achieving high performances and lowering power consumption [24][25].
Above all, the fields in which the usage of FPGAs is firmly established and continuously growing is aerospace and high-energy physics experiments [18][15].
Such applications typically require complex and massive real-time operations with additional strict constraints in terms of area and power consumption budget and strong reliability requirements. Furthermore, inherent characteristics of such systems are to operate in radiation environments and to be generally designed and

deployed as a unique sample making unreasonable the cost for their realization as ASIC components.
For these reasons, reconfigurable SRAM-based FPGAs represent ideal candidates for applications deployed in this context.
In fact, thanks to the dynamic and partial reconfiguration feature it is possible to implement several optimizations by updating the system functionality to varying conditions without the need to stop the application or adjusting its performances to the effective payload required at a given time, saving both area and power. Additionally, reconfiguration enables the detection and correction of errors not only during development but even after deployment. In fact, in SRAM-based FPGA reading configuration data at run-time allows verifying their correctness by comparing them with golden values while writing configuration data enables to refresh the correct values in the memory cell to prevent or correct errors [16][15].

1.3.2 Reconfiguration Procedure Performances

As dynamic reconfiguration represent a valuable feature to gain performance and flexibility improvements for complex computing tasks, when employed it becomes an inherent part of the application execution and introduces its own time and resource overhead.
In fact, the operations performed to access the configuration memory for both reading and writing configuration data are performed through dedicated agents on the application layer, such as dedicated control units, microprocessors, and configuration interface controllers, and imply a delay that should be considered within the application timing budget.
Therefore, during the development of dynamically reconfigurable applications it is fundamental to consider and minimize such delay to perform the reconfiguration as efficiently and quickly as possible maximizing the overall performance gain achievable through run-time optimizations [6][18].
In general, reconfiguration time is strictly related to the efficiency of the reconfiguration procedure, which depends on the configuration interface characteristics and the amount of data involved, and can be summarized as in Equation 1.1:

$$t_{\text{rec}} = N_{\text{atomic memory unit}} \cdot B_{\text{bits per unit}} \cdot t_{\text{download bit}} \tag{1.1}$$

where $\text{N}_{\text{atomic memory unit}}$ represents the number of atomic addressable memory segments required to program a specific design on the target device and involved on its reconfiguration, $\text{B}_{\text{bits per unit}}$ is the data width of this addressable memory unit for the target reconfigurable fabric, and $\text{t}_{\text{download bit}}$ term is the mean time required for each bit memory transfer through the target configuration interface. Thus, the first term of Equation 1.1 is *design-dependent* while the two others are imposed by the reconfigurable *target device*.

As it will be further discussed in Chapter 4, considering the recent Xilinx 7 Series FPGAs as an example, the data width of the smallest addressable memory segment, called frame, is 3,232 bits and the time required for loading or storing it in the configuration memory through the ICAP is around 100 μs [23]. In fact, depending on the size of the target circuit, the time required for its full or partial reconfiguration could require several milliseconds (ms).
Therefore, when relying on commercial devices, the addressable memory segments size and configuration interfaces are fixed constraints. Although the minimization of the reconfiguration overhead perceived by the user can be achieved by cleverly managing the reconfiguration scheduling to hide this delay within the application execution, the time required to perform reconfiguration is inherently bounded by the amount of configuration data. As the applicability of the first approach is strictly related to the target application and still bounded to $N_{\text{atomic memory unit}}$, to directly act on the minimization of this parameter represents an effective and more general solution to address the reconfiguration time overhead [18].
In detail, as it will be further explained in Chapter 3 and 5, a successful and generalized approach for the reconfiguration time optimization consists of the minimization of $N_{\text{atomic memory unit}}$ during the design implementation phase, which is the process that translates the behavioral circuit description into configuration memory data and thus is not related to specific applications [26][27].
Besides, these solutions show their major efficacy when exploited in *module-based reconfiguration*, and thus when partial dynamic reconfiguration is applied to defined sub-regions of the design containing the whole target task to be allocated, updated, or substituted.
When dynamic and partial reconfiguration must be applied to specific resources or small design sub-regions to make minor and very frequent modifications on detailed configuration memory words, it is defined as *difference-based reconfiguration*. In this case, the current architecture of commercial FPGAs could be strongly limiting. In fact, even if a single resource configuration is defined by a small number of bits, the configuration memory organization is arranged in vertical frames, composed of bits partially configuring elements even distant among each other rather than be mapped to single components.
Thus, even to make detailed modifications on single resources the amount of configuration data involved in the process is disproportionally higher. Additionally, for such detailed and frequent modifications, the overhead in terms of area, resources, and latency introduced by the current configuration mechanism based a controller agent managing the reconfiguration interface could be unjustified.
Thus, as it will be further discussed in Chapter 3 and 6, to maximize the performance benefits achievable through fine-grained and frequent resources reconfiguration, novel reconfigurable architectures and mechanisms should be investigated, as the Equation 1.1 parameters to be addressed are $B_{bits\ per\ unit}$ and $t_{download\ bit}$, which are device-dependent and embedded in the current FPGA fabric.

1.3.3 Reconfigurable FPGAs Dependability in Radiation Environment

As anticipated in Section 1.1.3 and further discussed in Section 2.2.2 and Chapter 7, the benefits of performing in-field upgrades and to implement error self-detection and self-recovery techniques had driven the increasing interest in the usage of reconfigurable FPGAs in space and mission-critical application deployed radiation environments, such as satellite electronics and high-energy physics (HEP) experiment instrumentation [15][14].
However, as for all semiconductor technologies, long-term FPGAs exposure to high radiation doses implies damaging effects which can either be permanently disruptive or produce unwanted and unexpected transient misbehavior in the application. The impact and the related failure modes induced by radiation on programmable devices strongly depend on the environment radiation levels and by the supporting technology.
In general, long exposure to radiation source has over time a cumulative and permanent damaging effect on MOS technologies, altering their working characteristics (e.g., threshold voltages, delay, leakage current), and this effect is considered as the maximum Total Ionizing Dose (TID) a device can receive before interrupting to meet its functional requirements.
Differently, single ionizing particles with high energy crossing the lattice can produce instantaneous and transient effects which impact depends on the quantity of energy transferred by the particle. These phenomena fall in the macro-category of the Single Event Effects (SEEs), which can lead to different misbehavior and failure modes.
The most relevant and common effects among SEEs are the Single Event Upset (SEU), consisting of a change of the state of one or more memory cells, and the Single Event Transient (SET), consisting of a transient voltage glitch that can propagate through the circuit and, if sampled by a register, become an SEU [15][28].

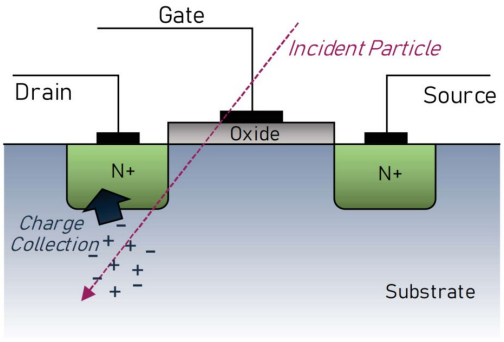

Figure 1.7: Ionizing Particle Effect on MOS Technology

FPGAs Sensitivity to Radiation Effects

The FPGAs sensitivity versus radiation effects depends on the technology on which they rely. In fact, both Antifuse and Flash-based FPGAs are in general more robust versus SEUs and SETs, which mainly concern user registers and memories rather than the configuration layer. On the other hand, Antifuse and Flash-based FPGAs are more sensitive to long term and permanent TID effects.
Conversely, while having a higher TID threshold, the major concern for SRAM-based FPGAs is represented by their configuration memory vulnerability to SEUs. In fact, in dynamically reconfigurable FPGAs, the SRAM cells of the configuration memory present a high susceptibility to upsets caused by high-energy particles changing their states.
Even if these effects, also referred as soft-errors or bitflips, are not permanent and can be recovered by refreshing the correct value inside the storage element through reconfiguration, they can still be highly critical for the correct system operation, as it is expressly defined by the configuration memory content [28][29].

Dependability Characterization Approaches

The increasing usage of reconfigurable FPGAs in high-performance and mission-critical systems deployed in presence of radiation makes essential to perform accurate analysis and validations of target devices and applications according to the target deployment environment [15][30].
In fact, different radiation profiles imply different doses and spectrum which have different interactions with semiconductor devices and applications. Systems deployed in space environments have to face complex and varying radiation conditions over time and location, HEP equipment has to tolerate radiation profiles specific to the experiment and depending on its distance to the source, while even some terrestrial systems, as the ones requiring high reliability, consisting of large clusters, or located in specific areas, can suffer the effects of rare radiation particles.
Thus, to characterize system dependability according to the deployment environment, three main approaches are possible: radiation testing, fault-injection campaigns, and analytical methods.
Radiation testing consists of exposing the device to radiation sources able to mimic the target environment radiation profile to experimentally observe its impact on the configuration memory. Fault-injections consist of emulating the radiation effects by loading corrupted configuration data into the FPGA configuration memory and represent an accurate approach for SEUs evaluation. Finally, the analytical methods consist of software programs and tools able to estimate the device and application dependability versus specific radiation profiles according to previously computed models [30].

Hardening Techniques

In addition to dependability analysis and characterization, many techniques have been proposed and implemented to mitigate, cover, or repairs errors in the configuration memory, and the most consolidated rely on the concepts of redundancy and repairs.
Redundancy can be applied both *temporally* and *structurally* and respectively consists of replicating operations or modules to avoid single points of failures.
In FPGAs, the most exploited structural redundancy is the Triple-Modular Redundancy (TMR) consisting of a three-times replication of hardware modules at a given granularity and majority-voting their outputs to mask errors [31][32].
As repairing technique, the most consolidated and exploited in FPGAs, and possible thanks to dynamic partial reconfiguration, consists of periodically re-writing configuration memory data to correct or prevent upsets accumulation in memory cells. This procedure is called Scrubbing and can be performed blindly on the whole configuration memory, selectively on memory segments detected as corrupted from a read-back procedure or with hybrid approaches [10][33].
In general, the best approach to guarantee reliability in SRAM-based FPGAs follows the Markov model of *Redundancy with Repair*, consisting of coupling the Scrubbing recovery capability with the TMR approach to mask errors possibly rising between repair cycles [15].

1.4 Main Objectives and Contributions

Currently, SRAM-based FPGAs are the only commercial devices supporting dynamic partial reconfiguration. Although they had proven to provide great benefits in terms of high integration, performance, and in-field upgradability, these platforms present some characteristics that must be further optimized and addressed to increase their effectiveness when targeting high-performance and reliable reconfigurable applications.
One of the main aspects to be tackled to enhance reconfigurable system performances consists of the optimization of the time overhead required to perform run-time reconfiguration that, if not properly managed, can jeopardize the performance gain achieved through to the in-field optimizations.
This overhead is strongly related to the amount of data involved in the run-time updates and the mechanism used to perform this procedure.
In this view, the research carried out to address *dynamically and partially reconfigurable architectures performance* has been focused on novel approaches to optimize the configuration data in commercial devices and novel architectural solutions for the reconfigurable fabric.
In detail, the main contributions on the optimization of the reconfiguration time overhead presented in the *first part* of the dissertation consist of:

- The in-depth study of the link among the reconfigurable resources and configuration memory layers in state-of-the-art commercial devices.
- The development of the Frame-driven Routing Algorithm (FeDRA), a generalized approach to be integrated into the development process of commercial FPGAs for the optimization of the configuration data used for the target circuit and that is able to produce a consistent reduction of the reconfiguration time overhead if compared with solutions obtained with the standard process.
- The design of a novel reconfigurable cell, called Reconfigurable Multipotent (ReM) Cell, able to implement bit-level reconfiguration within a single clock cycle involving a minimal amount of configuration data through the key feature of reconfiguring itself and the surrounding units, facing the limitations of the commercial architectures.

Furthermore, SRAM-based FPGAs are often integrated into high-performance and mission-critical applications, as in aerospace and HEP domains, which operate in presence of radiation.
To guarantee the reliability of these applications, one of the main aspects which demand to be addressed consists of the SRAM configuration memory sensitivity to radiation-induced errors, which must be carefully evaluated before deployment to characterize the system dependability according to the mission environment to avoid failures during operations.
Therefore, the research carried out to address *self-reconfigurable systems dependability* has been focused on an in-depth analysis of the radiation sensitivity and failure modes of the key component managing the in-field configuration memory access in dynamically and partially reconfigurable applications, enabling its usage in the instrument a cost-effective setup for fast and efficient FPGA radiation testing.
In detail, the main contributions on the self-reconfigurable system dependability characterization presented in the *second part* of the dissertation consist of:

- A soft-error analysis of different implementations of the self-reconfiguration controller that has been performed by emulating different radiation profiles through detailed fault injections and that provided indications about the controllers' dependability and applicability in different radiation environments according to their operational goals.
- The instrumentation of a self-monitoring setup for the online and automated radiation analysis of SRAM configuration memory sensitivity that uses as core component the controller identified as the most suitable in the previous analysis and that strongly reduce the time and the cost required by typical radiation testing facilities and approaches.

Organization

The content of the dissertation is organized into two main parts, the **Part 1** dedicated to the presentation of the main contributions on dynamically reconfigurable systems performance while the **Part 2** to the ones on self-reconfigurable systems dependability analysis.
In detail, *Chapter 2* provides an overview of the state of the art in reconfigurable computing with a major focus on architectural models, commercial devices, and their deployment in high-performance and high-dependability applications, and covering aspects which will be addressed in both the dissertation parts.
Part 1 follows, providing the preliminaries and the contributions on dynamically reconfigurable systems performance.
Firstly, the related works to contextualize the proposed approaches are provided in *Chapter 3* discussing the FPGA development flow and its optimizations in the domain of dynamically and partially reconfigurable applications as well as providing an overview of reconfigurable processing systems and architectural solutions exploring different trade-offs among granularity, heterogeneity, and configurability.
Subsequently, in *Chapter 4* the technical background to support the presentation of FeDRA and ReM is provided focusing on the commercial reconfigurable fabric organization and configuration memory structure, discussing the complex link between them, and presenting the typical reconfiguration techniques and interfaces and the relative bottlenecks.
In *Chapter 5* the Frame-driven Routing Algorithm (FeDRA) is presented, discussing the key idea behind its development, its implementation, the gain achieved through its optimization over standard approaches in different scenarios, and the analysis performed on its algorithmic performances.
Chapter 6 presents the Reconfigurable Multipotent (ReM) Cell, detailing its architectural model, highlighting the benefits achieved through distributed reconfigurable architectures for fast and real fine-grained dynamic self-reconfiguration. The implementation of several benchmark circuits on the proposed architecture is discussed, followed by a comparison with the state-of-the-art FPGA in terms of configuration bits and time.
Part 2 follows, providing the preliminaries and the contributions on reconfigurable system dependability characterization.
In *Chapter 7,* the related works and background about the opportunities and challenges involved in the deployment of dynamically reconfigurable SRAM-based FPGAs in radiation environments are provided to contextualize the achievements of the proposed dependability analysis.
Subsequently, in *Chapter 8* the analysis performed on self-reconfiguration controllers dependability versus soft-errors is firstly presented, discussing the methodology used and the obtained results. Follows the presentation of the cost-effective and efficient radiation testing instrumentation for FPGA radiation analyses, which

relies on the self-reconfiguration controller identified as the most suitable, and the gain achieved through the proposed setup in terms of test time and costs is discussed.
Finally, *Chapter 9* concludes the dissertation summarizing and discussing the main research advancements and the foreseen future developments.

Chapter 2

State of the Art

In this chapter the trend towards the current dynamically reconfigurable devices and architectures is discussed with a major focus on the vendors and applications directions on increasingly integrated and heterogeneous platforms. Subsequently, the state-of-the-art architectural model and benefits for the deployment of hybrid and modular reconfigurable systems in the fields of high-performance and high-dependability computing are discussed together with an overview of innovative and successful applications in these contexts.

2.1 Reconfigurable Devices

Over the past decades, the rapid growth in popularity of reconfigurable FPGAs and their applicability range drove both market and academia in the design and development of devices able to efficiently support dynamically and partially reconfigurable applications.
The research exploration and the commercial spread of the earliest reconfigurable architectural models motivated vendors on the development of increasingly heterogeneous devices, which evolution and description will be presented in the following. In detail, this process started from the elder Island-style architecture until the current FPGA-based platforms integrating multiple on-chip processors, peripherals, and interfaces, to fulfill the growing computational complexity and flexibility requirements of today's cutting-edge applications.

2.1.1 Milestone Architectures

In the '90s many architectural approaches have been investigated and proposed by both academia and industries exploring different levels of granularity and reconfigurability to efficiently supports varying reconfigurable applications.

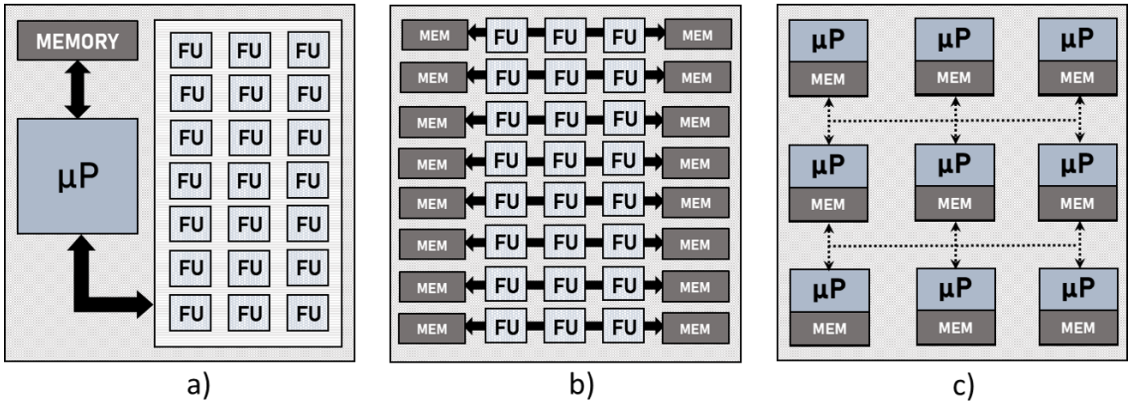

Figure 2.1: Architectural Models: a) Hybrid Architecture, b) Array of Functional Units, c) Array of Processors

The main architectural models that have emerged fall into three main classes summarized in Figure 2.1: the *Hybrid Architectures*, consisting of reconfigurable fabrics of different granularities coupled with a host processor managing both the array configuration and the main computing routine; the *Arrays of Functional Units*, consisting of reprogrammable units of variable sizes with their dedicated configuration manager; and the *Array of Processors*, coarse-grained architectures consisting of cluster of standard processors which programmability is implemented in their programmed software routine and the reconfigurable inter-processor connection network [7].
Several architectures have been proposed in those years according to these architectural classes as models or prototypes, while few have reached the silicon implementation.
Among the most remarkable there are MATRIX (Multiple ALU architecture with Reconfigurable Interconnect), developed at the Massachusetts Institute of Technology (MIT)[34], and REMARC (Reconfigurable Multimedia Array Coprocessor) designed at Stanford [35], both consisting of Coarse-grained *Arrays of Functional Units* and mainly oriented to data-parallel and multimedia applications.
As an *Array of Processors*, the most relevant is RAW (Reconfigurable Architecture Workstation) from MIT [36] consisting of a mixed-granularity 2-D mesh scalable array of simple processors interconnected by a reconfigurable network and oriented to support general-purpose computations.
Furthermore, among the most renowned reconfigurable architectures, there are MorphoSys [37] and GARP [38], respectively developed by the Irvin and Barkley Universities of California in the late '90s. Both are based on the *Hybrid Architecture* model and thus consist of a main processing system coupled with a fabric of reconfigurable units.
In detail, MorphoSys relies on TinyRISC processor, its reconfigurable array is made

by coarse-grained Arithmetic Logic Units (ALUs) and its application domain is focused on data-parallel and image processing applications, while GARP uses a standard MIPS processor coupled with a fine-grained reconfigurable logic for the acceleration of critical sections of its execution.
Although many of the proposed reconfigurable architectures proved to be efficient either for research purposes or for their effective commercial deployment, the architectural model that emerged for the effective implementation of dynamically reconfigurable applications is the *Hybrid Architecture*.
In fact, the research exploration of the earliest reconfigurable architectural models has inspired and accompanied vendors on the development of commercial reconfigurable devices, moving from the elder Island-style FPGA architecture composed by a regular array of basic reconfigurable Functional Units (FUs) to the current hybrid, heterogeneous and highly integrated platforms, such as Multiprocessor Systems-on-a-Chip (MPSoCs) and SoC with Embedded FPGAs (eFPGA), as summarized in Figure 2.2 and discussed in the following [3][7][18].

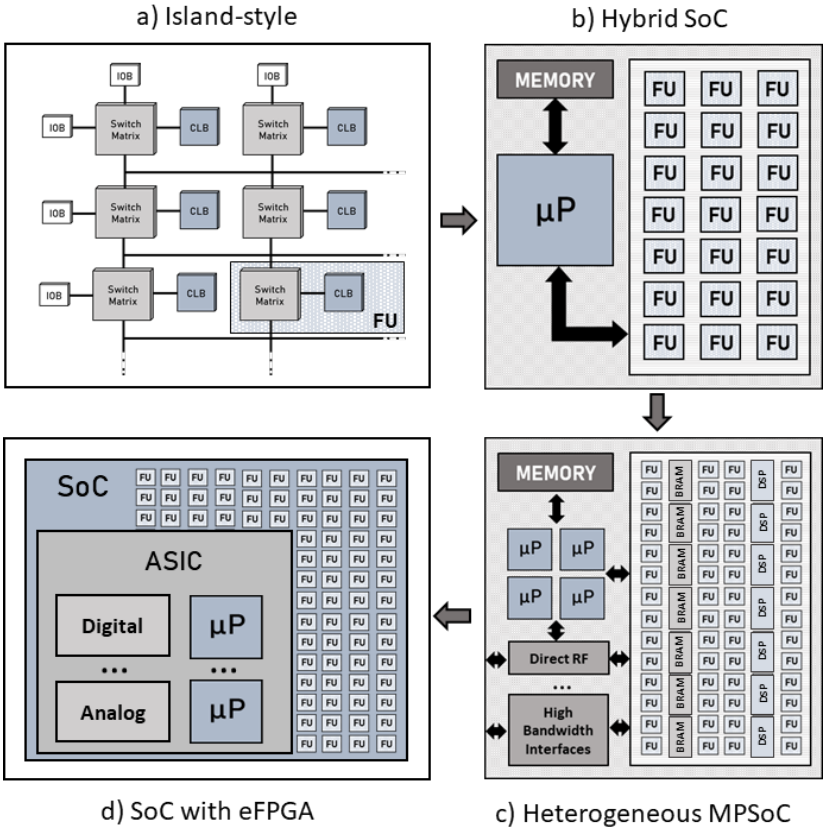

Figure 2.2: Architectural Trend towards Higly Integrated and Heterogeneous Devices: a) Island-bases Architecture, b) Hybrid SoC Architecture, c) Heterogeneous MPSoC Architecture, d) SoC with Embedded FPGA (eFPGA) Architecture

2.1.2 Commercial Devices

Although many vendors and companies have been present over the time on the market producing FPGAs and programmable fabrics, such as Atmel and Tabula, historically the two main producers of reconfigurable devices have been Xilinx and Altera.
In the last decade, the growing demand for platforms supporting increasingly complex and intensive computations drove the evolution of commercial reconfigurable devices from simple arrays of programmable functional units to heterogeneous and hybrid devices.
This trend has been foreseen and corroborated by the acquisitions and merging of the main processors and reconfigurable devices vendors. In fact, Altera has been acquired by Intel, former Atmel and Microsemi have been absorbed by Microchip, while very recently Xilinx has been integrated within AMD.
As it will be discussed in the following, today these companies are leading the market in the production of highly integrated heterogeneous platforms integrating ad-hoc reconfigurable fabrics, high-performance multiple processors, digital, analog and radio frequency components, as well as dedicated accelerators.

Xilinx

Xilinx was the first company to introduce on the market dynamically reconfigurable devices, consisting of the XC600 family produced in the middle '90s and highly resembling the *Array of Functional Unit* model [39]. Although its highly regular connection among configuration memory data and configurable resources made run-time reconfiguration simpler than in modern devices, dynamic partial reconfiguration gained higher popularity with later Xilinx devices.
In fact, in the fabric of later Virtex-II and Virtex-II Pro the complexity and the computational capability of the basic CLB element was increased and more advanced primitives for dedicated computations were integrated, as blocks for user memories and multipliers [40][41]. Additionally, these have been the first devices in which was possible to perform in-field self-upgrades thanks to the introduction of an internal configuration access port, namely the ICAP [42].
Towards Virtex-4, Virtex-5, and Virtex-6 architectures, several improvements have been introduced to ease and make safer the placement and the connectivity with the reconfigurable areas of the design avoiding signal integrity violation. Additionally, the width of the ICAP interface was increased to enhance the reconfiguration throughput and efficiency while additional integration and performances were provided through more capable Block RAMs and DSP units [43][44][45].
The more recent Xilinx 7 Series, including Artix-7, Kintex-7, and Virtex-7 presented an architecture highly similar to their predecessor, but increasing the resource quantity and density, and thus furtherly enhancing design integration and resource optimizations [19][23]. Subsequently, Xilinx introduced the Zynq *Hybrid*

Architecture, based on the same 7 Series programmable logic architecture but integrating hardwired microprocessors into a reconfigurable fabric as for the Xilinx Zynq-7000 SoC which integrates a Dual-Core ARM processor.
The integrated processing system is instrumented to communicate with on-chip memories and peripherals and allows flexible and scalable interconnections with the circuitry programmed on reconfigurable fabric thanks to the possibility to rely on the Advanced eXtensible Interface (AXI) Interconnect [46]. This, coupled with the introduction of an additional configuration interface between the processing system and the reconfigurable fabric, namely the Processor Configuration Access Port (PCAP), strongly improves the efficiency and communication opportunities for the implementation of more performant modular and dynamically reconfigurable applications [22][47].
Furthermore, the later Xilinx Ultrascale and Ultrascale+ FPGA families based on a different transistor technology provide additional integration and capability improvements, also in terms of dynamic reconfiguration by enabling the reconfigurability of primitives which were static in the previous architectures (as PLLs and IO buffers) and introducing an additional configuration port, the Media Configuration Access Port (MCAP), similar to the ICAP but providing access to additional media and communication hard-macros [48].
Finally, following the direction of growing demands in today cutting-edge technology in the fields of artificial intelligence, massive networking, parallel computation, and advanced autonomous systems, Xilinx increased in its devices the supports for these features, moving from SoC to MPSoCs with multiple on-chip processing systems, and introducing RFSoCs devices, consisting of MPSoCs with additional Radio Frequency interfaces and components [49].
The latest Xilinx release consists of even more heterogeneous and flexible support as the Adaptive Compute Acceleration Platform (ACAP), which provides on a single platform high-performance software computation, dedicated acceleration hardware, as well as a dynamically reconfigurable fabric for run-time adaptability [50].

Intel – former Altera

Altera, which has been acquired by Intel in 2015, started to produce dynamically reconfigurable FPGAs some year after Xilinx and the architectural evolution of its devices followed a similar trend.
The earliest dynamically reconfigurable FPGAs developed by Altera have been Stratix-V, Cyclone-V, and Arria-10 series, with basic functional units made of LUTs and FFs , as the Xilinx CLB, and called Adaptive Logic Modules (ALMs). These elements are coupled with additional circuitry for a more efficient support of arithmetic computations and primitives for DSP and user memory implementation [18].
The programmability of logic nodes and routing switches relayed on a mechanism similar to Xilinx devices, using analogous approaches also for preserving the signal

integrity at the reconfigurable module boundaries.
As the Stratix architecture had many similarities with Virtex Series [51], also the more recent Arria-10 and Arria-10 SoC follow the *Hybrid Architectural* model of the 7 Series and SoC Xilinx FPGA, relying on an ARM-based processing system integrated with the reconfigurable fabric and embedding dedicated interfaces among programmable and memory layers for both internal and external dynamic partial reconfiguration [52].
More recently, following the higher demand for computation acceleration, virtualization, and networking, Intel strated to deploy more complex and heterogeneous platforms integrating high-density and dynamically reconfigurable Stratix-10 FPGAs, high-performance scalable Xeon processors, Ethernet connectivity, and fast communication interfaces [53]. Furthermore, Intel is proposing novel architectural models as structured ASIC or eASIC, which merge the features of both ASICs and FPGAs by allowing their customization during manufacturing to support specific computation through the integration of dedicated hardwired processing systems with the selected peripherals and customized portions of dynamically reconfigurable logic [53].

Other Vendors

Besides Xilinx and Altera other vendors have produced dynamically reconfigurable devices with different architectural solutions. Some of the most remarkable in past had eventually left this market, others have been absorbed or repurposed, while more recent companies are currently opening their way proposing new integration solutions.
Among the elder most interesting devices, there are the CLAy fine-grained reconfigurable architecture from National Semiconductor based on an external host for the low-level reconfiguration [54], the more recent AT40K FPGA from Atmel based on memory-mapped-like reconfiguration mechanism and particularly optimized for adaptive filters and acceleration [55], and ABAX from Tabula consisting of a novel programmable technology called *Spacetime* and based on multi-context reconfiguration [56].
Although each of these devices embedded promising features, their limited adoption mainly related to poor logic density, weak development toolchains, or high power consumption, eventually made Xilinx and Altera prevailing on the market [18].
However, Atmel together with Microsemi had been later absorbed by Microchip, which today is highly present on the market manufacturing highly-integrated programmable Flash-based FPGAs and MPSoC devices, such as IGLOO, SmartFusion, and PolarFire. In general, Microchip produces devices especially valuable for their low-power and high-reliability which are successfully deployed in aerospace, defense, and communication applications.

Furthermore, the novel concept of Embedded FPGA (eFPGA) has recently appeared on the market opening the way to other vendors such as Achornix and Menta. An eFPGA consists of a dedicated portion of reconfigurable fabric that could be integrated into custom SoC or ASIC during the manufacturing process, allowing the production of chips with the precise amount of needed dynamically reconfigurable tiles and dedicated hardwired resources.
This allows to considerably lower production costs, minimize area, and increase specific application performances relying on dedicated and ad-hoc reconfigurable accelerators integrated with custom components [57][58].

2.2 High-Performance and Dependable Systems

Dynamically and Partially Reconfigurable platforms based on SRAM-based FPGAs have increasingly gained attention in the field of high-performance and mission-critical applications.
In fact, the computational capabilities of modern heterogeneous FPGA-based platforms, coupled with their high integration, make them highly suitable for complex computationally intensive applications, which could gain tremendous benefits from their run-time upgradability.
The opportunity of performing in-field optimizations is valuable for those applications running computationally intensive tasks which would gain higher efficiency by adapting their purpose and effort according to the instantaneous needs as well as for those systems with strong dependability requirements, either related to the criticality of their mission or their deployment in harsh environments, for which self-testing and self-repair could be fundamental.
In fact, today's heterogeneous platforms are successfully deployed for running mission-critical tasks in radiation environments and are increasingly used in high-performance applications, such as networking, Cloud, and Exascale computing [15][59][60].

2.2.1 Dynamically Reconfigurable Hybrid Architecture

Nowadays, the architectural model for the implementation of dynamically reconfigurable applications tailored on state-of-the-art devices consists of a hybrid modular architecture composed of multiple reconfigurable regions interacting and cooperating with static and on-chip components executing those tasks which do not requires run-time upgrades.
This organization takes the name of Dynamically Reconfigurable Processing Module (DRPM) and it is one of the most used for the deployment of computationally intensive tasks requiring in-field upgradability for adaptability, acceleration, or dependability purposes [17].

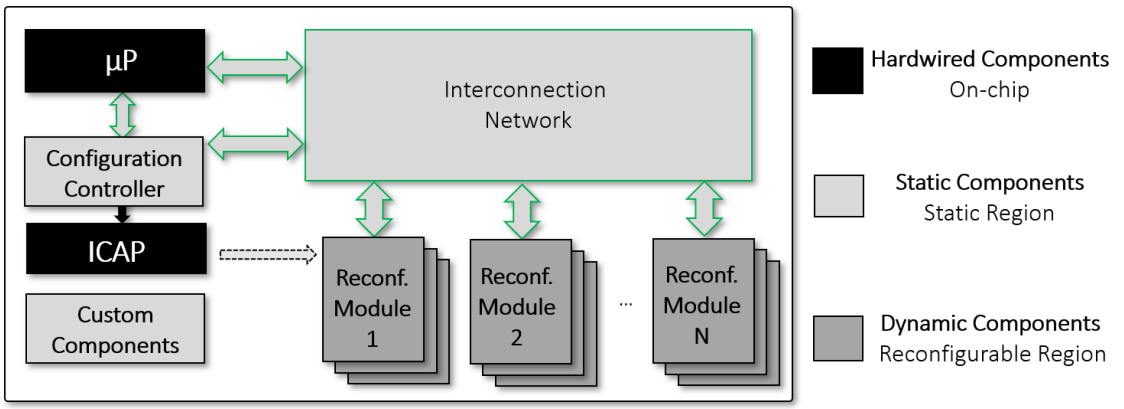

Figure 2.3: DRPM Architecture Overview

The DPRM provides many opportunities for the execution of high-performance and real-time applications, as well as for the implementation of error detection and correction mechanisms through the in-field scheduling, relocation, and refresh of hardware tasks on the dynamic reconfigurable areas.
In detail, the DRPM architectural model consists of three main areas, the Hardwired Region, the Static Region, and the Reconfigurable Region, and can be efficiently mapped on current heterogenous SRAM-based FPGA SoC devices, as depicted in Figure 2.3.
In fact, the Hardwired Region consists of on-chip components, like Xilinx Zynq microprocessors [22], internal configuration access ports (e.g., ICAP, PCAP, and MCAP [47]), external memories, and digital or analog onboard application-specific circuitry.
The Static Region instead, consists of all those elements that manage, exploit, and interact with components in the Hardwired and Reconfigurable Region that are deployed in the programmable fabric but are not subject to dynamic and partial reconfiguration, such as self-configuration controllers, custom Intellectual Property (IP) cores, and the interconnection network among all these elements, which is typically called Bus Macro and can be efficiently implemented in commercial devices through AXI interfaces and Interconnects [46].
Finally, the Reconfigurable Region consists of several dynamically reconfigurable partitions (or modules, RM) where components and circuitry can be allocated and deallocated at run-time according to the system mission and instantaneous requirements through dynamic partial reconfiguration, such as dedicated accelerators which functionality could demand to be updated and critical cores which can achieve higher dependability by refreshing or displacing their configuration settings.
This modular architecture provides high scalability to computing tasks versus varying conditions and payloads enabling the time-sharing among reconfigurable resources to increase area efficiency, tune power consumption, and flexibly adapt to

computational requirements and impairments changing over time to improve the application performance and dependability [8][9][10].

DRPM Performances

To maximize the computational and flexibility gains provided by these architectures many optimizations could be introduced. In fact, besides the optimization related to the in-field reconfiguration time and mechanism that will be further discussed in Chapter 3, the efficient floorplanning of reconfigurable partitions and the implementation of fast and safe data transfers among Hardwired, Static, and Reconfigurable components are key aspects to be tackled in the designs of DRPMs.
In fact, especially when the number of reconfigurable partitions for the dynamic allocation of different hardware tasks with different connectivity and floorplanning requirements increases, the optimization of interconnections and the resource usage as well as the efficient placement of dynamically reconfigurable partitions have a crucial role in the performance achievable by the application.
The main criticalities involved in the interconnection network design are due to the varying requirements related to the different reconfigurable cores implemented over time in the same dynamic partitions, to the intense data traffic to be supported, and to possible signal integrity violations at the boundaries between the Static and Reconfigurable regions during reconfiguration.
Tackling this aspect as a further contribution to the optimization of dynamically and partially reconfigurable architecture performances, the author has developed and proposed in [61] a light and efficient Interface-based Interconnection Structure (IbIS) for DRPM architectures which presents a relevant reduction of the routing congestion, resource usage, and delays and efficiently scales as the number of reconfigurable modules increases when compared with the state-of-the-art AXI and Wishbone Crossbar. This has been possible by designing a light and predictable interface considering its detailed mapping on the basic reconfigurable resources at module boundary and implementing a simple communication protocol based on few control signals mastered by the host processor, reducing resource explosions and delays as well as ensuring signal integrity.
Other works in literature have addressed these points focusing on minimizing the resource overhead introduced by the interconnection macro and reconfigurable partition floorplanning while maximizing the reliability of overall infrastructure in modular dynamically reconfigurable architectures, like the one discussed in [62]. In detail, the proposed solution improves the efficiency of reconfigurable modules placement and their communication infrastructure avoiding placement and signal integrity violations thanks to a data-graph overlapping approach for the optimal selection of the areas for the placement of the reconfigurable partitions and through an ad-hoc and resource-efficient communication macro highly adaptable to different modular architectures.

DRPM for Dependability

Another important aspect of dynamically reconfigurable modular architectures where is possible to introduce relevant optimizations is represented by the opportunities they provide in terms of self-test and self-repair capabilities for the enhancement of their dependability and availability, extending the system lifetime.
In fact, modules dedicated to the implementation of error-detection techniques could be implemented in the Hardwired or Static Regions, while critical circuits deployed in Reconfigurable Region could be moved away from permanently damaged areas of the device or correcting on the field transient errors in faulty modules by refreshing their configuration settings.
These features have been remarkably exploited in [63] where a hardened dedicated agent has been introduced on the Static Region to detect transient faults in the Dynamic Region, with the added capability to temporarily switch off modules to avoid error propagation while exploiting partial reconfiguration to restore the correct functioning in target module.
Another approach leveraging the modularity of the architecture has been proposed in [64], where dedicated and synergic self-repairing techniques are implemented according to the modules failure modes, enabling to perform selective transient error recovering through partial scrubbing while reverting to the dynamic module relocation when the errors result persistent.

2.2.2 Dependable Platforms for Radiation Environments

As anticipated in Section 1.1.3 and 2.2.1, dynamically reconfigurable modular architectures can offer many opportunities for the deployment of fault-tolerant and high-performance processing units. Therefore, their usage in mission-critical and dependable applications in harsh environments is increasingly growing.
In fact, the features provided by dynamic and partial reconfiguration coupled with the system modularity allows the implementation of a plurality of mitigations techniques, which could be implemented in Commercial-of-the-Shelf (COTS) devices enabling and broadening the usage of non space-grade SRAM-based FPGAs in high dependability applications deployed in radiation environments.
In fact, Xilinx and other vendors offer specific devices qualified as Radiation-Hardened and Radiation-Tolerant which are manufactured with dedicated processes to provide intrinsic higher robustness to the reconfigurable fabric versus radiation effects. However, these devices are extremely expensive and typically available for older FPGAs families as the space qualification of the state-of-the-art technology can require several years.
Therefore, both National and International research organizations, as well as academic and scientific institutions, have worked on the development of dynamically reconfigurable COTS-based platforms for aerospace electronics and high energy

physics experiments to leverage their higher availability, performance, and reduced costs.
In the following, an overview of remarkable fault-tolerant applications based on dynamically reconfigurable COTS FPGAs developed for the dependable deployment in radiation environments is provided [65][66][67][68][69].
The fault-tolerant DRPM for reliable and performant satellite payloads computation has been developed and verified within a collaboration among Politecnico di Torino and Bielefeld University and supported by the European Space Agency (ESA) and other research institutions. The proposed platform consists of a cluster of dynamically reconfigurable boards which provide high flexibility and scalability thanks to the possibility of integrating up to six units either as Communication Modules or Processing Modules that can operate both as single nodes or in tandem. Each Processing Module is deployed on an SRAM-based Virtex-4 FPGA and consists of a hybrid modular and reconfigurable architecture, made of a static area including the processing system, the self-configuration controller, and the inner interconnection macro, and a dynamic area consisting of a dynamically and partially reconfigurable array for the efficient and flexible implementation of the payload processing. Partial and dynamic reconfigurations on the target Processing Module can be triggered internally or externally by Communication Modules to scale power consumption, to recover radiation-induced soft-errors, to adapt the functionality of the dynamic array, or to activate new processing or computing nodes at run-time. Furthermore, to increase the DRPM dependability, besides the possibility to exploit according to the needs partial reconfiguration for blind and readback scrubbing, a dedicated placement strategy has been used for its implementation on the device providing additional hardening by design at the resource level. The efficacy of these techniques versus soft-errors in the configuration memory has been evaluated through analytic and fault injection approaches confirming the achievement of high fault-tolerance [65] .
Another COTS-based dynamically reconfigurable platform confirming the effective usage of COTS SRAM-based FPGA in mission-critical space applications has been developed within the Universidad Politècnica de Madrid with the support of the European Horizon 2020 Research and Innovation Program and other industrial partners [66]. The proposed platform consists of an on-board processing reconfigurable module for vision-based navigation to be deployed on mission-critical space applications on the state-of-the-art Xilinx Ultrascale+ MPSoC, and it is composed by the on-chip multi-core processing system running a real-time operating system coupled with a run-rime upgradable architecture implemented in the programmable logic and based on the dynamically reconfigurable acceleration template architecture ARTICo^3 proposed in [67] to efficiently fasten computationally intensive tasks. In this approach, reconfiguration is exploited to scale system performances and power consumption as well as to increase design fault-tolerance by implementing

error detection, isolation, and recovery. In detail, to enhance the platform dependability both temporal and structural redundancies are applied to the real-time processing system which is also in charge of managing in-field hardware upgrades, while low-latency scrubbing is performed at the reconfigurable resource level.
A remarkable example of dynamically reconfigurable platforms relying on SRAM-based FPGAs successfully deployed in radiation environments has been the Cibola Flight Experiment (CFE) satellite [68] developed and verified by the Los Alamos National Laboratory with the collaboration of Brigham Young University and supported by other governmental institutions. In detail, the CFE satellite includes three Reconfigurable Computers (RCC) made of three Xilinx Virtex reconfigurable FPGAs each and implementing a Software-Defined Radio (SDR) for ionospheric and lightning analysis. The mission goal was to guarantee the system operation in the Low Earth Orbit (LEO) for at least 4 years while reconfigurability was exploited approximately every two weeks to perform in-field updates for adapting the application processing algorithms and deploying varying mitigation techniques. Thanks to the preliminary radiation analysis and testing performed on the platform and the implemented fault-recovery and mitigation techniques, such as a CRC-based transparent Scrubbing mechanism aimed at SEU detection, correction, and information logging, enabled to extend of the CFE lifetime to 8 years, doubling the initial expectations. Furthermore, the mission allowed the collection of useful data on run-time reconfigurability for in-field upgrades and soft-error mitigations confirming the effectiveness of these methodologies for the safe deployment of reconfigurable SRAM-based FPGAs for high-dependability applications in aerospace radiation environments.
Finally, in the field of high energy physics experiments, it is worth mentioning the Reconfigurable Readout Control Unit (RCU) based on a reconfigurable Virtex II and integrated into the ALICE Time-Projection Chamber experiment at Large Hadron Collider (LHC) [69] developed at the European Organization for Nuclear Research (CERN) in collaboration with others academic and scientific institutions. In detail, the Virtex II represents the main FPGA of the RCU system and partial run-time reconfiguration is exploited to detect and recover soft-errors in the configuration memory induced by high-energy particles without the need of interrupting the application execution. The detection and recovery mechanisms rely on a support Flash-based Actel FPGA which manages the interaction with an additional flash memory holding the golden configuration data, both integrated on the same motherboard.

2.2.3 HPC: Networking, Cloud and Exascale Computing

Thanks to the vendors' effort in providing increasingly sophisticated and heterogeneous platforms, the FPGAs application in High-Performance Computing (HPC) cutting-edge applications as large scale communication, data centers, and massive

computation is rapidly growing [50][53].
In fact, in the recent years we have witnessed an increasing demand for fast and distributed communication and computation infrastructure to support new technologies oriented to both consumers and industries, such as the Internet of Things (IoT), high-bandwidth 5G wireless networking, data centers for Cloud computing, and processes automation [24][49][70].
Cloud networks consist of large clusters of computing elements supporting the capability to be repurposed and shared among different customers and to execute varying tasks. To maximize such flexibility and resource time-multiplexing is necessary to massively exploit virtualization while hiding low-level details to users and maintaining high privacy, performance, and dependability levels [59].
Furthermore, today communication infrastructures need to support a sharply increasing number of nodes providing ubiquitous and reliable connectivity with high data rates and bandwidths while maintaining latency and power consumption low. As the novel 5G network represents a promising technology to fulfill such requirements by supporting low-power and low-latency distributed wireless infrastructures for a large number of different platforms, such as mobiles and IoT devices as well as edge and cloud serves, the demand for virtualization, scalability, and flexibility increases [49][53].
This technology trends have been supported by the vendors' effort in producing heterogeneous reconfigurable platforms providing higher integration, connectivity, and acceleration opportunities. Therefore, today reconfigurable FPGA-based systems including high-performance processing systems and both analog and RF components are effectively becoming an optimal solution to fulfill these computational requirements for the implementation of single computing nodes or as platforms integrated into large-scale systems deployed in cloud servers, 5G networks, and high-performance Exascale applications [24][50].
In fact, large-clusters of FGPAs have been already successfully deployed on commercial servers, such as the in F1 instances of Amazon Web Service EC2 [71] and in the Bing Search Engine to increase the performance of its ranking algorithm [72]. In detail, Amazon AWS has integrated Xilinx Ultrascale+ devices within its Cloud Servers while the Microsoft Catapult Project succeeded in accelerating the ranking algorithm of the Bing Search Engine through the usage of reconfigurable Intel FPGAs obtaining a significant improvement in terms of speed and power consumption [73].
As this confirmed the many opportunities reconfigurable hardware can provide for instances virtualization and dynamic adaptation to varying computing and transmission requirements, FPGAs have become an appealing solution to increase devices integration for 5G, edge, and cloud computing applications [50][53].
In fact, FPGAs dynamic and partial in-field upgradability make them highly suitable for the implementation of 5G network nodes enabling run-time adaption to

different functions as well as to different power, latency, and communication conditions. This, coupled with their high working frequency, and efficiency in real-time computation, makes them an optimal candidate for the deployment of radio signal processing. Finally, reconfigurable FPGA modular and parallel architectural provides many opportunities for efficient multi-kernel computations and virtualization [59].
The benefits that FPGA-based systems can provide in clusters and massive computation have driven an increasing interest for their deployment in Exascale computing, the leading edge of high-performance computing targeting the execution of 10^{18} Floating Point Operations per Second (exaFLOPS).
As, in general, High-performance Computing demands high scaling capabilities and massive parallelism, Exascale architectures aim to maximize these aspects. For this reason, HPC platforms require computing nodes with high specialization and acceleration capabilities able to provide at the same time a reasonable trade-off with power consumption.
In this context, reconfigurable FPGAs can play a key role thanks to their flexibility versus varying computational requirements and the acceleration opportunities their architectural and computing models can provide [60]. Thus, there is an increasing focus on the efficient integration of reconfigurable platforms in Exascale Systems which have been corroborated by the European project euroEXA which aims at the implementation of the first Exascale supercomputer using low budget nodes consisting of low-power microprocessors and COTS SRAM-based FPGAs [74].
Within euroEXA project framework, the three main sub-projects focused on reconfigurable FPGAs are ExaNeSt [75], ExaNode [76], and ECOSCALE [77].
The ExaNeSt Project focuses on the development of the ExaNeSt node consisting of a platform made of four Zynq Ultrascale FPGAs. Two of them are used as computing nodes, one is used for external memory management and interfacing, while the latter implements the platform interconnection network.
ExaNode project consisted of the design of ExaNode, a prototype node integrating two Xilinx Zynq Ultrascale+ MPSoCs, and the characterization of its fabrication process, while ECOSCALE project focuses on the design of reconfigurable and scalable FPGA-based accelerators for Exascale computation through the development of the supporting toolchain for architectural optimizations in terms of latency, delay, and data traffic.

2.3 Discussion and Highlights

As discussed in this chapter, the growth in popularity and applicability domains of reconfigurable FPGAs has driven the effort of academia and industries in designing and producing increasingly heterogeneous devices able to support dynamically and partially reconfigurable applications to fulfill market requirements.

This trend enabled the earliest Island-style reconfigurable hardware fabric to evolve towards to current commercial heterogeneous FPGA-based platform, which demonstrates its benefits in meeting and enhancing the growing computational capability and flexibility demanded by today's cutting-edge applications.
In detail, current hybrid and highly integrated devices enable to merge on a single platform ad-hoc reconfigurable fabrics, high-performance multiple processors, digital and analog components as well as dedicated accelerators.
One of the most used architectural solutions to be tailored on the state-of-the-art commercial devices for the deployment of computationally intensive tasks requiring in-field upgrades for adaptability, acceleration, or dependability purposes is the Dynamically Reconfigurable Processing Module (DRPM).
These platforms based on commercial SRAM-based FPGAs have proven to be especially suitable for the deployment of high-performance and mission-critical domains, ranging from massive computation for large-scale communication and data centers to high-reliability electronics deployed in radiation environments, like the aerospace and high energy physics experiments (HEP).
In fact, these applications can gain tremendous advantages from the in-field optimizations to adapt their functionality and effort according to the instantaneous needs as well from the self-testing and self-repairing capabilities enabled by reconfiguration.
Despite these advantages, to maximize performance gain achievable through runtime reconfiguration the time required to perform such procedure needs to be considered and properly managed.
As one of the main aspects to be tackled to improve high-performance reconfigurable applications consists in the optimization of the time required to perform in-field reconfiguration, it has been the core of the research presented in the *first part* of the dissertation.
In fact, commercial devices still require a relatively long time to access both in reading and writing the configuration memory. As it will be detailed in Chapter 4, the current commercial devices present a complex link between the reconfigurable fabric organization and the configuration memory structure, and this reconfiguration overhead is strongly dependent on the amount of data involved in the reconfiguration and in the mechanism used to transfer these data among the two layers.
In this view, the Frame-driven Routing Algorithm (FeDRA) presented in Chapter 5 has been developed as a generalized approach for the minimization of the configuration data to address these limitations in commercial devices thanks to the awareness of their configuration settings .
Furthermore, the Reconfigurable Multipotent (ReM) Cell, presented in Chapter 6 has been designed as the basic functional unit for novel fine-grained architectures oriented to fast, detailed, and concurrent self-reconfiguration by rethinking the configuration granularity and mechanism.
On the other hand, when reconfigurable SRAM-based FPGAs are deployed in

mission-critical applications, like the aerospace and HEP domains that operate in presence of radiation, one of the main aspects which demand to be addressed consists of the SRAM configuration memory sensitivity to radiation-induced errors. As to guarantee the readiness to the mission and radiation environment to avoid failures during operations the application dependability must be carefully characterized before deployment, the core of the research presented in the second part of the dissertation has been centered on the dependability analyses of self-reconfigurable applications.
In the detail, the main focus of the analysis has been the controller managing the in-field configuration memory access in dynamically and partially reconfigurable applications, and that, as discussed in Chapter 7, represents the key component for enabling the successful implementation of run-time optimizations, either oriented to performances or self-recovery and self-testing.
Therefore, as presented in Chapter 8, an in-depth soft-error analysis has been performed on different self-reconfiguration controllers reliability versus different radiation environments and mission goals enabling the instrumentation of a cost-effective setup for fast and efficient SRAM-based FPGA radiation testing that uses as core component the controller identified as the most reliable.

Part I

Dynamic Reconfigurable Architecture Performance

Chapter 3

Related Works

This chapter provides the related works to contextualize the dissertation contributions on reconfigurable system performance through the optimization of the reconfiguration time and data overhead.
In detail, several approaches aimed at the optimization of dynamically and partially reconfigurable applications within the FPGA development flow are presented to introduce Frame-driven Routing Algorithm (FeDRA), while an overview on architectural solutions proposed to optimize the trade-offs among granularity, heterogeneity, and configurability are presented to introduce Reconfigurable Multipotent (ReM) Cell.

3.1 Overview

The FPGA development flow is the process through which the target circuit is translated from its Hardware Description Language (HDL) model to its final format for the deployment on the target device.
This process consists of several intermediate steps involving complex and computationally intensive operations typically performed by Computer-Aided Design (CAD) tools and its efficiency is fundamental for the successful, optimal, and safe implementation of the target system determining its real feasibility.
For these reasons, the research and the realization of efficient algorithms and approaches for its enhancement have been and still are crucial for the evolution and optimization of FPGA-based applications, as it will be discussed in Section 3.2.1.
When dynamic and partial reconfiguration is implemented this process becomes more challenging due to the additional complexity, requirements, and constraints. On the other hand, as it will be discussed in Section 3.2.2, it also enables the integration of reconfiguration-driven or oriented optimizations after the design stage and thus independently from the target application.

Therefore, many works have made use of the development flow to enhance the flexibility, reliability, and performances of reconfigurable applications, either in the form of stand-alone tools or additional assets integrated within the vendor toolchain.
As it will be discussed, this latter approach has been the one leveraged by the Frame-driven Routing Algorithm (FeDRA), with the specif aim of reducing the reconfiguration data and time in commercial FPGAs within the circuits routing stage without affecting other performance parameters.
However, as the utilization domains and computational requirements of reconfigurable systems have been rapidly broadened demanding increased flexibility and performances, many efforts have been done in the exploration of novel solutions to provide alternative architectures capable to overcame specific limitations of commercially available devices in optimally tailoring novel and specific applications.
In fact, as commercially available devices still require relatively long reconfiguration time, the design of novel architectures has been focused on the exploration of efficient trade-offs between granularity, performances, and complexity as well as on different models and approaches for the efficient integration of reconfigurable processing elements with hardwired components with special attention on the optimization of the reconfiguration capabilities in terms of time and data involved in the procedure.
As it will be discussed in Section 3.3.1, obtaining an optimal trade-off between granularity and area to provide a high level of flexibility without losing performances for specific applications represents a complex problem which solution should carefully consider the proportion among the programmed design and the circuitry required for its configuration as well as the complexity of the underlying mechanism.
These have been the key considerations for the development of the ReM Cell, which has been designed as the basic reconfigurable element for novel and distributed reconfigurable fabrics able to efficiently support those applications requiring frequent and detailed fine-grained reconfiguration by relying on an ad-hoc configuration mechanism that minimizes the time and data required to update circuits functionality.

3.2 Development Flow Optimizations

The FPGA development flow translates through several intermediate steps the behavioral description of a design into configuration data to be written into the configuration memory for its deployment.
When FPGA design involves Dynamic Partial Reconfiguration, in addition to the challenges and the optimizations embedded in the standard development flow, additional constraints need to be taken into account to preserve signal integrity and floorplanning requirements.
On the other side, the development flow itself can be improved to implement and

generalize approaches oriented to the optimization of in-field upgradable circuits which are not efficient or appliable at the design stage or which are possible only thanks to dynamic reconfiguration.
Therefore, in the following, preliminaries on the relevance and challenges of development flow in standard FPGA design are provided, with a major focus on the routing stage and algorithms. Subsequently, remarkable approaches oriented to increase dynamically and partially reconfigurable applications flexibility, safety, and performance either as stand-alone frameworks or techniques to be applied at run-time are discussed.

3.2.1 Standard FPGA Development Flow

The FPGA development process is generally performed within the vendors' toolchain and, as summarized in Figure 3.1, it consists of the following stages: Synthesis, Placement, Routing, and Bitstream Generation.
These steps are conceptually and namely similar to the ones required for the ASIC implementation but within the FPGA development they acquire different meanings and restrictions.
Firstly, through the Synthesis, the gate-level representation of the circuit is obtained from the HDL, where the gates consist of the functional blocks and primitives embedded in the FPGA fabric, such as LUTs, FFs, IO Pins, BRAMs, and DSPs.
Subsequently, the blocks and primitives identified during the Synthesis are bounded to the hardware resources physically available on the target part and the programmable routing paths to connect them are selected defining the circuit routing.
These two phases, respectively called Placement and Routing, are strictly related since the length and amount of the programmable connections strongly depend on the location of the logic nodes, and together define the physical resources to be used and programmed on the target FPGA for the implementation of a given design.

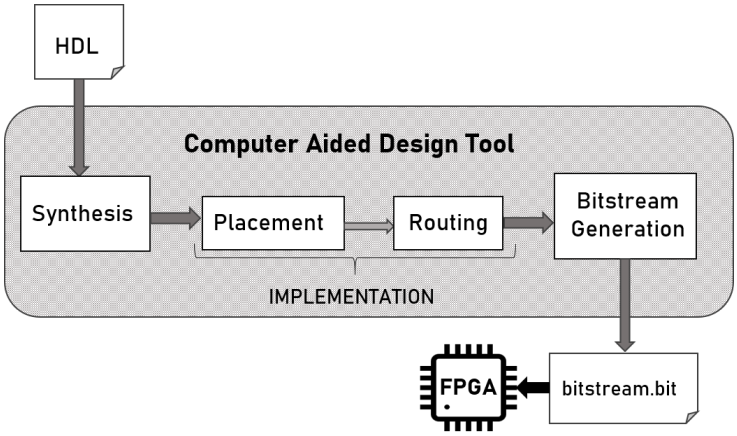

Figure 3.1: FPGA Development Flow

For these reasons, they are often referred to as a unique phase called Place&Route (P&R) or Implementation, which in general is the most crucial and computationally intensive phase within the design development. In fact, differently from the freedom provided by ASIC layouts, the count and location of the resources physically present into target FPGA parts are fixed, and thus how they are used according to the overall availability and positions imposes bounds on the final design implementation and feasibility.
Therefore, as the quality of P&R solutions strongly influences the system performances, FPGA Placement and Routing algorithms have been a historical and crucial research field for providing efficient trade-offs among area, delay, and power consumption despite the lower degree of freedom FPGA offers.
Finally, once the Implementation of the circuit is obtained, through the Bitstream Generation this representation is coded in the configuration data to be loaded inside the configuration memory for the application deployment on the part.

FPGA Routing Algorithms

As mentioned the routing phase has a crucial role in the final circuit performance and its objective is to find a feasible routing solution to connect all the placed nodes of a circuit. In general, the routing algorithms used in this phase can be classified according to the algorithmic approach used to solve the problem and the policy on which they prioritize performance parameters.
Several algorithmic approaches have been proposed adapting older ASIC routers to the FPGA scenario modeling the routing problem as an oriented graph, such as the most renowned Maze Router, A* and Pathfinder geometric algorithms [78][79]. In this context, one of the most relevant has been the PathFinder algorithm which exploits an iterative approach based on negotiation to provide a routing solution with an optimal trade-off between routability and performance by giving higher priority to critical paths [80].
Many commercial and non-commercial routers have been based on the PathFinder approach, as the VPR (Versatile Place and Route) and its extensions which rely on a timing-driven adaption of it and have represented valuable frameworks for the development of other tools and routing algorithms for FPGA architectures [81].
As the complexity and density of FPGAs devices increased, different algorithms have been proposed relying on parallel and concurrent approaches by translating the routing problem into Boolean equations, as the Boolean-based Algorithms which enable the simultaneous evaluation of multiple routing nets. To pursue this goal several approaches have been suggested, as Binary Decision Diagrams (BDDs), satisfiability (SAT) solvers and graph-based hybrid solutions [82][83].
Besides the algorithmic approach used to successfully find a routing solution for the circuit, routing algorithms can embed additional optimization policies that are typically oriented to the minimization of circuit delay and routing congestion, like the

one performed by TRACER_fpga [84] and CeRA [85] algorithms, or dependability enhancement, like the Reliability-oriented Place & Route Algorithm (RoRA) [86].
In detail, TRACER_fpga strongly reduces the circuit delay by focusing on the reduction of nets length but, as drawbacks, the algorithm execution requires long times and does not consider congestion during its optimization, while CeRA algorithm focuses on routability aiming at optimizing routing congestion while maintaining its execution time short. The Reliability-oriented Place & Route Algorithm (RoRA) instead focuses on dependability and routes nets avoiding the usage of resources with high criticality if contemporarily activated. It works in combination with an ad-hoc placer performing redundant placement increasing FPGA designs robustness to soft-errors at the cost of a slight performance reduction.
In general, although more than one routing policy can be pursued at the same time, obtaining efficient trade-offs is challenging since, besides the used approaches and cost functions, all routing algorithms imply very complex and intensive computations.
Additionally, the efficiency of a routing solution is strongly related to the quality of the placement, which imposes the starting conditions for the router and has the greatest impact on the overall design performances and feasibility.

3.2.2 Dynamic Reconfiguration Scenario

Dynamic and partial reconfiguration introduces in the design development process additional requirements and constraints to optimize area efficiency, preserve signal integrity, and satisfy floorplanning requirements.
In fact, for module-based reconfiguration the regions of the design statically programmed on the FPGA and the ones subject to run-time reconfiguration must be declared at the beginning of the development procedure as well as their placement since typically the dynamic modules are processed within separated and parallel implementation flows to produce the partial configuration files.
On the other hand, the dynamic nature of reconfigurable FPGAs enables the possibility to move or perform parts of Place &Route execution online to increase reconfigurable applications flexibility, as proposed in works discussed in the following and specifically oriented to the optimization of the algorithms complexity and performance for their execution at run-time [87][88][89].
In detail, JITPR toolset [87] focuses on the fast placement and routing for in-field implementation of reconfigurable modules. It is based on a Just-In-Time (JIT) placer able to rapidly estimate the most suitable regions to deploy target circuitry assigning them through a Simulated Annealing (SA) approach and looking ahead to the routing stage, following the VPR tool strategy. The JIT router relies on a modified version of the VPR PathFinder expressly instrumented to achieve better execution times at the cost of a slightly reduced routability. Although JITPR highly outperforms VPR execution times while maintaining or even increasing the

target circuit performance, the time required for its execution is in the order of seconds, and thus still critical for its deployment at run-time.
In [88] instead, a P&R approach aiming at obtaining even faster execution times through the minimization of the algorithm complexity is proposed. Moving forward popular SA, the proposed approach exploits a single-try placement and locality for the routing through graph traversals, demonstrating the possibility to achieve execution times in the order of milliseconds and thus more compliant with the run-time deployment. Although this approach requires algorithmic execution times that strongly outperform both VPR and JITPR, it is based on an FPGA model which does not include more complex primitives embedded in modern devices, such as BRAM and DSPs.
Finally, in [89] the author has proposed as a further contribution on dynamically and partially reconfigurable architecture performances and dependability an approach for the in-field routing of nets in the DRPM context. In detail, the approach consists of a self-rerouting algorithm oriented to fault tolerance and performance in-field optimizations to be deployed as a dedicated IP core statically programmed on the FPGA, introducing low area overhead within the DRPM. The core implements a simplified hardware version of the PathFinder routing algorithm able to route target nets at run-time without the needs of external hosts and without degrading the resources usage and delay of the original nets.
However, one of the main advantages provided by Placement & Routing algorithms consist of enabling the introduction within the development flow stages of additional optimization techniques oriented to increase reconfigurable applications flexibility, reliability, and performances downstream from the design phase, and thus independently from the target functionality to be deployed in the reconfigurable fabric.
Therefore, several approaches in this context have been proposed to enhance reconfigurable applications through the development flow either in the form of tools able to interact with the vendor or stand-alone frameworks, as the ones discussed in the following [62][90][26].
The approach suggested in [62] focuses on the optimization of the placement of the reconfigurable modules and the efficiency of the interconnection network among these modules. In detail, the flexibility and reliability of the in-field reconfigurable module placement and connectivity are improved starting from the synthesis phase. The run-time placement efficiency is increased and floorplan and connectivity violations are avoided thanks to a dedicated approach for the evaluation of the possible placement positions and intersections through the introduction of a novel concept of sub-regions and an ad-hoc flexible communication macro supporting the proposed reconfigurable region framework.
The Dreams tool proposed in [90] instead enables similar optimizations starting from post-implementation netlists of reconfigurable modules and elaborating them to introduce safe interfaces to manage routing conflicts using an ad-hoc router and

a novel communication scheme, finally producing the corresponding partial configuration files with enhanced portability and flexibility for the module relocation.
The Place and Route Tools set called TPaR presented in [26] instead aims at minimizing the FPGA resources involved in the run-time reconfiguration and thus the time required to perform the procedure relying on the concept of Dynamic Circuit Specialization (DCS). The proposed TPaR allows a preliminary circuit elaboration performed offline that produces intermediate design representations which maximize the LUTs and routing resource reusage across different configurations and thus enabling their fast online processing to minimize the configuration settings to be updated at run-time for changing the circuit functionality.

3.2.3 Discussion

As discussed, the quality of P&R solutions has a strong impact on the system performances and, for this reason, the FPGA Placement and Routing algorithms have a key role in the FPGA design development representing valuable assets for the optimization of both standard and run-time reconfigurable circuits.
Although many techniques, tools, and algorithms have been studied and proposed to enhance different aspects of dynamically and partially reconfigurable circuits, many of them rely on FPGA models which highly differ from the state-of-art commercial architectures.
Furthermore, to the best of the author's knowledge, the TPaR toolset presented in [26] represents the only approach that considers as optimization goal within the development flow the minimization of the reconfiguration time and data, which is a crucial parameter to enhance the overall system performances.
Although the TPaR framework has demonstrated a strong reduction of the reconfigurable resources involved in the in-field upgrades, minimizing area and reconfiguration time, it imposes strong placement constraints and does not consider in its elaboration complex primitives and hard macros embedded in the current commercial FPGA architecture, as BRAMs and DSPs. This can imply area and performance losses as the size and complexity of the target circuit increases making it difficult or even impossible to efficiently play the proposed optimizations.
Additionally, when targeting commercial architectures and considering the complexity and the impact of the placement algorithm on circuits performance, not taking advantage of the optimal placement solution available through the vendor placer could result in a loss in the overall application performance in terms of delay, congestion, and power consumption that can jeopardize the achievements obtained with the proposed reconfiguration optimization.
Contrarily, as it will be further discussed in Chapter 5, the main contribution of the optimization technique implemented in the proposed **FeDRA** routing algorithm consists of its ability to reroute circuits to strongly reduce the time required

for their configuration on Xilinx FPGA without interfering with the rest of standard development and thus with the circuit performance.
In fact, to achieve the minimization of the reconfiguration time without paying penalties in other fundamental circuits performance parameters, FeDRA has been instrumented to start from the original placement solution obtained through the vendor placer and to perform its optimization only on the routing resources, which typically are the 70% of FPGA implementations. In this way, the proposed technique does not interfere with the mapping and placement of computing nodes and hard-macros, such as BRAM and DPS, already optimized by the vendor tool also in the vision of the next routing stage. In detail, this has been possible by integrating into FeDRA a routing policy that exploits the awareness of the link among programmable resources and memory layers in commercial devices to routes circuits minimizing the configuration data overhead introduced by routing segments by weighting them according to their configuration settings.

3.3 Architectural Approaches

The growing computational demand of modern applications towards heterogeneous and highly integrated reconfigurable systems has driven the exploration of different architectural models and frameworks to maximize the efficiency of reconfigurable architecture according to the target application.
In fact, as it will be further discussed in Chapter 4, for specific applications requiring frequent and detailed in-field reconfiguration the current commercial FPGA architecture could result penalizing. This is due to the complex link among the single resources on the programmable layer and the configuration memory organization as well as to the long times required to access the configuration memory through the current reconfiguration mechanism which could result inefficient and resource consuming for the update of specific resources or small components functionality.
Therefore, as discussed in the following, alternative architectures have been proposed to explore efficient trade-offs between the basic reconfigurable element granularity and complexity, considering novel mechanisms and approaches for the efficient integration of reconfigurable processing elements with hardwired components to optimize the reconfiguration capabilities in terms of time and data involved in the procedure.

3.3.1 Reconfigurable Fabric Granularity and Heterogeneity

As anticipated in Section 1.1.2, reconfigurable architectures can be classified as Fine-grained or Coarse-grained according to the data size of the elements their reconfigurable unit they can elaborate.

In general, functional units with a finer granularity are especially valuable for custom bit-level computation but they pay the prices in terms of area and complexity. On the other side, as the functional unit granularity increases, the execution of more sophisticated and modular software-like applications results more efficient, as the basic functional elements are closer to standard processing units, but pays the price of reduced flexibility [7][11].
Although the fine-grained FPGA architecture has eventually emerged as the golden commercial solution for reconfigurable computing thanks to the efficient balance it provides among complexity and performances, many other architectural models have been proposed with finer or variable granularity to increase the fabric flexibility for tailoring specif computations.
The finest granularity can be achieved on reconfigurable architectures where the programmability reaches the transistors level, enabling the possibility of tuning online their characteristics or combining them to implement both analog and digital components at run-time.
These architectures take the name of Field Programmable Transistor Arrays (FPTAs) and among the most remarkable there are the ones presented in [12] and in [13].
In detail, in [12] a basic functional unit enabling the in-field self-adaptation of its transistors configuration for evolvable hardware computations is proposed. It is composed of 4 NMOS and 4 PMOS transistors which connectivity is arranged through programmable switches that enable the run-time configuration of both analog and digital circuitry, from current mirrors and differential pairs at the finest granularity to logic gates and operational amplifiers at a higher abstraction level.
The more recent PAnDA (Programmable Analog and Digital Array) architecture proposed in [13] instead enables reconfiguration at multiple layers, resembling the FPGA fabric at the highest abstraction level while providing the possibility to adapt at run-time low-level fabric parameters, such as the transistor characteristics and the analog behavior of the basic computational unit to address process variability and faults effects as well the optimization of circuits performance for speed and power consumption improvements.
Although the increased benefits the FPTA concept provides in terms of flexibility and in-field low-level adaptability, the overhead introduced by such in-depth reconfigurability in terms of area and complexity becomes extremely high.
Therefore, other intermediate solutions between fine-grained and coarse-grained reconfigurable systems have been explored, as the Variable Grain Logic Cell (VGLC) architecture proposed in [91] and the Re-Configurable Mixed-grain (ReCoM) architecture presented in [92] and [93].
In detail, the VGLC presented in [91] consists of a basic functional element able to adapt its granularity according to the requirements of the target application. In detail, applications focused on bit-level computations are implemented on fine-grained

cells while more complex arithmetic calculations are tailored on the available functional units with a higher granularity. In fact, the proposed cell includes both coarse-grained and fine-grained configurable logic and has been designed to act either as a dedicated core to be integrated with heterogeneous systems or for the implementation of basic FPGA functional units strongly reducing the configuration data with the respect to commercial devices.
The more complete ReCoM architecture presentd in [92] and [93] instead consists of RISC host processor coupled with an array of mixed-grained reconfigurable units that provide support for high performance and complex operations allowing both word-level and bit-level computations according to the needs and maximizing scalability and parallelism. The proposed architecture allows the fast and transparent in-field adaptation to different tasks by providing the possibility to perform configuration upgrades on reconfigurable fabric from both the host reconfiguration control unit and the reconfigurable cell array itself allowing their concurrent access to the context memory within a single clock cycle.
Although both the VGLC and ReCoM pay a price in terms of area overhead, they provide enhanced flexibility and configuration capabilities. Furthermore, the former highlights the benefits of having functional units that could be used either for dedicated cores in heterogeneous systems or for building regular Island-style fabrics while minimizing the amount of configuration data, while the latter introduces the possibility for the reconfigurable cells array to be configured not only from the on-chip host processor but also to manage its own reconfiguration to speed-up the procedure.
These features highly suite the trends towards the increased heterogeneity to maximize computational capabilities while maintaining high flexibility through the integration on the same platform of high-performance hardwired processing units, custom ICs, and dedicated accelerators based on reconfigurable fabrics.
As anticipated in Section 2.1.2, this trend led to the concept of Embedded FPGAs (eFPGAs), which key feature consists of using only the precise amount of reconfigurable resources and primitives required for the target platform coupling the benefits of ASIC customization for control and application-specific computations with the advantages of in-field upgradability for flexible data-paths and accelerators.
Therefore, several works have addressed the optimization of eFPGA cells and their integration within computing platforms to widen their performances and applicability range, as the ones presented in [94] and [95].
In detail, in [94] different models of integration and coupling between the reconfigurable fabrics and the hardwired processing system are discussed leading to the realization a custom eFPGA reconfigurable tile, consisting of clusters of basic units with specific logic and sequential capabilities interconnected through dedicated internal and global routing infrastructures and able to provide enhanced flexibility, area efficiency, and power consumption for the execution of the target arithmetic

computations.
The framework presented in [95] instead is focused on supporting the realization and validation of specialized eFPGA-based architectures relying on a high-level template. The proposed flow allows the customization of the eFPGA configuration layer organization to maximize the sharing of configuration bits among contiguous tiles and enabling the optimization of both configuration data and resource usage with a consequent reduction of configuration time and memory cells overhead.

3.3.2 Discussion

As discussed, the exploration of architectural models providing optimal trade-offs on reconfigurable unit granularity and the efficient integration with standard processors and other components has been addressed by many to maximize the benefits achievable through reconfigurable fabrics for different applications while optimizing configuration time and data overhead over commercial devices.
In fact, commercially available FPGA architectures currently are not optimized for tailoring those applications requiring frequent and detailed updates of the reconfigurable fabric as the time and the amount of data required for such modifications would be unjustified.
However, the identification of efficient architectural trade-offs between performances, granularity, and area to achieve optimal flexibility for specific applications represents a complex problem.
Deeply fine-grained reconfigurable fabrics as the ones proposed in [12] and in [13] provide the maximum degree of flexibility and configurability but, in addition to imply a resources explosion, strongly increase the complexity of fabric as well as the flow to tailor on them the target functionality.
Architectures aimed at finding a more efficient trade-off between reconfigurable units granularity and complexity, like the ones proposed in [91] and [92], reduce the complexity providing to the user additional flexibility for tailoring target applications, enabling different integration approaches to couple the reconfigurable fabric with the hardwired components, and introducing the possibility of reconfigurable array self-reconfiguration with the additional goal of optimizing reconfiguration time and data.
Similar optimizations have been further explored moving towards the concept of eFPGAs, as proposed in [95] and [94], enabling the optimization of the reconfigurable cells configuration mechanism and their custom integration within computing platforms.
However, this enhanced flexibility still introduces an area overhead that is not negligible and the proportion among the programmed design and the circuitry required for its configuration as well as the complexity of the underlying mechanism represent fundamental aspects to be considered in the design of these architectures.
The evaluation of the current commercial architecture limitations and the different

architectural trade-offs between granularity and configurability led to the design of the **ReM Cell**, which has been proposed as the basic reconfigurable element for novel and distributed reconfigurable architectures addressing these aspects.
As it will be further discussed in Chapter 6, the main considerations driving the design of the ReM cell have been oriented to implement reconfigurability in a distributed manner and at the bit-level granularity maximizing flexibility and modularity. This has been possible by identifying a granularity level fine enough to support the elementary computational block and by providing a local configuration engine to each cell to perform self-reconfiguration as well as trigger the reconfiguration of the surrounding units.
In detail, the ReM Cell has been devised to behave as Logic, Memory or Connectivity element, using a minimal amount of configuration data and with additional reconfiguration capabilities, enabling concurrent and fine-grained in-fields upgrades within a single clock cycle.
Furthermore, thanks to their high modularity, ReM Cells enables their detailed and bit-level reconfigurations to be triggered both internally from the cell array and externally through host processing units when embedded in more complex heterogeneous eFPGA-oriented platforms.

Chapter 4

Technical Background

In this chapter, the technical background to support the presentation of FeDRA and ReM goals and achievements is provided.
In detail, the current commercial SRAM-based FPGA architecture is described, using as reference the Xilinx 7 Family, with a major focus on its reconfigurable fabric organization and configuration memory structure and discussing the complex link between them.
Subsequently, the typical reconfiguration techniques and interfaces are discussed as well as the reconfiguration time bottlenecks embedded in the current architecture.

4.1 SRAM-based FPGAs Architecture

As mentioned, the current commercial architecture of SRAM-based FPGAs is a heterogeneous platform integrating programmable resources that interleaves with hardwired components and which functionality is controlled by the content of the SRAM configuration memory.
To detail the architecture of these platforms and highlight their the main features and limitations, the Xilinx 7 Series FPGAs have been taken as a reference.
In fact, the 7 Series is one of the most recent Xilinx FPGA families and includes the Spartan-7, Artix-7, Kintex-7, and Virtex-7 devices which rely on the same internal reconfigurable fabric based on 28 nm technology and which mainly differs from the resource count, performance and packaging [19].
As mentioned, the SoC Zynq family relies on the same internal architecture while including an on-chip hardwired processing system, based on ARM processors [22].
The 7 Series FPGA architecture relies on the classic two-layers fabric where the functionality of programmable resources is defined by the data stored in the configuration memory.

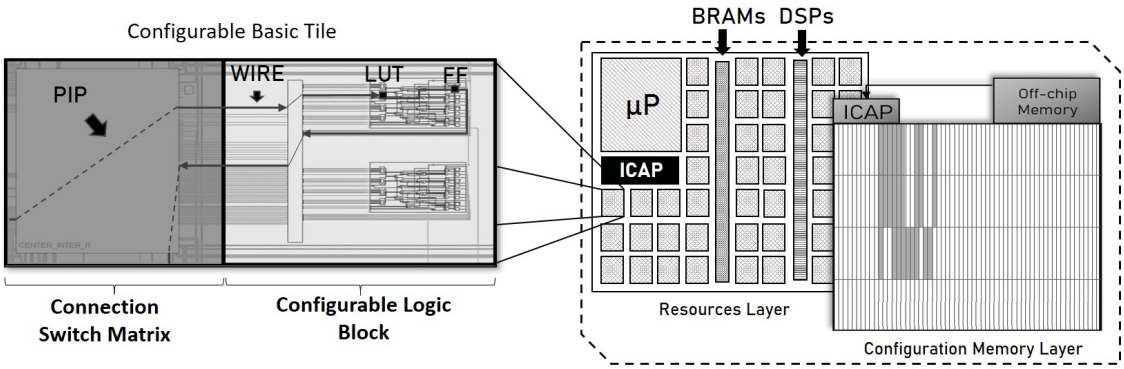

Figure 4.1: 7 Series Basic Tile and Heterogeneous Architecture

As summarized in Figure 4.1, the resource layer consists of a regular array of Tiles, containing both programmable logic within the Control Logic Block (CLB) and Programmable Interconnect Points (PIPs) within the Switch Matrix (SM), and interleaving with hardwired primitive for efficiently tailoring user memories and complex arithmetic operations, such as RAM Blocks (BRAM) and Digital Signal Processing (DSPs) Units.
Furthermore, additional hardwired macros are present on 7 Series devices for implementation of specialized purposes, such as in-field reconfiguration (ICAP), analog interfacing and monitoring (XADC), clock management (CMT and PLLs), and high-throughput connectivity, which will be detailed in following together with the reconfigurable Tile resources and configuration memory layer organization description [19].

4.1.1 Reconfigurable Resource Layer

The Reconfigurable Resource Layer consists of a regular and periodic array of Tiles, which are connected among them and with the other hard macros and primitives through fixed wires composed of horizontal and vertical lines of different lengths.
Each Tile has a functional section called CLB and containing the programmable logic and sequential elements in the form of LUTs and FFs for the implementation of computing nodes of the design, and a connection section called SM and containing the Programmable Interconnect Points which activation enables the connectivity with the fixed wires. The fabric array is divided into Clock Regions, which are sub-portions of the FPGA that are connected to a dedicated Global Clock Line (GCL) for the efficient management of the clock distribution and domains.

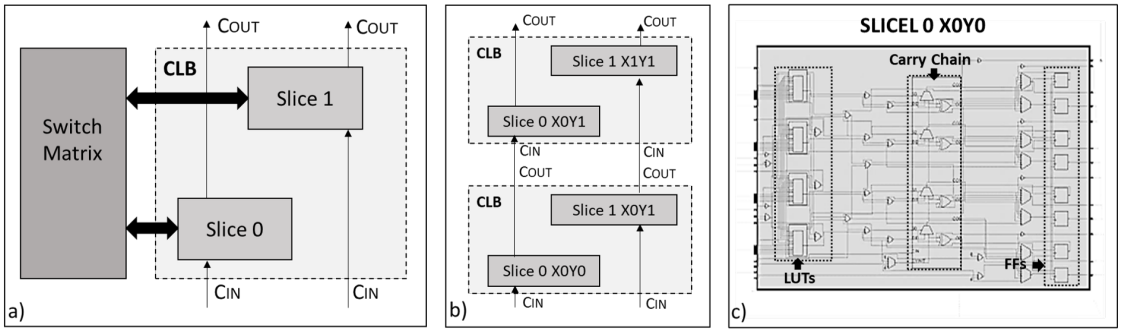

Figure 4.2: 7 Series CLB: a) Slice Arrangement and SM Connectivity; b) Column Connectivity between CLB Slices; c) SLICEL Schematic from Vivado View

Control Logic Block

The CLB is the Tile section devoted to the implementation of the combinational and sequential nodes of the circuit and includes two Slices, which contains LUTs, FFs, and carry-chains, and has its own access to the SM for the connectivity with the other CLB and primitives, as summarized in Figure 4.2a [96]. In fact, the direct connectivity among Slices of different CLBs generally is not present, except for the available independent carry-chains within the same column (Fig. 4.2b).
Each Slice contains eight FFs and four 6-input LUTs, which can be either used as single 6 to 1 LUT or as two separated 5-input LUTs with shared address and logic inputs but one dedicated output that could be optionally registered, and additional multiplexers and dedicated carry logic for the configuration of various arithmetic functionalities, as reported in Fig. 4.2c where a sample Slice of type SLICEL is shown. In fact, in 7 Series FPGAs two kinds of Slice exist, the aforementioned SLICEL which represents two-thirds of the total, and the SLICEM, with additional circuitry for the efficient implementation of distributed RAM and Shift Registers.

Switch Matrix & Programmable Routing

The Switch Matrix is the Tile section devoted to the implementation of the connectivity among CLBs and other FPGA primitives, as DSP, BRAM, and IO pins.
The SM contains the PIPs, which are programmable routing segments acting as configurable switches between SM inputs and SM outputs, also called Junctions.
In detail, each SM output Junction links programmable routing segments to the FPGA fixed wiring resources, which according to their length, direction, and type reach the input Junctions of other SMs or primitives within the fabric. Thus, the connection among two functional nodes is made by the alternation of PIPs and wires, which takes the name of Net, and the final routing of the circuit consists of the collection of such Nets.

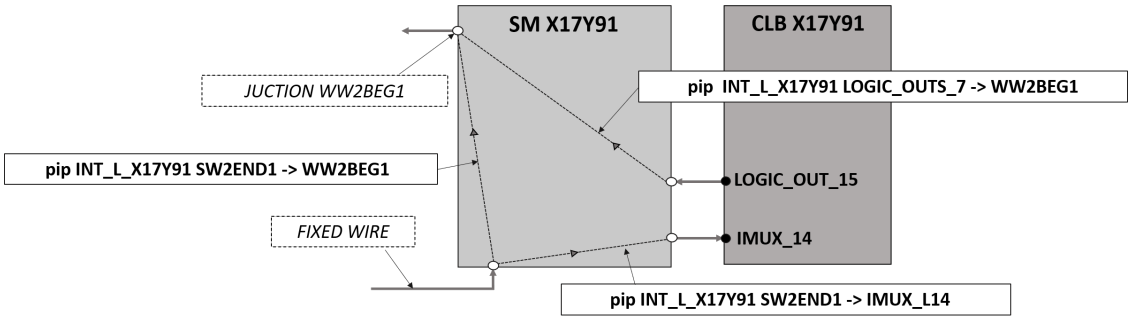

Figure 4.3: Switch Matrix Organization: PIPs, Junctions, and Wires Function and Taxonomy Examples

The Net composition and PIPs organization and classification can be obtained through the study of their Xilinx Design Language (XDL) format [97]. In detail, each PIP segment is named upon the coordinates of the SM it belongs on the tile array coupled with the information about the direction and length of the wiring segments or primitives it put in communication.
Thus, if a routing resource belonging to the SM of coordinates X=17 and Y=91 connects the end (END) of a wire coming from a Tile at South-West (SW) with the beginning (BEG) of a wire going West (WW), it would be identified as:

pip INT_L_X17Y91 SW2END1 -> WW2BEG1

where INT_ L_X17Y91 states the PIP is located in the SM of the Tile in such coordinates while the strings at the sides of the arrow state the Junctions it connects giving information about the direction and the distance covered by the wires.
If a PIP is instead the last segment of a Net, and thus it is connected to the input of a CLB (called IMUX), it will be defined as:

pip INT_L_X17Y91 SW2END1 -> IMUX_L14

meaning that it is connecting to the 14th input mux of the CLB on the same Tile. Instead, in the case in which the PIP is the first segment and thus it directly takes the 7th output of the CLB (LOGIC_OUTS), the nomenclature will be:

pip INT_L_X17Y91 LOGIC_OUTS_7 -> WW2BEG1

Due to the regularity and periodicity of FPGA resources within the fabric, this formal notation allows the classification and characterization of routing resources position and functionality, as summarized in Figure 4.3 reporting the aforementioned examples.

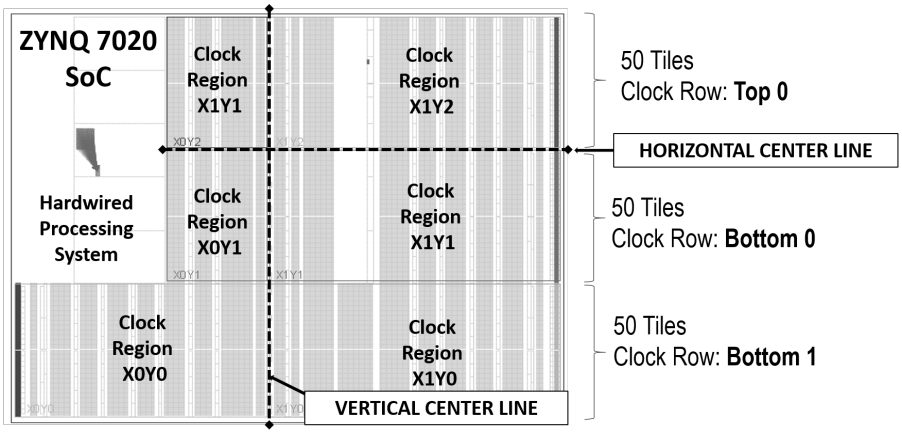

Figure 4.4: ZYNQ 7020 SoC Clock Regions Organization

Clock Regions & Distribution

With respect to older devices, in the 7 Series, the Clock distribution and management is enchanted by the introduction of the Clock Regions. Thus, the implementation of complex clocking configuration is optimized through the usage of both global and regional clock resources by enabling each Clock Region to support up to 12 different clock domains and the possibility to combine several global clock buffers for their distribution [98].
In detail, each Clock Region has a fixed height spanning in vertical across 50 CLBs and covering horizontally approximately half of the device. As summarized in the example of Figure 4.4 reporting the Clock Regions organization in ZYNQ 7020 device, the Vertical Center Line divides the FPGA into two adjacent Clock Regions while the Horizontal Center Line divides the device into a Bottom and a Top half sections, each one of them including a number of clock regions which depends to the fabric size.

4.1.2 Configuration Memory Layer

As the resources layer has a periodic and regular organization, the configuration memory presents its own regularity and periodicity, although the mapping among the two layers is not straightforward.
In fact, the configuration data are organized in frames, consisting of very long bit words representing the smallest atomic memory segments used for reconfiguration while partially configuring different elements on the same resource layer column.
In detail, for the Xilinx 7 Series, each frame consists of a 3,232-bit word that covers in vertical the height of a single Clock Region and contains data partially configuring the resources within a column of 50 Tiles, being composed by bits that incompletely control resources possibly distant and unrelated within the physical

reconfigurable layer.
As it will be further discussed in Section 4.2, this results in a complex encoding between configuration data and programmable resources which interpretation becomes even more difficult due to the lack of information provided from the vendors about this link [99].
The content of the configuration memory layer is obtained at the end of the development process as a binary file, called bitstream, to be loaded inside the SRAM memory for the application deployment which encodes the description of the target circuit according to the memory cells state required for its implementation.
In fact, bitstream files are automatically produced by the Xilinx tool and, although different formats and coding are available according to the target purpose, delivery mechanism, and target storage, the most common is *.bit* format, consisting of three main sections: the *Header*, the *Configuration Data* and the *Tail*.
In detail, the *Header* embeds data and settings that are processed during the Initialization phase for the setup of the target design, such as information for the bus width detection, the synchronization word, the target device ID, the CRC instrumentation, and the Configuration Frames words count stating the amount of data contained in the following *Configuration Data* section, which consist of the frames for the design implementation. Finally, the *Tail* closes the bitstream and contains the data for the CRC check and the Startup Sequence, which confirm that all the configuration bits have been successfully received and the deployed system is ready for operation [23].

4.1.3 Hardwired Components

As mentioned the reconfigurable Tiles interleave with hardwired components and primitives, which can be used for different purposes, such as the hardwired Processing System (PS), Digital Signal Processing Units (DSP), RAM Blocks, (BRAM), and Configuration Access Port macros such as the Internal Configuration Access Port (ICAP).
In detail, DSP and BRAM hardmacros are fundamental for the implementation of such functionalities which could require an unjustifiedly high resource usage or provide lower performance if tailored on the standard reconfigurable logic.
The DPS units embedded in the 7 Series provide efficient support for the implementation of enhanced arithmetic units involving massive additions and multiplications as well as for complex circuitry, such as bus multiplexer and wide comparators, with the added possibility to be time-multiplexed or cascaded to increase the computational capabilities [20].
The BRAM macros instead are devoted to the deployment of large user dual or single-port RAMs, ROM, and FIFOs modules, as their size increases and their implementation on distributed RAM (i.e., based on SLICEM programmable resources) results no longer efficient. For this purpose, different primitives are available that

present a memory capacity of up to 36 Kb and can be easily cascated to increase the size as well as being used as independent components when more elements with less dept are required. Additionally, BRAM primitives include a built-in Hamming error correction code (ECC), which can be transparently activated for the in-field single errors correction and double errors detection [21].
Furthermore, with the Zynq 7000 SoC Family, Xilinx integrated into the 7 Series programmable fabric single or dual-core ARM processors with enhanced connectivity with the on-board memory and peripherals. This enables the implementation of a complete and efficient processing system (PS) that allows the integration of software approaches for computation, power, and configuration optimizations by supporting the deployment of both simple software routines and high-level operating systems within the application [22].
Finally, although the Zynq SoC Family introduced the additional Processor Configuration Access Port (PCAP) for accessing configuration data directly from the PS, all the 7 Series devices embed the ICAP, which acts as an interface for the exchange of data among the reconfigurable application and configuration memory layers.
In fact, the ICAP allows managing partial reconfiguration directly from the reconfigurable fabric relying either on vendor or custom configuration controllers, which are in charge of receiving both software and hardware reconfiguration triggers for loading or downloading configuration data through the ICAP while managing handshaking and decoupling procedures [47].

4.2 Resource and Configuration Memory Link

In general, the awareness of the link among the reconfigurable resource layer and configuration memory represents a key asset in the design of reconfigurable FPGA applications for the implementation of low-level techniques oriented to both dependability and performability.
In fact, knowing the bitstream encoding and its link with the programmable logic enables to acquire information about the in-depth fabric organization and to perform accordingly specific optimizations and investigations, as detailed design manipulations without reprocessing its netlist and accurate characterizations of configuration memory upsets on the resources and applications functionality.
Unfortunately, to acquire such knowledge the little documentation proved by vendors and the higher abstraction level of the commercial tools make necessary an in-depth study on bitstream files supported by preliminary assumptions and aided by custom tools.
For this purpose, the author has developed the COnfiugration MEmory Tool [99], in brief COMET, which goals and features are discussed in the following.

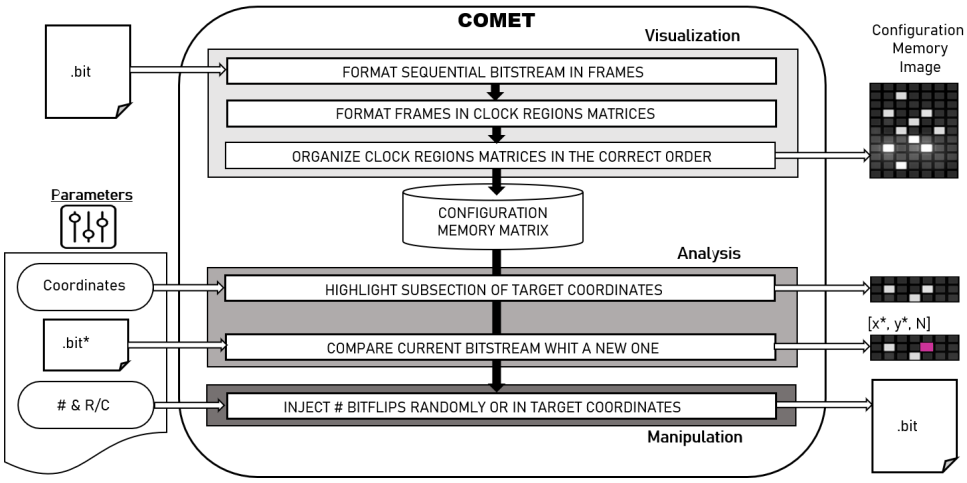

Figure 4.5: COMET Tool Flow and Functions [99]

4.2.1 COMET: COnfiguration MEmory Tool

COMET has been devised for providing to the user visibility on 7 Series FPGA configuration memory to enable its detailed analysis and decryption for performing fine-grained manipulations aiding detailed different-based reconfiguration and fault-injection campaigns.
The tool flow, schematized in Figure 4.5, consists of three main features: the Visualization stage, representing the main core of the tool, and the following steps consisting of the Analysis and Manipulation features.
In detail, the Visualization allows reorganizing the target bitstream in a form that is meaningful for the user according to the resource layer arrangement, while the Analysis and Manipulation can be used as stand-alone features as well as in cooperation. In detail, the Analysis feature allows the focus and highlight of the target device regions and to perform comparisons with additional bitstreams for decoding purposes. Furthermore, by coupling this with the Manipulation feature it is possible to perform detailed modification at the bit level relying on a coordinates organization meaningful for the user either to emulate soft-errors and to perform detailed DPR.
Above all, the synergic usage of the Manipulation, Analysis, and Visualization tasks represents a valuable means for linking configuration memory data with the specific resources they are configuring enabling the bitstream composition decoding [99].

4.2.2 Bitstream Composition

The 7 Series bitstream format and composition obtained through an in-depth analysis aided by COMET is summarized in Figure 4.6 using the Kintex-7 as the target device.

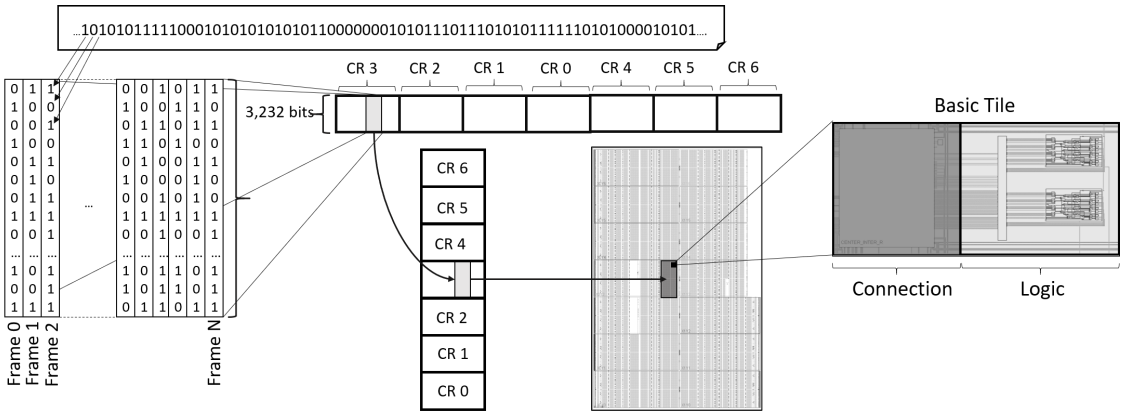

Figure 4.6: Bitstream logic organization for Xilinx 7 Series FPGA on Kintex-7 Device [27]

In detail, to map the stream of bits serially written on the bitstream on the physical resources of the programmable layer, header and tail should be removed and the so obtained data should be divided into frames, i.e. in 3,232-bit words, to be reorganized. Firstly, this series of frames must be regularly spitted in sections considering that each section contains all the frames relative to a row of Clock Regions (CR).
To obtain the mapping among these CR sections and device organization, their order should be rearranged, as typically the Clock Regions within the Top Half (as referred to in Fig. 4.4) are coded in the first part of the bitstream file mirrored while the ones in the Bottom Half are coded in the second part following the same sequence on which they appear on the device, as detailed in Fig. 4.6.
Going into the details of the configuration data of the 7 Series basic Tiles, each one of them is defined by a sub-matrix 36×64 of configuration bits. In fact, tacking as reference the Xilinx Kintex-7 XC7K325T FPGA, the reconfigurable layer of Tiles is organized on a 90×350 matrix while the configuration memory layer consists of a $3{,}240 \times 7$ frames matrix, as shown in Figure 4.7a.
Thus, each column of Tiles within a Clock Region is programmed through a sequence of 36 frames following a periodic distribution in the configuration of the logic, sequential, and interconnect elements along the X axis. Concerning the Y axis, each frame partially configures resources belonging to 50 different Tiles, as each sub-word of 64 bits is involved in the configuration of one Tile, following a periodic pattern as well except for the 32 bits placed at the half of the frame configuring Clock Resources.
Accordingly, as summarized in Figure 4.7b, the configuration data organization for a single Tile consists of a sub-matrix composed of 64-bit slices of 36 different but subsequent frames.

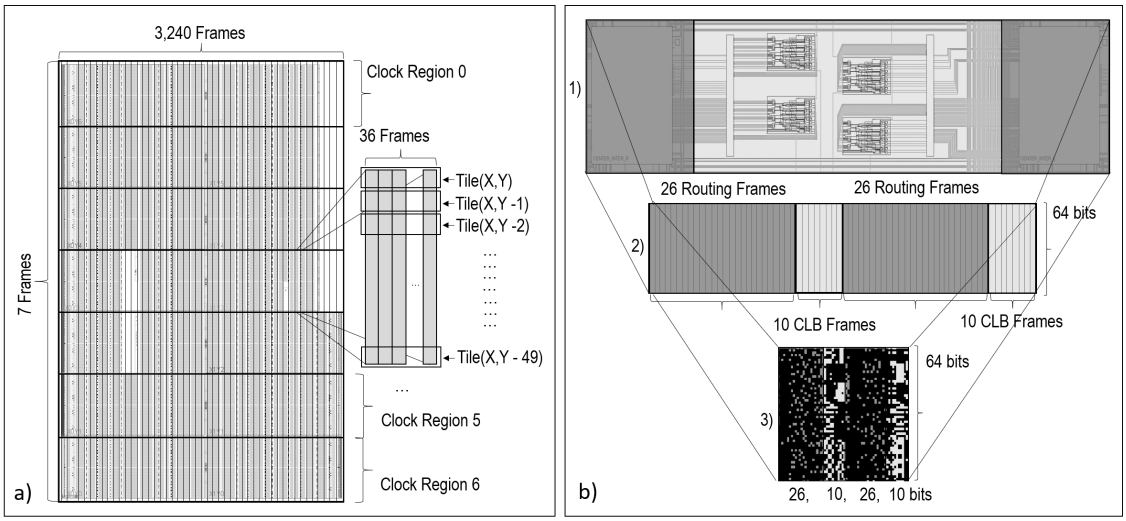

Figure 4.7: Configuration Memory Mapping of Kintex7 (a) and its Tile (b) [27]

In detail, Fig. 4.7b.1 shows the vendor device view of two contiguous Tiles while Fig. 4.7b.2 reports their configuration frames sub-matrix, where 26 frame slices program the Switch Matrix while the other 10 the CLB functionality. Finally, in Fig. 4.7b.3 the relative memory bitmap obtained with COMET from a sample design is reported: the white pixels represent the programmed bits and highlight the different distribution of routing and CLBs configuration data [99].

4.3 Reconfiguration Approaches and Interfaces

Currently, depending on the target purpose, application, and device, different techniques for run-time and partial reconfiguration exist and several interfaces can be accordingly used to access the configuration memory, as detailed in the following.

4.3.1 Reconfiguration Techniques

Today, the three main approaches to perform dynamic reconfiguration consist of *Full Context*, *Module-Based* and *Difference-Based*.
These approaches are summarized in Figure 4.8 where a reference configuration memory usage is considered (Fig. 4.8a) and frames are represented as white when not programmed and grey if *used* for configuring the sample circuit, while according to the exploited techniques, they are light-grey if not programmed but involved in the run-time reconfiguration and dark-grey if *used* and necessarily involved in the process. In detail, the frames classified as *used* are the ones that according to purposes need to be rewritten, erased, or modified at run-time.

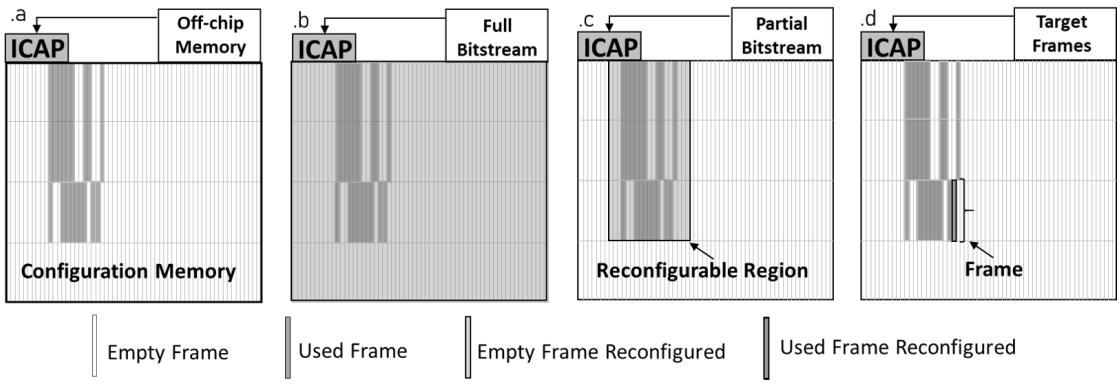

Figure 4.8: Techniques: Full Context (b), Module-based (c), and Difference-based (d) Reconfiguration on a Sample Configuration Memory Usage (a) [27]

The Full Context technique (Fig. 4.8b), implies the full and serial reconfiguration of the configuration memory content. For instance, this approach is followed in Blind Scrubbing, mostly used in the aerospace domain to avoid the accumulation of upsets in the configuration memory and consisting of the periodic in-field refresh of all the configuration data by loading them from an off-chip memory storing full bitstreams, and thus involving both used and unused frames [15][33].
When instead the Dynamic Reconfiguration is Partial, the approach can be either Module-based and Difference-Based.
In Module-based reconfiguration (Fig. 4.8c) the portions of the system defined at the design stage as dynamically reconfigurable are bounded into Reconfigurable Regions (RR), which content can be refreshed, erased, or modified at run-time by relying on pre-obtained partial bitstreams relative to target areas and stored in the off-chip memory [8][65]. Thus, also in this scenario, all the frames within the RR are involved in the run-time reconfiguration, including the empty ones.
Difference-based approaches instead (Fig. 4.8d) consist of reprogramming at run-time only the frames directly involved in the circuit upgrades that can be either stored on the off-chip memory or computed on-the-fly by the application [10][89]. Thus, even this approach implies additional effort in the design and development processes, it is the most efficient for the minimization of the configuration data and times required for the in-field upgrades.

4.3.2 Interfaces

In general, run-time reconfiguration is managed and scheduled by dedicated agents, from both software application deployed on microprocessors or from custom configuration controller, either inside or outside the target FPGA.

Table 4.1: Maximum Throughput for Configuration Ports in 7 Series Devices [47]

Configuration Interface	***Maximum Frequency***	***Data Width***	***Maximum Bandwidth***
JTAG	66 MHz	1 bit	66 Mb/s
Serial Mode	100 MHz	1 bit	100 Mb/s
SelectMAP	100 MHz	32 bit	3.2 Gb/s
ICAP	100 MHz	32 bit	3.2 Gb/s

According to the purpose, different ports can be used for accessing the configuration memory, as the consolidated Serial Mode and JTAG interfaces based on serial protocols and typically controlled from external agents with lower transmission rates, and the more recent Xilinx SelectMAP and ICAP, which have an higher throughput thanks to the higher parallelism and their hardware characteristics, as summarized in Table 4.1.
More recently, Xilinx introduced in its devices additional internal access ports which have the same transmission characteristics of the ICAP but can be driven from the on-chip microprocessor, as Processor Configuration Interfaces (PCAP) introduced in the the Zynq SoC, and from specific PCIe blocks, as the Media Configuration Access Port (MCAP) introduced in the Ultrascale Family.
In detail, the ICAP is the most consolidated port for self-reconfiguration, consisting of an internal version of the SelectMAP that can be driven through dedicated controllers implemented either in the software or hardware layer of the application. The PCAP instead is a valuable alternative in devices embedding on-chip microprocessors and highly resembles the ICAP (although the simultaneous usage of both ports is not allowed) while the MCAP, available only on Ultrascale devices, consists of an additional interface among the ICAP and specific PCIe modules.
In general, partial reconfiguration can be performed through any of these interfaces. In fact, each partial bitstream has the same format of a full one, embedding its own Header and Tail sections. Thus partial bitstreams can be delivered either from the internal access ports for in-field self-configuration or from external interfaces, as the Serial JTAG and Slave SelectMap ports, for debugging purpose or when the same medium needs to be used for both full and partial reconfigurations.

4.4 Discussion

As anticipated in Section 1.3.2, the reconfiguration time is a crucial parameter in determining the overall performance of a dynamically reconfigurable application and for this reason, it is fundamental to minimize its overhead. From its formulation presented in the Equation 1.1 of Section 1.3.2 recalled here:

$$t_{\text{rec}} = N_{\text{atomic memory unit}} \cdot B_{\text{bits per unit}} \cdot t_{\text{download bit}} \tag{4.1}$$

reconfiguration time depends on amount of data ($N_{\text{atomic memory unit}}$) involved in the reconfiguration of the target design and on the efficiency of the reconfiguration mechanism that is strictly related to the characteristics of the target device ($B_{\text{bits per unit}}$) and interface ($t_{\text{download bit}}$).
Thus, relying on commercial FPGAs, to reduce the reconfiguration time the only addressable parameter is $N_{\text{atomic memory unit}}$ since the other two are embedded in the architecture, as for 7 Series the atomic memory unit is a 3,323-bit frame that reacquires about 100 μs to be transferred through the ICAP [23].
Furthermore, when partial reconfiguration needs to be applied frequently on detailed resources, the weight of the $B_{\text{bits per unit}}$ and $t_{\text{download bit}}$ parameters in the above-discussed architecture becomes unjustified, since even if a single resource is configured by few bits (e.g., 8 for a LUT function), the full 3,232-bit frame spanning 50 CLBs must be written to update its behavior.
In this view, FeDRA represents a generalized approach for the optimization of the reconfiguration time on commercial devices by minimizing the $N_{\text{atomic memory unit}}$ exploiting the awareness of the routing configuration settings within the development flow [100][27], while ReM provides a novel architectural solution for fast, detailed, and concurrent self-reconfiguration by rethinking the configuration granularity and mechanism to minimize both $B_{\text{bits per unit}}$ and $t_{\text{download bit}}$ [101].

Chapter 5

FeDRA: a Frame-driven Routing Approach for Fast Partial Reconfiguration

In this chapter, the Frame-driven Routing Algorithm (FeDRA) developed for maximizing reconfigurable system performances through the minimization of the routing frames within the development flow is presented and discussed.
In the following, the main contribution of FeDRA is highlighted, the key intuition behind its development is detailed together with the core of the routing algorithm, and finally, the optimizations obtained on a pool of benchmark circuits with varying size, constraints, and reconfiguration purposes are presented with additional analysis on routability and execution times for different placement solutions [27] [100].

5.1 Overview

The main contribution of FeDRA consists of the high gain it provides in terms of frame saving and, subsequently, of reconfiguration time coupled with its transparency within the standard development process and other circuit performances.
The proposed solution targets commercially available Xilinx FPGAs, does not require any additional reconfiguration port, interface, or mechanism, and works in cooperation with the available vendor toolchain acting only on the routing stage and relying on it for all the other development stages.
This is possible thanks to the introduction of a novel frame-driven routing policy, which is based on an in-depth study of SRAM-based FPGA configuration data and bitstream encoding.
In fact, by starting from the optimal placement solution provided by the vendor tool for a given circuit, FeDRA is able to route its nets optimizing the usage of

their configuration frames by selecting routing segments that either minimize the absolute number of frames or maximize their sharing with other resources.
In fact, by weighting routing resources according to their configuration settings and not only from their routability characteristics is possible to implement optimizations at the lowest level without interfering with the higher level and computationally intensive optimizations performed by commercial tools, such as the efficient mapping and placement of the computing nodes of the circuits, which have the higher impact on the overall design performances.
As matter as fact, the optimizations achieved by FeDRA algorithm have confirmed its efficacy on a various set of benchmark circuits, ranging in size, composition, and compression, which has been successfully rerouted providing an average reduction of configuration frames of the 35% if compared with the one obtained on the standard flow.
Furthermore, besides demonstrating its efficient scaling with design size, composition, and congestion and its transparency on other performance parameters, FeDRA has confirmed the benefit of exploiting the awareness of the configuration memory organization in performing reconfigurable system low-level optimizations.

5.2 FeDRA: Frame-driven Routing Algorithm

FeDRA is a Frame-driven Routing Algorithm able minimize the number of configuration memory frames used by the routing resources of reconfigurable FPGA designs representing a novel and generalized approach for the optimization of their reconfiguration time.
This has been obtained through the introduction of a frame-driven routing policy within the routing stage of the FPGA development process contrived for the purpose and based on an in-depth investigation on the link among reconfigurable layer resources and the bitstream encoding which enabled the characterization of the routing resources configuration settings.
In the following, the in-depth analysis performed on routing resources and their frames upon which relies the realization of the frame-driven routing policy is presented. Then, the main intuition behind the Frame-driven Routing Algorithm and its routing policies are explained and followed by the description of its algorithmic approach.

5.2.1 Routing Frames Decoding

The enhancement of the reconfiguration procedure through the low-level awareness of the relation among programmable resources and their configuration bits implies an in-depth knowledge of the bitstream format and encoding for the target device.

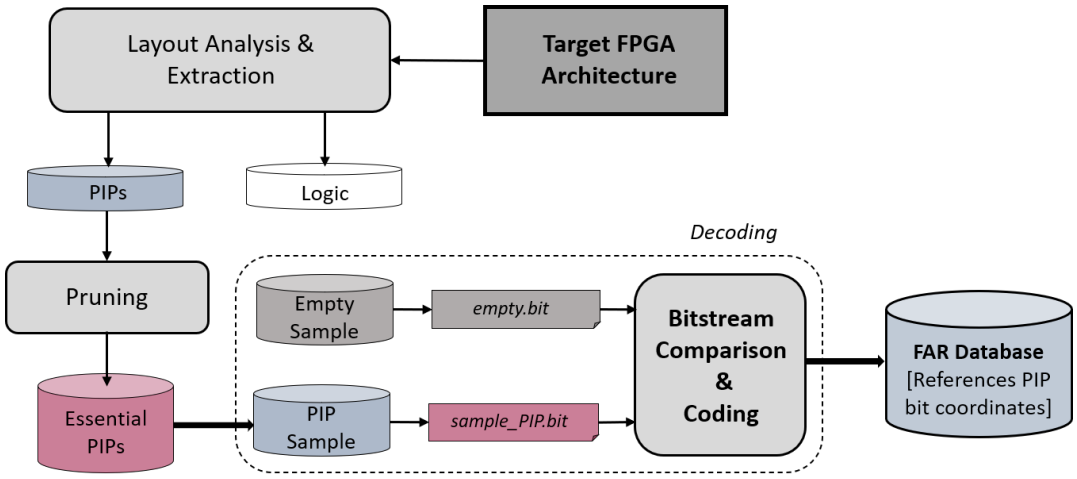

Figure 5.1: Flow Diagram of the Routing Frames Decoding

Therefore, the first fundamental step has consisted of an intensive analysis and characterization of the Xilinx routing segments organizations followed by the study of their configuration settings within the bitstream format.
The study has begun with a detailed classification of the routing resources of the Xilinx 7 Series, briefly summarized in Section 4.1.1, which enabled the characterization of PIP segments according to their connectivity properties with primitives and fixed wiring segments in terms of direction and length as well as their spatial location in the Switch Matrix array.
This analysis, coupled with the available documentation enabled the realization of a methodology supported by an ad-hoc toolchain for obtaining the frame mapping of all the routing PIPs and coding it into a database, as summarized in Figure 5.1. In detail, firstly all the PIPs of the target architecture have been extracted in their XDL-like format [97] and collected in an input database to be processed.
Since the relative amount of information for the whole fabric is enormous, to reduce the time and the computational effort for its elaboration a first pre-processing has been performed on the database to isolate the routing segments *Essential* for the decoding.
In fact, due to the high regularity of the Tile array, the behavior and the relative configuration of most of the routing resources is periodic, and thus their settings can be retrieved through offset calculations on their coordinates and based on the information obtained on a minor subset of resources identified as the baseline for the translation.
In detail, this process is called *Pruning* and enables the identification of all the resources, labeled as *Essential*, which coding cannot be retrieved from other PIPs drastically reducing the elaboration effort for the next stage. In this way, the *Essential* PIPs processed in the following *Decoding Stage* for the Kintex-7 XC7K325T FPGA are 21,081 against the original amount, which is around 124 millions.

During the *Decoding Stage*, for all so obtained PIPs special bitstreams where only the target segment is programmed are iteratively generated and their configuration frames are extracted in terms of coordinates and coded as Frame Address Register (FAR) information through a dedicated TCL-based framework interacting with the vendor tool and COMET.
Therefore, at the end of this process, an output database containing the reference bit coordinates of all the PIPs is obtained, including the *non-Essential* ones, which coding is retrieved thanks to the periodicity of resources and bitstream format, providing precise information about the number and location of the bits within the frames used to program each routing segment [23][100][99].

5.2.2 Frame-driven Routing Policy

Relying on a detailed study of the routing frames database has been possible to observe that routing PIPs are configured by multiple bits that are distributed on a number of frames which depends on the specific segment.
In detail, routing resources can be configured by bits distributed on 1 up to 4 frames which, as discussed, are possibly involved in the configuration of all the resources on the same column of Tiles within the Clock Region height and thus, can include bits which are in common to multiple routing resources in the same Switch Matrix or others within the same CR column.
To clarify, in Figure 5.2 four sample routing segments are reported with the identification number of the frames involved in their configuration settings.
In detail, the examples of PIPs configured by bits of 1, 2, and 4 frames are highlighted as well as the case in which their configuration bits belong to the same frames.

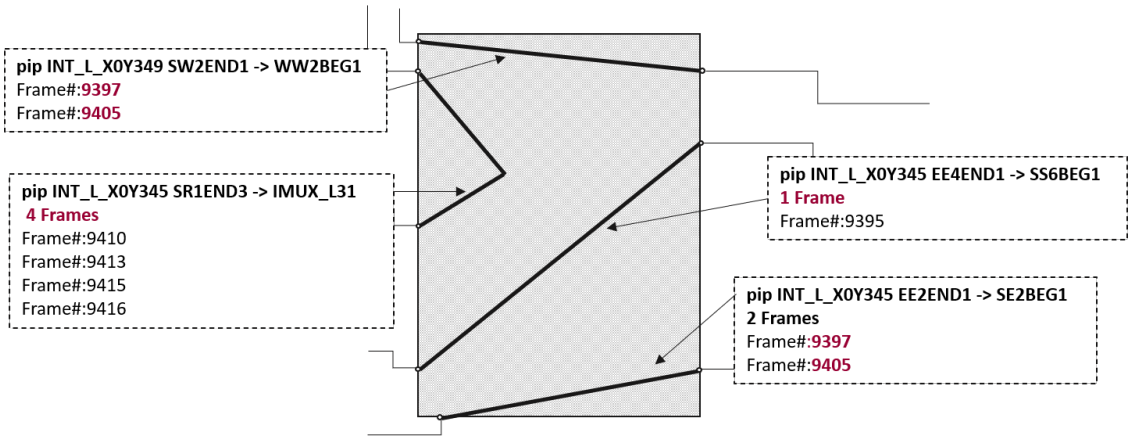

Figure 5.2: Sample Configurations of PIPs within the Switch Matrix: 1, 2 or 4 Programmed Frames and their Possible Overlapping

Considering this, the main intuition behind FeDRA consists of the possibility of routing nets inserting an additional goal aimed at the minimization of the overall amount of frames by evaluating PIP segments according to their configuration setting with the policy of minimizing the overhead they introduce in the design.
In fact, each SM includes several PIPs that allow the same connectivity through the same direction, as each Junction represents the source or the drain for an average of thirty PIPs that, consequently, have the same signal driver and possibly enable the same connectivity towards a given direction while relying on different frames for their configuration.
Thus, according to their bit coding, the weight of two nets implementing the same topological connectivity can be different in terms of configuration frames usage, making it possible to locally change the routing connecting already placed logic nodes by reshaping nets using segments which either introduce less new frames or are configured by the same frames already used by other resources without modifying the original circuit topology.
To elaborate more, when one PIP has to be selected during the creation of a net, according to its configuration settings, frames can be saved considering three different situations, the *Absolute, Intra-net, and Inter-net scenarios*, which enables accordingly two different routing policies.

Absolute Scenario

The Absolute Scenario happens when two resources have the same topological behavior but one of them introduces fewer programming frames in absolute terms, as reported in the example in Figure 5.3.

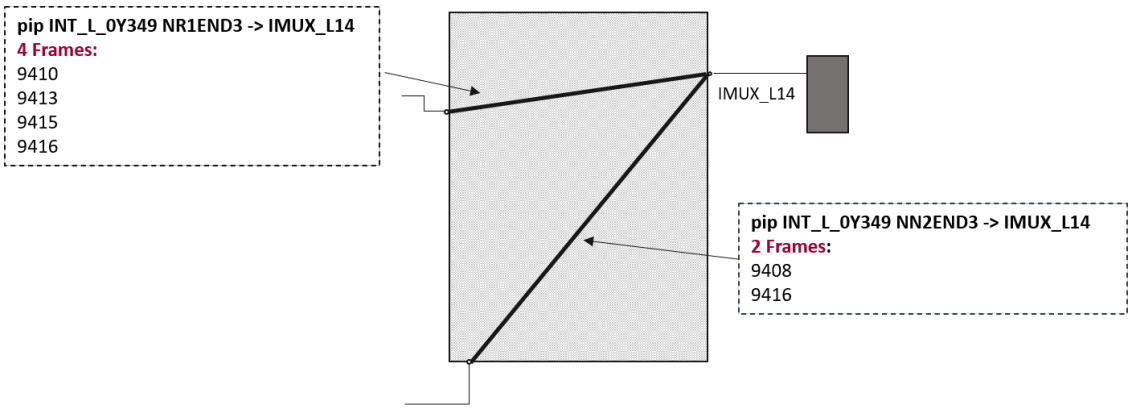

Figure 5.3: Absolute Scenario: between two PIPs with the same Connectivity Behavior the the one introducing the Absolute Minimal Number of Frames is Selected

In fact, the two PIPs show similar connectivity properties since both of them put in communication a wire coming from a SM located at the South with the Input MUX 14 of the CLB in the target Tile, as both NN2 and NR1 wires come from one Tile South allowing the connection with distance 1 in direction North. The difference between the two segments consists of the number of frames used for their configuration. Thus, without considering any other programmed frames in the target SM column, through the selection of the routing segment:

pip INT_L_0Y349 NN2END3 -> IMUX_L14

two configuration frames are saved while reaching the same target point.
This optimization policy can be considered as an *Absolute Routing Policy* since no other programmed routing is considered.

Intra-net Scenario

The Intra-net scenario considers the situation in which some of the PIPs used to build the same net belong to the same Tile Column and thus one or more of their configuration bits can be distributed across the same frames.

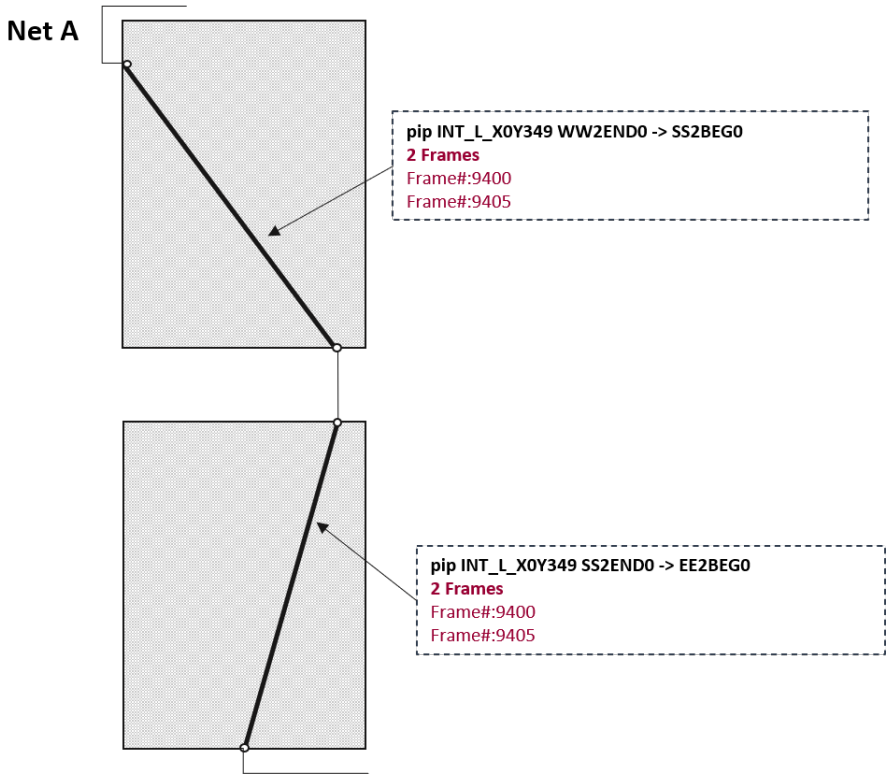

Figure 5.4: Intra-net Scenario: PIPs within the Same Net which Belong to the Same Tile Column Allow their Selection to Maximize the Frame Overlapping

This can happen when the PIP segments involved on the same path are in SMs that have the same X coordinate and their Y coordinates fall within the same Clock Region. In this situation is highly probable that PIPs have multiple configuration frames in common.
In Figure 5.4 a simple example of this situation is reported in which two subsequent PIPs on the same Net A are activated by two bits that belong exactly to the same two frames.
This scenario allows optimizations that are related to the frames used by other resources enabling the maximization of the frame overlapping and thus has been classified as a *Relative Routing Policy*.

Inter-net Scenario

The Inter-net scenario considers the situation in which there are not shared frames among PIPs belonging to the same net while among the same Tile Column other routing resources are programmed on which is possible to perform the frame overlapping.
In this case, the new routing PIP can be programmed by bits that belong to the frames used by other routing resources routed in previous routing steps.
In Figure 5.5 a basic example of the Inter-net scenario is reported:

pip INT_L_X0Y349 EE2END1 -> SE2BEG1

is routed for Net B and it is programmed by the same frames used to program the following segment which belongs the Net A:

pip INT_L_X0Y349 SW2END1 -> WW2BEG1

As for the Intra-net scenario, this optimization consists of a *Relative Routing Policy*, since the minimization of the frames again involves the overlapping with the configuration data of other routing segments.

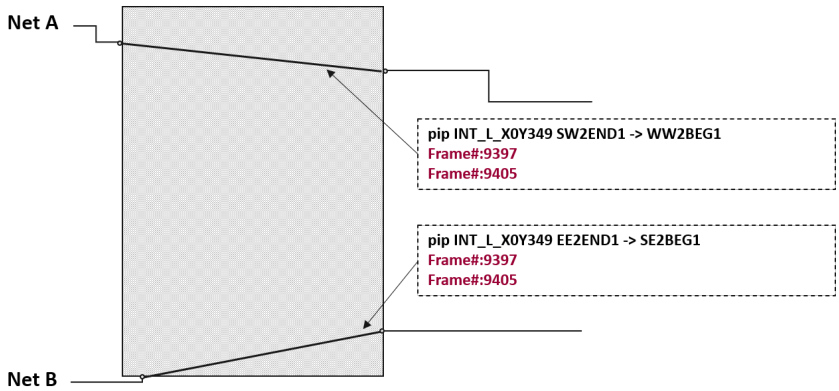

Figure 5.5: Inter-net Scenario: PIPs Building Different Nets and Belonging to the same Tile Column Allow their Selection to Maximize the Frame Overlapping

Routing Policies Definition

With respect to the aforementioned possible scenarios, two main routing policies have been conceptually identified:

- **Relative Routing Policy**: considering the programmable segments accessible through the target Junction with the same routing properties, it is selected the one that provides the maximal overlapping with the already programmed frames in the current Tile Column within the target Clock Region for minimizing the configuration data overhead introduced by the new resource.
- **Absolute Routing Policy**: when any overlapping opportunity is available within the same Tile Column with already programmed frames, among the possible PIPs with the same routing properties, it is selected the one which uses the lower amount of frames for its configuration for minimizing the configuration data overhead introduced by the new resource.

In Figure 5.6 a demonstration of the two policies applied on simple net is provided: Fig. 5.6a reports the routing solution provided for the target net by the standard vendor router while Fig. 5.6b reports the routing solution obtained with FeDRA implementing the Frame-driven optimizations.

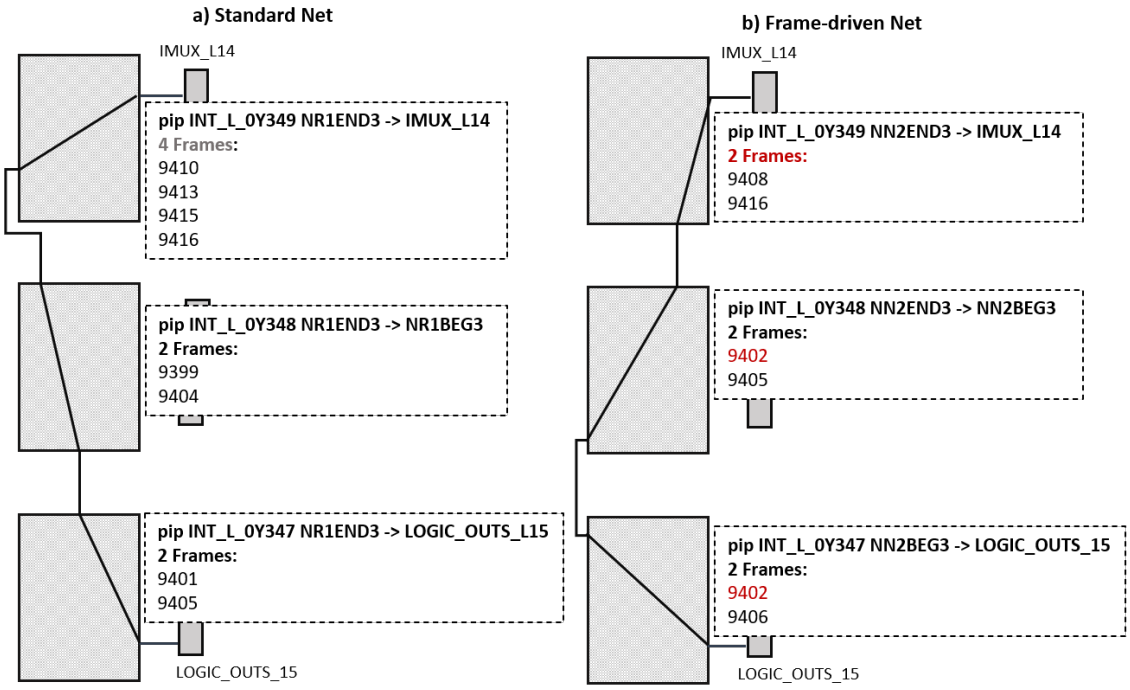

Figure 5.6: FeDRA Routing Policies: a) Standard Net Routed by the Vendor Tool; b) The Same Net Routed with FeDRA Optimization with 3 Saved Frames

As it is possible to see, the two nets present the same topological connectivity driving a signal from the Output 15 (LOGIC_OUTS_15) of one Slice to the Input 14 (IMUX_L14) of another Slice of a Tile located at North through three routing steps and using for each one of them one PIP for each SM.
The net routed with the standard router requires programming bits which are distributed among eight frames for its configuration. In detail, two frames are used to switch on the first connection with the Slice Out, the second routing segment requires two additional configuration frames, while other four are needed in the last step for the connection with the Slice Input and any of these frames is configuring more than one resource among the target net.
The net routed with FeDRA instead implements the same connectivity but uses only five frames. In fact, the first two segments of the rerouted net are configured by bits distributed as well on two frames, but one of them (i.e., the frame 9402) is in common among the two resources as one Frame has been saved through the Relative Routing Policy. Additionally, the last PIP used to reach the Input 14 is configured by bits belonging only to two frames, which is half of the ones used for this connection on the Standard Net, saving two other frames through the Absolute Routing Policy. Thus, FeDRA routes the same net relying on the same amount of resources while saving three configuration frames.
As a final consideration to introduce FeDRA from the algorithmic point of view, besides the aforementioned scenarios and routing policies, the FeDRA algorithm manages the frame optimizations relying on the same computational approach. In fact, from the algorithmic perspective, every time a new routing segment is considered the same mechanism is exploited for evaluating and weighting the overall configuration data overhead introduced by the new resources, as will be discussed in the next Sections.

5.2.3 FeDRA Algorithm

The FeDRA algorithm has been instrumented to cooperate with the Xilinx Tool toolchain to provide a complete routing solution starting from the optimal circuit placement solution obtained through the Vivado Placer, which consists of the information about the computing nodes, as LUTs, FFs, hard macros, and IO pins location and how they should be connected.
This information is acquired by FeDRA from the vendor tool and through a TCL-based framework integrated on it, which translates this input into a format suitable to be processed, such information is used as initial constraint to reshape all the nets according to the aforementioned policies.
In fact, by enebling the Frame-driven router to rely on the commercial toolchain for the other Implementation steps it is possible to gain the maximum benefits from vendor optimization policies while introducing the additional frame saving almost transparently versus the original circuit topology and performance parameters.

Approach

To summarize FeDRA optimization approach a first streamlined version of its pseudo-code is provided in Figure 5.7.
Initially, the optimal placement solution is acquired from the vendor toolchain and the final routing solution, consisting of the information about the routed nets, is empty, as well as the database tracing the position of frames used within the design. At this point, the information about the placed nodes to be connected is iteratively processed until all the routing of all the circuit nets is obtained.
In detail, a net is defined by its placed Source and Drain (or Drains, since each net can have more than one Drain). The path connecting these couples can be split into intermediate steps, each one of them consisting of the iterative individuation of temporary Sources and Drains which allow being joined through the activation of one PIP segment.

```
//Initialization Phase
READ Vivado_Placement_Solution
Routing_Solution = {EMPTY}
Current_Design_Frame_Vector = {EMPTY}

//Net Extraction
While (NET_DATABASE != {EMPTY}){
    READ NET
    while(NET != {ROUTED}){
      SET Source_tmp
      SET Drain_tmp
      //Frame Filtering
      for each PIP(Source_tmp; Drain_tmp) in PIP_DATABASE{
          cost_new = compute_cost_function()
          if (cost_new < cost_tmp){
            cost_tmp = cost_new
            PIP_best = PIP
            next_DFV = tmp_Design_Frame_Vector
          }
      }
      //Update
      ADD PIP_best to NET
      Current_Design_Frame_Vector = next_DVF
    }
    ADD NET to Routing_Solution
}
Routing_Solution = {COMPLETE}
```

Figure 5.7: The FeDRA Approach Pseudo-code [27]

Since for each one of these temporary pairs a pool of PIPs exists for implementing their connection, the PIP within this subset introducing the minimum overhead in terms of additional frames is identified through a dedicated cost function. In detail, this cost function is defined and evaluated in such a way that both Absolute and Relative Policy are considered according to the values stored in the so-called *Current Design Frame Vector*, a dedicated flag vector in charge of keeping records of the frames incrementally programming the routing of the design.
In fact, when a new segment is under evaluation for being added to the routing solution, the frames involved for its configuration are compared through basic bitwise logic operations with the one already programmed and stored in *Current Design Frame Vector* to obtain the absolute number of additional frames which would be introduced by the current resource.
The computation of the cost function is schematized in Figure 5.8. The *Current Design Frame Vector* holds the flags for the routing frames usage within the FPGA by setting the matching bits at 1 when already programmed or leaving them at 0 if not yet included in the configuration of the design.

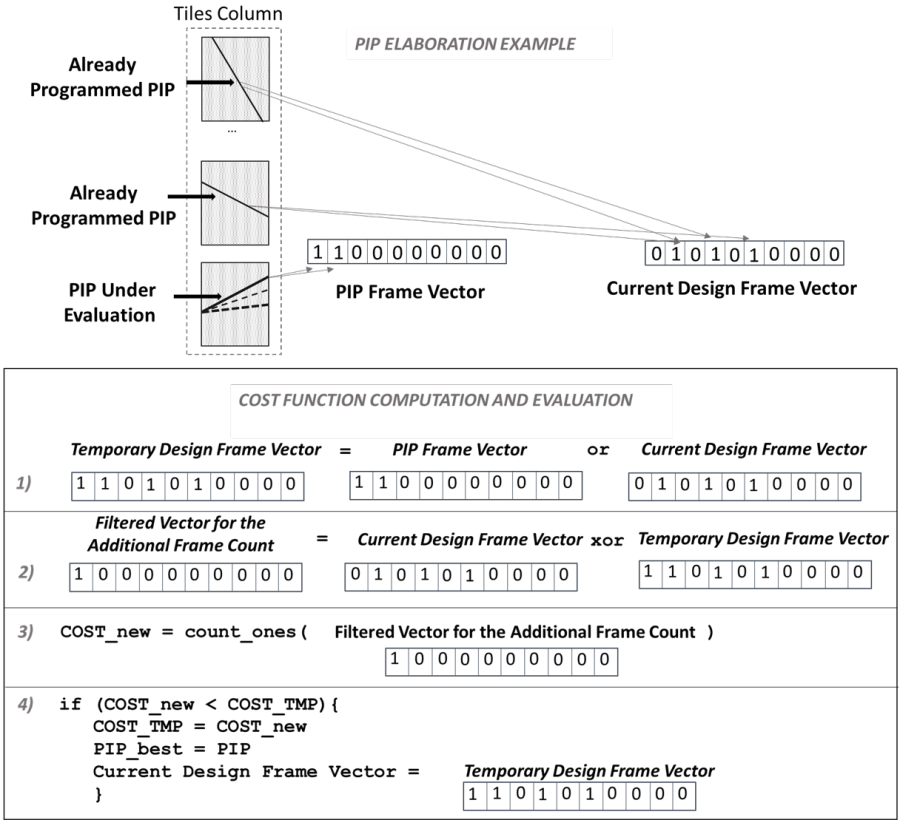

Figure 5.8: PIP Elaboration Example: Highlights of Cost Function Computation and Evaluation

Each PIP on the database is associated with its relative *PIP Frame Vector*, which holds the frames involved in its configuration, and every time it is evaluated the bitwise *or* among its *PIP Frame Vector* and the *Current Design Frame Vector* is performed producing the so-called *Temporary Design Frame Vector*. This vector provides an image of the total amount of frames used if the new segment is added to the final routing solution and allows to consider both the potential overlapping and absolute frame saving in one single computing step.
Then the bitwise *xor* among this vector and the *Current Design Frame Vector* is performed providing a new vector, the *Filtered Vector*, which states only the additional frames introduced by the current PIP and enable to obtain the cost involved in its configuration by counting the 1s it stores. If such cost is lower than one which is currently set as the best, it takes its place and the segment under evaluation is saved as the best temporary selection.
When the whole pool of PIPs under evaluation for the temporary Source & Drain pair has been evaluated, the one currently set as the best candidate thanks to its optimal cost is validated and updated in the routing solution, the old *Current Design Frame Vector* is replaced by the relative *Temporary Design Frame Vector*, and the old Drain became the new Source for next algorithm iteration.
All these steps are iteratively performed until the routing of the target net is obtained and, processing subsequently net by net, the complete routing solution is found.

Algorithm

The Frame-driven algorithm, which detailed description is provided in Figure 5.9, is based on a core Pathfinder [80] router which integrates the proposed frame optimization policy and its execution can be conceptually divided into five different phases.
Once the Vivado placement solution P_{sol} is obtained, the Initialization Phase begins. In this phase the storage databases and arrays are initialized: the Routing Solution R_{sol} is empty as well as the *Current Design Frame Vector* (V_{fid}), which elements are set to 0.
As mentioned, *Current Design Frame Vector* is used to keep track of the Routing Frames (f_{SM}) programmed within the design, thus it is dimensioned according to the target FPGA.
In detail, each one of its elements corresponds to a target frame within the configuration memory array, and thus V_{flD} is addressable through a pair of coordinates, the first identifying the position of the configuration frame with respect to device X-axis, while the second indicating its location according to the device vertical axis and called G, where each G value represents a Clock Region Row consisting of a Column of 50 Tiles.

```
P_sol = Placement Solution

//1.Initialization Phase
R_sol = {0};
∀ X,G ∈{MAX_X,MAX_G} → V_fid (X,G)[f_SM] = {0};

//2.Nets Extraction Phase
∀ net ∈ P_sol → list [net];

//3.Routing
for n ∈ net : V_e(S,D) = list_SD[net]{
   for D ∈ E(S,D) : (S,D_i){
      S_TMP = S;
      D_TMP = S;
      while (D_TMP ≠ D_i){
           D_TMP = pathfinder(S_TMP, D);
           set_PIP = list(S_TMP, D_TMP);

           //4.Frame Filtering Optimizations
           W_max = f_SM;
           ∀ PIP ∈ set_PIP do{
                  V_fID_TMP[f_SM] = V_fID (X_STMP,G_STMP) or V_DB_PIP(PIP);
                  H = card(V_fID_TMP[f_SM] xor V_fID(X_STMP,G_STMP));
                  if (H < W_max){
                         W_max = H;
                         V_fID_BEST = V_fID_TMP[f_SM];
                         PIP_BEST  = PIP;
                  }
           }

           //5.Update Routes and Frames
           R_sol = update_route (PIP_BEST);
           V_fID(X_STMP,G_STMP)[f_SM] = V_fID_BEST;
           S_TMP = D_TMP;
           }// PIP_i is routed

   }// S-D_i is routed
}// Net is routed
// Circuit is routed
```

Figure 5.9: FeDRA Algorithmic Description [100]

Within the second phase, the Next Extraction, the complete list of nets *list[net]* to be routed with the current Placement Solution is obtained. In detail, the nets are identified according to their Source S and their Destinations D_i, as either single and multi-drain nets are taken into account by the algorithm.

At this point, the Routing Phase begins, in which all the pairs of S and D_i are iteratively processed through a Pathfinder-based approach which recursively identifies all the steps required to build the path to connect S and D_i.
Thus, each net is decomposed in subsequent steps and each step consists of the definition of intermediate couples of temporary Source and Drain (S_{TMP}, D_{TMP}) which can be connected through a single PIP segment. Thus the net is incrementally routed by extracting through the function *pathfinder()* a D_{TMP} at each iteration, which will become the new Sources S_{TMP} for the next one as soon as the valid connection to reach it is identified. In detail, by elaborating at the Tile level, the *pathfinder()* function computes the Tiles reachable from the current destination within one step and identifies which is the most reasonable Temporary destination D_{TMP} to be reached from the temporary Source S_{TMP} in terms of Manhattan Distance.
Once the most suitable destination is identified, the function *list()* provides all the routing segments which enable from the current S_{TMP} (the current Junction in the current Switch Matrix) to go toward D_{TMP}, which are saved the set_{PIP} list and consist of all possible PIPs that go in a routability-equivalent direction.
At this point, the router enters the fourth phase, in which the frames optimization takes place enabling the selection of the PIP within the set_{PIP} list which has the lower weight W according to the cost function summarized in Fig. 5.8. To perform this calculation, every time a new group of PIPs is evaluated the initial weight W_{MAX} is set to the maximum f_{SM}, which consists of the worst case in which all the 26 routing frames within a Switch Matrix Column are used.
Thus, each PIP within the set_{PIP} is evaluated performing the logic *or* between the aforementioned *Current Design Frame Vector* $V_{fID}(X_s,G_s)$ holding the current frame usage within the target column and row and the *PIP Vector* (V_{fPIP}) containing the frame required for the target PIP configuration.
After performing the logic *xor* between the so-obtained *Temporary Design Frame Vector* (V_{fID_TMP}) and $V_{fID}(X_s,G_s)$, the frame overhead H involved in the usage of the current segment is obtained through the function *card()*, which counts the flags currently high in the so obtained vector.
If the obtained of H is lower than the current W_{MAX}, the new H value replaces the old W_{MAX} and the PIP under evaluation is set as the temporary optimal solution (PIP_{BEST}), together with its relative temporary frame configuration it would imply (V_{fID_TMP}).
After all the routing segments in the set_{PIP} list are elaborated, the fifth phase begins and the Update is performed: the PIP currently saved as PIP_{BEST} is confirmed as a new routing segment and added to routing solution through the *update_route()* function; the relative V_{fID_TMP} becomes the new *Current Design Frame Vector* $V_{fID}(X_s,G_s)$ and the last D_{TMP} is set as the new S_{TMP} for the next iteration.
Phases three, four, and five are reiterated until all the nets are routed and a complete routing solution for the design is obtained [100].

5.3 Experimental Results

The evaluation of the efficiency of FeDRA algorithm optimizations has been performed on several benchmark circuits using as target device the Kintex-7 XC7K325T FPGA and comparing their frame usage and reconfiguration time when implemented through the standard Xilinx P&R flow with the ones achieved through the integration of FeDRA in the routing stage.
To make a comprehensive analysis, FeDRA optimizations have been applied on circuits of different sizes and compositions, as well as introducing different area and reconfiguration constraints.
Additionally, the performance of FeDRA has been characterized in terms of execution time and routability by evaluating the composition of its routing solutions in terms of short segments and both horizontal and vertical long lines according with varying placement constraints.

5.3.1 Experimental Setup

To characterize the efficiency of FeDRA optimizations the benchmark circuits pool has been selected to present high variability in the size and resource usage and additional area constraints have been eventually introduced in the Placement to evaluate its impact and efficacy for different levels of circuit routing congestion. Therefore, to evaluate FeDRA optimizations, two versions of each benchmark circuit have been implemented, one relying on the standard vendor tool flow within Vivado (which will be referred to as Standard in the following), and the other obtained through the insertion of FeDRA within the routing stage.
At the end of both development flows, FeDRA and Standard, the benchmark bitstreams have been obtained for performing frame usage analysis and for the reconfiguration time measurement through the deployment on the target device, as summarized in Figure 5.10.

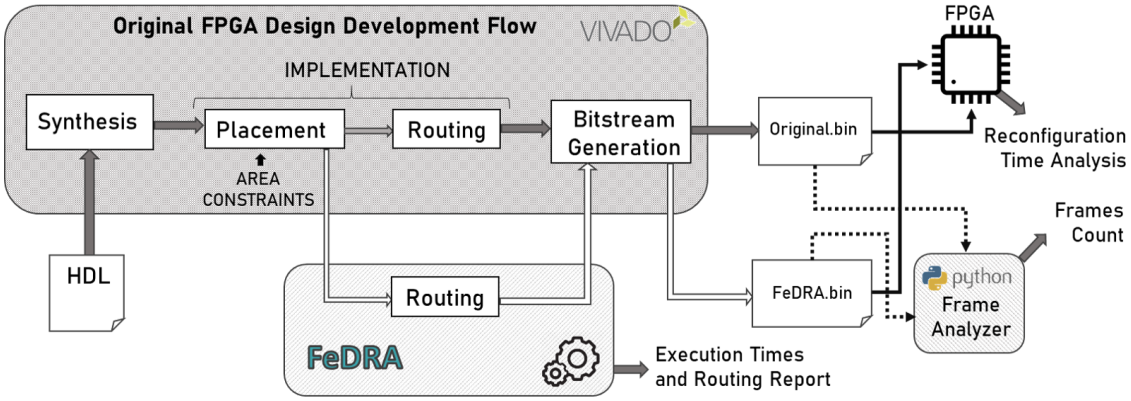

Figure 5.10: FeDRA Evaluation framework [27]

Table 5.1: Characteristics of Benchmark Circuits Implemented within Xilinx Vivado IDE on Kintex-7 in Standard Condition

Circuit	*LUTs Count*	*FFs Count*	*Nets Count*	*PIPs Count*	*IOs Count*
b05	84	34	110	1,596	39
b12	223	119	242	3,776	13
Cordic_r2p	949	1,001	1,120	29,163	74
b14	2,123	219	2,103	29,083	88
miniMIPS	2,712	2,000	4,797	72,987	175
b22	3,686	613	2,942	42,203	56
b17	4,897	1,350	4,850	70,937	136
b18	11,904	2,842	11,152	158,886	61
b19	22,565	5,686	20,877	304,124	53

In detail, the analysis on the frame optimization has been performed relying on the Frame Analyzer, a dedicated software tool developed for this purpose and capable of elaborating the target design bitstreams to provide the amount and position of routing frames with programmed bits, and thus involved in the configuration of the circuit, while detailed report can be generated by FeDRA from which has been possible to obtain information about execution time and routability metrics.
The usage of a variegated pool of benchmarks coupled with the additional area and placement constraints inserted in a second time allowed the characterization of FeDRA performances in terms of optimization, routing solutions, and execution times and confirming its efficacy and scalability versus the circuit size, and congestion, and routability. In detail, to perform the evaluation on circuits with different characteristics in terms of composition and resource utilization seven ITC'99 benchmarks with varying sizes and functionalities [102], a Cordic (COordinate Rotation DIgital Computer) Core [103] in rectangular to polar configuration (r2p), and miniMIPS processor [104] have been the selected as benchmarks.
The implementation characteristics of the benchmark circuits obtained in standard condition (i.e., with the nominal frequency of 100 MHz and no placement constraints) obtained through the Vivado toolchian and targeting the Kintex 7 FPGA are reported in Table 5.1 in terms of resource usage and composition.

5.3.2 Optimization in Standard Condition

The first evaluation of FeDRA optimizations has been performed on the benchmark circuits implemented in standard conditions, meaning that no placement and area constraints have been introduced and the standard frequency of 100 MHz has

been used.
Two versions of each circuit have been implemented on the Kintex-7 XC7K325T device, respectively routed with the Standard vendor router and with FeDRA.
The comparison in terms of frame usage and optimization for the two versions is reported in Table 5.2 showing that the circuits routed with FeDRA present a strong reduction in the routing frames needed for their programming and achieving an optimization that never goes below the 35%.
In general, smaller circuits show a higher frame saving, reaching the best optimization for Cordic Core and b12, respectively 41.3% and 38.8%, still following a positive relationship between the efficiency and the design size as presenting a slight dependency with the benchmark dimensions.
Besides, the trend is also related to the composition of the circuit in terms of logic nodes, registers, IO ports, and the different amount of nets needed to connect them. As an example, it is possible to notice that FeDRA optimization for the miniMIPS is slightly lower than the one achieved for b22 and b17.
In fact, by looking at the ratio between the routing segments and the placed nodes for this design it is possible to observe that the miniMIPS presents a higher routing density.

Table 5.2: Routing Frame Usage Comparison between Benchmarks Routed with the Vendor Flow and Routed with FeDRA in Standard Condition

Benchmark Circuits	***Frame Usage [#]***		***Optimization***
	Standard Router	***FeDRA Router***	
b05	183	113	**38.3%**
b12	237	145	**38.8%**
Cordic_r2p	702	412	**41.3%**
b14	1,794	1,126	**37.2%**
miniMIPS	2,542	1,645	**35.3%**
b22	1,595	1,016	**36.3%**
b17	2,910	1,844	**36.6%**
b18	4,514	2,890	**36.0%**
b19	7,005	4,555	**35.0%**

The additional analysis on the routing resource usage on the circuits elaborated with FeDRA confirmed that the introduced overhead is infinitesimal and inversely proportional versus the circuit size becoming irrelevant for bigger circuits: as for b05 the PIP usage results increased by 0.97%, for the miniMIPS the percentage drops to 0.16%.
This is in line with the fact that any negative influence on the benchmarks routed with FeDRA has been found in terms of delay, as they keep meeting the original timing requirements. This further confirmed the advantage and efficacy of relying on the original and optimal placement obtained through the vendor placer, which acting as the initial constraint for the routing algorithm results in the fact that the circuit topology is not altered by the optimization performed.
The analysis performed on the frame distribution for the benchmark b14 is reported of Figure 5.11, and consits of a graph showing the amount and location of the frames programmed on the Kintex-7 device for the circuit b14 for the two implementations: the one obtained with the vendor tool (Fig. 5.11a), and other with FeDRA (Fig. 5.11b).
In detail, on the Y-axis the Clock Region Rows are reported, which correspond to the full height of one frame, while on the X-axis takes into account the Tiles and each bar of the graph represents the number of frames used within a column of 50 Tiles to fully program b14 circuit.
From the graph, it is possible to observe that the frame distribution, and thus the circuit topology, remains unchanged, while the number of programmed frames results lower for the circuit routed with FeDRA.

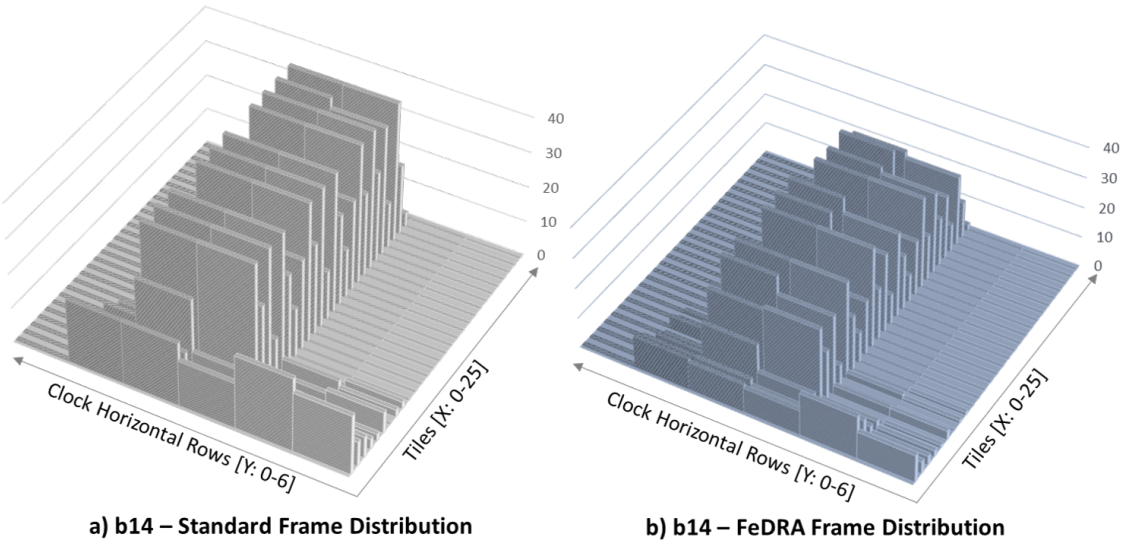

Figure 5.11: Frame Distribution and Circuit Topology comparisons for b14 benchmark: obtained with the vendor tool (a), and with FeDRA routing (b)

5.3.3 Optimization with Tight Area Constraints

The FeDRA algorithm has been further evaluated to characterize its performance in the frame optimization when higher area constraints are imposed.
In detail, the benchmarks placement has been constrained in the smallest possible area allowed by the resources used for the circuits, to cover the scenario in which their area overhead must be minimized and to produce increased routing congestion.
In fact, by forcing the designs to fit in the minimum device area the routing congestion for the target circuit increases as harder connectivity requirements needs to be met due to denser placement introduced by the compression.
The detail of the so-obtained benchmark compressions are reported in Table 5.3, showing the Slice usage required for holding all the computing nodes of each circuit, the ones available within the minimum possible bounding box for the target benchmark placement, and the consequent circuit congestion percentage considering the slices within the constrained area.
As is possible to see, the compression rate of some circuits is lower, as in the worst case of b17, mainly due to their different composition which implies varying requirements in terms of memory slices (SLICEM) and IO pins, and thus reducing the degree of freedom in the placement of the nodes.
In Figure 5.12, the different placement solutions obtained in standard condition (Fig. 5.12a) and with the constraints within minimal bounding box (Fig. 5.12b) are shown for b18 benchmark highlighting their different distribution.

Table 5.3: Benchmark Circuits Compression Metrics Under Tight Area Constraints: Slice Usage and Congestion Rate

Benchmark Circuit	***Slice Required*** *[#]*	***Slice Available*** *[#]*	***Module Congestion*** *[%]*
b05	24	24	100
b12	56	56	100
Cordic_r2p	291	300	97
b14	241	252	95.63
miniMIPS	888	900	98.67
b22	803	816	98.41
b17	1,321	1,786	73.96
b18	2,899	3,096	93.64
b19	5,953	6,392	93.13

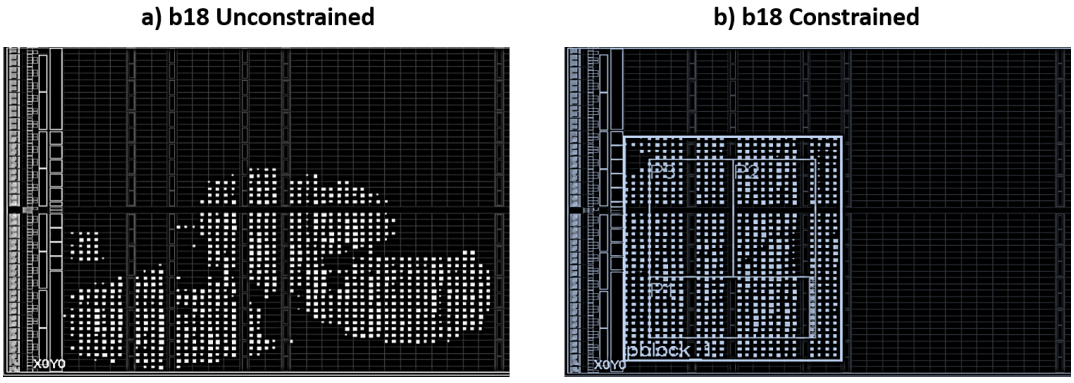

Figure 5.12: Slice Placement Solutions for b18 Benchmark: s) Unconstrained, and b) Constrained [27]

Thus, both the Standard vendor and FeDRA routers have been applied to the benchmark circuit placements obtained by imposing the tightest area constraints to evaluate the optimization achieved in case of high routing congestion and node density.
The results of this analysis are provided in Table 5.4 reporting the frame usage in the two cases and highlighting the saving percentage.
In this case, FeDRA optimizations range from a maximum of 39.2% for the smallest circuit to a minimum of 32.2% for the biggest one, showing a slight degradation of the frame saving as the circuit size increases.

Table 5.4: Routing Frame Usage Comparison between Benchmarks Routed with the Vendor Flow and Routed with FeDRA Under Tight Area Constraints

Benchmark Circuits	***Frame Usage [#]***		***Optimization***
	Standard Router	***FeDRA Router***	
b05	143	87	**39.2**%
b12	170	104	**38.8**%
Cordic_r2p	335	208	**37.9**%
b14	442	291	**34.2**%
miniMIPS	1,409	912	**35.3**%
b22	1,145	764	**33.3**%
b17	2,070	1,352	**34.7**%
b18	3,185	2,108	**33.8**%
b19	5,817	3,942	**32.2**%

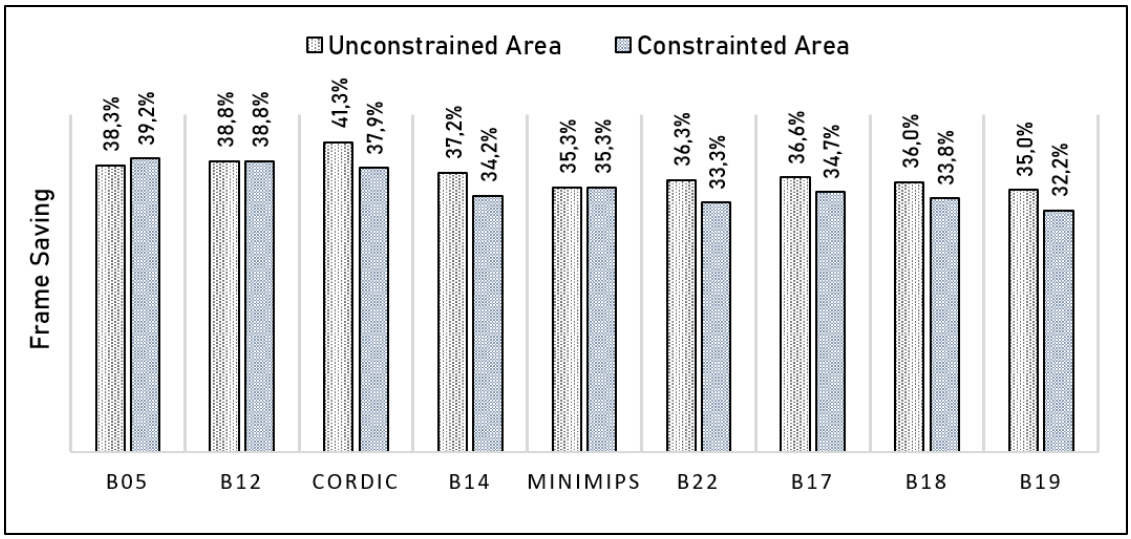

Figure 5.13: FeDRA Frame Saving Comparison among Benchmark Circuits Unconstrained (Unconstrained Area) and with Tight Placement Area Constraints (Constrained Area)

In relation to the achievements obtained without constraints, in Figure 5.13 a bar graph providing a comparison with the FeDRA optimizations in the case of bounded placement and higher congestion is reported to highlight their trends.
From this comparison emerges that higher congestions imply a slight degradation of FeDRA efficiency which again scales with the circuit size. Additionally, in the case of very small circuits, higher congestion can be beneficial, as for b05 benchmark, which unconstrained version with a minor amount of resources with a more scattered placement provides fewer opportunities for the frame overlapping versus its constrained and more compact version.
Besides, the observed trends confirm that in general FeDRA optimization scales efficiently with circuit size and congestion, as in both cases its efficacy shows a minimal dependency with such parameters keeping the frame saving around the 40% for smaller and unconstrained circuits and never going below the 32% for the bigger and more congested ones.

5.3.4 Performance and Routability Analysis

To further characterize the FeDRA router, additional evaluations on its execution time and routability performance have been carried out.
To perform this evaluation, FeDRA has been applied on the benchmark circuits with different levels of congestion starting from the worst case of the minimum bounding box obtained for the analysis under tight area constraints and gradually enlarging this constraint horizontally, allowing wider circuit placements along the X-axis.

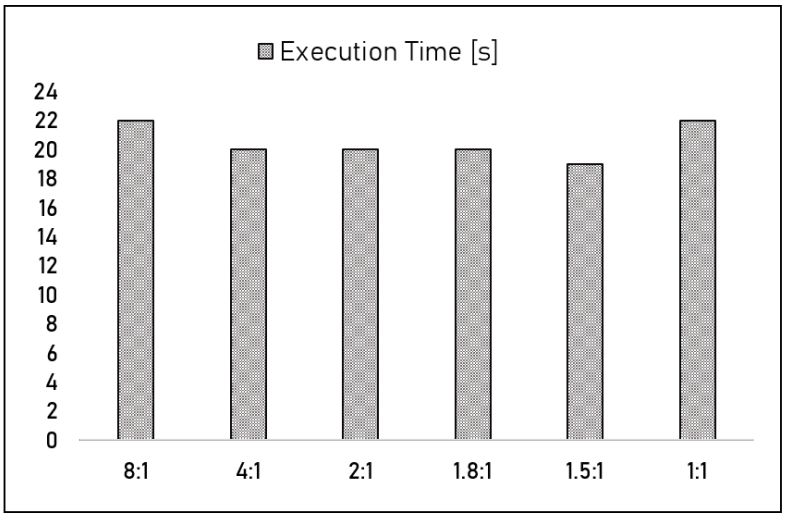

Figure 5.14: The FeDRA Algorithm Results on Execution Times with different Area Constraints for the Benchmark b12

In detail, the analysis has been performed on several placement solutions ranging from the smallest constrained area, referred to in the following as 1:1, and a bounding box 8 times wider, respectively referred to as 8:1. In total, six placement solutions within this range have been considered and referred in the following to as the ratio among the most constrained area and the relaxing step. As an example, placement 2:1 is a placement that allows twice of the area along to the X-axis with respect to 1:1.
The trends observed in the execution times highlight that longer execution times are required for the edge cases of the smallest and the largest bounding placements. In fact, starting from the more extended placements, the execution times decreases reaching the minimum approaching more tight area constraints (as 1.5:1), but rising again for the minimum bounding box of 1:1, as it is shown in Figure 5.14 where the execution times obtained for the b12 circuit are reported as an example.
The circuits obtained starting from these placement conditions have been analyzed in terms of routability by evaluating the relative routing solutions obtained with FeDRA with respect to the utilization of short segments and both horizontal and vertical long lines.
The trends observed with this evaluation show that, as expected, the amount of horizontal long lines used in the designs decreases as the area is reduced along the X-axis. At the same time, the quantity of vertical long lines rises to balance the reduction of the horizontal lines and prevent the routing of a higher amount of short segments, which need more time to be routed and introduce higher delays within the circuit. On the other hand, the amount of short segments strongly rises for the widener placement, since in a larger and scattered circuit the distance to be covered to connect the resources is longer and more segments are needed.

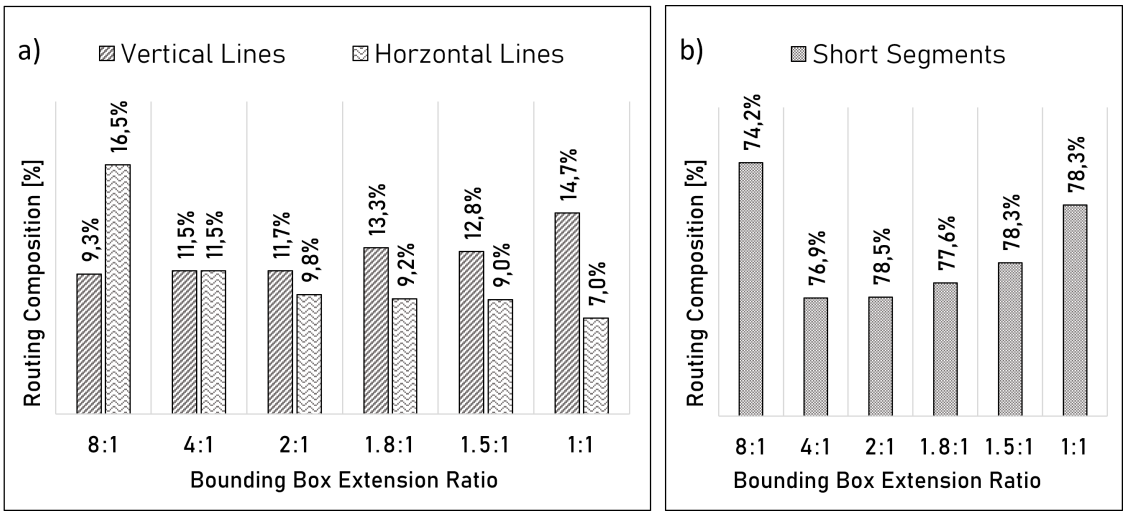

Figure 5.15: The FeDRA Algorithm Horizontal Long Lines and Vertical Long Lines (a) and Short Segments (b) Utilization Ratio for Different Area Constraints for the Benchmark b12

By reducing the of half the area, as for the 4:1 placement, the usage of short segments decreases since, in this situation, the nets are shorter because the circuit is smaller, but not enough to dramatically increase congestion. Instead, by further shrinking the placement area, the congestion begins to show its impact on routing congestion, leading to a progressive increment of short segment utilization.
Such trend is summarized in Figure 5.15 by providing the results obtained for the b12 benchmark. In detail, the compensation trends of vertical and horizontal long lines (Fig. 5.15a) and the usage of short segments (Fig. 5.15b) for the six different placement solutions are highlighted, reporting the relative percentages on the usage of these elements within the routing of the target design.
This trend is in line with the results on the execution time, as the routing of wider areas implies longer nets and more resources need to be routed, increasing the required execution time. As the area in which the circuit is mapped decreases the time required of its routing progressively reduces. However, as the congestion is pushed to the limits, the time required for routing circuits with the tightest area constraints rises since more effort is required to face congestion and the increased amount of short segments needed to fulfill such requirements.

5.3.5 Modules Reconfiguration Scenario

In dynamically and partially reconfigurable systems the partial reconfiguration is oriented to perform three main optimizations: area efficiency, power saving, and reliability. In fact, as discussed, the feature of updating at run-time the functionality of reconfigurable modules enables the possibility to time-multiplex the fabric

resources repurposing them to virtually obtain an increased area upon the one physically provided by the device or to adapt the execution to meet on the fly varying computational requirements.
Additionally, the possibility to disable an unnecessary functionality by erasing the configuration memory section programming a dynamic module in idle while re-implementing it when needed allows optimizing the power consumption of the computing system. In fact, deploying all the functionalities only when the full effort is required allows adapting and scaling the computational effort and the related power consumption to the current demand and thus enables the implementation of efficient power-saving techniques
Finally, the partial or full scrubbing of the configuration memory data is one of the most consolidated repairing techniques to increase the reliability of SRAM-based FPGA applications. In fact, the refresh of the configuration settings of critical modules, either periodically or on-demand, enables the enhancement of the reconfigurable application dependability.
Therefore, except for this last optimization goal that requires also empty frames to be refreshed, the difference-based reconfiguration can be optimally exploited for both power-saving and area efficiency techniques. In fact, when hardware tasks need to be deployed and erased to scale the system power consumption only the frames effectively programmed are involved in the procedure.
This could change when different functionalities need to be allocated within the execution on the same reconfigurable partition to time-multiplex the available resources for increasing area and computing efficiency. In detail, if blanking configurations are used the reconfiguration involves only the programmed frames for the target circuits, and the approach is conceptually similar to the programming and erasing for power saving since the old module needs to be first blanked before the new circuit is allocated. Instead, if such intermediate blanking configurations are not exploited to speed up the process, the frames involved in the reconfiguration for swapping between two different functionalities consist of the sum of the frames programmed in the two circuits, as summarized in Figure 5.16 reporting the configuration frames on the same sample Tile Column used for Cordic and b14.

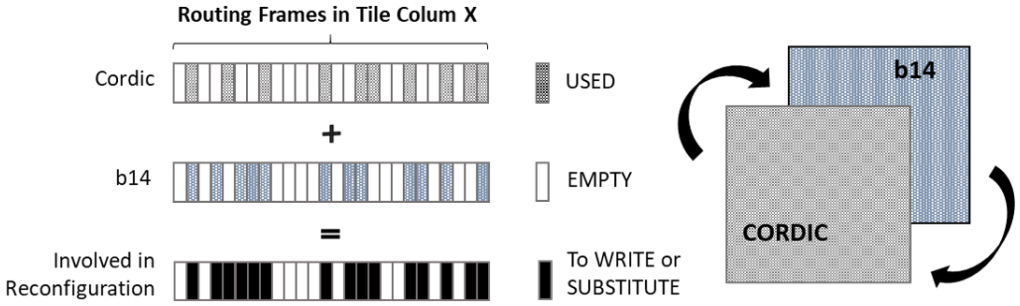

Figure 5.16: Routing Frames Overlapping in Tiles Column for Cordic and b14

In this case, only the frames empty for both the circuits are not involved in the difference-based reconfiguration. Therefore, maximizing the number of unused frames increases the chance that the unprogrammed frames for two designs correspond.
To evaluate further the FeDRA performance an additional analysis has been performed by considering the case in which on the same reconfigurable area different benchmarks must be implemented extending the study on the circuits programming and erasing to the direct swap among them.
The analysis has been performed on the same benchmark circuits by coupling them considering their sizes and constraining each pair within same the smallest possible area, and evaluating the frames involved in the reconfiguration across the same reconfigurable region comparing the solutions obtained with the vendor tool and with FeDRA. As expected, the results of this evaluation, reported in Table 5.5, show that in the situation in which the circuits are swapped the frames involved in the reconfiguration increases, due to the additional overlapping of such frames which are used in one design but unused in the other and vice-versa. However, these results confirm again that circuits routed with FeDRA require fewer configuration frames, providing an efficient optimization even in the case of the swapping of highly constrained circuits.
As shown in the last column of Table 5.5 the frame optimization ranges from more than 37% for the circuits with a reduced size to a minimum of 32.05% for the larger ones, confirming the positive and efficient scaling observed in both standard and constrained scenarios previously discussed also when module swapping is required.

Table 5.5: Comparison between Standard and FeDRA Router Frames Usage for Programming, Erasing and Swapping of Benchmark Hardware Tasks

Task & Bancmarks	***Frame Usage [#]***		***Optimization***
	Standard Router	***FeDRA Router***	
Program/Erase b05	143	87	
Program/Erase b12	170	104	
Swap b05 and b12	**210**	**131**	**37.2%**
Program/Erase Cordic	335	208	
Program/Erase b14	442	291	
Swap Cordic and b14	**466**	**307**	**34.12%**
Program/Erase miniMIPS	1,409	912	
Program/Erase b22	1,145	764	
Swap miniMIPS and b22	**1,590**	**1,049**	**34.03%**
Program/Erase b18	3,185	2,108	
Program/Erase b19	5,817	3,942	
Swap b18 and b19	**5,888**	**4,001**	**32.05%**

5.4 Research Advancement

The main contribution on dynamically and partially reconfigurable architectures performance of the proposed approach consists of an ad-hoc Frame-driven Routing Algorithm that is able to reroute target circuits minimizing their configuration time by optimizing the usage of the frames programming routing segments through a dedicated policy based on the awareness of the configuration memory link with programmable resources.
The in-depth study on the configuration memory organization and its link with the routing resources enabled the key intuition of performing optimizations at a deeper level by working on the frame layer to achieve a relevant saving of configuration data and time, without interfering with the original and optimal circuit placement and mapping, and thus without any notable impact on the circuit topology and performance.
FeDRA efficacy confirms that the awareness of the link among reconfigurable resources and configuration memory represents a powerful means to improve and optimize the performance of dynamically and partially reconfigurable applications, as the interaction between the memory and application layers represents an inherent part of their execution.
Therefore, an additional achievement of the proposed approach consists of increasing the body of knowledge about low-level and fine-grained optimization techniques for partially and dynamically reconfigurable applications while confirming their benefits.
In fact, the experimental results obtained on a various pool of benchmarks confirmed that circuits routed with FeDRA show a strong reduction of the configuration frames required for programming the FPGA routing if compared with the ones routed with vendor tool, thus minimizing the related reconfiguration time.
In detail, FeDRA enabled an average optimization of the data and time involved in reconfiguration in the range between 41% to 32% if compared to the vendor tool flow, scaling efficiently with different circuit parameters, constraints, and reconfiguration scenarios, and confirming its efficacy as the circuits size and the congestion increase as well as when applied to the dynamic deallocation, allocation, and swapping of reconfigurable modules without affecting other performance parameters.
Although an evaluation of FeDRA execution times and routability has been performed, further analysis and improvements on the algorithm performance can be investigated and implemented as future developments.
In detail, similarly to what has been performed with the area constraints, additional experiments can be performed by narrowing as well the timing constraints to evaluate the achievable optimization level and its impact on the maximum frequency of the target circuits.
This analysis could aid and be accompanied by the evaluation of different routing strategies.

In fact, studies on the impact of the net routing order on timing and optimization level can be performed and selective optimizations can be considered, such as keeping the vendor routing solution for static and critical nets.
Finally, cross-correlation approaches can be introduced when routing circuits that require to be allocated in the same area to increase the frame overlapping among them and further reduce the procedure overhead for the module swapping scenario.

Chapter 6

Fast and Distributed Reconfigurable Architecture with ReM Cells

In this chapter, a novel basic unit able to be fastly reconfigured with multiple functionalities and called Reconfigurable Multipotent (ReM) Cell, is proposed for the implementation of distributed reconfigurable architectures with enhanced flexibility and scalability for fast and real fine-grained dynamic self-reconfiguration.
In the following, the main contribution embedded in the ReM Cell is presented, as well as its detailed architecture and computing model. Then, its layout and the mechanism on which circuits can be tailored on it are discussed, providing technology mapping examples as well as details on the implementation of several benchmark circuits on the proposed architecture, followed by a comparison with their counterparts implemented on the state-of-the-art FPGA in terms of configuration bits and time [101].

6.1 Overview

The main contribution of ReM consists of providing a novel architectural model which enables reconfigurability at the finer granularity and within a single clock cycle thanks to a different configuration mechanism, which is directly embedded on the Cell itself rather than be based on host controllers that act on a separate layer. In fact, the key idea behind ReM consists of acting as an atomic reconfigurable unit that can be either configured for implementing basic logic, memory, or connectivity nodes while integrating its own Reconfiguration Engine, designed to be as simple as possible and to be able to trigger the Cell self-configuration as well as the repurposing of the neighbor Cells.
By relying on a minimal amount of configuration bits and being cleverly designed

to provide high node flexibility for the enhanced connectivity and basic functions implementation, ReM Cell arrays enable the implementation of parallel and concurrent reconfigurable circuits, maximizing the concept of scalability by relying on a distributed configuration layer and easing its integration on more complex heterogeneous fabrics.
The realization of a ReM layout and the mapping of several benchmark circuits on its architecture have confirmed the feasibility of the proposed computing approach as well as the saving of both configuration data and time, which results extremely valuable in the case of minor and frequent configuration upgrades if compared to the current FPGA architecture based on very long configuration frames, partially configuring programmable resources and thus unsuitable for the detailed and fast bit-level repurposing of dedicated computing cores.

6.2 ReM-based Architecture

The key principle which guided the design of the ReM Cell has been based on maximizing the trade-off among simplicity and flexibility for the optimization of the mechanism which allows the repurposing of its behavior according to the minimum amount of configuration bits, accessible either through the in-cell Reconfiguration Engine and from the ones of the neighbor units.
In detail, the ReM Cell has been devised as the basic building block of a two-dimensional array of tiles, which computation and configuration propagate in a systolic fashion. The amount and arrangement of ReM Cells can be specifically sized and shaped according to the needs to implement either a regular matrix following an FPGA-like island architecture or an irregular cluster to be integrated into hybrid and heterogenous ICs following the eFPGA concept.
In fact, its inner architecture has been designed as a fine-grained unit tightly coupled with its own Reconfiguration Engine and thus capable of acting both as a dedicated reconfiguration agent for itself and companion Cells or as a reconfigurable element for computation.
To enable such reconfiguration and computing paradigm the major focus has been put on the individuation of a clever trade-off between complexity and granularity coupled with a high effort on contriving the most synthetic architectural design to maximize the functional behavior flexibility while minimizing the configuration settings required for its implementation.
In the following, the ReM Cell architecture is described presenting its inner architectural details and both computing and configuration capabilities. Additionally, some examples are provided on how circuits can be mapped on ReM clusters architecture, highlighting the technology mapping approach of the proposed architecture.

6.2.1 ReM Cell Overview

In Figure 6.1 the high-level architecture of a single ReM Cell to be integrated within a wider array is provided.
As mentioned, within the ReM Cell are integrated all the features required in a reconfigurable computing system, enabling each Cell to be seen as a single and stand-alone element that can be fleixibly and scalably coupled with other identical units to implement more complex reconfigurable circuitry thanks to four directed couples of Inputs and Outputs.
In detail, each ReM Cell can be configured to implement either Logic, Memory, or pure Connectivity elements with the added possibility to manage its own configuration settings through an ad-hoc Reconfiguration Engine and its own configuration register.
For each functional behavior, different modes are possible according to the bits stored in the configuration register CNFG_REG. The content of the CNFG_REG can be dynamically updated according to the so-called Update Rules, which are stored into an additional small register RULES_REG, which consists of 3 bits and defines the way in which reconfiguration propagates among Cells or can change within the Cell itself, as will be clarified in the following together with the other Cell features.

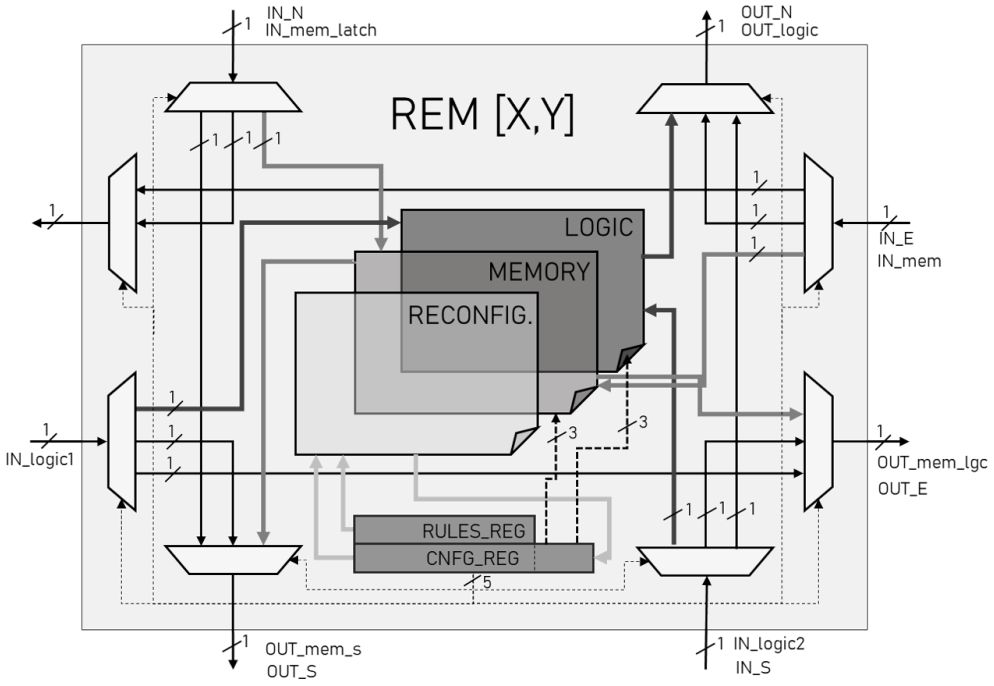

Figure 6.1: ReM Cell High-level Architecture with the Highlights of its Functional Modes and Connectivity [101]

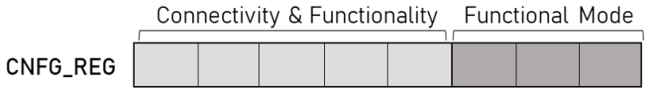

Figure 6.2: ReM Cell Configuration Register CNGF_REG Detail

ReM Configuration Register

The Configuration Register CNFG_REG, reported in Figure 6.2, holds the settings defining the ReM Cell behavior and functionality mode.
In fact, for each functional behavior different modes are possible according to values of the bits stored in the register CNFG_REG.
In detail, it consists of an 8-bits register capable of efficiently encoding all the available configurations. The value of the 5 left-most bits of this register allow to set Cell connectivity with the surrounding cells allowing at the same time to switch between Cell functionalities (e.g., from Logic to Memory), while the other 3 bits configure the functionality modes (e.g., different logic operators, sequential behaviors, or signal bypass), as summarized in Figure 6.2 and detailed in the following.

ReM as Connection Element

Differently from FPGA tiles, which have two separate sections, one for the computation and one from the interconnection, in ReM Cells connectivity is embedded and tightly coupled within the computing and sequential elements. In detail, Connectivity configuration settings define the Cells networking while implicitly activating and deactivating the configuration of inner functional blocks.
In fact, each ReM Cell has four Inputs and four Outputs, having one pair of IO on each flank. Each Input is connected to the input port of a Demultiplexer while each Output Port coincides with the output of a Multiplexer. According to the Mux and Demux selection signals, which are driven by the Cell configuration bits, the internal connectivity of the Cell can be arranged to implement pure connectivity to carry signals from or to other surrounding Cells or to drive signals through the Logic or Memory sections of the Cell itself and implicitly activating them.
In detail, as the Mux settings define which signals are active they act as switches that turn on and off the other ReM inner blocks as a side effect, while implicitly changing the configuration meaning of the other three remaining bits.
Considering such instrumentation, to reduce the complexity and keeping the interconnection structure as streamlined as possible, the connectivity reachable from each port is not exhaustive.
This means that not all the directions are reachable from every port since the most relevant have been prioritized considering the possible configuration and usage of the logic and memory elements to cleverly exploits the inner architecture and maximize the saving of the configuration bits, as will be furtherly clarified.

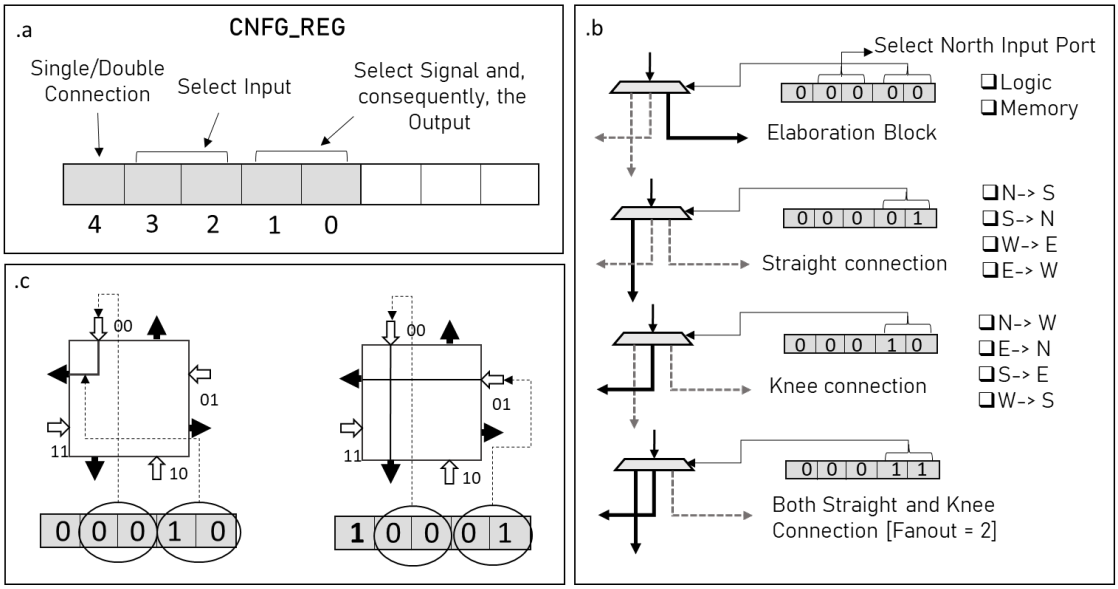

Figure 6.3: ReM Connectivity: a) Configuration Register Format; b) Signals Activation Encoding; c) Double Connection Example [101]

The configuration and connectivity capabilities of the ReM Cell are summarized in Figure 6.3. As highlighted in Fig. 6.3a, the 5 left-most bits of the CNFG_REG are used to set the configuration of the ReM Cell connectivity. In detail, the first two bits [1:0] from the right set the activation of the Demux Output (or Outputs) of the target Input port stated in two following bits [3:2].
In this way, the bits [1:0] are defining through which signal the target input would pass, as summarized in Fig. 6.3b.
In detail, the possible connections are the following:

- CNFG_REG [1:0] = **'00'** *Elaboration*: Logic or Memory, according to the Input Port stated by [3:2]
- CNFG_REG [1:0] = **'01'** *Straight*: North to South and vice versa, East to West and vice versa
- CNFG_REG [1:0] = **'10'** *Knee*: North to West, South to East, East to North and West to South
- CNFG_REG [1:0] = **'11'** *Fanout 2*: Both Knee and Straight connections are active at the same time

Finally, the leftmost bit [4] has been instrumented to increase the routing capabilities, allowing to activate at the same time two orthogonal ports of the Cell, as reported in Fig. 6.3c. In fact, when this bit is set to '1' both couples [3:2] and [1:0] are used as port indexes which are automatically activated as Straight connection.

ReM as Elaboration Element

From the computation point of view, ReM enebles two kinds of Elaboration: Logic and Memory. As mentioned, the configuration bits of the CNGF_REG setting the Elaboration behavior are 3, and they are interpreted according to the other 5 bits programming the connectivity.
In detail, when the configuration bits for the connection [1:0] are set to '00' and the ones selecting the active ports [3:2] are indicating as input port the one which provides connectivity with one of the inputs of the Logic Block (i.e., referring to Fig. 6.1 the Input port on the West Edge *IN_Logic1/IN_W* or the one on the South Edge, *IN_Logic2/IN_S*) the configuration bits devoted to the Elaboration, will be used for setting accordingly the behavior of the logic, while if the input ports connected to Memory Block (i.e., the ports on the East or North Edge, respectively *IN_mem/IN_E* or *IN_mem_latch/IN_N*) are activated the same bits would be used to set the synchronous or asynchronous behavior of the Cell.
On the other side, if the configuration bits [1:0] have a value which is different than '00', and thus are programming the Cell in pure connectivity mode, these other three bits are discarded and not considered.
In Figure 6.4 the inner architecture of both Memory (Fig 6.4.a) and Logic (Fig 6.4.b) are reported while the relative configuration bit encoding for the two configuration is reported in Table 6.1.

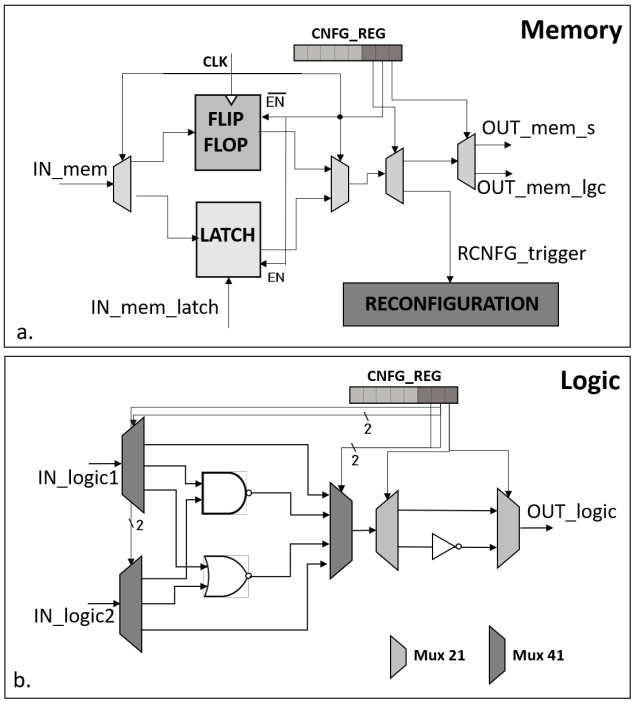

Figure 6.4: ReM Elaboration: a) Memory Block; b) Logic Block [101]

Table 6.1: Elaboration Configuration Encoding for Memory and Logic Units

Bit Encoding	***Memory***		***Logic***		
	Output	***Behavior***	***Function***	***Mux 21***	***Mux 41***
000	South	Flip-Flop	NOT In1	0	00
001	East	Flip-Flop	AND	0	01
010	South	Latch	OR	0	10
011	East	Latch	Not In2	0	11
100	Reconfiguration	Flip-Flop	Buffer In1	1	00
101	Reconfiguration	Flip-Flop	NAND	1	01
110	Reconfiguration	Latch	NOR	1	10
111	Reconfiguration	Latch	Buffer In2	1	11

Memory: The Memory Unit is composed by an Edge Triggered Flip-Flop and a Level Sensitive Latch, which receive their relative inputs respectively from *IN_mem* or *IN_mem_latch* according to the settings of the Input Demuxes.
Based on the value stored in 3 bits of the CNFG_REG configuring the Elaboration and acting as selection signals for the inner Muxes and Demuxes, the Inputs can either be sampled by one of the two sequential elements or bypass them providing additional routability without any elaboration; then, again according to the configuration, this signal can be forwarded to one of the two available outputs, *OUT_mem_lgc* and *OUT_mem_s*, or directly as a trigger for the Reconfiguration Engine, as summarized in Figure 6.4.a and Table 6.1.
This last configuration is the one enabling the Cell self-reconfiguration as well as the trigger for other Cells dynamic repurposing within the cluster, which can be performed either synchronously or asynchronously according to the needs.

Logic: the Logic Unit instead can receive up to two inputs, *IN_Logic1* and *IN_Logic2* , and according to the coding of the bits programming the Elaboration produces one output *OUT_Logic*, thus elaborating at the bit-level. In detail, the Logic Unit contains one NAND and one NOR gate placed in parallel as well as a NOT gate on their outputs.
Again, the logical path followed by the input signals depends on the configuration bits acting as selection signals for the internal Muxes and Demuxes enabling eight different functionalities, such as two Inputs AND, NAND, OR, and NOR, as well as NOT Input1 or NOT Input2 and Buffer Input1 or Buffer Input2 as summarized in Figure 6.4.b and Table 6.1.
Again the buffering options are available to increase the routability and add further connection possibility among Cells.

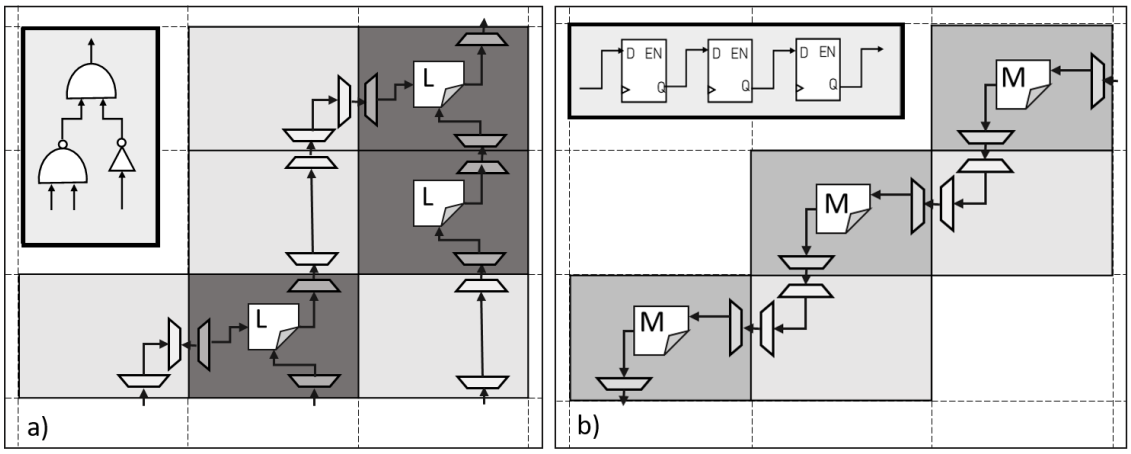

Figure 6.5: 3-Inputs Function (a) and Flip-Flop Chain (b) Implemented Programming respectively a Cluster 7 and 5 ReM Cell within a Cells Matrix

As a final remark on interconnection topology and Elaboration Blocks instrumentation, the connectivity organization of both Memory and Logic Blocks and their integration with the possible configurations of the connections have been designed to aid the efficient implementation of both logic and sequential circuits.
Two examples are shown in Figure 6.5, where is highlighted the optimized connectivity between Logic and Memory nodes with respect to their functional topology within a matrix of ReM Cells reporting a simple 3-inputs function and a chain of 3 Flip-flops. In detail, the location of both Memory and Logic inputs and outputs coupled with the selective connectivity implementation ease the cascade of multiple logic gates (Fig. 6.5a) or FFs enabling the realization of chains (Fig. 6.5b) and aiding the efficient insertion of sequential elements in logic functions thanks to the dedicated Memory output port on the left edge.

ReM Reconfiguration

The main feature of the ReM Cell consists of integrating its own Reconfiguration Engine which is capable to repurpose at run-time the functionality of the Cell itself as well as triggering the reconfiguration of their neighbors within the cluster. In this way, ReM-based reconfigurable architectures present a distributed configuration memory and management rather than a parallel and monolithic layer which access is possible only through configuration controllers or agents acting on a higher level. In this way, fine-grained and detailed reconfigurations are possible at the bit-level without recurring to complex and time-consuming infrastructures requiring an unreasonable amount of data exchange for performing minor changes within the design.
Although two kinds of Reconfiguration have been foreseen for ReM array, namely Defined and Undefined, and the Cell Architecture has been envisioned to support

both of them, in the current version only the first one is implemented. In detail, in the Defined Dynamic Reconfiguration, the possible Cell configurations for a given application are known and identified a priori, while the Undefined one will enable the possibility for the Cell to repurpose itself with configurations that could be determined at run-time according to unknown in-field requirements.
The inner architecture of the Reconfiguration engine is reported in Figure 6.6 highlighting its interactions with the Computation Blocks of the Cell as well with the Reconfiguration units of the Cells on its sides.
In detail, the Reconfiguration Engine enables the interaction with both the Left and Right Cells thanks to four additional signals. As two of them are output signals that when activated act as a trigger for the Reconfiguration Engines of the side Cells, at the same time the Cell itself can receive a reconfiguration trigger from them through the complementary signals it receives as inputs.
Additionally, as mentioned while describing the Memory unit, a trigger Input can be received from the Elaboration engine and consisting of an internal signal which can be raised when the Cell Memory block is active and it is configured in Reconfiguration Mode.
These five additional signals are unrelated and independent from nominal Cell Inputs and Outputs for the connectivity with the other Cells since reconfiguration should be possible at any moment and regardless of the current state programmed on the Cell. Besides its connectivity, the Reconfiguration Engine consists of 8 Circular Buffers and a Finite State Machine (RFSM).

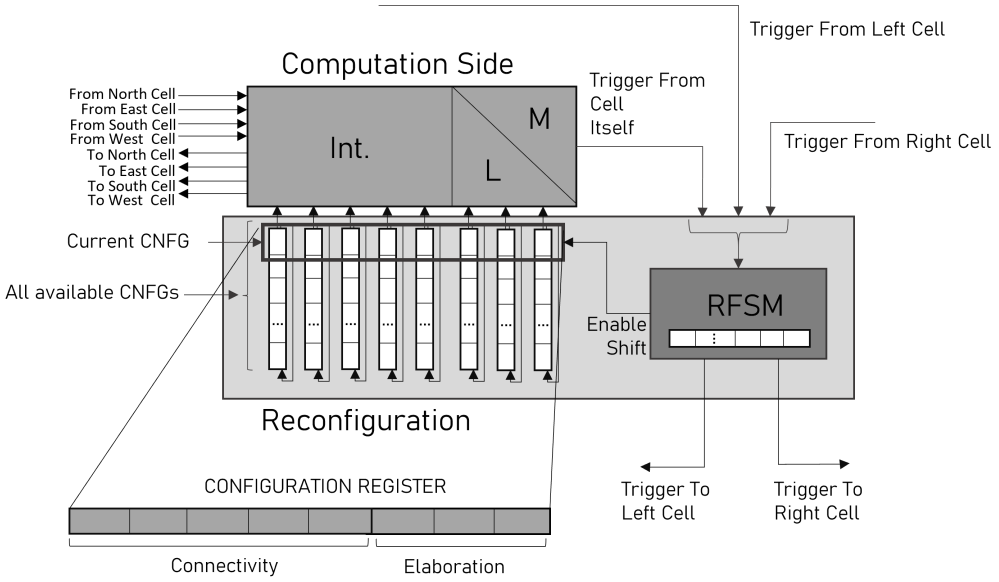

Figure 6.6: Overview of ReM Reconfiguration Engine: Mechanism and Interaction with Neighbor Cells [101]

In the Defined Reconfiguration, the Circular Buffers content is written before deployment with the set of all the foreseen configurations that the Cells needs to implement within the operation of the target circuit, and the most significant bits of such buffers consist of the initial configuration state of the Cell, as they coincide with the Configuration Register CNFG_REG.
On the other side, the Reconfiguration Finite State Machine is in charge of managing the shifts of Circular Buffers to update the configuration of the Cell it belongs to according to the received triggers as well as sending reconfiguration triggers to the neighbor Cells.
In detail, the actions performed by the RFSM are based on the so-called *Update Rules*, which are identified and written before the Cells array start-up and consist of additional programming bits stored in the inner RULE_REG defining the rules on which the triggers are processed for shifting the CNFG_REG as well as how triggers are forwarded to the side Cells.
To provide an example, the *Update Rules* of a given ReM Cell can state that as an input trigger is received from the Left Cell, no updates should be performed in the inner configuration CNFG_REG while an output trigger should be forwarded to the Right Cell; again as an example, the Reconfiguration Engine of such Right Cell, according to its relative *Update Rules*, as it receives a trigger from Left should update its inner CNFG_REG by shifting the Buffers of one or two positions and forward an output trigger to both its Right and Left Cells or stopping the reconfiguration triggers forwarding.
While for the Defined Reconfiguration 3 bits for storing the *Update Rules* have been devised for setting all the possible scenarios when Undefined Reconfiguration is considered the width of the RULE_REG would be increased as well as the complexity of the RFSM. In fact, in this case, the Circular Buffer Array should allow random access both in reading and writing as its content should be updated and modified at run-time either than being statically written only once at the start-up and the set of rules necessary for controlling such operations would increases.

6.2.2 Circuit Mapping

ReM fine-grained Cell and connectivity organization enable the implementation of any digital circuit, considering the possibility to rely on all the basic computing and connection elements.
The additional bit-level repurposing capability allows to exploit small clusters of Cells which can instantaneously change their behavior reshaping connections and basic functional units in a systolic fashion.
Furthermore, thanks to their modularity, ReM Cells be either used to build regular matrices of tiles arranged in an standard Island-based FPGA fashion, or integrated as ad-hoc clusters on more complex ICs, providing the precise amount of

fine-grained reconfigurable fabric expressively oriented to bit-level dynamic computations pursuing the concept of eFPGAs.
As each ReM Cell can be either used for Computation or Connection, a novelty of ReM design consists on the fact that Routing and Placement can be processed in the same way during the circuit mapping, since "Routing" two "Placed" ReM Cells implementing logic nodes consist of "Placing" accordingly ReM Cells which would be programmed in the required connection mode.
Thus, the implementation flow of both Dynamic and Static circuitry consists of identifying the number and the shape of the Cell cluster while setting the configuration of their Circular Buffers and Update Rules according to requirements in terms of computing nodes, connections, and reconfiguration triggers.
In the following, two examples of circuit mapping on ReM Cells are provided, one describing the implementation of a 6-input reconfigurable logic function, the other of a N-bits Full Adders, to highlight all the computing modes of ReM Cells.

6-Input Reconfigurable Function

To highlights the Memory and Reconfiguration mechanism for dynamically reconfigurable circuits on ReM Cells the example of the implementation of a 6-bit reconfigurable function that switches its behavior every 4 Clock Cycles is reported in Fig. 6.7.
The dark-grey part of the design on the right has to be considered as static (i.e., is not involved in reconfiguration) and represents a 4 bits counter that triggers the function reconfiguration through the usage of 4 ReM Cells which Memory Unit in Synchronous Reconfiguration Mode is active.

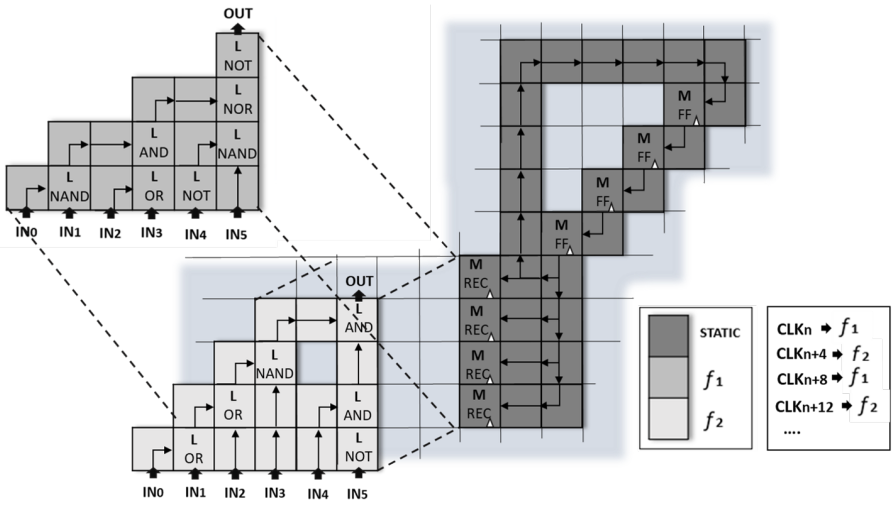

Figure 6.7: 6-Inputs Dynamically Reconfigurable Function Implementation within a ReM Cells Cluster

On the left side are reported two different configuration relatives to the two 6 inputs functions, f_1 and f_2. In this scenario, it is possible to observe that the reconfiguration is performed row-wise within a single Clock Cycle, as each one of the ReM Cells implementing the dynamic function receives the trigger from the one on the right and changes its functionality updating the CNFG_REG while forwarding its own trigger the Cell on the left, until the last active one on the row that stops the triggers propagation.
In other words, according to the Update Rules stored at the beginning in the RFSM inside the Reconfiguration Engine, all the Cells instantaneously forward the trigger from right to left in a systolic manner while updating their configuration based on their inner rules.

N-bit Full Adder

In Figure 6.8 the mechanism for implementing N-bits Full Adders using ReM Cell is shown.

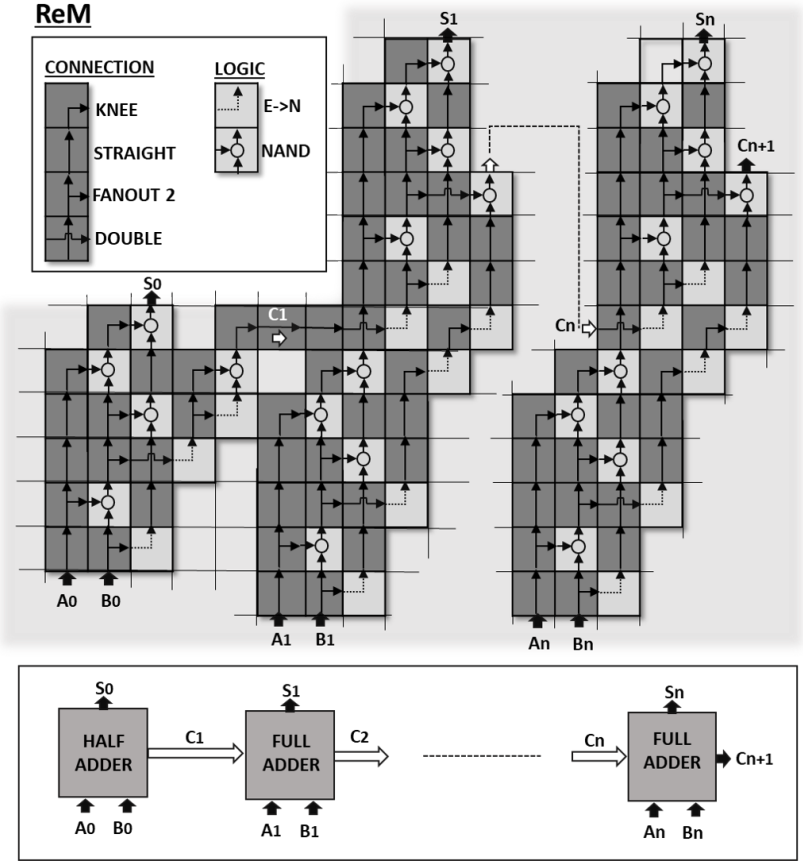

Figure 6.8: N-bits Full-Adder Implementation within a ReM Cells Cluster

They are composed of one Half-Adder followed by the required amount Full-Adder. From this example is possible to notice the high regularity provided by ReM Cells, as well as many of their Connection and Logic Modes.
In detail, many ReM Cells are used in Logic Modes either configured as *2 Inputs NAND Gates* and *Buffer Input 1*, for providing additional connectivity from West to North, while the others are used in Connection Modes, providing examples of all the possible configurations, such as *Straight, Knee, Fanout 2*, and *Double* crossed Connections.

6.3 Circuit Implementation with ReM

Additional steps and analysis have been performed to demonstrate the feasibility and benefits provided by tailoring circuits on ReM-based architectures.
The design of the ReM Cell has been firstly developed enabling its implementation by performing its placement and routing and obtaining its layout.
Then the mapping of several ITC '99 benchmarks on ReM Cells has been performed to obtain the number and configuration modes of the Cells required for their implementation and highlighting the gain achieved in terms of reconfiguration time and data if compared with their implementation on the state-of-the-art FPGA

6.3.1 ReM Layout

The design of the ReM Cell has been realized in VHDL to confirm its feasibility and to evaluate its implementation metrics while providing a reference point for evaluating its characteristics with respect to the FPGA technology.
As no information can be obtained on the target Xilinx FPGA configuration layer area and a fair comparison with the reconfiguration interface and controller used in such context cannot be performed, the Reconfiguration Engine of the ReM Cell has been excluded from the evaluation.
In detail, the ReM Cell VHDL model has been firstly synthesized relying on the Synopsys Design Compiler tool to obtain its Netlist consisting of the logic gates and their interconnectivity.
For synthesizing the ReM Cell has been used the whole Nangate Open Cell Library, which is an open source Standard Cell Library based with a 45 nm technology including a large set of logic gates.
The ReM Cell area has been firstly estimated on the so obtained Netlist that consists of 251 equivalent NAND Gates and considering the target library technology requires a logic area of 200.298 μm^2.
Furthermore, relying on such Netlist the ReM Layout has been produced to obtain a further characterization of the chip area and to confirm the feasibility and routability of the Cell.

Table 6.2: Placement and Routing Resources for ReM Cell Layout

Layout Resources	*Count*
Standard Cells	190
Nets	131
Segments	274
Routing Metal Layers	12

In detail, the Cell Layout has been obtained with the PDD_Place&Route tool, an in-house software realized in Politecnico di Torino [105], using the nominal effort for the elaboration and thus without inserting additional constraints and optimization options.
The so obtained ReM Cell Layout consists of 190 Standard Cells and its details in terms of routing segments and metal layers are provided in Table 6.2.
Thus the required area for the Cell Layout, which consists of a Standard Cells array organized in 13 x 16 matrix as reported in Fig. 6.9, is equal to 207.063 μm^2. This value results consistent with the one obtained through Design Compiler estimation in terms of NAND gates as the slight difference is related to the inclusion of the routing segments that consume an additional amount of space.

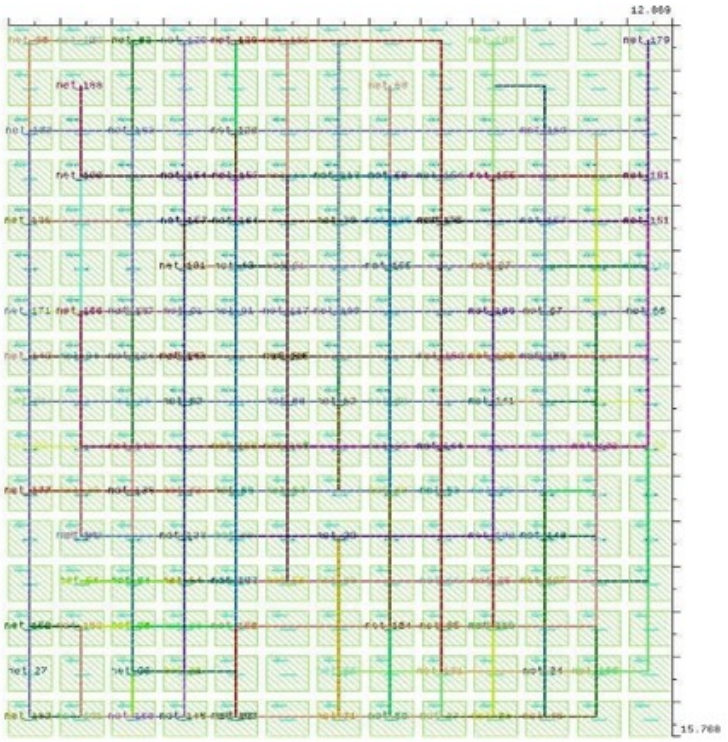

Figure 6.9: ReM Cell Layout of the Placed and Routed Standard Cells

6.3.2 ITC'99 Benchmarks ReM Implementation

To further confirm the feasibility in terms of computation and implementation of digital circuits on ReM-based architecture a pool of circuits has been selected from the ITC'99 benchmarks to be mapped in ReM Clusters for evaluating the number of Cells required for their implementation as well as the relative amount of consumed area. In detail, the first six circuits of the ITC'99 set have been used, which characteristics in terms of used combinational gates, filp-flops, and functionality are reported in Table 6.3 to highlight their different sizes and computing capabilities [102].
As a complete toolchain for the automatic mapping of designs on ReM Cells is not yet available, a mixed approach based on Design Compiler and support scripts has been instrumented for obtaining the benchmark mapping results on ReM-based Architecture. In detail, the Logic and Memory nodes required for the circuit implementation have been extracted from Design Compiler, which settings have been customized in order to synthesize the target circuits only with the subset of the gates within the Nangate Open Library that corresponds to the one which can be configured on ReM Cells. More specifically, the gates usage among the library has been restricted to the AND2_X1, NAND2_X1, OR2_X1, NOR2_X1, and INV_X1 that correspond to the available Logic modes of Cell and including the available memory elements for the implementation of the Memory modes presented in Table 6.1.
Once the computing nodes of the circuits have been obtained, this information has been processed by a custom script that individuates the number and the configurations of ReM Cells in Connectivity modes required for mapping the circuits interconnections.
This has been possible by interpolating the details extracted from the benchmarks Netlists obtained with Design Compiler on circuits nodes and net arcs with the encoding of all the possible connectivity modes achievable with ReM Cells, either in pure Connection mode or through the bypass features available in the Elaboration Blocks.

Table 6.3: ITC '99 Benchmarks Characteristics: Combinational Gates, Filp-Flops and Functionality [102]

ITC'99	***Gates [#]***	***Flip-Flops [#]***	***Functionality***
b01	46	5	*FSM for Serial Flow Comparison*
b02	20	4	*FSM Binary Coded Numbers Recognition*
b03	149	30	*Resource Arbiter*
b04	597	66	*Minimum and Maximum Computation*
b05	935	34	*Memory Content Elaboration*
b06	60	9	*Interrupt Handler*

Table 6.4: Benchmark Implementation with ReM Cells: Type, Count, Configuration Bits and Total Area

ITC '99	*ReM Cells [#]*				*Configuration*	*Area*
	Logic	*Memory*	*Connection*	*Total*	*Bits [#]*	*[μm^2]*
b01	38	5	52	**95**	757	19,588.14
b02	23	4	35	**62**	497	12,858.60
b03	139	30	203	**372**	2,974	76,985.96
b04	416	66	578	**1,060**	8,483	219,569.45
b05	575	34	731	**1,340**	10,718	277,422.81
b06	57	9	86	**152**	1,214	31,432.14

The details of the obtained ReM implementations of the benchmark circuits are reported in Table 6.4 highlighting the amount and the mode of the required ReM Cell as well as the involved configuration bits and required chip area.
It is possible to observe that the Cells in Connection modes represent on average slightly more than 50% of the total. Within this amount must be considered also the contribution of Cells in Memory and Logic modes that are used either as bypass or signal buffers to implement further directional connectivity. Besides, further optimizations and analysis on this ratio could be performed through the development of a dedicated toolchain for ReM Cells mapping.
As expected, the in-depth Cell configurability pays a price in terms of area, as it is in the order of thousands of μm^2 for all the benchmark implementations. In detail, more than two-thirds of this area is consumed by the inner control switches and multiplexing while the remaining part consists of combinational logic gates.
However, it must be considered that ReM Cell area has been obtained relying on a cell library based on a 45 nm technology and further improvements can be achieved through the usage of different gate libraries based on smaller technologies as well as through the introduction of additional layout optimizations.

6.3.3 Comparison with FPGA Implementation

As a further evaluation, the six benchmark circuits mapped with the ReM Cells have been implemented on Xilinx 7 Series architecture relying on Vivado Toolchain and using as target device the ZYNQ 7020 to make a comparison about resource usage, configuration bits, and time required to dynamically program the circuits on the two implementations.
In detail, the FPGA usage consists of the basic reconfigurable resources required to deploy the target circuits on the fabric in terms Look-Up-Tables, Flip-Flops, and Programmable Interconnect Points, while the configuration bits consist of all the bits belonging to the frames required to program such resources.

These FPGA resource usages are reported on the left side of Table 6.5, while on the right side the configuration bits and time involved in the circuits reconfiguration are reported for both ZYNQ and ReM Implementation.
From this last comparison is possible to see that the amount of configuration data involved in ReM programming is considerably lower than the one required for the FPGA programming.
In fact, the distributed architecture and configuration memory of ReM Cells allow defining the circuit functionality at the bit-level granularity, and thus enable direct access to the single configuration registers of the Cells specifically involved in the circuit implementation.
On the other side, FPGA architecture is only virtually fine-grained since even if the configuration of a single LUT or FF is defined by a small number of bits, these bits are part of the whole 3,232-bit frame that is the smallest memory segment possibly involved in the reconfiguration procedure and holds also configuration data of unused resources which belong to the same column.
This shows the greater gain ReM provides in terms of circuit reconfiguration thanks to its distributed and fine-grained model.
In fact, every frame carries also useless information and requires a time of around 100 μs to be transferred to the ICAP and its controller. Furthermore, even considering the most synthetic difference-based approach, the frames programming a circuit are serially written one after the other into the configuration memory. Thus, the reconfiguration of circuits deployed on FPGAs requires an enormous amount of time if compared with the detailed and systolic repurposing enabled by the ReM Cell, which according to the received trigger can be performed instantaneously and in parallel within a single Clock Cycle.

Table 6.5: Benchmark Implementation on ZYNQ and Reconfiguration Data and Time Comparison with ReM-based Clusters (100 MHz has been used as Nominal Frequency for both Architectures)

ITC'99	***ZYNQ 7020***					***ReM Cluster***	
	LUT[#]	***FF[#]***	***PIP[#]***	***Conf. Bits***	***Conf. Time [ns]***	***Conf. Bits***	***Conf. Time [ns]***
b01	5	5	101	84,032	2.6×10^6	757	0.1
b02	4	4	72	71,104	2.2×10^6	497	0.1
b03	15	30	420	158,368	4.9×10^6	2,974	0.1
b04	112	66	2,140	429,856	13.3×10^6	8,483	0.1
b05	84	34	1,985	1,383,296	42.8×10^6	10,718	0.1
b06	8	9	187	145,440	4.5×10^6	1,214	0.1

Of course, the gain in terms of reconfiguration granularity and flexibility pays a price in terms of area overhead.
However, although the area required for the implementation of one ReM Cell as well as the one needed for relative benchmarks mapping has been obtained through the realization of its Netlist and Layout, a fair identification of its area overhead versus FPGA implementation is not streamlined.
In fact, besides the different technologies (i.e., the 45 nm of the Nangate Library and 7 Series 28 nm), to retrieve information about the area required by an FPGA tile considering the contributions of both the computing CLB side and connection Switch Matrix to make a fair comparison with the ReM basic functional unit is not trivial.
Furthermore, within a given FPGA or device sub-area used to map a circuit not all the tiles are programmed and the active ones are interleaved and surrounded by unprogrammed resources. On the other hand, the ReM architecture has been contrived to enable Cells integration either according to a regular and Island-Style topology or as specifically sized and arranged clusters to be embedded in more complex heterogeneous fabrics.
Especially considering the latter possibility, the area required for small and medium-sized components that won't fill an entire FPGA but need fast and detailed reconfiguration could be less penalizing than a monolithic commercial approach maximizing the gain achieved through the ReM Cells fine-grained and distributed reconfigurability.

6.4 Research Advancement

The main contribution of the proposed ReM Cell consists of representing a novel architectural solution for those dynamically and partially reconfigurable applications that would take great advantages from fast, detailed, and frequent in-field upgrades, overcoming the limitations of the current FPGA architecture, which implies unjustified overheads related to the frame length, its mapping with single resources, and the reconfiguration mechanism instrumentation required for minor in-field updates.
In particular, the design of the ReM Cell has been driven by an in-depth analysis of current commercial architecture limitations and by the exploration of different trade-offs between granularity and configurability to maximize the concept of modularity, distribution, and parallelism.
Therefore, the ReM Cell architecture has been proposed as the basic unit for a novel reconfigurable architectures able to provide fine-grained and fast online repurposing of its resources at the bit-level, which is not efficiently implemented on the state-of-the-art FPGA architecture, highlighting the benefits of implementing reconfigurability in a distributed manner and at a lower granularity.

In detail, the proposed architecture enables its basic reconfigurable elements to reconfigure themselves and their surrounding units thanks to the introduction on the ReM Cell of its own simple and dedicated Reconfiguration Engine. In fact, the enhanced flexibility and fast repurposing have been achieved by providing to the ReM Cell the possibility to behave either as Logic, Memory, and Connectivity elements with the additional capability to perform in-field self-upgrades, by cleverly instrumenting its inner architecture and configuration mechanism for using the minimum amount of bits, and maximizing the concept of modularity and scalability.
To confirm the benefits and the feasibility of the proposed reconfiguration model, a VHDL prototype of the ReM Cell has been realized allowing the development of its placed and routed layout.
The implementation of several benchmark circuits on ReM Cell arrays has been performed and evaluated. This analysis highlights a high gain over the state-of-the-art Xilinx FPGAs in terms of configuration data and time involved in the reconfiguration procedure. In fact, ReM Cells allow performing circuit upgrades using an amount of configuration data two order of magnitude lower and within a single clock cycle thanks to their efficient configuration coding and to the distributed and fine-grained reconfiguration approach based on the systolic propagation of reconfiguration triggers among them.
As expected, the enhanced flexibility provided by distributed and fine-grained reconfigurability pays a price in terms of area overhead.
As discussed, it is not streamlined to perform detailed comparisons among FPGAs Tiles and ReM Cells in terms of the required area. In fact, in addition to the fact that the ReM Cell layout has been obtained with a library based on a gate size which is not competitive versus the state-of-the-art technology, the current limitations in performing a fair comparison among the two solutions are mainly due to the lack of information about the inner composition and layout of commercial devices coupled with the different architectural, mapping, and reconfiguration approaches. However, further analysis can be performed in the future by retrieving more precise information about Xilinx fabric to identify effective comparison metrics.
Besides, additional ReM Cell layout solutions can be investigated and realized through the usage of different standard cell libraries with reduced size and by exploring different optimizations strategies to improve the area efficiency.
Additionally, further optimizations on the inner Cell architecture as well as on the fabric heterogeneity can be evaluated to enhance the achieved trade-off between reconfiguration granularity and resources usage and extend the advantages of the proposed approach to larger and more complex circuits.
For example, different levels of granularity can be further explored to reduce the ratio among functional elements and circuitry devoted to their configuration within a single Cell, or ancillary Connection and Elaboration blocks can be designed and integrated within standard ReM Cells arrays to efficiently support more demanding routability, computational, and timing requirements.

Besides the opportunities to further investigate distributed and fine-grained reconfigurable architectures, other foreseen developments would consist of the consolidation of a complete toolchain for aiding the circuit mapping on ReM-based architectures, enabling the implementation of bigger circuits and supporting the aforementioned analyses and optimizations, possibly leading to the development of a first optimized silicon prototype of ReM Cell Arrays.

Part II

Dynamic Reconfigurable Architecture Dependability

Chapter 7

Related Works & Background

In this chapter, the related works and background about the opportunities and challenges involved in the deployment of Dynamically Reconfigurable SRAM-based FPGAs in radiation environments are provided to contextualize the main contributions of the proposed dependability analysis.
In detail, an overview on FPGA-based Reconfigurable applications in radiation environments is provided followed by a discussion on the most popular techniques used to characterize and validate applications dependability versus the target radiation environment, highlighting their benefits and limitations.
Subsequently, a background on the soft-errors sensitivity of the SRAM configuration memory in relation with the characteristics of the deployment environment is provided, presenting the advantages related to the possibility to access at run-time the configuration memory data for both detection and recovery techniques. Finally, the self-configuration controller, which is the key component enabling such features, is detailed to introduce the proposed dependability analysis on self-reconfigurable FPGAs.

7.1 Related Works

As discussed in Sections 2.2.1 and 2.2.2, the deployment of Dynamically Reconfigurable FPGA-based platforms on mission-critical and high-dependability applications in radiation environments has proven its efficacy and benefits [16][15].
This is still a growing trend, which has been corroborated by the possibility to include in such high-performance and computationally intensive systems mitigation strategies based on both fault-tolerance and fault-repairing techniques, which broadened the usage of the cheaper and more available Commercial-of-the-Shelf (COTS) SRAM-based FPGAs rather than their more expensive space-grade counterparts [14].
Although the promising benefits of performing in-field adjustment, testing, and

repairs provided by SRAM-based Dynamically Reconfigurable FPGAs, their configuration memory sensitivity as well as the relative application failure modes for different radiation doses and profiles must be properly evaluated and characterized to enable the implementation of efficient hardening and recovery techniques and to prevent mission failures [28][29][30].

7.1.1 High-dependability Reconfigurable Systems

Generally, SRAM FPGA-based modular reconfigurable platforms offer multiple benefits for computationally intensive applications thanks to their in-field upgradability that enables the possibility to tune both computational effort and power consumption according to the application instantaneous requirements as well as adapting their functionalities to varying conditions increasing the efficiency and saving area [8][9].
These features become even more valuable for those applications deployed in harsh radiation environments, such as aerospace and energy physics experiments fields, in which the demand for the area and power efficiency strongly increases as well as the reliability requirements related to mission criticality of such applications.
In fact, another key feature enabled by SRAM-based reconfigurable FPGAs consists of the possibility to introduce additional fault mitigation and recovery techniques based on Dynamic and Partial Reconfiguration [14].
In detail, the possibility to modify at run-time the SRAM memory content coupled with the introduction of self-monitoring and diagnosis capability allows extending the application lifetime either by moving computing modules from permanently damaged areas of the devices or to correct at run-time transient errors in the configuration memory by refreshing the original golden values in the target region, as proposed in [10], [63], and [64].
These possibilities, coupled with the additional opportunities provided by FPGA fabric for the implementation of structural hardening techniques, either based on high-level redundancy, as proposed in [32], or acting at the resources level within the design Place&Route as proposed in [86], have motivated the study, development, and deployment of several reconfigurable FPGA-based systems for mission-critical applications operating in radiation environments [17].
Several Dynamically Reconfigurable Platforms based on COTS SRAM-based FPGAs have been realized through the collaboration among Academic, Research, and Governmental Institutions demonstrating the advantage provided by such architectures in terms of efficiency and dependability, as discussed in Section 2.2.2.
In detail, the DRPM proposed in [65] is based on Virtex-4 FPGA clusters and thanks to its in-field upgradability confirmed its efficacy in dynamically adapt to varying satellite computational requirements providing increased reliability versus radiation-induced soft-errors by combining blind and readback scrubbing with a dedicated hardening-driven placement strategy.

The platform proposed in [66], which is oriented to vision-based space navigation and deployed on a single Xilinx Ultrascale+ MPSoC, exploits partial reconfiguration to speed up the most critical computing task while providing increased dependability by implementing both structural and temporal redundancy on the on-chip microprocessor while performing frames scrubbing through the ICAP.
Other remarkable examples of FPGA-based Reconfigurable Platforms successfully deployed in radiation environments are the CFE satellite, in which the in-field upgrades performed every two weeks to update system functionalities and mitigation techniques have extended the mission duration in the LEO orbit from 4 to 8 years [68], and the Reconfigurable Readout Control Unit integrated into the electronics for the high-energy physics experiments, based on a Virtex II in which partial run-time reconfiguration was used to transparently detect and correct soft-errors caused by high-energy particles [69].
However, to guarantee the safe operations of these applications a preliminary and mandatory step consists of performing accurate radiation analysis to confirm their dependability and readiness in performing their mission in relation to the deployment environment.

7.1.2 Dependability Characterization Techniques

As discussed, to optimally exploit the in-field adjustment, testing, and repair features of SRAM-based Dynamically Reconfigurable FPGAs for high-dependability applications operating in presence of radiation, it is crucial to evaluate the sensitivity and failure modes of the target device and application according to the deployment environment.
In fact, the accurate characterization of the system vulnerability to different radiation doses and profiles is fundamental for the integration and validation of the proper hardening, mitigation, and recovery techniques and, above all, to avoid critical application failures during its operation.
For this purpose, three main techniques exist and consist of accelerated radiation ground testing, fault injection campaigns, and analytical evaluations, which according to the selected strategy could be exploited either separately or in synergy. In fact, each one of them has its own advantages and challenges, as discussed in the following.

Accelerated Radiation Testing

Among the dependability characterizations, accelerated radiation testing represents the most accurate solution to obtain experimental data according to the deployment environment.
It consists of exposing the device to a radiation source able to mimic the target

environment according to the required particles it can generate to obtain information about the target application failure rate and modes versus the radiation effect under evaluation.
The evaluations of Single Event Effects, as SEUs, in SRAM-based FPGAs consist of event-based tests, where the number of events observed during the experiment in relation with the quantity of radiation dose received is correlated with the one estimated for the deployment environment defining the error probability for the device and the application under analysis providing useful information about their readiness for the mission as well as on the needs or the efficacy of additional mitigation techniques [30].
Many approaches and techniques are possible to perform radiation testing depending on the target application and test goals. In fact, from the point of view of the target device, one important parameter consists of its functioning state during the experiment, which accordingly can be classified either as static, semi-static, or dynamic.
In detail, when tests are performed statically, the system is not active, like in some approaches dedicated to memories in which they are written before the experiment and read afterwards easing the testing procedure but providing incomplete results on the dynamic behaviour of the component.
In both semi-static and dynamic tests instead, some parts of the component are active during irradiation allowing to observe the dynamic behaviour of the system and maximizing the irradiation time. A possible semi-static approach for memories consists of reading their content and correcting their values when a mismatch is found to avoid the accumulation of upsets and to better distinguish between single bit upsets and multiples bits upsets. In this case, the active parts of the component and the interactions with the dynamic monitoring should be carefully weighed to avoid interference with the scope of the analysis.
Considering instead the required environmental specifications and the source availability, different ionizing particles can be used to irradiate the device.
The most common for SEE FPGA testing are heavy ions and neutrons. As the reference guidelines for heavy ion testing can be found in JESD57 Standard and the ones for neutron testing are provided in the JEDEC89, many SEE radiation testing experimental analyses have been conducted on Xilinx SRAM-based FPGAs, like the one performed with neutron beams on Spartan-3 discussed in [106] and on Virtex-5 (presented in [107]) as well as the one performed with heavy ions [108] and ultra high energy heavy ion [109] both on Kintex-7 device.
When heavy ions are not required or available, proton testing can be exploited, as performed in the radiation test presented in [110].
In general, neutron testing is extremely efficient for upsets evaluation. In fact, neutrons result highly penetrating due to the absence of charge and interact with the semiconductors through indirect ionization by emitting gamma rays when absorbed. Furthermore, especially for SEE testing, neutrons are extremely valuable

and preferable to protons, which could lead to an underestimation of the upset error rate and to an increased device exposure to other unnecessary permanent damaging effects as TID.
However, although the possibility to mimic the device deployment radiation environment through the acceleration of the specific particles that would be present during its mission represents the main advantage of radiation tests, the availability of these sources is extremely low and costly.
In fact, focused and large yield beams of the target particles to irradiate the device are obtained by accelerating nuclear reactions. Since the facilities that can host such instrumentation are extremely few and the demand for their utilization is higher than their availability, the opportunities to perform radiation testing are rare and highly expensive.

Fault-Injection Campaigns

Especially when considering SRAM-based FPGA configuration memory upset, Fault Injection is an extremely valuable alternative to radiation testing for defining the application error rate.
In fact, as radiation-induced SEUs cause an inversion of the value stored inside a memory element, also called bitflip, it is possible to emulate this effect by loading inside the FPGA corrupted configuration data. In detail, fault-injection campaigns allow evaluating the system reliability and failures mode by iteratively loading in the sample device corrupted versions of the bitstreams implementing the target application in which the value of one or more bits have been inverted and to observe for each one of them the effect on the application output with respect to the golden expected value.
Typically, to obtain high fault coverage in such campaigns the number of faulty bitstreams evaluated during one experiment is maximized and the position for each injection can be either selected randomly to emulate the radiation environments or on dedicated locations to observe and to model specific failure modes of target component or resources.
Furthermore, the number of corrupted bits for each injection can be tuned according to the target experiment and the bits cluster size, shapes, and position can be defined according to pre-existent radiation test on the same target device, as performed in [111] and as it will be explained when discussing [112].
In fact, in addition to the advantage of avoiding permanent damages to the device under test which can be produced when performing radiation testing, by controlling the whole injection process at the low level, it is possible to address specific configuration resources and to link configuration upsets with the observed effect and vice-versa or to accurately reproduce radiation profiles based on experimental data obtained through radiation testing .
In fact, to achieve and maximize the controllability of this kind of fault-injection

experiment it is possible to exploit the awareness of the link among configuration data and application layer by producing or decoding faulty bitstreams through ad-hoc fault injectors and decoding tools that enable the observability of the location of the injection, as the one proposed in [99] and [113].
As a drawback, the time required to perform accurate injection campaigns can be extremely long as for each injection the faulty data needs to be loaded inside the device, the application must be executed, and the output logs need to be collected. In fact, considering as an example the configuration memory of the Zynq 7020, the total number of configuration data consists of 10,008 frames 3,232 bits long, for a total of 32,345,856 possible locations for the injection.
Thus, according to the size of a configuration memory, the time for massive or pseudo-exhaustive evaluations becomes huge, as the number of injection locations and their possible combinations reach the order of millions.
In fact, the typical approach to perform fault injection is the static one, which consists of generating a priori as many faulty bitstreams as the number of upsets to be simulated, loading the corrupted bitstreams at the start-up from external configuration ports, and comparing the results with the golden ones. This approach pays its easier implementability at the cost of execution time, as for each injection the whole bitstream needs to be loaded and the application needs to be booted.
Other approaches have been suggested to speed up this procedure relying on dynamic fault-injection, which consits of updating the configuration memory content at run-time through internal configuration access ports by implementing the fault-injector inside the on-chip microprocessor or in the reconfigurable logic, as proposed in [107] and [114].
In fact, it is possible to rely on an internal mechanism to readback a target frame through the ICAP or PCAP, flip one bit in one of its words, and write it back. However, although the injection time is shorter, more complexity is introduced in the test setup and the evaluation of the results as in this case the injection happens at run-time and is tightly integrated within the application execution.

Analytical Methods

The Analytical Methods represent an efficient approach to estimate devices and designs reliability without reverting to expensive and not easily accessible radiation facilities or avoiding the long time required by fault-injection campaigns.
These approaches consist of software programs and tools able to provide accurate estimations of application vulnerability by exploiting pre-obtained models of the system or technology under test to simulate the target radiation effect and environment relying on probabilistic and analytical methods. Among the most remarkable there are platforms as FASTRAD [115] and GEANT4 [116], which enable low-level radiation analysis on the target technologies and systems.
As these tools are oriented more generally to the semiconductor technology level

and with a broadened set of phenomena, many approaches have been developed targeting more specifically FPGAs and their sensitivity to transient errors as SEUs.
The approach proposed in [117] expressly addresses SEUs in SRAM-based FPGAs by analytically evaluating error probability on computing nodes and their propagation through the design considering its topology for the investigation of different error models.
More accurate analysis can be obtained through the VERI-Place framework proposed in [118] thanks to more detailed modeling of the design topology and layout, which enable the characterization of SEUs effects in both user and configuration memories and provide more detailed information about the system dependability for the imposed reliability constraints with a major focus on TMR architectures.
This framework and its extensions have confirmed to provide accurate error rate analytical estimations when compared to both fault-injection analysis [119] and radiation testing, as discussed in [120] where the detailed comparative analysis on the application error rate extracted from VERI-Place and the one achieved through radiation experiments confirms the efficacy of the proposed approaches also for Ultrascale+ devices.
Although analytical methods enable to save both times and costs in performing radiation analysis providing accurate estimations and represent an extremely valuable asset to perform preliminary analysis defining reliability requirements, to achieve further insurances on the application reliability and failure modes, and to obtain data closer to the real deployment environment additional evaluations based on radiation testing and fault injections could be required.

The Proposed Soft-errors Analysis Approach

The proposed soft-errors analysis on self-reconfigurable FPGAs has leveraged on the fault injection increased controllability that, coupled with the awareness of the configuration memory organization, enabled the execution of a detailed campaign according to pre-existent radiation test data, allowing to properly mimic different radiation environments.
In detail, the analysis has been focused on the self-configuration controller, which represents a crucial component for the operation and mitigation of dynamically reconfigurable applications, to evaluate the dependability of different implementation solutions with respect to different radiation profiles and application goals.
Furthermore, the so obtained results allowed the identification of the most suitable self-reconfiguration controller to be deployed as core component in an efficient and cost-effective technique for radiation analyses based on a low-cost, small-sized, and easy-operable neutron generator source, minimizing the time and the cost involved in classic radiation test instrumentation.

7.2 Background

Although SRAM cells' sensitivity to radiation-induced transient soft-errors is an inherent and well-known characteristic of SRAM-based FPGAs, the possibility to access and update their configuration memory at run-time provides many opportunities for the implementation of in-field fault detection and recovery techniques. In fact, the feature of reading the SRAM memory during the application execution coupled with the one of modifying its content enables the introduction of self-monitoring and self-repairing techniques that allow to increase applications dependability and to extend missions lifetime.
Therefore, is the following, an overview about the soft-errors sensitivity of the SRAM configuration memory in accordance with the deployment environment is provided. Subsequently, the opportunities and techniques enabled by the in-field reconfiguration are presented together with the self-configuration controller, which is the component allowing them.

7.2.1 SRAM-based FPGAs in Radiation Environments

As discussed in Section 1.3.3, one characteristic that makes SRAM-based FPGAs extremely valuable for the deployment in radiation environments is their high resistance to long-term permanent damaging effects caused by the accumulation of radiation dose (TID) if compared to Antifuse and Flash-based FPGAs.
On the other side, they have a higher susceptibility to soft-errors, such as Single Event Upsets (SEUs) that are transient phenomena consisting of a change in the state of one or more memory cells induced by radiation particles which can be recovered by refreshing the original cell value.
Although SEUs can affect the user memory elements of all FPGA technologies, they result especially critical in SRAM-based FPGAs since their configuration memory consists of SRAM cells that present a high sensitivity to such effects if compared with other memory technologies and, above all, directly control the application functionality as highlighted in Figure 7.1.
Therefore, for properly instrumenting mitigation and recovery techniques as well as for confirming their efficacy in preventing mission failures is fundamental to evaluate the device and application vulnerability to radiation-induced transient errors according to the deployment environment.
In fact, the radiation doses and spectrum that a system has to tolerate or mitigate as well as its failure modes during its mission strongly depend on the target radiation environment, as different interactions with the device semiconductor can be produced according to particle profiles and energies that could be encountered over time or in different locations.

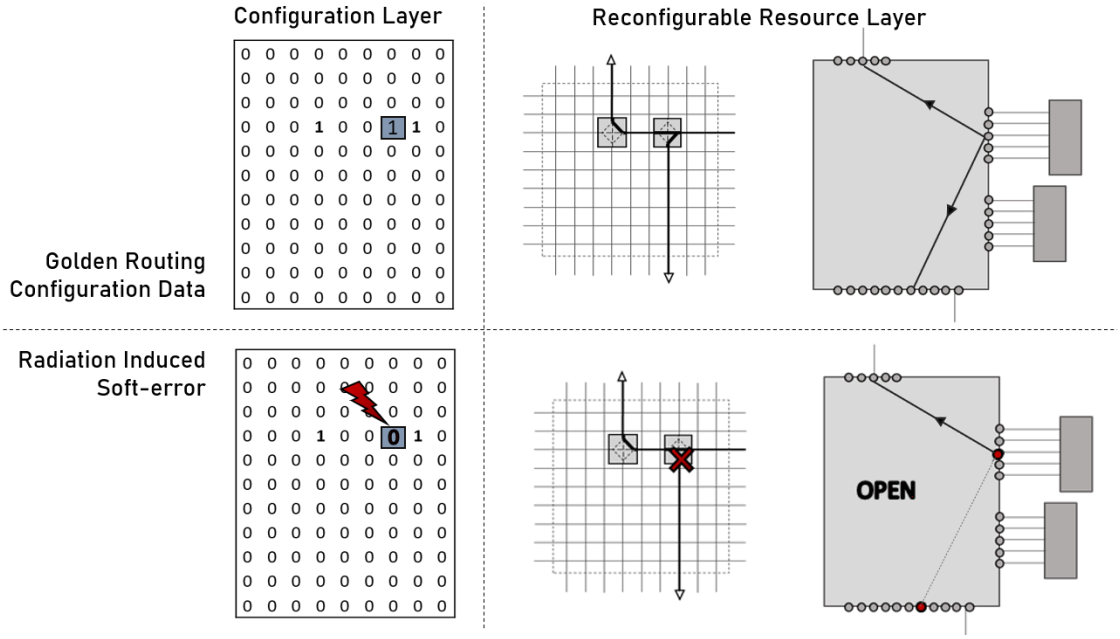

Figure 7.1: Configuration Memory Upset: Sample Routing Configuration Settings before and after a Radiation-induced Soft-error

In detail, when dealing with upsets in SRAM configuration memory, depending on the energy of the particle crossing the device, more than one memory cell can be affected by a Single Event, and this effect is also influenced by the initial state of the cell.
In fact, for a deployment environment in which the radiation dose is low, the main concern is represented by Single Bit Upsets (SBUs) as for the terrestrial and avionic scenarios, while for space environments and high-energy physics experiments also Multiple Bit Upsets (MBUs) must be considered due to the increased presence of particles with higher energies and their effect on the target application should be properly characterized and quantified.

7.2.2 Configuration Memory Run-time Access

Although Single Event Upsets represent the main concern for SRAM-based FPGAs deployed in radiation environment, being transient effects they can be recovered by refreshing the correct value inside the memory cell. In fact, the possibility enabled by dynamically reconfigurable SRAM-based FPGAs to read configuration memory data at run-time allows the detection of corrupted memory bits, while the writing enables their correction.
The key components enabling the implementation of in-field configuration memory upset self-detection and self-correction consist of the ICAP and its dedicated

self-configuration controller which allows managing the data exchange with the application layer.
In detail, the self-reconfiguration controller typically is the most crucial core for the execution of all the partial reconfiguration tasks oriented to the application execution optimization either in terms of computation and reliability.

Reliability-oriented Reconfiguration Techniques

Typically, the highest reliability on SRAM-based FPGAs is achieved by coupling structural redundancy to mask errors, as TMR techniques, with repairing capabilities to avoid the accumulation of errors. As TMR can be applied at different levels of granularity, different approaches to implement recovery techniques are possible [15][14].
As mentioned, the most consolidated FPGAs repairing technique for SEUs consists of correcting affected memory cells by rewriting on them their original value. This technique is generally referred to as *scrubbing* and can be performed with different approaches.
In detail, scrubbing can be either *blind* and thus performed preventively without any awareness of the current configuration memory corruption or based on *readback* procedures allowing the reading of the configuration memory content to check its sanity and perform the correction accordingly. Although *readback scrubbing* instrumentation introduces an increased complexity in its deployment if compared with the simpler *blind scrubbing*, it has the advantage of writing data into the configuration memory only when needed reducing the overhead and risks involved in performing this operation when unnecessary.
Additionally, scrubbing can be distinguished according to the configuration memory portions it is applied to since both full context, frames, and partial scrubbing are possible. The *full context scrubbing*, which is applied the whole configuration memory, has a simpler instrumentation but involves in the procedure also unnecessary information. The *partial scrubbing* instead is applied to limited areas of the configuration memory in which are configured specific modules or functionalities while *frame-driven scrubbing* is applied only to specific faulty frames, typically identified through readback procedures. In general both partial and frame-drive scrubbing pay the price of an increased implementation complexity but minimize the potential risk involved in the procedure as well as its overhead, especially when performed on-demand as in [66] and [107].
In fact, besides the aforementioned methodologies, scrubbing can be performed either periodically or according to specific policies or events based on self-monitoring and detection techniques, increasing the efficiency of the procedure and minimizing its ovehread, as proposed in [63] and [64].
Finally, as summarized in Figure 7.2, scrubbing can be implemented externally or internally according to needs and the target platform [14][33].

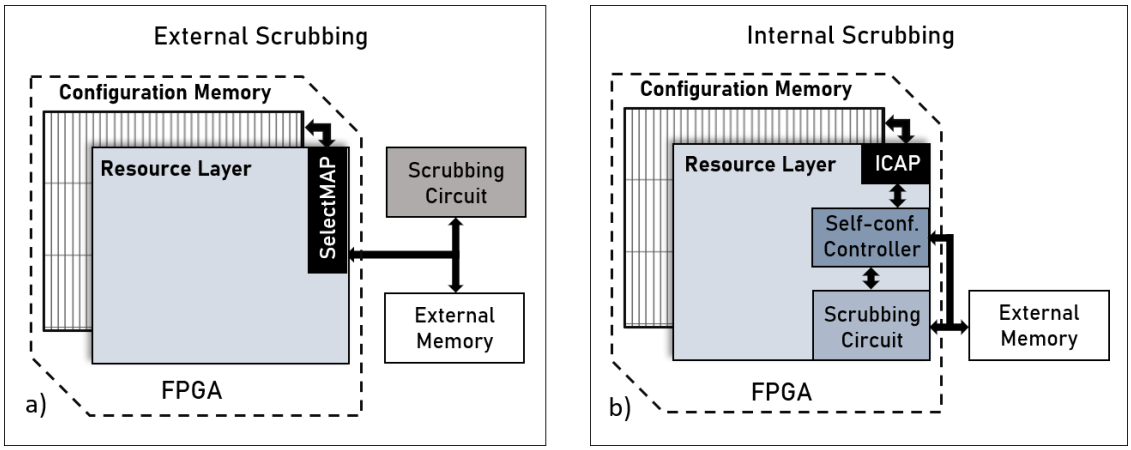

Figure 7.2: External (a) and Internal (b) Scrubbing Mechanisms

In the first case (Fig. 7.2a), the Scrubber circuitry is deployed outside the target device and typically the access to the configuration memory is performed through the SelectMap Port.
In second case (Fig. 7.2b), the Scrubber mechanism is integrated within the target device application layer relying on internal configuration ports, as the ICAP, and supported by dedicated self-configuration controllers, as in [66] and [107].
In fact, in Dynamically and Partially Reconfigurable modular systems, in addition to performance-oriented in-field upgrades, the access to the configuration memory from the application layer provided by the ICAP and its controller allows to efficiently perform selective readback and detailed error recoveries on-demand and on specific modules, saving both time and internal resources.

Self-configuration Controller

As discussed, the key component enabling dynamic and partial reconfiguration to perform run-time upgrades for both optimizations and reliability purpose is in Xilinx FPGAs is ICAP. In fact, differently from PCAP and MCAP which are respectively available only on SoC and in Ultrascale devices, the ICAP represents the most consolidated interface for internal reconfiguration, being available in all the recent FPGA families. In detail, the ICAP consists of a hardwired primitive providing a communication interface between the reconfigurable resource layer and the configuration memory layer for both reading and writing [47].
To manage configuration data traffic across the ICAP from the application layer a dedicated controller is required. In fact, the in-field access to the configuration memory, according to the target purposes and architecture, could be scheduled either from the software application running on the soft/hardwired processor or

from dedicated circuitry implemented in the programmable resources. In both cases, a self-reconfiguration controller managing low-level signals exchange with the on-chip ICAP primitive is required.
The simplified architecture of this component is depicted in Figure 7.3 considering the situation in which the in-field access to the configuration memory is scheduled from the microprocessor. Generally, the self-configuration controller consists of a Finite State Machine that directly handles the inputs and the outputs ports of the ICAP primitive according to the command received from the microprocessor and stored in the Control Registers.
In detail, when frames or partial bitstreams need to be acquired from the application layer either to be processed or to be stored in the external memory, these data are serially read through the ICAP by the FSM which is in charge to forward them to the Read FIFO, from which they could be accessed from the application layer.
When instead the data needs to be written into the configuration memory by the application, the process would be complementary, as the configuration data are extracted by the external memory or computed on the fly by the application layer. These data would be forwarded to the Write FIFO and serially processed by the FSM to serially write them into the configuration memory. In detail, the Read and Write FIFOs as well as the Control Registers consist of user RAMs.
For the efficient deployment of the self-configuration controllers, Xilinx provides a dedicated core implementing the aforementioned architecture, the AXI_HWICAP [121]. In detail, the main features of this core consist of maximizing the efficacy of the communication and synchronization procedure with the ICAP and enabling the usage of the AXI protocol and communication macro to ease the interfacing with the rest of the system [46].

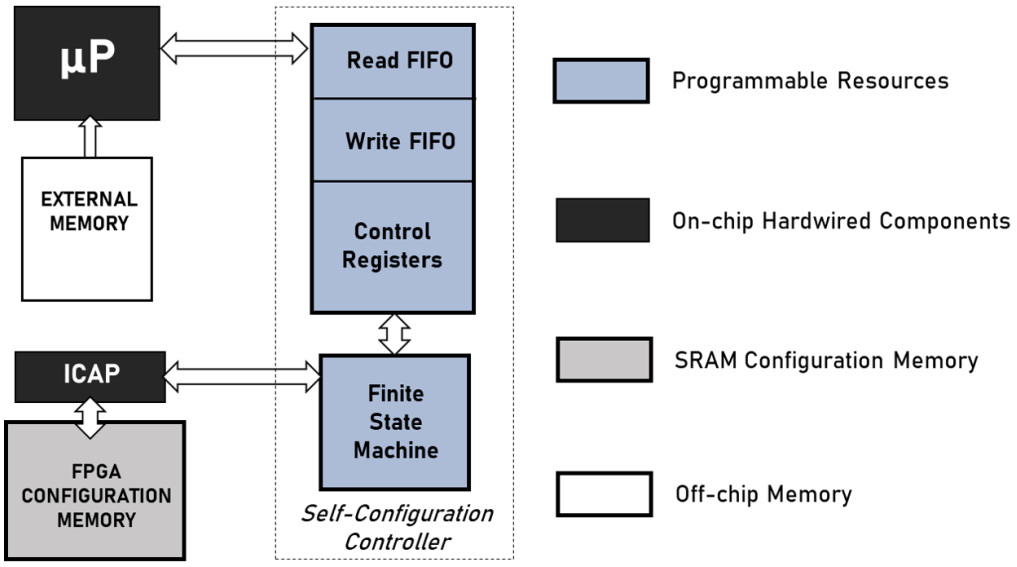

Figure 7.3: Simplified Architecture the of Self-configuration Controller

Other versions of the self-configuration controller have been proposed with the aim of increasing its dependability by cleverly introducing hardware redundancy techniques, as proposed in [122] and [123], as well as optimizing its size [124].
However, the vendor ICAP controller represents a more consolidated and standard solution as it is generally more productive thanks to the possibility to be efficiently optimized and customized within the Xilinx toolchain [18].
In fact, many design parameters of the AXI_HWICAP can be customized in the application design phase: in addition to the more flexible AXI interface and other settings, the size and the target implementation for the memories storing Read and Write FIFOs and Control Registers can be optimized according to the needs. In detail, for smaller memories Distributed RAMs can be used for their faster execution when the size is reduced while for bigger memories the BRAM macros can be used to save resources and increase performances [96][21][121].

7.3 Discussion and Outlook

The high benefits in terms of in-field reconfigurability provided by SRAM-based reconfigurable devices in the implementation of run-time optimization and system self-monitoring and recovery made dynamically reconfigurable applications an optimal solution for the deployment of high-performance and dependable computing system in radiation environments.
The key component enabling these optimizations is the self-reconfiguration controller, which is in charge of managing the exchange of data between the application layer and configuration memory through the internal configuration port for the implementation of techniques either oriented to the increase design performance or dependability through self-detection and self-correction mechanisms.
As this component is tailored on the reconfigurable resource layer and the SRAM configuration memory cells used to store its configuration are sensitive to radiation-induced transient errors, it has been chosen for its criticality as a priority target to be addressed for the characterization of self-reconfigurable systems dependability.
As discussed, many approaches are possible to perform preliminary evaluations on the device and system vulnerability to such effects, either through fault-injection campaigns and radiation testing.
If radiation testing is capable of providing experimental data on the device and application error rates in accordance with the target deployment environment, the cost and difficulty of accessing facilities in which radiation sources are available could make these tests unaffordable. On the other side, fault injections require long times to be executed but represent an extremely valuable alternative for the evaluation of soft-errors in the FPGA configuration memory.
Additionally, the possibility to perform such campaigns exploiting the awareness of the configuration memory structure and link with the reconfigurable resources

has the advantage of making detailed and controlled injections on specific resources and, when performed according to pre-existent radiation data, also to mimic the target radiation environment.
For these reasons, this approach has been followed to evaluate the dependability of the self-configuration controller
In detail, two implementations of vendor the self-reconfiguration controller have been evaluated in the DRPM architecture for different radiation scenarios considering SBUs, which are typical of low radiation environments as the avionic or terrestrial ones, and MBUs, which probability rises in a high radiation environment, as HEP and deep space.
This evaluation provided information on the different failure modes and probabilities as well as deployment guidelines for the two self-configuration controller implementations according to the environment and the DRPM goal [112].
Accordingly, the self-configuration controller has been used as the core for the instrumentation of an efficient and low-cost radiation test setup for the evaluation of the configuration memory sensitivity to both single and multiple bit upsets either from 0 to 1 and from 1 to 0.
In detail, the proposed instrumentation consists of an online semi-static self-test performed within the ZYNQ SoC that exploits the in-field access to the configuration memory to efficiently monitor the configuration memory corruption state.
The instrumented setup enabled the possibility of relying on a neutron generator Source, which is extremely cheaper and easily available if compared to the classic beam particles accelerators, and to obtain in a reduced amount of time useful information about the configuration memory sensitivity providing preliminary results on the impact of resources distribution and utilization [125].

Chapter 8

Self-Reconfigurable Systems Dependability Analysis

In this chapter, the analyses performed on dynamically and self-reconfigurable systems are presented and discussed.
Firstly, the dependability evaluation versus soft-errors of the self-reconfiguration controller within the DRPM framework is presented, discussing its implementation parameters, sensitivity, and failure modes for different radiation conditions achieved through dedicated fault-injections [112].
Subsequently, a cost-effective and efficient radiation testing instrumentation based on a semi-static online self-test for evaluating the error probability of the SRAM-based ZYNQ FPGA SoC under the low-cost and easy-operable neutron generator radiation source is presented, highlighting the advantages of obtaining meaningful data within a short amount of time and strongly reducing the typical costs implied in classic radiation testing [125].

8.1 Methodology

The proposed soft-error analysis for reconfigurable SRAM-based FPGAs has been focused on the self-reconfiguration controller dependability as well as on its advantages in performing self-monitoring and diagnosis.
In fact, it represents the crucial component that enables all the key in-field optimizations either for increasing the computation efficiency and for implementing self-diagnosis and self-repairing techniques in reconfigurable applications.
For this reason, the first part of the analysis has been focused on different implementation solutions of this component in the framework of a DRPM architecture addressing its reliability and failure modes versus different radiation conditions.
In fact, single bitflips represent the main concern in a radiation environment with a lower radiation dose as the one faced by avionic and terrestrial applications. For

deep space or high energy physics experiments applications instead, the presence of particles with higher energies able to produce multiple bits upset must be considered and characterized, as well as its different impact on the application functionality.
Therefore, a detailed fault injection mimicking these two scenarios has been performed. In detail, clusters of different shapes and sizes, which profile has been based on the radiation test experiment performed under Ultrahigh Energy heavy ions on the same technology presented in [109], have been used for the injection, monitoring their effect on the application behavior with respect to the controller implementation.
In detail, two versions of the self-reconfiguration controller have been considered. The first one relies on distributed RAMs for the implementation of the registers supporting the data exchange among the configuration memory and application layer. The other implementation instead uses the BRAM hard macros and the main differences with the other one are related to the density, distribution, and functions of the reconfigurable resources programmed on the FPGAs.
To evaluate the impact of their different characteristics and the relative operational failures, the two controllers have been integrated into two versions of a benchmark DRPM deployed on the ZYNQ 7020 SoC device, and their dependability has been analyzed by injecting on the configuration memory section devoted to the controllers' implementation both single and multiple bitflips.
The analysis performed enabled the identification of different sensitivities for the two implementations of the controller according to the target scenario as well as the relative DRPM failure modes, allowing the identification of guidelines for their deployment according to target environment and the application main goal [112].
From this evaluation, it has been possible to validate and select the implementation of the self-configuration controller for the instrumentation of an efficient setup to perform fast and low-cost radiation analysis on SoC FPGAs configuration memory.
In detail, the self-monitoring capabilities enabled by the controller allowed the implementation of an online semi-static self-testing approach deployed on the ZYNQ 7020 device able to speed up and ease the collection of data on its configuration memory radiation sensitivity.
In detail, the proposed test instrumentation enabled the evaluation of both single and multiple bit upsets considering the cases in which the initial state of the memory cell is either 0 or 1. This has been possible through the manipulation of the bitstream and to the knowledge of its composition coupled with the selective reading of configuration frames allowed by the ICAP and its controller while the sample was exposed under a neutron generator radiation source.
In fact, the key feature of the proposed implementation consists of enabling the fast collection of useful data about the device sensitivity to different bit upsets and their relation to the FPGA resource distribution while relying on this unpopular and low-cost radiation source, strongly reducing the cost, time, and effort typically required by radiation testing [125].

8.2 Soft-errors Analysis on Self-reconfiguration Controllers

In this section, the evaluation framework as well as the system and injection setup used for evaluating the self-reconfiguration controller dependability are detailed. Then the experimental results obtained according to the target deployment environment and reconfigurable DRPM application failure modes are presented and discussed.

8.2.1 Evaluation Framework

The main objective of this evaluation consists of evaluating the dependability of the self-reconfiguration controller managing the FPGA ICAP versus different radiation profiles.
Although several self-configuration controllers have been proposed in the literature (e.g., [122], [123], and [124]) the vendor-provided AXI_HWICAP controller [121] has been considered as it represents the most standard and consolidated solution. In fact, beside being already optimized by the vendor and enabling higher productivity thanks to increased flexibility provided by its AXI interface, it enables the possibility of easily customize its parameters within the design and development process on Xilinx toolchain.
To perform this analysis, the two versions of the controller that have been evaluated consisted of the one based on Distributed RAMs, indicated in the following as *Fabric*, and the one based the hardwired BRAM primitive, that will indicated in the following as *BRAM*.
Generally, distributed memories are preferred when their size is restricted as in these cases they provide better performances, while as the memory size increases BRAM implementations result preferable to save resources and accordingly achieve higher speeds.
Therefore, they strongly differ in the number, functions, and distribution of low-level resource and configuration bits utilizations. In detail, considering for the two implementations the same target portion of the configuration memory, the BRAM version implies a minor amount of programmable resources, as the main memory components are tailored on the on-chip hardmacro strongly reducing the design congestion if compared with Fabric version, which uses a large amount of SLICEM and routing to implement the memory architectures.
On the other side, the programmable resources used for the BRAM version imply a higher criticality as they are used for implementing the Data, Address, and Control signals, while in the Fabric RAM these signals are distributed and integrated with other elements of the component.
In fact, considering the different distribution both in terms of functionality and

configuration data, their sensitivity to different deployment environments implying varying probabilities of radiation-induced single and multiple bit upsets would present diverse rates and failure modes.
To evaluate these scenarios and the relative criticality with respect to the different operations involved in dynamic and partially reconfigurable systems, the two versions have been used within the same DRPM benchmark consisting of six reconfigurable computing modules that could be erased, refreshed, or swapped to accelerate either floating-point multiplications or additions.
The different radiation profiles instead have been emulated by tuning the bitflip injection amount and shapes. In detail, for mimicking low radiation environments where single bitflips are more likely to happen single bit corruptions have been performed in the area configuring the self-reconfiguration controller while for emulating higher radiation doses clusters of varying sizes and distribution have been injected in the same area according to the one observed in [109].
In order to produce accordingly the faulty bitstreams and to run the experiments, a dedicated injection platform has been instrumented enabling to define the target configuration memory area for the injection as well as to produce the wanted cluster shapes of bits to be corrupted.
The collection and classification of the results on the DRPM failure rates and modes have been then performed considering both tolerable and critical errors, according to the possibility to regain the system control after their emergence, and identifying the execution stage they are more likely to affect.

8.2.2 Board Setup

To perform the analysis on the self-reconfiguration controller, it has been inserted in the context of a dynamically and partially reconfigurable modular application, namely the Dynamically Reconfigurable Processing Module (DRPM), which represents one of the most popular architectural solutions for the deployment of run-time upgradable systems, as discussed in Section 2.2.1.
In detail, in the DRPM used for this evaluation, the dynamic region of the system consists of six reconfigurable modules in which floating-point multiplier and adder accelerators can be allocated or deallocated at run time, according to scheduling performed within the main task running on the on-chip microprocessor and in charge of managing the dynamic access to the configuration memory. Instead, the communication bus macro connecting the on-chip microprocessor with the reconfigurable module and the ICAP controller are statically programmed on the reconfigurable fabric, following the scheme reported in Figure 8.1.
This architecture has been used to evaluate both the BRAM and Fabric implementations of the self-configuration controller and thus two versions of the DRPM have been accordingly realized and the deployed on the ZYNQ 7020 for performing the fault-injection campaigns.

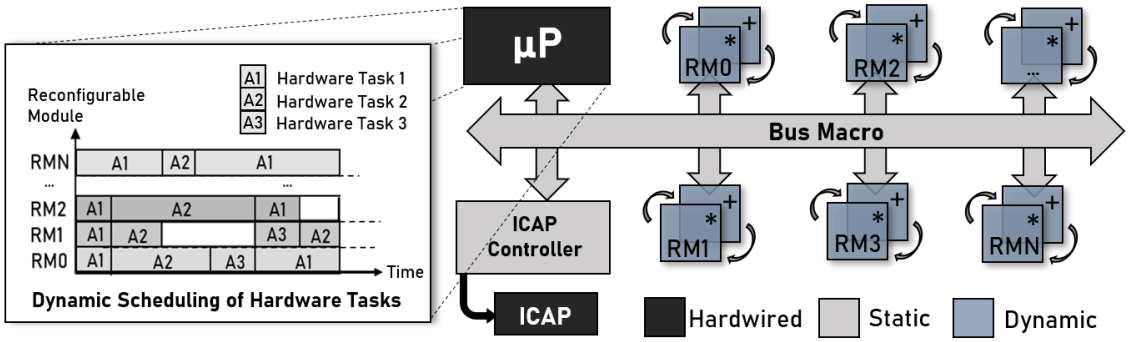

Figure 8.1: Simplified Architecture of the DRPM System

In detail, the DRPM system realized for the experiment consists of the on-chip ZYNQ processing system [22] that executes the main computation routine and manages the self-configuration controller, which consists of the AXI_HWICAP [121], either implemented with Fabric or BRAM registers.
Thus, in both versions, the ICAP self-configuration controller communicates with the processing systems and the other components through with an AXI Interconnect [46], which is also used for all the other communications as it connects the on-chip processors with the dedicated System Reset Core, used for trigger system reboot when required, and with the Dynamic modules.
In detail, each module, which according to the computation task required can be programmed ehiter as a floating-point multiplication or addition accelerator, is connected to the AXI communication macro for exchanging data with the main processing system via Partial Reconfiguration Decouplers [126], which consist of dedicated interfaces to support the fast and efficient exchange of data with the reconfigurable areas and to preserve the signal integrity at module boundaries when different tasks are allocated.
The DRPM system implemented on ZYNQ 7020 SoC is reported in Figure 8.2.

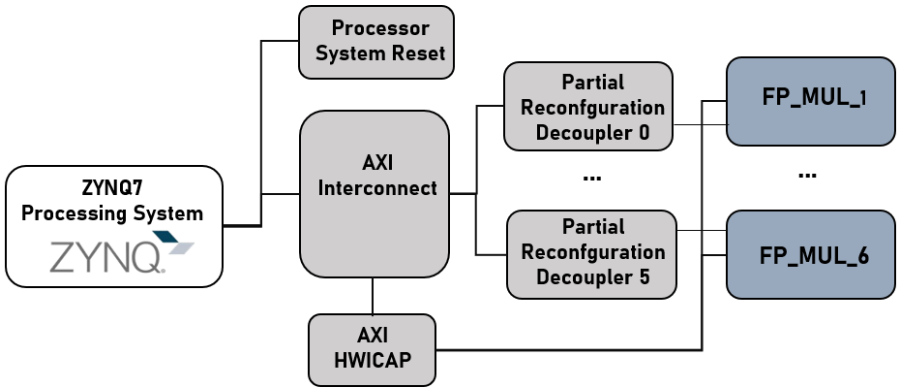

Figure 8.2: Block Design of the DRPM Implemented on ZYNQ 7020 SoC

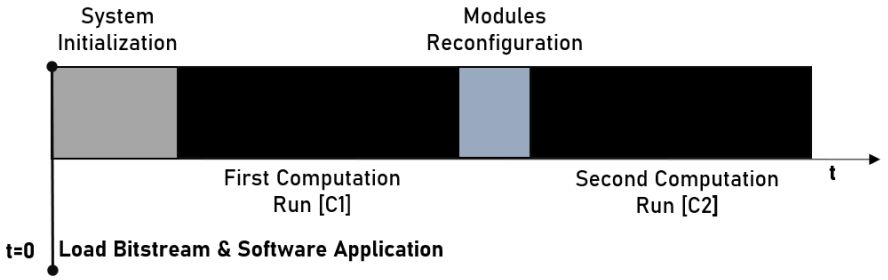

Figure 8.3: The DRPM Execution Flow Instrumented for Soft-errors Monitoring and Stage Classification

To conduct the injection experiments and analyze the failure rates and modes of different execution stages, the DRPM operation has been instrumented to be streamlined and to allow the identification of the error stage and its criticality.
In detail, at the system boot, the full bitstream, the partial bitstreams, and the software application executable are loaded inside the device respectively in the configuration memory, in the off-chip data memory, and the on-chip microprocessor.
Then the system components are initialized and the first run of computations (referred as to C1 in the following) is performed and the computation results are compared with the golden expected values.
After this, all the modules are reconfigured and the second computation (referred in the following as to C2) is run, checking again the obtained results, as schematized in Figure 8.3.
During the modules reconfiguration, the self-reconfiguration controller loads the partial bitstreams of the modules from the off-chip memory and stores them in the target configuration memory sections and the main routine running on the processing system is in charge of triggering these operations as well as verifying that they are successfully executed.
In fact, in addition to executing the main computing task, the software application embeds self-monitoring functions to check that the results of the operations performed in C1 and C2 are coherent with the expected values and that all the initialization and reconfiguration routines are performed correctly.
In detail, if errors are detected during the computation executions the on-chip microprocessor sends through the serial interface an error message on the output, stating the faulty computational section and the relative wrong answer.
If instead a mismatch is found in the return signatures of the functions involved in the system initialization or modules reconfiguration, the obtained signature is notified on the output.

8.2.3 Fault-injection Platform

For performing the fault injection on the target application and ad-hoc Fault Injection Manager has been instrumented within the Python framework and interacting with the Xilinx toolchain.
The Fault Injector core relies on PyXEL [113], which is a software tool developed starting from the building functions of COMET [99] that has been firstly extended for the analysis on the fault effects on 7 Family FPGAs routing and then broadened for the detailed and automatic manipulation of Xilinx bitstreams, providing an agile asset for low-level investigations and modifications.
In detail, the Fault Injection Manager developed for the target experiment integrates PyXEL, providing it the information for the generations of the corrupted bitstreams according to the parameters specified for the test, and is tightly coupled with the Xilinx environment for running the campaign and performing the proper collection and classification of the results, as summarized in Figure 8.4.
In detail, the Fault Injection Menger setup is performed by feeding it with the golden application bitstream, consisting according to the test of the DRPM architecture under evaluation with the ICAP controller either implemented with Fabric and BRAM, and the parameters related to target test, consisting of the area of the configuration memory implementing the self-configuration controller, the number of Injection for the target campaign and the bitflip cluster size, and produce as outputs all the relative faulty bitstreams and scripts for calling and managing the Xilinx tool in batch mode.
Once all the outputs files are generated, the fault injection campaign starts, iterative calling the Xilinx tools for programming the FPGA with the target application and collecting the output logs.

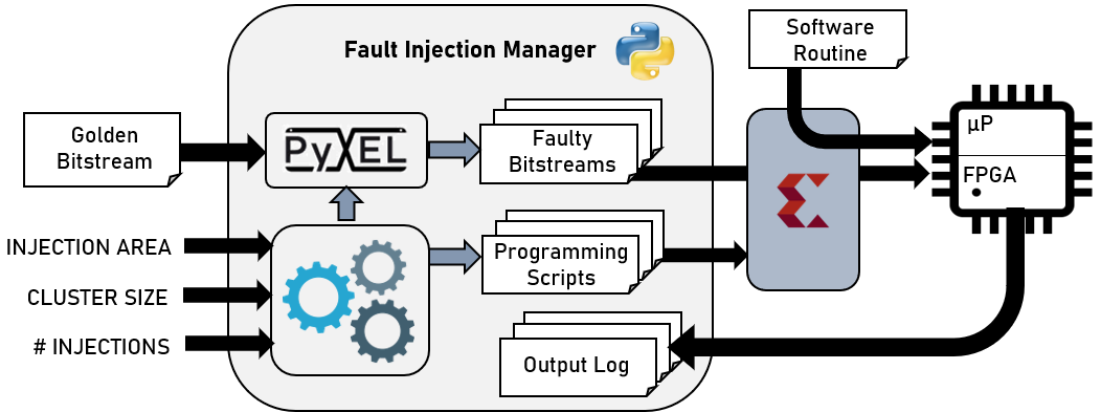

Figure 8.4: Fault Injection Manager and Instrumentation Overview

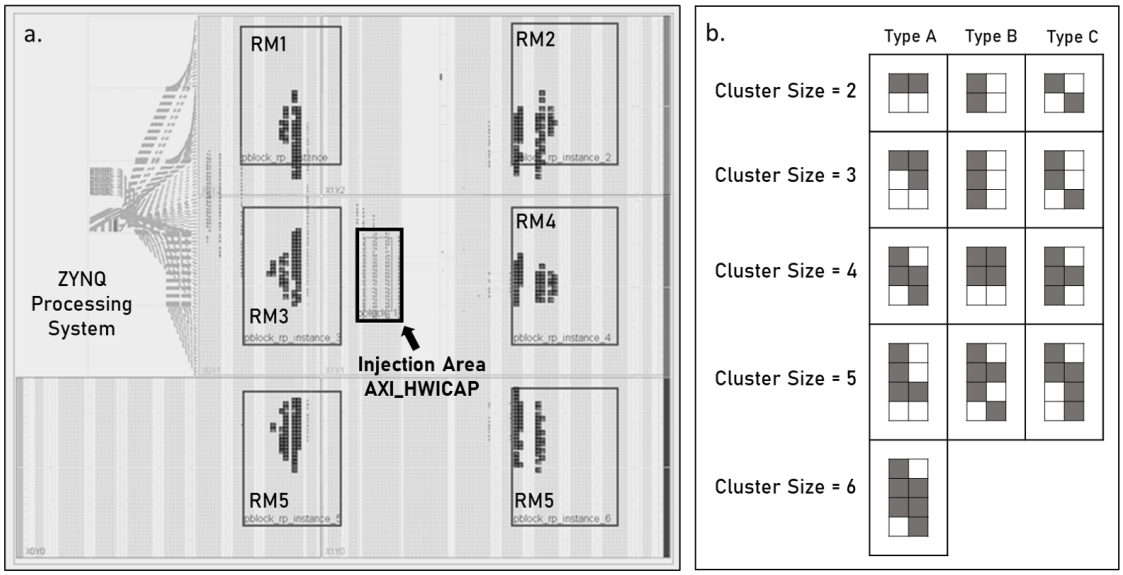

Figure 8.5: Fault Injection Parameters: a) Highlights of the AXI_HWICAP Fault Injection Area within the DRPM on ZYNQ 7020; b) Injection Cluster Sizes and Shapes as Observed in [109]

In detail, the target area for all the injection campaign experiments consists of the bounding box set for the placement of the Xilinx AXI_HWICAP, which has been designed to be the same for both implementations. The area in which the self-reconfiguration controller is placed and routed is shown in Figure 8.5a where the Vivado Implementation view on ZYNQ 7020 of the DRPM static components is reported highlighting as well the bounding boxes defined for the allocation of dynamic modules.
The cluster size instead is defined according to the target experiments. In fact, for the evaluation in low radiation environments single bitflips are corrupted in the bitstream to emulate SBUs while for the emulation of high radiation doses clusters of size 2, 3, 4, 5, and 6 bits are flipped according to the MBUs shapes observed in the ultra-heavy ions radiation test presented in [109] to accurately mimic the effect of particles with higher energies.
For each one of these cluster sizes, a different campaign of injection run is performed producing a set of faulty bitstreams in which the shapes identified for each size are randomly selected among the ones identified and reported in Figure 8.5b.
As the number of faulty bitstreams to be produced and the size of the upsets cluster is set for the experiment, the Fault Injection Manager instruments PyXEL for producing the relative amount of copies of the golden configuration file corrupted in the target area and creates all the TCL support scripts to be launched within Vivado in Command Line Mode to load the faulty bitstream and the software program.

At this point, the Fault Injection Manager is ready to start the campaign entering the Fault-Injection stage and iteratively calling the produced scripts for programming the device and monitoring its execution. At the end of each DRPM run with a faulty bitstream, the output logs obtained during its execution are collected and a new injection is performed.

8.2.4 Experimental Results

The dependability analysis on the two versions of the AXI_HWICAP self-configuration controller has been performed on the two implementations of the DRPM system deployed on ZYNQ SoC consisting of a dual-core Cortex-A9 processor integrated within a ZYNQ 7020 FPGA.
The resource usage with respect to the availability on the target device of the two controller versions, referred as to Fabric Controller and BRAM Controller, are reported in Table 8.1 highlighting the different usages in terms of Slices, RAM blocks, and routing segments and providing information about the configuration memory portion dedicated to their configuration, which has been kept identical.
For the dependability analysis of the two designs with respect to different radiation levels, two separate experiments have been carried out and for each one of them 2,500 faulty bitstreams with corrupted bits in the area implementing the controllers have been produced and evaluated.
To obtain a coherent comparison, besides locating the self-configuration controller in the same device bounding region, the same randomly selected locations of the corrupted bit or bit clusters within this region have been used for the two experiments.
As mentioned, the first experiment focuses on the Single Bit Upsets (SBUs) typical of environments with low radiation doses like the one faced by avionic and terrestrial applications, while the second taking into account environments with particles with higher energies, like the one deployed in space or HEP experiments, in which the probability of Multiple Bit Upsets (MBUs) results higher, and emulated according to the radiation data obtained in [109] on the same target FPGA fabric.

Table 8.1: ZYNQ 7020 Resources Availability and Usage for the Two Self-reconfiguration Controller Implementation: BRAM and Fabric

Resources	***Available***	***BRAM Controller***	***Fabric Controller***
LUT Slices [#]	13,300	354	722
Register Slices [#]	106,400	1,008	1,047
BRAM RAM18 [#]	280	2	0
Routing PIPs [#]	-	568,378	1,977,640
Configuration Frames [#]	10,008	280	280

The detected erroneous or unexpected behaviors of the application have been classified according to their criticality and the DRPM operation stage they corrupt. In detail, the errors causing the application to hang without the possibility to recover the execution control or performing a safe reboot have been classified as *critical*, while the ones which bring the system in an erroneous but defined state for their recovery or safe restart have been classified as *recoverable*.
Dealing with the affected operation section of the system, four stages have been identified and the errors have been classified accordingly: all the errors emerging before the first run of computation have been attributed to the Initialization Stage, referred as to INIT; many errors have been identified corrupting the computation section of the execution providing the wrong answers in both the computation runs and they have been labeled as affecting C1 & C2, while others were affecting only the second computation run that follows the Reconfiguration procedure and have been labeled as C2; finally, the errors happening during the reconfiguration procedure (referred as to RCNFG) have been classified accordingly.

Single Bit Upset Analysis

The results of the Single Bitflip Injection on both Fabric and BRAM Implementations are reported in Figure 8.6, both in terms of Total Error Rate percentage of the injected faults as well is in terms of criticality and failure stages.
In detail, in the graph of Fig. 8.6a, the Total Error Rate is reported distinguishing among the fault criticality, from which is possible to observe that the overall sensitivity, i.e., the percentage of injection producing misbehaviors in the application, of the BRAM Implementation is slightly lower if compared with the Fabric version. Considering instead only the *critical* and non-recoverable errors, the BRAM version presents a higher sensitivity with respect the Fabric one.

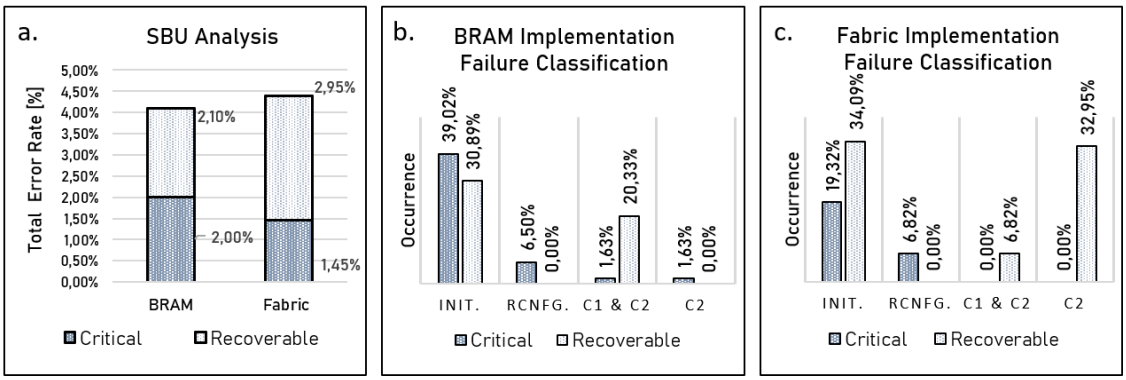

Figure 8.6: Soft-errors Analysis Comparison for Single Bit Upset in BRAM and Fabric Implementation: Total Error Rate (.a) and Stage and Criticality Classification Rates (.b and .c) [112]

Furthermore, looking at the failures distribution in terms of Stage and Criticality, reported in 8.6b and 8.6c, it is possible to observe that BRAM self-configuration controller presents the main ratio of failure in the Initialization Stage and most of them are *critical* errors, differently from the Fabric Implementation in which the main part of the errors happening in this stage are *recoverable*.
The failure probability and criticality in the Reconfiguration Stage instead have a similar profile for both implementations, as no *recoverable* errors are present and the BRAM version presents a slightly lower sensitivity percentage of such errors (6.5 %) versus the Fabric one (6,82 %).
The main observable difference between the two implementations consists of the failures affecting the computation stages. In fact, although in the Fabric Implementation the overall percentage of failures in computation results higher, no *critical* errors have been detected and the main part of them is affecting only the second computation run. On the other side, although their percentages are low, *critical* errors affecting either both C1 and C2 or only C2 have been detected in the BRAM implementation.

Multiple Bit Upset Analysis

As mentioned, when the deployment environment implies particles with higher energies, a single event can lead to upsets in more than one bit within the same configuration memory area.
In fact, bitflips on neighbor cells are possible and, according to the energy transferred, different clusters of different sizes can be produced, like the ones observed in [109] and reported in Fig 8.5b that have used in this experiment to emulate such environments. In detail, the identified clusters can involve up to 6 bitflips on 2 contiguous frames distributed across a maximum of 4 cell rows.
The 2,500 injections of this experiment have been performed considering clusters from 2 to 6 bits and producing accordingly corrupted bitstream where the shape of the target cluster was randomly selected among the types reported in Fig 8.5b.
The SBU error rates, criticality, and affecting stages have been obtained for both BRAM and Fabric Implementations following the same classification used for the other SBU experiment.
The Total Error Rate probability versus the cluster size is reported in the bar graph in Figure 8.7.
As expected, the error probability rises as the size of the injected MBUs increases, showing an higher sensitivity for the Fabric Implementation.
In fact, for the BRAM Implementation, the Total Error Rate probability starts from the 4.11 % observed in the SBU analysis and reaches a maximum of 13.7 % when the injected cluster involves 6 bits.
The Fabric version instead has a Total Error Rate of the 4.35 % when single bitflip are considered and rise up to the 19.57% in the worst case of 6-bit MBUs.

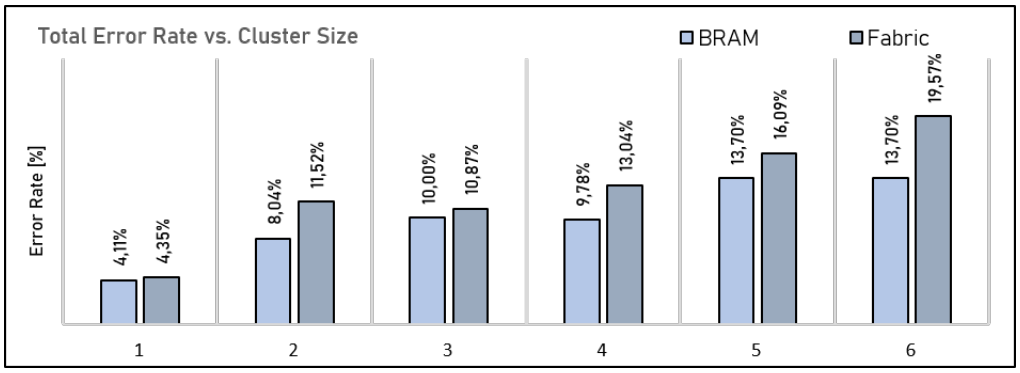

Figure 8.7: Total Error Rate for Multiple Bit Upset in BRAM and Fabric Implementations for Different Cluster Sizes

In Figure 8.8 the comparison among the observed *critical* errors causing a system hang, on the left, and the *recoverable* ones, on the right, is reported for both BRAM and Fabric Implementations.
From this comparison is possible to see that the errors causing the application to hang represent the minor component of the Total Error Rate, as the main component is represented by *recoverable* errors.
In fact, *recoverable* errors probability results higher if compared with the one of the errors causing system hangs and the increment of their probability results sharper as the cluster size increases, especially for the Fabric self-reconfiguration controller Implementation.
A comparative analysis on the failure stage probability for both implementations has been performed as well and it is reported in Figure 8.9 in terms of percentage on the total detected errors for cluster sizes ranging from 2 to 6, while the highlights of this failure are provided for both Initliazation and Computation stages in Figure 8.10.

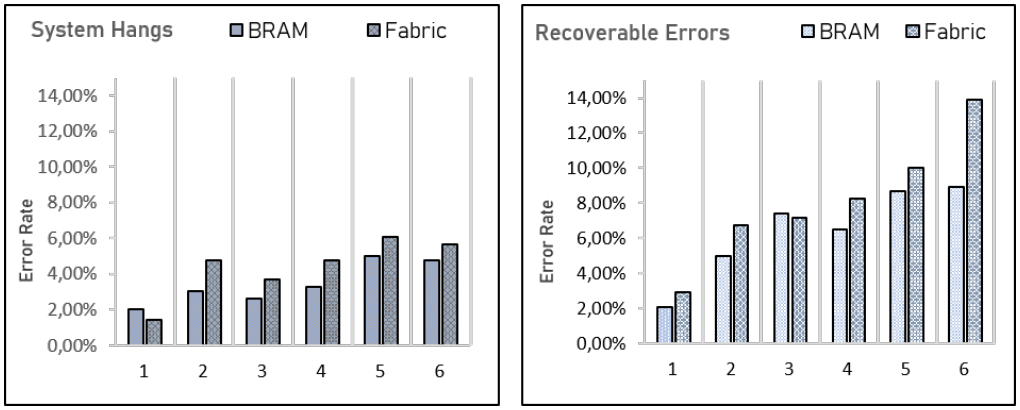

Figure 8.8: Multiple Bit Upset Criticality Comparisons: System Hang vs. Recoverable Errors for BRAM and Fabric Implementations for Different Cluster Sizes

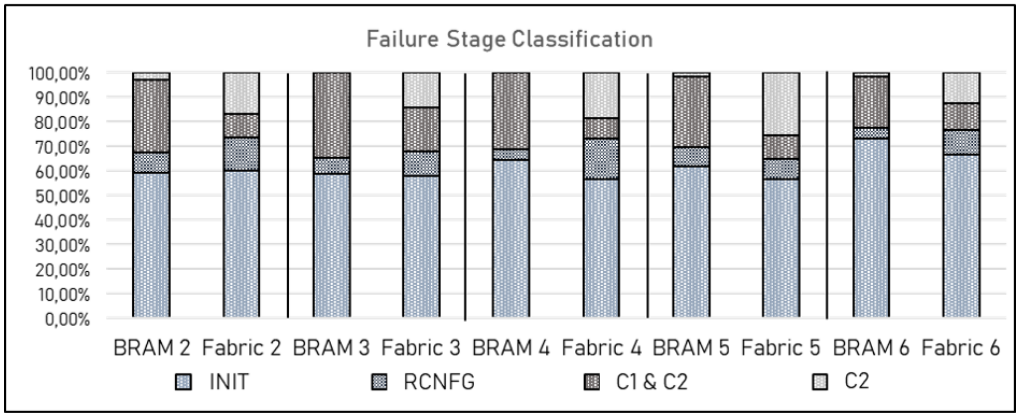

Figure 8.9: Failure Rate Distribution on DRPM Operational Stages according to the MBU cluster Sizes for BRAM and Fabric Implementations

In detail, the same trends observed for the SBU analysis have been found, as for both implementations the main part of the errors affects the Initialization stage representing on average 60% of the total observed error and showing a slightly higher ratio in the BRAM implementation, as reported in the left side of Fig. 8.10. The portion of errors affecting the Reconfiguration stage instead is smaller as its weight on the total error rate never exceeds 15%. However, it is possible to observe that, besides the cluster size, the failure rate in this stage is generally higher for the Fabric Implementation.
Dealing with the failures in the computation stages again is possible to see that, as in the case of SBUs, the BRAM implementation shows a higher rate of errors affecting both C1 and C2 runs, as highlighted in the right side of Figure 8.10, while in Fabric Implementation is more likely that only the second run C2 is affected.

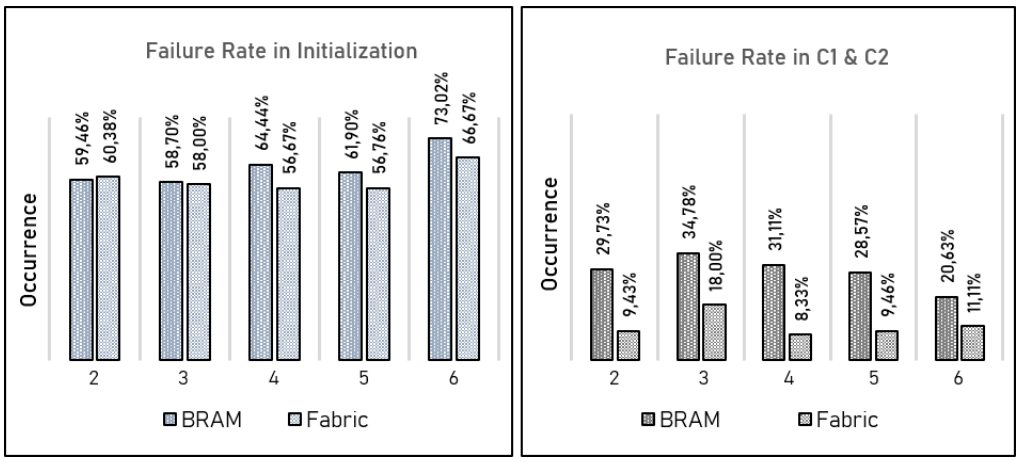

Figure 8.10: Highlight of Initialization and Computation (C1&C2) Stages Percentages on the Total Errors Rate according to the MBU cluster Sizes

8.2.5 Discussion & Highlights

The analysis performed on the two implementations of the vendor-provided self-reconfiguration controller highlights that for low radiation profiles the Fabric Implementation for the AXI_HWICAP is preferable, as the number of critical failures is lower, while the BRAM version presents higher dependability for higher radiation profiles.
In fact, the higher usage of reconfigurable slices and, above all, routing segments of the Fabric version related to the implementation of the memories using Distributed RAMs is reflected on a higher amount of programmed configuration bits which is more vulnerable to MBUs, as a Single Event has an increased chance to affect active bits on the same area that are programming multiple resources.
On the other side, the BRAM implementation strongly reduces the number of programmed resources, as it relies on the hardwired memory macros. However, the routing resources used in this version have a higher criticality as they mainly program the memory Data, Address, and Control lines, where even a single bitflip can produce a heavily incorrect behavior in the component increasing the criticality of SBUs, differently from the Fabric version in which there are more programmed bits but with an average lower criticality.
Furthermore, coupling this analysis with the one on the impact of the faults on the DRPM operational stages is possible to define guidelines on which implementation would be preferable according to the target applications and environment, summarized in Table 8.2.
In fact, apart from applications in which total failures need to be minimized, for radiation environments in which SBUs are the main concern, the AXI_HWICAP based on the Fabric Implementation is preferable for its lower amount of *critical* errors, especially in the Computational section of the DRPM execution, which motivates its usage also in the applications in which reconfiguration is often called.
For radiation environments in which MBUs are more likely to happen, the BRAM version of the controller is generally preferable due to its lower sensitivity both in terms of failure rate and criticality, especially in the reconfiguration procedure. As a unique exception for these environments, for the application mainly focused on computations, the Fabric Implementation could be considered due to the lower error rate obtained in the computational sections of the execution.

Table 8.2: Controllers Applicability to Radiation Environments and Missions

	Low Radiation Level	***High Radiation Level***
Computation-intensive Applications	Fabric Controller	Fabric Controller
Reconfiguration-oriented Applications	Fabric Controller	BRAM Controller
High-reliability Applications	BRAM Controller	BRAM Controller

8.3 Self-testing Technique for Cost-effective Radiation Analysis

In this section, the approach used for the development of a cost-effective neutron generator radiation testing for SRAM-based FPGAs relying on their self-monitoring capabilities is presented, followed by the description of online semi-static self-test board setup and the test instrumentation and equipment. Finally, the experimental results obtained thought the proposed approach are provided and discussed.

8.3.1 Overview

As mentioned, radiation testing is the most accurate approach to obtain information about devices and applications vulnerability to radiation-induced effects and consists of exposing them to radiation sources able to mimic the target deployment environment for observing their impact on the target system.
Concerning SRAM-based FPGAs testing for the evaluation of the SEE portability in their configuration memory for a given flux, one of the most popular radiation sources consists of neutrons that, due to their absence of charge, present a high penetration capability coupled with the advantage of avoiding other unwanted long terms damaging effect on the part.
These sources are obtained through nuclear reactions with acceleration that are typically produced thanks to complex and vast particle accelerators, such as LINAC, Cyclotrons, and Van de Graaff systems, which can provide high-flux and tunable beams that can be focused on the device under test.
However, the facilities hosting such instrumentation are extremely few worldwide and require high costs to be accessed. Furthermore, as the demand for accessing these instrumentations is generally higher than their availability, the opportunities to exploit them are extremely few and typically strictly bounded by tight schedules.
In this view, although device radiation testing has not been intended as their main utilization, Neutron Generators (NGs) can produce neutrons by fusing hydrogen isotopes and can represent an attractive alternative to large-yield and costly particles accelerators thanks to their small size, low price, and safe and easy operability. In fact, with a typical size that does not exceed 1 meter and their isotropic nature, they can provide a neutron yield useful for many experiments and applications enabling to be deployed in facilities and laboratories where enormous and expensive accelerators cannot be hosted [127][128].
Therefore, although their flux has lower energies and is not uniform as the one of typical radiation testing equipment used for electronics validation, NGs can be considered as a valid alternative by taking advantages of the easier setup and scattered source that, if coupled with a clever and efficient test instrumentation, enable to induce both single and multiple upsets in the FPGA configuration memory and to obtain useful information on its cross-section within a shorter time while reducing

the test cost and complexity.
In detail, to speed up and maximize the test efficacy, an online semi-static FPGA self-test based on the in-field reading of the configuration memory has been proposed confiming the benefits provided by SoC Reconfigurable FPGAs self-monitoring capabilities.
In fact, as the BRAM version of the AXI_HWICAP self-configuration controller has confirmed to be the most reliable implementation in presence of both single and multiple bit upsets for reconfiguration oriented applications, it has been used to support the system self-monitoring, which has been performed relying on the on-chip microprocessor elaboration capability coupled with the awareness of the low-level configuration frame organization.
The proposed board setup implemented in the ZYNQ SoC, coupled with an efficient, automated, and streamlined radiation test instrumentation, has confirmed to be a valid methodology for the fast and cost-effective collection of experimental data on the FPGA configuration memory susceptibility to radiation-induced errors minimizing the challenges and maximizing the benefits of relying on neutron generator radiation source.

8.3.2 Hardware Setup

The main goal of the proposed test consists of evaluating the SRAM configuration memory sensitivity to both single and multiple bit upsets either in the case in which the cells are not programmed, thus with their initial value set to 0, and in the one in which there are programmed to 1.
To achieve fast and efficient collection of data a semi-static online self-testing approach has been instrumented on the board under test providing to the system self-monitoring and self-refreshing capabilities while irradiated.
In detail, the proposed testing approach has been classified as semi-static since the setup has been contrived to maximize the number of unused resources to be tested by relying on a streamlined and low-overhead oriented on-board monitoring system based on the synergy between the on-chip microprocessor and the in-field access provided by the Internal Configuration Access Port and its controller.
In this view, the test setup has been instrumented to cover both the initial cell state conditions and has been developed for the target part pursuing the minimization of the active resources implementing the self-monitoring system, which consists of the on-chip microprocessor, the System Reset IP, the BRAM version of the AXI_HWICAP, and the AXI Interconnect for the exchange of data among these components, as reported in Figure 8.11.
In fact, as the target board consists of the ZYNQ 7020 SoC FPGA that includes an ARM-based on-chip processing system, the main test routine is running on the microprocessor.

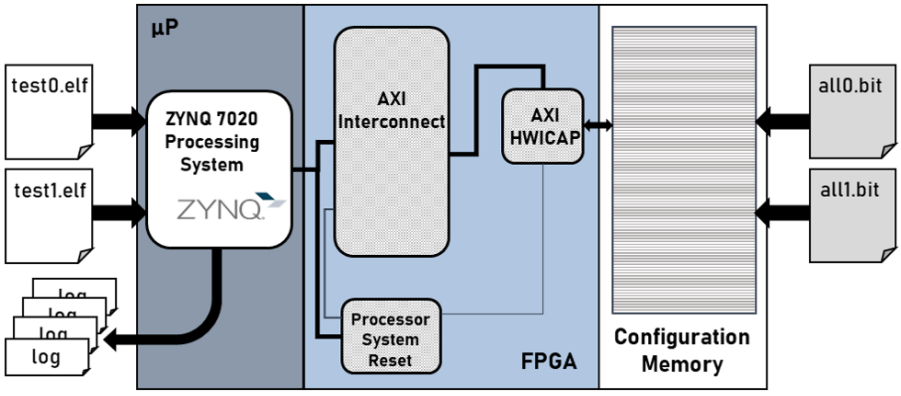

Figure 8.11: ZYNQ 7020 SoC Setup for the On-line Semi-static Self-testing [125]

For evaluating the upset probability from 0 to 1, indicated in the following as $0 \Rightarrow 1$, and for the upset from 1 to 0, indicated as $1 \Rightarrow 0$, two complementary test software routines have been developed for selectively reading the frames on the configuration memory, programmed accordingly with two complementary bitstreams. In detail, the software test routine for $0 \Rightarrow 1$ running on the processing system periodically and iteratively calls the AXI_HWICAP for reading all frames left unused, and thus defined valid and under the test, and compares word by word the obtained values with the expected one, which consists for this test consist of all 0s. When mismatches are found, the on-chip processor logs the position and the amount of the corrupted bits, to allow in the post-processing phase to distinguish among uncorrelated events and multiple bit upsets and, at the end of the reading cycle, it performs a hard reset of the board rebooting itself and reprogramming the FPGA with the golden bitstream, as summarized in the pseudocode of the self-testing routine reported in Figure 8.12.

```
while (1){
      Init(HWICAP)
      for (i= 0; i<n_frames_unsed; i++){
           frame_under_eval = read_frame(i)
           differences = compare_golden_0(frame_under_eval)
           if (differences != 0){
              upset_count= upset_count + differences
              log_far_address(frame_under_eval, differences)
           }
      }
      if (upset_count != 0)
          reset_fgpg_up()
      else
          wait (read_back_period);
}
```

Figure 8.12: Online Self-testing Software Routine Pseudocode [125]

As it will be discussed in the following, the fast and efficient self-reboot of the system is possible thanks to a software function call accessing the board low-level reset registers and to the fact that the device boot has been instrumented to rely on the onboard SD-card.
Although the complimentary test *1* ⇒ *0* follows the same approach used in the previous one, in this case, in this case the configuration memory bitstream has been manipulated relying on PyXEL tool manipulation feature to set to 1 all the memory frames programming unused resources [113].
In details, all the frames configuring the unused LUTs and BRAMs initialization has been forced to be programmed at 1 and have been labeled as valid and under test, while the frames configuring the routing resources have been discarded from the evaluation since they cannot be all active in real application scenarios as their organization does not follow a compact distribution [99].
In Figure 8.13a the bitmap of the configuration memory bitstream used for the *0* ⇒ *1* test is reported, where the white parts represent the programmed bits used for configuring the AXI_HWICAP-based Monitor, consist of the AXI Interconnect, the System Reset IP Core, and the BRAM-based ICAP controller. As the black pyxel represents the non-programmed bits, it is possible to observe that they represent the major components in the target bitstream.
In Figure 8.13b instead, the configuration memory bitmap relative to *1* ⇒ *0* test is reported, where is also possible to distinguish the LUT frames under test (the white slim columns) and the one dedicated to the user memory (i.e., BRAMs, the white larger horizontal stripes on the right). Finally, in Figure 8.13c, the Vivado Implementation view of the *1* ⇒ *0* test configuration is reported to show the link among configuration memory frames and the resource layer.

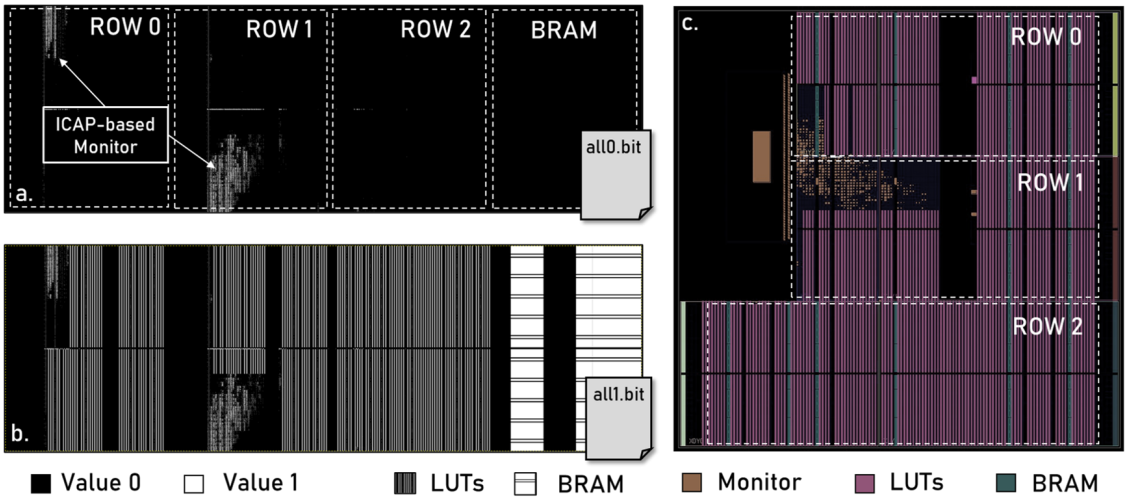

Figure 8.13: Bitstream Bitmaps for *0* ⇒ *1* (a) and *1* ⇒ *0* Tests (b) Obtained with PyXEL and the Vivado Implementation View of the *1* ⇒ *0* Setup (c)

8.3.3 Radiation Test Instrumentation & Equipment

The radiation test instrumentation consists of one side of the aforementioned board setup deployed on the sample irradiated on the neutron generator chamber and on the other of the automated test controller running on the host computer outside the room.

Automated Test Setup

The radiation test setup instrumented for the evaluation is reported in Figure 8.14 and consists of components placed in the neutron generator test chamber, as the sample board and the neutron generator, which yield is monitored through a neutron detector to monitor, and the host computer running the experiment software manager for logging the results and perform board power cycles in case of hangs that is placed outside the chamber together with the dedicate power supply controller.
In detail, the host computer is connected to the sample board through the serial interface with a 5 meter USB cable and runs a Python program that is in charge of managing the board output logs as well as the sample power supply.
In detail, the test manager starts the logs of board outputs received through the COM port when the neutron generator is turned on, records the received logs, and detects board hangs by elaborating the received outputs and checking for communication timeouts.
Furthermore, the automated monitoring program is in charge of managing the sample power supply through the Power Switch Controller which is connected with the computer through the Ethernet protocol.

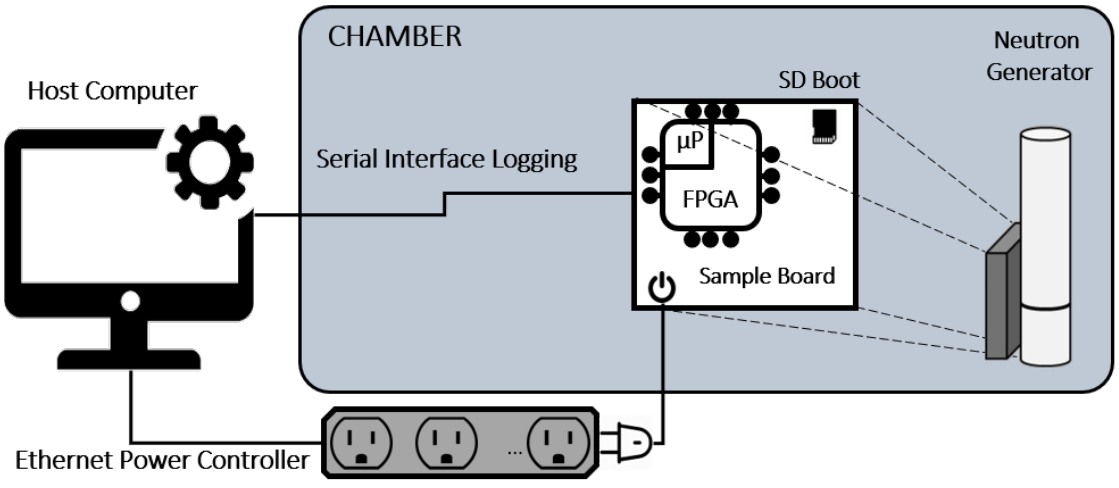

Figure 8.14: Overview of the Instrumented Neutron Generator Radiation Test Automated Setup [125]

In fact, every time a board hang is detected as a 5 seconds time-out, the test manager closes and saves the current log, performs a 2 seconds board power cycle by triggering through Ethernet the Power Controller, and starts a new log taking a trace of the occurred event.
On the other side, the device under test has been instrumented to boot itself from the SD card integrated into the board, and thus, every time the power supply is activated it can fastly reboot itself without the need of being reprogrammed by the host computer through the slower serial interface.

Test Equipment

As mentioned, the target part for the radiation analysis consisted of a ZYNQ-7020 SoC FPGA embedded on PYNQ Z2 board, which integrates a ZYNQ XC7Z020-1clg400c FPGA with a Cortex-A9 dual-core processor and mounts a DDR3 external memory as well as the support for both JTAG and SD-card programming modes.
As already discussed, one of the main features of the test setup consists of using a minimal amount of device resources for the implementation of the online self-monitoring system to maximize the number of unused resources, and thus configuration memory frames, that can be monitored during the test.
The resource usage of the on-board monitoring system, which is identical for both $0 \Rightarrow 1$ and $1 \Rightarrow 0$ tests, is reported in Table 8.3 in terms of percentage with respect to the ZYNQ FPGA availability, confirming its minimal area overhead.
The neutron generator used as radiation source instead consisted of a ThermoFisher P385 [129], which Yields and operation specifications are reported in Table 8.4.

Table 8.3: ZYNQ 7020 FPGA Resource Availability and the Relative Utilization for the Self-Monitoring System [125]

Resources	***ZYNQ 7020***				
	Logic Slices	***Look-Up Tables***	***Flip-Flops***	***Block RAMs***	***DSPs***
Available [#]	13,300	53,200	106,400	630 KB	220
Used [%]	4.10 %	1.8 %	1.76 %	0.71 %	0%

Table 8.4: Neutron Generator Source Technical Specifications [129]

ThermoFisher P385 Technical Specification	
Nominal Neutron Yield	3.0E+08 n/s
Maximum Neutron Yield	5.0E+08 n/s
Operating Voltage	from -40 kV to -130 kV
Power Consumption	∼75 Watts
Control Interface	RS-232, RS-422, RS-485

The used model provides digital control support enabling to manage its operation either manually or by relying on ad-hoc timers, integrates different safety features, such as a built-in key-lock and a external interlock, and eases its remote management through an open-source software interface running outside the radiation chamber.
Furthermore, the neutron fluence provided by the NG source has been constant during the experiment, and to obtain the actual flux at the part it has been computed according to the manufacturer stated yield and verified using the Far West Albatross 2080 Model [130] neutrons monitor detector, which was placed in the radiation chamber.
In Figure 8.15 a photograph of the aforementioned setup and instrumentation is provided, showing the device under test mounted on a polyethylene support to maintain the FPGA as close as possible to the radiation source, which is placed at the bottom of the neutron generator. In detail, the part is kept at 1 cm from the NG surface that added to the 5 cm between the surface and the inner radiation source makes the total distance between the FPGA and the neutron source equal to 6 cm. Additionally, it is possible to see the Albatross detector on the right of the neutron generator and the 5 meters long USB cables used for transmitting the application output data and the ones carrying the power supply going outside through a passage in the wall between the radiation room and radiation-free area in which are placed the host computer and the Ethernet power controller.

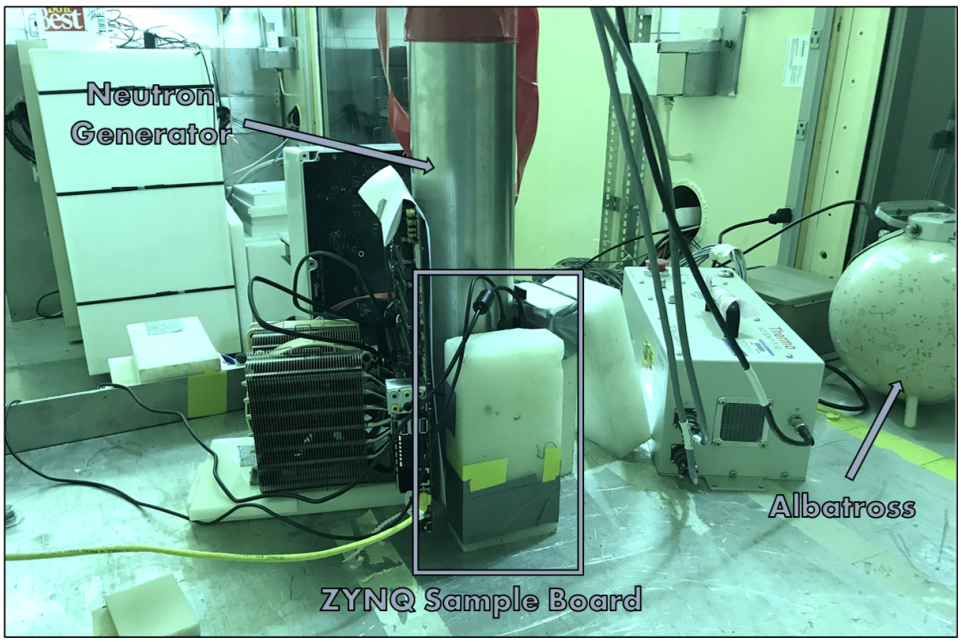

Figure 8.15: Instrumented Neutron Generator Radiation Test Setup: the ZYNQ-7020 SoC FPGA Sample Board Juxtaposed to the ThermoFisher P385 NG, which is Monitored by the Albatross 2080 [125]

8.3.4 Experimental Results

The main goal of the proposed test consists of evaluating the SRAM configuration memory sensitivity to both single and multiple bit upsets either in the case in which the cells are not programmed, thus with their initial value set to 0, and in the one in which there are programmed to 1.
The results obtained with the proposed neutron generator instrumentation and self-testing online approach for ZYNQ SoC 7020 FPGA configuration memory are reported in Figure 8.16 in terms of Upset Probability for both $0 \Rightarrow 1$ and $1 \Rightarrow 0$ Tests.
In detail, the obtained results have been gathered with a computed constant flux at the part of 1.18 x 10^6 [n/cm^{2}s] within a total available irradiation time of 3 hours of which 1 hour and 59 minutes have been used for performing the $0 \Rightarrow 1$ test, while 58 minutes have been used for the $1 \Rightarrow 0$ complimentary test.
Thus, the obtained results have been normalized according to the effective duration of each test and to the frames actually under evaluation in the two situations. In fact, the configuration frames under evaluation on each reading cycle for the $0 \Rightarrow 1$ test have been 4,804 over the total of 10,008 while for the $0 \Rightarrow 1$ test 2,360 frames have been considered.
As for the SEU and MBU $0 \Rightarrow 1$ and SBU $1 \Rightarrow 0$ evaluations the number of detected events has been higher than 50, the relative errors bars have been calculated relying on the standard deviation approximated with the Normal distribution, while for the SBU $1 \Rightarrow 0$ events, which have been less the 50, the 95% confidence intervals for the Poisson statistic have been used, as indicated in [30].
From the performed analysis has been possible to observe that when the initial state of the cell is 0 the probability that a single event upsets only one bit is higher than the occurrence of SEU causing multiple bit upsets.

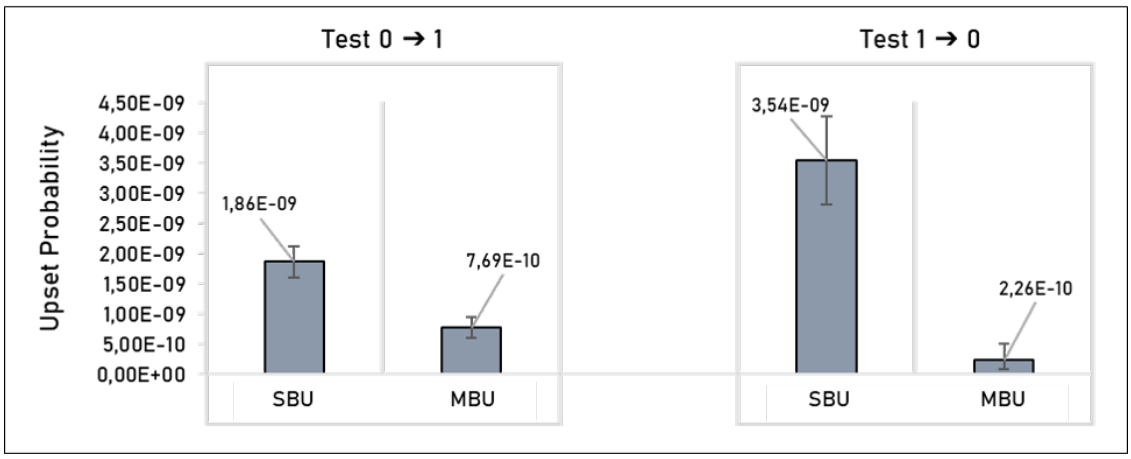

Figure 8.16: SBU and MBUs Probability in transitions from $0 \Rightarrow 1$ and $1 \Rightarrow 0$ with a 1.18 x 10^6 Constant Flux [n/cm^{2}s] [125]

This behavior can be observed also in the case in which the initial cells state consist of programmed 1s, although in this case the SBU probability results increased if compared with one observed in the *0 ⇒ 1* evaluation while the MBU probability results lower, presenting a minimal probability and widening the gap among the two probabilities when the most of the configuration memory bits are set to 1.
Observing this behavior it is possible to make considerations about the wider disparity amount SBUs and MBUs observed in the two tests. In fact, the higher Single Bit Upset rate observed in the *1 ⇒ 0* can be attributed to the frames programming the initial state of the BRAMs, which present a more compact and dense distribution of the programmed bits, and that in this test the have a higher weight than in the other one.
Furthermore, again thanks to the possibility of post-processing the bitflips positions within the frames to correlate the number of upsets to each event, an additional classification of the obtained upsets has been performed providing the percentage of the occurrences of both Single Bit Upsets and differently-sized Multiple Bit Upsets, as reported in Table 8.5. In detail, the correlated bitflips consist of upsets affecting no more than four adjacent cells on the vertical axis belonging at the most to two contiguous frames on the horizontal axis.
This highlights as the MBUs percentage as well as their cluster sizes results higher in the case in which the initial cell state is equal to 0, while in the case the initial state of the cells is programmed to 1 a minor percentage of MBUs occurs and with involving a maximum of 3 bits.
These results, besides confirming the efficacy of the proposed test instrumentation for the fast collection of data on the configuration memory sensitivity to both single and multiple upsets, provide insights about the impact of the distribution of the programmed bits can have on application radiation sensitivity.
In fact, according to the performed analysis a more compact and constrained design implementation, which implies a higher density of programmed bits, for a given flux could be more tolerant versus MBUs induced by a single particle while being more prone to SBUs if compared to its widened version.

Table 8.5: Correlated Bit Upsets Percentage of Occurrences in Test *0 ⇒ 1* and Test *1 ⇒ 0* [125]

Correlated Bit Upsets per Event	***Percentage of Occurrences***	
	Test *0 ⇒ 1*	**Test *1 ⇒ 0***
1	78.38%	94.17%
2	14.9%	4.85%
3	0.68%	1.94%
4	1.35%	0.00%
7	0.68%	0.00%
8	0.68%	0.00%

8.3.5 Discussion & Highlights

Thanks to the online self-monitoring capabilities provided by the Internal Configuration Access Port and its controller coupled with the awareness of the configuration memory organization it has been possible to instrument a cost-effective and streamlined radiation testing based on low-cost and easily available neutron generator radiation source.
The proposed instrumentation enabled gathering information about FPGA configuration memory sensitivity considering both Single and Multiple radiation-induced upsets within a short amount of time and strongly reducing the cost required by the expensive and difficultly available accelerators used in classical radiation testing approaches.
In detail, the proposed methodology relies on one side on a self-monitoring circuitry deployed on target ZYNQ SoC 7020 FPGA part based on the BRAM implementation of AXI_HWICAP and managed by an online self-testing routine running on the on-chip microprocessor that selectively reads the configuration memory portions under test while irradiated, and on the other on an automated test manager running on a host computer outside the radiation chamber controlling the experiment for the fast and efficient collection of the data.
In fact, thanks to the in-depth knowledge about the configuration memory organization and its detailed manipulation have been possible to obtain information on the device cross-section for SBUs and MBUs considering both the cases in which the cells programmed with 0s and 1s within a total irradiation time of 3 hours and with a constant flux of at the part of 1.18×10^6 produced by a low-cost and small-sized neutron generator.
The experimental results obtained with the proposed approach confirmed its efficacy in gathering useful data within a short amount of time by relying on low-cost and easy-operability instrumentation coupled with the self-monitoring feature enabled by dynamic and partial reconfiguration. Furthermore, the collected data provided preliminary insights about the impact of the programming bit distribution on the device and application sensitivity to radiation.
This last consideration opens the way for future developments on the investigation of mitigation approaches based on robust FPGA resource mapping for increasing the design dependability through the awareness of the link among programming bits and reconfigurable resources.

8.4 Research Advancement

As the demand for high performance and dependable reconfigurable systems based on reconfigurable FPGAs to be deployed in radiation environments is continuously growing, the sensitivity of the SRAM configuration memory of these devices to radiation-induced transient effects represents a crucial point to be addressed to

ensure the target system readiness in safely performing its mission.
For this reason, radiation analysis based on both fault-injections and radiation testing represent key procedures for verifying the device and application dependability according to the target deployment environment and for designing or validating efficient mitigation techniques.
As performing run-time upgrades, either for execution optimizations or error recovery and mitigation, is one of the main properties that makes dynamically and partially reconfigurable FPGAs extremely valuable for these applications, the performed radiation analysis has been centered on the self-reconfiguration feature, both for evaluating its dependability within reconfigurable applications and for taking advantages of the self-diagnosis capabilities it can provide.
In detail, a dedicated soft-error analysis has been performed targeting the self-reconfiguration controller within the context of a dynamically and partially reconfigurable modular system evaluating the sensitivity of two different versions of the component implementation versus different radiation environments [112].
The strategy used for this analysis consisted in a detailed fault-injection campaign performed according to pre-existent radiation test data on the same device technology for emulating different radiation scenarios, like the one typical of avionic and aerospace applications where single bitflips represent the main concern and the one where particles with higher energies are present and the chance for a single event to produce multiple bitflips increases, as for space and HEP experiment radiation environments.
In fact, leveraging on the knowledge of configuration memory organization and the possibility to control the bitflip injections by reproducing the wanted bitflip cluster sizes and shapes, different radiation profiles have been emulated. This enabled obtaining indications about the different sensitivity and failure modes of the vendor AXI_HWICAP self-reconfiguration controller when implemented using Distributed RAM or BRAM hardwired primitive and to draw guidelines on their relative applicability.
As observed, the distributed AXI_HWICAP version results more tolerant versus low radiation profiles, providing higher reliability especially in the computational section of the application. Instead, when the bitflips cluster size increases due to the presence of particles with higher energies, the BRAM-based version of the controller shows higher robustness, presenting a lower total error rate and a reduced occurrence of application critical errors.
The differences observed in the error probabilities and failure modes in the application execution stages are related to the different resources functionality and distribution of the two implementations, which are reflected on different density and criticality of their programming bits within the configuration memory.
As the BRAM-based version of the self-reconfiguration controller has resulted more robust in a radiation scenario involving both single and multiple radiation-induced upsets, especially for applications requiring frequent access to the configuration

layer, it has been used as the core component for the development of a cost-effective radiation test instrumentation for the fast and efficient analysis of FPGA configuration memory sensitivity [125].
In detail, the proposed methodology enabled obtaining information on the ZYNQ 7020 device sensitivity to SBUs and MBUs considering both the initial state of the configuration memory cells and relying on a neutron generator radiation source that, thanks to its smaller size, low cost, and easy operability, strongly reduces the budget and challenges related to the usage of the highly expensive and difficultly available particles accelerators typically used for this purpose.
This has been possible through the implementation of a streamlined self-monitoring setup deployed on the device and based on the AXI_HWICAP coupled with an automated test manager to efficiently run the experiments from outside the NG chamber. In fact, by taking advantage of the awareness of the low-level frame organization has been possible implement a online semi-static self-test that selectively reads configuration data while the device is irradiated enabling the efficient gathering of the single and multiple upset probabilities from both 1 to 0 and 0 to 1 within 3 hours of radiation test and providing preliminary indications about the impact of the resources mapping and programming bits distribution on the system sensitivity.
In fact, both the performed analyses highlight how the distribution of the configuration memory bits involved in the resources programming can have a key role in the robustness of the application deployed on reconfigurable fabric and orient future developments towards the optimization of their density and the related criticality to increase design dependability by acting the low-level configuration layer organization.

Chapter 9

Conclusion and Future Developments

The research advancements presented and discussed in this dissertation have been oriented to the enhancement of the performances of dynamically reconfigurable architectures as well as on the characterization of their dependability versus radiation-induced transient errors.
In detail, as the time involved in the reconfiguration procedure represents a key parameter to be optimized to maximize reconfigurable application performances and strongly depends on the amount of configuration data involved in the procedure and the mechanism used to perform this operation, the main contributions have been centered on the optimization of these aspects.
On the other side, addressing self-reconfigurable platforms deployed in radiation environments and the susceptibility to radiation-induced soft-errors of their SRAM configuration memory, the characterization of these systems has been centered on the key component managing the in-field access to the configuration memory, either by analyzing its dependability for different radiation environments and application as well as exploiting the self-monitoring features it enables.
A common thread to the discussed contributions is represented by the link among the configuration data organization and encoding with the reconfigurable resources, which analysis and study have been the starting point for identifying both the challenges and opportunities of current reconfigurable architectures.
In fact, what all reconfigurable systems have in common is that they rely on a two-layer architecture, and especially for dynamically and partially reconfigurable applications, the interaction among the reconfigurable resource layer and the configuration memory layer defining its behavior represents an inherent and crucial part of their execution.
Thus, the first baseline contribution has consisted of an in-depth study of this link, with a major focus on 7 Series Xilinx SRAM-based FPGAs and their routing resources, which has been aided by the realization of ad-hoc tools for their analysis.

This enabled to highlight the limitation related to the current configuration memory organization, which follows a complex encoding and relies on very long configuration words, the 3,232-bit long frames, which partially configure resources that can be distant and unrelated in the resource layer introducing a high data overhead, especially in the case of frequent and detailed in-field upgrades.
On the other side, this study allowed identifying novel strategies to reduce the configuration time overhead by acting on this link and represented a useful asset for the efficient evaluation of the configuration memory sensitivity and the relative application failure modes.
In fact, the awareness of the encoding of the configuration frames programming the routing resources enabled the development of the first Frame-driven Routing Algorithm (FeDRA) that consists of a generalized approach able to achieve an average optimization of 35% of the reconfiguration data and time in Xilinx SRAM-based FPGAs by routing circuits relying on an ad-hoc policy able to weight and evaluate routing resources according to the configuration data overhead they introduce.
On the other side, when dynamically and partially reconfigurable applications are oriented to detailed and bit-level upgrades, the frame bottleneck and the overhead introduced by the mechanism to perform the reconfiguration procedure in commercial FPGAs becomes unjustified.
In this view, the Reconfigurable Multipotent (ReM) Cell has been designed as the basic reconfigurable element of a novel and distributed architectural model for reconfigurable platforms able to perform fast and bit-level in-field repurposing within a single clock cycle while minimizing the amount of involved configuration data.
The awareness of configuration memory encoding and organization has enabled and aided the efficient characterization of dynamically and partially reconfigurable SRAM-based FPGAs deployed in radiation environments.
In fact, although the high computational capability coupled with the feature of performing in-field adjustments has increasingly broadened their usage in high-performance applications operating in presence of radiation, the SRAM cells sensitivity to radiation-induced transient soft-errors is an inherent and well-known characteristic of SRAM-based FPGAs and must be properly evaluated and characterized before the application deployment to ensure its readiness to its mission.
However, the possibility enabled by dynamic and partial reconfiguration to access and update the configuration memory at run-time provides many opportunities for the implementation of in-field fault detection and self-repairing techniques.
For this reason, the performed analysis has been centered on the self-configuration controller, which consists of a dedicated circuitry implemented in the reconfigurable logic to manage the interactions among the configuration and application layers.
In fact, the self-configuration controller is the key component enabling the execution of run-time performance-oriented optimizations in dynamically reconfigurable modular systems as well as the implementation of self-monitoring and self-repairing techniques for increasing application reliability.

Thus, the first part of the analysis has been dedicated to the evaluation of the sensitivity and the failure modes of different implementations of the self-reconfiguration controller versus different radiation environments. In fact, relying on the acquired knowledge about configuration memory organization, it has been possible to emulate different radiation scenarios through dedicated fault injection campaigns.
This enabled to evaluate the controller implementations deployability according to the target radiation environment and their operational goal while providing insights on the impact of their configuration bits distribution in the application dependability.
Accordingly, the controller implementation identified as more reliable for frequent configuration memory accesses in radiation environments causing both single and multiple bit upsets has been used for self-monitoring purposes in the instrumentation of a streamlined and efficient approach oriented to minimize the cost and the complexity involved in typical FPGA radiation testing.
In this case, thanks to self-reconfiguration controller self-monitoring capabilities coupled with the configuration memory awareness and manipulation, an automated and online self-testing setup has been instrumented for the fast and cost-efficient radiation analysis under a neutron generator source. The proposed setup strongly reduces the time and the cost of typical radiation testing experiments while providing useful preliminary data about the impact of the programming bits distribution of the mapped resources on the system sensitivity.
Looking ahead of the aforementioned achievements and the possible advancements specific to each contribution, these works open the way for further focusing on the link among resource and configuration layer.
In fact, this knowledge can be deepened and exploited for the development of novel approaches able to transparently increase application performance and dependability in commercial devices by cleverly mapping reconfigurable resources during the development process. In fact, future directions can be oriented to the introduction of dedicated techniques aimed at further reducing the configuration data overhead involved in in-field reconfiguration as well as optimally distributing configuration settings for increasing design dependability or to ease recovery mechanisms.
Furthermore, as the opportunities provided by further investigating different tradeoffs among granularity and reconfigurability have been highlighted, novel architectural solutions going toward an increased heterogeneity and a more efficient ratio among the programmable resource area and the relative configuration settings can be explored. In fact, future directions on novel architectures can be oriented to further increase reconfiguration granularity and performances as well as providing reconfigurable fabric an higher reliability through the discretization of its configuration registers.

Publications

Journals

1. C. De Sio, S. Azimi, L. Bozzoli, B. Du, L. Sterpone, "Radiation-induced Single Event Transient Effects during the Reconfiguration Process of SRAM-based FPGAs", *Microelectronics Reliability* 100, 2019

2. L. Bozzoli, L. Sterpone, "An Optimized Frame-Driven Routing Algorithm for Reconfigurable SRAM-based FPGAs", *IEEE Access*, vol. 8, 2020, pp. 116226-116238

Conferences

1. L. Bozzoli, L. Sterpone, "Self rerouting of dynamically reconfigurable SRAM-based FPGAs", in *IEEE Adaptive Hardware and Systems NASA/ESA Conference (AHS)*, 2017, pp. 77-84

2. L. Bozzoli, L. Sterpone, "COMET: a Configuration Memory Tool to Analyze, Visualize and Manipulate FPGAs Bitstream", in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems,* VDE, 2018, pp. 1-4

3. Bozzoli, L. Sterpone, "Fast partial reconfiguration on SRAM-based FPGAs: A frame-driven routing approach", in *14th International Symposium on Applied Reconfigurable Computing (ARC)*, 2018, pp. 319-330

4. L. Bozzoli, L. Sterpone, "IbIS: Interface-based Interconnection Structure for Dynamically Reconfigurable FPGAs", in *IEEE International Symposium on Circuits and Systems (ISCAS),* 2018, pp. 1-5

5. L. Bozzoli, L. Sterpone, "MATS**: An On-Line Testing Approach for Reconfigurable Embedded Memories", in *31th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology (DFT)*, 2018, pp. 1-6

6. L. Sterpone, S. Azimi, L. Bozzoli, B. Du, T. Lange, M. Glorieux, D. Alexandrescu, C. Boatella Polo, D. Merodio Codinachs, "A Novel Error Rate Estimation Approach for UltraScale+ SRAM-based FPGAs", in *IEEE Adaptive Hardware and Systems NASA/ESA Conference (AHS)*, 2018, pp. 120-126

7. L. Bozzoli, C. De Sio, L. Sterpone , C. Bernardeschi, "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing", in *International Symposium on Rapid System Prototyping (RSP)*, 2018, pp.70-75

8. L. Bozzoli, L. Sterpone, "ReM: A Reconfigurable Multipotent Cell for New Distributed Reconfigurable Architectures", in *15th International Symposium on Applied Reconfigurable Computing (ARC)*, 2019, pp. 295-304

9. S. Odintsov, L. Bozzoli, C. De Sio, L. Sterpone, A. Jutman, "A new FPGA-based Detection Method for Spurious Variations in PCBA Power Distribution Network", in *22th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2019, pp. 1-6

10. L. Sterpone, L. Bozzoli, C. De Sio, B. Du, S. Azimi, "A new Method for the Analysis of Radiation-induced Effects in 3D VLSI Face-to-Back LUTs", in *IEEE International Conference on Synthesis, modeling, analysis and Simulation methods and applications to circuit design (SMACD)*, 2019, pp. 205-208

11. L. Sterpone, L. Bozzoli, C. De Sio, B. Du, S. Azimi, "Analysis of Radiation-induced SETs in 3D VLSI Face-to-Back LUTs", in *30th IEEE Radiation and its Effects on Components and Systems (RADECS)*, 2019

12. B. Du, S. Azimi, C. De Sio, L. Bozzoli, L. Sterpone, "On the Reliability of Convolutional Neural Network Implementation on SRAM-based FPGA", in *32nd IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology (DFT)*, 2019, pp. 1-6

13. L. Bozzoli, L. Sterpone, "Soft-Error Analysis of Self-reconfiguration Controllers for Safety Critical Dynamically Reconfigurable FPGAs", in *16th International Symposium on Applied Reconfigurable Computing (ARC)*, 2020, pp. 84-96

14. L. Bozzoli , C. De Sio, B. Du, and L. Sterpone, "A Neutron Generator Testing Platform for the Radiation Analysis of SRAM-based FPGAs", in *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2021, pp. 1–5

Bibliography

[1] G. Estrin, "Reconfigurable computer origins: the UCLA fixed-plus-variable (F+V) structure computer", 4, vol. 24, 2002, pp. 3–9. DOI: 10.1109/MAHC.2002.1114865.

[2] ——, "Organization of Computer Systems-the Fixed Plus Variable Structure Computer", in *Managing Requirements Knowledge, International Workshop on*, vol. 1, Los Alamitos, CA, USA: IEEE Computer Society, May 1960, p. 33. DOI: 10.1109/AFIPS.1960.28. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/AFIPS.1960.28.

[3] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable Computing Architectures", *Proceedings of the IEEE*, vol. 103, no. 3, pp. 332–354, 2015. DOI: 10.1109/JPROC.2014.2386883.

[4] W. A. Najjar and P. Ienne, "RECONFIGURABLE COMPUTING Introduction", *Ieee Micro*, vol. 34, no. 1, pp. 4–6, 2014.

[5] A. DeHon and J. Wawrzynek, "Reconfigurable computing: what, why, and implications for design automation", in *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, 1999, pp. 610–615. DOI: 10.1109/DAC.1999.782016.

[6] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software", *ACM Computing Surveys (csuR)*, vol. 34, no. 2, pp. 171–210, 2002.

[7] Zain-ul-Abdin and B. Svensson, "Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing", *Microprocessors and Microsystems*, vol. 33, no. 3, pp. 161–178, 2009, ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2008.10.003.

[8] N. Abel, S. Manz, F. Grull, and U. Kebschull, "Increasing Design Changeability Using Dynamic Partial Reconfiguration", *IEEE Transactions on Nuclear Science*, vol. 57, no. 2, pp. 602–609, 2010. DOI: 10.1109/TNS.2009.2037916.

[9] T. G. Cervero, J. Caba, S. López, J. D. Dondo, R. Sarmiento, F. Rincón, and J. López, "A Scalable and Dynamically Reconfigurable FPGA-Based Embedded System for Real-Time Hyperspectral Unmixing", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2894–2911, 2015. DOI: 10.1109/JSTARS.2014.2347075.

[10] C. Carmichael, "Correcting Single-Event Upsets Through Virtex Partial Configuration", vol. Xilinx Tech. report, XAPP216 (v1.0), 2000.

[11] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods", *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005. DOI: 10.1049/ip-cdt:20045086.

[12] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, and A. Thakoor, "Reconfigurable VLSI architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-oriented chips", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 227–232, 2001. DOI: 10.1109/92.920839.

[13] J. A. Walker, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "PAnDA: A Reconfigurable Architecture that Adapts to Physical Substrate Variations", *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1584–1596, 2013. DOI: 10.1109/TC.2013.59.

[14] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications", *ACM Comput. Surv.*, vol. 47, no. 2, Jan. 2015. DOI: 10.1145/2671181.

[15] M. Wirthlin, "High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond", *Proceedings of the IEEE*, vol. 103, no. 3, pp. 379–389, 2015. DOI: 10.1109/JPROC.2015.2404212.

[16] N. Battezzati, L. Sterpone, and M. Violante, *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*. Springer, 2011, ISBN: 9781441975966. [Online]. Available: https://books.google.it/books?id=UyAIswEACAAJ.

[17] D. Sabena, L. Sterpone, M. Schölzel, T. Koal, H. T. Vierhaus, S. Wong, R. Glein, F. Rittner, C. Stender, M. Porrmann, and J. Hagemeyer, "Reconfigurable high performance architectures: How much are they ready for safety-critical applications?", in *2014 19th IEEE European Test Symposium (ETS)*, 2014, pp. 1–8. DOI: 10.1109/ETS.2014.6847820.

[18] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications", *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–39, 2018.

[19] Xilinx, *7 Series FPGAs Data Sheet: Overview - DS180 (v2.6.1)*. 2020.

[20] ——, *7 Series DSP48E1 Slice User Guide UG479 - (v1.10)*. 2018.

[21] ——, *7 Series FPGAs Memory Resources User Guide - UG473 (v1.14)*. 2019.

[22] ——, *Zynq-7000 SoC Data Sheet: Overview - DS190 (v1.11.1)*. 2018.

[23] ——, *7 Series FPGAs Configuration User Guide - UG470 (v1.13.1)*. 2018.

[24] R. Chéour, S. Khriji, D. E. Houssaini, M. Baklouti, M. Abid, and O. Kanoun, "Recent Trends of FPGA Used for Low-Power Wireless Sensor Network", *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 10, pp. 28–38, 2019. DOI: 10.1109/MAES.2019.2901134.

[25] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The Promise of High-Performance Reconfigurable Computing", *Computer*, vol. 41, no. 2, pp. 69–76, 2008. DOI: 10.1109/MC.2008.65.

[26] E. Vansteenkiste, B. A. Farisi, K. Bruneel, and D. Stroobandt, "TPaR: Place and Route Tools for the Dynamic Reconfiguration of the FPGA's Interconnect Network", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 3, pp. 370–383, 2014. DOI: 10.1109/TCAD.2013.2291659.

[27] L. Bozzoli and L. Sterpone, "An Optimized Frame-Driven Routing Algorithm for Reconfigurable SRAM-Based FPGAs", *IEEE Access*, vol. 8, pp. 116 226–116 238, 2020.

[28] R. Katz, K. LaBel, J. J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift, "Radiation effects on current field programmable technologies", *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 1945–1956, 1997. DOI: 10.1109/23.658966.

[29] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics", *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 583–602, 2003. DOI: 10.1109/TNS.2003.813129.

[30] H. Quinn, "Challenges in Testing Complex Systems", *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, 2014. DOI: 10.1109/TNS.2014.2302432.

[31] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs", vol. Xilinx Tech. report, XAPP197 (v1.0), 2001.

[32] K. S. Morgan, D. L. McMurtrey, B. H. Pratt, and M. J. Wirthlin, "A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs", *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2065–2072, 2007. DOI: 10.1109/TNS.2007.910871.

[33] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis", *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2259–2266, 2008. DOI: 10.1109/TNS.2008.2001422.

[34] Mirsky and DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources", in *1996 Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, 1996, pp. 157–166. DOI: 10.1109/FPGA.1996.564808.

[35] T. Miyamori and K. Olukotun, "REMARC : Reconfigurable Multimedia Array Coprocessor", *IEICE Transactions on Information and Systems*, vol. 82, pp. 389–397, 1998.

[36] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: a computational fabric for software circuits and general-purpose programs", *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002. DOI: 10.1109/MM.2002.997877.

[37] H. Singh, Ming-Hau Lee, Guangming Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications", *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000. DOI: 10.1109/12.859540.

[38] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor", in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*, 1997, pp. 12–21. DOI: 10.1109/FPGA.1997.624600.

[39] Xilinx, *Programmable Logic Data Book*. 1996.

[40] ——, *Virtex-II Platform FPGA User Guide - UG002 (v2.2)*. 2007.

[41] ——, *Virtex-II Pro and Virtex-II Pro X FPGA User Guide - UG012 (v4.2)*. 2007.

[42] ——, *Partial Reconfiguration User Guide - UG702 (v13.3)*. 2011.

[43] ——, *Virtex-4 FPGA User Guide - UG070 (v2.6)*. 2008.

[44] ——, *Virtex-5 FPGA User Guide - UG190 (v5.2)*. 2009.

[45] ——, *Virtex-6 FPGA Configuration User Guide - UG360 (v3.9)*. 2015.

[46] ——, *Axi Refernce Guide - UG1037 (v4.0)*. 2017.

[47] ——, *Vivado Design Suite User Guide, Partial Reconfiguration - UG909 (v2018.1)*. 2018.

[48] ——, *UltraScale Architecture Configuration User Guide - UG570 (v1.13)*. 2020.

[49] ——, *Key Attributes of an Intelligent IIoT Edge Platform - WP493 (v1.0)*. 2017.

[50] ——, *Versal: The First Adaptive Compute Acceleration Platform (ACAP) - WP505 (v1.1.1)*. 2020.

[51] Intel, *Stratix V Device Overview - SV51001*. 2020.

[52] ——, *Intel Arria 10 Device Overview - A10-OVERVIEW*. 2020.

[53] I. Simon Stanley, *Capitalizing on Synergies across 5G, Edge, and Cloud Platforms - A Heavy Reading white paper produced for Intel*. 2020.

[54] National Semiconductor, *Configurable Logic Array (CLAy) Data Sheet*. 1993.

[55] Atmel, *AT40K05, AT40K10, AT40K20, AT40K40 Datasheet*. 2013.

[56] Tabula, *ABAX Product Brief. Technical Report*. 2010.

[57] Menta, *Menta eFPGA products brief: Menta eFPGA IP - THE Solution for Embedded FPGA*. 2020.

[58] Achronix, *Speedcore eFPGA Product Brief (PB028)*. 2020.

[59] V. Chamola, S. Patra, N. Kumar, and M. Guizani, "FPGA for 5G: Reconfigurable Hardware for Next Generation Communication", *IEEE Wireless Communications*, vol. 27, no. 3, pp. 140–147, 2020. DOI: 10.1109/MWC.001.1900359.

[60] D. Stroobandt, A. L. Varbanescu, C. B. Ciobanu, M. Al Kadi, A. Brokalakis, G. Charitopoulos, T. Todman, X. Niu, D. Pnevmatikatos, A. Kulkarni, E. Vansteenkiste, W. Luk, M. D. Santambrogio, D. Sciuto, M. Huebner, T. Becker, G. Gaydadjiev, A. Nikitakis, and A. J. W. Thom, "EXTRA: Towards the exploitation of eXascale technology for reconfigurable architectures", in *2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2016, pp. 1–7. DOI: 10.1109/ReCoSoC.2016.7533896.

[61] L. Bozzoli and L. Sterpone, "IbIS: Interface-based Interconnection Structure for Dynamically Reconfigurable FPGAs", in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2018, pp. 1–5.

[62] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert, "Design Optimizations for Tiled Partially Reconfigurable Systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1048–1061, 2011. DOI: 10.1109/TVLSI.2010.2044902.

[63] M. S. Reorda, L. Sterpone, and A. Ullah, "An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems", *IEEE Transactions on Computers*, vol. 66, no. 6, pp. 1022–1033, 2017. DOI: 10.1109/TC.2016.2607749.

[64] M. G. Gericota, L. F. Lemos, G. R. Alves, and J. M. Ferreira, "On-Line Self-Healing of Circuits Implemented on Reconfigurable FPGAs", in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, 2007, pp. 217–222. DOI: 10.1109/IOLTS.2007.50.

[65] L. Sterpone, M. Porrmann, and J. Hagemeyer, "A Novel Fault Tolerant and Runtime Reconfigurable Platform for Satellite Payload Processing", *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1508–1525, 2013. DOI: 10.1109/TC.2013.80.

[66] A. Pérez, A. Rodríguez, A. Otero, D. G. Arjona, Á. Jiménez-Peralo, M. Á. Verdugo, and E. De La Torre, "Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation", *IEEE Access*, vol. 8, pp. 59 891–59 905, 2020. DOI: 10.1109/ACCESS.2020.2983308.

[67] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. De la Torre, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework", *Sensors*, vol. 18, no. 6, p. 1877, 2018.

[68] H. Quinn, D. Roussel-Dupre, M. Caffrey, P. Graham, M. Wirthlin, K. Morgan, A. Salazar, T. Nelson, W. Howes, E. Johnson, J. Johnson, B. Pratt, N. Rollins, and J. Krone, "The Cibola Flight Experiment", *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 1, Mar. 2015, ISSN: 1936-7406. DOI: 10.1145/2629556.

[69] J. Alme, Y. Andres, H. Appelshauser, S. Bablok, N. Bialas, R. Bolgen, U. Bonnes, R. Bramm, P. Braun-Munzinger, R. Campagnolo, P. Christiansen, A. Dobrin, C. Engster, D. Fehlker, P. Foka, U. Frankenfeld, J. Gaardhòje, C. Garabatos, P. Glassel, C. Gutiérrez, P. Gros, H. Gustafsson, H. Helstrup, M. Hoch, M. Ivanov, R. Janík, A. Junique, A. Kalweit, R. Keidel, S. Kniege, M. Kowalski, D. Larsen, Y. Lesenechal, P. Lenoir, N. Lindegaard, C. Lippmann, M. Mager, M. Mast, A. Matyja, M. Munkejord, L. Musa, B. S. Nielsen, V. Nikolic, H. Oeschler, E. K. Olsen, A. Oskarsson, L. Osterman, M. Pikna, A. Rehman, G. Renault, R. Renfordt, S. Rossegger, D. Ròhrich, K. Roed, M. Richter, G. Rueshmann, A. Rybicki, H. Sann, H. Schmidt, M. Siska, B. Sitar, C. Soegaard, H. Soltveit, D. Soyk, J. Stachel, H. Stelzer, E. Stenlund, R. Stock, P. Strmeň, I. Szarka, K. Ullaland, D. Vranic, R. Veenhof, J. Westergaard, J. Wiechula, and B. Windelband, "The ALICE TPC, a large 3-dimensional tracking device with fast readout for ultra-high multiplicity

events", *Nuclear Instruments and Methods in Physics Research Section A-accelerators Spectrometers Detectors and Associated Equipment*, vol. 622, pp. 316–367, 2010.

[70] A. P. Fournaris, C. Alexakos, C. Anagnostopoulos, C. Koulamas, and A. Kalogeras, "Introducing Hardware-Based Intelligence and Reconfigurability on Industrial IoT Edge Nodes", *IEEE Design Test*, vol. 36, no. 4, pp. 15–23, 2019. DOI: 10.1109/MDAT.2019.2908547.

[71] Amazon, *Amazon EC2 F1 Instances*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls, (Accessed: 30.12.2020).

[72] Intel, *Microsoft's Bing Intelligent Search with Intel FPGAs*. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/b/bing-intelligence-search-with-intel-fpgas.html, (Accessed: 30.12.2020).

[73] N. Tarafdar, N. Eskandari, T. Lin, and P. Chow, "Designing for FPGAs in the Cloud", *IEEE Design Test*, vol. 35, no. 1, pp. 23–29, 2018. DOI: 10.1109/MDAT.2017.2748393.

[74] EuroEXA, *Redefining HPC, from infrastructure to application*. [Online]. Available: https://euroexa.eu/overview, (Accessed: 31.12.2020).

[75] R. Ammendola, A. Biagioni, P. Cretaro, O. Frezza, F. L. Cicero, A. Lonardo, M. Martinelli, P. S. Paolucci, E. Pastorelli, F. Simula, P. Vicini, G. Taffoni, J. A. Pascual, J. Navaridas, M. Luján, J. Goodacree, N. Chrysos, and M. Katevenis, "The Next Generation of Exascale-Class Systems: The ExaNeSt Project", in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 510–515. DOI: 10.1109/DSD.2017.20.

[76] Y. Beilliard, M. Godard, A. Ioannou, A. Damianakis, M. Ligerakis, I. Mavroidis, P.-Y. Martinez, D. Danovitch, J. Sylvestre, and D. Drouin, "FPGA-based Multi-Chip Module for High-Performance Computing", *arXiv:1906.11175 - preprint*, 2019.

[77] I. Mavroidis, I. Papaefstathiou, L. Lavagno, D. S. Nikolopoulos, D. Koch, J. Goodacre, I. Sourdis, V. Papaefstathiou, M. Coppola, and M. Palomino, "ECOSCALE: Reconfigurable computing and runtime system for future exascale systems", in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 696–701.

[78] P. Kannan and S. Sivaswamy, "Performance driven routing for modern FPGAs invited paper", in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–6. DOI: 10.1145/2966986.2980082.

[79] M. Gort and J. H. Anderson, "Reducing FPGA Router Run-Time through Algorithm and Architecture", in *2011 21st International Conference on Field Programmable Logic and Applications*, 2011, pp. 336–342. DOI: 10.1109/FPL.2011.67.

[80] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117. DOI: 10.1109/FPGA.1995.242049.

[81] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, K. Kent, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 1–23, 2011.

[82] S. Mukherjee and S. Roy, "SAT Based Multi Pin Net Detailed Routing for FPGA", in *2010 International Symposium on Electronic System Design*, 2010, pp. 141–146. DOI: 10.1109/ISED.2010.35.

[83] K. Heyse, K. Bruneel, and D. Stroobandt, "Mapping logic to reconfigurable FPGA routing", in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 2012, pp. 315–321.

[84] Ching-Dong Chen, Yuh-Sheng Lee, A. C. .-. Wu, and Youn-Long Lin, "TRACER-fpga: a router for RAM-based FPGA's", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 371–374, 1995. DOI: 10.1109/43.365127.

[85] Nak-Woong Eum, Taewhan Kim, and Chong-Min Kyung, "CeRA: a router for symmetrical FPGAs based on exact routing density evaluation", *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 829–842, 2004. DOI: 10.1109/TC.2004.20.

[86] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs", *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, 2006. DOI: 10.1109/TC.2006.82.

[87] H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris, M. Hübner, and J. Becker, "Jitpr: A framework for supporting fast application's implementation onto fpgas", *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 6, no. 2, pp. 1–12, 2013.

[88] R. Ferreira, L. Rocha, A. G. Santos, J. A. M. Nacif, S. Wong, and L. Carro, "A Runtime FPGA Placement and Routing Using Low-Complexity Graph Traversal", vol. 8, no. 2, 2015, ISSN: 1936-7406. DOI: 10.1145/2660775.

[89] L. Bozzoli and L. Sterpone, "Self rerouting of dynamically reconfigurable SRAM-based FPGAs", in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, IEEE, 2017, pp. 77–84.

[90] A. Otero, E. de la Torre, and T. Riesgo, "Dreams: A tool for the design of dynamically reconfigurable embedded and modular systems", in *2012 International Conference on Reconfigurable Computing and FPGAs*, 2012, pp. 1–8. DOI: 10.1109/ReConFig.2012.6416740.

[91] M. Amagasaki, R. Yamaguchi, M. Koga, M. Iida, and T. Sueyoshi, "An embedded reconfigurable IP core with variable grain logic cell architecture", *International Journal of Reconfigurable Computing*, vol. 2008, 2008.

[92] L. Sterpone and M. Violante, "ReCoM: A new Reconfigurable Compute Fabric Architecture for Computation-Intensive Applications", in *2006 IEEE Design and Diagnostics of Electronic Circuits and systems*, 2006, pp. 52–56. DOI: 10.1109/DDECS.2006.1649570.

[93] E. Rhod, L. Sterpone, and L. Carro, "A new RC design for mixed-grain based dynamically reconfigurable architectures", in *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, 2009, pp. 984–987. DOI: 10.1109/ICECS.2009.5410843.

[94] T. von Sydow, M. Korb, B. Neumann, H. Blume, and T. G. Noll, "Modelling and Quantitative Analysis of Coupling Mechanisms of Programmable Processor Cores and Arithmetic Oriented eFPGA Macros", in *2006 IEEE International Conference on Reconfigurable Computing and FPGA's (ReConFig 2006)*, 2006, pp. 1–10. DOI: 10.1109/RECONF.2006.307777.

[95] B. Neumann, T. von Sydow, H. Blume, and T. G. Noll, "Design flow for embedded FPGAs based on a flexible architecture template", in *2008 Design, Automation and Test in Europe*, 2008, pp. 56–61. DOI: 10.1109/DATE.2008.4484660.

[96] Xilinx, *7 Series FPGAs Configurable Logic Block User Guide - UG474 (v1.8)*. 2016.

[97] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases", in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, 2011, pp. 1–8. DOI: 10.1109/ReCoSoC.2011.5981545.

[98] Xilinx, *7 Series FPGAs Clocking Resources User Guide - UG472 (v1.14)*. 2018.

[99] L. Bozzoli and L. Sterpone, "COMET: a configuration memory tool to analyze, visualize and manipulate FPGAs bitstream", in *ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems*, VDE, 2018, pp. 1–4.

[100] L. Sterpone and L. Bozzoli, "Fast partial reconfiguration on SRAM-based FPGAs: A frame-driven routing approach", in *14th International Symposium on Applied Reconfigurable Computing, ARC 2018*, 330, vol. 10824, 2018, p. 319.

[101] L. Bozzoli and L. Sterpone, "ReM: A Reconfigurable Multipotent Cell for New Distributed Reconfigurable Architectures", in *International Symposium on Applied Reconfigurable Computing*, Springer, Cham, 2019, pp. 295–304.

[102] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results", *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, 2000. DOI: 10.1109/54.867894.

[103] J. E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959. DOI: 10.1109/TEC.1959.5222693.

[104] OpenCores, *miniMIPS*. [Online]. Available: https://opencores.org/projects/minimips, (Accessed: 03.02.2021).

[105] L. Sterpone and B. Du, "SET-PAR: place and route tools for the mitigation of single event transients on flash-based FPGAs", in *International Symposium on Applied Reconfigurable Computing*, Springer, 2015, pp. 129–140.

[106] M. Violante, L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, C. Andreani, G. Gorini, A. Pietropaolo, G. Cardarilli, S. Pontarelli, and C. Frost, "A New Hardware/Software Platform and a New 1/E Neutron Source for Soft Error Studies: Testing FPGAs at the ISIS Facility", *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 1184–1189, 2007. DOI: 10.1109/TNS.2007.902349.

[107] J. Tonfat, F. Lima Kastensmidt, P. Rech, R. Reis, and H. M. Quinn, "Analyzing the Effectiveness of a Frame-Level Redundancy Scrubbing Technique for SRAM-based FPGAs", *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 3080–3087, 2015. DOI: 10.1109/TNS.2015.2489601.

[108] D. S. Lee, M. Wirthlin, G. Swift, and A. C. Le, "Single-Event Characterization of the 28 nm Xilinx Kintex-7 Field-Programmable Gate Array under Heavy Ion Irradiation", in *2014 IEEE Radiation Effects Data Workshop (REDW)*, 2014, pp. 1–5. DOI: 10.1109/REDW.2014.7004595.

[109] B. Du, L. Sterpone, S. Azimi, D. Merodio Codinachs, V. Ferlet-Cavrois, C. Boatella Polo, R. G. Alía, M. Kastriotou, and P. Fernandez-Martínez, "Ultrahigh Energy Heavy Ion Test Beam on Xilinx Kintex-7 SRAM-Based FPGA", *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1813–1819, 2019. DOI: 10.1109/TNS.2019.2915207.

[110] R. Giordano, S. Perrella, V. Izzo, G. Milluzzo, and A. Aloisio, "Redundant-Configuration Scrubbing of SRAM-Based FPGAs", *IEEE Transactions on Nuclear Science*, vol. 64, no. 9, pp. 2497–2504, 2017. DOI: 10.1109/TNS.2017.2730960.

[111] J. Tarrillo, J. Tonfat, L. Tambara, F. L. Kastensmidt, and R. Reis, "Multiple fault injection platform for SRAM-based FPGA based on ground-level radiation experiments", in *2015 16th Latin-American Test Symposium (LATS)*, 2015, pp. 1–6. DOI: 10.1109/LATW.2015.7102494.

[112] L. Bozzoli and L. Sterpone, "Soft-Error Analysis of Self-reconfiguration Controllers for Safety Critical Dynamically Reconfigurable FPGAs", in *International Symposium on Applied Reconfigurable Computing*, Springer, Cham, 2020, pp. 84–96.

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

[114] J. R. Azambuja, G. Nazar, P. Rech, L. Carro, F. L. Kastensmidt, T. Fairbanks, and H. Quinn, "Evaluating Neutron Induced SEE in SRAM-Based FPGA Protected by Hardware- and Software-Based Fault Tolerant Techniques", *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4243–4250, 2013. DOI: 10.1109/TNS.2013.2288305.

[115] T. Beutier, E. Delage, M. Wouts, O. Serres, and P.-F. Peyrard, "Fastrad new tool for radiation prediction", in *Proceedings of the 7th European Conference on Radiation and Its Effects on Components and Systems, 2003. RADECS 2003.*, IEEE, 2003, pp. 181–183.

[116] G. Santina, P. Nieminen, H. Evansa, E. Daly, F. Lei, P. Truscott, C. Dyer, B. Quaghebeur, and D. Heynderickx, "New Geant4 based simulation tools for space radiation shielding and effects analysis", *Nuclear Physics B-Proceedings Supplements*, vol. 125, pp. 69–74, 2003.

[117] G. Asadi and M. B. Tahoori, "An analytical approach for soft error rate estimation of SRAM-based FPGAs", *Proc. Military and Aerospace Applications of Programmable Logic Devices (MAPLD)*, pp. 149–160, 2004.

[118] L. Sterpone and M. Violante, "A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs", *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2217–2223, 2005. DOI: 10.1109/TNS.2005.860745.

[119] M. Desogus, L. Sterpone, and D. M. Codinachs, "Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs", in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, 2014, pp. 111–115. DOI: 10.1109/IOLTS.2014.6873681.

[120] L. Sterpone, S. Azimi, L. Bozzoli, B. Du, T. Lange, M. Glorieux, D. Alexandrescu, C. B. Polo, and D. M. Codinachs, "A Novel Error Rate Estimation Approach forUltraScale+ SRAM-based FPGAs", in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, IEEE, 2018, pp. 120–126.

[121] Xilinx, *AXI HWICAP v3.0 LogiCORE IP Product Guide - PG134*. 2016.

[122] J. Heiner, N. Collins, and M. Wirthlin, "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing", in *2008 IEEE Aerospace Conference*, 2008, pp. 1–10. DOI: 10.1109/AERO.2008.4526471.

[123] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, "A novel high-performance fault-tolerant ICAP controller", in *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 259–263. DOI: 10.1109/AHS.2012.6268660.

[124] W. Guohua, L. Dongming, W. Fengzhou, A. Adetomi, and T. Arslan, "A tiny and multifunctional ICAP controller for dynamic partial reconfiguration system", in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 71–76. DOI: 10.1109/AHS.2017.8046361.

[125] L. Bozzoli, C. De Sio, B. Du, and L. Sterpone, "A Neutron Generator Testing Platform for the Radiation Analysis of SRAM-based FPGAs", in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2021, pp. 1–5. DOI: 10.1109/I2MTC50364.2021.9459804.

[126] Xilinx, *Partial Reconfiguration Decoupler v1.0 LogiCORE IP Product Guide - PG227*. 2016.

[127] *Neutron Generators for Analytical Purposes*, ser. Radiation Technology Reports 1. Vienna: INTERNATIONAL ATOMIC ENERGY AGENCY, 2012, ISBN: 978-92-0-125110-7.

[128] "Technology and Applications of Neutron Generators Developed by Adelphi Technology, Inc", *Physics Procedia*, vol. 60, pp. 203–211, 2014, 3rd International Meeting of the Union for Compact Accelerator-driven Neutron Sources, UCANS III, 31 July–3 August 2012, Bilbao, Spain, & 4th International Meeting of the Union for Compact Accelerator-driven Neutron Sources, UCANS IV, 23-27 September 2013, Sapporo, Hokkaido, Japan, ISSN: 1875-3892. DOI: https://doi.org/10.1016/j.phpro.2014.11.029.

[129] ThermoFisher, *P 385 Neutron Generator Specification Sheet*. [Online]. Available: https://www.thermofisher.com/order/catalog/product/10135952#/10135952, (Accessed: 21.04.2021).

[130] F. W. Technology, *ALBATROSS PULSE NEUTRON MONITOR MODEL 2080*. [Online]. Available: https://www.fwt.com/hpi/hpi_2080ds.htm, (Accessed: 21.04.2021).

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.