Exzellenzcluster
Cognitive Interaction Technology
Kognitronik und Sensorik
Prof. Dr.-Ing. U. Rückert

Run-time reconfigurable, fault-tolerant FPGA systems for space applications

zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEUR (Dr.-Ing.)

der Technischen Fakultät
der Universität Bielefeld

genehmigte Dissertation

von

M.Sc. Dario Cozzi

Referent: Prof. Dr.-Ing. Ulrich Rückert
Korreferent: Prof. Cinzia Bernardeschi

Acknowledgment

This is a fantastic and important achievement of my career. I have to thank a huge number of people for all the tips, experience, and support that they have provided me during this path.

I should primarily thank the people that have followed me during all my PhD path. I thank Prof. Dr.-Ing. Ulrich Rückert and Dr. Ing. Mario Porrmann that have supported me and they have provided me a huge amount of experiences. I thank Jens Hagemeyer for the great and unlimited support for the problems (challenge) that I have encountered in my path. I thank Sebastian Korf, for his friendship, collaboration, and support in all the work that we have made together.

I thank Prof. Cinzia Bernarderschi, who accepted to review my thesis and for the collaboration that we have had in all these years. I thank the member of the examination commission JProf. Dr. Elisabetta Chicca and Dr. rer. nat. Robert Haschke.

I thank all the KS group, all my colleagues. It has been extremely motivating to work with you all. We have shared funny moments and you contributed to expand my career knowledge. In particular, I thank Dirk Jungewelter, Martin Kaiser, and René Griessl. Special thanks to my colleague/friend Luca Cassano that has deeply contributed to my academic career.

I thank all the students that I have supervised, which have been fundamental for my research: Dominik Kleibrink, Timo Schlüssler, Martin Vorfeld, Domenico Sorrenti, Luca Santangelo, and Filippo Mascolo.

I want to thank all the little-Italy community of Bielefeld. I have shared with you a lot of fully moments. I just mention one name for everyone: Giorgio Ferrari. Of course, I thank Paderborn and all its people because it is there where all the story started.

At this point of my career, I extremely feel how all the educators, professors, and teachers have contributed in a unique way on my knowledge and preparation. Thanks to all of you.

I extremely thank my parents Enrico and Serenella, for the support, critics, and motivation in all my life. I am really grateful to you both.

Any effort has no sense without love! I thank my fantastic wife Natalia and my little princess Eleonora. You have been always by my side. This achievement is for you.

Dario Cozzi
Bielefeld, Germany

Abstract

The aim of this thesis is to investigate the use of Dynamic Partial Reconfiguration (DPR) on Commercial Off-the-Shelf (COTS) FPGAs in space applications.

Reconfigurable systems gained interest in a wide range of application fields, including aerospace, where electronic devices are exposed to a harsh working environment. COTS SRAM-based FPGA devices represent an interesting hardware platform for this kind of systems since they combine low cost with the possibility to utilize state-of-the-art processing power as well as the flexibility of reconfigurable hardware. FPGA architectures have high computational power and thanks to their ability to be reconfigured at run-time, they became interesting candidates for payload processing in space applications.

The presented Dynamic Reconfigurable Processing Module (DRPM) has been developed to investigate the use of the DPR approach for satellite payload processing. This scalable platform combines dynamically reconfigurable FPGAs with the required avionic interfaces (e.g., SpaceWire, MIL-STD-1553B, and SpaceFibre). In particular, a novel communication interface has been developed, the Heterogeneous Multi Processor Communication Interface (HMPCI), which allows inter-process communication with small latency and low memory footprint.

Current synthesis tools do not support fully the DPR capabilities of FPGAs. Therefore, this thesis introduces INDRA 2.0: an INtegrated Design flow for Reconfigurable Architectures. The key part of INDRA 2.0 is DHHarMa: a Design flow for Homogeneous Hard Macros, which generates homogeneous hard macros for Xilinx FPGAs starting from a high-level description (e.g., VHDL). In particular, the homogeneous DHHarMa router is explained in detail, providing novel terminologies and algorithms, which have enabled the generation of homogeneous routed designs. Results have been shown that Design flow for Homogeneous Hard Macros (DHHarMa) can route homogeneously a communication infrastructure utilizing just between 1% and 31% more resources than the Xilinx router, which cannot provide a homogeneous solution.

Furthermore, the permanent faults that can occur on FPGAs have been investigated. This thesis presents OLT(RE)^2: an on-line on-demand approach to testing permanent faults induced by radiation in reconfigurable systems used in space missions. The proposed approach relies on a test circuit and custom placer and router. OLT(RE)^2 exploits DPR to place the test circuits at run-time. Its goal is to test unprogrammed areas of the FPGA before using them. Experimental results of OLT(RE)^2 have shown that is possible to generate, place, and route the test circuits needed to detect on average more than 99 % of the physical wires and on average about 97 % of the programmable interconnection points of a large arbitrary region of the FPGA in a reasonable time. Moreover, the test can be run on the target device without interfering the functional behavior of the system.

Contents

1 Introduction

Reconfigurable architectures have become key implementation platforms for digital circuits in many application areas. Although Field Programmable Gate Arrays (FPGAs) progressively changed from homogeneous architectures containing identical logic cells to heterogeneous architectures containing different types of cells (e.g., Configurable Logic Block (CLB), Block Random Access Memory (Block RAM), Digital Signal Processor (DSP)), the structure itself is still regular and homogeneous.

The regularity of resources within an FPGAs is a remarkable feature, which is utilized in timing critical application domains; an example is a time-to-digital converter where the regularity of the routing structure is exploited to time difference measurement between two pulses. Dynamic Partial Reconfiguration (DPR) is another application domain, where the regularity of the FPGA resources is exploited; a hardware module that is placed in a specific area can be relocated in a different one, which has the same resources of the starting area. Thus, the regularity of the partially reconfigurable region increases the flexibility of the hardware modules placement.

Reconfigurable hardware has also gained a steadily growing interest in the domain of *space applications*. The reconfiguration of the hardware at run-time combined with the high computational power of modern FPGAs turn these devices interesting candidate for data processing in space applications. The DPR of FPGAs allows flexibility and can increase performance, improve energy efficiency, and enhance fault tolerance. The utilization of DPR on Commercial Off-the-Shelf (COTS) Xilinx FPGAs in space application is investigated in this thesis. A prototyping platform, novel tools, and fault mitigation mechanisms are provided.

The main parts of the thesis are: Dynamic Reconfigurable Processing Module (DRPM) platform that gives the possibility to exploit DPR in space applications; INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0) framework that enables the creation of DPR scenarios, which support the relocation of hardware module; On-Line Testing of Permanent Radiation Effects in Reconfigurable System (OLT(RE)^2) tool that generates testing circuits, which can detect on-line permanent faults on an FPGA.

1.1 Dynamic Reconfigurable Processing Module

Dynamic Reconfigurable Processing Module (DRPM) is a demonstrator that aims to exploit run-time adaptability and high-performance of payload processing systems, which is capable of exploiting DPR on SRAM-based FPGAs in space mission scenarios. The platform also embeds advanced tools and methods that mitigate radiation effects (e.g., Single Event Upset (SEU) and Total Ionizing Dose (TID)).

Thanks to DPR, blind and readback scrubbing are supported; the scrubbing rate can be adapted individually to different parts of the design. The demonstrator is based on the *RAPTOR-X64* prototyping environment, developed by the *Cognitronics and Sensor Systems* at *Bielefeld University*. DRPM combines dynamically reconfigurable FPGAs with avionic interfaces (e.g., SpaceWire, MIL-STD-1553B, and SpaceFibre).

The thesis provides information regarding the software infrastructure of the DRPM. Moreover, this work introduces a novel general-purpose multiprocessor communication protocol, which allows a fast and reliable communication among different processors of the system: the Heterogeneous Multi Processor Communication Interface (HMPCI). Furthermore, the DRPM proves the presented CAD tools INDRA 2.0 and OLT(RE)^2. The DRPM is a European Space Agency (ESA) funded TRP project (22424/ 09/ NL/ LvH) [37] and has been developed by the *Cognitronics and Sensor Systems* at *Bielefeld University* in collaboration with *Swiss Space Technology*, *TWT GmbH Science & Innovation*, and *Politecnico di Torino*.

1.2 INtegrated Design flow for Reconfigurable Architectures 2.0

Commercial FPGA tools do not allow exploiting all the DPR capabilities. For example, they are unable to create DPR systems that support the relocation of Partial Reconfigurable Modules (PR Modules). Therefore, this thesis introduces a new DPR flow: *INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0)*. This flow provides novel functionalities that enable the creation of a custom and advanced DPR scenarios in modern FPGAs. The key parts of this novel flow are the *Design flow for Homogeneous Hard Macros (DHHarMa)* and the *Post-Synthesis Rerouter (PSRerouter)*.

DHHarMa automatically generates homogeneous hard macros for Xilinx FPGAs starting from a high-level description, such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog HDL. Starting from HDL gives the designer the ability to quickly create and modify designs, which require being homogeneously placed and routed. The core components of DHHarMa are

a homogeneous packer, placer, and router, which generate a regular structured design, based on a user-defined FPGA partitioning. This thesis presents the homogeneous routing phase.

PSRerouter is a tool of INDRA 2.0. It can reroute nets of a design after the synthesis process. This operation is needed because vendor's Place and Route (PAR) tools do not permit inserting specific constraints on the routing policy. Therefore, PSRerouter enables to reroute specific nets of a design, allowing the creation of DPR systems that ensure Partial Reconfigurable Module (PR Module) relocation.

DHHarMa and PSRerouter rely on a custom database containing the complete description of the resources of a given FPGA: *Datastructure for Xilinx FPGAs (DXF)*. It has been created from the Xilinx *XDLRC* report and *FPGA-Edline* scripting language. This database allows having a deep interaction with the Xilinx commercial tools (e.g., Integrated Software Environment (ISE) and FPGA Editor). These tools are suitable for a wide number of Xilinx FPGAs: Virtex-4, Virtex-5, Virtex-6, Spartan-6, 7 Series (i.e., Artix-7, Kintex-7, and Virtex-7), and Zynq.

1.3 On-Line Testing of Permanent Radiation Effects in Reconfigurable System

OLT(RE)^2 is a software flow for the generation of hard macros for on-line testing and diagnosing of permanent faults due to radiation in SRAM-based FPGAs used in space missions. Radiations in the atmosphere may damage electronic devices employed in space systems. In particular, radiations may induce both instantaneous and long-term damages. Instantaneous damages are typically *SEUs* (i.e., modifications of the content of memory elements in the device) and Single Event Transition (SET) (i.e., transient undesired electrical impulses). Differently, the long-term damages induced by radiations are caused by *TID*, i.e., the accumulation of charge trapped in the oxide layer of transistors in CMOS circuits.

In the last decade, many works focused SEUs, which are soft errors that can occur in space missions. Nevertheless, permanent faults (e.g., caused by TIDs) need to be considered in present/future reconfigurable systems designed for space missions. OLT(RE)^2 connects ideas and background from different inventors that aimed mitigation and detection of SEU effects to detect permanent faults caused by TID as well. The test approach of OLT(RE)^2 exploits the DPR mechanism provided by modern SRAM-based FPGAs.

The testing technique is meant to be applied on-line and on-demand to detect permanent faults in reconfigurable systems. On the one hand, it can help designers in making use of high-performance unreliable COTS FPGAs viable for space applications; on the other hand, it can help the increase of low-cost application

scenarios systems where high-end radiation-hardened devices are not affordable.

The presented approach is compatible with a wide number of Xilinx FPGAs: Virtex-4, Virtex-5, Virtex-6, and Spartan-6. Moreover, the flow has been validated on the DRPM platform. OLT(RE)^2 has been partially funded by *European Space Agency (ESA)* and developed by the *Cognitronics and Sensor Systems Group* at *Bielefeld University*, in collaboration with *Pisa University* and *Politecnico di Torino*.

1.4 Organization

Chapter 2 provides the needed background concepts of the presented thesis work. The Xilinx FPGA architecture is introduced, by providing terminology and details of the different FPGA families. A description of the DPR mechanism is given, which is a concept strongly used in all the aspects of this thesis. Successively, the radiation effects on FPGAs are presented, giving information regarding the possible errors that can be induced in the routing structure. Finally, since the thesis provides new tools that can be utilized with the vendor ones, the Xilinx tools are presented, focusing on their properties and integration methods.

Chapter 3 presents the state of the art of this thesis. First, it is explained why the Xilinx tools do not allow exploiting DPR capability of the FPGAs completely; then, related works are investigated as well. Moreover, the DRPM motivations are presented, highlighting the novel features that this new platform introduces compared to the current reconfigurable platforms. Finally, testing of FPGAs background is provided, focusing on the leak of on-line testing of FPGAs in space applications.

Chapter 4 describes the DRPM platform, focusing on its hardware and software properties. The base platform (*RAPTOR-X64*) and the modular boards (*DB-V4* and *DB-SPACE*) are presented. Moreover, a deep description of its inter-processors communication protocol called *Heterogeneous Multi Processor Communication Interface (HMPCI)* is provided. Finally, the DRPM GUI is presented, which allows configuring and validating the overall platform.

Chapter 5 presents INDRA 2.0. This flow allows exploiting the full capability of DPR into modern Xilinx FPGAs. The flow supports the integration with the official Xilinx tools. Specifically, INDRA 2.0 enables to create scenarios in which reconfigurable modules can be relocated; the flow relies on a custom *Datastructure for Xilinx FPGAs (DXF)* created with the XDL-reports, which contains the structure information of the FPGAs. A special Rerouter tool has been implemented, which allows unrouting/routing specific design nets to enable module relocation.

Chapter 6 provides a detailed description of DHHarMa. Its key parts (i.e., Homogeneous Packer, Homogeneous Placer and Homogeneous Router) are presented. It gives an extensive explanation of the Homogeneous Router parts, considering the novel concepts that allow generating a homogeneous routed design. Moreover,

it provides an analysis of the routing structure of different Xilinx FPGA families. Finally, case studies and results are discussed.

Chapter 7 presents OLT(RE)^2. First, the flow and its innovation in on-line testing are discussed. Then, a dedicated testing circuit is explained, which permits performing the on-board test on an FPGA device. The chapter also introduces new concepts and solutions to categorize the different FPGA's routing resources. Furthermore, a detailed explanation of the PAR algorithm (*U-TURN*) is provided. Finally, the performance and the test coverage of the testing approach on different devices is considered.

2 Background

Field-Programmable Gate Arrays (FPGAs) are prefabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system [15; 91]. Initially, these devices provided simple logic functions. Nowadays, FPGAs can implement complex systems in one chip; hence, they are used in a growing number of applications [38]. The biggest available FPGA, the Xilinx Virtex UltraScale VU440, integrates 5.5 M logic blocks and 2,880 DSP blocks [140].

In contrast to Application Specific Integrated Circuits (ASICs), which are designed for specific applications, FPGAs are configured after their fabrication. Furthermore, they can be reconfigured multiple times. The configuration is done starting from a hardware description language (HDL) of a certain digital circuit, which is compiled to a bitstream and downloaded to the FPGA.

One of the main advantages of using FPGAs is the decreasing time-to-market required for an application. FPGAs allow designers to concentrate on the development of applications, without focusing on the device fabrication problems. Moreover, the target device of an application is already available since the beginning of the development phase.

On the contrary, when the target implementation is an ASIC, the device is available just at the end of the fabrication process. For these reasons, ASIC implementation is usually advantageous when a large scale production is planned. Therefore, FPGAs can be a much cheaper and faster solution, when a specific and small-scale task is required.

SRAM-based FPGAs allow designers to reuse and reconfigure a device several times; hence, reconfigurability gives different remarkable advantages, such as:

- **Testing**: FPGAs can be utilized as prototype platforms [59], allowing a deeper and faster validation of an application (compared to the simulation software).

- **Application Update**: the use of a reprogrammable device can extend the lifetime of a product. Nowadays, algorithms and standards are chaining fast (e.g., video decoding algorithms, communication protocols); updating an algorithm can allow using a product longer and more efficiently.

- **Time-share Applications**: to reduce cost and space of a device, different tasks can be programmed into a single device, at various instance of time.

- **Partial-Reconfiguration**: some SRAM-based devices offer the property to reconfigure just parts of an FPGA. In particular, during a partial reconfiguration, the non-reconfigured part can still operate without being interrupted.

FPGAs are utilized in space missions as well. In this field, the final application has specific targets and a limited number of exemplars; the cost of implementing an ASIC for each new satellite can turn in a higher final cost. Thereby, FPGAs are extraordinary increasing in space market; furthermore, they can offer all the advantages of device reprogrammability (as describe above).

However, due to the harsh-environment they operate in, FPGAs have to respect special requirements and be fault tolerant to radiations. Differently from a ground application, a critical fault on space applications cannot be solved by analyzing the components with a typical repairing process (e.g., unmount the device, exchange a broken part, execute off-line tests).

In the following, the organization of the chapter is presented. Section 2.1 provides the FPGA's background information to understand the implementation part of this thesis (i.e., FPGA architecture, different Xilinx devices, Dynamic Partial Reconfiguration). Section 2.2 focuses on DPR, introducing important concepts, such as FPGA partitioning and communication infrastructures for reconfigurable systems. Section 2.4 introduces the space harsh-environment, analyzing temporary and permanent faults effects that radiations can cause. Then, the radiation effects on SRAM-based FPGAs are considered, focusing on the routing resources effects of a fault (e.g., antenna, open). Finally, Section 2.3 presents the Xilinx ISE software design flow, highlighting the parts that are mostly used in this thesis.

2.1 SRAM-based FPGA Architecture

FPGAs are mainly divided into three categories: the first, *antifuse FPGAs*, consists of electronically programmable configuration memories that can be programmed only once; the second, flash-based FPGAs, consist of devices which are configured with flash configuration memory; the third, *SRAM-based FPGAs*, comprises devices that are based on a Static Random Access Memory (SRAM) configuration memory, which controls the FPGA configuration. Hence, SRAM-based FPGAs can be programmed multiple times [5]. When FPGAs were introduced in the 80s, antifuse devices were preferred, thanks to their greater stability compared to the first SRAM-based models. However, in the next years, the SRAM memories became more stable, allowing a fast spreading of SRAM-based FPGAs.

Antifuse, SRAM-based, and flash-based FPGAs have the same high-level architecture, which consists mainly of three components: Logic Blocks, Interconnection matrices (INTs) and Input Output Blocks (IOBs). Figure 2.1 shows how these components are interconnected.

In the following, the FPGA's components are presented:

- *Logic Blocks*: are the main logical resources that implement sequential and combinatorial circuits; Xilinx calls them Configurable Logic Blocks (CLBs). The structure of a CLB is hierarchically divided into logical cells (called *slices*) that contain *Look Up Tables (LUTs)*. In FPGAs of different manufacturers, the number of logical cells within a Logic Block can differ; in current Xilinx FPGAs, the number of slices can be either 2 or 4 according to the FPGA family.

- *Input Output Blocks (IOBs)*: provide outside connections of the FPGA. These blocks are generally placed at the borders of the FPGA, and they are used to getting the signals into and outwards the FPGA.

- *Interconnection matrices (INTs)*: all the FPGA logic blocks are connected among them with a complex general routing structure. With an INT, also called switch matrix, it is possible to drive signals within the FPGA (e.g., connect two different CLB blocks). The routing is provided activating the Programmable Interconnection Points (PIPs), which are located within the switch matrix. All INTs are connected among them utilizing a complex structure of fixed connections, which is presented in detailed in Section 2.1.5 and Section 6.1.

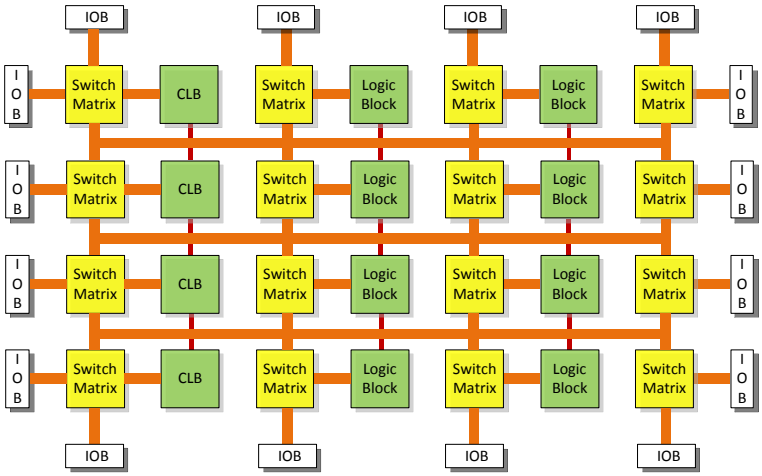


Figure 2.1: The general architecture of a Xilinx FPGA.

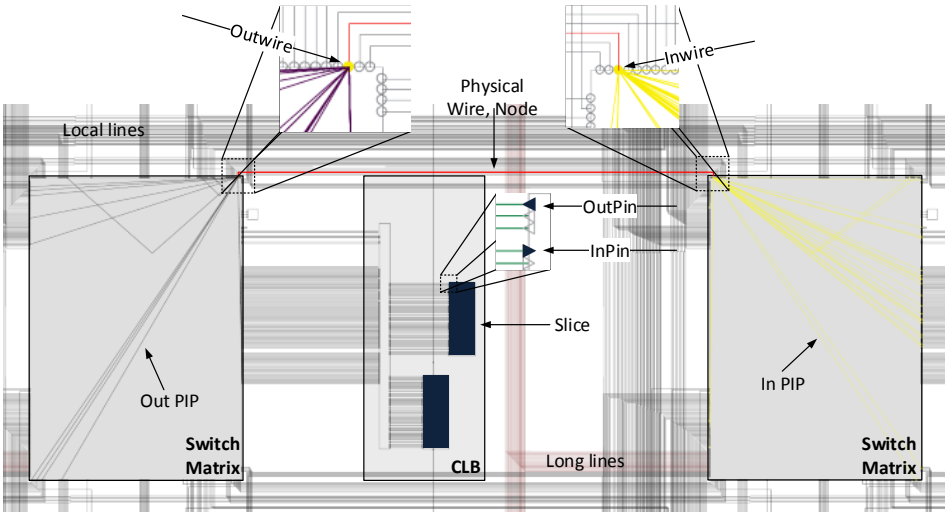


Figure 2.2: A summary of the FPGA resources terminology. A screenshot of FPGA Editor [128] has been utilized.

2.1.1 Terminology

This thesis focuses on the low-level architecture of the FPGA. In the following, basic terminologies and concepts are provided, which are extensively used in the rest of the thesis. It is worth to mention that these terminologies are strongly related to Xilinx FPGAs, which are the target devices for this work.

Figure 2.2 shows a screenshot of a Xilinx FPGA taken from the FPGA Editor tool (presented in Section 2.3.2). The screenshot shows a portion of an FPGA, highlighting the main components considered in this works; according to the depicted in Figure 2.1, this picture shows a CLB, two INTs, and physical connections lines.

This supports the following terminology:

- *Slice*: the basic logic building block of an FPGA. A slice includes the configurable resources for the implementation of Boolean functions, as well as flip-flops and carry propagation logic.
- *Physical wire (PW)*: a hard-wired non-configurable interconnection between either two switch matrices or a CLB and a switch matrix.
- *Pin*: a connection point between a slice and a physical wire (PW). In case the direction of the signal goes into the slice, the Pin is called *inPin*; *outPin* otherwise.

Table 2.1: Clock Region (CR) properties of Xilinx FPGAs.

	Virtex-4 [145]	Virtex-5 [148]	Spartan-6 [138]	Virtex-6 [150]	7 Series, Zynq [122]
Number of CRs	8-24	6-24	8-24	6-18	4-24
Height of a CR	16 CLB	20 CLB	16 CLB	40 CLB	50 CLB
Clocks within a CR	8	10	16	12	12

- *Wire*: a connection point between a PW and a switch matrix. In case the signal direction goes into the INT, the Wire is called *inWire*; *outWire* otherwise.
- *Programmable Interconnection Point (PIP)*: a configurable connection between two wires belonging to the same INT. It is worth noting that multiple PIPs are connected to the same inWire as well as multiple PIPs are attached to the same outWire. PIPs are discussed in Section 2.1.3.

2.1.2 Clock Regions

For clocking distribution purpose, Xilinx FPGAs are partitioned into *clock regions*. Clock regions are the fundamental parts of an FPGA, which allow low clock skew (i.e., unsynchronized distribution of the clock cycles) across the device. The number of clock regions varies from device to device. However, all the considered Xilinx FPGAs are partitioned in a matrix of clock regions, which always have two columns (a clock region spans in all the cases half of the die). Moreover, for each FPGAs family, a clock region has always fix high.

Table 2.1 shows how the number and dimension of a clock region change according to a certain device. The height of a clock region goes from 16 CLB in the Virtex-4 to 50 CLB in the 7 Series and Zynq (these two families have the same FPGA architecture). The different Xilinx FPGA families are presented in Section 2.1.6. On the contrary, the number of clock regions is almost the same in newer devices. It is worth to mention that a column of a clock region is called *frame*. A frame is the smallest addressable element of the Xilinx FPGA bitstream (see Section 2.1.4).

2.1.3 Programmable Interconnection Points (PIPs)

The PIPs are programmable CMOS transistors that allow a certain signal to be routed into the device (Figure 2.3) [64]. The logic blocks inside an FPGA communicate among them utilizing a complex structure of fixed connections (see Section 6.1). It is possible to route a signal over this routing network activating PIPs.

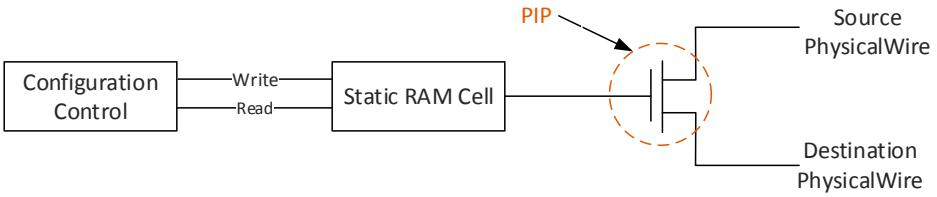


Figure 2.3: Programmable Interconnection Point (PIP) representation.

More in detail, the SRAM technology has five transistors that form the SRAM cell and a pass transistor that creates the contacts [32; 64]. Thereby, PIP transmits a signal depending on the value in its configuration memory cell. The PIP concept is extensively utilized within all the presented work, considering their utilization and testing. The number of PIPs can vary according to the device dimension and Virtex FPGAs.

2.1.4 Configuration Memory (Bitstream)

The configuration memory is the part that controls the overall configuration of an SRAM-based FPGA. The information stored in the configuration memory is called *bitstream*. The FPGA families considered in this work have a slightly different organization of their configuration bitstreams; however, the general structure is the same.

The bitstream is composed of a set of *frames*. A frame corresponds to an FPGA clock region column, and it spans vertically from the top to the bottom of the configuration memory. Moreover, a frame is the atomic part of the bitstream that can be configured. When a frame is transferred into the device, a final state machine takes care of transferring the data in the correct position of the configuration memory.

For every FPGA family, Xilinx provides guides regarding its configuration memory [123; 139; 142; 147; 151]. The documents give details and information regarding the configuration interfaces (i.e., Serial, SelectMap, Master Serial Peripheral Interface (SPI), Master Byte-wide Peripheral Interface (BPI) and Joint Test Action Group (JTAG) configuration interface), dynamic reconfiguration ports and bitstream generation.

However, Xilinx does not provide the information of how a certain bit of the configuration frame can control a specific hardware function of the FPGAs. These kinds of details are provided for older FPGAs families only [143]. Nevertheless, in the last decade, a lot of works and projects reverse-engineered the bitstream structure of Xilinx FPGAs, allowing manipulation of the bitstream for custom applications (e.g., Replica [58] and Torc [109]).

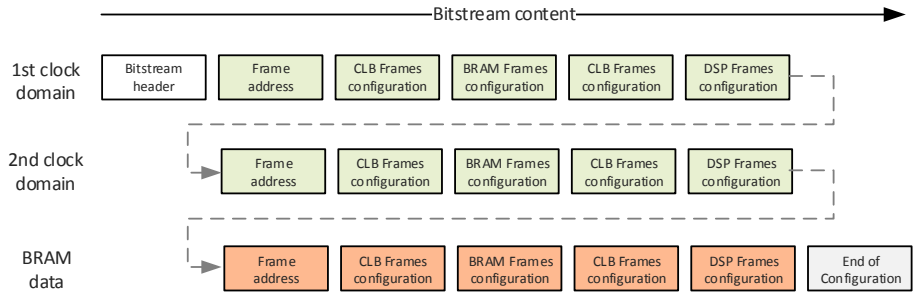


Figure 2.4: Bitstream structure.

Figure 2.4 depicts the bitstream structure. The bitstream is composed of (1) a *bitstream header*, then (2) *n Logic Frames Configuration* according to the dimension of the device, (3) *BRAMs frames* and finally (4) *the end of configuration*.

As shown in Figure 2.4, every frame starts with one *Frame Address Register (FAR)*, which allows determining the position of the frame inside the configuration memory. Even if it seems an overhead for a full bitstream, this property is remarkable when just some frames need to be configured. This is the case of a small design where unutilized frames do not need to be configured. Also, when DPR is performed, typically just some frames are reconfigured.

[125] provides all the information to create a bitstream, either full or partial. Table 2.2 in Section 2.1.6 shows the dimension of a full bitstream for each FPGA family.

The process of configuring an FPGA can be modeled in 4 sequential steps [142]:

1. *Setup*: consists in power up the device and clear the previous configuration memory.

2. *Device initialization*: initializes the state machine that loads the bitstream.

3. *Bitstream Loading*: is the process of writing all the frames in the correct position of the configuration memory. This process is controlled by the state machine.

4. *Startup*: is the phase where the loaded configuration is electronically implemented into the device. Hence, it can start to operate.

2.1.5 Routing Physical Wires

INTs are connected among them using general purpose routing connections, which are called *physical wires (PWs)*. In the following, these connections are classified, giving a better overview of the FPGA routing.

In all the Xilinx families considered in this work (Virtex-4, Virtex-5, Virtex-6, Spartan-6, 7 Series and Zynq), the PWs are divided into two main groups: *local PWs* and *long PWs*. The first type provides mostly connections with adjacent tiles. Instead, the long PWs are connections that span an extended portion of the FPGA, allowing direct connection among distant INTs.

The direction of these PWs is another aspect that differs in these two categories; the local PWs are unidirectional connection while the global PWs are bidirectional.

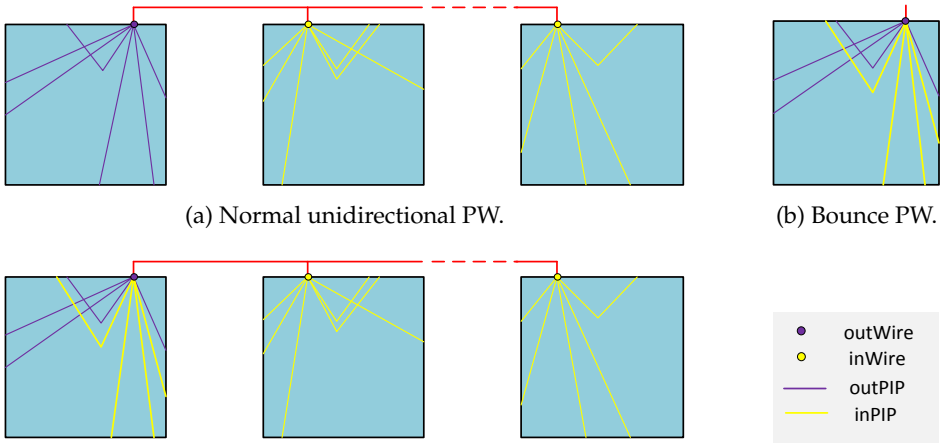
Local Physical Wires

As explained in Section 2.1.1, the connection point between a PW and an INT is called *wire*. In the specific, a wire is an *inWire* if the signal goes from the INT to the PW; in the opposite case, a wire is called *outWire*. According to this classification, it is possible to have two different PIPs: *inPIPs* and *outPIP*. In inPIPs the signal pass first through the PIP and then to the wire; in the opposite case a PIP is called outPIP.

In Figure 2.2 the outPIPs and inPIPs are depicted in purple and yellow respectively, according to the representation in FPGA Editor [128]. It is important to clarify that every PIP can be at the same time inPIP and outPIP; the classification is given with respect to the wire that is considered. Every PIP connects one inWire and one outWire, hence, considering the inWire the PIP is presented as an inPIPs; in the other case, respect to the outWire the same PIP is classified as an outPIP.

Figure 2.5 shows the different types of local PWs, which are:

- *Normal Unidirectional PW*: this type covers most of the FPGAs connections. These PWs are composed by one outWire connected to one or more inWires (Figure 2.5a). To the outWire of this PW type, only outPIPs are connected.
- *Bounce Unidirectional PW*: these PWs (see Figure 2.5c) are similar to the Normal Directional PW, except for the PIPs connected to the outWire; these are either of the outPIP or inPIP type. In this way, the wire can be used like a bounce PW (next PW type). So the signal it is not routed through the PW but the wire is just used to "bounce" the signal.
- *Bounce PW*: this PW (see Figure 2.5b) is composed by one outWire, where inPIPs and outPIP are connected to it. Therefore, these PWs can be used only like a bounce PW; passing through this wire is possible to reach another outWire of the same INT.



(a) Normal unidirectional PW.

(b) Bounce PW.

(c) Bounce unidirectional PW.

Figure 2.5: Xilinx FPGA PWs types. The light-blue boxes represent SMs, and the red lines represent the PWs.

Long Physical Wires

This PWs span a wider number of INTs. For example, in Virtex-4 long PWs span 24 INTs and in Virtex-6 they span 12 INTs. Their direction is either vertical or horizontal. In addition, they can route signal in both directions, differently from local PWs.

Often, long PWs create problems in a DPR system; the possible issues are:

- *Reliability*: in DPR systems, the FPGA is partitioned into static and reconfigurable areas. In this scenario, one of the main topic considered is the fault-tolerance of the system. Many works have investigated how a fault occurred in one area can affect another one. One of the ways to propagate errors is an unexpected configuration of a PIP. Therefore, in this case, if a PIP related to a long PW is activated, the error can be propagated to more reconfigurable areas, due to the intrinsic length property of long PWs.

- *Inhomogeneity of DPR systems*: as Section 3.2.1 presents , when a module is placed and routed (P&R) within a certain reconfigurable area, the tools verify that the resources are not utilized by other circuits. In most of the cases, the static design is configured, then reconfigurable modules are considered. In this kind of scenario, a PAR tool just needs to verify that the FPGA resources are not occupied by other circuits. A possible problem can occur in the case of module relocation. The bidirectional property of the long PWs gives to

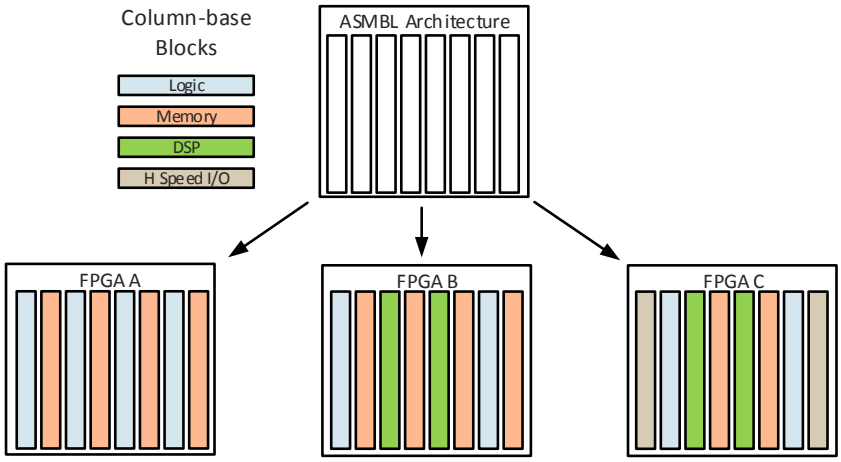


Figure 2.6: Application Specific Modular Block (ASMBL) architecture.

the P&R other condition to consider; whenever a PR Module needs to be synthesized in an area of a particular type, for example *type A*, the PAR needs to verify in all the area of *type A* if no PWs are in conflict with the design. Therefore, a possible solution to avoid issues is to exclude the use of long PWs in the routing of a PR Module.

2.1.6 Xilinx FPGA families

In this work, the Xilinx FPGAs are the target devices. These SRAM-based FPGAs offer advanced functionalities and allow the implementation of a single application on a single chip. Furthermore, these devices are partial homogeneous across different families and devices. In this thesis, all devices starting from the Virtex-4 family [144] are considered and studied. These devices, even if they are more than a decade old, are still used in many kinds of applications.

For example, the new Solar Orbiter satellite (presented in Section 3.3.2), has on board a radiation tolerant Virtex-4 device; it will be launched in 2018, and it will have a mission duration of 10 years. This gives the idea of how supported and valuable are these devices, even after more than a decade since their introduction in the market.

Xilinx started to utilize a new production method for their devices from the Virtex-4 family: the Application Specific Modular Block (ASMBL^{TM}) [84]. This kind of modular approach allowed Xilinx to produce multiple kinds of devices, which embed in some cases different dedicated hardware (e.g., PowerPC®, DSP).

ASMBL^{TM} supports the concept of multiple domain-specific platforms through the use of a column-based architecture approach (presented in Figure 2.6). Each column represents a silicon sub-system with a specific capability (e.g., CLBs, BRAMs, IOBs, DSPs).

This has allowed having different kinds of devices in term of space and functionalities, enabling deployment of multiple domain-specific FPGAs to target different customers and different applications [84] (i.e., the customer can select a specific device with specific functionalities, avoiding buying an expensive device). The ASMBL^{TM} production method is still used for the latest Xilinx FPGAs produced.

Table 2.2 summarizes the main features and difference among the Xilinx FPGAs considered in this work: *Virtex-4, Virtex-5, Spartan-6, Virtex-6, and 7 Series*. In total, the devices considered are 207.

Within seven years, the maximum number of logic cells available in the device is ten times bigger, going from 200 k in Virtex-4 to 1,954 k in Virtex-7. Moreover, in newer families, Xilinx introduced fast I/O connections, i.e., high-speed transceiver and PCIe interfaces. This motivates the growing interest and used applications of these kinds of devices. In the following, properties of each FPGA family are presented.

Virtex-4

Virtex-4 devices [144], introduced in 2004, are produced on a 90 nm copper process. This family provides three different platform subfamilies of devices: *LX*, *SX*, and *FX*. The LX sub-family is oriented to applications that require high logic functionality; SX devices are oriented to high-performance solutions for DSP applications; FX FPGAs provide high-performance, full-featured for embedded platform applications. Virtex-4 FX FPGAs have on board 1 or 2 IBM PowerPC processors, RocketIO transceiver blocks.

As mention above, these devices have been the first produced with the ASMBL^{TM} method. This has allowed providing 29 different FPGAs of the Virtex-4 family Table 2.2.

Virtex-5

In 2006, the Virtex-5 family was introduced [146]. These devices are built with a 65 nm CMOS technology using the second generation of ASMBL^{TM}. Like the Virtex-4 family, these devices are divided into subfamilies, according to their performance and dedicated embedded features.

Five subfamilies are available: *LX, LXT, SXT, TXT*, and *FXT*. LX subfamily is oriented to high-performance general logic applications. The letter *T* within the family name indicates the presence of Integrated Endpoint blocks for PCIe (x1, x4 or x8 supported) and Tri-mode 10/100/1000 Mb/s Ethernet MACs. LXT and

Table 2.2: Comparison of Xilinx FPGAs.

	Virtex-4	Virtex-5	Spartan-6	Virtex-6	7 Series			
					Artix-7	Kintex-7	Virtex-7	Zynq-7000
Date	2004	2006	2009	2009	2011	2011	2011	2011
Number of devices	29	41	45	28	27	18	20	17
CMOS technology	90 nm	65 nm	45 nm	40 nm	28 nm	28 nm	28 nm	28 nm
Logic-Cells	12 - 200 k	Max 330 k	3 - 147 k	74 - 758 k	16 - 215 k	65 - 477 k	326 - 1954 k	28 - 444 k
Max Block RAM	1 Mb	18 Mb	4.71 Mb	37.4 Mb	13 Mb	34 Mb	68 Mb	26.5 Mb
Max DSP blocks	512	1,056	180	2,016	740	1,920	3,600	2,020
Transceiver Blocks	0 - 24	0 - 24	0 - 8	0 - 72	4 - 16	8 - 32	28 - 72	0 - 4
Max transc. Speed	6.5 Gb/s	6.5 Gb/s	3.2 Gb/s	11 Gb/s	6.6 Gb/s	12.5 Gb/s	28 Gb/s	12.5 Gb/s
PCIe Interface	x1 Gen1	x8 Gen1	x1 Gen1	x8 Gen2	x4 Gen2	x8 Gen2	x8 Gen3	x8 Gen2
Bitstream size (MB)	0.6 - 6.1 MB	0.8 - 9.9 MB	0.3 - 4.0 MB	3.1 - 18.7 MB	2.1 - 9.3 MB	2.9 - 17.9 MB	13.3 - 45.9 MB	2.0 - 16.6 MB

SXT devices have RocketIO GTP transceivers (design to run from 100 Mb/s to 3.75 Gb/s). The subfamilies TXT and FXT have on board from 8 to 48 RocketIO GTX transceivers, which run from 150 Mb/s to 6.6 Gb/s. Finally, the FXT sub-family is the only one that mounts 1 or 2 PowerPC 440.

With respect to the previous family, Virtex-5 introduced a new general purpose routing structure (presented in Section 6.1). The modification consists of the introduction of a more dense routing structure, which allows having a diagonal connection to the INTs as well as horizontal and vertical. This permits the reduction of the Nets' latency, improving the overall performance of the system.

Spartan-6

Introduced in 2009, the Spartan-6 [137] targets applications with a low-power footprint, low-cost and high-volume availability. This family is built on a 45 nm process technology. This family provides a small form-factor packaging and a different number of supported I/O protocols. The logic cell density ranges from 3,840 to 147,443.

Two sub-families are available: *LX* and *LXT*. LX is oriented to applications that require mostly logic components; LXT version provides integrated Endpoint block for PCIe design and high-speed GTP transceivers (bandwidth up to 3.2 Gb/s).

Virtex-6

Introduced in 2009 (in parallel with the Spartan-6 family), Virtex-6 [149] is built with the third-generation of ASMBL^{TM} column-based architecture. It is based on a 40 nm copper CMOS process. Starting from Virtex-6, no device is provided with embedded processors anymore; Soft IP-CORE microprocessors are available and can be programmed on the device.

28 different devices are produced, where logic cell capability varies from 74,496 to 758,784; the BRAM capability varies from 5.621 kb to 38.304 kb. All the devices provide up to 4 PCIe blocks, up to 4 Ethernet MACs connections and up to 48 GTX transceivers.

Virtex-6 is divided into three sub-families: *LXT*, *SXT*, and *HXT*. LXT provides high-performance logic with advanced serial connectivity; SXT targets signal processing capability with advanced serial connectivity; HXT mounts up to 25 GTH transceivers with a bandwidth up to (6.6 GB/s).

7 Series

In 2011, 7 Series FPGAs [124] was introduced, which is built on a 28 nm high-k metal gate (HKMG) process technology. This family provides SelectIO technology with support for DDR3 interfaces up to 1,866 Mb/s. 7 Series unifies all the different

subfamilies of devices, under a main series (differently from the Spartan-6 and Virtex-6 family).

This has allowed having:

- Xilinx IP cores reused across all the 7 Series devices.
- Unified tools optimized for this kind of family; the new CAD flow Vivado [153] is introduced (provided in parallel to the ISE design flow, Section 2.3.1).

The series is divided into three families: *Artix-7*, *Kintex-7*, and *Virtex-7*. Artix-7 family can be seen as the updated version of the Spartan-6. Therefore, it targets the lowest cost and power consumption with small form-factor packaging. Kintex-7 is the mid-class family providing transceivers with a speed up to 12.5 Gb/s and logic cells up to 478 k. The Virtex-7 family provides the highest system performances (as the previous Virtex devices) and can have up to 1,955 k cells, up to 68 Mb of Block RAM, and transceivers with a speed up to 28.05 Gb/s.

Zynq-7000

Introduced in 2011, the Zynq-7000 family [163; 164] is based on the Xilinx All Programmable SoC architecture. These products integrate a Dual-Core ARM Cortex-A9 based processing system and a 28 nm Xilinx programmable FPGA in a single device (corresponding to the 7 Series architecture). The ARM Cortex-A9 CPUs include on-chip memory, external memory interfaces, and peripheral connectivity interfaces.

The Zynq-7000 family offers flexibility and scalability of an FPGA while providing performance, power, and ease of use device. These devices allow designers to target cost-sensitive as well as high-performance applications.

UltraScale

Introduced in 2013, the UltraScale family is the successor of the 7 Series family [140; 157; 158]. It provides up to 5.5 Logic Cells at 20nm. Similarly to the 7 Series family, they are divided into two sub-families:

- Kintex UltraScale: these devices focus on price/performance. They embed a high number of DSPs, block RAM, and transceivers. They provide an optimal tradeoff between capability and cost.
- Virtex UltraScale: these devices target the industry. The highest system capacity, bandwidth, and performance. Compared to the Kintex UltraScale, these devices provide higher system logic blocks, up to 5,541 (see Table 2.3), and up to 30.5 Gb/s transceivers.

Table 2.3: Comparison of UltraScale and UltraScale+ families FPGAs [162, pp. 1,26][141, pp. 18,19][141, pp. 878,879].

	UltraScale		UltraScale+		
	Kintex UltraScale	Virtex UltraScale	Kintex UltraScale+	Virtex UltraScale+	Zynq UltraScale+ MPSoC
Date	2013	2013	2015	2015	2015
Number of devices	25	24	17	22	55
CMOS technology	20	20	16	16	16
Logic-Cells	318 - 1451 k	783 - 5541 k	356 - 1,143 k	862 - 3780 k	103 - 1143 k
Max Block RAM	75.9 Mb	132.9 Mb	34.6 Mb	94.5 Mb	34.6 Mb
Max DSP blocks	768 - 5520	600 - 2880	1368 - 3528	2280 - 12288	240 - 3528
Transceiver Blocks	12 - 64	36 - 120	16 - 76	40 - 128	0 - 72
Max transc. Speed	16.3 Gb/s	30.5 Gb/s	32.75 Gb/s	32.75 Gb/s	32.75 Gb/s
PCIe Interface	x8 Gen3	x8 Gen3	x8 Gen4	x8 Gen4	x8 Gen4
Bitstream size	15,3 - 46,0	23,9 - 123,0	14,7 - 34,7	25,5 - 76,4	5,3 - 34,7

UltraScale+

Introduced in 2015, the UltraScale+ FPGAs based on 16Fin FET+ technologies [159]. Moreover, these devices provide UltraRAM blocks and PCie x8 Gen4 [140]. This family is divided into two FPGAs sub-families:

- Kintex UltraScale+: these devices have increased performance memory, providing the ideal mix of high-performance peripherals and cost-effective system implementation.
- Virtex UltraScale+: these devices have the highest transceiver bandwidth, highest DSP count, and highest on-chip memory available in the industry for the ultimate in system performance [140].

Another subfamily of the UltraScale+ family is the Zynq UltraScale+ MPSoC, which is explained in the following.

Zynq UltraScale+ MPSoC

Introduced in 2015, the Zynq UltraScale+ MPSoC is an all programmable SoC, successor of the Zynq-7000 [140; 159; 162]. This family integrates a 64-bit Qual-Core or Dual-Core ARM Cortex-A53 and Dual-Core ARM Cortex-R5 based processing system and Xilinx programmable logic UltraScale architecture in a single device.

They also include an on-chip memory, multiport external memory interfaces, and a rich set of peripheral connectivity interfaces.

This family is divided into three subfamilies [161]:

- CG devices: they are featured with a Dual-Core Cortex-A53 (Application Processor) and a Dual-Core Cortex-R5 real-time processing unit. These devices target industrial sensor fusion, motor control, and Internet of things (IoT) applications.
- EG devices: they feature a Qual-Core ARM Cortex-A53 platform running up to 1.5GHz (Application Processor) and with Dual-Core Cortex-R5 real-time processors, a Mali-400 MP2 graphics processing unit. These devices have specialized processing elements that can target 5G wireless infrastructure, cloud computing, and aerospace applications.
- EV devices: they are based on the EG devices presented above; in addition, they integrate H.264 / H.265 video codec capability that allows simultaneous encode and decode up to 4Kx2K (60fps). These devices are ideal for automotive ADAS, multimedia, and embedded vision applications.

2.1.7 Space-Grade devices

Space is one of the target applications of FPGAs. Nowadays, the antifuse FPGAs are strongly utilized in space missions. Nevertheless, SRAM-based FPGAs are getting more and more utilized, as well as COTS FPGAs, in particular for missions with shorter lifetime and less critical constraints.

Reconfigurable space-grade products can be divided into two categories:

- *Radiation-Tolerant*: these SRAM-based FPGAs provide immunity to certain radiation effect and specific mitigation for others.
- *Radiation-Hard*: these SRAM-based FPGAs thanks to radiation-hardened process and radiation-hardened design, they are immune to the effects of radiations.

Xilinx provides qualified devices for space: the Virtex-4QV (Radiation-Tolerant) and the Virtex-5QV (Radiation-Hard). The main difference from the COTS devices is the production and qualification phase. They are fabricated on a thin epitaxial wafer and high-reliability ceramic flip-chip packaging technology. The resistance to radiations is validated using in-beam testing (equivalent of millions of device years in space radiation environment).

Nevertheless, it is important to mention that in space-grade devices, configuration memory and the high-level architecture is the same of COTS devices; this

allows a direct porting from commercial to space-grade devices. Moreover, the Xilinx design flows (e.g., ISE, EDK) are compatible with this space-grade devices.

These FPGAs offer the latest solution for addressing the needs of critical space missions where design changes can be accommodated late in the program or through reprogrammability, even after launch [135]. In the following, a description of the two Xilinx space-grade devices is given: Virtex-4QV and Virtex-5QV.

Virtex-4QV

Virtex-4QV family [136] was introduced in 2007. These devices provide immunity to Single-Event Latch-up (SEL) and high tolerance against SEU and TID. They fall in the category of space-grade radiation-tolerant devices. Xilinx provides four different FPGAs: the *XQR4VSX55*, *XQR4VFX60*, *XQR4VFX140* and *XQR4VLX200* [136]. As mentioned above, these devices correspond to a specific commercial device, sharing the same modular architecture and pin packaging (i.e., XC4VSX55, XC4VFX60, XC4VFX140 and XC4VLX200 [144]).

It is worth to mention that the configuration memory and configuration controller of FPGAs do not have specific mitigation techniques in the fabric phase. Hence, SEUs are mitigated thanks to specific mitigation techniques at application-level (e.g., Triple Modular Redundancy (TMR)).

Virtex-5QV

Virtex-5QV [135] is a rad-hard by design (RHBD) device. It is total immune to SEL and provides 1 Mrad(Si) [26, p. 227] TID performance. Introduced in 2010, this family consists of a unique device, the *XQR5VFX130*. This device provides all the new features introduced with the Virtex-5 family combined with a rad-had design technology. This device embeds Error Detection and Correction (EDAC) and autonomous write-back for high-performance block memory SEU mitigation.

The Virtex-5QV design utilizes dual-nodes latches that control write operations to memory cells. Writes occur only when both latches are enabled synchronously. This implementation offers 1,000 times the hardness to SEUs compared to the commercial FPGA version.

According to [156], SEU immunity in the configuration memory and control logic is defined regarding deployment in a GEO environment about a space platform that travels 36,00 km/day. Based on 35 Mbits of configuration memory that could be subject to SEUs, the FPGA suffers $3.8\text{x}10^{-10}$ error per bit per day.

RT Zynq UltraScale+ MPSoC

In 2016, Xilinx announced a family of radiation-tolerant devices, called RT Zynq UltraScale+ MPSoC [50]. Currently, a unique device is expected to be avail-

able in 2018: the RT ZU19EG [50]. This device corresponds to the commercial ZU19EG [140].

Differently from the previous Xilinx space-grade devices, this device is an all programmable SoC. The RT ZU19EG integrates a programmable logic part (PL) with 1143 k Logic cells (10 times more than the Virtex-5QV device) and a processing system (PS). The PS consists of an application processing unit based on a Qual-Core ARM Cortex-A53, a Real-Time Processor Unit based on a Dual-Core ARM Cortex-5, and a Mali-400 MP2 graphics Processing Unit.

2.2 Dynamic Partial Reconfiguration

As mentioned in Section 2.1, in the last two decades, the utilization of SRAM-based FPGAs increased in a wide range of applications, providing the possibility to reconfigure hardware circuits multiple times. Later, the requirements of the market increased, therefore, developers and researchers started to investigate the possibility to reconfigure just a portion of an FPGAs (e.g., change just a specific functionality of a circuit), without the need to reconfigure the rest of the system.

Then, dynamic partial reconfiguration has been introduced, which has allowed reconfiguring just a portion of the FPGAs having the rest of the device still operating; this property of FPGA is called *Dynamic Partial Reconfiguration* (also known as *Run-Time Reconfiguration*). It was introduced in the Xilinx devices in the late 90s, on the Xilinx XC6200 series [21].

DPR can be performed in two different approaches [63]:

- *Module-based*: it is performed partitioning the FPGAs in a certain number of portions (*tiles* or *slots*). In this way, a certain hardware component (called *module*), can be placed in one slots/tiles dynamically.

- *Difference-based*: this approach does not require any partitioning of the system; as the name indicates, it compares two different configurations, a *based-* and a *target-bitstream*. As results, just a bitstream that contains the differences from the base-bitstream to the target-bitstream is generated. This method is suitable when the differences between two configurations are small, allowing a reconfiguration just in specific parts of the FPGAs.

The first approach executes a reconfiguration in a certain portion of the FPGA area, without considering the previous configuration. On the contrary, when a DPR is provided with the difference-based method, the new partial bitstream depends on the previous configuration of the device.

After that researchers have started to evaluate and investigate this property [17; 27], Xilinx provided the first support for DPR in 2006; the Early Access Partial

Reconfiguration (EAPR) plug-in [127], based on the ISE design flow. This module-based approach allowed running DPR on the Virtex-2 and Virtex-4 devices.

With the wide integration and utilization of the EAPR flow, Xilinx provided from the version ISE 12.1 (2010) [34; 131] a direct integration of DPR in ISE/PlanAhead, presenting new methodologies and support for a wider range of FPGAs (Virtex-4, Virtex-5, Virtex-6, Artix-7, Kintex-7, and Virtex-7). This new approach is called *Xilinx PR* (see Section 3.2.1).

2.2.1 Benefits

Dynamic Partial Reconfiguration (DPR) is a remarkable feature of FPGAs [34]; it gives the designer the ability to reconfigure a certain part of the FPGA at run-time without influence the other ones.

Utilizing DPR, a system can have the following benefits:

- **Reduce cost**: the device can be time-scheduled, configuring a specific functionality only when is required. This allows the adaptation of a smaller device, rather than have a bigger one with all the functionalities implemented.
- **Change a design on the field**: DPR increases the flexibility of the device, allowing updates to a certain functionality without changing the overall system.
- **Reduce power consumption**: power is always an important aspect to consider in electronic devices. The power consumption of an FPGA is usually divided in *I/O power*, *dynamic power* and *static power* [160]. On the one hand, dynamic power can be reduced, since that, just a needed application can be configured when is needed; this avoids having unused circuits powered-on on the device. On the other hand, static power can be reduced as well, utilizing a smaller device. Moreover, considering the overall power consumed by FPGA, the impact of static power consumption is increasing, as CMOS technology shrinks [**Kuon**; 132].
- **Increase reliability (fault-tolerance)**: SRAM-based FPGAs can be affected by radiations, which can change the configuration of a circuit (radiation effect on FPGA is discussed in Section 2.4). Thanks to the DPR, when a soft error occurs, it can be easily corrected overwriting it (i.e., blind-scrubbing [18]).
- **Reduce memory requirement**: if the FPGAs needs to be multiple time reconfigured, one of the problems can be the space required for the bitstream; as discussed in Section 2.1.4, a bitstream can occupy up to 46 MB. DPR allows having just a set of partial bitstreams, which can be reconfigured according to a specific use scenario.

Moreover, a DPR scenario can bring benefits in the final design complexity. Even if the development of a DPR communication infrastructure needs a higher cost for the design implementation, on the contrary, the final design can be less complex than a static one. More details about the comparison between static and DPR designs are provided in [98].

2.2.2 FPGA partitioning

In the following, it is described how a DPR system can be partitioned. This section summarizes the work presented in [61]. As presented at the beginning of this section, DPR can be applied in two different approaches: *module-based* and *difference-based*.

Difference-based is utilized for small changes on a design. However, the module-base approach offers more features and capabilities. A system that adopts DPR is called Partial Reconfigurable System (PR System). Xilinx provides an extensive guide, which explains how module-based DPR can be created and utilized [131].

Figure 2.7 represents a heterogeneous FPGAs, which contains different logic blocks; in addition, it shows how a PR System can be created. A PR System design requires a partitioning of the FPGA to reconfigure only specific areas. In particular, two different regions are created: *base region* and *dynamic region* (also called Partial Reconfigurable Region (PR Region)). In Figure 2.7 they are indicated in dark-gray and light-gray respectively.

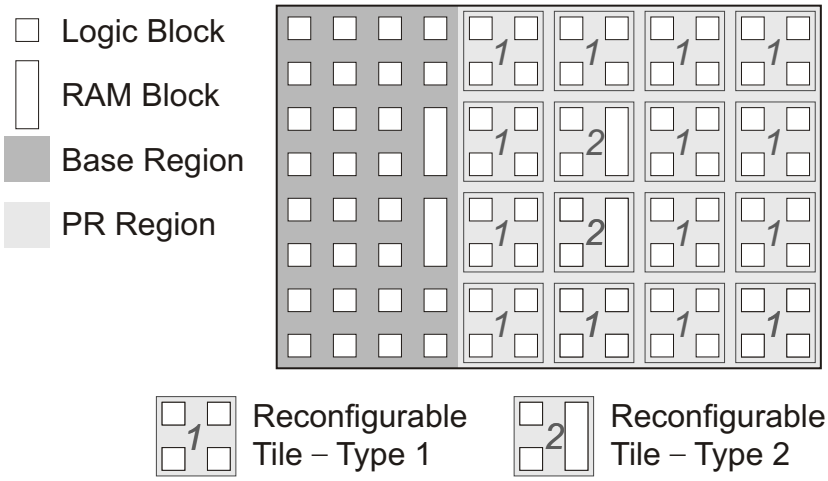


Figure 2.7: FPGA Partitioning using Partial Reconfigurable Regions (PR Regions) with Reconfigurable Tile [61].

The base region contains components that are not reconfigured. Therefore, the configuration of this region is made once in the initialization of the system, and it is not changed at run-time. On the contrary, the reconfigurable region is used for run-time reconfiguration; all the dynamic system components are located in this region.

To fully exploit the DPR capabilities, another partitioning of PR Region is required. A PR Region is divided in *Reconfigurable Tiles (PR Tiles)*, which are the smallest partial reconfigurable units in the system. According to the heterogeneity of the FPGAs, a PR Region may contain different types of PR Tiles, which contain different resources (e.g., CLBs, Block RAMs and DSPs). For example, in Figure 2.7, the PR Region is divided in PR Tiles of two different types.

The dynamic logic components are represented by the PR Modules, which can be placed and removed at run-time. The placement is done by reconfiguring suitable contiguous PR Tile. According to the PR Tile types, the PR Module can be placed in different position, using an equivalent configuration; this mechanism is called *bitstream*.

In this kind of scenario, the communication infrastructure is one of the key components, which allow communication among static regions and all the PR Tiles. Section 2.2.3 discusses in detail how a communication infrastructure, which allows relocation of PR Modules, can be created.

2.2.3 Communication Infrastructure in a PR system

Communication infrastructure is a key part of a DPR design; it allows the interconnection of all the different areas of the system. Different communication infrastructures can affect the reconfigurable property of the system (e.g., homogeneity, interrupt free reconfiguration, dedicated signal). This section summarizes the communication infrastructure presented in [61].

In 2006, with the EAPR flow [127], Xilinx provided the first approach to establishing communication between different areas (static or dynamic): the bus-macro [63]. Bus-macros are instances of the FPGA logic and routing resources. They are intended to lock the routing between different regions, making possible to connect the pins of either the static part or a PR Module. In this way, whenever a PAR is executed, the resources occupied by a bus-macro are reserved. Hence, in the EAPR plug-in, the bus-macros were the only communication channels crossing the reconfigurable regions.

Although the term bus-macro seems to indicate a macro for implementing bus structures, its use does not go in this direction. On the contrary, this connection is commonly used in single-module PR Regions for the communication link between a PR Module and the base region; this type of connections are referred as *link macros*.

In addition to bus-macro, Xilinx provides the possibility to create any functionality that can lock logic and routing resources: *hard macros*. They are pre-P&R design blocks, which can be created once for a certain family of FPGAs, without be related to a certain location of a device. Therefore, all logic and routing resources of a hard macro can be moved together to maintain the same "shape".

Hard macros permitted investigating different communications infrastructures, overcoming the limitation of the Xilinx bus-macros. In the following, different communication approaches are presented. In particular, it is highlighted the capability of a communication infrastructure to keep the PR System homogeneous. The term "homogeneous" indicates that the communication infrastructure utilizes the same logic and routing resources in the same relative position; this allows to a PR Module to be relocated to a different location.

Link Macros Between Tiles (LMBT)

As presented in [61], in this communication infrastructure, link macros are used to interconnect neighboring tiles. Moreover, this communication infrastructure can realize 1-D as well as for 2-D PR Tiles partitioning (as shown in Figure 2.8a).

The disadvantage of this approach is that link macros are only used to establish the connection from one PR Tile to another. Therefore, the bandwidth suffers from a large delay across multiple PR Tiles.

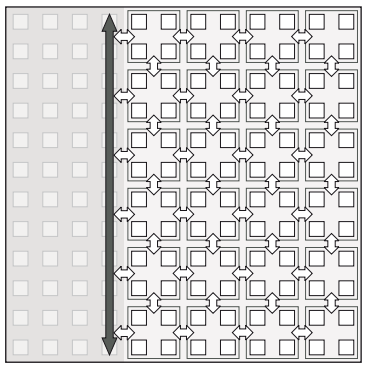
In addition, the connection of the link macros within a PR Tile. Then, the communication infrastructure is module-dependent; this means that the routing can be interrupted and can change during the reconfiguration process of a PR Tile.

Consequently, the implementation of the communication for PR Tile depends on the communication infrastructure of the surrounding PR Tiles. Each new PR Module placement requires changes to all other modules that are involved in its communication, causing additional reconfiguration overhead.

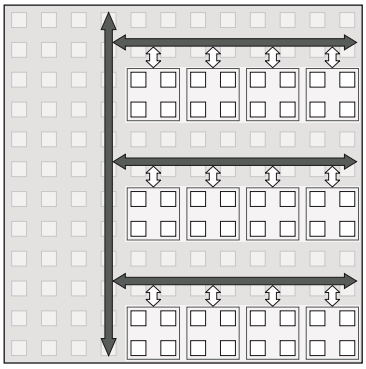
Link Macros combined with Communication Channels (LMCC)

LMCC communication infrastructure ensures that each PR Tile can be directly connected to the base region (differently from the LMBT approach); this is achieved combining link macros with communication channels [61]. An example of this communication infrastructure is illustrated in Figure 2.8b. The communication channels are part of the base region, and the link macros are placed between this region and the partially reconfigurable region.

The partitioning shows that the PR Region is split into multiple PR Regions segments since that the area of the communication channels can be only used for communication and static logic. Then, the maximum size of a PR Module is limited to the size of a segment since that the communication channel separates the PR Region segments.



(a) Link Macros Between Tiles (LMBT).



(b) Link Macro and Generic Routing.

(c) Wormhole Routing Scenario.

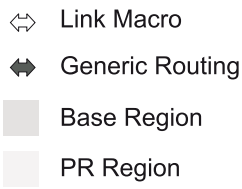


Figure 2.8: Typical Communication Infrastructures in a PR Region scenario [61].

Hence, comparing this communication infrastructure with Link Macros Between Tiles (LMBT), the number of PR Tile are less (having the same PR Region, as depicted in Figure 2.8). However, this kind of approach can handle modules relocation and DPR.

Wormhole Routing

Wormhole routing [74] is a method to realize a communication infrastructure that circumvents the limitations of LMBT and LMCC. With wormhole infrastructures, communication channels span over one or more PR Tiles; more distant is the

PR Tile from the static region, higher is the number of PR Tiles crossed by its communication channel [61].

Whenever the reconfigurable area comprises many PR Tiles, this approach consumes a considerable amount of routing resources, as can be seen in Figure 2.8c. Therefore, the routing resources that are available for PR Modules linearly decrease from right to left. Since only point-to-point connections between a PR Tile and the base region are used, every Partial Reconfiguration (PR) requires an exclusive set of communication lines.

The disadvantage of this approach is the lack of homogeneity in the system; the communication channels routing within a PR Tile differs according to the PR Tile column (as it shown in Figure 2.8c). Moreover, PR Module relocation cannot be performed.

2.2.4 Embedded Macros

Embedded macros, introduced in [61], provide a method to create a communication infrastructure that is embedded into the PR region. Therefore, part of the communication infrastructure is included in the PR Module as well.

The embedded macro is not a point-to-point connection; instead of using multiple instances of simple link macros, one monolithic macro is created, as shown in Figure 2.9. The macro connects all PR Tiles with the base region homogeneously; it combines the advantages of the three link macro variants described, without sharing their drawbacks.

The main property of this communication infrastructure are:

- *Homogeneity*: the communication infrastructure uses the same resources in the same way in the PR Tiles. Therefore, is possible to place a PR Module in difference positions. Wormhole Routing does not support this feature.

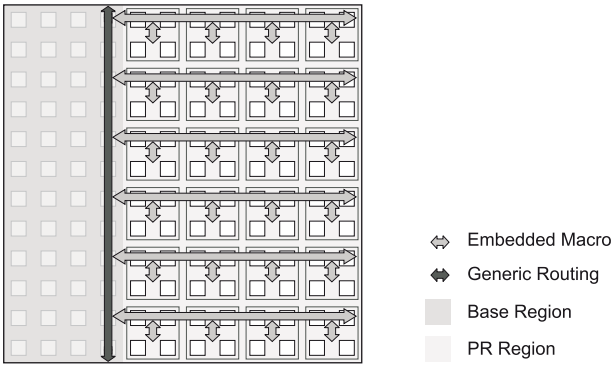


Figure 2.9: Communication Infrastructure using Embedded Macros [61].

- *Interrupt free reconfiguration*: the communication infrastructure is implemented to operate even during the reconfiguration phase. This property is not supported in LMBT; the communication channels pass through the PR Module interrupting the signal during the reconfiguration process.
- *Dedicated signal*: embedded macros utilize this kind of connection that are not presented in LMBT communication macro, which reduces communication latency.

2.3 Xilinx Design Flow

FPGAs allow the implementation of complex circuits in hardware. The circuits are translated to a bitstream and then configured on the devices. Xilinx provides tools and instruments that enable users to implement their applications, for example starting from an HDL representation. Moreover, Xilinx provides tools and languages to interact with custom solutions and implementation of the synthesis flow, allowing the possibility to researchers of enhancing their functionalities; one of this is the ISE design flow (Figure 2.10). ISE comprises the following steps: *design entry*, *design synthesis*, *design implementation* and *device programming*.

2.3.1 ISE

ISE design tool flow gives the overall context and framework for the development cycle of FPGAs. It provides all the steps to bring a certain design from the high-level representation to a configurable bitstream.

This thesis focuses on the PAR, which is part of the design implementation phase. This phase gets in input a synthesized design and then is mapped and placed within the logic blocks of the design; then, it is routed through the general purpose routing matrix. At the end of this step, the design is then P&R.

Figure 2.11 presents the ISE PAR phase and its intermediate files:

- The *Translate* process merges all input net-lists design constraints and generates a Xilinx Native Generic Database (NGD) file, which describes the logical design in the Xilinx primitives format.

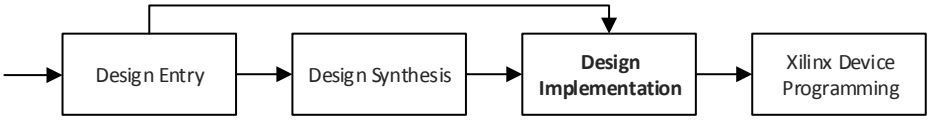


Figure 2.10: Xilinx ISE Design Flow.

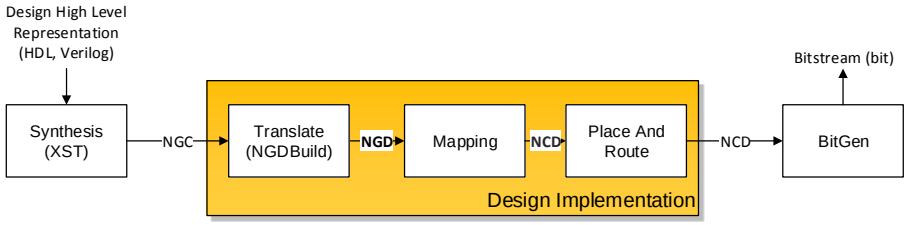


Figure 2.11: Xilinx ISE Design Implementation.

- *Mapping* process maps the logic defined by an NGD file into FPGA elements, such as CLBs and IOBs; the output is a Netlist Circuit Description (NCD) file.
- *Place and Route* process takes a mapped NCD file, places and routes the design, and it produces another NCD file.
- *Bitgen* process produces a bitstream for Xilinx device configuration (a *.bit file); at this point, the target FPGA can be configured.

In all these steps, the P&R design is represented in a Xilinx property language (NCD), which is unusable by external tools. Nevertheless, Xilinx has instructed an intermediate language: the Xilinx Desing Language (XDL). XDL describes a design using a human-readable syntax. Moreover, Xilinx has provided a dedicated tool, FPGA Editor, which allows the user to operate custom modification on an implemented design (e.g., modify the placement of one component or the routing of one net).

ISE, and their sub-programs (i.e., FPGA Editor and XDL), supports all the FPGAs presented in Section 2.1.6: Virtex-4, Virtex-5, Virtex-6, Spartan-6, 7 Series and Zynq-7000. Then, Xilinx decided to utilize a new program starting from the 7 Series FPGAs: *Vivado*.

This new tool replaced the ISE design flow starting from the UltraScale series (in the 7 Series, both flows are kept). Then, the tools of the presented thesis need to be adapted to the new Vivado flows to be compatible with the latest Xilinx FPGAs.

However, this thesis targets the space harsh-environment, which consists in a longer life-time utilization of the considered devices, and therefore, to their design tools. One example is the reconfigurable Virtex-4QV mounted on the Solar Orbiter (see Section 3.3.2); this satellite will be launched in 2018 and its planned operation time is ten years.

2.3.2 FPGA Editor

FPGA Editor [128] is a graphical editing tool for physical designs implemented in Xilinx FPGAs; it is a subpart of the ISE design flow. The FPGA Editor requires either an NCD or an NMC file. These files contain the logic of a design mapped to components (such as CLBs and IOBs).

FPGA Editor allows the user to perform different modifications on a implemented design, either if the design is just mapped, placed or routed. Moreover, it is able to apply just one of the mentioned operations for a single or small set of components. Some possible operations are:

- *Place and route components* (before running the PAR default tool) and *Finalizing placement and routing* (if the routing program does not completely route your design). In this way, the user can partially control the PAR of a design. If a manual placement can be performed easily in FPGA Editor, on the contrary, this is not the case of manual routing; a manual route requires extensive knowledge about the FPGA routing structure. Moreover, to route a certain net, all the resources need to be selected in right order, starting from the outpin and going to the inpin [129].
- *Add probes to your design* to examine the signal states of the targeted device [129]. Probes are used to route the value of internal nets to an IOB for analysis during the debugging phase.
- *Integrated Timing Analyzer* to cross-probe a design [129].

Thanks to these advanced features, FPGA Editor is extensively used in the DHHarMa tool of INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0) and in the OLT(RE)^2 flow.

FPGA-Edline

One utility of FPGA Editor is *FPGA-Edline*, which is a command-line style version of FPGA Editor. The main advantage of FPGA-Edline is that commands can be executed directed in FPGA Editor.

So, despite the XDL language, when a certain script uses FPGA-Edline commands, the design does not need to be converted in XDL and then back to NCD. As presented in Section 2.3.3, the conversion in XDL has some drawbacks, and in some cases, it cannot be performed for a full design.

FPGA-Edline is utilized in the PSRerouter of INDRA 2.0 flow to extract FPGA routing information (not included in XDL) and to reroute single nets of complex designs. Furthermore, it is utilized in OLT(RE)^2 to highlight routing resources of an FPGAs, giving graphical information regarding their testability. [128] gives more details about the FPGA-Edline commands.

2.3.3 XDL tool

Xilinx provides the users of ISE with two powerful tools to describe an FPGA: The first tool, FPGA Editor, is presented in Section 2.3.2. The second one is a command line tool, named *XDL*; it provides the mechanism for gaining external access to design data. XDL generates a human-readable file (in XDL format) to describe either an FPGA architecture or a design textually.

Xilinx documents neither the XDL language nor the XDL tool. A detailed explanation of the syntax and semantic of an XDL can be found in [8; 41; 44; 184].

The tool *XDL* offers three different running modes:

1. *ncd2xdl*: conversion of an NCD to an XDL-file.
2. *xdl2ncd*: conversion of an XDL to an NCD-file.
3. *report*: description of an FPGA architecture.

The first two modes give the designer the opportunity to textually modify a build design file (NCD) or a hard macro file (NMC). Differently, the report running modes is oriented to provide information of the FPGA architecture; it generates an XDL-report file (the so-called XDLRC format). The XDL-report contains information about the whole FPGA in an ASCII-formatted text file.

Since its introduction, the intermediate language XDL has been used by many researchers to deploy new functionalities on FPGAs. The XDL language is used as an intermediate language in the RecoBus-Builder [60], GoAhead [9], Rapid-Smith [66] and Torc [109] (see Section 3.1) as well as in DHHarMa and OLT(RE)^2 tools, which are presented in this thesis. More detailed about XDL are presented in [25].

Macro Hardware XDL

In the following, the syntax of an XDL design is presented. Figure 2.12 shows part of an XDL file, generated from an NCD file.

As presented in [184], the XDL can be divided into three parts:

- *Properties and Ports*: this contains the main properties of the design, e.g., name of the design and target FPGA. Moreover, it contains a list of all the ports of the design.
- *Instances*: it lists all used blocks (e.g., slices, IOBs, Block RAMs) of the design. These blocks are called *primitive instances (inst)*. The example in Figure 2.12 shows one primitive instance, which represents one slice of a Virtex-6 CLB (one CLB hosts two slices in Virtex-6 FPGAs). One slice consists of four *parts*, where each consists of two LUTs, two registers, and a carry chain.

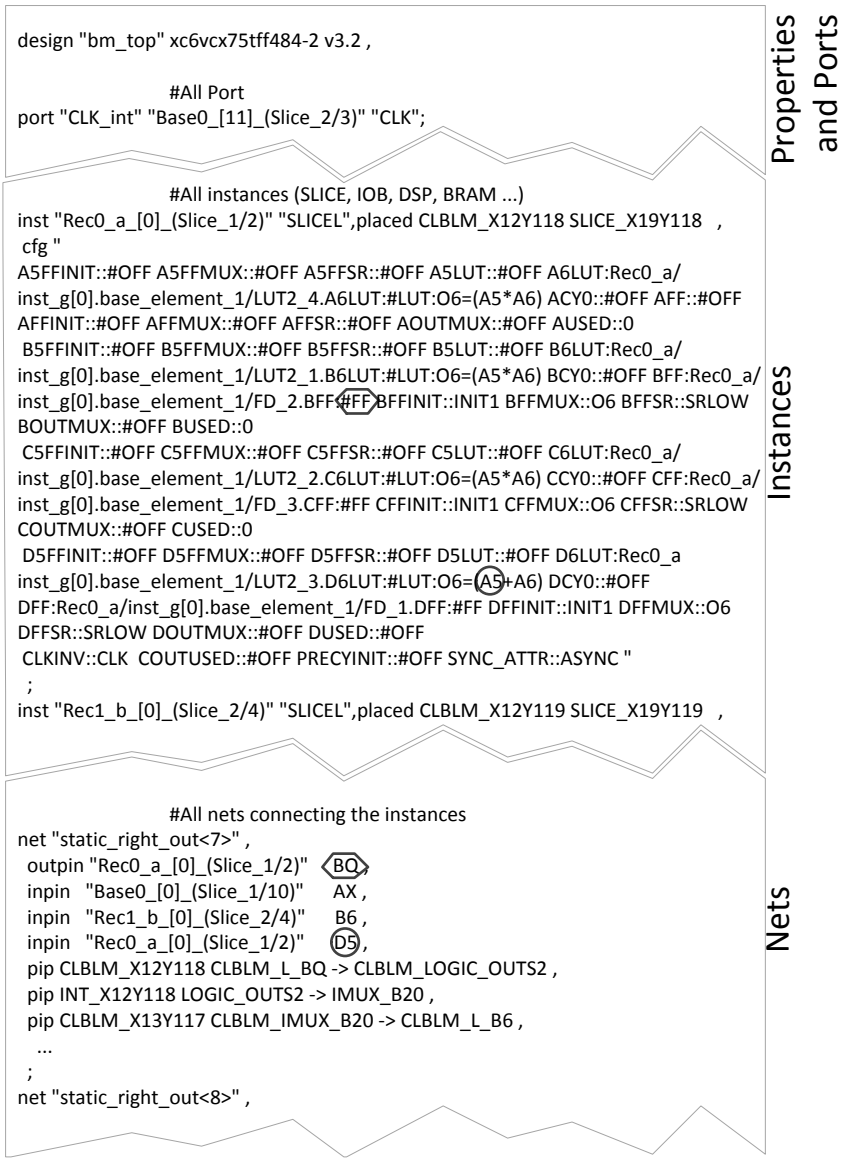


Figure 2.12: An example of an XDL-file of a design for a Xilinx Virtex-6 FPGA [184].

- *Nets*: it contains the information about the instances' connections of the design. A net consists of a set of pins, where the first pin in the list defines the output pin (outPin) and the other define the input pins (inPins). Each pin is linked to one previously defined instance. If the net is routed, the net section contains the PIPs that are utilized.

XDL Report

The report is the only standalone Xilinx file that describes a whole Xilinx FPGA with all components and their connections. Section 3.1 presents works that have created proprietary and open-source databases, which embed this information, targeting different kinds of Xilinx FPGA families.

It is possible to create three different depths of detail for each FPGA, depending on the parameters passed to the Xilinx *XDL* tool. The installed FPGAs can be determined with another Xilinx tool, called *PARTGen*. It can be launched from a command prompt: *partgen -arch <FPGA-family>* (e.g., $partgen - archvirtex6$ lists all Virtex-6 FPGAs).

In the following the possible parameters of the Xilinx report are presented:

1. *-report*: activates the creation of a coarse overview of the FPGA.
2. *-pips* : adds wires and PIPs information.
3. *-all_conns*: adds *all* connections of wire.

For example, about the three functionalities, a full representation of the FPGAs, including *Tiles*, *Primitive Sites*, *Pinwires Wires*, *Connections*, and *PIPs*, can be created with this command:

```
xdl -report <designName> - pips -all_connects
```

2.3.4 Vivado

Xilinx decided to introduce a new design flow, starting with the 7 Series families: *Vivado* [153]. This decision has been made to meet the newer requirements and functionalities that the market needs. As mentioned in Section 2.3.1, ISE is the standard flow till the Virtex-6 family, and it provides the XDL. This intermediate language has been utilized by researchers to add new functionalities to the standard ISE flow.

The new Vivado tools do not embed the XDL tool and language anymore. In substitution, Xilinx provides to the user the FPGA architecture details and low design information through a Tcl scripting language [154]. Tcl is a standard language in the semiconductor industry for application programming interfaces.

It performs interactive queries to design tools in addition to executing automated scripts, as well as provides the ability to query questions interactively of design databases. Moreover, it provides functionalities to get information about tools, design settings and state. This tool is not considered in this thesis.

2.4 Radiation Effects

This section provides a categorization of the radiation faults and effects that may occur in the space environment, focusing on the permanent ones. Failures caused by radiations in space is one of the most challenging issues on modern complex electronics systems [97].

Radiations origin either from the sun (i.e., solar flares, coronal mass ejections, solar wind) or from outside the solar system (i.e., galactic cosmic rays) [36]. Radiations are set of particles that can interact with the electronic systems, exchanging energy; these particles can easily move in the vacuum of the space environment.

The earth atmosphere acts as a shield for these kinds of particles; the molecules that the atmosphere is composed of, reduce the energy of these particles, protecting the terrestrial surface. Therefore, electronic circuits used in terrestrial applications are consequently safer than the ones used in space.

For these reasons, space is considered a *harsh-environment* for electronics systems [36]. Many efforts have been spent in the last decades to measure, model, and mitigate radiation effects; the problem has been faced with different techniques at various abstraction levels.

From the electronic system point of view, faults induced by the radiations can be categorized into two classes: *Single Event Effects (SEEs)* and *TID*. The different kinds of errors are represented in Figure 2.13.

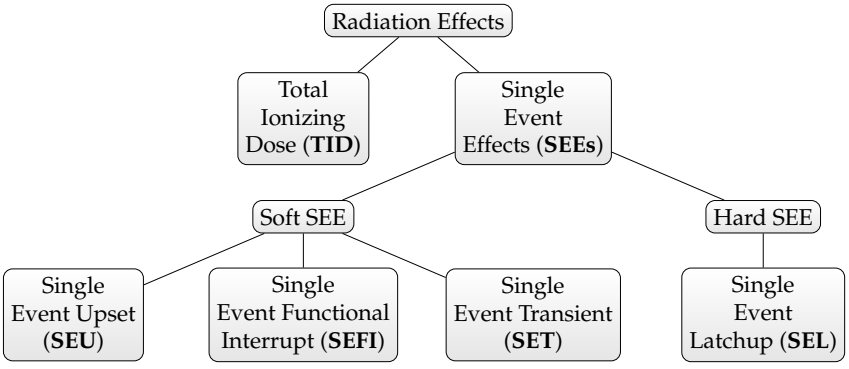


Figure 2.13: Radiation Effects classification [121].

SEEs may cause both instantaneous and long-term damages in electronic systems. Instantaneous damages are the well-studied SEUs and SETs [7]; long-term damages are usually caused by the TID, i.e., the accumulation of charge trapped in the oxide layer of transistors in CMOS circuits [110]. TID first causes degradation of the performance of the system, and ultimately it may cause failures [5].

Other effects of radiations can be displacement damages; these kinds of effect are not investigated in this work, since that are not targeting FPGAs (i.e., displacement damage effect concern electro-optic, sensors, diodes, opt-couplers, solar cells, wide-base bipolar transistors) [36].

2.4.1 Single Event Effects

SEEs are caused by the passage of a single high energy proton or heavy ion through a device or a certain sensitive region of a microcircuit. Depending on the strike location, the electric fields and the energy of the incident particle, the passage can produce different functional behavior [121].

SEEs can be temporary faults, *Soft SEEs*, if the error induced is reversible. On the contrary, if the fault is permanent (damaging the device), it is called *Hard SEEs*. In the following, the different SEEs are presented: SET, SEU, Single Event Functional Interrupt (SEFI), SEL [5; 36].

SET

These faults occur when high-energy particles impact a combinatorial path of a device and induce a voltage or a current spike. If the pulse-width of the spike is enough, it can propagate a fault through the circuit. This kind of fault affects the device for a certain period (usually until a power cycle is performed). In some cases, a SET can result in an SEU.

SEU

It is a soft error that causes the state change of a bistable element. This effect occurs because of the change deposit by ions and protons. SEU is the most common effect on SRAM-based FPGAs.

SEUs are not usually permanent faults because the correct value can be restored overwriting the wrong value on the affected memory element. However, there are some cases where the memory element could not be written again; this changes the SEU effect into a permanent one until a reset or power cycle is performed.

One typical example of SEU is a bit-flip, which can occur in the configuration memory of the device; assuming that a bit-flip occurs in a memory location that controls a switch matrix, this can lead to a faulty-route of the device. It is possible to recover from this kind of error overwriting the affected memory location.

SEFI

This kind of effect can appear in complex microcircuits. It is similar to a memory SEU, however, an SEFI leads to a temporary non-functionality of the affected device; SEFI may be not recoverable unless a global reset is performed.

For example, an SEFI can be a fault in the program counter or in the status register of a processor that brings the processor in a faulty state; the correct state can be restored just with a global reset. In the same way, faults in the reconfiguration control logic of FPGAs may interrupt the reprogramming functionality, thus requiring a global reset to restore the correct state of the device.

[4] provides a categorization of failures that SEFI can induce in FPGAs. In general, SEFI is not accompanied by a high current consuming condition, despite the SEL effect.

SEL

Differently from the others SEEs, SEL is a permanent error that generates an increase of current. SEL is the results of a parasitic PNPN thyristor within a CMOS [36]. In some cases, the latch-up can be clearable with a power cycling.

2.4.2 Total Ionizing Dose

Differently from the SEEs, TID is the effect of the accumulation of the charge injected by radiation in the oxide layer of transistors in CMOS circuits. TIDs in space and avionic applications are mainly due to the effects of protons and electrons and the consequent secondary particles generated by the interaction of the former with the device [5].
TID effects are usually measured in Radiation Absorbed Dose (rad); a rad is equivalent to 0.01 Gy (Gy = $\frac{J}{kg} = \frac{m^2}{s^2}$)[100].

The amount of accumulated charge depends on the exposure time, the flux of the particles and their Linear Energy Transfer (LET), i.e., the amount of energy that ionizing particles transfer to the material traversed per unit area [5]. TID induces charge accumulation and displacement damages that, together, lead to different malfunctions.

First, a global worsening of the device performance is registered, decreasing the performance and increasing power consumption. In memory circuits, TIDs affect the sensitivity of the logic states of memory cells asymmetrically, causing an imbalance. This effect is due to mobility and transistors threshold changes resulting from ionizing radiation.

The second effect of TID is the change in the SEE sensitiveness; specifically, the accumulated charge within the crystal lattice of the device can make the device more sensitive to SEEs. One consequence is that SEUs can cause "stuck-bits", which

are memory cells whose value is modified by an SEU, however, because of the ionizing dose, their correct value cannot be restored.

In general, TID effects can be annealed by heating the device, to provide enough energy to the crystalline lattice; in this way atomic locations can be restored and trapped charges can be released. Of course, it is hard to apply this kind of method safely in space missions.

In conclusion, TIDs turn in these effects on a device: performance degradation, power consumption increase, and programmability loss. The first effect leads to having slower devices, whose maximum operating frequency is reduced. The second effect is caused by the leakage currents in a transistor, which increase power consumption. Finally, TIDs can lead to losing reprogrammability in the FPGA configuration memory, turning in a permanent fault [5].

2.4.3 Radiation Sensitiveness on SRAM-based FPGAs

First in the 1950s, it was documented that radiations can have adverse effects on electronic circuits. In the next years, the errors were so rare that the study of SEUs was just an academic research. Later, circuit dimensions got smaller, hence, voltages were reduced with process shrinks, the stored charge at a node became smaller and smaller, and the error rate became more significant. Moreover, with the advent of the modern ICs, the possibility of a radiation-induced error grew; this of the case of the FPGAs devices [53].

From the introduction of the first FPGA devices, Xilinx considered their sensitiveness to radiation. Therefore, in 2002 Xilinx and the Jet Propulsion Laboratory founded the *Xilinx Radiation Test Consortium (XRTC)*, which aims to investigate and assess radiation-induces effects and studies methodology to mitigate them.

Xilinx provides studies and mitigation techniques for both commercial and space-grade devices; Table 2.4 shows a summarized result of SEU and TID effects. The TIDs, as explained in Section 2.4.2, are represented in rad. Instead, SEUs are represented in FIT/MB (FIT, Failure in time per million hours) [126].

COTS devices

In the last 15 years, the XRTC started a set of experiments targeting different FPGA devices, constantly sharing the updated results [126]. All the Xilinx FPGAs (commercial, industrial, and military) are qualified for their resistance to radiations. This is possible performing extensive testing in accelerated neutron beams (at the Los Alamos Neutron Science Center (LANSCE)).

In 2005 the results of the Xilinx Rosetta experiment [70; 71] were first published. Rosetta relies on continuous tests on FPGA devices, placed in 10 different world locations, at various altitude (from -490 to 3800 meters). Moreover, for each Xilinx FPGA family, around 300 devices are under test.

Table 2.4: Sensitiveness of Xilinx FPGAs to SEUs and TIDs.

	Device	Tech Node	SEU Conf. Memory	TID
COTS	Virtex-4	90nm	263 Fit/MB [126]	300 Krad(Si) [43]
	Virtex-5	65nm	165 Fit/MB [126]	340 Krad(Si) [43]
	Spartan-6	45nm	179 Fit/MB [126]	380 Krad(Si) [43]
	Virtex-6	40nm	105 Fit/MB[126]	380 Krad(Si) [43]
	7 Series	28nm	85 Fit/MB [126]	500 Krad(Si) [68]
Space-grade	Virtex-4QV	90nm	263 Fit/MB [126]	300 Krad(Si) [136]
	Virtex-5QV	65nm	Immune [135]	1 Mrad(Si) [135]

The main goal of the experiment is to investigate the real effects of atmospheric SEUs on FPGAs, which were previously just estimated [167]. Of courses, the SEU rates change and vary according to the environment is operating in; however, this test gives homogeneous results about the sensitiveness of the COTS FPGA devices to radiations. The results of this experiment are summarized in Table 2.4.

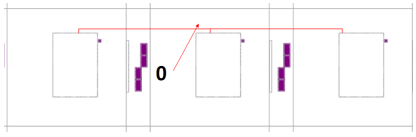
As presented in Table 2.4, it is possible to see how modern devices are less tolerant to radiation effects (from 263 to 85 FIT/Mb); these results directly depend on the CMOS process technology. Moreover, with the shrinking of the CMOS, also the TID effects decreased, going from 300 Krad to 380 Krad. Thanks to their decreasing sensitiveness to radiations, these commercial devices are becoming more attracting and valuable to be applied in harsh-environments.

Space-grade devices

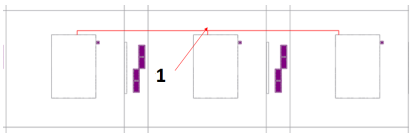
Space-grade devices are specially designed for harsh environments. They are immune to SELs and in some cases immune to SEUs as well. These devices were the first SRAM-based FPGAs utilized in space applications.

It is worth to notice that, compared to the commercial ones, space-grade devices require longer design and verification processes. For example, the first Virtex-4 space-grade devices came three years after the COTS Virtex-4; about Virtex-5, the space-grade device has been produced after four years. This decision can be motivated by the high-production-cost, the increasing demand of computational power, and the increasing reliability of COTS devices. In 2016, it has been announced the introduction of a new space-grade device, the RT Zynq UltraScale+ MPSoC.

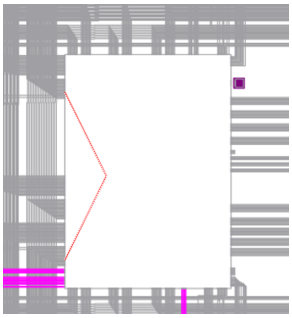
As mentioned in Section 2.1.7, Virtex-4QV devices are radiation tolerance, therefore, they are immune just to certain types of radiation effects. In the specific, these devices are sensible to SEU effects. Therefore, Virtex-4QV FPGAs need extra mitigation techniques, such as TMR and memory scrubbing.



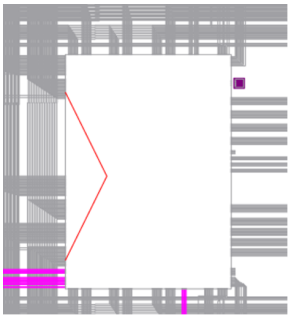
(a) stuck-at-0 (wires).



(b) stuck-at-1 (wires).



(c) stuck-off (PIP).



(d) stuck-on (PIP).

Figure 2.14: Permanent fault effect cases.

2.4.4 Permanent Faults in Routing Resources

Radiations in space can cause both instantaneous (SEUs and SETs) and long-term (TID) damages in electronic devices. In FPGAs, the routing resources represent up to 90% of the whole chip area [10]. When an SEU or a permanent fault occurs in an FPGA device, the fault may affect either the routing resource itself, i.e., the fault directly damages Physical wires (PWs), or the configuration logic associated with routing resources, i.e., the fault affects PIPs.

In the following, four categories of possible faults are presented (see Figure 2.14): *stuck-at-0*, *stuck-at-1*, *stuck-off*, and *stuck-on*. The presented routing resource fault model is based on the works described in [86; 111; 112].

First, the faults are grouped according to which routing resource is affected, either *wire faults* or *PIP faults*:

- *Wires fault stuck-at-0 and stuck-at-1*: a PW that connects two or more switch matrices (SMs) is stuck-at-0 (Figure 2.14a) or stuck-at-1 (Figure 2.14b). It can be caused by a permanent fault that affects the routing resources of the SM.

- *PIP faults stuck-off* and stuck-on: when a permanent fault affects a bit of the configuration memory, this can turn into a permanent fault of a PIP (Figure 2.14c and Figure 2.14d).

These concepts are extensively used in the presented OLT(RE)^2 flow (Chapter 7). Its target is to verify the correct behavior of PWs and PIPs of FPGAs, proving that the resources are free of stuck-at faults. In the following, it is presented how these faults can affect a design.

Faults Effects on Design

A routing resource fault can have different effects in an FPGA design. The most of the configuration memory bits are related to the SMs, which control the routing resources. Each net of a circuit is realized by connections of logic modules through PIPs.

An SEU (or permanent fault) in the configuration bit that controls a PIP can alter or interrupt the propagation of one or more signals. The schematic representation of the effect scenarios can be described considering the original interconnection condition, illustrated in Figure 2.15, that provides the implementation of two different routing nets `net1B` and `net4D` using the two PIPs `1`→`B` and `4`→`D` respectively. All the possible effects of a fault are then shown in Figure 2.16.

In the following the different cases are considered:

- *Open*: the PIP corresponding to the `net4D` is not programmed anymore. Therefore, `4` and `D` are not connected. There are two cases of open errors. The first case is illustrated in Figure 2.16a where the `net4D` is deleted. The second case is illustrated in Figure 2.16b; the `net4D` is deleted and a new net

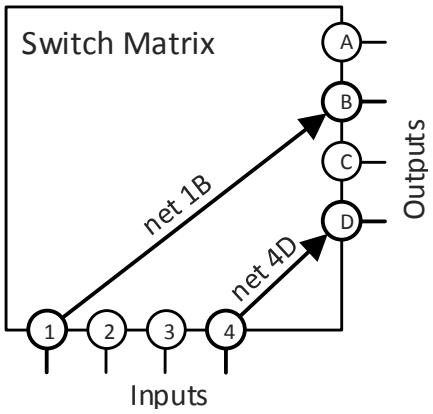
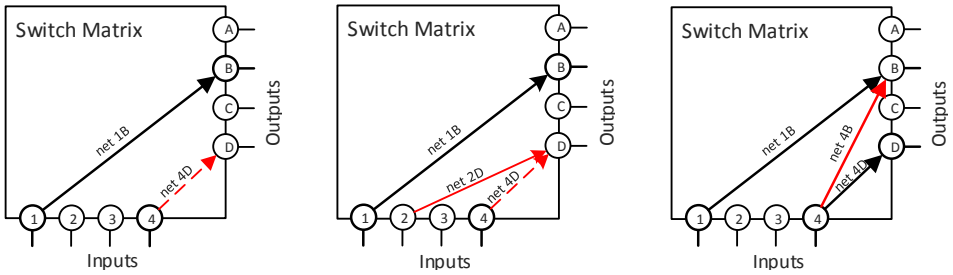


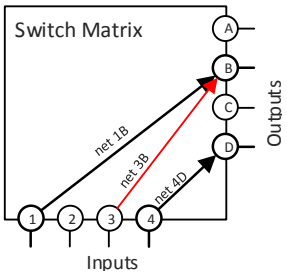
Figure 2.15: Routing condition without error. The figure represents a simplified version of an SM with four inWires and four outwires [5].



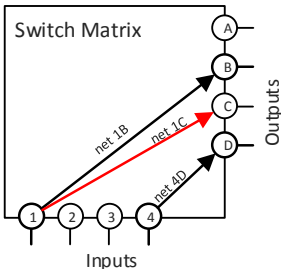
(a) Open effect, first case.

(b) Open effect, second case.

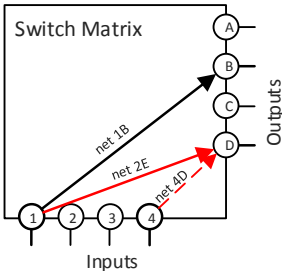
(c) Conflict effect.



(d) Input Antenna effect.



(e) Output Antenna effect.



(f) Bridge effect.

Figure 2.16: Permanent fault effect cases [5].

(`net2D`) connects an unused input wire to the previously used output; the signal `net2D` has a logic value that is not identifiable.

- *Conflict*: a new PIP, corresponding to the `net4B`, is added between an input wire and an output wire, both previously used. Figure 2.16c shows this case. The new PIP creates a conflict on the output `wire B`.
- *Input Antenna*: a new PIP, corresponding to the `net3B` is added between an unused input wire and an active output wire, as illustrated in Figure 2.16d. The new PIP can influence the behavior of the output wire depending on the logic output value assumed by the wires.
- *Output Antenna*: a new PIP, corresponding to the `net1C` is added between an active input wire and an unused output wire (as illustrated in Figure 2.16e). The new PIP does not influence the behavior of the implemented circuits.
- *Bridge*: The PIP corresponding to the `net4D` is disabled while a new PIP,

corresponding to the `net2E`, is instantiated between an active input wire and the output wires of the previously used `net4D`, as depicted in Figure 2.16f. The circuit behavior is modified.

If a fault modifies the routing of the FPGA, without affecting the design of the system, this effect can be categorized in:

- *Tolerant*: an activated PIP that is not connected to a net. Consequently, the modification of the bit in the configuration memory has no impact on the behavior of the design.
- *Unrouted*: the modification of the PIP cannot be classified in any of the considered classes.

More details about the effects of permanent faults can be found in [5]. Table 2.5 shows how a specific fault on a routing resource can turn in one or more effects presented above. OLT(RE)^2, presented in Chapter 7, considers these effects in order to detect possible permanent faults on routing resources.

Table 2.5: The relation between a permanent fault and its effect [5].

Permanent Fault	Permanent Fault Effect
Stuck-at-1 (Wire)	Open
Stuck-at-0 (Wire)	Open
Stuck-off (PIP)	Open
Stuck-on (PIP)	Conflict, Antenna, Bridge

3 State of the Art

This chapter gives an extensive comparison of the thesis with the state of the art. This thesis provides platforms and tools that help the integration of Dynamic Partial Reconfiguration (DPR) in space-applications; moreover, it covers different aspects of the DPR, considering the missing instruments and tools in the state of the art. As main goal, the presented novel approaches always target the easy integration with existing platform/tools.

Section 3.1 gives the motivation for the need of a new and novel general purpose database for Xilinx FPGAs, which is based on the XDL and FPGA-Edline language: the *Datastructure for Xilinx FPGAs (DXF)*. In the last decade, different research groups provided proprietary and open source databases that allow manipulating Xilinx designs.

Section 3.2 presents the tools that enable the creation of DPR systems. The analysis spans the history of the Xilinx support of DPR; in addition, it is provided a description of how other works tried to create different DPR systems. Moreover, a strong motivation for the need of a new and novel DPR flow (*INDRA 2.0*) and an HDL-based communication macro generator (*DHHarMa*), is given.

Section 3.3 presents how the space-community started to adopt DPR on their systems and that are the current target applications. In order to help the integration of DPR in space-systems, this thesis presents the novel *Dynamic Reconfigurable Processing Module (DRPM)* platform, which allows investigating the use of run-time reconfiguration in real space-mission scenarios.

Section 3.4 focuses on permanent faults that can affect FPGAs. In particular, it provides information regarding currents on-line and off-line test of FPGA resources. Finally, the motivations for introducing a new testing tool, *On-Line Testing of Permanent Radiation Effects in Reconfigurable System (OLT(RE)^2)*, are given.

3.1 XDL-based databases and APIs

As explained in Section 2.3, Xilinx provides proper tools to synthesize and configure a certain application on an SRAM-based FPGA. In addition, Xilinx created an intermediate language, the XDL, which can be utilized to integrate custom features in the official ISE tool flow. Therefore, researchers started to create a database from the XDLRC report (presented in Section 2.3.3).

XDL representation is provided in an ASCII-text style. Therefore, the first effort of researchers was to parse these text files and provide Application Programming Interfaces (APIs) to produce a user-friendly mechanism to modify XDL-based designs.

The presented *DXF* database provides functions and APIs to modify the XDL-based design as well. This thesis focuses on the creation of the PW database of DXF Section 5.3.4. Furthermore, DXF is able to directly modify the design in NCD format, utilizing the FPGA-Edline language. More information about DXF are provided in Section 5.2.1; implementation details are provided in [98].

In the following, the related works are presented, highlighting the advantages and disadvantages compared to DXF database.

3.1.1 ReCoBus and GoAhead

ReCoBus tool [60] was released in 2008; its successor is GoAhead, which is available since 2012. ReCoBus is one of the first works that is XDL-based. In fact, the supported FPGAs are the outdated Virtex-2, Virtex-2 Pro, and Spartan-3. For each of these devices, a file in a proprietary format has been created: a **.binFPGA* file.

The functionalities of ReCoBus are presented in Section 3.2.3. About the parsing of the XDLRC FPGA representation and XDL designs, no information are provided. These tools have integrated APIs to modify XDL designs, placing components and changing placing and routing information.

GoAhead [9] is also operating with a proprietary representation of the FPGA structure (**.binFPGA* format) and it is able to provide modifications of XDL designs, exactly like ReCoBus. Differently, this tool supports the Virtex-5, Virtex-6 and Spartan-6 FPGAs. No specific information on the FPGA data structure is available.

3.1.2 RapidSmith

RapidSmith [65; 67] framework consists of a set of tools written in Java that aims to provide researchers with an easy-to-use platform to try out experimental ideas and algorithms on Xilinx FPGAs. Released in 2011, it is the first work that provides general-purpose APIs for modifying an XDL design. RapidSmith is an XDL-based tool and it allows importing XDL files, manipulate, place, route, and export designs among a variety of design transformations.

Moreover, it also contains functionalities that can parse/export bitstreams at the packet level; in the specific, the manipulation of the bitstream is done according to the official Xilinx documentation, which provides just information about the header of bitstream packets (bitstream structure is presented in Section 2.1.4). Therefore, RapidSmith is unable to manipulate the configuration bitstream, changing routing resources or logic configuration; manipulation of the bitstream just refers to the frame addresses.

According to its documentation [67], RapidSmith supports all the Virtex, Spartan, and 7 Series FPGAs families. For all these devices, APIs allow modifying the placement of a certain design. On the contrary, the routing APIs support just Virtex-4 and Virtex-5 families.

Finally, RapidSmith provides a Graphical User Interface (GUI) that allows browsing an FPGA design graphically; this GUI is comparable to the FPGA Editor and PlanAhead. RapidSmith APIs are used in the Dreams tool (presented in Section 3.2.5).

3.1.3 Torc

Tools for Open Reconfigurable Computing (TORC) [29; 109] is a C++ open-source framework that allows reading, write and manipulating XDL-based design and provides wire and logic information of Xilinx FPGAs, creating easy-to-read databases based on the XDLRC reports (presented in Section 2.3.3).

Published in 2011, this tool relies on previous tools, which target just outdated FPGAs: TORC database is based on the ADB project (Alternative wire device DataBase for Xilinx FPGA) [108]. Moreover, TORC integrates and enhances the functionality of JBits; a Java-based software, which provides APIs to access Xilinx FPGAs bitstream, modifying the logic and routing structure of a design.

For each step of the design implementation, TORC provides specific classes as routines. Furthermore, Torc integrates functionalities to build an own CAD tool for specific target systems; for example, the project OpenPR (presented in Section 3.2.4) relies on this tool. TORC can support a wider range of devices: Virtex, Virtex-E, Virtex-2, Virtex-2 Pro, Virtex-4, Virtex-5, Virtex-6, Spartan-3, and Spartan-6.

3.1.4 Tincr

As discussed in Section 2.3.4, Vivado replaced the standard design flow ISE, starting from the 7 Series (just for this series, ISE is provided as well). In Vivado, XDL is not supported anymore. However, the meta-information of a design can be extracted with the Tcl scripting language.

Tincr [120], introduced in 2014, is a suite of Tcl libraries written for Xilinx Vivado IDE. The goal of Tincr is to enable users and researchers to build their own CAD tools on top of Vivado, similarly to the XDL language for the ISE flow. Hence, Tincr supports 7 Series devices and Zynq.

Tincr provides two different libraries: *TincrCAD* and *TincrIO*. TincrCAD is a Tcl-based API built on top of native Vivado Tcl commands. It consists of a set of APIs that are common in the development of custom CAD tools; this provides the user higher levels of abstraction, performance gains, and a greater wealth of information. TincrIO comprises a set of APIs for getting design and device architecture data out of the Vivado tool. TincrIO enables the users to generate

Table 3.1: Comparison of CAD tools based on Xilinx intermediate languages.

		ISE		Vivado	
	published in	XDL-based	FPGA Edline		FPGAs support
ReCoBus [60]	2008	✔	✘	✘	V-II, V-II Pro, S3
GoAhead [9]	2012	✔	✘	✘	V5, V6
RapidSmith [66]	2011	✔	✘	✘	From V-2 to 7s, Zynq
Torc [109]	2011	✔	✘	✘	From V-2 to V6
Tincr [120]	2014	✘	✘	✔	7 Series, Zynq
DXF	2011	✔	✔	✘	V4, V5, V6, S6, 7S

XDLRC device descriptions and export designs out of Vivado into an XDL format style.

The developers of Tincr have utilized the TincrIO libraries to create an equivalent XDLRC representation of the 7 Series FPGAs. Investigating the possibility to convert the information extracted with Tcl in XDLRC can give the possibility to use the XDL-based tools for the FPGA devices that are supported just with Vivado [140].

3.1.5 Comparison

In this thesis, an open-source database of the Xilinx devices is utilized: Datastructure for Xilinx FPGAs (DXF). Table 3.1 presents a comparison among DXF with the existing tools/flows discussed in this section.

The table shows that:

- **XDL-based integration**: in ISE tool flow, XDL provides the main mechanism for gaining external access to design data. Therefore, most of the presented tools are based on this intermediate language (except for Tincr that operates with Vivado).

- **FPGA-Edline-based APIs**: this functionality is missing in all the existing tools. This new capability inserted with DXF allows modifying designs directly in the native NCD format; this is required when a full design needs to be modified, and the XDL file cannot be created. As described in Section 2.3, XDL conversion is suitable for small designs/hard macros. On the contrary, for full designs, in the conversion NCD-XDL-NCD some information are lost (e.g., IOBs and Digital Clock Managers (DCMs) settings).

- **FPGAs support** of different Xilinx families: DXF provides support for different kind of FPGAs. Apart for the compatibility with the out-dated Virtex-2 FPGAs, DXF supports a wide range of families, providing full functionality for all of them. On the contrary, as discussed for RapidSmith in Section 3.1.2, even if all the FPGAs from Virtex-2 are considered, the routing functionalities are supported just in Virtex-5 and Virtex-6.

- **Vivado-integration**: Tincr is the only tool that provides with integration for Vivado. As an advantage, this APIs can be easily adapt to support the future devices. On the contrary, Tincr can not be used with older devices, which are supported just by ISE.

DXF database provides an extensive set of APIs and functionalities for modifying Xilinx full designs and hard macros. It supports five families of Xilinx FPGAs, which corresponds to 208 different devices in total. In this thesis, DXF is utilized for the creation of homogeneous communication infrastructures (Chapter 6), the re-router part of INDRA 2.0 (PSRerouter, Chapter 5) and for the generation of testing circuits that detect permanent faults (Chapter 7).

3.2 Dynamic Partial Reconfiguration Tools

Section 3.1 has discussed how researchers created databases and APIs to enable more flexibility in design changes. Of course, this gives the capability of providing features that are not embedded in the official Xilinx design flow.

As explained in Section 2.3, Xilinx developed an advanced design flow that follows users from the hardware high-level representation to the bitstream configuration. Nevertheless, researchers started to investigate the remarkable advantages of using FPGAs in a DPR environment.

DPR in the past years has been slowly integrated into the ISE flow. Therefore, different groups of researchers started to create tools aiming to add support for this functionality; INDRA 2.0, is one of them. In the specific, the Design flow for Homogeneous Hard Macros (DHHarMa), which is part of the INDRA 2.0 flow, aims to generate homogeneous hard macro for tile-based DPR systems, giving the possibility to create it starting from an HDL description.

Communication infrastructures are a key part of a DPR system. Different implementations can affect the property of reconfigurable systems (e.g., run-time reconfiguration and module relocation). In the following, related works that provide CAD tools oriented to DPR systems are presented.

3.2.1 Xilinx ISE DPR

In the following, it is presented how, in the last decade, Xilinx has modified its default design flow to integrate DPR. This topic is also discussed in Section 2.2. Xilinx introduced the first support of DPR in 2006, with ISE 9.2*i*; in the specific, the Early Access Partial Reconfiguration (EAPR) was introduced [127]. This Xilinx toolkit, even if was not directly integrated into the ISE design flow, provided the first official instruments to create DPR systems on Xilinx FPGAs.

EAPR provided a methodology to generate a slot-based Partial Reconfigurable Region (PR Region), which can be reconfigured with one or more Partial Reconfigurable Module (PR Module). At that time, the communication infrastructure among the static region and PR Region was created thanks to *bus-macros*; pre-P&R components containing two terminals, which are placed in a fix position. Two different types of bus-macros were available: *narrow* and *wide*. Narrow bus-macros had just a width of 2 CLBs; the *wide* ones had a width of 4 CLBs.

One of the advantage to utilize bus-macros was the possibility to create DPR systems that allow relocation of PR Modules. However, this way to utilize bus-macros introduces high logic utilization and delay overhead.

Since 2011, bus-macros communication mechanism and EAPR plug-in are not supported by Xilinx anymore. With the introduction of ISE 12.1 [34; 131], Xilinx embedded the DPR functionality directly in the official design flow; the new method is called *Xilinx PR*. Moreover, to overcome the limits of bus-macros, a new way of creating the communication infrastructure was introduced. Bus-macro were replaced by *Proxy Logic* (also called Proxy LUTs).

Proxy Logic is a single LUT1 element, belonging to the static area, which is placed in a fixed position of the reconfigurable area. The routing between Proxy Logic and the static region is maintained constant for both static and dynamic implementation design. An advantage of this new mechanism is to avoid of creating/placing the bus-macros before that the actual static design and modules are synthesized. The user just needs to define the connection points of the communication between static and reconfigurable regions.

In contrast, the ISE does not prevent the routing of Proxy Logic to be the same in the area of the same type; in addition, ISE allows static nets the cross reconfigurable areas. For these reasons, this new approach does not support PR Module relocation (i.e., bitstream relocation).

Furthermore, with the new Xilinx PR flow, static design and PR Modules are designed at the same time. If on the one hand, this can allow an optimal result for the considered design, on the other hand, it prohibits the creation of new PR Modules at a later time.

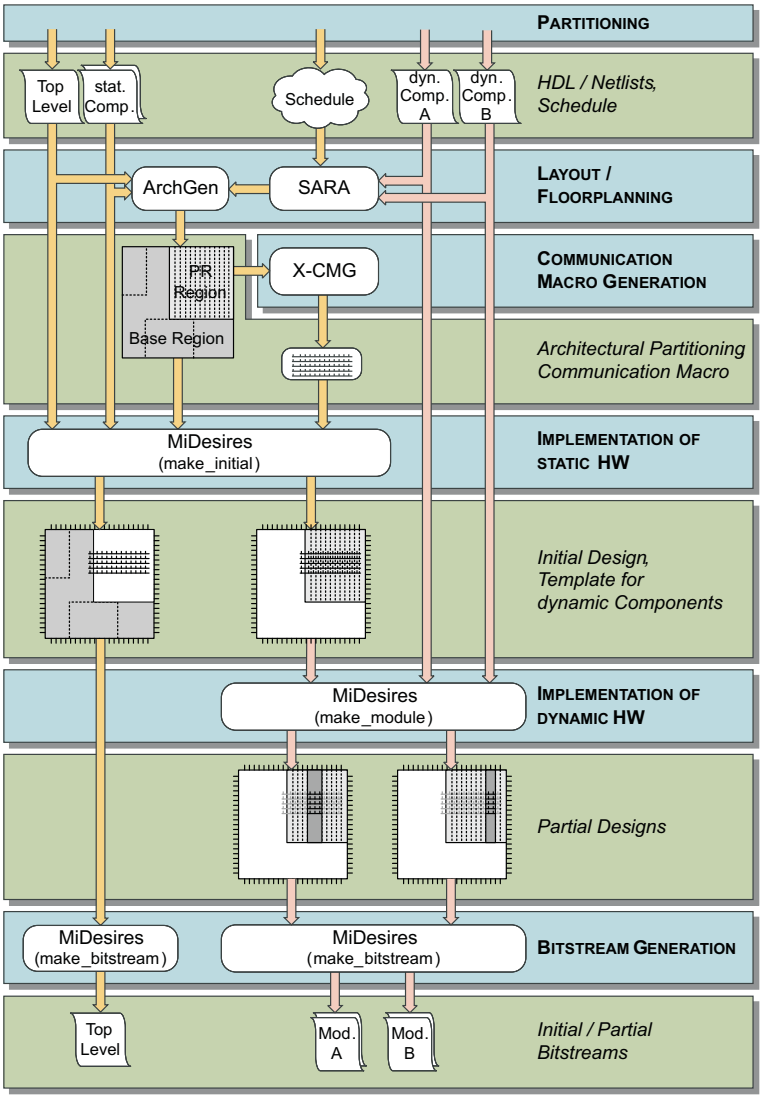


Figure 3.1: INDRA flow overview [45].

3.2.2 INDRA

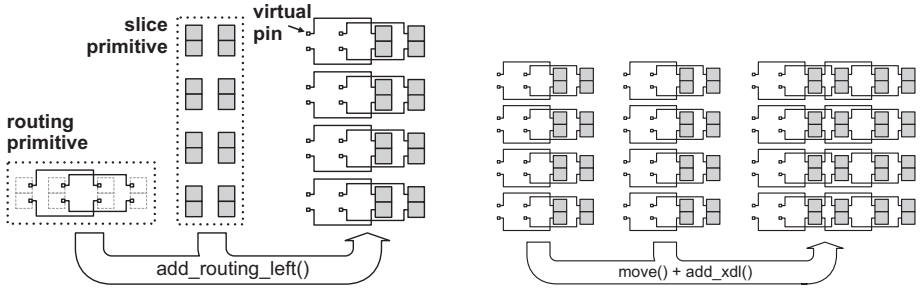
INDRA [45] is an INtegrated Design flow for Reconfigurable Architectures, which offers the possibility to utilize DPR. This flow has been introduced to overcome the limits of Xilinx EAPR. Figure 3.1 shows an overview of the flow; INDRA takes in input the FPGA partitioning and the HDL of static design and PR Modules. After that the components are synthesized, according to the PR Tile, a suitable communication infrastructure that preserve homogeneity and run-time configuration is generated.

XDL-based Communication Macro Generator (X-CMG) [44] is used to generate this kind of communication macro. In the following, its features and limitations are presented. In this thesis, an updated version of INDRA is presented: INDRA 2.0. It is worth to mention that, DHHarMa has been first designed to overcome the limits of the X-CMG tool.

XDL-based Communication Macro Generator (X-CMG)

X-CMG [44] is an XDL-based Communication Macro Generator that allows generating a homogeneous communication infrastructure, utilizing the logic presented in Section 2.2.4. X-CMG utilizes multi-level, primitive-based communication macro generation approach.

In order to build a homogeneous communication infrastructure, this application used primitives that are combined together. According to the embedded macro communication infrastructure, the primitives can be used to implement dedicated signals and shared signals.



(a) Routing Primitive of X-CMG routing.

(b) Connecting the primitives in a homogeneous communication infrastructure.

Figure 3.2: X-CMG routing capabilities for slice based routing [44].

The Figure 3.2 shows how it is possible to create a communication channel combining routing primitive; in Figure 3.2a are depicted the primitive utilized by the router, while the Figure 3.2b explains how the single primitive can be connected to create a routing channel.

This kind of solution utilizes a high amount of resources; along the communication channel, most of the slices are used in order connect the routing primitives among them (as depicted in Figure 3.2b). On the contrary, in DHHarMa communication hard macros, the utilized slices are only the ones involved in the communication logic. In fact, the connection among PR Tile is made creating a dedicated routing for every case, taking in account the homogeneity property of the system.

3.2.3 ReCoBus and GoAhead

The proprietary XDL-based databases of these tools are presented in Section 3.1.1. As mentioned, GoAhead [9] is the successor of ReCoBus [60]; moreover, the two tools provides the same functionalities. The difference is in the targeted FPGAs: ReCoBus targets Virtex-2, Virtex-2 Pro, and Spartan-3; instead, GoAhead supports Virtex-5 and Virtex-6 FPGAs.

ReCoBus and GoAhead allow building complex monolithic communication macros for easily implementing run-time reconfigurable systems on Xilinx FPGAs. They integrate into Xilinx ISE and support the physical implementation of initial static systems as well as the implementation of the PR Modules. On the one hand, the generated communication infrastructure respects homogeneity constraints given by the reconfigurable areas; on the other hand, it is impossible to create a custom communication infrastructure starting from an HDL code.

Regarding the routing phase, these tools just route the nets utilizing a specific portion of the communication channel infrastructures (i.e., defined macro blocks). In this way, the routing is not custom generated for each connection; instead, a certain communication channel is created combining blocks of routing resources. This kind of solution turns in a higher utilization of logical resources and longer delay of the communication channels. DHHarMa gives the possibility to have a homogeneous communication infrastructure (as ReCoBus and GoAhead), combined with the advantage of starting from HDL language; this allows optimization in all the computational steps: mapping, placing, and routing.

3.2.4 OpenPR

OpenPR [104] toolkit was introduced in 2011. This open-source work is strongly related to the Xilinx EAPR tool and provides similar functionalities. Its main goal is to provide EAPR functionalities in a more automatic way, overcoming some limitation of it.

OpenPR is an XDL-based toolkit, written in C/C++, that allows the creation of DPR systems on Virtex-4 and Virtex-5 devices. It relies on the Torc APIs (described in Section 3.1.3) for editing Xilinx XDL designs.

This work provides two remarkable functionalities:

- *Automatic bus-macro placement*: the designer can define the area and the number of the communication channel, then, OpenPR takes care about the bus-macros placement.

- *Avoid crossing of static nets through the reconfigurable area*: as explained in Section 3.2.1, this problem was already present in the EAPR tool, and it also remained in its successor: the Xilinx PR of ISE 12.1. OpenPR solves this issue utilizing a *Router Blocker Macro*. This kind of macro is placed in the PR Region, before that the PAR of the static design is executed. In this way, the macro occupies all the routing resources within the PR Region; therefore, the nets of the static design can not utilize any resources within the PR Region.

During the publication of this work Xilinx decided to replace the EAPR flow with the Xilinx PR, which does not support bus-macros. For this reason, the tool has not been adapted to newer FPGAs families.

3.2.5 Dreams

Dreams is a Tool for the design of Dynamically Reconfigurable Embedded and Modular Systems [83]. This tool was published in 2012. It is based on the RapidSmith libraries, describe in Section 3.1.2. Therefore, the tool is XDL-based as well.

The tool flow provides some extra steps and partitioning compared to the standard Xilinx PR. In the specific, Dreams is the only work that embeds a routing algorithm of the nets that realize DPR systems, which support PR Module relocation.Dreams takes in input a P&R static and dynamic design, an XML file where the property of the DPR system are given. The XML file allows having at the end an automatized flow.

As output, Dreams provides the bitstreams of the DPR design (static and reconfigurable) that supports the relocation of the modules. In addition, another important feature is the possibility to introduce a new PR Module at a later time.

As mentioned in Section 3.1.2, RapidSmith provides just routing APIs for the Virtex-4 and Virtex-5 families. Moreover, this kind of DPR systems supports just an LMBT communication infrastructure.

Table 3.2: Comparison of DPR tools.

	Communication Infrastructure	Module Reloc.	Supported FPGAs	Special Feature
Xilinx PR [34]	proxy logic based	✘	from V4 to 7S	official flow
ReCoBus [60]	macro-blocks	✔	V-II, V-II Pro, S3	GUI
GoAhead [9]	macro-blocks	✔	V5, V6	GUI
OpenPR [104]	macro-blocks	✔	V4, V5	Router Blocker Macro
Dreams [83]	LMBT	✔	Virtex-5	custom Router
INDRA [45]	macro-blocks	✔	Virtex-2	embedded macros
INDRA 2.0	HDL-based	✔	V4, V5, V6, S6	DHHarMa, PSRerouter

3.2.6 Comparison

In the following, a comparison of the DPR flows is provided. Table 3.2 shows the main features of them. As mentioned in the introduction, the main leak of the official Xilinx PR is the possibility of creating DPR systems that are capable of supporting PR Module relocation. For this reason, researchers put the main efforts to introduce this remarkable feature on a reconfigurable system.

All the tools rely on the XDL-based databases and APIs presented in Section 3.1. Different approaches and different communication infrastructures are proposed. Table 3.2 shows that most of the works exploit the concatenation of macro-blocks to preserve homogeneity of the reconfigurable regions.

Dreams is the only work that has a dedicated router, which takes care of establishing the homogeneity in a certain DPR scenario. Nevertheless, this flow is oriented just on LMBT communication infrastructure.

To overcome the limitations of these tools, a new flow INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0) is presented in this thesis. The work has been made in collaboration with [98]. INDRA 2.0 approach unifies the features of the mentioned tools, within one single flow:

- **HDL-based Communication Infrastructure Generation**: despite all the existing approaches, DHHarMa can create a homogeneous communication infrastructure, starting from a user-define HDL representation. This gives the possibility to use the tools in different scenarios, avoiding being related to a single communication infrastructure methodology.

- **Support of PR Module Relocation**: DHHarMa guarantees that in reconfigurable regions of the same type, the communication infrastructure utilizes

the logical and routing resources in the same relative position. It provides dedicated packing, placing, and routing mechanism.

- **Reroute of static nets directly in NCD format**: XDL tool allows converting an NCD in a human readable format (XDL), ad vice-versa. In some designs, the conversion NCD-XDL-NCD is not possible because some design information are lost (as mentioned in Section 2.3.3). For this reason, INDRA 2.0 provides the unique feature to allow rerouting of specific nets, utilizing the FPGA-Edline scripting language, modifying the net directly on the NCD format; this step is provided by the PSRerouter.
- **A Wide range of FPGAs supported**: INDRA 2.0 is the only flow that targets the widest range of currently used Xilinx FPGAs. Compared to the Xilinx PR just the 7 Series family is not supported. This allows the user for utilizing the approach considering which FPGA family suites better the final requirements.

3.3 Reconfiguration in Space Applications

FPGAs are nowadays extensively used in space missions. Space applications are never mass products, therefore, in most of the cases, the use of ASICs is not the best solution in term of price and availability. Thanks to their general purpose structure, FPGAs target well the need of space: a device that can be configured as required and provides high performances [40; 80].

The first FPGAs utilized in space missions were antifuse. These kinds of devices can be just configured once through a hard configuration; therefore, they are more tolerant to radiations. The current antifuse FPGAs utilized in space are the RTAX devices from Microsemi [100]. On the contrary, SRAM-based FPGAs, they are more sensible to SEEs and TIDs; their most critical part is the configuration memory.

Differently from the antifuse FPGAs, the SRAM-based ones have the remarkable property to allow multiple reconfigurations. This feature is interesting in space missions and can be performed in both on-ground and in-flight scenarios; in the first case the SRAM-based FPGA can be configured at later phases of the development of space system; in the second case, the SRAM-based FPGA can be reconfigured after the launch. In addition, the SRAM-based FPGA can be reconfigured with a different application once is needed. Furthermore, the system can embed self-healing mechanisms, which can recover the SRAM-based functionality once a SEE or a permanent fault occurs.

For these reasons, space is considered a challenging environment. On the one hand, the electronics systems need special mechanisms to mitigate radiations that can compromise the platform (i.e., there is the need of a high reliable system); on

the other hand, a space mission needs a strong planning phase, which permits utilizing a system for a long period (i.e., need reconfigure the system).

In the last decade, manufacture, and researchers tried to cut this trade-off allowing to have a reliable and safe use of SRAM-based FPGAs in space missions. However, SRAM-based FPGAs are not used in critical parts of space missions due to their sensitiveness to radiation effects; instead, they are extensively used in non-critical parts, e.g., payload processing.

This thesis presents a novel prototyping platform for space applications: the Dynamic Reconfigurable Processing Module (DRPM). The system can emulate real space mission scenarios, which relies on the DPR.

This section presents the related works that utilize reconfigurable FPGAs in space missions. Moreover, it presents how the COTS devices are becoming more and more utilized in space applications. Finally, the summary gives a comparison between the DRPM system and the related works.

3.3.1 DPR research platforms

The utilization of SRAM-based FPGAs in space applications require specific tests and validation steps to be utilized safely; in addition, the FPGA memory is high sensible to SEEs and TIDs. For these reasons, different research platforms and approaches have been developed in the last decades, which allow porting SRAM-based FPGAs into real space missions: the DRPM is one of them. In the following, the related works are presented, and their main features are summarized in Table 3.3.

Braunschweig DRPM

As the DRPM system presented in this thesis, the Braunschweig DRPM [78] has been developed in collaboration with the European Space Agency (ESA). Both systems share the same starting project requirements, i.e., the creation of a reconfigurable FPGAs-based platform that allows performing payload processing.

The Braunschweig DRPM demonstrator comprises one or more modules; each module is composed of a radiation-hardened reconfiguration controller and two Virtex-4 devices.

The reconfiguration controller, depicted on the left-hand side, is implemented on a reliable antifuse FPGA. It comprises a SoC with an LEON3 CPU and several peripherals, such as memory controllers. The Virtex-4 FPGAs are divided into reconfigurable partitions that are interconnected via a SoCWire routing switch.

A Reliable Reconfigurable Real-Time Operating System (R3TOS)

A Reliable Reconfigurable Real-Time Operating System (R3TOS) was introduced in 2010 [56] and its main target is to provide a reliable reconfigurable real-time operating system. R3TOS [56] presents an infrastructure for executing dedicated hardware tasks on a single reconfigurable FPGA device, achieving flexibility for both gaining system performance and tolerating the occurring faults at run-time.

The real-time operating system offers the possibility to consider a hardware functionality as a task. R3TOS provides the user a set of software routines, which allow abstracting the FPGAs reconfigurable mechanism in a software-based approach, enabling an easy to use DPR. Finally, R3TOS has been tested on a Virtex-4 LX200.

SCARS

The Scalable Self-Configurable Architecture for Reusable Space Systems (SCARS) is a Virtex-5-based DPR system, introduced in 2008 [105]. It is composed of five Virtex-5 FPGAs, each including a MicroBlaze IPcore that is responsible for fault mitigation and detection.

The Virtex-5 FPGAs are interconnected to a master node utilizing a wireless network. The PR FPGAs are partitioned together with redundant copies; the partition is slot-based and provides connection to the MicroBlaze.

Once a fault is detected in a specific slot, the MicroBlaze performs a scrubbing through the ICAP interface; in the case the error persists, the PR Module is relocated either in a different slot of the FPGA or in another FPGA of the system.

3.3.2 In-flight reconfigurable space-missions

The possibility to reconfigure a device can bring different benefits to space systems. An application can be updated introducing new functionalities or correcting bugs not detected in the test phase; moreover, the system can automatically reconfigure itself once a fault occurs in the system, providing a fast correction and avoiding catastrophic propagation effects.

The Gravity Recovery And Climate Experiment (GRACE) [117] was the first space mission that had on board a reconfigurable Xilinx FPGA: the radiation-hardened (rad-hard) XQR4036XL [134]. Unfortunately, the system was unable neither providing in-flight reconfigurability nor proving SEU mitigation techniques.

The FedSat [94] was the first space mission that introduced in-flight reconfiguration of FPGAs. It was launched in 2002 carrying a rad-hard Xilinx XQR4036XL [134]. The system has also an embedded SEU detector based on a readback controller.

Progressively, these devices were introduced in many other applications. The Mars Exploration Rover Mission (MER) [82] is a robotic space mission that brought two rovers (Spirit and Opportunity) to Mars (launched in 2003). Reconfigurable FPGAs have been extensively used in both landers and rovers [155]. FPGAs has been utilized in landers for pyrotechnics, retrorockets, parachute deployment (in the specific Xilinx XQR4062XL [134]); on the Rovers, Xilinx XQVR1000 [133] has been used in all wheel motor controllers.

In 2005, the Mars Reconnaissance Orbiter (MRO) [76] was launched in orbit. Its camera (named HiRISE camera) is connected CCD Processing, and Memory Module (CPMM), which relies on a rad-hard Xilinx Virtex 300E FPGA [152]. The CPMM performs control, signal processing, and data compression. The system provided readback for SEU detection and scrubbing for SEU correction. In 2011, The Mars Science Laboratory (MSL) and its rover Curiosity were launched. The rover MAHLI uses Xilinx rad-hard Virtex-II FPGAs for image processing.

Apart from these brief overview of the use of reconfigurable SRAM-based FPGAs, in the following two interesting novel applications are presented: the Solar Orbiter and the FOBS satellite. Both systems utilize reconfigurable FPGAs for payload processing, giving the possibility of in-flight reconfiguration.

Solar Orbiter (PHI DPU)

The Solar Orbiter [35] is a mission that intends to explore the sun surface. Its launch is planned for 2018, and it will operate till 2028. The system embeds the Polarimetric and Helioseismic Imager Data Processing Unit (PHI DPU) [20], which contains two 2048x2048 cameras that will provide high image data stream (3.2 Gbit/set). The telemetry link with the earth is just 100 Mbits/set. Therefore, to reduce the data from 3.2 Gbit/set to 100 Mbits/set, the space system process internally the raw data, utilizing a Data Processing Unit (DPU).

DPU system has been developed by the University of Braunschweig, adapting its DRPM platform presented in Section 3.3.1. The DPU is based on two rad-hard Xilinx Virtex-4 XQR4VSX55 and can provide Image Stabilization Systems (ISS), data acquisition, data preprocessing, and Radiative Transfer Equation (RTE) inversion. The system implements different kinds of SEU mitigation providing fine-grain scrubbing as well. In addition, the system can be reconfigured according to the needed tasks.

Two running modes are performed: *Data Acquisition Mode* and *Processing Mode*. In the first case, the two FPGAs are configured with functionalities that provide ISS controlling and data acquisitions; in the second case, the FPGAs provides data preprocessing and RTE inversion functionalities. Therefore, the functionalities are time partitioned in the reconfigurable area. Moreover, the algorithms can be updated during the lifetime of the satellite.

This system is not proving DPR functionalities, since that switching from one

operation mode to another, the whole FPGA is reconfigured. Differently, the presented DRPM system developed in Bielefeld University allows DPR and PR Module relocation.

Fraunhofer On-Board Processor (FOBP)

In 2020 is planned the launch of the Heinrich Hertz Satellite (H2Sat), which will operate in the Geosynchronous Equatorial Orbit. Its expected lifetime is 15 years. The target of the H2Sat is to investigate new software and communication technologies in-orbit, adapting to the new telecommunication standard requirements.

The space system embeds the Fraunhofer On-Board Processor (FOBS) [87], which relies on two SRAM-based Xilinx Virtex-5QV: one master and one slave. The presented FOBS is composed of four hardware modules: a radio frequency card, a power supply unit, and two DSP cards. Each DSP card has an analog to digital converter and a Xilinx Virtex-5QV FPGA.

The system will give the possibility to operate DPR in space. For example, when a new telecommunication standard is released, the on-board processor can be updated with an in-flight DPR. Once the system is powered-up, an external processor configures both FPGAs with an initial bitstream. The master FPGAs is configured with a DPR scenario; the static area contains a SOC controlled by an LEON3-FT. At this point, the master FPGA is capable of performing DPR of itself and full reconfiguration of the slave FPGA. The master FPGA is also in charge of scrub the configuration memory of the slave FPGA against SEUs.

The system can communicate with the earth with two different links: a high reliable virtual telemetry/telecommand (vTM/TC) link with 2 Mbps (gross bit rate) and 1 Mbps (net bit rate); this link utilized for sending new bitstream from earth. The second one is an experimental link that is able to achieve 306 Mbps (gross bit rate).

3.3.3 Commercial FPGAs in Space

As discussed in Section 2.1.7, Xilinx provides dedicated space-grade FPGAs, which allow having either higher fault-tolerance or higher fault-resistance to SEE. At the same time, COTS FPGAs became more resistant to radiation effects for two more reasons: first, the shrinking of the CMOS reduces the sensitiveness of the device to faults; second, many kinds of fault mechanisms were developed, which allow reaching high-fault tolerance. Therefore, thanks to their high-availability, higher performance compared to space-grade devices and their lower cost, COTS FPGAs started to be utilized in space applications.

One example is the Advanced Responsive Tactically Effective Military Imaging Spectrometer (ARTEMIS) [118] reconfigurable payload processor that is on board of the TacSat-3 satellite, which was launched in 2009 and completed its

operation in 2012. The ARTEMIS contained a so-called Responsive Avionics Reconfigurable Computer (RA-RCC) that relies on three COTS Virtex-4 LX160. The system supported in-flight reconfiguration.

SpaceCube

SpaceCube [39] consists of reconfigurable platforms that are based on COTS Xilinx FPGAs. The goal of the SpaceCube program is to provide improvements in on-board computing power while lowering power consumption and cost.

The system is Radiation Hardened By Software (RHBS), which provides protection to radiation's effects. SpaceCube project started in 2006 and is still ongoing; four different types of systems were developed (SpaceCube v.1.0, v.1.5, v2.0-EM and v2.0-FLT).

In total, nine systems were produced and flew in space, utilizing 22 Xilinx COTS FPGAs. The SpaceCube were utilized in many applications: e.g., The SpaceCube v.1.0 was payload processing real-time image tracking and data compression on the Hubble Space Telescope (HST); the mission MISSE-7 intended to investigate radiation effect on Xilinx commercial FPGAs; finally, in the mission STP-H4 a SpaceCube 2.0-EM operated on the International Space Station (ISS), providing process and stream of HD images in real-time.

The first version of the SpaceCube embedded two Xilinx Virtex-4 XC4VFX60. It was launched in 2009 and it utilized for the first time COTS Xilinx Virtex-4 FPGAs. The system mitigated SEEs with a voting system; the FPGAs were connected to a voter implemented in a rad-hard Aeroflex UT6325 FPGA, which compares the outputs of four processing nodes implemented in the two Virtex-4 FPGAs that run in parallel. From the version v1.5, the COTS Xilinx Virtex-5 has been utilized (i.e., XC5VFX130T).

3.3.4 Comparison

This section has presented how the use of reconfigurable FPGAs in space increased in the last decades: rad-hard as well as COTS FPGAs are utilized. Moreover, the possibility to provide in-flight configuration allowed stepping forward in proving computational power into space missions. These works motivate how, on the one hand, the use of this technology is becoming almost legacy in space missions; on the other hand, DPR capabilities are not yet fully exploited.

In Table 3.3 the presented DPR platforms are summarized. Just few systems embed the possibility of DPR, and just one of them supports the relocation of PR Modules (the R3TOS). However, the R3TOS platform has a fixed architecture based on a single FPGA, which does not allow a full integration in a space scenario. On the contrary, the DRPM system comes with different avionic interfaces (e.g.,

Table 3.3: Comparison of DPR platforms for space applications.

	N. FPGAs	FPGA model	DPR	Module Reloc.
DRPM Br [78]	2 - 6	XC4VSX55	✘	✘
SCARS [105]	5	XC5VLX50	✔	✘
R3TOS	1	XC4VLX200	✔	✔
PHI DPU [20]	2	XQR4VSX55	✘	✘
FOBP [97]	2	XQR5VFX130	✔	✘
SC 1.0 [39]	2	XC4VFX60	✔	✘
SC 1.5/2.0 [39]	2	XC5VFX130T	✔	✘
DRPM	1 - 5	XC4VFX100	✔	✔

CAN, SpaceWire, SpaceFibre), which allows advanced prototyping of real space scenarios.

3.4 Testing of Routing Resources

Fault detection in FPGAs devices is a challenged topic, which is present in many phases of the device lifetime. On the one hand, a test is needed after the manufacturing phase (off-line test); on the other hand, the device needs to be monitored during its lifetime with on-line tests.

In the area of fault detection in FPGA devices, much work has been done in the last two decades, addressing hardware structural defects due to the manufacturing process [1; 48; 85; 102]. On the contrary, the problem of detecting faults induced in FPGA devices by the long-term exposure to radiations has not yet exhaustively been explored. The approaches to FPGA testing can be classified into two families: *application-dependent* and *application-independent* ones (see Figure 3.3).

Application-dependent methods, such as [11; 24; 114], focus only on those resources of the FPGA used by a given system. For this reason, these techniques are meant to be applied by the end-user of the FPGA device, either off-line or on-line, after the system design has been defined. The rationale behind these techniques is that an FPGA-based system occupies only a subset of the resources provided by the FPGA device. Therefore, it is enough to demonstrate that the resources used by the implemented system are fault-free to guarantee the correct operation of the system itself. Application-dependent methods have been proposed for in-service testing of both structural defects [24; 114] and SEUs [11]. These kinds of tests suit well non-reconfigurable FPGA systems.

Application-independent methods, such as [1; 48; 85; 102], are meant to detect structural defects due to the manufacturing process of the chip. These approaches

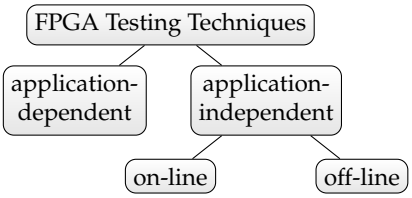


Figure 3.3: FPGA Testing Techniques.

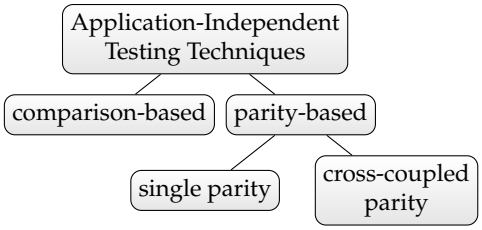


Figure 3.4: Application-Independent Testing Techniques.

are intended to be performed off-line, by the FPGA manufacturer, and they target every possible fault in the architecture without any consideration of which parts of it are used for a given design. These techniques generally employ multiple test configurations and the associated generated test patterns. Each test configuration is intended to test a subset of the possible faults of the chip. This approach is interesting especially for DPR systems.

In this way, a user can make sure that certain FPGAs resources are free of faults before that a specific application is configured in the system. In this section, the existing approaches are presented, highlighting the differences with the OLT(RE)^2 approach.

3.4.1 Fault Detection mechanism

Application-independent techniques exploit multiple configurations of the FPGA, each one intended to test a subset of the resources available in the device. These test configurations are generally composed of a *Test Pattern Generator (TPG)*, which provides input stimuli to the set of resources under test and of an *Output Response Analyzer (ORA)*, which observes the output of the resources under test and determines whether they are faulty or not. These techniques can additionally be divided into two sub-categories: *comparison-based* and *parity-based* [111] (see Figure 3.4).

In the comparison-based techniques, such as the ones presented in [1; 96], the ORA knows both the expected output (associated with the input stimuli generated by the TPG) and the actual output produced by the resources under test; by comparing them, it can determine whether a fault occurred in the system. The main drawback of these techniques is that they are unable to detect faults in the TPG and those faults that do not interfere with the output of the resources under test, i.e., *equivalent faults*.

To overcome these limitations, parity-based testing techniques, such as [113; 165], have been introduced; here the TPG additionally calculates a parity bit

on its outputs, while the ORA calculates the parity bit on the received signals. Comparing these two parity bits the ORA can to detect the occurrence of a fault in the resources under test. Then, the ORA does not need to know the expected output, but it only relies on the parity bit generated by the TPG, therefore, also faults affecting the TPG and the equivalent faults may be detected.

Parity-based testing approaches may be additionally classified in two families: *single parity* [113] and *cross-coupled parity* [165]. In the single parity-based technique, the TPG is an n-bit counter, producing $n + 1$ output bits; the last bit is the parity bit calculated on the previous n-bit. The ORA receives $n + 1$ input bits, calculates the parity on the first n bits and compares it with the received parity bit. The drawback of single parity-based techniques is that some faults in the TPG and some equivalent faults may still not be detected by the ORA; moreover, it is important that the parity bit is sent on a fault-free wire.

In cross-coupled parity-based techniques the TPG is composed of two independent n-bit counters, let us call them `TPG_a` and `TPG_b`; each TPG produces n output bits plus one parity bit. Similarly, the ORA is duplicated: `ORA_a` receives the n input bits from `TPG_a` and the parity bit from `TPG_b`; conversely, `ORA_b` receives the n input bits from `TPG_b` and the parity from `TPG_a`. In this way, all the faults occurring in the TPGs and all the equivalent faults may also be detected. OLT(RE)^2 utilizes a cross-coupled parity-based testing techniques to ensure that all the possible faults are considered.

3.4.2 Off-line application-independent testing

Many works have been presented, which perform a test on FPGA devices just after the fabrication. Apart from the published works, manufactures execute extensive test on their FPGA devices to ensure that are permanent fault free; unfortunately, no information are provided.

Therefore, three different works provided a built-in self-test (BIST) approach to test modern FPGAs. In all the cases, the tests rely on specific testing circuits that are configured off-line on the device. In 2006, [33] presented a BIST approach to test logic resources of Virtex-4 FPGAs. The fault detection mechanism utilized is comparison based. The work shows that all the CLBs of Virtex-4 FPGAs can be tested with just 24 BIST configurations. In 2008, [166] presented a similar approach to testing Virtex-1 FPGAs. It is worth to mention that both works did not consider routing resources test.

The off-line test that targets routing resources is [165](published in 2009), which provides an extensive BIST approach able to test routing resources on Virtex-4 FPGAs. A cross-coupled parity-based approach has been adopted. It can test the global routing resources of an LX Virtex-4 with 51 BIST configurations; according to other Virtex-4 subfamilies, the required BIST configurations can be up to 81.

3.4.3 On-line application-independent testing

Permanent faults caused by TIDs have not yet been extensively addressed by testing techniques. In the last years, the ongoing shrinking of the feature size of the CMOS technology made SEUs the predominant radiation effect in electronic devices. Thus, researchers focused much more on the detection of SEU effects than on TIDs. Nevertheless, the significant voltage scaling and the dramatic increase in the number of transistors make TID again significant in modern space electronics [88].

It is worth noting that manufacturing defects and radiation-induced permanent faults have similar effects from a functional point of view; in both cases, open and short faults may be observed during the functioning of the system. Therefore, many ideas belonging to application-independent testing may be borrowed and reused by approaches aiming at detecting permanent effects to radiations.

The Roving STARs

Introduced in 1999, the *Roving STARs* [1] was an integrated approach for on-line testing, diagnosis, and fault tolerance for FPGAs. The flow partitioned the FPGA in reconfigurable areas; utilizing DPR, the test placed a testing block in each of the reconfigurable regions. During the test of a particular region, the rest of the system was available for other operations. Furthermore, this kind of approach allowed testing both routing and logic resources. *Roving STARs* was applied to the ORCA FPGAs from Lattice Semiconductor [1]. Many of Roving STARs concepts has been utilized in the works presented in Section 3.4.2.

The OLT(RE)^2 approach is based on many concepts of the Roving STARs as well. It utilizes DPR to permit a fine-grain test of the FPGA, allowing the rest of the system to still operating. Despite Roving STARs, OLT(RE)^2 supports modern Xilinx FPGAs, facing with further architectural constraints and a heterogeneous architecture (differently from the old ORCA FPGAs).

OTERA

The only work addressing the problem of detecting faults induced in modern FPGA devices by the long-term exposure to radiation is *Online Test Strategies for Reliable Reconfigurable Architectures (OTERA)* [6]. This work focuses on the architecture used for testing the reconfigurable areas inside an FPGA device and on the scheduling of the test activities; in the work presentation, just a Virtex-5 is considered as a target device. Little information about how the test circuits and associated test patterns are generated is given. Moreover, the presented test architecture addresses any possible faults in those resources of the FPGA device actually used by the implemented design, leaving out the remaining resources.

Table 3.4: Testing approaches comparison.

	Year	Testing Method	Testing		Target FPGA
			Logic	Routing	
Dhingra [33]	2006	off-line	✔	✘	Virtex-4
Zhang2008 [166]	2008	off-line	✔	✘	Virtex-1
Yao [165]	2009	off-line	✘	✔	Virtex-4
Roving Star [1]	1999	on-line	✔	✔	ORCA
OTERA [6]	2012	on-line	✔	✘	Virtex-5
OLT(RE)^2	2015	on-line	✘	✔	V4, V5, V6, S6

On the contrary, OLT(RE)^2 focuses on the FPGAs routing resources, which represent up to 90% of the whole chip area in modern FPGAs [10].

3.4.4 Motivation

This section has provided all the motivation for the OLT(RE)^2 flow. Table 3.4 summarizes all the presented testing approaches. The Roving STARs provides a test that has the same final goal of OLT(RE)^2: an on-line test, utilizing DPR, which can detect logic and routing faults. Unfortunately, Roving STARs targets just ORCA FPGAs.

Considering Roving STARs as a milestone for on-line FPGA testing, many researchers tried to adapt the same concept to the modern and more complex Xilinx FPGAs architectures. They had to face to specific architectural constraint and with the increasing number of different devices produced. For these reasons, researchers decided then to spit the problem of testing FPGAs resources.

Hence, the works presented in [33; 166], they adapted Roving STARs approach to test Virtex FPGAs (Virtex-4 and Virtex-1 respectively). Nevertheless, they do not provide an on-line test using DPR. They rely on a certain amount of bitstreams that can be configured as requested; therefore, during a test the whole chip is utilized.

[165] is the only work that targets testing of routing resources with an application-independent approach. Unfortunately, it does not allow to have an on-line test; additionally, just the Virtex-4 family is considered. The only work that considers an application-independent on-line test is OTERA; its limit is the fact that just local resources are tested, and it targets just Virtex-5 FPGAs. Differently from these approaches, OLT(RE)^2 provides an application-independent on-line test for routing resources, which is unrelated to a specific Virtex family architecture.

3.5 Summary

This chapter has provided details about the related works of this thesis. In particular, it presented the current DPR tools that are utilized in modern Xilinx FPGAs, providing a comparison with the presented DXF database and the INDRA 2.0 flow.

Moreover, it presented how FPGAs, and DPR in particular, are becoming more and more utilized in space applications. Prototyping platforms, as well as real space applications that embed the DPR, are presented comparing them with the novel DRPM platform.

Finally, the chapter has motivated the need for a new flow, which investigates the permanent faults that can occur on FPGA due to radiation OLT(RE)^2. Current testing methods were analyzed, and their limits were discussed.

4 Dynamically Reconfigurable Processing Module

As presented in Section 3.3, reconfigurable hardware has gained hey interest in the domain of space applications. Nowadays, FPGA architectures have high computational power and thanks to their ability to be reconfigured at run-time, they became interesting candidates for payload processing in space applications.

DPR of FPGAs enables maximum flexibility and can be utilized for performance increase, improve energy efficiency, and enhance fault tolerance. To prove the effectiveness of this novel approach for satellite payload processing a prototyping environment has been developed, which combines dynamically reconfigurable FPGAs with different avionic interfaces such as CAN, SpaceWire, MIL-STD-1553B, and SpaceFibre.

The presented DRPM is an ESA funded TRP project (22424/ 09/ NL/ LvH) [37] and has been developed by the *Cognitronics and Sensor Systems* at *Bielefeld University* in close cooperation with different partners [180]. DRPM consists in a high-performance embedded system characterized by heterogeneous general-purpose processors and dedicated hardware accelerators working together. Communication among these processors and interfaces is one of the main aspects of this platform.

Therefore, this thesis introduces a novel communication interface for heterogeneous embedded multiprocessor systems (Heterogeneous Multi Processor Communication Interface (HMPCI)). This interface is intended to be used for data-flow synchronization among the processing elements of the system; it has been designed with the aim of finding a trade-off between performance, flexibility, and usability.

HMPCI has been developed for managing the communication of the DRPM. However, it is designed to be adopted in different heterogeneous multi-processors systems.

4.1 System Architecture

The architecture of the DRPM platform is based on the modular rapid prototyping system RAPTOR-X64 (presented in Section 4.1.1): up to six different modules can be connected to it. The DRPM platform is composed of two different kinds of modules: a communication module (called DB-SPACE, presented in Section 4.1.2) and

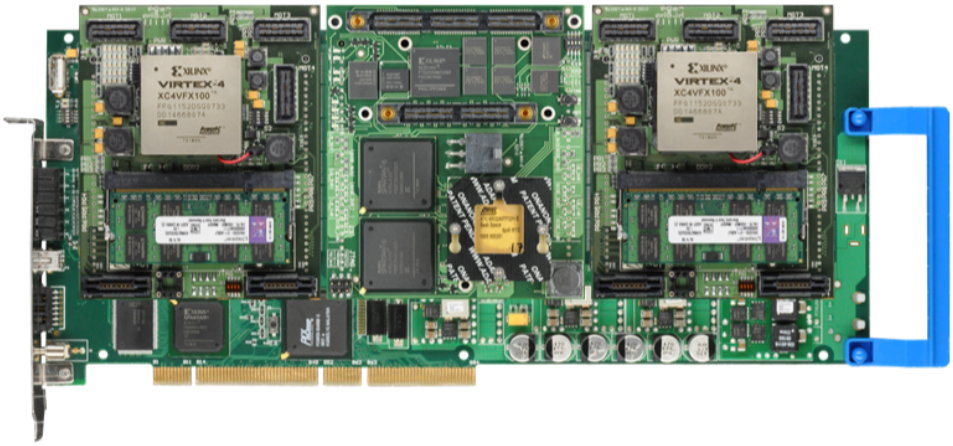


Figure 4.1: RAPTOR-X64 baseboard with one DB-SPACE and two DB-V4 modules placed [180].

a processing module (called DB-V4, presented in Section 4.1.3). All the modules implement their own processor subsystem. Therefore, every module can operate autonomously: this allows a high flexibility and scalability of the platform.

All modules are connected to the LocalBus of the RAPTOR-X64, providing extensive debugging and monitoring capabilities. The Point to point connections of adjacent modules enables high- speed parallel communication between the different modules.

The communication module is designed explicitly for the DRPM demonstrator and consists of two parts: the DB-SPACE daughterboard that can be connected as a module of the RAPTOR-X64 and the external DB-SPACE Frontpanel that provides easy access to the interfaces of the DB-SPACE. The processing modules of the system are the so-called DB-V4 daughterboards; they are equipped with a Xilinx Virtex-4 FX100 FPGA and a DDR2-Modules with up to 4 GByte.

In the common configuration of the DRPM one communication module and two processing modules are placed. The DB-SPACE communication module embeds a SpaceWire-RTC processor, which is in charged to control the overall DRPM systems. Figure 4.1 shows the RAPTOR-X64 with one DB-SPACE daughterboard in the middle and two DB-V4 daughterboards surrounding.

Local software routines are implemented to execute a specific operation within a module. However, if a placed processor in certain module needs to execute an operation on another module, a special remote procedure called approach is utilized.

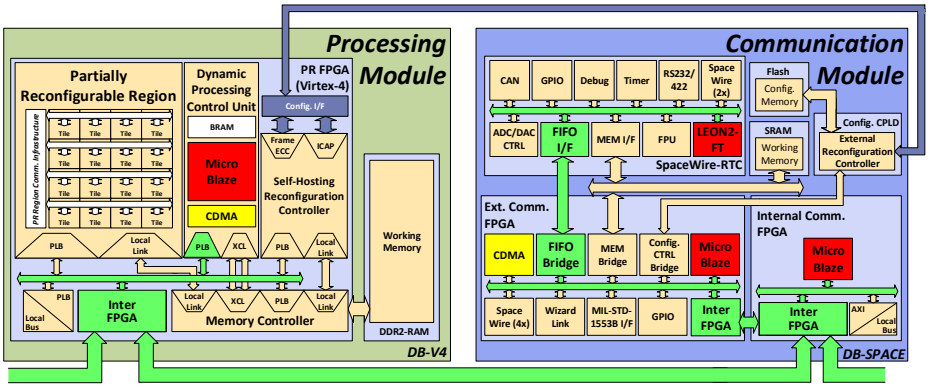


Figure 4.2: Block diagram of the DRPM, highlight the structure of one processing module and one communication module [180].

4.1.1 RAPTOR-X64

The rapid prototyping system RAPTOR-X64, the successor of RAPTOR2000 [59], integrates all key components to realize circuit and system designs with a complexity of up to 200 million transistors [16]. Together with rapid prototyping, the system can be used to accelerate computationally intensive applications and to perform DPR of Xilinx FPGAs. Therefore, this board has been chosen as a base board of the DRPM system.

RAPTOR-X64 is designed as a modular rapid-prototyping system: the base system offers communication and management facilities, which are used by a variety of extension modules, realizing application-specific functionalities. For hardware emulation, FPGA modules are equipped with Xilinx FPGAs and dedicated memories.

The prototype of complex SoCs is achieved by additional modules proving different communication interfaces, e.g., (Ethernet, USB, FireWire) as well as analog and digital I/O. The Localbus and the broadcast bus, both embedded in the baseboard architecture, add up to a powerful communication infrastructure that guarantees high-speed communication with the host system and between individual modules. Furthermore, direct links between neighboring modules can be used to exchange data with a bandwidth of more than 20 Gbit/s.

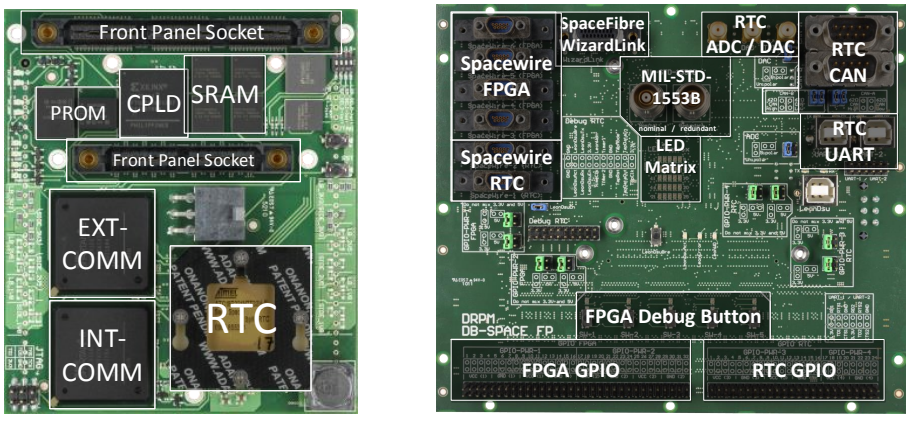
For communication with the host system (HostPC), either a PCI-X interface or a USB-2.0 interface can be used. Both interfaces are directly connected to the Localbus, thus creating a closely coupled, high-speed PCI-X-based communication, or a loosely coupled, USB-based communication.

Configuration and application data can either be supplied directly from the host system or stored on a compact flash card. In addition to these features, RAPTOR-X64 offers several diagnostic functions: besides monitoring of the digital system environment (e.g., status of the communication system), relevant environmental information like voltages and temperatures.

4.1.2 DB-SPACE

The daughterboard DB-SPACE has been specially designed for space related target applications; it is the central unit of the DRPM system. Its physical view is presented in Figure 4.3a. It hosts a *SpaceWire Remote Terminal Controller (SPWRTC)* [3], several interfaces, and a configuration device to allow controlling reconfiguration of other FPGA-based daughterboards (i.e., the DB-V4 presented in Section 4.1.3). Moreover, the board is based on two Spartan-6 FPGAs: the so-called *EXT-COMM* (XC6SLX150) and *INT-COMM* FPGAs (XC6SLX100). A block diagram of the DB-SPACE is depicted in Figure 4.2.

SPWRTC is based on a PR-FPGAs processor and several interfaces (ADC/-DAC, CAN, RS422, SpaceWire, General Purpose Input Output (GPIO), and debug interfaces). The EXT-COMM FPGA implements additional interfaces, such as WizardLink, SpaceWire, and GPIOs. The configuration controller is implemented on a Complex Programmable Logic Device (CPLD), providing basic reconfiguration features, such as DPR and blind scrubbing. Interconnection of the DB-SPACE components is realized via the INT-COMM FPGA.



(a) DB-SPACE daughterboard.

(b) DB-SPACE front panel.

Figure 4.3: DB-SPACE communication module components [180].

The DB-SPACE Front Panel board is an extension board for DB-SPACE (presented in Figure 4.3b). The Front Panel offers physical interconnects for the interface components of DB-SPACE. Therefore, the DB-SPACE communication module supports the following standards: 6x SpaceWire, WizardLink/SpaceFibre, ADC/-DAC, CAN, Debug interfaces (i.e., JTAG, TRACE, and UART), Timer signals and GPIO pins. In the following, the two EXT-COMM and INT-COMM FPGAs are described in details.

EXT-COMM FPGA

The EXT-COMM FPGA is mainly used to connect all data source interfaces to the rest of the system. Consequently, it also handles the communication streams between the different communication and processing modules. The EXT-COMM FPGA can be considered as a reconfigurable extension of the SPWRTC.

The different IP-Cores of the EXT-COMM FPGA are connected using the Advanced Microcontroller Bus Architecture (AMBA) AXI4 communication infrastructure. This allows multiple parallel communication streams between different bus subscribers, as the AXI4 is implemented as a crossbar.

Two basic Direct Memory Access (DMA) mechanisms have been integrated into the system architecture. One is the *CDMA* engine, which supports DMA transfers for every AXI4-Slave in the system. Furthermore, two cores have a dedicated DMA by using the DMA engines. These DMA engines transform the data read from the AXI bus into an AXI stream, which can be utilized directly by the target device, or vice versa. This allows higher throughput compared to a CDMA solution, especially when combined with larger burst transactions.

A MicroBlaze processor has been integrated with the necessary debug environment. This processor is used for testing purposes and is not supposed to be utilized in the final system, as the EXT-COMM FPGA is controlled directly by the SPWRTC.

INT-COMM FPGA

The aim of the INT-COMM FPGA is to interconnect all FPGAs of the DRPM system using the inter-FPGA interface and to connect them to the host PC via Localbus bridge. Both interfaces are configured by a set of registers accessible through either the Localbus or the inter-FPGA connection.

Additionally, the INT-COMM FPGA provides the startup, clock, and reset management for the SPWRTC. To handle fast data streams to and from inter-FPGA, a central DMA core is integrated. A MicroBlaze IP Core is utilized for debugging purposes.

4.1.3 DB-V4

The DB-V4 is an FPGA-based module for the RAPTOR-X64 board, which hosts a Xilinx Virtex-4 FX100 FPGA and 4 GByte DDR2 RAM (as depicted in Figure 4.1, where two DB-V4 are connected to the RAPTOR-X64 base board).

The FPGA includes two embedded PowerPC processors and 20 serial high-speed transceivers, each capable of transceiving 6.5 Gbit/s in full duplex. Utilizing these transceivers, four copper-based data links with a throughput of up to 32.5 GBit/s each are realized on the DB-V4 module [55].

PR-FPGA

The PR-FPGAs of the DRPM are the payload processing parts of the system. These FPGAs can be dynamically reconfigured allowing environment situation changes. Static processing components integrate a memory controller, connected to a DDR2-RAM, and all communication bridges (between internal and external cores) to the Processor Local Bus (PLB) main bus.

The bridges to the external cores are connected to the INT-COMM FPGA and the Localbus. The Localbus is the bus on the RAPTOR board that is connected to the PCI-bus, thus to the host PC. Internally, a bridge is needed for communication with the Reconfigurable Tile (PR Tile), which build the Partial Reconfigurable Region (PR Region). The communication infrastructure of the DPR system is generated with DHHarMa (presented in Chapter 6).

Apart from the memory management and ECC/EDAC functions of the multi-port memory controller, the Error Correcting Code (ECC) core has been extended to record more information on single and double bit errors. Self-hosting reconfiguration components comprise the Self Hosting Reconfiguration Controller (SHRC) and a dynamic processing control unit (DPCU) based on a MicroBlaze processor.

The Self Hosting Reconfiguration Controller (SHRC) is capable of providing DPR through the Internal Configuration Access Port (ICAP) with readback-scrubbing capability. The SHRC takes an important role in the on-line tests performed by OLT(RE)^2 (see Chapter 7).

4.1.4 Memory Resources

The memory structure of the DRPM communication (DB-SPACE) and processing modules (DB-V4) is presented in Table 4.1. The software of the DRPM (presented in Section 4.2) needs to face with limited available working memory. In the specific, the SPWRTC has just 8 MByte of working memory, and the MicroBlaze processor placed into the PR-FPGA has just 128 KByte.

The next section presents the overall software structure of the DRPM. On the one hand, it provides extensive functionalities, e.g., APIs and test routines; on

Table 4.1: Memory resources of the DRPM system.

Module	Resource	Size
DB-SPACE	SPWRTC working memory (async. SRAM)	8 MByte
	Configuration Memory (Flash)	128 MByte
	Data Buffer - EXT_COMM (BRAM)	2 x 64 KByte
	Data Buffer - INT_COMM (BRAM)	2 x 64 KByte
DB-V4	Data Buffer - PR_FPGA (BRAM)	2 x 64 KByte
	DPCU working memory - PR_FPGA (BRAM)	128 KByte
	PR_FPGA working memory (DDR2-SDRAM)	1 GByte

the other hand, its modular structure allows the running software to fit into the available working memory.

4.2 DRPM Software

As presented in Section 4.1, the DRPM system is a heterogeneous multi-processor platform. This section explains how APIs and drivers are utilized by the different processors.

The DRPM system is controlled by the PR-FPGAs processor, which is embedded into the SPWRTC of the DB-SPACE daughterboard.

More in general, the PR-FPGAs provides three main tasks:

- *Handle avionic interfaces and appropriate services* (e.g., PUS standard [28]).
- *Handle integrated source data interfaces*; the data need to be forwarded to the reconfigurable core.
- *Handle global components of Fault Tolerance and Error Correction Management Unit (FTECMU) and Resource and Reconfiguration Management Unit (RRMU)*, which respectively especially gathers statistics data and controls reconfiguration for several reconfigurable cores.

These tasks need to be executed concurrently in a time critical environment. Therefore, a real-time operating system is required. Apart from the SPWRTC, the Dynamic Processing Control Unit (DPCU) of the PR-FPGA hosts a softcore processor, which have to handle the following main local functionalities:

- Routing of source data interfaces to appropriate PR Modules.
- Handle the software routines of the local RRMU and the FTECMU.

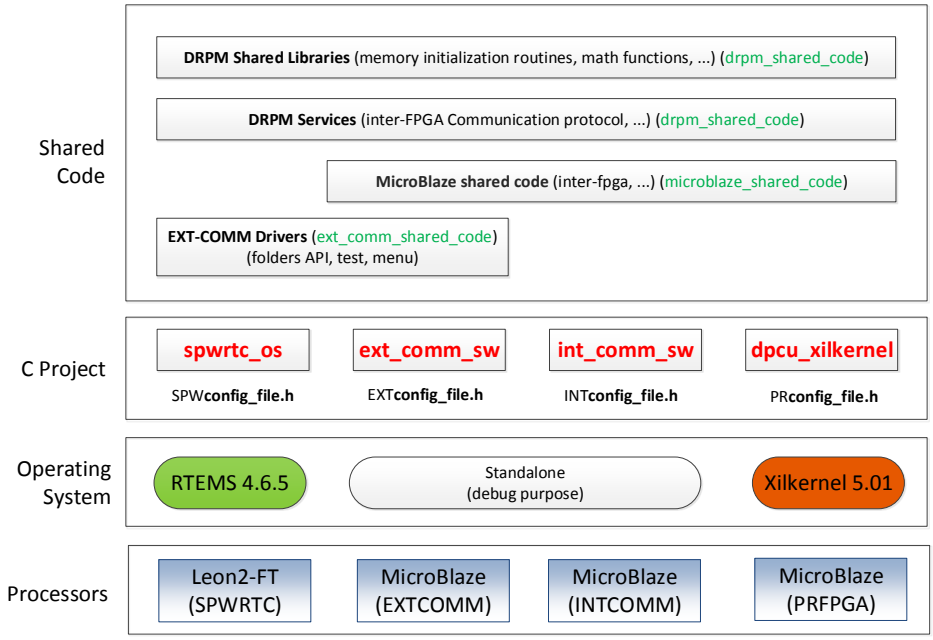


Figure 4.4: Software architecture of the DRPM [30].

- Provide services to make local RRMU and local FTECMU accessible by the master processor of the DRPM system: the PR-FPGAs. For this task, the HMPCI communication interface is used (see Section 4.3).

The DRPM platform relies on the Real-Time Operating System for Multiprocessor Systems (RTEMS) [93], which is a fully featured RTOS that supports a variety of open API and interface standards. In the specific, RTEMS 4.6.5 for Sparc V8 is operating on the SPWRTC; it provides functionalities like memory management and task scheduling for the PUS services [28].

4.2.1 Software Structure

As mentioned in Section 4.1, the DRPM is a heterogeneous multi-processor system, which embeds two different kinds of processors: the PR-FPGAs and the MicroBlaze.

The DRPM code structure is presented in Figure 4.4. Four different processors are utilized in the systems: PR-FPGAs and MicroBlazes on the INT-COMM, EXT-COMM, and PR-FPGA. Two different operating systems are running on

the DRPM. The mentioned RTEMS on the PR-FPGAs and Xilkernel 5.0 on the PR-FPGAs MicroBlazes. The MicroBlazes of the INT-COMM and EXT-COMM FPGAs are used just for debugging purposes, therefore, no operating system is utilized.

For every target processor, a C project has been created (using ECLIPSE IDE). It is possible to configure the project utilizing specific preprocessor directives, which are located in the files named *"*config_file.h"*. These allow the final executable to fit on the device memory [30].

The processors of the DRPM can work autonomously, therefore, they can run the operations and routines. For example, the interface APIs of the EXT-COMM device can be used by MicroBlaze and PR-FPGAs processor. Therefore, the code is organized in shared libraries, which are included in every C project. The shared code is divided in EXT-COMM drivers, MicroBlaze shared code, DRPM services and DRPM shared libraries.

4.3 Heterogeneous Multi Processor Communication Interface (HMPCI)

Modern embedded applications are demanding of computational power. Therefore, the number of processors deployed in such applications is becoming larger and larger [180; 49]. Such embedded multiprocessor systems can be characterized by heterogeneity of the employed microprocessors and microcontrollers (e.g., LEON, Nios II or MicroBlaze), which can cooperate with DSP- or FPGA-based dedicated hardware accelerators [23].

Besides the hardware accelerators, also many modern I/O systems are designed to support high data rates and require a powerful interconnect within the system. In such data-flow-centric computing systems, the processors distributed across the system are mainly used for control tasks, ensuring the data is transferred to the right component at the right time, while the hardware accelerators are used for the actual processing of the data streams.

[57] discusses the hardware design aspects of heterogeneous embedded multiprocessor systems during the last years, while the problem of developing efficient software for this kind of platforms has not yet been extensively faced. One of the crucial aspects of the design of a heterogeneous embedded multiprocessor systems is the communication among the various processors [99].

This section introduces a novel communication interface for heterogeneous embedded multiprocessor systems (HMPCI). It is inspired by the Remote Procedure Call (RPC) paradigm and allows any processor in the system to invoke functionalities (passing the appropriate parameters) implemented by other processors and receive back the results. In other words, it is intended to be used to synchronize

the data-flow between hardware accelerators and high-speed I/O interfaces.

The interface has been designed with the aim of finding a trade-off between performance, flexibility, and usability. To assess its performance and usability, the interface has been implemented and applied to a heterogeneous multiprocessor architecture used for payload processing: the DRPM. The presented section has been extracted from the previous work [174].

4.3.1 Related Works

Works addressing the communication among the processors of a heterogeneous embedded multiprocessor system can be found in the literature. These works are based on two different paradigms: *Message Passing Interface (MPI)* and *the RPC*.

The works MMPI [42], MSG Libraries [52], SoC-MPI [73] and MCAPI [75] focus on the MPI paradigm; on the contrary, the works presented in [47; 72] are RPC-based communication interfaces. Some of these works are focused on the hardware infrastructure needed to allow inter-processor communication [62], while other are more focused on the software interfaces and middlewares [42; 46; 79].

The works similar to the presented HMPCI interface are those presented in [47; 72; 81]; in both works, RPC-like communication interfaces are proposed. Nevertheless, the interface proposed in [47] is specifically focused on dual-core processors. Moreover, the interface proposed in [72] allows only one process in the software architecture to invoke functionalities, while all the others processes are meant to be providers of functionalities. [81] focuses on grid computing systems, which are different from embedded systems. Therefore, the presented communication interface is the first that uses the RPC paradigm in heterogeneous embedded multiprocessor systems.

Remote Procedure Call (RPC) and Message Passing Interface (MPI)

RPC paradigm has been chosen for the presented HMPCI because it is more easy-to-use than MPI, as it has been discussed in [101]. However, RPC requires more effort by the designers and programmers of the communication interface.

Thus, RPC enables to design a communication interface that could be easily employed by the final application programmer. Moreover, with the RPC paradigm, users call remote functions like local functions [101]. On the contrary, MPI is easy to implement from the interface designer point of view, but it requires more effort for application developers to manage function invocations and marshalling/unmarshalling of the data.

4.3.2 Inter-Processor Communication Interface

The proposed HMPCI communication interface is intended to allow applications executed on one particular processor in the system to easily invoke functions (and pass the corresponding parameters) to another application executed on a different processor of the system. The interface allows exchanging control information and data among any application running on any of the heterogeneous processors of a multi-processor system, e.g., applications running on a LEON processor, as well as on a MicroBlaze or a Nios microcontroller.

Figure 4.5 represents the overall software structure of a processor running the proposed communication interface. The top level is occupied by the applications defined by the user and executed by the processor. The core of the proposed interface is represented by the *physical layer-independent API (PHY-IND)*, which implements the higher level of the communication interface. In particular, PHY-IND is composed of a set of functions that are used by the applications to communicate among each other.

PHY-IND relies on lower-level functions that allow the high-level communication functions to access the physical communication infrastructure. These lower-level functions may be provided by an operating system or by a specific designed *physical layer-dependent API (PHY-DEP)*, in case the processor does not run an operating system or the operating system does not support the particular communication infrastructure. Finally, the lowest level of the communication interface is represented by the communication infrastructure itself, i.e., the *physical layer (PHY)*.

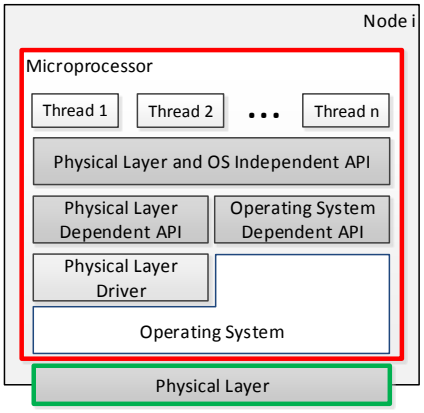


Figure 4.5: HMPCI structure on a generic node [174].

4.3.3 HMPCI Interactions

The core of the proposed communication interface is composed by a set of functions that implement the RPC-based communication mechanism. These functions are called by the applications threads executed on the processors. *Client nodes* are defined as those nodes that invoke functionalities implemented by other nodes. Similarly, *server nodes* are defined as those nodes that provide functionalities that can be invoked by client nodes.

Client nodes send parameters to server nodes and server nodes send results to client nodes according to the invoked functionality. It is worth noting that a node that acts as a server in a given interaction can serve as a client in a different interaction. Three interaction modes (represented in Figure 4.6) have been defined: *just-once*, *periodic* and *jumbo*.

Just-once

Just-once procedure calls (represented in Figure 4.6a) comprise functions that are invoked by the client and are executed by the server only once. These interactions can be both synchronous and asynchronous.

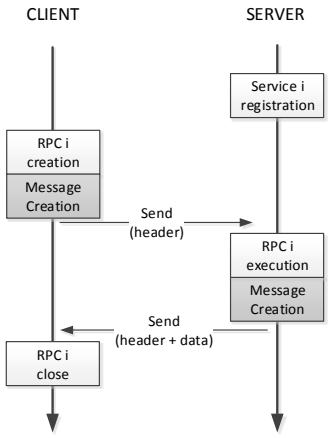
An example of just-once interaction is a client that asks to a server running on a hardware accelerator to calculate the Fast Fourier Transform of a given set of samples. The client sends the samples as parameters of the function, the server calculates the transformation and sends back the result.

About the DRPM platform, one example can be a request of performs DPR on the PR-FPGA device. The client (PR-FPGAs) sends the bitstream to be configured to the server (MicroBlaze), which performs the reconfiguration and sends back an acknowledgment to the client.

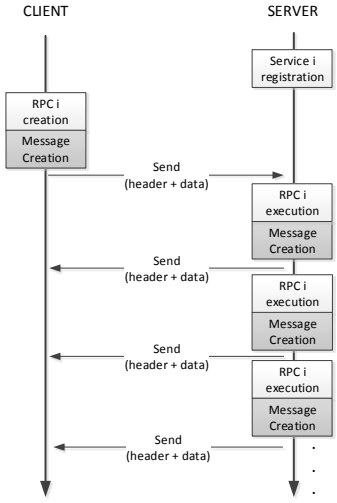
Periodic

Periodic procedure calls (represented in Figure 4.6b) involve those functions that are invoked by the client, are executed by the server and have results to be sent periodically. These interactions are always asynchronous.

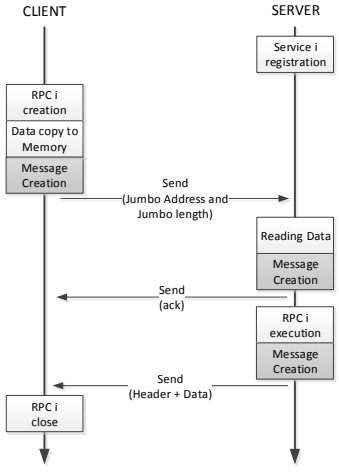
An example of periodic interaction could be a client that asks to a server running on an SRAM-based FPGA to maintain statistics about the occurrence of single event upsets in the configuration memory of the device, and to report to the client the number of occurred faults every N hours. The client sends N to the server, which starts monitoring the configuration memory, sending periodic reports to the client.



(a) Just-once Interaction.



(b) Periodic Interaction.



(c) Jumbo Interaction.

Figure 4.6: Examples of just-once, periodic and jumbo interactions.

Jumbo

Jumbo procedure calls (represented in Figure 4.6c) involve those functions whose execution needs (or produces) a large amount of data. In this interaction type, the function parameters or results (*jumbo data*) are copied into the destination memory space exploiting DMA capabilities. After this, two messages, one containing the memory address (*jumbo address*) where data have been copied and the other containing the length (*jumbo length*) of the data, are sent to the destination node. After reading the data, the destination node sends an acknowledgment, which informs the source that the memory space has been freed.

Exploiting the jumbo mechanism, a node can send a large data block fast exploiting DMA. It is worth noting that if the system architecture does not provide DMA capabilities, the jumbo mechanism of the proposed interface cannot be exploited.

An example of jumbo interaction could be a client that asks a server running on a hardware accelerator to perform a motion estimation on a large set of images. The client copies the set of frames to be analyzed in the server's memory space through DMA and sends the memory address and the data length to the server. The server reads the data and sends back an acknowledgment to inform the client, and finally, it performs its computation and sends back the result of the estimation.

4.3.4 Inter-Processor Communication Protocol Details

The proposed HMPCI implements a point-to-point half-duplex communication. The communication is structured into *flows*, where client nodes send *request messages* to server nodes to invoke functionalities and send parameters, and server nodes send *response messages* to client nodes to send back the results.

Each flow is composed of a request message and one or more response messages. Each flow is identified by three data fields: the ID of the client, the ID of the server nodes, and the flow ID. In this way, the same flow ID can be utilized by a client to communicate with different servers.

Each message (either request or response) is composed of *packets*. The structure of a packet is depicted in Figure 4.7. Every message consists of one header packet and zero, one or more data packets (messages carrying no data packets are acknowledged packets); each packet carries 32 bits.

The header packet is used to specify the functionality that has to be executed (in request messages) or that has been executed (in response messages) and the number of incoming data packets. Data packets can be either of *normal* or *jumbo* type. Normal packets carry parameters for a function invocation or results from an executed function. Jumbo packets are always sent in pairs and carry the jumbo address and the jumbo length.

In the following, it is presented a description of the different data fields of a packet:

0 - 3	4 - 7	8 - 11	12	13	14	16 - 23	24 - 31
SRC	DST	Sequence	PR	RR	PT (00)	OP-ID	Data length
SRC	DST	Sequence	PR	RR	PT (01)	Data	
SRC	DST	Sequence	PR	RR	PT (10)	Jumbo Address	
SRC	DST	Sequence	PR	RR	PT (11)	Jumbo length	

Figure 4.7: Structure of the packets transmitted using the proposed communication interface [174].

- *SRC* (4 bits): ID of the packet source node.
- *DST* (4 bits): ID of the packet destination node.
- *Sequence* (4 bits): sequence number of the flow to which the message belongs.
- *PR Flag* (1 bit): high (1) or low (0) priority flag.
- *R/R Flag* (1 bit): specifies whether the packet belongs to a request message (0) or to a response message (1).
- *PT Flag* (2 bit): specifies whether the packet is a header (00), a normal data (01), a jumbo address (10) or a jumbo length (11).

The remaining 16 bits carry, depending on the type of packet (PT Flag):

- *Header (PT = 00)*: the operation ID (OP-ID) and the number of incoming data packets. The OP-ID indicates the function to be executed.
- *Normal data (PT = 01)*: a function parameter or result.
- *Jumbo address (PT = 10)*: the address of the destination memory space in which data have been stored.
- *Jumbo length (PT = 11)*: the number of bytes copied into the destination memory space.

In conclusion, the HMPCI can work with a maximum of 16 nodes (4 bits dedicated for source and destination nodes). Every Client-Server pair can manage 16 operations at a time (i.e., 16 open flows); this limitation is given by the 4 bits of the sequence number. 8 bits are dedicated for the OP-ID, therefore, within the overall multi-processor system up to 256 operations can be defined. Finally, the priority level can be set for specific tasks (i.e., high or low priority).

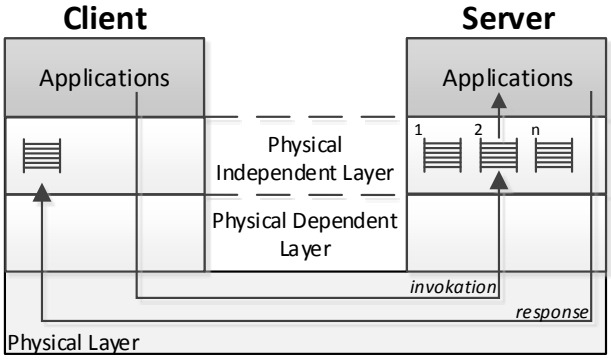


Figure 4.8: Overall functioning of the proposed communication interface [174].

4.3.5 Using the Inter-Processor Communication Interface

Whenever a remote procedure call is required, the client has to perform the following operations:

1. Create a message queue associated with the flow that is going to be opened.
2. Create a message.
3. Initialize the header of the message.
4. Initialize the data packets of the message with the data that have to be sent.
5. Send the message.
6. When the RPC ends, deallocate the queue associated with the flow.

It is worth noting that all the operations related to the creation, initialization, and transmission of data (steps 1 to 5 in the description above) are carried out by the same function. Thus, executing a remote invocation is easy from the application developer point of view.

Any application that wants to operate as a server has to create a queue for any of the functionalities that it wants to make available. These queues will be used to store requests for the corresponding functionalities.

To reply to requests, after having executed the invoked functionality, a server has to perform the same procedure performed by the client for the request, apart from the first step (operations 2 to 5 in the description above). Like for clients, also for servers, the operations related to answering to a remote function invocation are all carried out by the same function.

The overall functionality of the proposed communication interface, from an architectural point of view, is described in Figure 4.8.

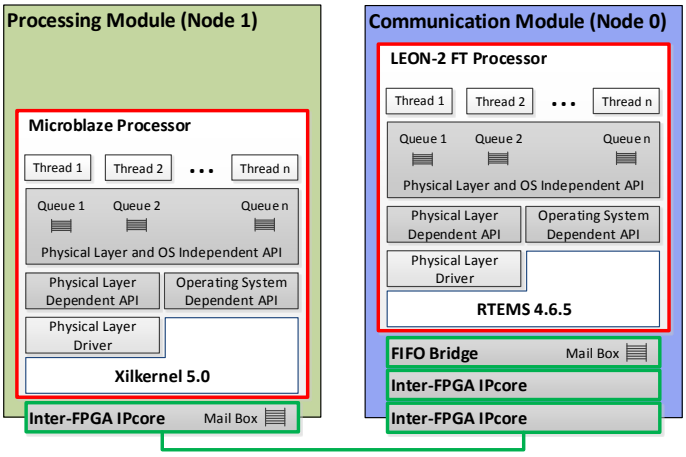


Figure 4.9: HMPCI implementation on the DRPM.

4.3.6 HMPCI on the DRPM

The proposed interface has been implementing for targeting the DRPM platform. As explained in Section 4.1, DRPM is a heterogeneous embedded multiprocessor system composed of multiple *communication modules* and *processing modules*.

The communication modules rely on SpaceWire-RTC AT7913E, based on a PR-FPGAs CPU (50 MHz working frequency) running the RTEMS operating system [93], acting as the system controller. The processing modules are equipped with a Xilinx Virtex-4 FX100 FPGA with an embedded MicroBlaze processor (100 MHz working frequency) running the Xilkernel operating system [130]. As mention in Section 4.1.2, the MicroBlazes of the INT-COMM and EXT-COMM FPGA are utilized just for debugging purpose, therefore, they are not used in the considered HMPCI test scenario.

Considering, the DRPM as the target platform, Figure 4.9 shows how HMPCI has been implemented.

The Physical Layer and the Physical Layer-dependent API

The physical layer of the proposed communication interface relies on the inter FPGA communication module presented in [89]. Each node in the system is provided with an inter-FPGA communication module. The inter-FPGA is equipped with one *local* interface and up to three *external* interfaces.

The local interface is used to connect to processors or peripherals on the local module. The external interfaces are used to connect the inter-FPGA to other com-

munication modules and thus to allow a node to communicate with components on other nodes, like processors, accelerators or I/O systems.

A DMA unit available in each node is used for efficient data transfer between the nodes using the external interfaces. Function invocations and a small set of data are passed between the processors using the mechanisms described in the following sections, while a large bunch of data transfer can be handled in hardware through DMA.

Each interface of the inter-FPGA is identified by an *interfaceID*. Similarly, each processor in the system is identified by a *processorID*. Each inter-FPGA in the system stores a routing table containing `<interfaceID, nodeID>` bindings. This allows communication between any couple of nodes in the system.

In particular, for every possible destination node, each inter-FPGA stores in its routing table the interfaceID of the interface through which the packet has to bee forwarded. Moreover, since an inter-FPGA can receive (forward) packets from (to) either the local interface or the external interfaces, multi-hop communication is available.

The inter-FPGA IPCore exchanges data through a physical layer communication protocol. Packets exchanged between communication modules carry 32 bits of control information (the source and destination *nodeIDs* and the number of following words of data), 32 bits used to specify the address of the memory space of the destination node in which the data will be stored, and 1 up to 2^{10} 32-bit words of data. The inter-FPGA physical layer communication protocol uses a 4-bit addressing scheme, then up to 15 nodes can be connected in the system.

Since the inter-FPGA physical layer of the DRPM is the result of a custom design, neither the RTEMS nor the Xilkernel operating systems offered the functionalities needed to access it. Thus, a dedicated PHY-DEP API for the physical communication infrastructure of the DRPM has been developed. The PHY-DEP API provides functionalities to configure the routing tables of the communication modules, and, once these have been configured, to exploit the physical layer communication protocol to send and receive data.

Implementation details

One of the main targets of the HMPCI protocol is to have a limited footprint. Table 4.2a shows the code occupation for RTEMS running on PR-FPGAs and Xilkernel running on MicroBlaze. The results are provided in Bytes and Logical System Lines Of Code (LLOC). The table is partitioned, according to the HMPCI software structure, in PHY-IND (marked in green), and PHY-DEP (marked in red).

Moreover, the PHY-DEP software is partitioned into three subcategories: base protocol, client specific and server specific. Hence, further decrease the footprint of the code is possible. In the specific, a node could operate just as server or client.

Table 4.2: HMPCI footprint on the DRPM.

(a) PHY-DEP (red) and PHY-IND (green) footprint.

	MicroBlaze, Xilkernel		PR-FPGAs, RTEMS	
	Size [Byte]	LLOC	Size [Byte]	LLOC
Server Specific	368	42	224	42
Client Specific	592	64	656	64
Base Protocol	7,536	431	5,854	259
Phy Layer APIs	828	203	378	199
Total	**9,324**	**740**	**7,112**	**747**

(b) HMPCI Client and Server.

	MicroBlaze, Xilkernel Size [Byte]	PR-FPGAs, RTEMS Size [Byte]
Client Node	8,956	6,888
Server Node	8,732	6,456
Client + Server Node	**9,324**	**7,112**

740 code lines have been written for the MicroBlaze: 537 for the PHY-IND API and 203 for the PHY-DEP API. In the case of the PR-FPGAs, 365 LLOC for the PHY-IND API and 199 for the PHY-DEP API. Most of the developed code could be compiled for both RTEMS and Xilkernel. The only platform-dependent code is the one for the management of the semaphores and the queues associated with the remote function invocations.

The HMPCI has been compiled for RTEMS and Xilkernel with the maximum optimization level for memory occupation. Using this optimization level, the RTEMS implementation of HMPCI occupies 7,112 Byte; the PHY-IND API occupies 6,57 kByte: 224 Byte server-specific, 656 Byte client-specific, and 5,854 Byte of base protocol code.

The Xilkernel implementation occupies 9,324 Byte. More in detail, the PHY-DEP API occupies 828 Byte; the PHY-IND API occupies 8,496 kByte (368 Byte server-specific, 592 Byte client specific and 7536 Byte of base protocol code).

Table 4.2b highlight the footprint of the three use cases: *Client*, *Server*, *Client + Server*. The results show that compiling the interface according to a specific use, memory space can be saved.

4.3.7 Experiment Results

Bandwidth

Table 4.3 shows the maximum available bandwidth of the proposed interface. Depending on the length of the data transfer, the DMA units on the different nodes are used to transfer the data. DMA is used to transfer data directly from an interface to a hardware accelerator or between accelerators, while the processor is just controlling the data flow. In this test, the data are transferred from the EXT-COMM to the PR-FPGA. For data bigger than 12 Bytes, it has been observed that is convenient to utilize the DMA engine embedded in the EXT-COMM FPGA. For transfer lengths $\geq$ 1 MByte, a utilization of $\geq$ 90 % is achieved. For bigger transfers (8 MByte) 98 % are observed.

Latency Evaluation

The one-way latency of the HMPCI protocol is evaluated as well, i.e., the number of clock cycles between the starting of a remote function invocation and the beginning of the function execution. Table 4.4 reports the latency calculated for 0 to 12 Bytes of transmitted data. In detail, the table reports the clock cycles spent by the client to open the flow and create the message, the clock cycles needed to transmit the message and the clock cycles needed by the server to pop the message from the queue and unpack it.

Table 4.3: Bandwidth/Throughput results for the presented interface [174].

Length of data transfer	Time for transfer	Bandwidth	DMA used
4 Byte	138.64 μs	28.18 kByte/s	✘
12 Byte	308.5 μs	37.99 kByte/s	✘
16 Byte	308.81 μs	50.60 kByte/s	✔
64 Byte	308.93 μs	202.31 kByte/s	✔
256 Byte	309.41 μs	807.99 kByte/s	✔
1 kByte	311.33 μs	3,14 MByte/s	✔
4 kByte	319.01 μs	12.24 MByte/s	✔
16 kByte	349.73 μs	44.68 MByte/s	✔
1 MByte	2.93 ms	341.27 MByte/s	✔
8 MByte	21.28 ms	375.93 MByte/s	✔

Table 4.4: Latency results for the presented interface.

		0 Byte [#CLKs]	4 Byte [#CLKs]	8 Byte [#CLKs]	12 Byte [#CLKs]
Client	open flow	390	390	390	390
	create message	107	755	1251	1599
	msg. transmission	113	273	2074	1989
	total client	**610**	**1418**	**2074**	**1989**
Server	pop msg. from queue	1050	3145	5235	7361
	unpack message	774	2369	3899	5472
	total server	**1824**	**5514**	**9134**	**12833**
Total		**2434**	**6932**	**11208**	**15425**

4.3.8 Summary

In this section, a communication interface for heterogeneous embedded multiprocessor systems has been presented (HMPCI).

Table 4.5 reports a performance comparison between HMPCI and some other interfaces. The comparison has been carried out considering the memory footprint (code+data) and number of clock cycles needed to transmit 4 Bytes at the communication infrastructure level. HMPCI has a memory footprint comparable to the footprint of the interfaces in [42] and [73] but much smaller than the ones in [52] and [75]. From the latency point of view, apart from the interface in [75], all the other interfaces show a higher latency than HMPCI. In particular, while the interface in [73] has a still comparable latency, the ones in [52] and [42] have much

Table 4.5: Performance comparison [174].

Reference	Platform	Memory footprint [KByte]	One-way latency [#CLKs]
SoC-MPI [73]	MicroBlaze	11.5 to 16	281
MSG [52]	CELL BE	∼256	1200
MCAPI [75]	Nios/PC	28.8	32
MMPI [42]	custom MPSoC	11	1000
HMPCI	PR-FPGAs/ MicroBlaze	6.95 / 9.11	273

higher latency.

Since the implementation is inspired by the RPC paradigm, the proposed interface offers a higher ease of use to embedded application designers than message passing-based communication interfaces. Moreover, thanks to the integrated DMA mechanism, a high channel utilization is achieved.

4.4 DRPM Evaluation and Validation Environment

The software implementation of the DRPM system (presented in Section 4.2) allows the DRPM to implement space scenarios. In the test and verification phase of the DRPM development, dedicated test cases have been created to prove the hardware implementation of the system.

In addition, the platform comes with a complete tool that runs on the Host PC: the *DRPM GUI*. The GUI enables a complete setting of the DRPM and provides an easy-to-use interface for users.

4.4.1 Avionic Interfaces Testing

The system has been validated with specific avionic devices. In the specific, the utilized devices are the *CAN-AC2-PCI Board*, the *SpaceWire-USB Brick* and the *STAR Fire* (Figure 4.10).

CAN

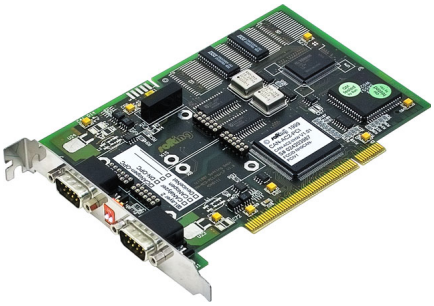
The CAN Application Controller CAN-AC2-PCI (Figure 4.10a) allows easy interfacing of PC applications to CAN-based networks. CAN-AC2-PCI offers access to two independent, optoisolated CAN networks.

Test scenarios have been generated utilizing the tool *CANalyzer 4.0* [119], which is a universal software analysis tool for Electronic Control Unit (ECU) networks and distributed systems. CANalyzer allows observing and analyze data traffic in CAN devices.

SpaceBrick

The SpaceWire Brick [106] (Figure 4.10b) allows the connection of a host PC to a SpaceWire device or network. It connects to the PC through a USB port, and it has two SpaceWire ports.

The unit contains a non-blocking routing switch using the same technology as the STAR-Dundee Router IP Core, which allows the user to route SpaceWire packets through any of the three ports (2 x SpaceWire, 1 x USB).



(a) CAN-AC2-PCI Board [103].

(b) SpaceWire Brick [107].

(c) StarFire [107].

Figure 4.10: Avionics devices utilized with the DRPM.

This device is utilized for testing the SpaceWire interfaces of the DRPM, connected to the SPWRTC and the EXT-COMM FPGAs. Moreover, technical APIs are also provided with the SpaceWire Brick, which has allowed integrating the setting and use of this device into DRPM GUI.

STAR Fire

STAR Fire [107] (Figure 4.10c) supports the evaluation and early adoption of SpaceFibre technology with a comprehensive test and development platform. STAR Fire can operate as a bridge between SpaceWire and SpaceFibre.

The SpaceWire APIs provided with the SpaceBrick are compatible with the Space Brick device. Therefore, the STAR Fire functionalities are also integrated into the DRPM GUI. Moreover, this device is mainly utilized in the Video Demo Scenario presented in Section 4.4.2; it allows verifying the utilization of the SpaceFibre and SpaceWire interface into a video stream scenario.

4.4.2 DRPM GUI

The DRPM GUI is a complete suite that allows configuring, validating and creating use case scenarios on the DRPM system. It has been designed utilizing QT Creator 2.6.2 and QT libraries 4.8.4 [116]. This thesis focuses on the SpaceWire/SpaceFibre video stream functionalities of the DRPM GUI.

The DRPM GUI embeds several functionalities such as:

- *FPGAs module configuration*: it is possible to load and configure a bitstream for every module placed on the RAPTOR-X64.
- *PROM configuration*: load the bitstreams into the FLASH memory placed on the DB-SPACE and set the PROM to have the automatic configuration of specific bitstream after a power-on of the system [171].
- *SpaceBrick and StarFire configuration*: as mentioned in Section 4.4.1, it is possible to configure and utilize these devices to validate the SpaceWire/SpaceFibre interface and to create specific test scenarios.
- *Serial Port debugging*: debugs the FPGAs of the systems thanks to serial connections to the FPGAs. Moreover, it enables to set specific registers of the system.
- *Scheduling tests on the DRPM*: it allows creating data-flow cases into the DRPM, which gives the possibility to investigate the best scheduling strategy to adopt [173].

Video Demo Scenario

The DRPM provides a complete platform to investigate the utilization of DPR in space applications. Moreover, it consists of a multi-FPGA system that embeds different kinds of processors. This thesis presents a test case scenario, which shows the capabilities of the system: the DRPM *video demo*.

The video demo is part of the DRPM GUI, and it consists of video stream frames into the DRPM, filter them and reading back the filtered frames. It is possible to choose different frame sources: *Local Webcam*, *Local Video* or *Local Picture*. Moreover, the data can be streamed into the DRPM utilizing the PCI interface, the SpaceWire interface (through either SpaceBrick or Star Fire) or the SpaceFibre interface (through the Star Fire).

For example, the user can set a webcam as a source, then send the data using the SpaceWire interface, execute a specific image filtering (which is placed on the PR-FPGA as a PR Module) and read back the results utilizing the SpaceFibre interface. In this kind of scenario, the data go first through the USB-interface into the StarFire device, then into the EXT-COMM FPGA through SpaceWire link; once

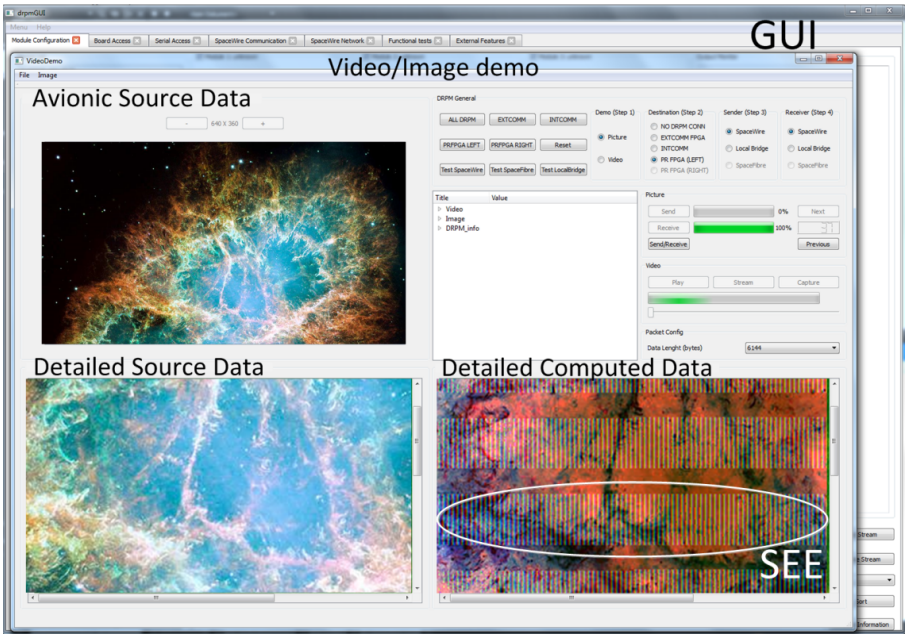


Figure 4.11: GUI of the demonstrator [183].

the data are received by the DRPM, they are routed through the INT-COMM to PR-FPGA (thanks to the inter-FPGA core presented in Section 4.3.6) and finally, the data goes to the specific filtering. On the one hand, this kind of scenario gives a validation of the functionalities of the DRPM; On the other hand, it provides an extensive video demo that can show the capability of the system.

In addition, the video demo shows the self-healing capability of the system, in the case of a SEE occurs: it is possible to insert a bit-flip into the configuration memory of the PR-FPGA and verify how the system corrects the error before it propagates into the system. Figure 4.11 shows this kind of verification: a picture is selected as *Avionic Source Data*, then it is sent into an inverter filter. The window *Detailed Computed Data* shows the computed picture; the picture has some erroneous pixel, in fact, the system has been set to insert regularly faults into the PR-FPGA system. The picture shows, how the system (thanks to the SHRC) can correct the erroneous bit, replacing the correct behavior of the system.

4.5 Summary

The DRPM, a novel scalable prototyping environment for space application, has been presented. The system allows applying DPR of FPGAs in space scenarios. The system combined dynamically reconfigurable Xilinx FPGAs, a rad-hard SoC (SPWRTC) and emerging avionic interfaces (e.g., SpaceFibre [182]) additionally to established ones (e.g., SpaceWire, MIL or CAN). The architecture of the DRPM has been presented, highlighting its novel architecture and modularity. More results and benchmarks of the DRPM platform can be found in [177; 179; 181].

The different processors of the systems (i.e., PR-FPGAs and MicroBlaze) are interacting utilizing an inter-processor communication protocol: the Heterogeneous Multi Processor Communication Interface (HMPCI). This allows having a remote procedure call based interface, which provides minimal footprint in the system and high flexibility in term of number of processors and operations that can be instantiated. Furthermore, Heterogeneous Multi Processor Communication Interface (HMPCI) can be adapted to different heterogeneous embedded multiprocessor systems.

The modular RAPTOR-X64 FPGA prototyping board has been used to create a highly scalable platform, which is then connected to a commercial PC, utilizing a PCI-X interface. A custom GUI enables to configure and debug the overall DRPM (e.g., download bitstream, processor console debugging, PROM configuration).

A special test case scenario has been created, which verify high-bandwidth of the system, integration with avionics devices (SpaceWire Brick and StarFire), DPR and self-healing [177; 183].

5 INDRA 2.0

This chapter presents the *INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0)* flow, which targets the creation of DPR systems on Xilinx FPGAs. These devices can be dynamically reconfigured, changing the functionality of a portion of the FPGA (as discussed in Section 2.2). However, the Xilinx tools do not allow exploiting fully DPR on their devices. Section 3.2 discusses how several research tools overcome the limitations of the Xilinx tools; INDRA 2.0 is one of them.

The flow enables the user to create advanced DPR systems. It starts with the HDL definition of a design and generates the bitstreams that are configured on the device. Moreover, INDRA 2.0 is integrated with the standards Xilinx tools, allowing utilizing them in the intermediate steps of the flow.

This thesis provided two main contributions on the INDRA 2.0 flow. It provides a *PW database* of the Xilinx routing resources, which allow the integration of INDRA 2.0 with the Xilinx FPGA-Edline tool. Then, the *PSRerouter* has been generated, which enables the user to reroute specific nets of a design, solving some limitation of the Xilinx router.

Section 5.2 presents the novel tool DHHarMa. They are introduced the *Datastructure for Xilinx FPGAs (DXF)*, the integration with the Xilinx tools, and the DHHarMa's parts (i.e., *Homogeneous Packer*, *Homogeneous Placer*, and *Homogeneous Router*). The Homogeneous Router is one of the key parts of this thesis, and it is discussed in detail in Chapter 6.

Section 5.3 presents the *PSRerouter*. This tool allows rerouting a P&R design without the need to convert it with the Xilinx XDL tool. In the specific, it is discussed how a dedicated database has been created to support this tool (the so-called *PW database*, which is integrated to the DXF database). Finally, it is presented how static nets that cross dynamic area of the FPGA can be rerouted.

5.1 Flow Description

INDRA 2.0 is a novel flow that generates DPR systems. It supports a wide range of Xilinx FPGAs (Virtex-4, Virtex-5, Virtex-6, Spartan-6 and 7 Series) and it is integrated wit the Xilinx ISE (version 14.7) [129]. On the contrary, the first version of INDRA (presented in Section 3.2.2) supported just the Virtex-2 family [45].

The overall flow is depicted in Figure 5.1. INDRA 2.0 provides supports in all the steps needed to create bitstream files, starting from an HDL representation. The INDRA 2.0 flow, and its DHHarMa and PSRerouter tools are available as open-source projects [54].

5.1.1 FPGA partitioning

First of all, the user needs to partition the FPGA according to the requirement of the DPR system. Then, he has to specify the static area (*Base Region*) and the dynamic area (*PR Region*) of the system. The partitioning depends on the properties of the selected device, such as the granularity of reconfiguration (length of a configuration frame) and the design components to be placed. 2D rectangular areas can be defined, according to the area constraints of the Xilinx tools. The partitioning of a DPR system is discussed in detail in Section 2.2.2.

The base region is the area of the FPGA that is configured just when the system is initialized. The configuration of the base region does not change at run-time. The PR Region is used for run-time reconfiguration. All dynamic system components are located in the PR Region. The PR Region is composed of one or more Reconfigurable Tiles (PR Tiles). A PR Tile is the smallest partially reconfigurable unit.

It is important to mention that the type of Reconfigurable Tile (PR Tile) is given by the logical resources contained in it. Therefore, two Reconfigurable Tiles (PR Tiles) are of the same type if they contain the same logical resources in the same internal position. The type of a PR Tile is discussed in Section 2.2.2.

After the partitioning step, the HDL source files are then divided into three groups:

- *Static components*: these components are placed in the static part of the FPGA.
- *Dynamic components*: these components are placed in the dynamic region of the FPGA. Every implementation of the dynamic components is the so-called PR Modules.
- *Communication components*: these components are located in both static and reconfigurable areas. They are in charge to manage the communication among the static and reconfigurable regions.

In the following, it is explained how this three kinds of components are synthesized on the device.

5.1.2 Communication Macro Generation (DHHarMa)

The creation of a DPR system that supports bitstream relocation requires a suitable communication infrastructure, which preserves the homogeneity of the system.

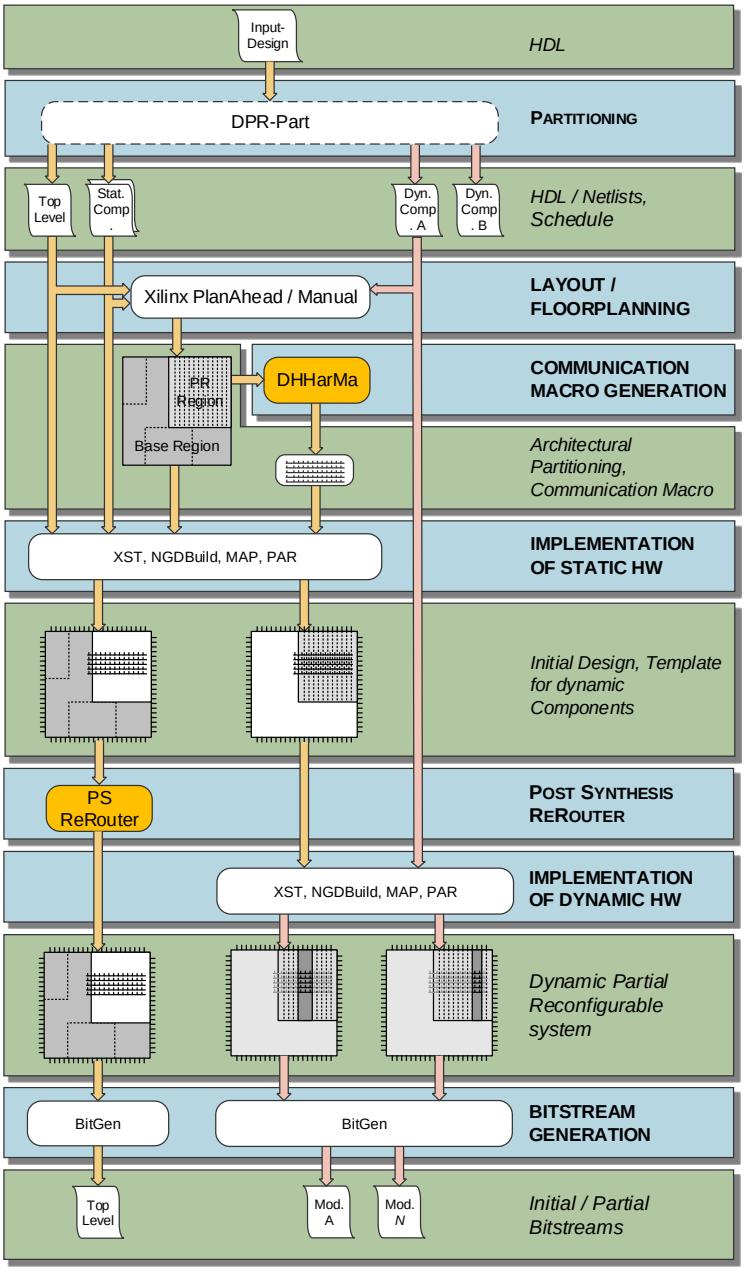


Figure 5.1: Overview of the INDRA 2.0 Flow. In yellow are highlighted the developed tools.

The communication infrastructure interconnects the PR Modules and the base region; it does not have to introduce any further heterogeneity in the system. Therefore, the communication infrastructure that has been described in Section 2.2.4 is utilized in this kind of scenario.

The communication infrastructure needs to be packed, placed and routed considering the homogeneity of the PR Tiles. For these reasons, DHHarMa has been developed. Despite the vendor's tools, DHHarMa provides advanced mechanisms to utilize the resources of the FPGA keeping the maximum flexibility of DPR designs.

In the specific, the resources of the PR Tiles of a specific type are always utilized in the same way by the communication infrastructure; this means that a PR Module that is placed initially in a specific PR Tile (*e.g., PR_Tile_Type1_a*) can be relocated in another one of the same type (e.g., *PR_Tile_Type1_b*).

At the end of this process a placed and routed communication infrastructure is generated, which is stored in the Xilinx logical description of the design and macro library (NMC) format. Details of the DHHarMa implementation are presented in Section 5.2. DHHarMa can be used either within the INDRA 2.0 flow or as a standalone tool.

5.1.3 Static PAR and PSRerouter

Once that the communication has been generated, the static design is placed and routed. First, the communication hard-macro generated with DHHarMa is placed into the FPGA. Then, the static components are placed and routed within the static area utilizing the standard Xilinx tools. The routing of the static components can cross the reconfigurable area, affecting the homogeneity structure of the design (this topic is discussed in Section 3.2.1).

In order to solve this issue, the Post-Synthesis Rerouter (PSRerouter) tool has been developed to provide a post-synthesis rerouting of a design. The problem of the crossing nets and the detail implementation of the PSRerouter are discussed in PSRerouter. At the end of these steps, the design is stored in an NCD format, and it contains all the static and communication components of the DPR system.

5.1.4 Dynamic Modules Implementation

The dynamic components are the ones that can be dynamically reconfigured during run-time, which are grouped in so-called PR Module. A PR Module can be placed and removed at run-time according to the needs of the application.

As presented in Section 2.2.2, the PR Region can be partitioned into PR Tile; the dimension of a PR Module can be either of one PR Tiles or a group of contiguous PR Tiles. Placing a module is equivalent to searching an area with as much contiguous free tiles as needed by the module.

The creation of the PR Modules is provided with the standard Xilinx PR flow. The user needs to define the part of the reconfigurable region where the PR Module has to be placed, and the connections with the communication infrastructure. At the end of this step, for every PR Module, an NCD file is generated.

5.1.5 Bitstream Generation

Once that all the components of the system have been placed and routed, the last step of the INDRA 2.0 is the generation of the configuration bitstream of the DPR system. For this step, the standard Xilinx PR flow is utilized, which generates the static design bitstream (*Top level bitstream* in Figure 5.1) and all the PR Module bitstreams (*Mod. A* and *Mod B.* in Figure 5.1).

INtegrated Design flow for Reconfigurable Architectures (INDRA) keeps homogeneous the static components and the communication infrastructure during the various implementation steps. Therefore, the PR Modules that have been generated for a specific portion of the PR Region can be relocated dynamically in a different area, which has exactly the same type of PR Tiles and same contiguous position. Therefore, the generated DPR system supports PR Modules relocation.

5.2 Design flow for Homogeneous Hard Macros

Hard macros are pre-placed and pre-routed designs, which can be integrated into a system design without requiring an additional place and route process. Suitable packer, placer, and router are required to obtain a homogeneous hard macro. In the following, it is presented the DHHarMa design flow, which targets the generation of homogeneous hard macros for Xilinx FPGAs.

The complete design flow is depicted in Figure 5.2. The *Xilinx-based front-end* generates an unplaced and unrouted design for the target FPGA device. Then, the *DHHarMa back-end* uses the output of the Xilinx-based front-end and generates a hard macro description by applying packing, placing, and routing homogeneously.

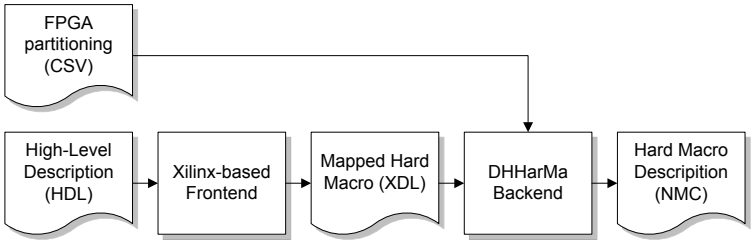


Figure 5.2: DHHarMa design flow for generating homogeneous hard macros [184].

In the following, first, it is presented the Datastructure for Xilinx FPGAs (DXF) database, which contains the FPGA structure information for all the FPGA devices of the Virtex-4, Virtex-5, Virtex-6, Spartan-6 and 7 Series families. The novel features of Datastructure for Xilinx FPGAs (DXF) database are discussed in Section 3.1.

Section 5.2.2 explains the Xilinx-based front-end, focusing on the steps executed to provide a proper input to DHHarMa back-end. Section 5.2.3 discusses the core parts of the DHHarMa flow, which consists of implemented so-called homogeneous algorithms that generate a hard macro with such property (homogeneity). Moreover, it provides details of how a certain FPGA can be partition in homogeneous areas.

Finally, Section 5.2.4 explains how the final XDL file is generated. This file provides the standard Xilinx tools a hard macro, which is homogeneously placed and routed.

5.2.1 Datastructure for Xilinx FPGAs (DXF)

The XDL report (described in Section 2.3.3) can take up to 73 GByte of space (Virtex-7 2000t), which implicates disadvantages in a direct use:

1. *The report stores redundant information*: a tool needs to manage them on-line, adding an extra computational cost.
2. *Space needed to get a full description of an FPGA*: storing all the reports in the XDL format needs lot of memory (about 10TB).
3. *The report does not reflect the inherent homogeneity of the FPGA fabric*: this makes difficult for a tool to exploit the homogeneity of a device.

These disadvantages are the main reasons to create a custom database for a Xilinx FPGA: *the Datastructure for Xilinx FPGAs (DXF)*. This database provides fast access to the various components of the device. For example, a router may check millions of possible paths when creating a single path between two pins; the faster the access to a wire component is, the lesser is the time needed to create a path.

One of the more important features of this database regards the PWs information. In the XDL report, a PW is not stored as a set of wires that composes it; on the contrary, the wires contains the information of the connection to one or more wires of the corresponding PW. Therefore, in order to extract all the wires that compose a PW, an algorithm needs to visit all the wires and verifies all their connections with other wires. In the DXF database, this information has been organized in so-called *PW objects*.

In addition, the PWs created from the XDL report are integrated with the PWs information extracted with the FPGA-Edline tool. The steps to extract this information and the use in the PSRerouter flow are explained in Section 5.3.4. As

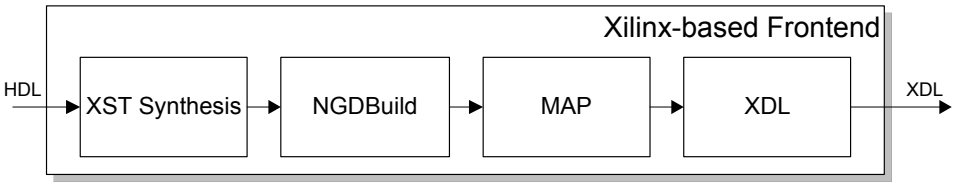


Figure 5.3: Structure of the Xilinx-based front-end for the design flow.

motivated in Section 3.1.5, the PWs information extracted with the FPGA-Edline tool are not provided in any other Xilinx FPGA database (either from the vendor or from academic projects).

More details and benchmarks regarding the DXF database implementation (starting from the XDL report) are provided in [98].

5.2.2 Xilinx-based front-end

The front-end of DHHarMa uses the Xilinx FPGA tool chain to perform the technology mapping of the hard macro design (Section 2.3.1). The front-end generates an XDL representation of the mapped hard macro (XDL is explained in Section 2.3.3) starting from a high-level description in HDL. Programs involved in this process are shown in Figure 5.3 and explained in the following:

1. The *XST synthesis* tool transforms the HDL source files into an architecture-specific netlist (Native Generic Circuit (NGC) format). This file contains the logical design data and additional constraints (e.g., timing or implementation constraints). The XST synthesis analyzes the VHDL code and infers specific design building blocks (LUT, RAM, MUX, etc.).

2. The NGC file is a direct input for *NGDBuild*, which reads the netlist and transforms it into a logical design file NGD. This file describes the design in elements such as logic gates (e.g., AND, OR, etc.), flip-flops and RAM.

3. The *MAP* tool applies the device-specific mapping of the logical design file and translates the design into a device-specific description (e.g., CLBs, IOBs, DSPs, etc.).

4. Finally, the *XDL* tool converts the output of the mapping into an XDL format, which serves as an input for the DHHarMa back-end.

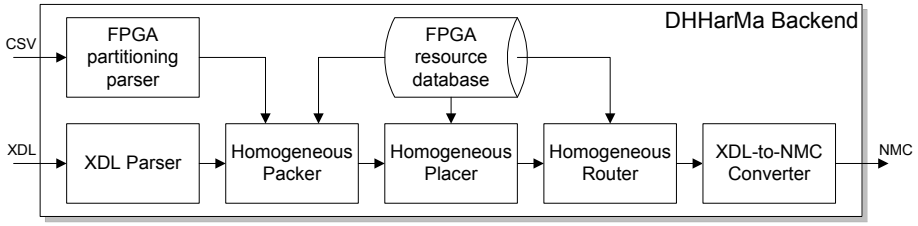


Figure 5.4: Structure of the DHHarMa back-end [184].

5.2.3 DHHarMa back-end

The *DHHarMa back-end* uses the technology mapped design description from the Xilinx-based front-end to apply homogeneous packing, placing, and routing. An overview of the DHHarMa back-end is presented in Figure 5.4.

DHHarMa inputs

In addition to the design description in XDL format (described in Section 5.2.2), DHHarMa takes in input an FPGA partitioning file (in Comma-separated values (CSV) format); this file defines the location, size, and type of region and tiles of the system. Each region (regardless of the type: static or reconfigurable) is specified by two coordinates; therefore, only rectangular-shaped regions are supported.

An example of the partitioning of a Virtex-6 FPGA with nine tiles of four different types ($STATIC$, $REC0$ - $REC3$) is shown in Figure 5.5. The FPGA Editor has been used to get the two coordinates for the regions (represented with bullets in the figure).

Another input of DHHarMa back-end is the DXF database (Section 5.2.1). The homogeneous packer, placer, and router require resources and structure information, such as the layouts of CLBs, switch-matrices, DSPs, and their connections between each other.

Homogeneous steps

After parsing the FPGA partition file and the mapped macro XDL description , the homogeneous steps are executed.

The *homogeneous packer* rearranges the resources to guarantee a homogeneous packing of the slices within each tile of same tile type. This is achieved by the *packing* process, wherein the slices within each tile are dissembled into the lowest atomic blocks called *Parts*. For example, a Part of a Virtex-6 FPGA comprises a functional unit (including two LUTs) and two registers. After the dissembling process, the Parts are reassembled to build a homogeneous slice configuration,

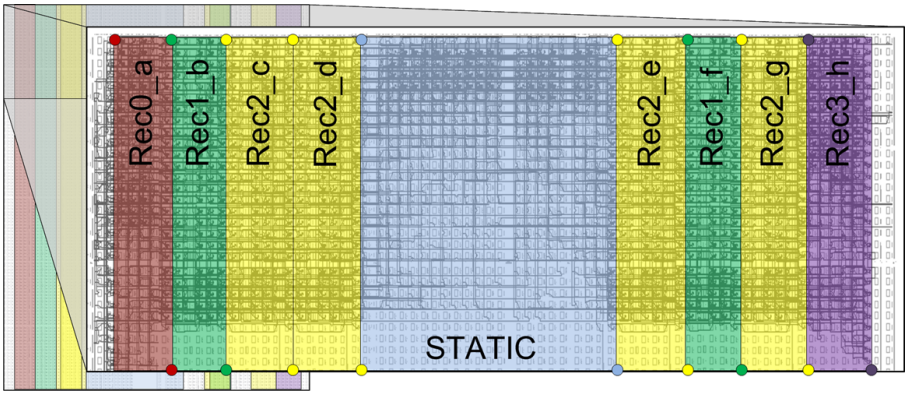


Figure 5.5: Example of the partitioning of a Xilinx Virtex-6 FPGA with nine tiles of four different types. A communication macro is placed respecting the homogeneity constraints [184].

which is valid for all tile of the same type. The functionality of the packer is described in detail in [98].

The *homogeneous placer* places the reassembled packed slices of each tile in a way that the homogeneity of placement is maintained for all tiles of the same type. Details of the placer are provided in [98].

The *homogeneous router* (the last homogeneous step) computes the paths for each net connecting the slices of the macro. The routing is performed such that each PR Tile of the same type uses the same routing resources. In the specific, homogeneity is achieved utilizing the PIPs in the same relative position within each PR Tile of the same type.

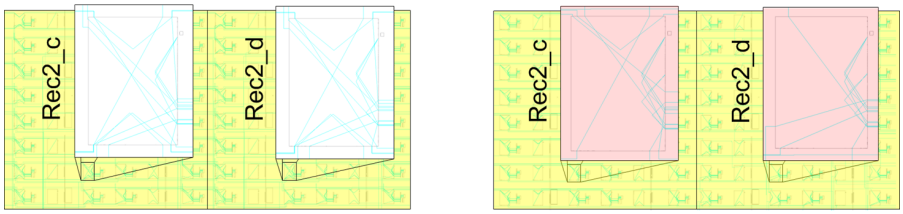
Figure 5.6a shows how the DHHarMa router ensures this kind of routing in two PR Tiles of the same type (i.e., Rec2_c and Rec2_d). Figure 5.6b shows how the Xilinx router does not respect the homogeneity in its routing algorithm. The DHHarMa router is described in detail in Chapter 6.

After the routing phase, an XDL of the homogeneous packed, placed and routed macro is generated. Finally, the XDL representation is transformed into an NMC file by using the Xilinx XDL tool.

5.2.4 Output XDL File

The output file generation is the last step of the DHHarMa flow. In particular, the input XDL is modified such that the complete communication infrastructure mapped, packed, placed and routed is stored.

Section 5.2.3 presents how the DHHarMa back-end is divided into three compu-



(a) DHHarMa Router. (b) Xilinx Router.

Figure 5.6: Comparison between inhomogeneous (Xilinx router) and homogeneous (DHHarMa router) routing of a hard macro.

tational parts: packer, placer, and router. At the end of each step, the XDL file is modified; this step by step modification of the XDL allows using the single parts of the DHHarMa back-end independently. For example, it is possible to take in input an XDL packed and placed by another tool and to execute just the DHHarMa router phase.

Figure 5.7a shows an example of an input XDL and Figure 5.7b present how the single steps of the DHHarMa back-end modify it. The packer can group in different ways the *parts* of an instance; therefore, it can also create or delete instances. Consequentially, if the outPins and inPins of the nets are modified, the packer changes the corresponding values in the nets part.

In the placement phase only the position of the wires is modified; therefore, the only modifications are on the slice allocation information of every instance (as highlighted in Figure 5.7b).

The last modification of the XDL is made by the router, where the pips utilized by the routed nets are added. For example, the net *static_right_out7* of Figure 5.7 shows how the pips are represented in the XDL file. Moreover, the router may modify the inPin of a net within the same instance part (this operation is called *pinSwap*). Eventually, XDL can be converted into an NMC file, and it can be utilized with the existing Xilinx tools.

5.3 PSRerouter

The routing of an FPGA cannot be fully controlled with standard tools. Several academic works provided studies and algorithms to find the better strategies to adopt. On the one hand, the utilized routing strategy by the vendor tools is not documented. On the other hand, the user can control just few setting of the router; for example, the user can decide if the routing policy has to either optimize the timing or the utilized routing resources.

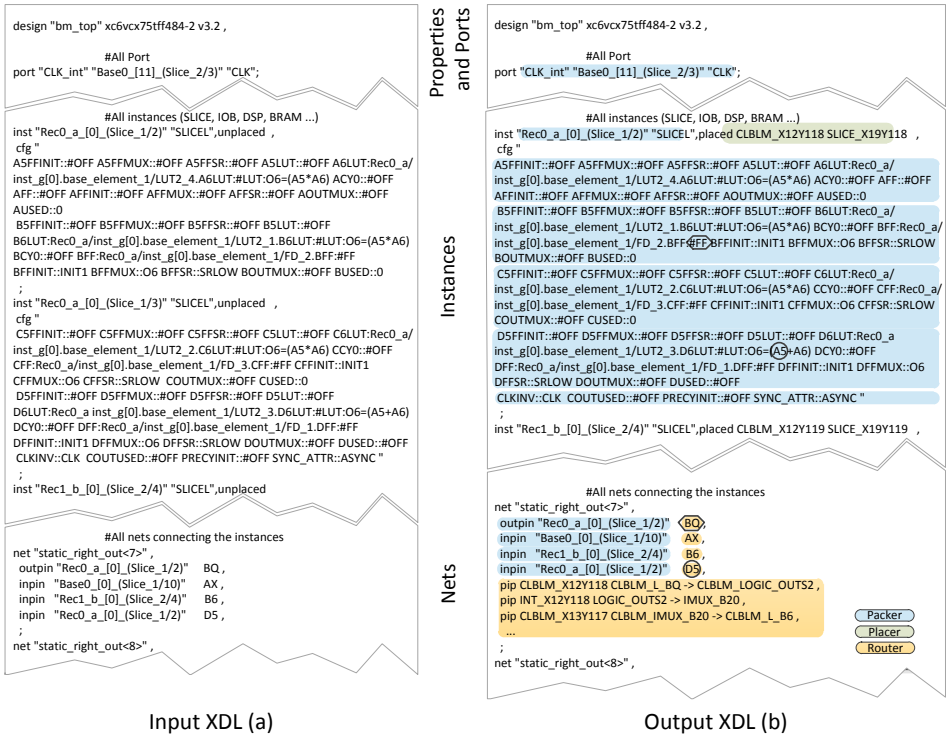


Figure 5.7: DHHarMa flow modifications of the XDL [184].

In this thesis, it is discussed how the routing of a design can be controlled and modified utilizing the XDL intermediate language. As shown in Section 2.3.3, the conversion XDL-NCD-XDL is suitable for small design (e.g., hard macros). Instead, in full designs, the conversion do not store all the information (e.g., IOBs and DCMs settings) due to the fact that the conversion of full design has not been fully considered by Xilinx.

Hence, the novel Post-Synthesis Rerouter (PSRerouter) has been developed, which enables the user to reroute specific nets of a design directly utilizing FPGA Editor (without the need of converting it in XDL). In the specific, PSRerouter allows rerouting specific nets of a static design, which cross the PR Region of the system.

Section 5.3.1 motivates the PSRerouter; it explains why it is important to extract the PWs information directly from FPGA Editor. Moreover, it presents the limitation of the Xilinx approach to control routing of specific nets. Section 5.3.2 discusses the main features of PSRerouter. It shows how the user can control the

routing of specific nets and modify them, without affecting the rest of the design. Section 5.3.3 presents the PWs information, explaining how they are extracted and how the PSRerouter can utilize them.

Section 5.3.4 explains how the PW database is generated and integrated into the DXF database. The section provides benchmarks of the database creation; the time required to create the database and the size occupied by the extracted information of different FPGAs are discussed. Finally, Section 5.3.6 explains the PSRerouter flow. It explains which input files are required and how the so-called "crossing-nets" are detected. Furthermore, it presents how the nets are rerouted. The FPGA-Edline script that modifies the design is presented.

5.3.1 Problem definition

Xilinx allows setting area constraints to create a DPR system. Setting area constraints can control the PAR algorithm to limit the placing and the routing of the design in a specific net. The placing phase respects fully this constraint; once that a specific part of a design has to be placed in a specific area of the FPGA, the Xilinx algorithm places it within the defined borders (in case the area does not contain enough resources the placer requests the user to increase the area's size). On the contrary, the routing phase does not always respect the area constraint provided by the user.

This can generate an inhomogeneous DPR design (see Section 3.2.6). Whenever a static region is defined, the Xilinx router does not respect the constraints of the area defined; this means that static nets that have both the outPin and inPin in the static region may utilize routing resources within the PR Region.

However, Xilinx provided such methods to control and constrain the routing of nets:

- *DIrected Routing Constraints (DIRT)*: the user can generate constraints that lock down the routing to Xilinx User Constraints File (UCF) by using the DIRT constraints. These can be generated utilizing the DIRT command to specify the location of the UCF file [128]. In this way, the user can fix the routing of specific nets in different PAR of a specific design.

- *XDL manipulation*: the design can be converted to XDL format. Then, the nets routing resources can be modified/added.

These two methods do not allow solving the issues to constrain the routing resources to generate a homogeneous DPR design. DIRT constraints, on the one hand, enable the user to route either automatically or manually a specific net and fix it as a constraint for further PAR of the design. On the other hand, it needs multiple reruns of the PAR algorithm to get the desired results; moreover, the

rerun of the PAR algorithm can introduce further problems in the designs (e.g., timing not respected, nets that do not respect area constraints, etc.).

Regarding the XDL manipulation, this approach is suitable for small design (e.g., hard macros). Instead, it cannot be utilized for full design due to the loss of information that the conversion NCD-XDL-NCD generates (Section 2.3.3).

As discussed in Section 3.2, researchers have provided different kinds of approaches to guarantee the relocation of the reconfigurable modules in DPR systems. For example, the works presented in Section 3.2.3 and Section 3.2.5 use so-called *block hard macros* to block the routing resources of the PR Region during the static design PAR.

Xilinx does not provide an easy-to-use way to control the routing of the nets. However, the routing of the nets can be manipulated utilizing FPGA Editor and FPGA-Edline, which allow the user to reroute nets manually or with a script. The issue of utilizing a scripted approach is the loss of routing information to be utilized. In the specific, Xilinx does not provide a database of the PWs and PIPs, which can be utilized to route a net. Therefore, the user needs to select in FPGA Editor the routing resources manually and to execute a manual rerouting.

Hence, it is necessary to retrieve the PWs information of the FPGA devices to provide an automatized way that enables the rerouting of a net. The Section 5.3.2 explains how this information can be extracted and how a net can be rerouted in order to preserve the module relocation property.

5.3.2 Implementation Idea

The main idea of the PSRerouter is to have a flow that is able to take in input a placed and routed design and to verify that the static nets are not crossing the static region borders. If some nets are not respecting this constraint, the PSRerouter reroutes them; an FPGA-Edline script is generated, which can be directly executed on the NCD design.

As explained in Section 5.3.1 a new PW database is required to provide an automatized way to control the routing of the nets.

The manual rerouting of the crossing nets can work for small design; however, the complexity of this operation grows up with the increase of the FPGA target device and design complexity. In addition, a manual rerouting needs a deep knowledge about the routing structure of an FPGA. Instead, FPGA-Edline script solution allows having a script, which can reroute nets in few seconds.

The information needed to select routing resources in FPGA-Edline is different from the XDL info. Therefore, it is necessary to extract the so-called “nodeidx” information, which is a unique index of a PW within an FPGA. This information is then integrated with the DXF database generated from the XDL report.

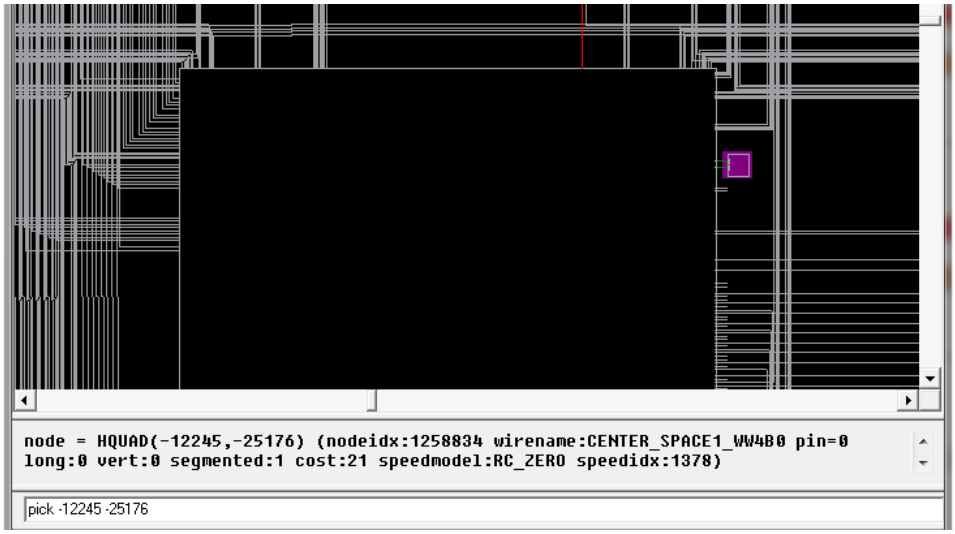


Figure 5.8: FPGA Editor screenshot [128] of a Virtex-6 LX75t FPGA. It shows how a PW can be picked, and the PW information is visualized.

5.3.3 Physical Wire Info

The nodeidx information is needed to select a PW in FPGA Editor or FPGA-Edline; it is necessary to find a relation between the XDL wire name and the PW nodeidx.

Unlike the XDL, Xilinx does not provide the nodeidx information as database or report. Therefore, it can be extracted just "clicking" on a specific PW in the FPGA Editor GUI. Figure 5.8 provides an example of the information that appears once that a PW is selected.

As the example shows, the wire's data contains the following key info:

- *type* of the wire (HQUAD).
- *reference point* of the selected PW ((-12245,-15176)). Every component of the FPGA can be selected via an XY coordinate. The coordinate (0,0) represents the center of the FPGA.
- *nodeidx*: an index that identifies univocally every PW of the FPGA.
- *wirename*: this information is the same information presented in the XDL.

Having this four information, it is possible to find a relation between the wires extracted from the XDL and the PW nodeidx. The router can utilize the DXF database and generate a script as output, which can be run with FPGA-Edline.

AutoIt

As mentioned, the needed PW information can be extracted just "clicking" on each PW of every FPGA; of course, this is an exhaustive approach that cannot be executed without the use of any scripting language. Hence, the extraction has been made utilizing AutoIt [115].

AutoIt is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. Therefore, AutoIt is able to execute the script on a GUI and get back the needed information; a set of scripts have been developed, which first select a PW and then extract the wire information.

Simple and Advanced PW Info

Xilinx does not provide a direct way to extract the required PW information. Therefore, the needed information is extracted utilizing the AutoIt scripting language. The wire information is represented in two different ways, according to the utilized selecting method. The PW information can be:

- *Simple info*: the extracted information just contains the wire type and its coordinate (e.g., node = HQUAD(-12245,-25176))
- *Advanced info*: the information extracted contains detailed information of the PW (e.g., node = HQUAD(-12245,-25176) (nodeidx:1258834 wirename: CENTER_SPACE1_WW4B0 pin=0 long:0 vert:0 segmented:1 cost:21 speedmodel:RC_ZERO speedidx:1378)

This categorization has been highlighted to motivate the implementation steps described in Section 5.3.4; moreover, detailed benchmarks show the time required to extract both the simple and the advanced information.

Tile info

The coordinate of the PW info shows the point of reference of the PW, which corresponds to a specific tile position. Therefore, in order to create a custom route, for every PW is necessary to have the reference coordinate and the tile name associated with its PW reference coordinate. As for the PWs, the tile info is extracted by "clicking" on a specific tile.

Figure 5.9 shows an example of the tile information of an INT. The coordinate (98,89) indicates the (row, column) coordinate of the selected tile; hence, the (0,0) coordinate corresponds to the upper left tile of the FPGA device. The *name* information provides the full name of the selected tiles, which also includes the (X,Y) coordinate of the tile (in the example, X39Y35); the coordinate X0Y0 corresponds

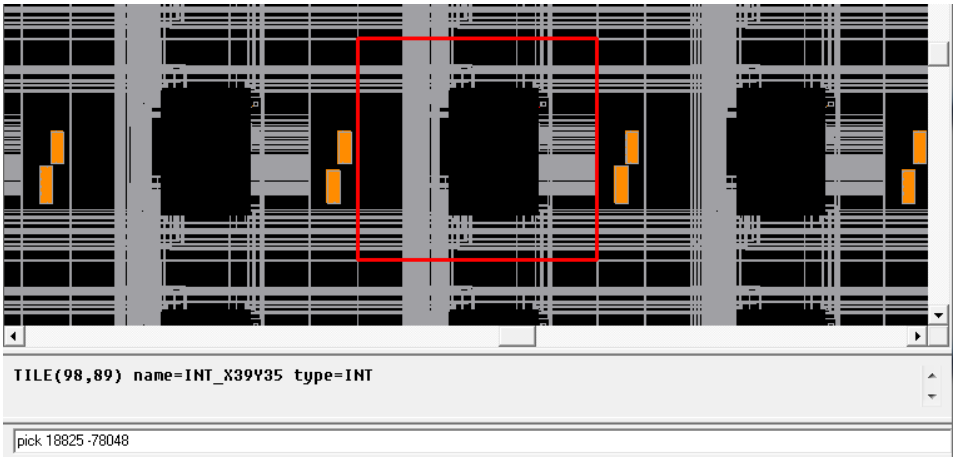


Figure 5.9: FPGA Editor screenshot [128] of a Virtex-6 LX75t FPGA. It shows how a tile can be picked, and its information is visualized.

to the bottom left tile. Finally, the *type* field provides the type of the selected tile (in the example, INT).

Hence, the tile info provides two different coordinates. Although this information seems redundant, they provide different information regarding the position of the tiles. The row/column coordinate is a unique coordinate for each tile of a specific device. On the contrary, the X/Y coordinate can be the same for different tiles.

More in detail, the row/column coordinate can be seen as the position of the tile in a 2D view of the FPGA; this view represents the low-level implementation of the FPGA and the abstracted view in FPGA Editor.

The X/Y coordinate represents the position of the tiles, grouping them according to the INT tiles. As explained in Section 2.1, every logical component of the FPGA is paired with an INT tile, which allow the signal to be routed within the FPGA. For example, every CLB and its INT tile have the same X/Y coordinate.

5.3.4 Database Creation Flow

This section explains how the PW database is generated and which are the steps executed to extract the PW information from FPGA Editor. The final database is composed of a set of text files, which contain the advanced wire info. For the considered FPGAs, the PWs info is categorized first according to the tile type; Then, for every tile, a text file has been generated containing the advanced information of the physical wires.

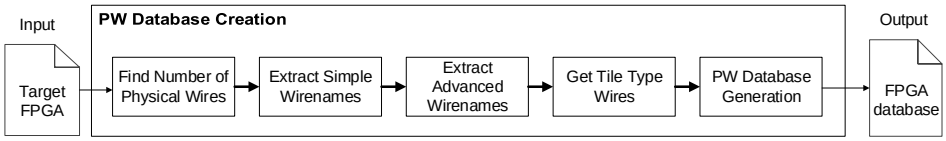


Figure 5.10: PW database creation flow.

Figure 5.10 shows which are the steps needed to create the database. As it is depicted, the PW database creation flow takes in input the name of the target FPGA and generate as output the FPGA PW database. More in general, the flow creates a set of AutoIt scripts that interact with FPGA Editor and FPGA-Edline, extracting and organizing the needed information.

Find Number of Physical Wires

For a target FPGA, this step finds the number of PWs; in the specific, the index of the last nodeidx corresponds to the total number of PW - 1. This value is extracted executing a dichotomic search on the nodeidx index. The algorithm stops when the last selectable nodeidx is found.

The function first executes FPGA Editor, and it opens an empty design; the opened empty design is then used for the research of the last nodeidx index. Once that the value is found, it is stored, and it is utilized for the further steps of the database creation.

The research is made utilizing a dichotomic search algorithm. The function selects a certain PW and verifies if it exists; the recursive function returns when the last nodeidx is found.

Extract Simple Wirename

In this step every PW of a target FPGA is selected on FPGA Editor and the simple wire information is extracted; AutoIt scripts perform this operation automatically. Once that all the information is extracted, they are stored in a file called "FPGAname_wirename_simple.log".

This is the slowest step of the creation of the database because for every PW the information is extracted from a pop-up window of FPGA Editor; the opening and closing of these windows turn in a delay that depends on the windows GUI management. In average, the extraction of the simple information of every wire takes about 70 ms.

The function *extractSimpleInfo* in Algorithm 1 presents how it is possible to extract the simple wire information from FPGA Editor (the simple wire information is

1: **function** extractSimpleInfo($targetFPGA$, $lastWireIndex$)
2: open FPGA Editor and load an empty design of the FPGA $targetFPGA$
3: **for** $iWire$ = 0; $iWire$ <= $lastWireIndex$; $iWire$++ **do**
4: send FPGA-Edline command: "unpost -all; unselect -all"
5: send FPGA-Edline command: "select wire -id" & $iWire$ {*e.g., select wire -id 12*}
6: send FPGA-Edline command: "post attr" {*Shows the info of the PW*}
7: get $PWinfo$, the info of the selected wire $wireId$ from the window
8: store the information on simplewire_database.log {*e.g., HQUAD(-101819,-303576)*}
9: close PW info window
10: **end for**
11: **end function**

Algorithm 1: Pseudo code for the extraction of the simple wirename.

presented in Section 5.3.3). The function takes in input the target $targetFPGA$ and the $lastWireIndex$ computed with the previous step .

extractSimpleInfo first launches the FPGA Editor program and then creates an empty design for the target FPGA. Every PW of the FPGA is selected, and the simple wire information is extracted and stored in the *simplewire_database.log* file.

In Algorithm 1 Line 5 the FPGA-Edline command *"select wire -id"* selects the PW; the command in Line 6 shows the simple properties of the PW.

Extract Advanced Wirename

Once that the simple PW information are stored, another AutoIt script extracts for every PW the advanced PW information. Despite the simple information extraction, this step can be executed utilizing FPGA-Edline. Therefore, since that FPGA-Edline is not a GUI but a prompt program, the information are extracted without having the delay given by the GUI. Moreover, different tasks running in parallel can be executed to reduce the extraction time. On average, running the extraction of PWs on eight different tasks this operation takes about 1 ms per PW.

Algorithm 2 shows the pseudo code of this step. Function *createExtractorScript* (Line 1) creates the FPGA-Edline script. The function takes in input the target FPGA ($targetFPGA$) and the $simplewireLogFile$, which has been generated in the previous step. The script is stored in the file $advancedInfoFPGAedlineScript$.

In the script are written first the FPGA-Edline commands required for the creation of an empty design of the target FPGA. Then, it is extracted from the simple wire log file ($simplewireLogFile$) the PW's coordinates; these coordinates are the point of reference of the PWs (see Section 5.3.3).

The function *extractAdvancedInfo* (Line 12) represents how the script generated by the previous function is executed and how the advanced information is extracted. In Line 14 the script is executed. At the end of the computation, FPGA-Edline

1: **function** createExtractorScript($targetFPGA$, $simplewireLogFile$)
2: create and open the file $advancedInfoFPGAedlineScript$
3: write on the file $advancedInfoFPGAedlineScript$ the FPGA-Edline command to open $simplewireLogFile$
4: write on the file $advancedInfoFPGAedlineScript$ the FPGA-Edline command "wire_layers_enabled"
{*each line contains the simpleWire inf, e.g., HQUAD(-101819,-303576*)}
5: **for** each line $iWire$ of $simplewireLogFile$ **do**
6: extract the x,y coordinate of the wire {*x: -101819, y: -303576*}
7: write on file $advancedInfoFPGAedlineScript$ the FPGA-Edline command: "pick -a " & x & " " & y {*pick -a -101819 -303576*}
8: **end for**
9: close files $simplewireLogFile$ and $advancedInfoFPGAedlineScript$
10: **return** $advancedInfoFPGAedlineScript$
11: **end function**

12: **func** extractAdvancedInfo($targetFPGA$, $advancedInfoFPGAedlineScript$)
13: open FPGA Editor and load an empty design of the FPGA $targetFPGA$
14: execute $advancedInfoFPGAedlineScript$
15: open FPGA Editor log file
16: create and open $advancedwireLogFile$ file
17: parse the file and extract for every "pick command" the advanced wire information {e.g., node = BENTQUAD(1225,-6740) (nodeidx:1860030 wirename: CLBLM_NW4END0 pin=0 long:0 vert:0 segmented:1 cost:3 speedmodel: RC_CLBLM_NW4END speedidx:603))}
18: write advanced information in $advancedwireLogFile$
19: **return** $advancedwireLogFile$
20: **end func**

Algorithm 2: Extract advanced PW information pseudocode.

generates automatically a log file that contains all the results of the commands executed. In the specific, once that the command *pick -a " & x & " " & y* is executed, if the coordinate corresponds to a PW, the advanced information of the wire is stored in the log file. It is important to mention that the coordinates have been extracted from the simple extraction steps; therefore, all the coordinates utilized for these pick operations are always "picking" on PWs.

Line 17 parses all the log file generated by FPGA-Edline and extracts the advanced information, storing them in the file $advancedwireLogFile$.

Get Tile Info

As explained in Section 5.3.3, every represented PW in FPGA Editor has a reference coordinate *(x,y)*. This Cartesian coordinate indicates the position of the

1: **function** createExtractorScript($targetFPGA$, $simplewireLogFile$)
2: create and open file $tileInfoFPGAedlineScript$
3: write on file $tileInfoFPGAedlineScript$ the FPGA-Edline command to open $simplewireLogFile$
4: write on file $tileInfoFPGAedlineScript$ the FPGA-Edline command "wire_layers_disable"
{*each line contains the simpleWire info, e.g., HQUAD(-101819,-303576)*}
5: **for** each line $iWire$ of $simplewireLogFile$ **do**
6: extract the x,y coordinate of the wire {*x: -101819, y: -303576*}
7: write on file $tileInfoFPGAedlineScript$ the FPGA-Edline command: "pick -a " & x & " " & y {*pick -a -101819 -303576*}
8: **end for**
9: close files $simplewireLogFile$ and $advanceInfoFPGAedlineScript$
10: **return** $tileInfoFPGAedlineScript$
11: **end function**

12: **func** extractTileInfo($targetFPGA$, $tileInfoFPGAedlineScript$)
13: open FPGA Editor and load an empty design of the FPGA $targetFPGA$
14: execute $advancdeInfoFPGAedlineScript$
15: open FPGA Editor log file
16: create and open $tilewireLogFile$ file
17: parse the file and extract for every "pick command" the tile wire information {e.g., TILE(75,103) name=INT_L_X30Y100 type=INT_L}
18: write tile information in $tilewireLogFile$
19: **return** $tilewireLogFile$
20: **end func**

Algorithm 3: Get Tile info pseudocode.

PW on the FPGA. In the step *Extract Advanced Wirename*, these coordinates have been used to extract the PW's advanced information, through a pick function. The advanced information is not enough to find a relation between the XDL representation and the FPGA Editor representation of a PW.

It is necessary to know also the name of the tile that corresponds to the reference coordinate of a specific PW. Then, this operation is essentially the same of the one presented in Extract Advanced Wirename; the only difference is in the fact that tiles are picked instead that PWs; therefore, the information stored in the log file generated by FPGA-Edline contains the tile, which corresponds to a specific coordinate pick. Algorithm 3 provides the psedocode of this step.

The function *extractTileInfo* (Line 12) represents how the script is executed and how the tile information is extracted. In Line 14 the script is executed. At the end of the computation, FPGA-Edline generates automatically a log file that contains all the results of the commands executed. In the specific, once that the command

```
xc6vcx130tff1156_PWs_database
├── ...
├── HCLK_VFRAME
├── INT
│   ├── TILE(3,1) name=INT_X0Y199 type=INT.txt
│   │   ├── node = DOUBLE(-151625,321052) (nodeidx:21400
│   │   │   wirename:NE2END0 pin=0 long:0 vert:0 ...
│   │   ├── node = DOUBLE(-151623,321056) (nodeidx:21599
│   │   │   wirename:NW2A1 pin=0 long:0 vert:0 ...
│   │   ├── node = DOUBLE(-151619,321060) (nodeidx:21997
│   │   │   wirename:NW2A2 pin=0 long:0 vert:0 ...
│   │   └── ...
│   ├── TILE(3,2) name=INT_X0Y198 type=INT.txt
│   ├── TILE(3,3) name=INT_X0Y197 type=INT.txt
│   └── ...
├── INT_INTERFACE_TERM
└── ...
```

Figure 5.11: PW database organization.

pick -a " & x & " " & y is executed, if the coordinate corresponds to a Tile, the tile information of the wire is stored in the log file. It is important to mention that the coordinates have been extracted from the simple extraction steps; therefore, all the coordinates utilized for these pick operations are always "picking" on Tiles.

Line 17 parses all the log file generated by FPGA-Edline and extracts the tile information, storing them in the file $tilewireLogFile$.

PW Database generation

In the previous steps, all the needed information have been stored. The following step organizes the extracted data in folder and file to permit users to utilize this valuable information. The information stored in the advanced log file and in the tile name log file are reorganized. For every tile's type (e.g., INT, CLB, BRAM, etc.) one folder is generated; then, for every tile of the FPGA, one file is created, which have as title the full name of the tile. At this point, each PW that have its coordinate reference within the considered tile is inserted into the file. Figure 5.11 shows an example of how the information is organized.

Table 5.1: Results of the PW database for different FPGAs.

FPGA	Number of PWs	Number of Tiles	Find n. PW	Extract Simple info	Extract Advan. Info	Get Tile info	Create PW DB	Total Time
V4 fx12	627,644	4,818	41s	9h:50m	12m:48s	23m:18s	1m:56s	~10h
V4 fx40	1,985,232	14,933	50s	31h:05m	9m:18s	34m:17s	5m:30s	~32h
V5 lx20t	711,807	5,896	48s	11h:12m	2m:28s	24m:52s	1m:41s	~12h
V5 fx70t	2,745,277	23,718	48s	41h:23m	13m:35s	41m:10s	8m:40s	~42h
V6 cx130t	4,613,058	37,769	50s	75h:33m	25m:55s	50m:12s	15m:32s	~77h
S6 lx9t	352,431	3,285	41s	4h:56m	3m:29s	21m:47s	1m:03s	~5h
S6 lx16	509,301	4,526	43s	5h:59m	6m:14s	23m:31s	1m:33s	~7h
S6 lx25t	793,011	7,290	47s	12h:59m	11m:14s	27m:20s	2m:11s	~14h
A7 8	702,554	6,930	47	11h:36m	2m:37s	25m:21s	5m:58s	~12h
K7 70t	2,677,838	24,453	52s	44h:3m	10m:50s	55m:56s	5m:10s	~45h
K7 355t	13,132,573	97,030	50s	217h:32m	01h:33m	1h:33m	40m:06s	~221h
V7 x330t	12,488,818	95630	51s	188h:11m	01h:27m	1h:31m	40m:23s	~192h
Zynq 010	1,129,281	13,335	48s	18h:29m	11m:32s	21m:57s	3m:16s	~19h

5.3.5 Benchmark

Table 5.1 presents the time for the creation of the PW database for different FPGAs; moreover, it is presented the time required for all the intermediate steps of the computation (described in Section 5.3.4). The steps have been executed on an Intel Xeon Processor W3565 with 24 GB of RAM.

The table shows on the left the family and the model of the FPGA considered. Then, it shows the *Number of PWs* and the *Number of Tiles* of the considered FPGAs; these results indicate the dimension of the FPGAs and motivate the different computation time for each FPGA. The PWs computational results are reported in *Find n. PW, Extract Simple info, Extract Advan. Info, Get Tile info, and Create PW DB*; as it will be explained in the following, these results are strongly related to the dimension of the considered FPGA. Finally, the column *Total Time*.

In Table 5.1 can be seen that the computational time can vary from about 5h (in the case of the V4 lx9t) to about 578h (in the case of the K7 355t). Moreover, it can be noticed that the great majority of the time is taken by the step *Extract Simple info*. This step (see section 5.3.4) depends on the number of PWs to be extracted: more in detail, the complexity of this step is $O(n)$, where n is the number of PWs.

This fact can also be observed in Figure 5.12 where it is depicted a cartesian graph of the time required for the creation of the Physical Wire DB, where the x-axis indicates the number of PWs and y-axis indicates the total time required for the creation of the DB (both axises are represented in logarithmic scale). The graph shows the linear complexity described above.

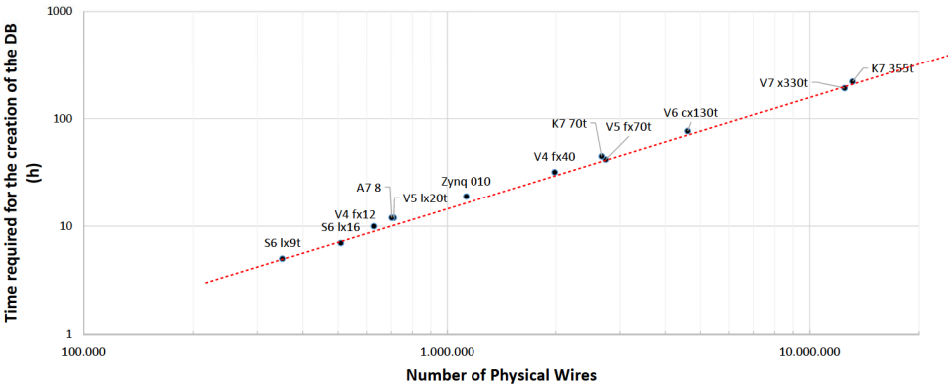


Figure 5.12: Time required for the creation of PhysicalWire DB.

This is due to the time required for the extraction of a single PW. In the 7 Series and Zynq FPGAs have been set different values for the extraction of the PWs, which turn in a higher computational time. For this reason, the two lines that indicate the linear relationship, they have a different slope (gradient).

5.3.6 PSRerouter flow

One of the key parts of the INDRA 2.0 flow is the PSRerouter. This tool is able to reroute certain nets of a P&R design. The PSRerouter can be used as a standalone program as well.

Figure 5.13 depicts how the tool operates; the design is first synthesized, translated, mapped, placed and routed with the standard Xilinx tools (Orange block). Then, the design is converted in XDL. The PSRerouter takes in input the converted design, the partitioning file, and the DXF database (Section 5.2.1).

The key steps of the tool are *the detection of the crossing nets*, *the internal nets rerouting*, and *the FPGA-Edline script creator*. At the end of the execution, the script can be launched in FPGA Editor, and it provides the rerouting of the nets.

Crossing Nets Detection

This step verifies that the static nets of the design are not crossing the dynamic area. In case one or more nets are crossing the reconfigurable area, these nets are detected and listed.

The Algorithm 4 shows how the detection is performed. First, just the nets that have inPin and outPin within the static areas are selected; these are the nets that have to be analyzed. In Line 12, the function checks for every net, if any utilized

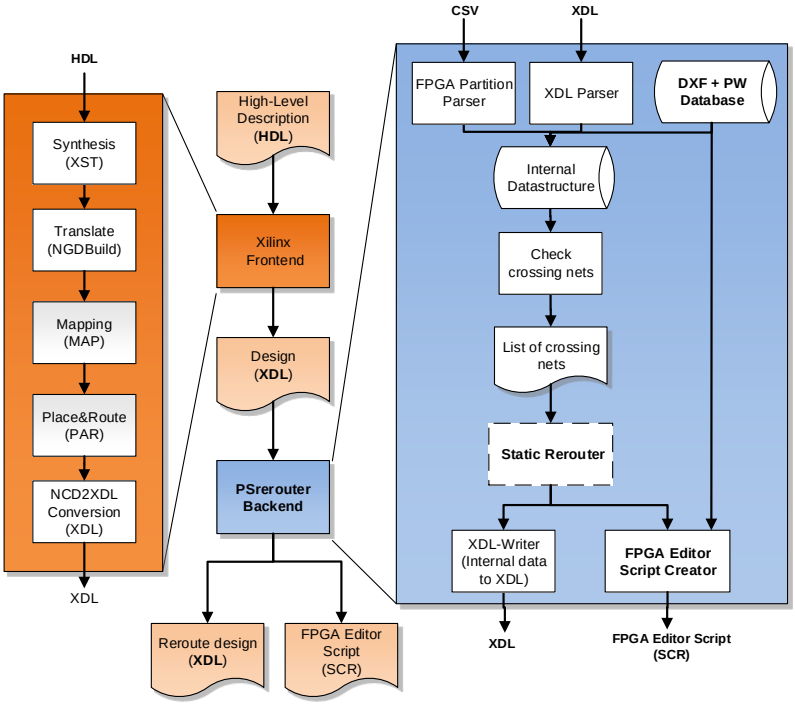


Figure 5.13: PSRerouter Flow.

routing resource is within the dynamic area. If such a net is detected, it is listed in the so-called $crossingNetsVector$.

In the case that all the analyzed nets are not crossing the dynamic region, the program terminates and the input design does not need any modification.

Rerouting of the nets

In this phase of the PSRerouter, the crossing nets are rerouted in such a way that all the utilized routing resources are within the static region. Algorithm 4 Line 19 shows how the rerouting is executed. First, the nets are "unrouted", therefore, all the resources that have been utilized by these nets are freed. Then, for every net a new path is found, utilizing just routing resources within the static region.

Once that a path is found for every net, the algorithm checks if there are no conflict in the solution found. In the case there are conflicts, different paths are found, and the check is performed till no conflicts are presented. In the end, a path for every net is stored.

1: **function** rerouteCrossingNets($fpfFile$, $designXDL$)
2: parse the $designXDL$ file and detecting all the routed nets of the design and their utilized resources. The nets of the design are stored in $designNetsVector$.
3: create vector $recomputeNetVector$
4: $recomputeNetVector$ = detectCrossingNets($fpfFile$, $designNetsVector$)
5: rerouteNets($fpfFile$, $designNetsVector$)
6: write the FPGA-Edline script named $rerouteScript.scr$ that reroutes the nets of the vector $recomputeNetVector$.
7: execute the FPGA-Edline script
8: **end function**

9: **function** detectCrossingNets($fpfFile$, $designNetsVector$)
10: select just the nets that have the outPin and the inPin within the static areas and store them in the vector $staticNetsVector$
11: create vector $crossingNets$
12: **for** each net $iNet$ of $staticNetsVector$ **do**
13: **if** $iNet$ utilizes routing resources of dynamic area **then**
14: add $iNet$ to $crossingNets$
15: **end if**
16: **end for**
17: **return** $crossingNets$
18: **end function**

19: **function** rerouteNets($fpfFile$, $crossingNets$)
20: **for** each net $iNet$ of $crossingNets$ **do**
21: unroute the net $iNet$ {All the routing resources utilized by the net are freed}
22: **end for**
23: **for** each net $iNet$ of $crossingNets$ **do**
24: find a path of net $iNet$ utilizing just routing resources of the static region
25: **end for**
26: **repeat**
27: check if the path of the nets $crossingNets$ they do not use the same resources.
28: **if** a conflict exists **then**
29: reroute the nets that create conflicts
30: **end if**
31: **until** no conflict exist
32: **end function**

Algorithm 4: PSrerouter pseudocode.

```
############Virtex4 XC4VLX100, ScriptName: rerouteScript.scr
open design C:\example\inputDesign.ncd


select net 'NET_0'       #Select the net to reroute
unroute #Unroute the net
unselect -all    #Deselect all the resource in the design
select pin SLICE_X4Y126.Y        #Select the outPin of the net
select wire -id 85028    #Select the PW connected to the outpin
select arc OUTPUT(-50696,107456)->OUTBOUND(-51379,108144)
#PIP that connects PW 3952164 to PW 3949768
select wire -id 81894
select arc IOBIN2OUT(146680,135656)->IOBOUTPUT(146600,135792)
select wire -id 90919
...
...
select wire -id 250777
select pin SLICE_X20Y126.G4Select net inpin
route    #Execute the routing
unselect -all
save
quit!    #Close the design and exit
```

Figure 5.14: Example of a rerouting script.

The PSRerouter utilizes the Iterative Deepening Depth-First Search algorithm (IDDFS) routing algorithm, which is explained in Section 6.2.2. The same routing algorithm has been used for the implementation of the homogeneous router of DHHarMa.

Output script

This phase creates FPGA-Edline scripts that perform the rerouting of the design. Figure 5.14 shows an example of the script where one net is rerouted. The figure shows how first a design is opened; then, a net is selected (in the example the *NET_0*). Once that the net is unrouted, with the command 'unroute' the new path is provided; the script selects the pin, PWs, and PIPs of the new paths. In the end, with the command 'route', the path is stored.

The script can be launched with the command *fpga_edline -p rerouteScript.scr*. At the end of the routing phase of the design, it is necessary to run the Timing Analyzer [129] tool to check if the rerouted nets respect the timing constraints.

5.4 Summary

This chapter has presented the novel INDRA 2.0 flow. The two main tools utilized in this flow have been analyzed in detail: *DHHarMa* and *PSRerouter*.

DHHarMa is a tool that enables the user the creation of communication infrastructure for DPR systems. Its main goal is to synthesize a communication infrastructure homogeneously; it utilizes all the resources in such a way that relocation of the reconfigurable modules can be performed. Its main parts are introduced, i.e., the DXF database, the homogeneous packer, the homogeneous placer and the homogeneous router.

PSRerouter performs the check of a static design of a DPR system and verifies that the nets do not use routing resources of the reconfigurable area. In the case such nets exist, the tool is able to detect and reroute them without affecting any other component of the system. This can be done executing the rerouting directly in FPGA Editor. It has been presented how the PW database needed for this tool has been created. Moreover, benchmarks for the database creation have been presented. Once that new net's paths have been found, the tool is able to create an FPGA-Edline script, which can be executed to reroute the nets of the design (without converting it in XDL).

6 DHHarMa Router

Since the introduction of FPGA devices, new algorithms and tools have been presented, which focus on optimizing the design's PAR (e.g., T-VPACK [13] and VPR [12; 14]). This chapter presents a routing algorithm that despite all the other algorithms can provide a homogeneous routing. The presented router is part of the DHHarMa flow (see Section 5.2) and it is capable of routing nets of a design, respecting the homogeneity constraints. Homogeneity is an important property to keep, e.g., whenever a design supports DPR and/or bitstream relocation.

Tools that create inhomogeneous designs can utilize different optimized routing resources to maximize the clock frequency. On the contrary, for homogeneous design with many PR Regions, the homogeneity forces using a small set of available routing resources. Therefore, the challenge of this router is to provide a homogeneous solution having a small drawback in the performance of the PAR design (i.e., routing resources utilized and maximum clock frequency).

Section 6.1 provides a detailed analysis of the routing structure of the modern Xilinx FPGAs. It compares them and shows how the routing structure evolves through the Xilinx FPGA families. Despite this evolution, the DHHarMa routing algorithm has been kept general and is compatible with most of the modern Xilinx FPGAs families.

Section 6.2 introduces the basis of the DHHarMa router. The router's core algorithm IDDFS is introduced. Moreover, new terminology and concepts are provided; these are important to categorize the nets for the homogeneous routing process (i.e., intra-nets, inter-nets, master regions and slave regions).

Section 6.3 explains the homogeneous routing algorithm and how design's nets need to be categorized to provide a homogeneous result. The main phases of the router are explained, i.e., homogeneous nets grouping, edge region route phase, and intra-route phase.

Finally, Section 6.4 provides detailed benchmarks of the DHHarMa router. For different kind of communication macros, a comparison has been made with the standard ISE routing algorithm. Furthermore, it is presented the communication infrastructure utilized in the DRPM platform (see Section 4).

Table 6.1: Comparison of the Xilinx FPGAs routing properties. This table highlights the routing resources for each INT of the considered FPGA family.

	Virtex-4	Virtex-5	Spartan-6	Virtex-6	7 Series
# OutWires per INT	96	104	104	104	104
# Connected INTs per INT	26	36	22	24	26
# Connections per INT	201	203	118	113	116
# PIPs per INT	3,312	3,992	3,459	3,636	3,744

6.1 General Purpose Routing Analysis

Xilinx interconnection matrix provides an array of routing switches between each internal component. Each programmable element (CLB) is tied to a switch matrix, which allows global routing through the device (see Section 2.1.3). Section 2.1.5 describes the structure of the FPGA's physical wires (PWs). In Addition, the structure of the general purpose routing system (composed by local and long PW) differs in the Xilinx FPGA families.

Differences are most in the number of inWires for every PW, length, and direction of the PWs. Table 6.1 summarizes the main properties of an INT in different Xilinx FPGAs families. In Figure 6.1 and Figure 6.4, for every Xilinx FPGA family, starting from a specific INT, the PWs connected to outWires are depicted; then, the PIPs of the inWires are selected (shown in yellow).

According to the classification of the PWs given in Section 2.1.5, the outPIPs (highlighted in purple) are only within the starting INT; every local PW has only one outWire. In the following, for the considered FPGAs, an analysis of local connections among the INTs is made. To evaluate the properties of the PWs, the concept of Manhattan distance is utilized [31].

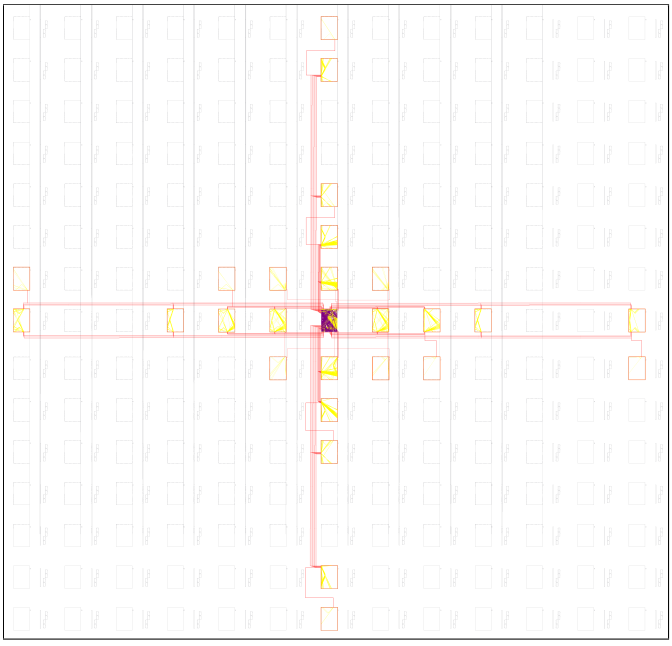
6.1.1 Virtex-4

The local PWs in the Virtex-4 family FPGAs are classified into three types: *Omux*, *Double*, and *Hex*. The global view of the wires is depicted in Figure 6.1(a). In the specific Virtex-4 FPGAs have 96 outWires for each INT (see Table 6.1).

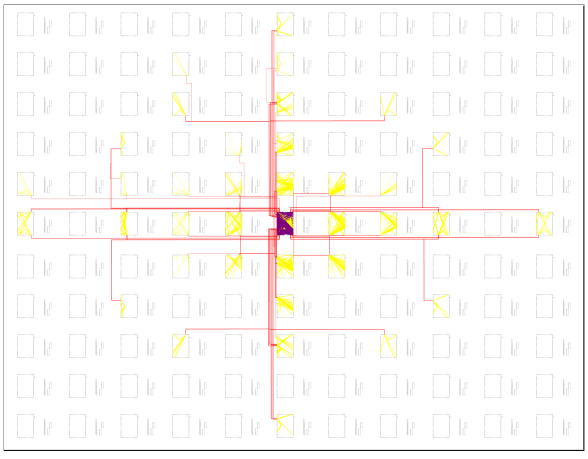
The difference among the PW' types are:

- *Omux*: these PWs connect the starting INT with all the adjacent ones (as it is depicted in Figure 6.2(a)). Vertical, diagonal and horizontal connections are provided. These PWs have mostly one inWires and they are all bounce unidirectional PW (see Section 2.1.5).

- *Double*: the Double PWs are designed to connect INTs to the next two



(a) Virtex-4.



(b) Virtex-5.

Figure 6.1: Connection structure in Virtex-4, and Virtex-5; FPGA Editor screenshots.

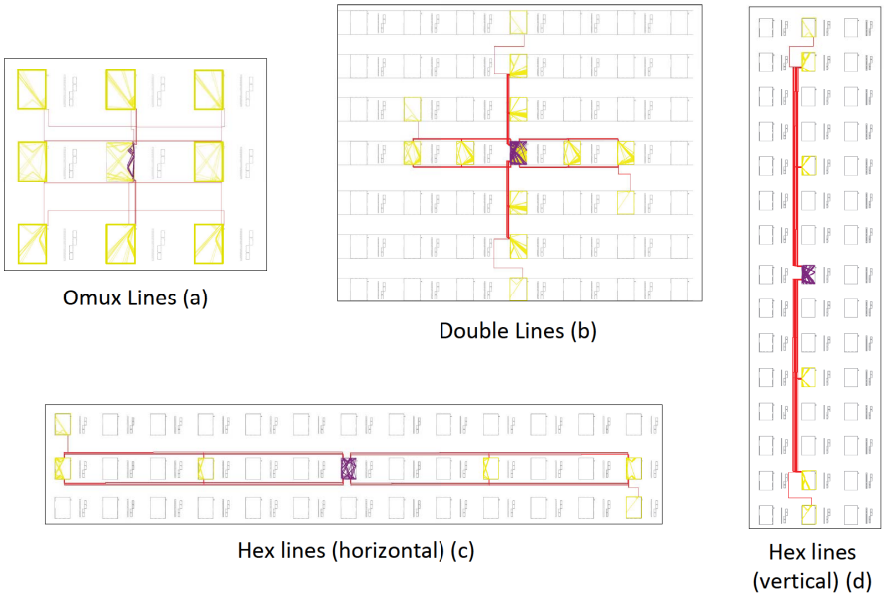


Figure 6.2: Virtex-4 local PWs types; FPGA Editor screenshots.

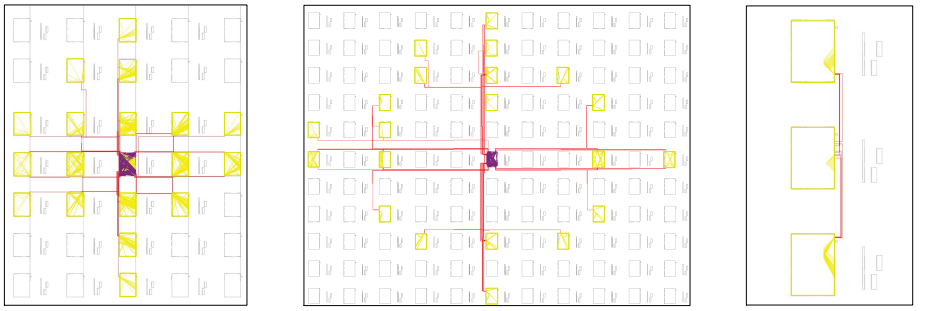
switches in horizontal and vertical directions. Hence, these PWs have two inWires; there is an exception in exactly four wires, where is presented one inWire more that connects an INT at Manhattan distance 3).

- *Hex*: these PWs aim to connect distant INT in vertical and horizontal positions. In particular, it is possible to connect switch matrix at Manhattan distance 3 and 6, from the starting INT; in few cases also a connection at distance 7 is provided. The horizontal and vertical Hex PWs are shown in Figure 6.2(c) and Figure 6.2(d) respectively.

6.1.2 Virtex-5

This family introduces a new routing architecture with an enhanced diagonal routing, improving the block-to-block connectivity [146]. The Virtex-5 family provides the most number of connections to INTs, starting from one specific switch matrix. The outWire in one INT are 104 (8 more than Virtex-4, see Table 6.1); considering all the PWs starting from these wires, it is possible to reach 203 inWire.

Hence, the most difference in the routing system from Virtex-4 and Virtex-5 are in the distribution of the inWire. Keeping almost the same number of outWire and



(a) Double (b) Pent (c) BounceAcross

Figure 6.3: Virtex-5 local lines types; FPGA Editor screenshots.

inWire, in Virtex-5 it is possible to reach 36 INTs (in Virtex-4 just 26). The aim of this new routing configuration is to connect CLBs using fewer routing resources.

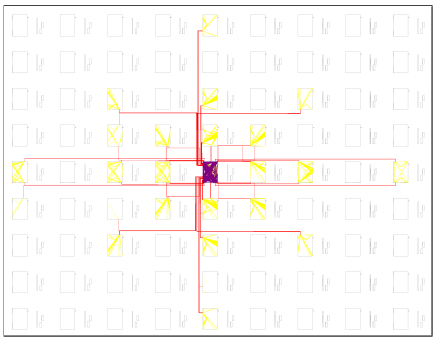
The different PWs types are:

- *Bounce Across*: these PWs connect only two adjacent INTs in a vertical position (as it shown in Figure 6.3c).
- *Double*: connect INTs near to starting switch matrix, in horizontal, vertical, and diagonal positional (Figure 6.3a). These PWs can be considered as a unified version of the Omux and Double PWs of Virtex-4.
- *Pent*: these PWs replace the Hex PWs of Virtex-4. They represent the biggest change from Virtex-4 to Virtex-5 routing structure. These PWs are depicted in Figure 6.3b. As in the previous family, they provide connections with long distant INT in horizontal and vertical directions. Moreover, they allow diagonal connection to distant INTs. This turns in reaching INTs in a horizontal and vertical direction to a maximum Manhattan distance 5; in addition, diagonal connections are provided till a Manhattan distance of 6.

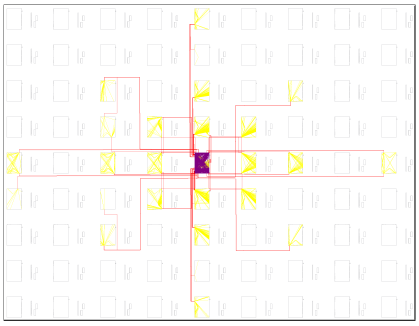
6.1.3 Virtex-6 and Spartan-6

The objective of the Virtex-6 family is to increase performance reducing power consumption [2], compared to the Virtex-5 family. Virtex-6 and Spartan-6 present a minor modification with respect to their Virtex-5 predecessor. In the following, the structure of the Virtex-6 is analyzed.

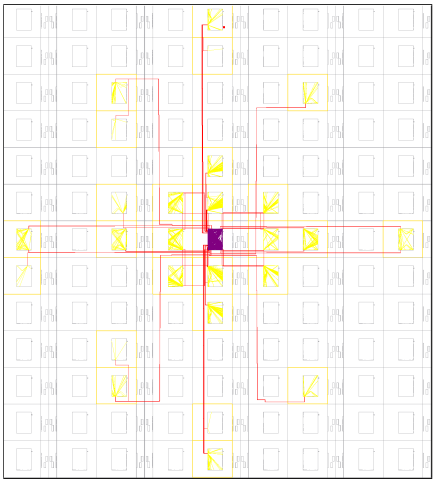
As it shown in Table 6.1, the connected INTs starting from one INT are less than Virtex-5; moreover, the global Manhattan distance of the connections is decreased.



(a) Spartan-6.



(b) Virtex-6.



(c) 7 Series and Zynq.

Figure 6.4: Connection structure in Virtex-6, Spartan-6, 7 Series, and Zynq; FPGA Editor screenshots.

In Virtex-4 and Virtex-5 families, every PW has on average two inWire; in Virtex-5 the outWire of one INT are 104 while the PWs connected to them have globally 204 inWire (in fact, 204/104 is about 2). Differently, Virtex-6 FPGAs have the same number of outWire per INT, despite the PWs connected have 113 connections, almost half compared with Virtex-5. Therefore, the most of the PWs have one outWire and one inWire; only 9 PWs have two inWires.

The Virtex-6 wire types are analyzed in the following:

- *Bounce Across*: as in the Virtex-5, these PWs provide connections with the INT directly above and below a certain switch matrix. These PWs are shown in Figure 6.5c.

- *Single*: these PWs are similar to the Omux PWs of Virtex-4 (see Section 6.1.1). Starting from one switch matrix, they provide connections with all the ad-

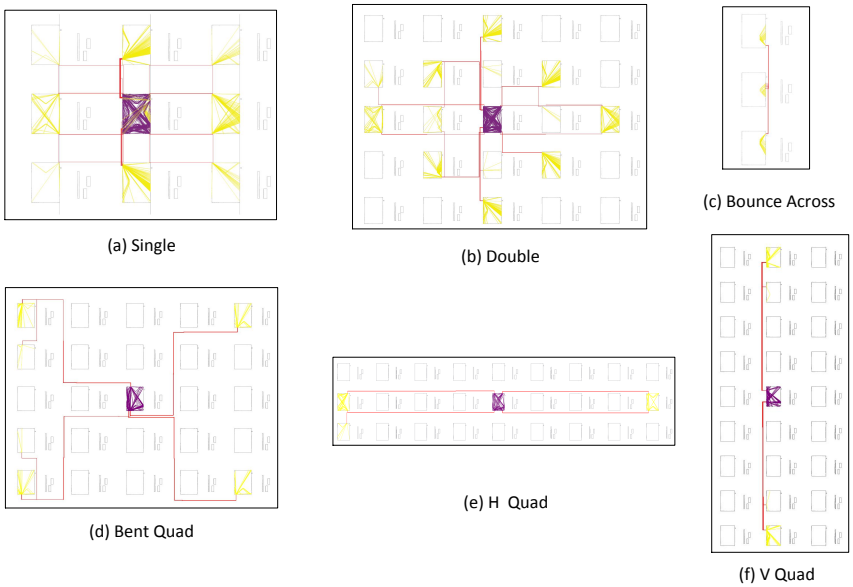


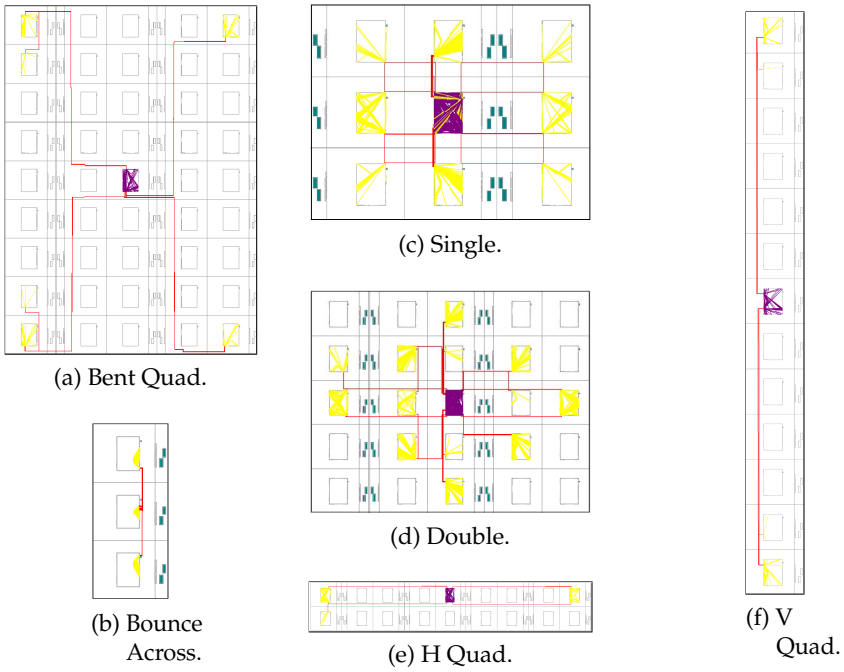
Figure 6.5: Virtex-6 local PWs types; FPGA Editor screenshots.

jacent INTs, in horizontal, vertical, and diagonal direction. Anyhow, the number of these PWs is double compared to Virtex-4 Omux PWs.

- *Double*: according to the reductions of inWires, these PWs provide fewer connections than the double PWs in Virtex-5, as can it shown comparing Figure 6.5(b) and Figure 6.3(a)

- *Quad*: these PWs provide connections with INTs not reached by the other type of PWs. Unlike Virtex-4 and Virtex-5, according to the direction of their connections, these PWs are divided into *Bent Quad* (diagonal connections), *Horizontal Quad*, and *Vertical Quad*. The PWs length, in term of Manhattan distance, is on average 4.

The routing architecture of the Spartan-6 is very similar to Virtex-6; the number of inWire per PW and the number of outWires per switch matrix are the same. Small differences are in the number of INTs connected starting from one switch matrix. Moreover, the global number of inWire of the PWs connected to the outWires of one INT are slightly less.



(a) Bent Quad.

(b) Bounce Across.

(c) Single.

(d) Double.

(e) H Quad.

(f) V Quad.

Figure 6.6: Connection structure in 7 Series; FPGA Editor screenshots.

6.1.4 7 Series and Zynq

The interconnect routing resources are slightly increased in size, quantity, and flexibility about the Virtex-6 FPGA family. The 7 Series family is the Xilinx FPGA with the most number of PIPs per INT, 3744 (see Table 6.1). Instead, the number of connection from one INT to other ones is exactly the same to the Virtex-6 family, 104 (see Table 6.1).

The types of connections are the same as well, as can be seen comparing Figure 6.5 and Figure 6.6. Only the Bent Quad Manhattan distance connection differs from the Virtex-6's Bent Quad.

Despite the general routing structure of global interconnection array, it does not differ much from the Virtex-6 family; the 7 Series introduces a new logic resources placement. For the previous families, the logic blocks are always placed to the right of the corresponding INT matrix. Instead, in the 7 Series family, the logic blocks are alternatively placed to the right and the left. This is well visualized in the line of Figure 6.6. DHHarMa router has been adapted to this new architecture keeping the possibility to provide a homogeneous PAR design.

6.2 Homogeneous Routing Base Concepts

This section introduces the base concepts of the DHHarMa homogeneous routing algorithm; these concepts are needed to understand the various steps of the homogeneous router. The Iterative Deepening Depth-First Search algorithm (IDDFS) is introduced and the global logic of the router is shown as well, such as the direction of the research.

6.2.1 Standard Routing Algorithms

FPGA routing algorithms have been one of the main topics since the introduction of the FPGA devices. Especially in the 90', researchers have developed and investigated different routing algorithms. [22] provides a detailed explanation and comparison of the existing routing algorithms.

Typically, an FPGA routing algorithm is divided in *routing resource graph generation*, *global routing*, and *detailed routing*. First, the FPGA routing resources are modeled in a graph, which abstracts the implementation details of the FPGA. The graph consists of nodes and arcs, where nodes represent the PWs of the FPGA and arcs represents the PIPs. An example of routing graph model is shown in Section 7.3. The global routing executes a coarse-grained routing for the nets. Each net is routed dependability, without considering the conflict with other nets. The last step, the detailed routing, provides a fine-grained routing. In the specific, it takes the results of the global routing and, whenever there is a conflict between the nets, it reroutes them.

One of the first global routers has been the LocusRoute [90]. After that, the standard global FPGA routers have become the PathFinder [77] and the VPR [14] routing algorithm. This two algorithm are based on an iterative routing, where at each iteration the nets are routed using the minimum cost. The cost is determined by the current costs associated with the outpin and the inpins of the net.

About the detailed routers, the most used are the CGE [19] and the SEGA [69] routing algorithms. These routers find first the conflicts between the nets. Whenever a conflict is found, these details routers unroute and reroute the nets utilizing alternative routing resources. More details are provided in [22].

6.2.2 Iterative Deepening Depth-First Search algorithm (IDDFS)

The IDDFS is a search strategy based on the depth-first search algorithm [95]. With this strategy, a depth-limited search is repeatedly run, increasing at every step the depth of the limit value. Using a depth-first search algorithm with a limit on the depth avoids the drawbacks regarding the length of the tree (i.e., a typical problem of the depth-first search algorithm).

The DHHarMa router uses this strategy, and it considers PWs of the FPGA as *nodes* and the PIPs resources connecting them as arcs. The limitation value of the algorithm represents the number of PIPs that can be activated for each path.

Therefore, starting with a limit value of 1, the router tries to reach the inPin with a depth-first algorithm, utilizing not more than one PIP. If it is not possible, the number of PIPs that can be utilized is increased by one: then, the research is run another time.

Whenever a possible path is found, it is stored as a possible routing path of a net. The algorithm terminates the execution when after a depth-limited search visit at least one solution is found. In this way, at the end of the computation, there is a set of possible paths that use a minimum number of PIPs.

The router needs to store more than one solution for the nets to be routed, because, in most of the cases, the routing of one net depends on the routing of another one; since that the goal of the router is to find a homogeneous solution, nets have to be routed in a particular way in order to keep this property.

The possibility to store more paths with a minimum number of resources used is one of a main characteristic of the presented homogeneous router. In fact, this allows having final solutions that optimize the routing on the number of routing resources utilized.

6.2.3 Routing Direction and Wrong Direction

As presented in Section 6.2.2, the routing algorithm is based on a depth-first algorithm. To save computational time during the router process, it is possible to exploit connection structure of the Virtex FPGAs. Hence, the *routing direction logic* is introduced. With a routing direction limitation is possible to prevent useless visit of paths inside the global routing system.

In the route of one net, the router considers that one step is in a *Wrong Direction*, if it increases the Manhattan distance [31] to the arriving point. Hence, for one net, the idea is to find a path taking as less wrong direction as possible.

Figure 6.7 shows how wrong directions are considered in every step of the routing process. In the figure three cases with different values of wrong directions are shown. In the specific, the black lines consider the path already computed. The blue and the red lines show the last step computed by the algorithm. Then, the blue ones indicate that the last part of the routing respects the routing direction constraint (i.e., does not increase the Manhattan distance); instead, the red line shows that the routing is taking a path in a *wrong direction* (i.e., increase the Manhattan distance).

Hence, first, the router executes the IDDFS on a net without taking any wrong directions. In the case the path is not found with this constraint (i.e., activating a high number of PIPs is not possible to reach the inPin), the router increases the

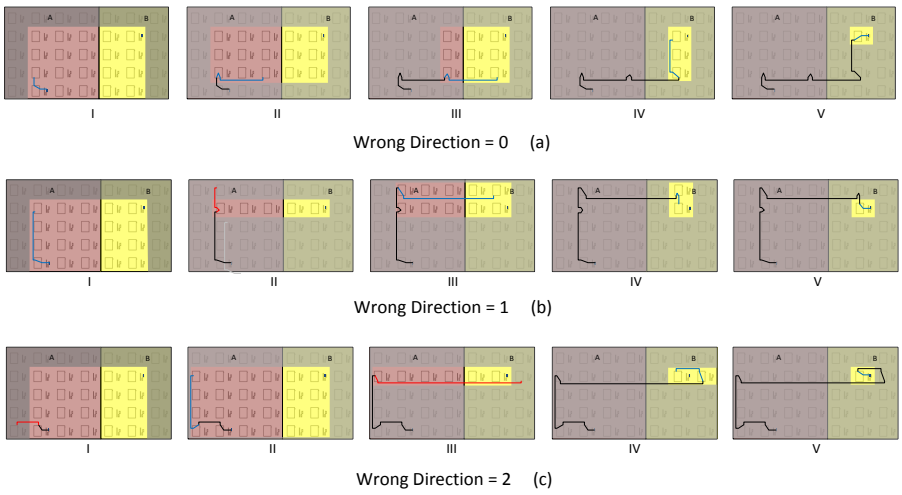


Figure 6.7: Direction constraints of the Router [176].

wrong direction limit by 1. Afterward, the routing process is recomputed with this new constraint value and so on, till a path in not found.

6.2.4 Nets Terminology

The routing algorithm has been written in the C++ object-oriented language. The objects used in the routing process, represent basically the information provided in the XDL file representation. Hence, for every component and net presented in the XDL, one object is created. Anyway, a special classification is necessary for the nets. In the XDL representation, a net is a set of one outPin and one or more inPin.

In this chapter, the XDL net is then called *XDL net*. In the homogeneous routing phase, a net is considered as a point to point connection (one outPin and one inPin). Hence, XDL nets with n inPins are separated into n independent point-to-point connections. Therefore, for every XDL net, are created n nets as much as the number of the inPins.

Figure 6.8 shows an example of an XDL net with more the one inPin. The XDL net *static_right_out<7>* has one outPin and three inPins. Therefore, the XDL net is split in three nets: *static_right_out<7>_0*, *static_right_out<7>_1*, and *static_right_out<7>_2*.

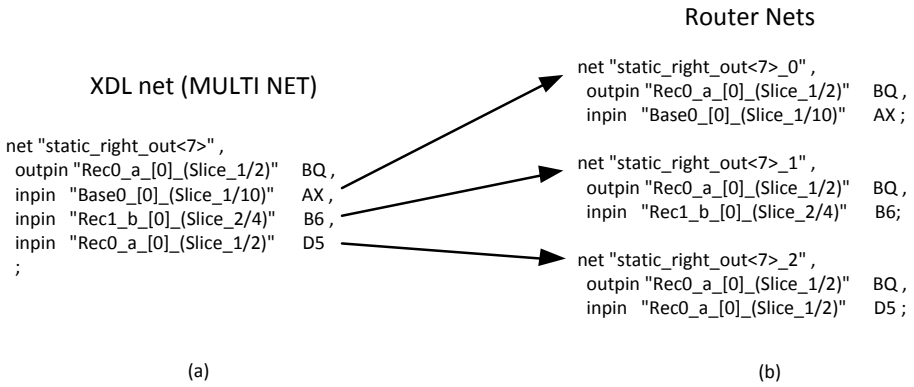


Figure 6.8: Example of XDL net Spitting [176].

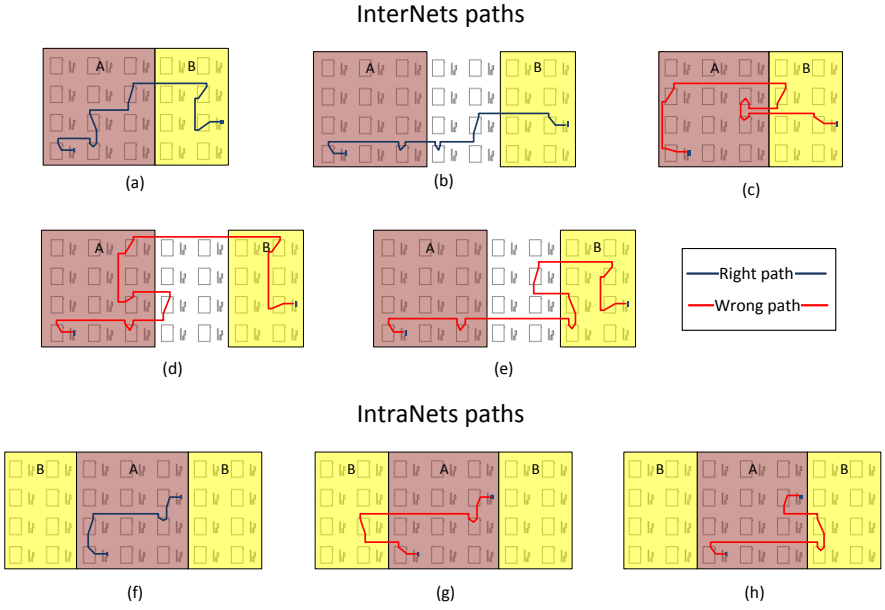


Figure 6.9: Example of right and wrong path to respect homogeneity [176].

Inter-net and Intra-net

As presented, a net is a point-to-point connection from one outPin to one inPin. It is possible to divide the nets into two types according to the partitioning of the FPGA: *intra-nets* and *inter-nets*. On the one hand, the intra-net is a net that have the outPin and the inPin in the same PR Region. On the other hand, the nets with outPin and inPin in different PR Regions are marked as *inter-nets*.

This categorization of the nets entails certain constraints in the routing of these nets. In the following, the constraints are explained:

- *Intra-net area constraints*: the intra-nets are connections within one PR Region. Hence, to respect homogeneity of the whole system, the routing algorithm finds paths that do not cross any area edge. In this way, others PR Regions are not involved in the routing of these nets.
- *Inter-net area constraints*: the inter-nets are connections between different areas. Then, a path may cross the edge of the starting PR Region about an empty area (i.e., part of the FPGA that is not categorized as either static or reconfigurable region) or to the destination PR Region. Once that a path uses a resource that is outside the starting PR Region, then it cannot use the starting area resources anymore. In the case of the path goes into an empty area, it can cross only the destination PR Region.

Figure 6.9 provides some examples of admitted and wrong paths, considering the area constraints, for both types of nets. Figure 6.9[a,b,c,d,e] provide examples of the inter-nets paths. The paths in Figure 6.9[c,d] are a wrong path because they cross more than one time the starting area. The path in Figure 6.9e is wrong because the net crosses more than one time the arrival area.

Figure 6.9[f,g,h] provide examples of intra-nets paths. The nets in Figure 6.9[g,h] are wrong because the paths are crossing the PR Region borders

6.2.5 Net Initialization

In the previous steps it is presented how the incoming net-list is modified and grouped to be utilized by the homogeneous router. Before to perform a global routing, each net is considered without taking care about the homogeneous relations with other nets.

In particular, two important information are retrieved: the minimum number of PIPs required by a net, and the computation of the first and last path. Hence, considering one net, the IDDFS algorithm is computed to find the minimum number of PIPs required to find at least one path. The algorithm is applied considering area constraints that are presented in Section 6.2.4. This information avoids useless path checks in further steps for the homogeneous routing when the computation of more nets is computed merging more IDDFS.

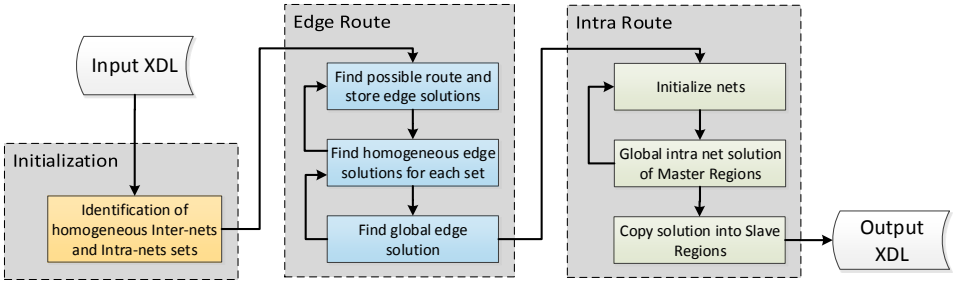


Figure 6.10: Design flow of the homogeneous router.

6.2.6 Master and Slave Regions

The DHHarMa router is capable of providing a homogeneous routing. As presented in Section 5.2.3 this means that the PIPs utilized for each path are activated in the same way in regions of the same type. For this reason, the PR Regions are categorized in *Master Region* and *Slave Regions*.

Considering the partitioning in Figure 5.5, where the FPGA has been partitioned into 8 PR Regions of four different types. Then, 4 PR Regions are marked as Master Regions: $Rec0\_a$, $Rec1\_b$,$Rec2\_c$, and $Rec3\_h$. Consequently, the other regions are marked as Slave Regions: $Rec2\_d$, $Rec2\_e$, $Rec1\_f$, and $Rec2\_g$.

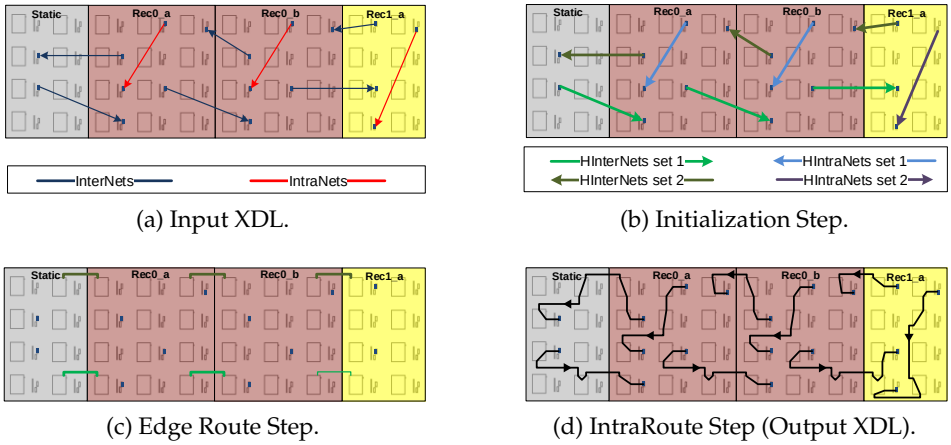
The idea of this categorization is to have a "reference" region, where the routing resources are occupied; then, to preserve the homogeneity of the design, the paths, and the routing resources are replicated into the Slave Regions.

6.3 DHHarMa Homogeneous Router Flow

In the following, the novel homogeneous routing algorithm is presented in more detail. Routing homogeneity is an important property to achieve to guarantee a homogeneous communication infrastructure (as describe in Section 2.2.4). The homogeneous router is part of the DHHarMa design flow (described in Section 5.2), and can route homogeneously a hard macro mapped and placed by the DHHarMa.

In Figure 6.10 the router flow is depicted. This is divided into three main parts: *Initialization phase*, *Edge Route phase*, and *Intra-Route phase*. The Initialization phase is described in more detail in Section 6.2.4. Its aim is to initialize all the input objects, adding some information needed by the router. The other two parts represent the homogeneous routing process.

The Edge Route phase fixes a routing at the edge of the regions for all the nets that cross a region (i.e., nets that have the inPin and outPin in two different regions). The Intra-Route phase provides the routing of the nets that have the inPin and



(a) Input XDL.

(b) Initialization Step.

(c) Edge Route Step.

(d) IntraRoute Step (Output XDL).

Figure 6.11: Example of the homogeneous routing of a design with 9 Nets. The Figure shows how the routing is built during the homogeneous routing process.

outPin in the same region. Moreover, it provides the routing to/from the edge routing computed in the Edge Route phase.

At the end of these three steps, the router gives in output a complete routed design. Afterward, the output XDL is provided (Section 5.2.4). In the following, the single parts of the homogeneous router are described in more detail.

The explanation of the homogeneous router flow is supported by the example in Figure 6.11. This small example shows a portion of the FPGA with three PR Regions: $Rec0\_a$, $Rec0\_b$, and, $Rec1\_a$. Nine nets have to be routed homogeneously in this example. More in detail, the Figure 6.11a shows the input of DHHarMa flow. Then, Figure 6.11[b,c,d] shows the output of the three phases of the DHHarMa router flow.

6.3.1 Initialization Phase

In this phase, the DHHarMa router groups the nets according to the PR Regions of the FPGA. To achieve routing homogeneity, it is essential to find the homogeneous relation among the nets of the design.

For example, if the router is computing the path of one net, it is necessary to know, which are the nets that depend on the routing of the considered net. According to the net's types presented in Section 6.2.4 (i.e., intra-nets and inter-

nets), every net is grouped either in a set of *Homogeneous intranet set (HInterNetSet)* or a set of *Homogeneous internet set (HIntraNetSet)*.

The position of the inPins and the outPins is considered to group the nets homogeneously. The phase is divided into two parts: *HInterNetSet* generation and *HIntraNetSet* generation.

Homogeneous inter-nets set generation

HInterNetSet generation phase can be divided into three steps:

- *Outpin relative position grouping*: in this step, for every PR Region type, and for every relative position of an outPin in the region, one group is created (relative position is explained in Section 5.2.3). At the end of this grouping phase, there are as many groups as the number of outPins presented in every PR Region type. The pseudo code of this step is provided in Algorithm 5.

- *Inpin relative position grouping*: according to the previous step, the nets are grouped considering the inPin relative position. The algorithm is the same used for the outPin relative position grouping (Algorithm 5). The only difference is in the pins selected (i.e., inPins instead of outPins).

```
1:  function outPinNetsGroupping(areaTypeVector)
2:  create outPinInterNetMatrix
3:  for each areaType_i in areaTypeVector do
4:     create interNetVector and add all interNets with outPin in areaType_i
5:     for each interNet_i in interNetVector do
6:        create new list homoOutPinList
7:        for each interNet_j in interNetVector do
8:           if (interNet_i.outPin.relativePos == interNet_i.outPin.relativePos)
             and (interNet_i.outPin != interNet_i.inPin) then
9:              add interNet_j to homoOutPinList
10:             remove interNet_j from interNetVector
11:          end if
12:       end for
13:       insert homoOutPinList in outPinInterNetMatrix
14:    end for
15: end for
16: return outPinInterNetMatrix
17: end function
```

Algorithm 5: Inter-nets grouping considering the outPin positions.

```
1: func homoInterNetSetGeneration(outPinInternetMatrix, inPinInternetMatrix)
2: InterNetMatrix = merge(outpinInterNetMatrix, inpinInterNetMatrix )
3: for interNetList_i in InterNetMatrix do
4:   for each interNetList_j in InterNetMatrix after the position of interNetList_i do
5:     if intersection(interNetList_i, interNetList_j) == true then
6:       interNetList_i = join(interNetList_i, interNetList_j)
7:       remove interNetList_j from InterNetMatrix
8:     end if
9:   end for
10: end for
11: end func
```

Algorithm 6: Homogeneous inter-net set generation.

- *Homogeneous inter-nets sets generation*: the groups created in the previous steps are merged, whenever a net is presented in different groups. The pseudo code of this phase in presented in Algorithm 6. It consists in a comparison between every couple of groups. When a couple shared at least one net, the two groups are merged.

At the end of the computation, every inter-net is only in one HInterNetSet.

Homogeneous intra-nets set generation

The generation of these sets is similar to the inter-net one. Nevertheless, considering only intra-nets, just one grouping phase is required. Hence, for every area type, one HIntraNetSet contains nets that have the inPin and the outPin in the same relative position. In the Algorithm 7 the algorithm of this group phase is presented.

6.3.2 Edge Routing Phase

This phase of the router finds a homogeneous routing in the regions' edges. Algorithm 8 provides the pseudo code of the edge routing phase. Line 1 shows that the function takes in input the HInterNetSet vector. As it has been explained in Section 6.3.1, these groups are composed just of inter-nets (i.e., nets that cross the regions of the design).

First of all, for every HInterNetSet and for every inter-nets (Algorithm 8 Line 2) the function *findPossibleRouteAndStoreEdges* is executed. This function executes the

```
1: Func homoIntraNetSetGeneration(areaTypeVector)
2: intraNetGroupedMatrix
3: for each areaType_i in areaTypeVector do
4:    intraNetVector = get IntraNets within region of area type areaType_i
5:    for each intraNet_i in intraNetVector not inserted do
6:       create homoIntraNetsSet
7:       insert intraNet_i in homoIntraNetsSet
8:       set intraNet_i as inserted
9:       for each intraNet_j in intraNetVector not inserted do
10:         if sameRelativePosition( intraNet_i.outPin, intraNet_j-> outPin and
            sameRelativePosition( intraNet_i.inPin, intraNet_j-> inPin) then
11:            insert intraNet_j in homoIntraNetsSet
12:            set intraNet_j as inserted
13:         end if
14:      end for
15:      insert homoIntraNetsSet in intraNetGroupedMatrix
16:   end for
17: end for
18: return intraNetGroupedMatrix
19: EndFunc
```

Algorithm 7: Homogeneous intra-net sets generation.

IDDFS algorithm on an inter-net, storing all the routing solutions on the edges of the regions.

The second step fixes a unique edge solution for each HInterNetSet. The pseudo code of this function is shown in Algorithm 8 Line 12. Since that this phase computes the edges of the paths, it means that the PIP before the edge and the PIP after the edge have to be activated in the same relative positions in same PR Regions of the same type.

If in the solution computed in the previous step (*findPossibleRouteAndStoreEdges* in Line 4) it is not possible to find a homogeneous edge solution, the findPossibleRouteAndStoreEdges is rerun. Before the rerun, the number of maximum PIPs that every path can use is increased to have more solutions for each HInterNetSet.

The third and last step provides a global solution of the edges of the HInterNetSet vector. Algorithm 8 Line 20 shows the pseudocode of this step. This means that for each HInterNetSet it is selected one edge solution that does not have conflicts with the other HInterNetSet.

```
1: func EdgeRoutingPhase(HInterNetSetVector)
2: for each HInterNetSet_i in HInterNetSetVector do
3:    for each internet_j in HInterNetSet_j do
4:       findPossibleRouteAndStoreEdges(internet_j)
5:    end for
6: end for
7: for each HInterNetSet_i in HInterNetSetVector do
8:    findHomogeneousEdgeSolutions(HInterNetSet_i)
9: end for
10: findGlobalEdgesSolutions(HInterNetSetVector)
11: EndFunc
```

```
12: func findHomogeneousEdgeSolutions(HInterNetSet_i)
13: for each HInterNetSet_i in HInterNetSetVector do
14:    for every net of HInterNetSet_i is possible to select an edge solution that
       respect the homogeneous constraint
15:    if not possible to find edge solution then
16:       MoreEdgeSolutions(HInterNetSetVector)
17:    end if
18: end for
19: EndFunc
```

```
20: func findGlobalEdgesSolutions(HInterNetSetVector)
21: globalEdgeSolutionFound = false
22: repeat
23:    globalEdgeSolutionFound = fix the edge routing for each inter-net
24:    if globalEdgeSolutionFound == false then
25:       increment maximum PIPs to be used by the nets HInterNetSetVector
26:       findPossibleRouteAndStoreEdges(HInterNetSetVector)
27:    end if
28: until globalEdgeSolutionFound == false
29: EndFunc
```

Algorithm 8: Pseudo code of the Edge Routing phase.

6.3.3 Intra-Routing Phase

The Intra-Routing phase is the last phase of the homogeneous router flow. At the end of this phase, the input design is fully homogeneously routed. After the Edge Routing phase, the router has fixed the edges for each region (see Section 6.3.2).

Then, in this step, the routing is provided for both intra-nets and inter-nets;

the inter-nets are just partially routed after the edge routing phase. In the Intra-Routing phase, the routing of the inter-nets is finalized, finding the path from the outPin to the edge and the path from the edge routing to the inPin. Differently, the intra-nets are completely routed in this phase.

Algorithm 9 shows the pseudo code of the Intra-Route phase. As represented in Figure 6.10 this phase is divided into three steps: $Initializationofthenets$, *Global intra-net solution of the Master Region* and *Copy solution into Slave Region*.

Since that the routing of this phase does not cross the border of any region, the routing can be computed just for the Master Region and the copy into the Slave Region. This allows having a homogeneous intra-routing in all the regions of the same type.

In the first step, all the nets to be routed are initialized. In the specific, it is executed the IDDFS for all the intra-nets and inter-nets of a specific master region's area type. Therefore, the Intra-Routing phase is executed once for every areaType of the partitioning. Then, in the example of figure Figure 6.11d the function *IntraRoutingPhase* is executed three times, for the 3 area types: $Static$, $Rec0a$, and $Rec0b$.

Once that all the nets are initialized, the find Global Routing step is executed (Algorithm 9 Line 8). This step finds intra-global routing and it considers one solution for each set, the algorithm tries to find a global solution without conflict among the nets. The router finds the best homogeneous solution in term of utilized routing resources. The pseudo code of this step is presented in Algorithm 9 Line 25.

If a global solution cannot be found, the sets are recomputed relaxing the limit PIPs value for every net. The final solution is the one that has the minimum number of activated PIPs, for each set of homogeneous inter-nets. The algorithm starts to analyze the solutions with the least PIPs activated and selects the first solution found. If the algorithm fails to find a solution, the exploration of path candidates for each set of homogeneous inter-nets is repeated with an increased maximum number of PIPs that can be used (Line 31).

Once that an intra solution has been found, the last step copies the routing of the nets within the master regions into the corresponding slave regions. Then, in the example in Figure 6.11d the solution found for the master region $Rec0\_a$ is copied into the slave region $Rec0\_b$.

In the end, the nets are all fully routed, respecting the homogeneity of the required homogeneity constraints. It is important to mention that the intra-routing can benefit from having a loop-back with the edge-routing phase. At the moment, this functionality can be integrated into a further version of the DHHarMa router.

```
1: func IntraRoutingPhase(areaType)
2: masterRegion = areaType.masterRegion
3: interNetVectorInPin = internets that have the inPin in masterRegion
4: interNetVectorOutPin = internets that have the outPin in masterRegion
5: intraNetVector = intranets with outPin and inPin within masterRegion
6: netVector = merge interNetVectorInPin,interNetVectorOutPin and
   intraNetVector
7: initNetsIntraRoutingPhase(interNetVectorInPin, interNetVectorOutPin,
   intraNetVector)
8: findGlobalRoutingForRegion(netVector)
9: copySolutionInSlaveRegions(areaType)
10: endfunc
```

```
11: func initNetsIntraRoutingPhase(masterRegion)
12: interNetVectorInPin = internets that have the inPin in masterRegion
13: interNetVectorOutPin = internets that have the outPin in masterRegion
14: intraNetVector = intranets with outPin and inPin within masterRegion
15: for each interNet_i in interNetVectorInPin do
16:    IDDFS_algorithm(interNet_i.getLastPWOfEdge, interNet_i.inPins)
17: end for
18: for each interNet_i in interNetVectorOutPin do
19:    IDDFS_algorithm(interNet_i.getLastPWOfEdge, interNet_i.outPin)
20: end for
21: for each intraNet_i in intraNetVector do
22:    IDDFS_algorithm(intraNet_i.outPin, intraNet_i.inPin)
23: end for
24: endfunc
```

```
25: func findGlobalRoutingForRegion(netVector)
26: globalIntraSolutionFound = false
27: repeat
28:    globalIntraSolutionFound = fix a routing for each net in netVector
29:    if globalIntraSolutionFound == false then
30:       increment possible PIPs to be used by the nets of netVector
31:       initNetsIntraRoutingPhase(netVector)
32:    end if
33: until globalIntraSolutionFound == false
34: endfunc
```

Algorithm 9: Pseudocode of the Intra-Routing step.

6.4 DHHarMa Results

In this section, the presented DHHarMa routing algorithm is verified and tested. In particular, it is shown how significant is the impact of the homogeneity with respect to the maximum clock frequency on routed communication macros. Therefore, it is provided a detailed comparison of a homogeneous hard macro with an inhomogeneous hard macro of the same functionality (i.e., routed with the ISE Xilinx router). However, the inhomogeneous hard macro cannot be used for the target application.

First of all, a dedicated test benchmark flow is presented. Section 6.4.1 explains how the test has been divided into two parts: DHHarMa homogeneous routing and Xilinx Routing.

Section 6.4.2 presents the results of different communication designs for DPR systems. Eleven examples of communication infrastructures have been tested on the Virtex-4 devices and 7 Series devices. On the one hand, this flow shows the effectiveness of the approach; on the other hand, it provides a comparison to the commercial Xilinx commercial router. In the specific, it is highlighted how the homogeneity affects the final routed design, in term of resources utilized and timing delay.

Section 6.4.3 presents the communication infrastructure that has been generated by DHHarMa and utilized in the novel DRPM demonstrator (see Chapter 4). This communication infrastructure guarantees DPR and bitstream relocation on the DRPM. Finally, Section 6.4.4 presents another possible application of DHHarMa: a time-to-digital converter.

6.4.1 Routing Experiment Flow

The experiments focus on verifying routing results of DHHarMa compared to the standard Xilinx ISE router. Figure 6.12 shows the routing experiment flow. The HDL communication macro is first synthesized and converted into an XDL format, utilizing the standard Xilinx Front-end of DHHarMa (see Section 5.2.2). Then, the DHHarMa backend is executed, which provides at the end an XDL that is homogeneously placed and routed.

In addition to the standard DHHarMa flow, the Xilinx router is executed as well. The XDL design that is generated after the homogeneous placement phase of DHHarMa is converted into the NCD format; then, the standard Xilinx router is executed (triggered via FPGA Editor). In this way, it is possible to benchmark just the DHHarMa routing, excluding from the comparison the homogeneous packer and placer steps.

Finally, the two routed design are analyzed, extracting two base information: *PIPs utilization* and the *Max. Clock Speed* of the designs. The Max Clock Speed is then calculated utilizing the Xilinx Timing Analyzer (called TRCE) [129].

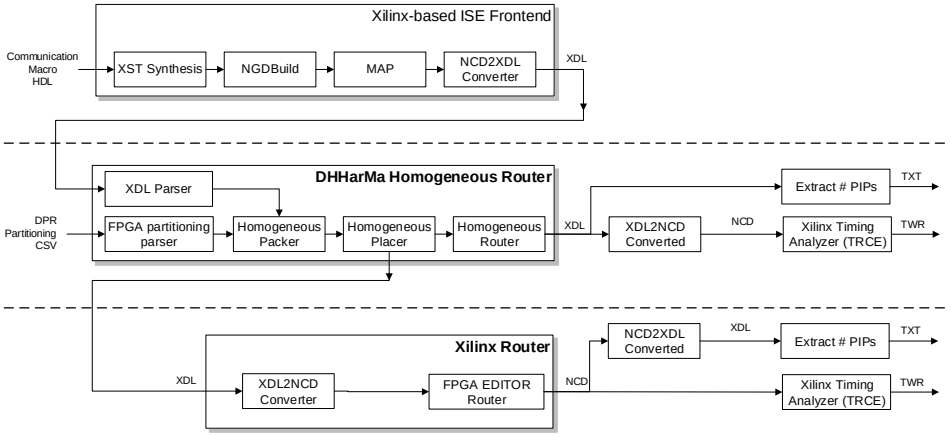


Figure 6.12: Routing Experiment Flow.

As it has presented in Section 6.3, the DHHarMa router finds a homogeneous routing, minimizing the number of resources utilized. Therefore, the Xilinx router is set with the resource optimization option, to have a fair comparison between two routing algorithms. Moreover, in both routers the Pin Swap option [128] is activated.

6.4.2 Routing comparison

Section 3.2.1 has presented that the Xilinx router cannot provide a routing of a design that supports module relocation. Researchers have been developed different routing algorithms for FPGA. The most important academic routing algorithm is the VPR [14]. To benchmark routing algorithms for FPGA, it has been created a dedicated CAD tool, VTR [92]. This tool allows comparing different algorithms on an ad-hoc FPGA design.

One of the main issues of benchmarking routing algorithms for FPGAs is the difficulties in comparing them with commercial ones. [51] is the only work that has created a bridge between VTR CAD tool and Xilinx tools. This has allowed comparing the benchmarks of the academic algorithms with the Xilinx routing algorithm. The main limitation of this work is the support of just one device: the Virtex-6 xc6vls240t. From this comparison, it has been shown an academic gap with respect to the commercial Xilinx tools. In the specific, academic routing algorithms have a gap of 15% compared to the Xilinx router.

Eleven examples of communication infrastructures have been tested on the Virtex-4 devices and 7 Series devices. The results are shown in Table 6.3, and Table 6.4. In this way, the DHHarMa router has been validated on different FPGA

Table 6.2: DHHarMa initialization step results for the tested communication infrastructures.

			Net connections			Homogen. Sets	
Design	# Reg	# XDL Nets	# Inter-nets	#Intra nets	Total	# HInter-NetSet	# HIn-traNet-Set
simpleBM 8Bit	2x2	193	96	160	256	32	64
simpleBM 8Bit	3x2	265	128	224	352	32	64
simpleBM 32Bit	2x2	769	384	640	1,024	128	256
simpleBM 32Bit	3x2	1,057	512	896	948	128	256
simpleBM 64Bit	2x2	1,537	768	1,280	2,048	256	512
FullSlaveBM 32Bit	2x2	1,920	1,082	1,550	2,632	380	602
FullSlaveBM 32Bit	3x3	3,805	2,412	3,130	5,542	380	602
FullSlaveBM 32Bit	5x2	4,296	2,792	3,446	6,238	380	602
FullMasterBM 32Bit	2x2	2,634	1,380	2,976	2,632	460	1,089
FullMasterBM 32Bit	3x3	5,465	2,990	6,121	9,111	460	1,089
FullMasterBM 32Bit	5x2	5,892	3,450	6,750	10,200	460	1,089

families, showing is compatibility with the latest devices that are supported by Xilinx ISE (i.e., 7 Series).

The eleven examples are based on three different communication macros architectures:

- *simple* communication macro (called Simple BM) which provides just data communication connections among static and PR Regions; the Simple BM can have a data-width of 8, 16, or 32 Bit.
- *Full Slave* communication macro: a wishbone communication infrastructure where the PR Regions are just slaves.
- *Full Master* communication macro: a wishbone communication infrastructure where the PR Regions are masters.

Table 6.2 shows details of the different designs utilized for the test. For each communication macro, it is indicated the number of PR Tiles (column # *Reg*) and the number of XDL nets (column # *XDL nets*). The rest of the information are the number of net connections and homogeneous nets that are generated in the initialization phase of DHHarMa. Columns # *Inter-nets* and # *Intra-nets* provide the number of net connections that DHHarMa created. As mention in Section 6.2.4, a net connection is a point to point connection between an outpin and an inpin.

Table 6.3: Routing benchmarks on the Virtex-4 family device. The simple example has been executed on the V4LX15 device. The FullSlave and the FullMaster examples have been executed on the V4FX100 device.

		DHHarMa Router			ISE Router			Comparison	
Design	# Reg	PIPs	Max clock [Mhz]	Time	PIPs	Max clock [Mhz]	Time	PIPs	Max clock [Mhz]
simple 8Bit	2x2	1,258	257	1m:44s	1,158	303	1s	+9%	-15%
simple 32Bit	2x2	5,279	230	4m:17s	4,947	280	1s	+7%	-18%
simple 64Bit	2x2	11,399	231	10m:44s	10,476	242	1s	+9%	-5%
simple 8Bit	3x2	1688	215	1m:33s	1,581	259	1s	+7%	-17%
simple 32Bit	3x2	7,077	198	5m:18s	7,049	238	1s	+17%	-1%
FullSlave 32Bit	2x2	16,032	112	1h:14m	14,442	128	7s	+11%	-13%
FullSlave 32Bit	3x3	33,678	56	1h:20m	30,123	121	7s	+12%	-54%
FullSlave 32Bit	5x2	41,357	51	1h:53m	34,873	65	8s	+19%	-22%
FullMaster 32Bit	2x2	26,715	148	46m:2s	21,001	186	8s	27%	-20%
FullMaster 32Bit	3x3	61,401	42	04h:11m	52,991	49	9s	+14%	-16%
FullMaster 32Bit	5x2	66,171	62	2h:41m	58,013	87	8s	+26%	-29%

Table 6.4: Routing benchmarks on the 7 Series family. The examples have been routed for the A7100T FPGA, except for the FullSlave 5x2 and the FullMaster 5x2 examples where the K325T device has been used.

		DHHarMa Router			ISE Router			Comparison	
Design	# Reg	PIPs	Max clock [Mhz]	Time	PIPs	Max clock [Mhz]	Time	PIPs	Max clock [Mhz]
simple 8Bit	2x2	1,684	259	34m:12s	1,626	240	6s	+4%	+8%
simple 32Bit	2x2	6,672	219	2h:28m	5,126	234	6s	+29%	-6%
simple 64Bit	2x2	14,408	194	1h:49m	11,791	202	6s	+22%	-4%
simple 8Bit	3x2	2,278	176	27m:4s	1,779	172	6s	+28%	+2%
simple 32Bit	3x2	9,457	169	1h:7m	7,238	155	6s	+31%	+9%
FullSlave 32Bit	2x2	18,972	82	03h:32m	16,282	91	8s	17%	-10%
FullSlave 32Bit	3x3	42,364	41	03h:45m	34,919	54	8s	21%	-24%
FullSlave 32Bit	5x2	51,197	52	05h:03m	40,906	71	13s	25%	-27%
FullMaster 32Bit	2x2	29,641	80	2h:45m	24,949	108	8s	+19%	-26%
FullMaster 32Bit	3x3	61,401	42	4h:11m	52,991	49	9s	+16%	-14%
FullMaster 32Bit	5x2	74,507	62	4h:23m	61,693	87	11s	+21%	-29%

The Homogeneous Sets column indicates how many homogeneous intra-net and inter-net sets are generated. As it is shown from this table, the designs have between 256 and 10,200 net connections. On the one hand, these values provide topological details about the different communication macros; on the other hand, they give an idea about the different complexity in the routing of these designs.

Table 6.3 and Table 6.4 show the results for the ISE router and the DHHarMa inhomogeneous router. The results show that the DHHarMa router utilizes between 1% and 31% more routing resources than the Xilinx Router.

In addition, the maximum clock frequency of the communication macros is between the range of +9% and -54% compared to the Xilinx router. This means that in some cases (i.e., in two 7 Series examples) the result of DHHarMa is even better than the Xilinx router in term of max frequency that can be reached by the design.

These results show that homogeneous DHHarMa router, despite the fact that is the only router providing a homogeneous router, introduces a delay of 15% with the standard Xilinx router. This is a remarkable result considering that the general, non-homogeneous, and academic VPR router introduces a delay of 15% [51].

Of course, the time required for the routing is much higher compared to the Xilinx ones, which is in all the tested cases less than 10 seconds. DHHarMa has routed the designs between 1m and 5h:03m. The high time required for the routing is given by the proposed IDDFS algorithm and the homogeneity check. IDDFS algorithm creates for each net connection, a set of possible paths that utilize the minimum number of PIPs. Instead, the homogeneity check verifies that the homogeneity constraints are respected within the homogeneous sets.

It is important to mention that the communication macros are intended to be computed once (as hard macros) and then be placed on a design. This means that, once that the communication macros are routed with DHHarMa, they can be reused as a placed-and-routed logical block without required extra computational time.

6.4.3 DRPM communication infrastructure

The communication infrastructure that allows supporting module relocation on the DRPM platform has been generated with DHHarMa. The communication macro connects all PR Regions in a homogeneous manner. This communication macro corresponds to the FullMasterBM 32 Bit routed on 5x2 regions (see Table 6.3).

Figure 6.13 shows an FPGA Editor screenshot of the DRPM communication infrastructure. As highlighted, the communication infrastructure spans the 10 PR Regions, which are all the same type. The FullMasterBM is a wishbone communication infrastructure that has 32 Bit data, 32 Bit addresses, 4 Byte enable signals, and 4 Bit auxiliary lines.

Table 6.5: Comparison of homogeneous and inhomogeneous communication macro of the DRPM.

		Homogeneous (DHHarMa PAR)					Inhomogeneous (Xilinx ISE PAR)				
Designs	# Reg.	max. Clock [MHz]	# LUTs	# Slices	# Nets	# PIPs	max. Clock [MHz]	# LUTs	# Slices	# Nets	# PIPs
Full 32Bit	5x2	62	4,445	2,362	5,892	66,171	121	4,445	3,993	14,514	83,444

Table 6.5 shows the results of this communication macro in the case of using the DHHarMa homogeneous PAR and the Xilinx PAR. It is important to mention that this comparison is different from the one presented in Section 6.4.2, where just the DHHarMa router and Xilinx router have been compared. In this case, it is benchmarked the overall PAR flow of DHHarMa and Xilinx.

The results show that first, the packer of DHHarMa reduces the number of slices by 41% (from 3,993 to 2,362). This is motivated with the fact that DHHarMa packer finds the best solution in term of occupied slices. On the contrary, Xilinx packer is not oriented to pack a design which is then reused in further design.

Then, The routing results show that DHHarMa router uses less routing resources. This is due to the fact that the number of slices is less than the Xilinx ISE PAR results. On the contrary, the maximum clock design for the DHHarMa PAR design and the Xilinx ISE PAR are 64 MHz and 121 Mhz, respectively. This means that DHHarMa PAR has a maximum clock delay that is 47% less than the Xilinx PAR. Again, this result is motivated by the fact that the Xilinx ISE PAR can optimize in a better way the delays of a path, since that it is aware of the delay of the connections of the global routing structure.

6.4.4 Further Applications of the Homogeneous Router

In Figure 6.14 a schematic of a delay line is shown, which is commonly implemented in FPGA-based time-to-digital converters. The delay line is used to measure the time between two pulses by utilizing the delay of the carry chains within the FPGA fabric.

For this example, it has been generated a corresponding homogeneous hard macro with 40 small regions of the same type, each containing a single CLB only. Figure 6.15 shows a homogeneous hard macro generated by DHHarMa and an inhomogeneous hard macro generated using the place and route tools from the Xilinx tool chain. The results are shown in Table 6.6.

When comparing the hard macros it can be realized that the homogeneous hard macro uses identical routing resources for each element, while the inhomogeneous

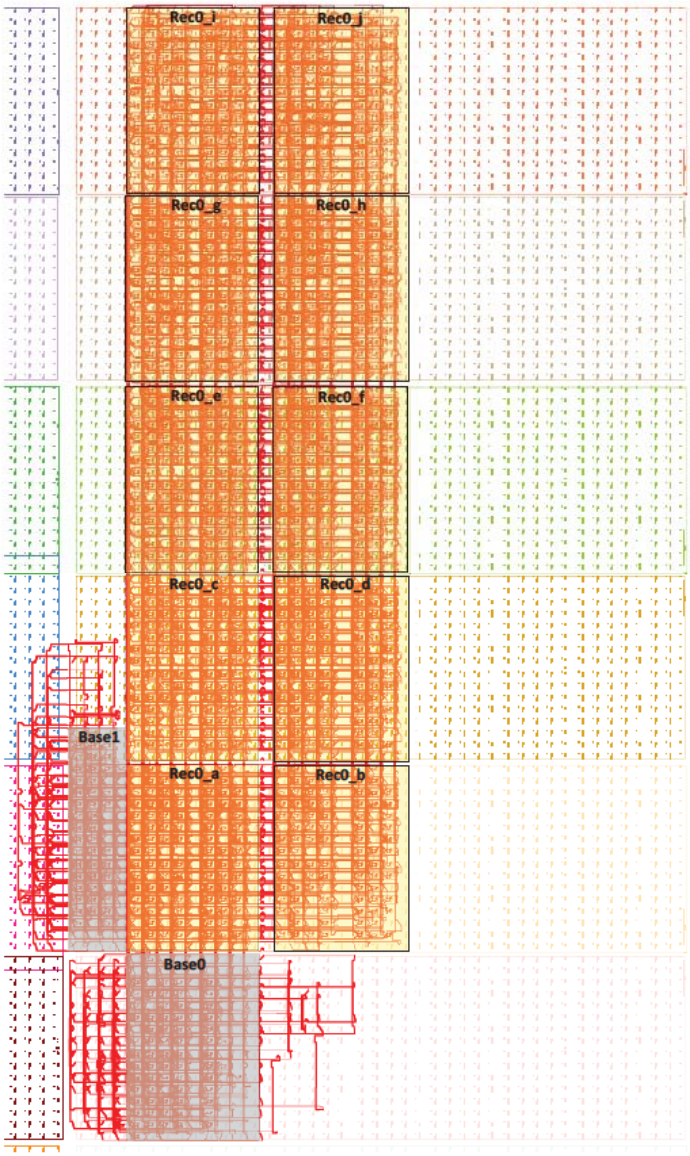


Figure 6.13: Homogeneous communication macro utilized in the V4FX100 FPGA of the DRPM system. The communication macro is a FullMaster 32bit Macro with ten regions (5x2). All the PR Region are of the same type.

Table 6.6: Comparison of homogeneous and inhomogeneous Delay Line design, executed on a V4LX15 [184].

		Homogeneous Hard Macro						Inhomogeneous Hard Macro (Xilinx ISE)					
Designs	# Re-gion	max. Clock [MHz]	# FFs	# LUTs	# Slices	# Nets	# PIPs	max. Clock [MHz]	# FFs	# LUTs	# Slices	# Nets	# PIPs
TDC Delay Line 160Bit	40	750	320	160	80	526	863	653	320	240	81	529	1731

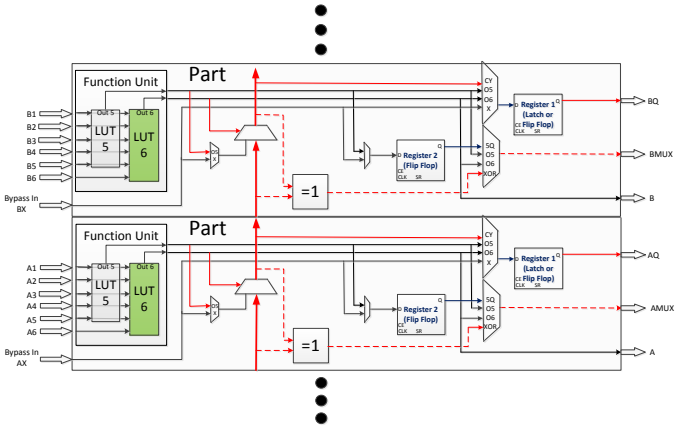
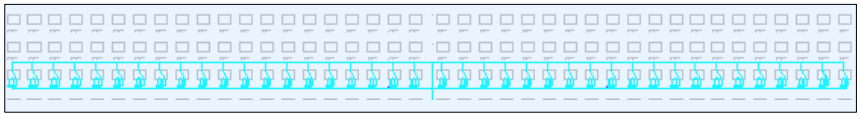
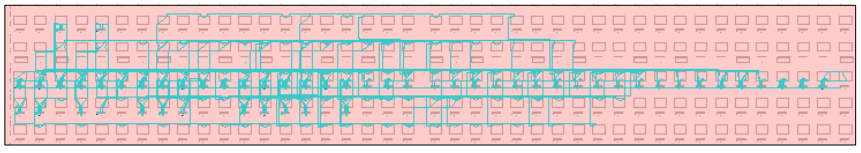


Figure 6.14: Schematic of the delay line example [184].



(a) DHHarMa Placer&Router (Homogeneous).



(b) Xilinx Placer&Router (Inhomogeneous).

Figure 6.15: Comparison of a homogeneous and an inhomogeneous hard macro for the example of a delay line circuit with 40 regions [184].

hard macro does not. The homogeneous hard macro requires 160 Look-Up-Tables (LUTs), 526 nets, and 863 PIPs, where the maximum clock frequency is 750 MHz. The inhomogeneous hard macro requires 240 LUTs, 529 nets, and 1731 PIPs with a maximum frequency of 652.74 MHz.

The 33.3 % less number of LUTs shows that the homogeneous packer of DHHarMa performs even better than the one provided by Xilinx. The 50.2 % less number of PIPs used in the homogeneous macro indicates that fewer lines are used, which reflects the result shown in Figure 6.15. The main reason for this is that the DHHarMa placer places interconnected instances as close as possible, which is not the main objective of the Xilinx placer. For the delay line example, the hard macro generated by DHHarMa outperforms the inhomogeneous one.

6.5 Summary

This chapter has introduced a novel homogeneous router for FPGA: the DHHarMa router. As part of the INDRA flow (see Chapter 5), it guarantees homogeneous routing of a specific design (e.g., DPR, time-to-digital converter). Details about the DHHarMa router, as well as details about the overall DHHarMa flow have been presented in [184].

The DHHarMa router introduces novel concepts and functionalities that leak in existing commercial and academic tools. The pinWires grouping algorithm allows categorizing a general XDL design to apply a dedicated homogeneous algorithm.

Details of the homogeneous algorithm have been provided explaining how the homogeneous solution is generated in all the steps of the routing. To achieve an optimal solution, an Iterative Deepening Depth-First Search algorithm (IDDFS) algorithm has been utilized, allowing the creation of sets of solutions for each net to be routed.

Results have shown that the homogeneous routing cost compared to best available routing algorithm for Xilinx FPGA (ISE router) is in average the 16% of routing resources (i.e., PIPs). Regarding the maximum clock speed of the design, the DHHarMa router suffers a reduction in an average of 15%. These results validated the presented approach and they show that the drawback in term of routing resources utilized, as well as the delay of the design, are compensated with a homogeneous design that can be used to support DPR and bitstream relocation.

One of the uses of the DHHarMa router is the generation of the DRPM homogeneous communication infrastructure. This communication infrastructure allows having 10 PR Region, where a module can be relocated in one of them.

7 OLT(RE)^2

Chapter 6 has presented how a DPR scenario can be generated, introducing the INDRA 2.0 flow. In particular, the chapter focuses on tools that allow reaching a full-homogeneous design, thanks to a static rerouter (PSRerouter) and a homogeneous communication infrastructure generator (DHHarMa). Then, DPR has been only considered for reconfiguring a certain design functionality, i.e., changing a certain PR Module in the system. Instead, this chapter considers the testing of the reconfigurable area in DPR systems.

Therefore, OLT(RE)^2 flow is presented: an on-line on-demand approach to testing permanent faults induced by radiation in reconfigurable systems used in space missions. The proposed approach relies on a test circuit and custom place-and-route algorithms. OLT(RE)^2 exploits DPR offered by today's SRAM-based FPGAs to place the test circuits at run-time.

The goal of OLT(RE)^2 is to test unprogrammed areas of the FPGA before using them, thus preventing functional modules of the reconfigurable system to be placed in areas with faulty resources. It is explained how it is possible to generate, place and route the test circuits needed to detect physical wires and programmable interconnection points of an arbitrarily large region of the FPGA in a reasonable time.

Section 7.1 introduces the overall structure of the flow. Two sub-flows are presented: Design-time flow and Run-time flow. Section 7.2 presents the testing circuit architecture that is utilized by the flow. In particular, it is explained how the nets of the circuits can verify that routing resources of the FPGA are permanent fault free. Section 7.3 describes how the routing infrastructure of the FPGA has been modeled in an oriented graph. This graph is used by OLT(RE)^2 to reach a full coverage of routing resources testing. Section 7.4 presents details regarding the Routing Resources Analyzer (RRA) tool, which is able to categorize the routing resources of the FPGAs according to their testability.

Section 7.5 explains in detail the core part of OLT(RE)^2: the U-TURN algorithm. This algorithm can place and route the testing circuit to reach an extensive test of the FPGA routing resources. Section 7.6 presents results of the OLT(RE)^2 flow: validation of the testing circuits, design-time results and run-time results. Section 7.7 summarizes the presented flow.

7.1 Flow Structure

The overall goal of OLT(RE)^2 is to support on-line on-demand testing of dynamically reconfigurable systems based on SRAM-based FPGAs. More in detail, OLT(RE)^2 exploits DPR capabilities offered by modern SRAM-based FPGA devices to place ad-hoc designed, placed-and-routed test circuits on the reconfigurable areas of the system.

These test circuits are meant to be placed and run before a functional module of the reconfigurable system has to be placed, to verify whether the area where the functional module are either placed is faulty or fault-free, thus avoiding to place the functional modules on faulty resources of the device. This increases the reliability of the reconfigurable system.

From a high-level point of view, the OLT(RE)^2 approach can be summarized in the following steps:

- *A test circuit is designed*, composed of a Test Pattern Generator (TPG), an Output Response Analyzer (ORA), and several Net Under Tests (NUTs).
- *The test circuit is placed-and-routed multiple times* (thus each placed-and-routed test circuit tests a subset of the routing resources of the Area Under Tests (AUT)) in such a way as to maximize the amount of resources under test while minimizing the number of test circuits (and thus the number of reconfigurations required at run-time);
- *Place a test circuit at a time on the AUT*. Once the test circuit is placed, there is no interaction between it and the IOBs. On the contrary, the test circuits internally generate test stimuli that are propagated to the resources under test and then received and analyzed by an ORA that determines whether the resources under test are faulty or not.
- Finally, *the result of the test is stored in DRAMs*, which are accessed through memory readback. By knowing the resources occupied by the failed test circuit, it is also possible to identify a subset of possible faulty resources, thus performing a coarse-grained fault diagnosis.

The proposed testing approach focuses on most of the routing resources available in an FPGA device; the only routing resources that are currently not supported by OLT(RE)^2 are the routing resources connected to DSPs, BRAMs, carry chains, and clock distribution resources.

Finally, OLT(RE)^2 is suitable for a wide range of Xilinx FPGA families, including Spartan-6, Virtex-4, Virtex-5, and Virtex-6. Due to its modular structure, OLT(RE)^2 can be extended to work with other Xilinx FPGA families. Additionally, the methodology can be utilized for FPGAs from other vendors.

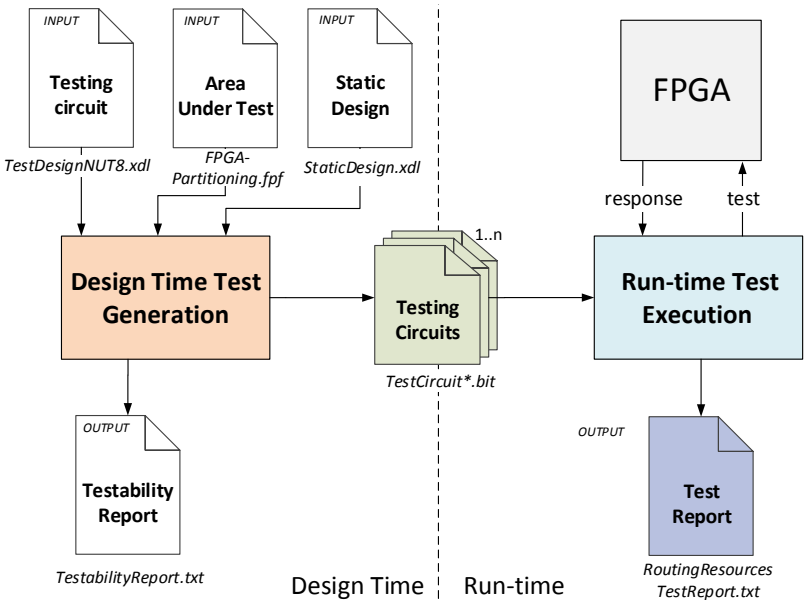


Figure 7.1: The overall OLT(RE)^2 CAD flow [178].

7.1.1 The OLT(RE)^2 CAD Flow

OLT(RE)^2 is composed of a set of C++ tools that are integrated into the standard Xilinx CAD flow, automating all the activities that need to be carried out to implement the proposed test strategy. The OLT(RE)^2 CAD flow (see Figure 7.1) can be divided into two parts: the *design-time test generation* and the *run-time test execution* sub-flows.

The design-time test generation is intended to be performed at design-time on a ground machine to generate all the test circuits. The run-time test execution is meant to be executed at run-time, on the reconfigurable system itself, before a module reconfiguration.

The flow utilizes the XDL intermediate language (see Section 2.3.3). Moreover, it relies on the typical PR flow of Xilinx; it takes a testing circuit design (mapped with the Xilinx tool), places and routes it, and generates a certain number of testing circuits (in XDL format). The FPGA's routing resources are accessible utilizing the DXF database (presented in Section 5.2.1 and Section 5.3.4). Finally, the testing circuits are converted to the Xilinx NCD format and can be integrated into the normal Xilinx flow.

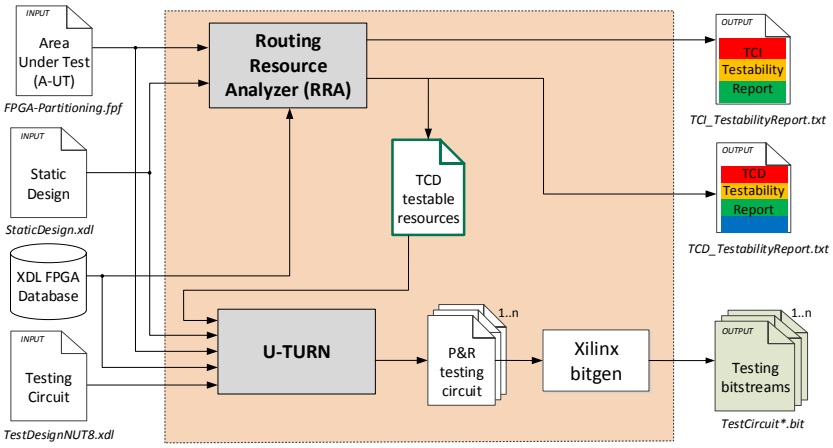


Figure 7.2: The design-time test generation sub-flow [178].

7.1.2 Design-time Test Generation Sub-flow

Figure 7.2 depicts the design-time test generation sub-flow, which is the first sub-flow of the overall CAD flow (depicted in Figure 7.1). The input files consist of a specification of the partitioning of the system (a .fpf file) that specifies the AUT, the design of the static region and the test circuit design (a .xdl file containing the specification of the test circuit described in Section 7.2).

The first tool, called *Routing Resources Analyzer (RRA)*, categorizes the routing resources (i.e., physical wires and PIPs), for each given AUT. Moreover, by reading the design of the static region, the RRA detects and excludes all the routing resources in the reconfigurable region that belong to the static region. In this way, the U-TURN algorithm can focus just on the testable resources, thus saving time avoiding testing unsupported ones Section 7.4. At the end of this process, a detailed report of the testability of the design is created, which lists all routing resources with their assigned categorization. The RRA step is presented in detail in Section 7.4.

The RRA's output contains crucial information for *U-TURN* phase. The execution of U-TURN produces multiple placed-and-routed test circuits, which cover all testable routing resources for the AUTs. To carry out this task, three input files are needed: the list of testable resources, the partitioning file, and the test circuit specification. The resulting placed-and-routed test circuits are specified in XDL and subsequently translated into a Xilinx design file (NCD) or optionally into a Xilinx hard macro file (NMC) and then into a (partial) bitstream with the standard Xilinx tools (xdl and bitgen). The U-TURN step is presented in detail in Section 7.5.

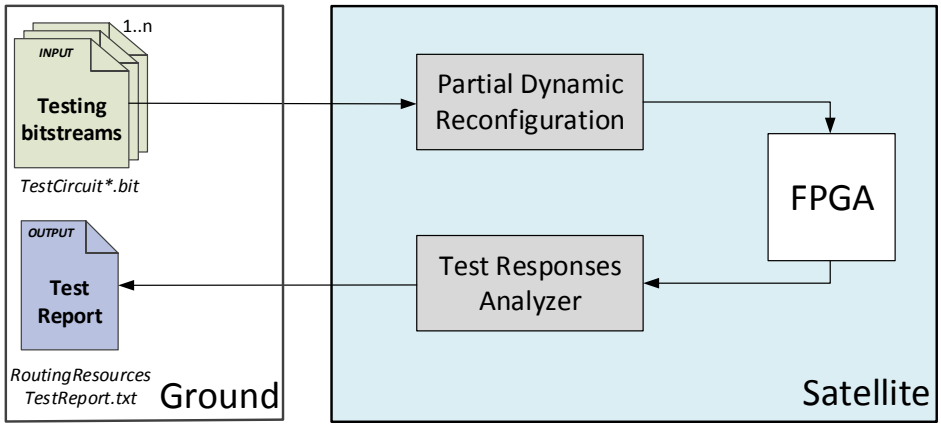


Figure 7.3: The run-time test execution sub-flow [178].

7.1.3 Run-time Test Execution Sub-flow

Figure 7.3 depicts the run-time test execution sub-flow, which is the second sub-flow of the overall CAD flow (depicted in Figure 7.1). This sub-flow is executed as soon as a reconfiguration of the system is needed. The test circuits (either stored in a dedicated fault tolerant persistent memory or received "just in time" from the ground station) are exhaustively placed on the AUT one at a time and then the test is executed. After each test circuit has been properly placed and run, the ORA stores the result of the test execution in a dedicated distributed memory. These results are read back by the *Test Responses Analyzer* that cumulatively creates the overall test report that is sent to the ground.

7.2 Circuits for Testing of Permanent Faults

The test circuit on which OLT(RE)^2 relies is composed of a TPG and an ORA. Figure 7.4 provides a high-level representation of the test circuit. All the connections between the outputs of the TPG and the inputs of the ORA represent the resources under test. More specifically, all the physical wires and all the PIPs connecting TPG and ORA represent the *Net Under Test (NUT)*. In the current version of the test circuit (originally designed for the Xilinx Virtex-4 family of devices, that provides 4-input look-up tables) the test circuit has 8 NUTs.

The goals of the test circuit design are the following:

- Detect 100 % of the faults in the routing resources of the AUT.

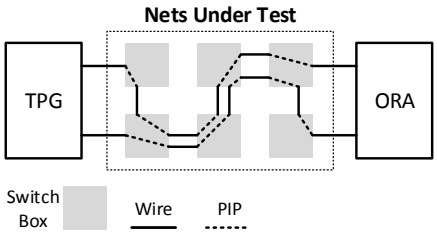


Figure 7.4: High-level representation of a test circuit [178].

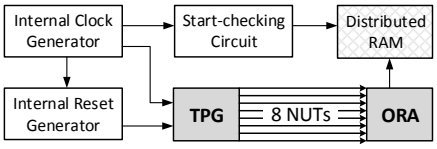


Figure 7.5: Structure of the 8-NUT test circuit [178].

- Occupy the smallest possible amount of resources of the FPGA to be applicable also when a large part of the resources of the FPGA is already occupied.
- Be as fast as possible, in order not to interfere with the normal functioning of the system.

The functional block-level structure of the designed test circuit is depicted in Figure 7.5: the clock and reset signals that are fed to the test circuit are generated by dedicated modules, to make the testing structure entirely independent of the region of the FPGA on which it is placed. No external clock and reset signals are used, and no input/output buffers are employed. Therefore, it is possible to change the NUT by only re-placing the test circuit, without any change in the logic.

The result of the test is stored in dedicated LUTs configured as distributed RAM, whose content can be read-back at the end of the testing activity. A ring oscillator is used to generate the internal clock signal. A parametric n-bit shift register pre-loaded with n 1s is used to generate the reset signal.

The start-checking circuit has been added to the test circuit to verify whether the test circuit has been correctly configured and the test correctly started. Some faults may prevent the test to start at all: in such a case, although a fault occurred, the ORA would not be able to detect any misbehavior. The testing circuit utilized by OLT(RE)^2 has been extensively described in [186].

7.2.1 The 8-NUT Hard-Macro

The 8-NUT cross-coupled parity-based hard macro has been designed to test the 8 input nets of the LUTs of one Virtex-4 slice.

In more detail, the TPG is composed of two 2-bit counters, each placed into two 4-input LUTs and using two flip-flops (thus, each counter occupies only one slice). One of the two counters is an *up counter*, i.e., counts from 0 up to 3, and produces an even parity bit; the other counter is a *down counter*, i.e., counts from 3 down to 0, and produces an odd parity bit.

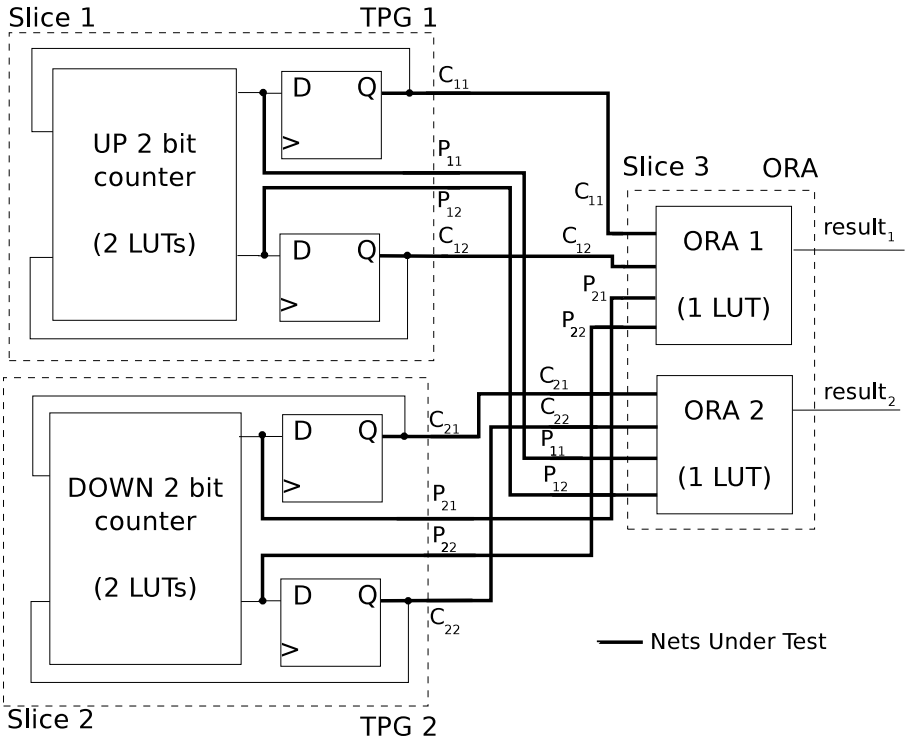


Figure 7.6: The detailed structure of the 8-NUT testing circuit [178].

The ORA is composed of two analyzers, one for each counter, each occupying one 4-input LUT (thus, the whole ORA occupies only one slice). By comparing the received 2-bit state with the received parity bit, each analyzer can determine whether a fault occurred in its input wires or the associated counter.

It is important to mention that the correctness of the used DRAM has to be verified before placing the testing circuit by utilizing a specific pattern [172]. The detailed structure of the testing circuit is depicted in Figure 7.6. [186] provides more information on the testing circuit implementation.

7.2.2 Routing Faults Test Principles

In the following concepts related to fault detection in the routing resources of an FPGA that can be found in the literature are provided [111]. The switch matrix shown in Figure 7.7 is utilized for the concepts' explanation; the switch matrix has

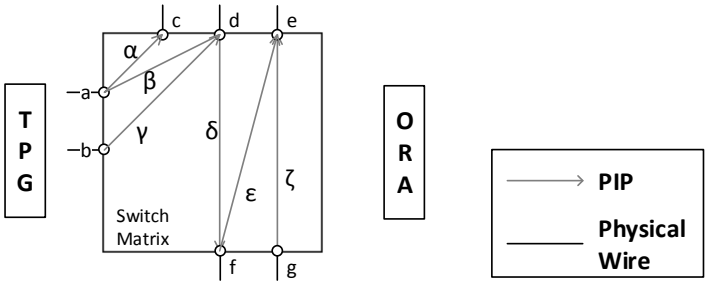


Figure 7.7: An example switch matrix [178].

6 PIPs (namely α, β, γ, δ, ϵ, and ζ) and 7 PW (namely *a*, *b*, *c*, *d*, *e*, *f*, and *g*). For the fault detection, it is called *PIP Under Test (PIP-UT)* the PIP under test and *PW-UT* the physical wire under test.

The principles discuss in the following have been integrated into U-TURN, used to place-and-route the test circuit. Section 2.4.4 presents the faults that can occur in FPGA's routing resources: stuck-at-0, stuck-at-1, stuck-on, and stuck-off.

Stuck-at and Stuck-off Fault Detection

Figure 7.8 shows how the stuck-at fault's detection on physical wires and stuck-off fault's detection on PIPs is provided. The figure represents a net that is routed from the TPG to the ORA, which uses two PWs and one PIP. As it is highlighted, the N-UT can detect stuck-at faults of the utilized PWs and stuck-on faults on the utilized PIPs. In other words, it is enough to use a routing resource and to write on it both a logic 1 and a logic 0 to be able to detect stuck-at 0/1 and stuck-off faults. Therefore, in Figure 7.8 it can be seen that the N-UT crosses PW*a* and PW*d* and PIPβ. Therefore, the N-UT detects stuck-at 0/1 faults on PW*a* and PW*d* and stuck-off faults on PIPβ.

Stuck-on Fault Detection

Figure 7.9 shows how the stuck-on fault's detection on PIPs is provided; when the target is on stuck-on faults on PIPs, two nets under tests are needed, each connected to one of the two ends of the PIP-UT (as shown in Figure 7.9, where the target is a stuck-on on PIPβ).

It is worth noting that in this case none of the two N-UTs actually uses the PIP-UT. The idea behind this is that a stuck-on fault on a PIP may cause a short between two nets. Thus, by routing two N-UTs through the two ends of the PIP-UT, it can be checked whether the two nets are shorted (and thus the PIP-UT is

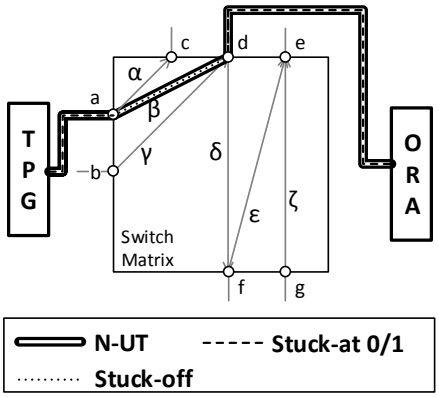


Figure 7.8: An example routing between TPG and ORA for stuck-at/off testing [178].

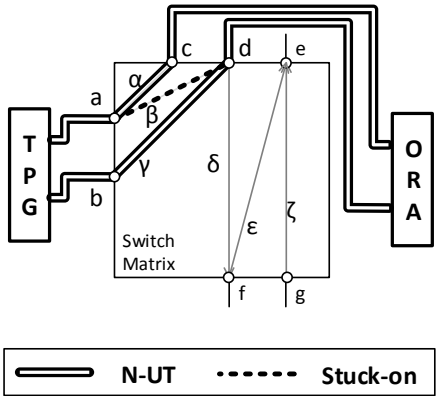


Figure 7.9: An example routing between TPG and ORA for stuck-on testing [178].

stuck-on affected) or not. Apart from the stuck-on fault on PIPβ, the two N-UTs also test stuck-at 0/1 faults on PW*a*, PW*b*, PW*c*, and PW*d* and stuck-off faults on PIPα and PIPγ.

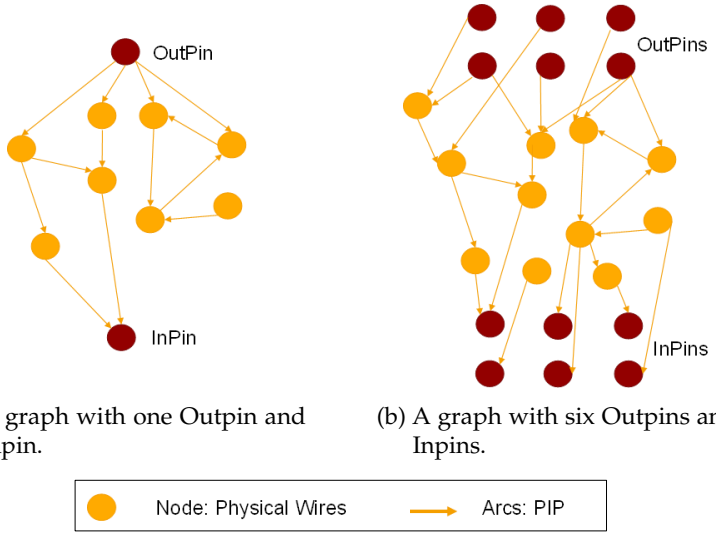
7.3 Graph Model of FPGA

This section describes the permanent faults models, their effects and how faults are analyzed to perform a real test. In an FPGA, a net represents the connection between two or more components and it is composed of physical wires and PIPs. The fault model considered is presented in Section 7.2.2.

A graph model is introduced to generalize the testing coverage problem of OLT(RE)^2. Figure 7.10 shows how the FPGAs routing resources are modeled in a *cyclic oriented graph*. The PWs are considered as *nodes* and the PIPs as *arcs*. This kind of representation allows considering the testing coverage problem as a graph coverage problem.

The Figure 7.10 represents two different kinds of graphs: a graph considering just one OutPin and InPin (Figure 7.10a) and a graph that considers six OutPins and six InPins (Figure 7.10b).

In the following, an explanation of how a path on the cyclic oriented graph representing a routing resource test is shown.



(a) A graph with one Outpin and Inpin.

(b) A graph with six Outpins and Inpins.

Figure 7.10: Cyclic Oriented Graph of the FPGA routing resources.

7.3.1 Stuck-at Coverage

The stuck-at fault model is explained in Section 7.2.2. A stuck-at-0 fault on a physical wire forces the logic value of the net to '0'. It is possible to check if a physical wire is stuck-at-0 free by using it in a NUT while the TPG sends a '1' over it. If the ORA receives a '0', it means that a fault occurred. Therefore, if the test is passed all physical wires that belong to the NUT are stuck-at-0 free. A similar approach can be applied in the case of stuck-at-1.

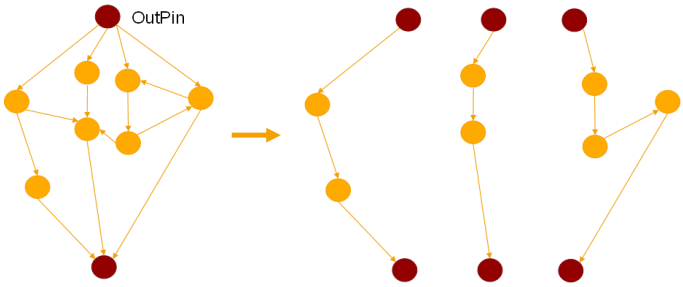


Figure 7.11: Testing stuck-at-0/1 in the graph representation.

Utilizing the presented graph implementation, the verification that all physical wires of an AUT are stuck-at-0/1 free is considered as resolving a *nodes-covering problem*; it is necessary to discover several paths such that all nodes are crossed at least once by a NUT.

For simplicity, it is considered the graph represented in Figure 7.10, where just one NUT can be routed; the graph has only one OutPin and one InPin. The NUT has to be routed among these nodes. Figure 7.11 highlights three different paths that cover all nodes of the graph.

7.3.2 Stuck-off Coverage

The stuck-off fault model is explained in Section 7.2.2. A PIP affected by this kind of fault is always deactivated, and the two physical wires paired by the PIP are unconnected. Therefore, if a path uses a stuck-off PIP, its logic value is unknown. It is possible to check if a PIP is stuck-off free by using it in a NUT, while the TPG sends a stimulus over it. In the case that a fault occurred, the ORA receives wrong values. If the test is passed, it is possible to assert that all PIPs that belong to the NUT are stuck-off-free.

Utilizing the presented graph implementation, the verification that all PIPs of the AUT are stuck-off-free, it is necessary to resolve an *edges-covering problem*.

In the example depicted in Figure 7.11, three solutions are needed to visit all the nodes to the graph; apart from the nodes, some PIPs are visited as well. Therefore, the NUTs verifies that the PIPs are stuck-off-free as well. Figure 7.12 highlights in bold the PIPs that are not verified by the three solutions of Figure 7.11; where the stuck-at fault has been targeted. It is needed to add some NUTs to the previous three tests, such that all edges are crossed at least once.

Figure 7.12 shows the two paths of the NUT that can be added to cover all edges. So, to cover all the edges of the graph of Figure 7.10, five different testing circuits are needed.

7.3.3 Stuck-on Coverage

The stuck-on fault model is explained in Section 7.2.2. A PIP affected by this kind of fault is always deactivated and creates a permanent connection between two physical wires. Then, this PIP can create an antenna or short two nets of the design.

The consequence of a stuck-on on a design can be a short, either a wired-AND short or a wired-OR short. It is possible to check if a PIP is stuck-on free using two NUTs that can be shorted by it (as explained in Section 7.2.2). If both TPGs of the NUTs send uncorrelated stimuli over their own NUT and the related ORA receives a different logic value, then a fault occurs. Therefore, if the test passes, all PIPs that can create a short between the tested NUTs are stuck-on-free.

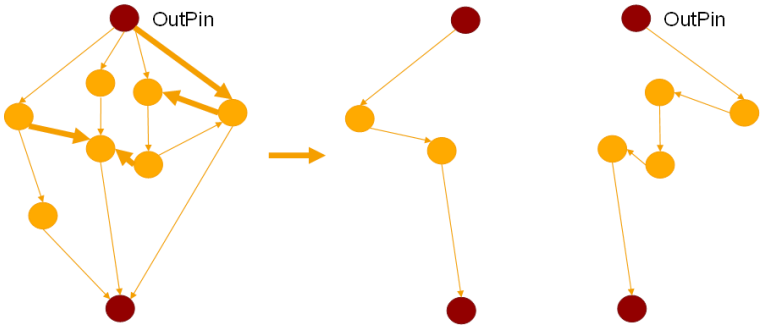


Figure 7.12: Testing stuck-off in the graph representation.

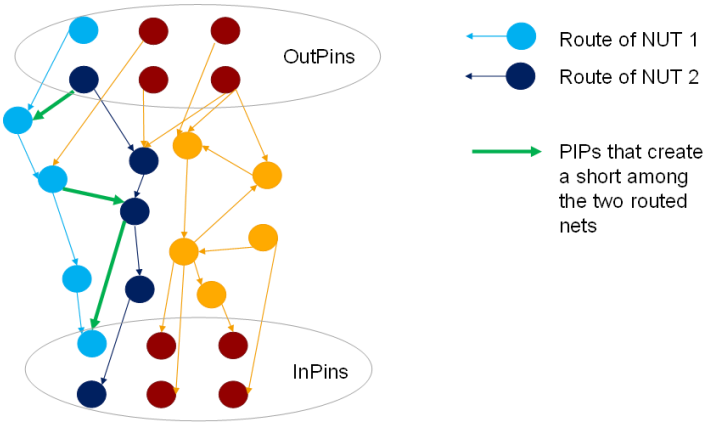


Figure 7.13: Testing stuck-off in the graph representation.

For this kind of fault at least two NUTs are needed. For example, Figure 7.13 shows the routing resources graph model, in the case that six nets under test are routed simultaneously; six outPins and six inPins are present. It is worth noting that these nets must be independent thus no resources can be shared between them (i.e. no conflict is created). For this example, just two nets have been routed; Figure 7.13 shows in green the PIPs that can create a short fault between 2 different routed NUTs (i.e., $NUT1$ in light blue and $NUT2$ in dark blue). If this test is passed, it can be assert that these PIPs are stuck-on free.

The stuck-on test verification is the most complex operation in term of time and number of testing circuits needed to execute the test.

7.4 Routing Resources Analyzer

The aim of OLT(RE)^2 is to find a flow capable of testing on demand routing resources of a specific FPGA's region. In addition, the test is oriented to critical systems (e.g., aerospace and satellite applications). The idea of the flow is to provide an online testing, within a partially reconfigurable scenario, where the FPGA is divided into one or more reconfigurable regions, which can be reconfigured at run-time.

However, with this kind of partitioning, there are routing resources that span more than one reconfigurable region. These kinds of resources can not be utilized because a misbehavior of the test can propagate an error in critical parts of the system, affecting its behavior.

Therefore, in order to execute a test that does not affect the behavior of the other regions, a static analysis of the routing resource has been developed: the Routing Resources Analyzer (RRA). All the routing resources (i.e., PIP and PWs) of the AUT are categorized according to their position and their testability.

The RRA main task is to provide to the U-TURN step, a list of resources that are "really" testable with the presented approach. As presented in Section 7.1.2, the RRA takes in input the FPGA partitioning and the list of partitions to be tested. During this phase, resources are marked depending on their testability.

In Section 7.4.1 the testability categories and the RRA are presented. Then, the steps of the RRA are explained in Section 7.4.2, focusing on how the routing resources are categorized "step-by-step". Finally, Section 7.4.5 presents how the RRA generates different outputs, which provide the user the testability information of the routing resources.

7.4.1 Testability of the Routing Resources

The routing resources are categorized according to their testability. In the following, different categories are presented. It is important to mention that the categorization has been made with respect to the dependency on the presented testing circuit. Therefore, the PW categories can be either Testing Circuit Independent (TCI) or Testing Circuit Dependent (TCD) (see Section 7.4.2). The categorization is divided into *physical wires testability* and *PIP testability*.

Physical Wire Testability

The presented fault detection mechanism can detect if a stuck-at fault is present on a PW. However, some PWs cannot be visited; then, the RRA marks them to give to the U-TURN algorithm the right set of PWs to be tested.

A PW can be categorized into the following four categories:

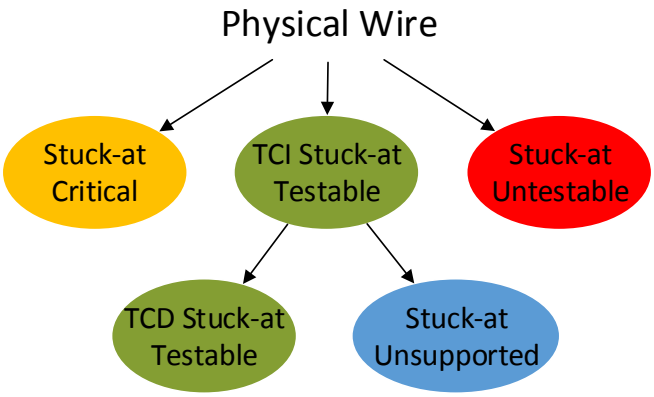


Figure 7.14: PW Testability Categories.

- *Untestable*: a physical wire is marked as untestable if it cannot be used by a net that is routed internally to the reconfigurable region.
- *Critical*: a resource is marked as critical if it can be used internally into the reconfigurable region, but its activation can affect the behavior of the other region of the FPGA.
- *Testing Circuit Independent (TCI) Testable*: a resource is marked as testable if it is possible to find a path for a NUT that utilizes the considered PW.

It is important to mention that a PW is categorized just with one of these three categories, as depicted in Figure 7.14. This categorization is provided by the TCI analysis (explained in 7.4.3).

Beside this first categorization, the TCI Testable PWs can be divided in:

- *Testing Circuit Dependent (TCD) Unsupported*: a resource is marked as unsupported if it is TCI stuck-at testable but it cannot be utilized by the considered testing circuits.
- *Testing Circuit Dependent (TCD) Testable*: a resource is marked as TCD testable if it is possible to find a path for a NUT that utilizes the considered PW, with the utilized testing circuit.

This second categorization is necessary to verify which routing resources are testable according to the presented testing circuit. In the specific, this categorization is executed by the TCD analysis (explained in Section 7.4.4).

PIP Testability

The presented fault detection method can detect stuck-on and stuck-off faults. The possible PIP categories are represented in Figure 7.15. With Respect to stuck-off faults the PIPs resources can be categorized in three different ways:

- *Stuck-off Untestable*: the PIP cannot be used by a net that is routed internally to the reconfigurable region.
- *Stuck-off Critical*: the PIP can be used internally into the reconfigurable region, but its activation can affect the behavior of the other region of the FPGA.
- *Stuck-off Testable*: The PIP is marked as testable if it is possible to find a path for a NUT that utilizes it.

It is important to mention that a PIP is categorized just with one of the three stuck-off categories, as depicted in Figure 7.15.

Beside this first categorization, the TCI Testable PIPs can be divided into:

- *Testing Circuit Dependent (TCD) stuck-off Unsupported*: a resource is marked as unsupported if it is not untestable or critical but it cannot be utilized by the considered testing circuits.
- *Testing Circuit Dependent (TCD) stuck-off Testable*: a resource is marked as TCD testable, if it is possible to find a path for a NUT that utilizes the considered PIP, with the utilized testing circuit.

With Respect to the stuck-on fault detection, the PIP can be categorized in:

- *TCI stuck-on Testable*: it is possible to route simultaneously two NUTs of the same testing circuit that utilize the two PW connected to the considered PW wire, but without using the PIP itself. The stuck-on detection is described in detail in Section 7.2.2.
- *stuck-on Untestable*: if it is not stuck-on Testable.

In addition, as for the stuck-off case, the TCI stuck-on Testable PIPs can be divided into:

- *TCD stuck-on Testable*: the PIP stuck-on fault can be verified with the utilized testing circuit.
- *TCD stuck-on Unsupported*: the PIP stuck-on fault cannot be verified with the utilized testing circuit.

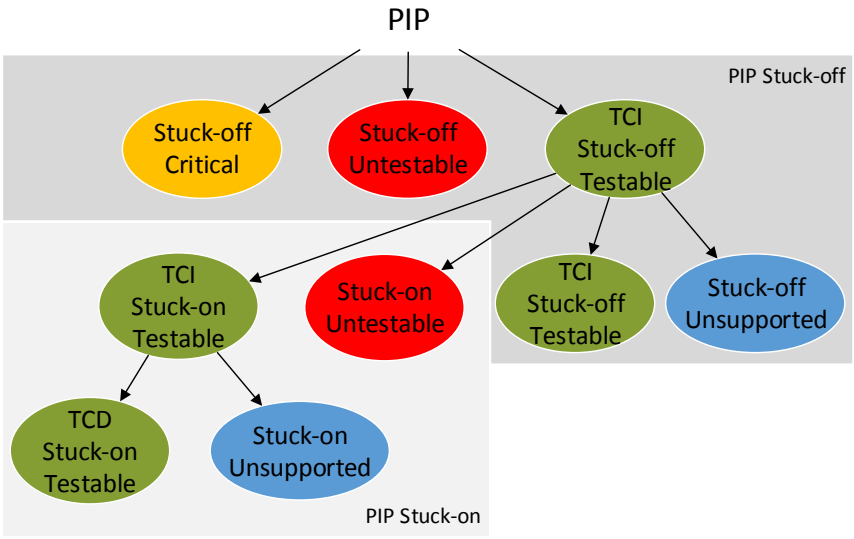


Figure 7.15: PIP Testability Categories.

7.4.2 Routing Resources Analyzer Flow

This section explains how the PWs and PIPs are analyzed and how for each routing component a testability category is given. Figure 7.16 shows how the RRA categorizes the routing resources. The flow is divided into two main parts, which are executed sequentially: the *TCI Analysis* and the *TCD analysis*. In the following, these two analyses and their sub-phases are described.

7.4.3 Testing Circuit Independent (TCI) Analysis

In this part, the RRA categorizes the routing resources without considering the target testing circuit of the testing approach. Therefore, the routing resources within one AUT are just considered according to their position.

This analysis is divided into three phases: *Initialization Phase*, *Dependency Phase*, and *stuck-on Testability Phase*.

Initialization Phase

In this phase, the testability of the PWs and PIPs is initialized according to the AUT considered. More in detail, PWs are initialized according to their InWire and the OutWire position.

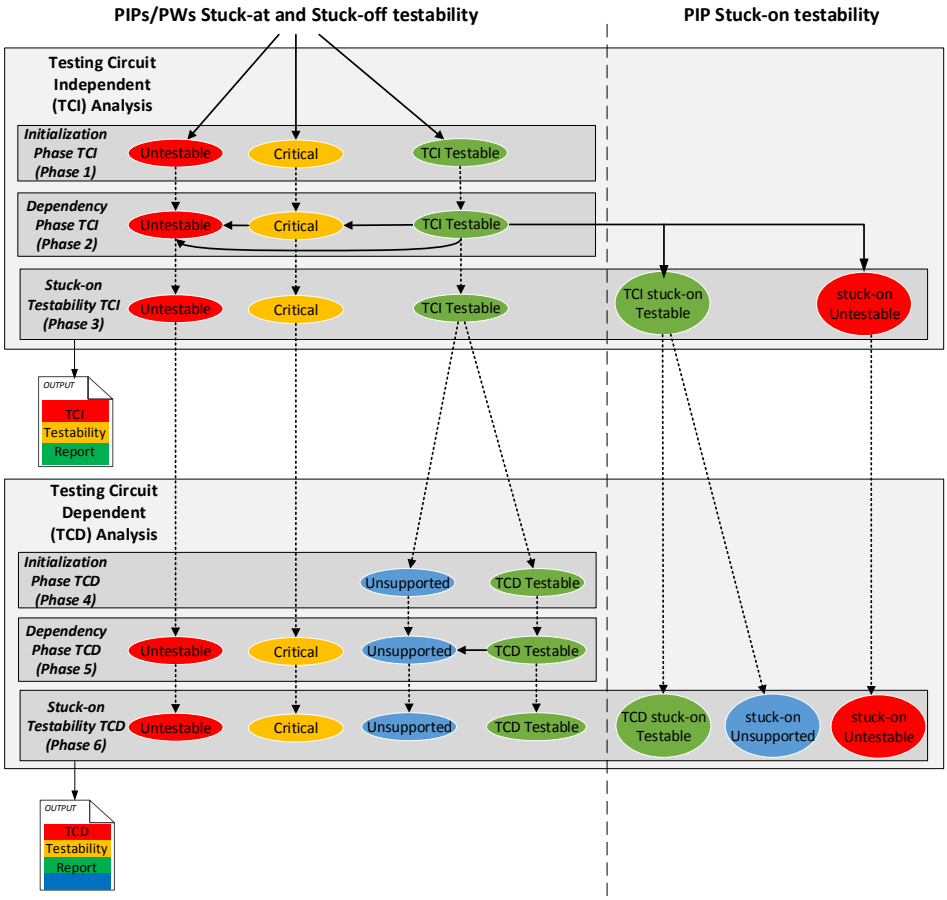


Figure 7.16: How the routing resources area categorized in the various phases of the RRA.

Figure 7.17 shows a simplified version of an FPGA that is partitioned into two different areas: one static and one reconfigurable that has to be tested. In the figure, five PWs are highlighted, where two are untestable, one critical, and two testable.

A PW is marked as:

- *Untestable*: if the OutWire stays in another region or all InWires stay in another region, then, it is impossible to drive a signal within the AUT (red PW in Figure 7.17).
- *Critical*: if the OutWire and the InWires are in the AUT, but there is also an

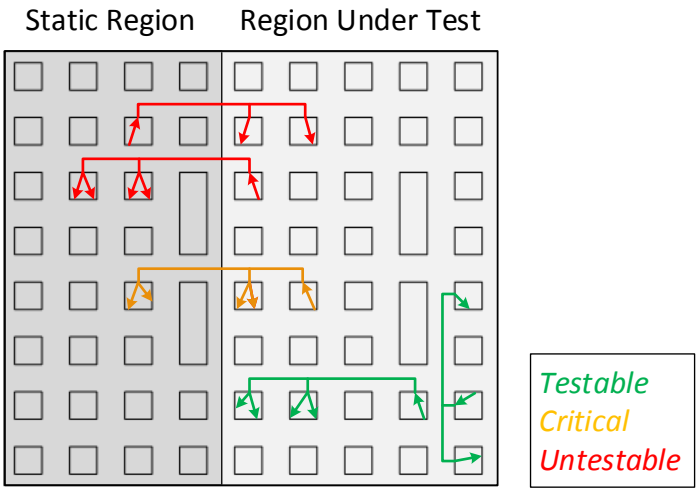


Figure 7.17: Analysis of testability.

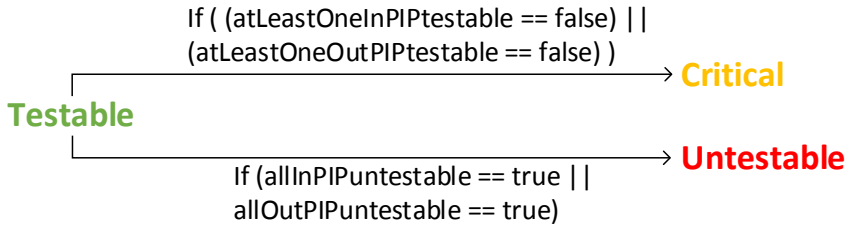
InWire in another region; if this physical is used an error can be propagated in the other region (orange PW in Figure 7.17).

- *Testable*: if and only if the OutWire and all InWires are in the area to be tested, then the PWs are marked as testable (green PW in Figure 7.17)

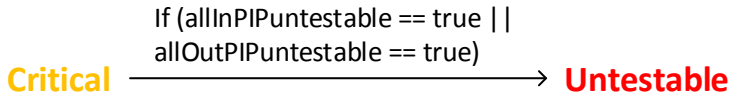
Once that all the PWs are categorized, all the PIPs are initialized as well. To all the PIPs of the AUT is assigned the testability of the connected PWs. Since that a PIP is always connected to two different PWs, it is probable that the two PWs connected to the PIPs have different testability categorizes (e.g., testable and untestable).

Therefore, three different conflicts are possible:

- Testable / Critical: the PIPs is marked as critical. The PIP is connected to a testable and a critical PWs. Therefore, this situation is considered as a critical case.
- Testable / Untestable: the PIPs is marked as untestable. The PIP cannot be used because one of the PW is marked as untestable.
- Critical / Untestable: the PIPs is marked as untestable. The PIP cannot be used because one of the PW is marked as untestable.



(a) Testable->Critical and Testable->Untestable change of testability of a PW.



(b) Critical->Untestable change of testability of a PW.

Figure 7.18: Change of testability of one PW according to its PIPs.

Dependency Phase

In the *Dependency Phase*, the dependency of the routing resources is analyzed. As explained above, the PIPs are categorized after the PW categorization; in the case of conflicts, this priority is considered: untestable/critical/testable.

A possible example can be a PW that is marked as testable, but just untestable PIPs are then connected to it; therefore, the PW needs to be marked as untestable.

For this reason, in the Dependency Phase, for every PW the testability of the connected PIPs is verified.

Figure 7.18 shows in which case the testability of the PW can change. Three cases are considered:

- Testable -> Untestable: the testability of a PW changes from testable to untestable if all the inPIPs are untestable, or all the outPIPs are untestable (Figure 7.18a).
- Testable -> Critical: the testability of a PW changes from testable to critical if there is no testable inPIP or no testable outPIP connected. Therefore, the PW is marked as critical (Figure 7.18a).
- Critical -> Untestable: the testability of a PW changes from critical to untestable if all the inPIPs are untestable, or all the outPIPs are untestable (Figure 7.18b).

It is important to mentions that, as in the Initialization Phase, once that all the testability of the PWs is verified, the PIPs are analyzed considering the updated testability categories of the PWs. Moreover, these two steps are cyclically executed till no testability of the routing resources is changed.

Stuck-on Testability Phase

In this phase, just the stuck-on testability of the PIPs is considered. As explained in Section 7.4.1, all the PIPs that are marked as stuck-off critical and stuck-off untestable are consequentially marked as stuck-on untestable.

For the stuck-off testable PIPs, two cases are possible:

- Stuck-on testable: considering the PWs (PWa and PWb) connected by the PIP ($PIPa$). Apart from the $PIPa$, if PWa has at least a testable PIP and PWb has at least a testable PIP, the $PIPa$ is marked as stuck-on testable.
- Stuck-on untestable: if the PIP is not stuck-on testable.

7.4.4 Testing Circuit Dependent (TCD) Analysis

In the Testing Circuit Dependent Analysis part, the RRA analyses the routing resource testability according to the capability of the utilized routing resources; the categorization presented in Section 7.4.3 depends on the FPGA partitioning. The testing circuits presented and developed in this work are focusing on the test of global routing resources (i.e., the connection among the SMs). This means that the current testing circuit can full verify and proof the effectiveness and correctness of behavior. The testing of further routing resources (e.g., DSP and BRAM resources) can be provided just implementing new testing circuits. With respect to the TCI Analysis, the unsupported category is considered.

As it is depicted in Figure 7.16, the TCD Analysis is executed just after the TCI Analysis. Moreover, it is worth noting that only testable physical wires can become unsupported (Figure 7.19). Therefore, at the end of the computation, the untestable and critical resources are the same as in the TCI Analysis.

The TCD Analysis is divided into three main phases: Initialization Phase TCD, Dependency Phase TCD, and stuck-on Testability TCI.

Initialization Phase TCD

In this phase, the RRA marks the routing resources that cannot be routed by the presented testing circuit ((e.g., DSP and BRAM resources). More in detail, in this phase, the RRA takes in input the testable resources given by the Phase 3 (stuck-on Testability TCI Phase) and divides them in TCD Testable and Untestable resources (as depicted in Figure 7.16).

First of all, the algorithm categorizes the PWs. Then, the PIPs are analyzed. As for the Initialization Phase TCI (described in Section 7.4.3), to all the PIPs of the AUT, it is assigned the testability of the connected PWs. Since that, a PIP is always connected to two different PWs, it is probable that the two PWs connected to the PIP have different testability categorizes (e.g., testable and unsupported).

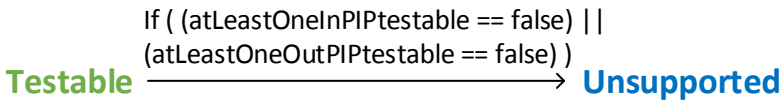


Figure 7.19: Change of testability of one PW according to its PIPs (TCD analysis).

The possible conflicts are: Unsupported / TCD Testable, Unsupported / Critical, Unsupported / Untestable. In all the cases, the PIP is marked as Unsupported.

Dependency Phase TCD

In the *Dependency Phase*, the dependency of the routing resources is analyzed. As explained above, the PIPs are categorized after the PW categorization, and in the case of conflicts, the PIP is marked as unsupported. For this reason, in the Dependency Phase, for every PW the testability of the connected PIPs is verified.

Differently from the Dependency Phase (explained in Section 7.4.3), in this phase just one transition is possible: TCD Testable -> Unsupported (as it is shown in Figure 7.16).

In the specific, a PW changes its state from TCD testable to Unsupported if there is no testable inPIP or no testable outPIP connected to it. Therefore, the PIP is marked as critical.

Stuck-on Testability Phase TCD

In this phase, just the stuck-on testability of the PIPs is considered. As for the other untestable and critical resources, the PIPs that are marked as stuck-on Untestable remain the same. On the contrary, the TCI stuck-on Testable PIPs can be of two different categories:

- TCD stuck-on testable: let's consider two PWs (PWa and PWb) connected a PIP ($PIPa$). Apart from the $PIPa$, if PWa has at least a TCD testable PIP and PWb has at least a TCD testable PIP, the $PIPa$ is TCD stuck-on testable.

- TCD stuck-on untestable: if the PIP is not stuck-on testable.

7.4.5 Result Output

At the end of the computation of the RRA, the results can be presented in three possible formats: *FPGA-Edline script, text reports, and heatmap*. In the following, the formats are presented.

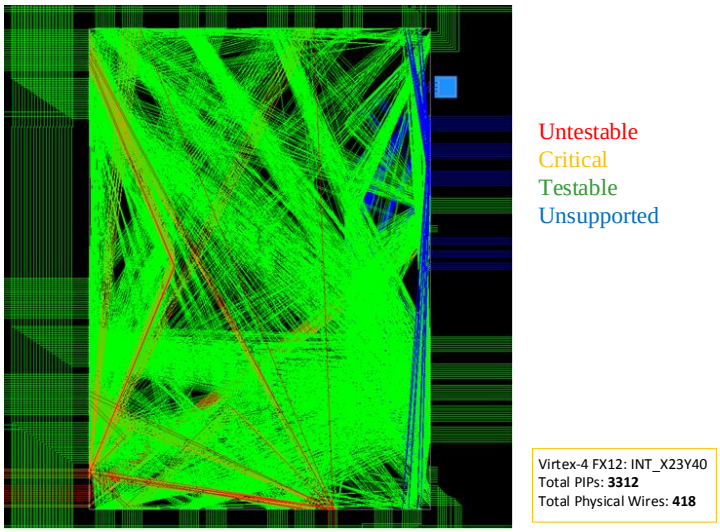


Figure 7.20: FPGA Editor script example; screenshot taken in FPGA Editor of a Virtex-4 FX12.

FPGA Editor Script

The RRA is capable of generating FPGA-Edline script files (see 2.3.2) to visualize resources in a different color according to their testability: testable in green, untestable in red, critical in orange and unsupported in blue.

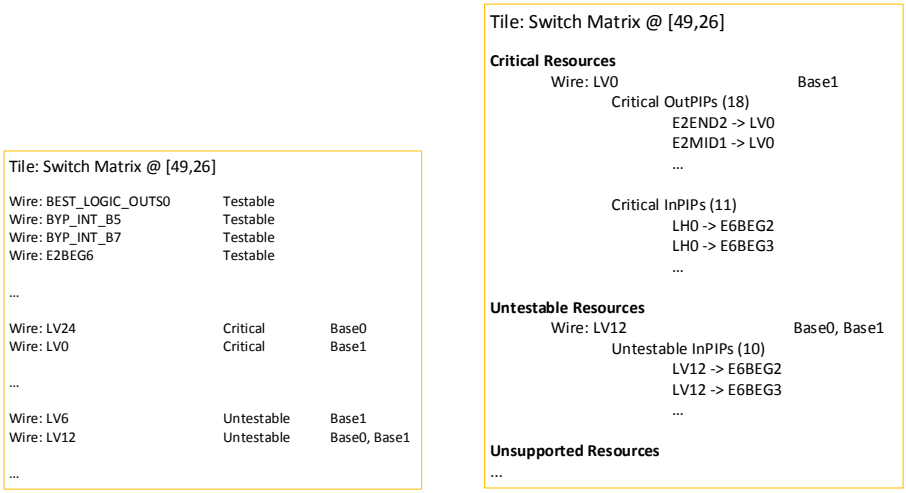
Figure 7.20 is an FPGA Editor screenshot that represents a switch matrix of the Virtex-4 FX12 where PWs and PIPs are highlighted: therefore, the figure visualizes the testability of the 3312 PIPs, and the 418 PWs connected the switch matrix.

With this kind of output, the user can visualize directly the results of the RRA directly in FPGA Editor.

Text Reports

The RRA generates two different reports as output: a *Physical wires testability report* and *PIPs testability report*. An example of the structure of these reports is shown in Figure 7.21a and Figure 7.21b.

Both reports are grouped in tiles. The "Tile" indicates a general component of an FPGA (e.g., an SM or a CLB); the tile's coordinates are indicated in the square brackets (in the example, [49,26]).



(a) Physical wires report (b) PIPs report

Figure 7.21: Text Report example.

The Physical wires (PWs) report lists physical wires testability of all tiles in the area. For non-testable wires, the conflicted areas are given. For example, in Figure 7.21a the wire $LV0$ is marked as critical and indicates that the conflict generated by one or more wires connected to the $Base1$ region.

The second report, PIPs testability report, gives details about the testability of the PIPs. As it is shown in the example Figure 7.21b, for every tile the PIPs are grouped according to their testability.

Heatmap

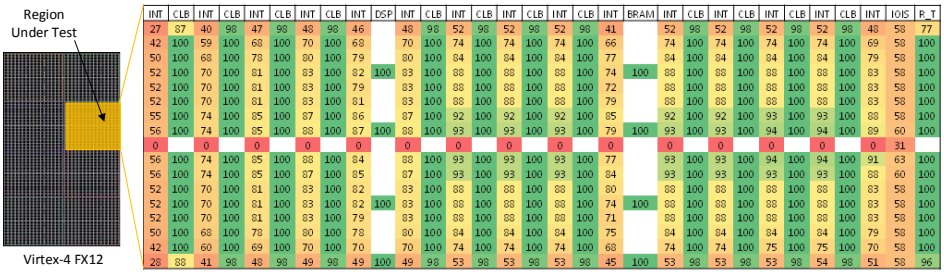
The heatmap is represented by a comma-separated value file (CSV). The idea of the heatmap is to have a faster feedback of the PIP's testability for each tile of the region under test. Two different types of heatmaps are created: *Testing Circuit Independent (TCI)* and *Testing Circuit Dependent (TCD)*.

Figure 7.22a and Figure 7.22b provide an example of TCI heat-map and TCD heat-map respectively. In the examples, one clock region of a Virtex-4 FX12 is considered as AUT. Every cell of the tables represents one FPGA's tile (e.g., INT, CLB, DSP). The numbers in figures represent the percentage of testable PIPs within the corresponding tile.

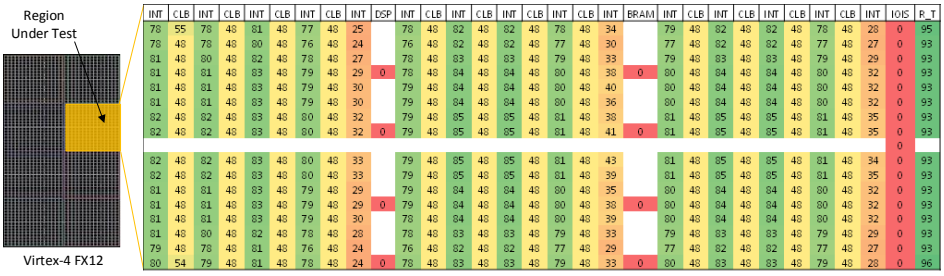
Figure 7.22a represents the maximum achievable values of testability according to the partitioning of the FPGA; therefore, this heatmap is Testing Circuit Indepen-

dent. It can be noticed that larger values appear in the middle of the region, while numbers decrease as the tiles get closer to the edges.

Figure 7.22b shows a Testing Circuit Dependent heatmap. The values of this heatmap represent the percentage of testable resources on the testable resources of the TCI report. Therefore, this heatmap highlight which are the routing resources that cannot be tested by the presented testing circuits. For example, for DPS tiles, the values within the table are 0. This is because, as explained in Section 7.2, the presented testing circuit is not able to test the routing resources of a DSP. Therefore, all the routing resources of a DSP are set as "unsupported". As mention in the Section 7.4.4, this limitation is due to the time required to implement new testing circuits. At the moment, just one testing circuit has been utilized for the proof of concepts.



(a) Test Circuit Independent heat-map



(b) Test Circuit Dependent heat-map

Figure 7.22: Heatmap example. The picture shows a screenshot taken in FPGA Editor of a Virtex-4 FX12.

7.5 The U-TURN Place-and-Route Algorithm

The U-TURN algorithm represents the core of the OLT(RE)^2 testing approach. The basic idea is that the previously presented test circuit (see Section 7.2), composed of a TPG, an ORA, and eight nets under test is placed once, and then the N-UTs are routed multiple times (through multiple iterations of the U-TURN algorithm). By changing the routing of the N-UTs (and leaving unaltered the placement of TPG and ORA), it is possible to test different resources in the AUT.

Therefore, each run of U-TURN generates a placed-and-routed test circuit that covers a given subset of the routing resources available in the AUT. In particular, after a set of dedicated experiments, it has been discovered a technological constraint on the Xilinx architectures: each single net (and thus also each NUT of the testing circuit) can occupy (and test) no more than 100 PIPs, and thus no more than 101 physical wires.

The constraint of 100 PIPs has been chosen as a result of experiments concluded on different Xilinx devices and families. When utilizing too many PIPs in one net, the signal on the particular net shows accumulated jitter and slew rate degradation, eventually resulting in failing circuit functionality. The effect depends on device family, temperature, supply voltage, and process variations. Choosing a limit of 100 PIPs guarantees a stable test circuit operation across all device families and environment conditions.

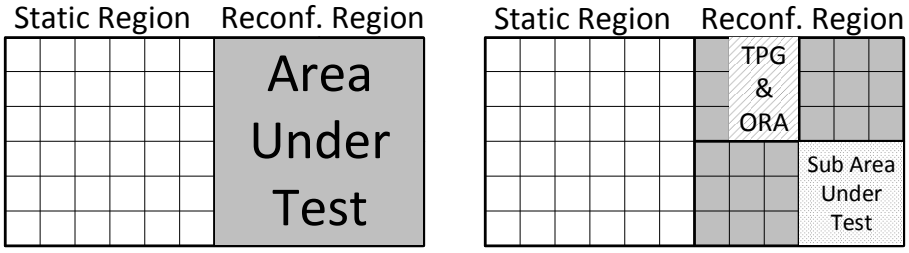
It is fundamental to point out that U-TURN relies on the database of architectural resources described in Section 5.2.1. The information stored in the database is used actually to drive the U-TURN algorithm. In the following, the TPG and ORA placement algorithm and the N-UTs routing algorithm area introduced.

7.5.1 The TPG & ORA Placer

As presented in Section 2.2.2, in a typical reconfigurable system, an FPGA is partitioned in a static region and a reconfigurable region. The reconfigurable region is the area where functional modules can be dynamically placed. The static region is the area of the FPGA where all the structures required to support the dynamic reconfiguration of the reconfigurable region are placed; thus, the content of the static region is fixed.

OLT(RE)^2 focuses on faults in the routing resources of the reconfigurable region (since it relies on DPR), while the static region has to be tested with dedicated approaches. Figure 7.23a shows an example of an FPGA partitioning where the right-bottom dark-gray box represents the AUT.

Algorithm 10 presents the pseudo-code of the U-TURN placement algorithm. The first step that the algorithm performs is the division of the AUT into N non-overlapping sub-areas under test (line 3 of Algorithm 10). The testing procedure focuses on one of the obtained sub-areas at a time. When testing a given sub-area,



(a) The Area Under Test.

(b) Partitioning of the Area Under Test and placement of TPG and ORA.

Figure 7.23: Testing partitioning [178].

```
1:  Function U-TURN()
2:  testedResources ← ∅
3:  divide the Area Under Test into n sub-areas under test
4:  for each Sub-Area Under Test a_i do
5:     Place TPG and ORA
6:     for each PIP pip_ut in a_i do
7:        testingCircuit ← ∅
8:        for each output tpg_j of the TPG do
9:           sm_ut ← getSwitchMatrix(pip_ut)
10:          route tpg_j to pip_UT according to the targeted fault
11:          nut ← N-UT_Router(pip_ut, 0)
12:          testedResources.add(getAllUsedResources(nut))
13:          route nut to input j of the ORA
14:          testingCircuit.add(nut)
15:       end for
16:       save testingCircuit
17:    end for
18: end for
19: EndFunction
```

Algorithm 10: The overall placement and routing algorithm [**\cite {Cozzib}**].

TPG and ORA are placed in one of the remaining sub-areas under test (line 5 of Algorithm 10); in this way TPG and ORA do not occupy resources belonging to the sub Area Under Test (sub-AUT), that can therefore be entirely tested.

Moreover, it is worth noting that TPG and ORA are always placed in one of the sub-AUT (and thus always into the AUT itself): in this way it is ensured that the

testing procedure does not interfere with the functioning of the modules placed outside the AUT. Figure 7.23b shows an example of partitioning of the AUT and placements of TPG and ORA.

After placing TPG and ORA (which remain in the same position for the test of an entire sub-area), the U-TURN algorithm has to be iterated multiple times in to generate all the test circuits needed to cover the sub-AUT entirely. For each test circuit, each output of the TPG has to be routed through the AUT (line 10 Algorithm 10) and then to an input of the ORA (line 13 Algorithm 10).

Although each test circuit covers multiple resources, the generation process of a test circuit focuses on one PIP, the PIP under test (PIP-UT, pip_{ut} at line 6 of Algorithm 10). The connection between the output of the TPG and the starting point of the N-UT depends on which fault the N-UT is meant for (as discussed in the previous section).

7.5.2 The N-UTs Router

Once the output of the TPG has been routed to the PIP-UT (line 10 of Algorithm 10), the recursive `N-UT_Router` algorithm is run to build the N-UT (line 11 of Algorithm 10). The pseudo-code of `N-UT_Router` is shown in Algorithm 11.

The N-UT exits from the switch matrix sm_{ut} through PIP pip_{ut}, and it reaches an SM sm_k through PW pw_j and it eventually occupies one of the PIPs reachable from pw_j (the PIP pip_x). All the resources that the N-UT occupies are stored in a temporary solution as well as marked as already visited (line 11 of Algorithm 11).

Each time the N-UT comes back to the switch matrix sm_{ut} the *checkSolutions()* procedure is invoked (line 14 of Algorithm 11). This procedure checks whether the temporary solution occupies more yet untested resources than the current best solution. If yes, the temporary solution becomes the best solutions, while no exchange is performed otherwise.

Given the technological constraint discussed above, the procedure is repeated until the N-UT occupies 100 PIPs. Finally, when the N-UT construction process is completed, the N-UT is routed from the PIP-UT to one of the inputs of the ORA (line 13 of Algorithm 10).

As previously said, even if each N-UT targets one PIP, after being routed, the N-UT covers multiple physical wires and PIPs. Thus, the test circuit generation procedure needs always to keep track of the already covered PIPs (line 12 of Algorithm 10) to avoid considering these as PIP-UT of the following N-UTs, and thus saving computational time.

```
1: Function N-UT_Router(pip_ut, nUsedPIPs)
2: if nUsedPIPs > 100 then
3:    return
4: end if
5: get the physical wire pw_j reachable from pip_ut
6: for each PIP pip_x not yet visited and reachable from pw_j do
7:    get the physical wire pw_y reachable from pip_x
8:    if pw_y has not yet been visited then
9:       add pip_x and pw_y to the temporary solution
10:      set pip_x and pw_y as visited
11:      if pip_x belongs to sm_ut then
12:         checkSolutions()
13:      end if
14:      N-UT_Router(pip_x, nUsedPIPs+1)
15:      remove pip_x and pw_y from the temporary solution
16:   end if
17: end for
18: return
19: EndFunction
```

Algorithm 11: The N-UT creation algorithm [178].

7.6 Results

Three sets of experiments have been performed to evaluate the correctness, effectiveness, and efficiency of the proposed testing approach.

First of all, the functionality of the designed basic test circuit has been verified (without considering the U-TURN place-and-route algorithm) to prove that the testing circuit can detect all the possible faults that may occur in the N-UTs. The second test regards the actual fault coverage that OLT(RE)^2 can assess. This test analyzes the coverage achieved with the generated placed-and-routed test circuits as well as its efficiency in terms of time required to generate the test circuits and of the size of the test circuits. The third test analyzes the run-time efficiency of the approach in terms of time needed to transfer the test circuits to the device under test and then run them.

7.6.1 Test Circuit Validation

This experiment validates the implemented basic test circuit. More in detail, the focus of this test has been assessing whether the circuit is actually able to detect all the faults that may occur in the nets connecting the TPG and the ORA.

Table 7.1: Area occupation of the designed test circuit for several FPGA families [178].

Family	#LUTs			#FFs	#slices
	logic	memory	shift reg.		
Virtex-4	35	3	1	11	24
Virtex-5	25	3	1	11	18
Virtex-6	25	3	1	10	20
Spartan-6	33	3	1	13	23

In order to do so, it has been emulated the occurrence of faults in the NUTs. In more detail, the XDL file of the testing circuit has been manipulated to reproduce the effect of faults. In this way, it is possible to inject stuck-on/off faults by activating/removing PIPs in the XDL file; similarly, it is possible to emulate stuck-at 0/1 by forcing the output of specific LUTs in the TPG to 0/1. Finally, the XDL file is translated into a "faulty" bitstream and downloaded in the FPGA device.

The fundamental result of this preliminary validation is that the designed test circuit can detect $100\ \%$ of the faults occurring in the N-UTs. Additionally, $100\ \%$ of the faults occurring in the routing resources occupied by the Internal Reset Generator and by the Distributed RAM are detected. The great majority (about $97\ \%$) of the faults occurring in the routing resources occupied by the Internal Clock Generator component are also detected.

The only critical sub-component of the test circuit is the Start-checking Circuit: when faults occur in the routing resources occupied by this sub-component, the test fails (because the Start-checking Circuit does not recognize the start of the test) even if the N-UTs are not affected by faults. More details about the validation of the testing circuit have been presented in [186].

In Table 7.1 the amount of resources (number of LUTs used for logic, DRAM and shift-registers and number of used flip-flops as well as the total number of occupied slices) occupied by the test circuit (i.e., TPG and ORA) for the four currently supported FPGA families (Spartan-6 and Virtex-4, Virtex-5 and Virtex-6) are summarized. The area occupation regarding percentage of used resources actually depends on the specific device: for example, the area occupation on the smallest Virtex-4 device, the FX12 (counting 5472 slices) is only 0.43 %; of course, the percentage of occupied area in larger devices is even smaller. Thus, as expected, the proposed testing technique could also be applied in case the majority of the FPGA area is already occupied.

Table 7.2: TCD routing testable resource of the tested FPGAs. The selected AUT has the dimension of one Clock Region [178].

Family	Device	#PW stuck-at	#PIP stuck-off	#PIP stuck-on
Virtex-4	FX12	38,784	444,476	427,792
	FX100	111,179	1,317,736	1,270,955
Virtex-5	LX20T	79,425	941,323	900,205
	LX330T	283,021	3,453,305	3,303,288
Virtex-6	CX130T	331,684	4,130,930	3,973,033
	LX760	937,398	11,728,174	11,253,950
Spartan-6	LX9	25,504	268,364	255,940
	LX150T	121,804	1,384,587	1,325,959

7.6.2 Design-time Performance Analysis

This analysis aims at assessing the performance of the design-time test circuit generation. The routing resource fault coverage is investigated for various FPGA families and devices, first evaluate the effectiveness of the approach (called *Effectiveness Analysis*). The time needed to place-and-route all the required test circuits is evaluated as well. This test is provided first for various FPGA families and devices, and then for various sizes of the AUT on the same device, to assess scalability of the approach (called *Efficiency Analysis*).

Experimental Setup

The design-time test generation flow has been executed on eight devices belonging to four different families (Spartan-6 and Virtex-4, Virtex-5 and Virtex-6). In this way, it is shown that OLT(RE)^2 can be utilized in a wide range of FPGAs families. In the effectiveness analysis, one clock region has been considered as AUT. When focusing on scalability, a Virtex-4 XC4VFX12 device has been utilized; they are considered areas of one clock region up to four clock regions as AUT.

The U-TURN algorithm has been launched on a PC equipped with an Intel Xeon Processor W3565 with 24 GB of RAM. After the generation of the test circuits by U-TURN, it has been evaluated the achieved fault coverage by measuring the amount of the routing resources of the AUT occupied by the whole set of test circuits.

Effectiveness Analysis

The fault coverage achieved by OLT(RE)^2 for several device families and models have been measured; Table 7.2 and Table 7.3 reports the results of this experiment.

Table 7.3: Effectiveness Analysis, Fault Coverage. The percentages show the covered resources with respect to the testable routing resources of Table 7.2 [178].

Family	Device	#PW sa	#PIP stuck-off	#PIP stuck-on
Virtex-4	FX12	38,784 (100.00 %)	439,074 (98.78 %)	427.646 (99,97 %)
	FX100	111,179 (100.00 %)	1,305,778 (99.09 %)	1,248,119 (98.20 %)
Virtex-5	LX20T	79,225 (99.75 %)	929,720 (98.77 %)	887,975(98.64 %)
	LX330T	282,821 (99.93 %)	3,304,783 (95.70 %)	3,199,157 (96.85 %)
Virtex-6	CX130T	329,262 (99.27 %)	3,905,723 (94.55 %)	3,866,526 (97.32 %)
	LX760	932,617 (99.49 %)	10,819,240 (92.25 %)	10,648,488 (94.62 %)
Spartan-6	LX9	25,242 (98.97 %)	255,771 (95.30 %)	243,091 (94.98 %)
	LX150T	120,594 (99.01 %)	1,318,006 (95.19 %)	1,274,644 (96.13 %)

For each FPGA family, two devices have been considered: a small- and a large-size one.

First, Table 7.2 reports the number of testable stuck-at faults on physical wires and the number of possible stuck-off and stuck-on faults on PIPs; these values have been extracted after the RRA step, where all the routing resources are categorized. Then, Table 7.3 reports the achieved coverage for the stuck-at, stuck-off, and stuck-on faults, respectively.

It can be observed that the proposed testing technique always achieves a high fault coverage (more than 98 % in most cases) both for stuck-at faults on physical wires and for stuck-off and stuck-on faults on PIPs. It is worth noting that these good results are achieved for all the device families, which are different regarding routing architectures. This demonstrates that the proposed technique is actually independent of the specific FPGA architecture as well.

Efficiency Analysis

Table 7.4 reports about the efficiency of the proposed approach. More in detail, for each device the table shows the number of test circuits generated to achieve the fault coverage values reported in Table 7.2, as well as the total size (in MB) of the test bitstream suite and the size of the test bitstream suite after bitstream compression and, finally, the time required at design time to generate them.

A first consideration that can be drawn from the results shown in Table 7.4 is that the time required at design time to generate the test bitstream suite is reasonable, ranging from about 3 hours for the smallest device up to about 16 days for the largest one; the generation of the test bitstreams has to be performed only once at design-time. Moreover, since the test circuit and the overall approach are

Table 7.4: Summary of the performance of OLT(RE)^2 [178].

Family	Device	# Testing circuits	Bitstream Size	Bitstream Size (Compr.)	Time
Virtex-4	FX12	8,058	249 MB	31 MB	12h:52m
	FX100	38,245	1623 MB	149 MB	78h:25m
Virtex-5	LX20T	19,562	1217 MB	96 MB	47h:22m
	LX330T	34,081	1555 MB	150 MB	87h:46m
Virtex-6	CX130T	41,053	5418 MB	264 MB	95h:27m
	LX760	120,568	7364 MB	719 MB	382h:17m
Spartan-6	LX9	4,179	98 MB	10 MB	3h:40m
	LX150T	16,328	914 MB	83 MB	40h:48m

application-independent, it is worth noting that, after having been generated for a given FPGA device, the test bitstreams can be used for several reconfigurable systems using the same device without any modification.

Looking at the size of the test bitstreams it can be seen that the "raw" size is large, ranging from hundreds of MBs up to some GBs. This may represent a limitation of the proposed testing technique, since, as previously discussed, test bitstreams have to either be stored in an on-board persistent memory or received "just in time" from the ground station.

This problem can largely be alleviated by using a compression algorithm to reduce the size of the bitstream. Even if bitstream compression is not in the scope of the presented work, it has been investigated how much the total size of the bitstream could be reduced. It can be observed that the overall suite size it has been reduced to about 10 % of the original size with a simple run of the ZIP algorithm.

It is important to mention that the current test flow can be integrated with new test circuits with the goal of performing a more fine-grained fault diagnosis in case a fault is detected at run-time, thus making possible to re-use partially faulty areas. In addition, this can allow having the possibility to place more testing circuits at the same time, reducing the number of reconfiguration required for a full test.

Scalability Analysis

This experiment assesses the scalability of OLT(RE)^2. Multiple tests have been executed utilizing a Virtex-4 device in which the size of the AUT has been increased from 1 clock region up to 4 clock regions (i.e., half of the entire device).

Table 7.5 and Table 7.6 report the results of this scalability analysis experiment. More in detail, Table 7.5 reports the size of the AUT in terms of number of clock re-

Table 7.5: TCD routing testable resource of for different size of AUT. This test has been executed on an XC4VFX12 device [178].

Area Under Test	#PW stuck-at	#PIPs stuck-off	#PIPs stuck-on
1 clock region	38,784	444,476	427,792
2 clock region	83,510	1,013,107	978,481
3 clock region	127,948	1,575,888	1,523,548
4 clock region	396,382	2,120,036	2,050,807

Table 7.6: Coverage of the testable routing resources. The percentages show the covered resources with respect to the testable routing resources of table (a). This test has been executed on an XC4VFX12 device [178].

Area Under Test	stuck-at	stuck-off	stuck-on
1 clock region	38,784 (100.00 %)	439,074 (98.78 %)	427,646 (99.97 %)
2 clock region	83,510 (100.00 %)	1,000,594 (98.76 %)	976,606 (99.81 %)
3 clock region	127,948 (100.00 %)	1,558,411 (98.89 %)	1,520,071 (99.77 %)
4 clock region	396,382 (100.00 %)	2,098,496 (98.98 %)	2,046,229 (99.78 %)

gions, the number of stuck-at faults on physical wires and the number of stuck-off and stuck-on faults on PIPs, respectively; then, Table 7.6 reports the achieved fault coverage values for stuck-at, stuck-off and stuck-on faults, respectively. It can be observed that the effectiveness of the proposed approach in achieving high fault coverage values scales well with respect to the size of the Area Under Test. In all cases, more than 98 % of faults are covered by the test bitstreams.

Table 7.7 reports the size of the AUT in terms of number of clock regions (first column), number of test circuits generated to achieve the fault coverage values reported in Table 7.6 as well as the total size (in MB) of the test bitstream suite and the size of the test bitstream suite after bitstream compression (third and fourth columns, respectively) and, finally, the time required at design time to generate them.

Again, it can be observed that the proposed approach scales well both test bitstreams size and test generation time with respect to the size of the AUT. These tests show that OLT(RE)^2 can be effectively and efficiently used for a wide range of device families and that it does not suffer from the increase in the size of the AUT.

Table 7.7: Analysis of the scalability of OLT(RE)^2 on the Virtex-4 XC4VFX12 device (number and size of the test bitstreams and generation time) [178].

AUT	#testCirc.	B. Size	B. Size (Compr.)	Time
1 clk reg.	8,058	249 MB	31 MB	12h:52m
2 clk reg.	19,014	999 MB	88 MB	34h:56m
3 clk reg.	30,552	1863 MB	148 MB	72h:07m
4 clk reg.	41,011	2912 MB	215 MB	102h:18m

7.6.3 Run-time Performance Analysis

The run-time performance of the testing technique has been evaluated on the DRPM to demonstrate that the proposed testing approach can be applied in a real-world scenario on a complete reconfigurable system (see Chapter 4). This test considers both the time needed to transfer the test bitstreams to the reconfigurable system and the time required to execute the entire test suite by partially reconfiguring the FPGA under test.

DRPM as Validation Platform

The data processing modules (see Section 4.1.3) host a Xilinx Virtex-4 FX100 FPGA, which provides DPR capability. This processing module is used for the validation process of the OLT(RE)^2 flow. The overall test diagram is shown in Figure 7.24. One of the key components is the self-hosting reconfiguration controller (SHRC), which interfaces the internal configuration access port (ICAP) and the FrameECC controller in the FPGA. Therefore, the SHRC is used to perform dynamic partial reconfiguration internally and to readback the configuration of the FPGA. A MicroBlaze CPU controls SHRC and initiates internal transfers from/to the DDR2 memory. The FPGA has been partitioned into two main regions: a static region and a reconfigurable region. Within the static region, the main components are the communication module, the SHRC, and the MicroBlaze. The reconfigurable region is the target area of the test (the AUT). Given its complexity and completeness, the DRPM represents the perfect run-time platform for this kind of experiment.

The test is executed as follows: (i) the test bitstreams are sent from a host PC to the DDR2-RAM of the target systems; (ii) the MicroBlaze is triggered and executes partial dynamic reconfigurations in order to exhaustively place all the test circuits on the Area Under Test; (iii) the results of the execution of each test circuit are readback from the DRAM. Finally, when all the test circuits have been executed, a report is sent to the host PC.

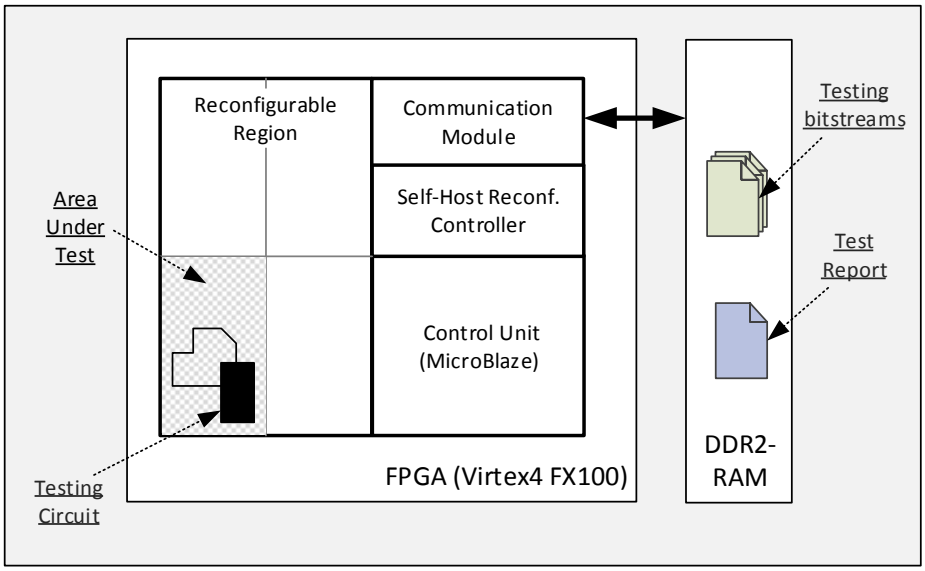


Figure 7.24: The architecture of the DRPM reconfigurable module [178].

Run-time Results

The execution of each test circuit on the DRPM took 30.281 ms, while the analysis of the results produced by each test took 1.373ms. Thus, the execution of the entire test set (38,245 test circuits) on the DRPM required about 20 minutes. The number of the testing circuit is given by the design time performance analysis on the Virtex-4 FX100 (Section 7.6.2), which is the device on board of the DRPM communication module. More details about the run-time tests are provided in [98].

This number is acceptable if it is taken into account that a reconfigurable system employed in a space mission does not execute frequent reconfigurations. Moreover, it can be estimated the time required to transfer the entire test suite from a hypothetical ground station to a hypothetical satellite hosting the DRPM: considering the bandwidth of 306 Mbps of the FOBS (described in 3.3.2) and considering the total size of the compressed suite reported in Table 7.4 (162 MB) it can be estimated a total transfer time of about 4 seconds, that, again is reasonable for a space mission.

7.7 Summary

This chapter ha described OLT(RE)^2, a testing technique meant to be applied on-line and on-demand to detect permanent faults in reconfigurable systems. The proposed technique is particularly interesting for two main reasons: on the one hand, it can help designers in making the use of state-of-the-art high-performance commercial-grade FPGAs viable for space applications; on the other hand, it can help in low-cost application scenarios where high-end radiation-hardened devices are not affordable.

Experimental results have shown that the proposed approach may be applied to a large set of FPGA families and models, allowing the great majority of faults both in the physical wire (PW) and PIPs to be detected. More specifically, OLT(RE)^2 have shown that is possible to generate, place, and route the test circuits needed to detect on average more than 99 % of the physical wires and on average about 97 % of the programmable interconnection points of a large arbitrary region of the FPGA in a reasonable time. Moreover, the test can be run on the target device without interfering the functional behavior of the system.

Furthermore, OLT(RE)^2 has demonstrated to be highly scalable both in terms of execution time and of number and size of the test circuits when increasing the size of the Area Under Tests (AUT). Finally, the run-time experiment carried out on the DRPM demonstrated that the time required to transfer and apply the test circuits makes the proposed approach viable for real-world space applications.

8 Conclusion and Outlook

This thesis has focused on reconfigurable architectures, which have become key platforms for many applications. A new prototyping environment (i.e., DRPM) and tools (i.e., INDRA and DHHarMa) have been introduced to investigate the use of Dynamic Partial Reconfiguration (DPR) in different use cases.

The presented DRPM is a complete working platform to study the DPR in space missions. A detailed comparison with the existing prototyping platform as well as in-flight platforms has been provided. Differently to the other platforms, the DRPM allows advanced prototyping of space mission scenarios based on DPR, thanks to the integration of modern reconfigurable SRAM-based FPGAs with different avionic interfaces (e.g., CAN, SpaceWire, SpaceFibre).

In particular, this thesis has presented a novel communication interface, which manages the communication among the different processors of the DRPM: the Heterogeneous Multi Processor Communication Interface (HMPCI) protocol. This communication interface has been developed to be general and to be adapt to different multi-processors systems, thanks to its easy to use implementation. Furthermore, HMPCI has small latency and low memory footprint compared to the state-of-art multi-processor communication protocols (i.e., MMPI, MCAPI, MSG, SoC-MPI).

The missing DPR functionalities on the commercial and academic FPGA toolchains have been filled with the introduction of the INtegrated Design flow for Reconfigurable Architectures 2.0 (INDRA 2.0). This novel flow allows creating a full working DPR environment that supports bitstream relocation. It generates a configurable bitstream starting from an HDL description of a design. INDRA 2.0 flow supports the modern Xilinx FPGAs families Virtex-4, Virtex-5, Virtex-6, Spartan-6, 7 Series, and Zynq.

This thesis has provided details about the creation of an FPGA PW database, which has been generated extracting the information from the Xilinx commercial tools. The PW database is parted of the Datastructure for Xilinx FPGAs (DXF) database, which has enabled the creation of a flow that is compatible with the standard Xilinx FPGA tools (i.e., Xilinx ISE).

One important tool developed for INDRA 2.0 is Design flow for Homogeneous Hard Macros (DHHarMa). In particular, this thesis presents the router of DHHarMa, introducing new concepts and algorithms that have allowed generating a homogeneous routing of a specific design (e.g., a communication infrastructure). Benchmarks have shown that the router can homogeneously route a communica-

tion infrastructure utilizing between 1% and 31% more routing resources than the Xilinx router, which provides just inhomogeneous solutions.

INDRA 2.0 has been used to generate the DPR environment of the DRPM. This has demonstrated the effectiveness of the presented approach and has allowed having a direct use of the INDRA 2.0 flow.

Finally, the thesis has investigated the permanent faults that can occur on the SRAM-based FPGA devices. The On-Line Testing of Permanent Radiation Effects in Reconfigurable System (OLT(RE)^2) has been introduced; a testing technique meant to be applied on-line and on-demand to detect permanent faults in reconfigurable systems. Parts of the OLT(RE)^2 flow are derived from the INDRA 2.0 flow, i.e., the Datastructure for Xilinx FPGAs (DXF) database, which provides a full open-source description of the modern Xilinx FPGAs. OLT(RE)^2 is compatible with the Virtex-4, Virtex-5, Virtex-6, and Spartan-6 FPGAs.

The proposed techniques can help designers in making the use of high performance unreliable commercial FPGAs viable for space applications. Furthermore, it can help in low-cost application scenarios where high-end radiation-hardened devices are not affordable.

This thesis focuses on the description of the overall flow of OLT(RE)^2, the categorization of the routing resources to be tested (i.e., the Routing Resources Analyzer (RRA)), and the place and route algorithm of the testing circuits (i.e., the U-TURN algorithm).

Experimental results of OLT(RE)^2 have shown that the proposed approach may be applied to a large set of FPGA families and models. It is possible to generate, place, and route the test circuits needed to detect on average more than 99 % of the physical wires and on average about 97 % of the programmable interconnection points of a large arbitrary region of the FPGA in a reasonable time.

Furthermore, OLT(RE)^2 has been validated on the DRPM, creating a full test-benchmark flow that verifies on-line that the routing resources of the reconfigurable regions are permanent fault-free. In addition, permanent faults have been emulated to verify the effectiveness of the testing approach. It is worth to mention that the OLT(RE)^2 tool flow has been inserted in the European Cooperation for Space Standardization (ECSS) book [36], which is a handbook that summarizes all the available testing techniques for radiation effects mitigation in ASICs and FPGAs.

8.1 Outlook

The DRPM can be easily adapted to future architectures by integrating additional daughterboards with new FPGAs or with additional new interfaces. For example, this has been the case of the SpaceFibre interface, which has been implemented after that the platform was already developed.

The developed DB-SPACE can be easily adapt to the new version of the Raptor-X64, the Raptor-XPress. This new modular board comes with further processing modules, i.e., DB-V5 and DB-V7, which embed a Virtex-5 FX100T and a Virtex-7 X690T respectively.

The INDRA 2.0 flow is compatible with the standard Xilinx ISE flow and can support a wide range of modern FPGAs. Xilinx has decided to create a new standard tool, Vivado, for latest and future FPGAs. Nevertheless, DHHarMa can be adapted to Vivado, which again leaks of the creation of DPR designs that supports bitstream relocation.

Xilinx ISE has an intermediate language, the Xilinx Desing Language (XDL), which allows creating a custom design that is compatible with the standard tools. At the same way, Vivado comes with a different intermediate language, the Tool Command Language (Tcl). Therefore, the DXF database and the DHHarMa output need to be adapt to this new scripting language.

OLT(RE)^2 has demonstrated to be highly scalable both regarding execution time and number and size of the test circuits when increasing the size of the area to be tested. Additionally, the flow can handle new kinds of new test circuits with the goal of performing a more fine-grained fault diagnosis in case a fault is detected at run-time, thus making possible to re-use partially faulty areas.

List of Figures

List of Tables

Acronyms

AMBA	Advanced Microcontroller Bus Architecture.
API	Application Programming Interface.
ARTEMIS	Advanced Responsive Tactically Effective Military Imaging Spectrometer.
ASIC	Application Specific Integrated Circuit.
ASMBL^{TM}	Application Specific Modular Block.
AUT	Area Under Tests.
BIST	built-in self-test.
Block RAM	Block Random Access Memory.
BPI	Byte-wide Peripheral Interface.
CLB	Configurable Logic Block.
COTS	Commercial Off-the-Shelf.
CPLD	Complex Programmable Logic Device.
CSV	Comma-separated values.
DCM	Digital Clock Manager.
DHHarMa	Design flow for Homogeneous Hard Macros.
DIRT	DIrected Routing Constraints.
DMA	Direct Memory Access.
DPCU	Dynamic Processing Control Unit.
DPR	Dynamic Partial Reconfiguration.
DRPM	Dynamic Reconfigurable Processing Module.
DSP	Digital Signal Processor.
DXF	Datastructure for Xilinx FPGAs.
EAPR	Early Access Partial Reconfiguration.

ECSS	European Cooperation for Space Standardization.
ECU	Electronic Control Unit.
EDAC	Error Detection and Correction.
ESA	European Space Agency.
FAR	Frame Address Register.
FOBS	Fraunhofer On-Board Processor.
FPGA	Field Programmable Gate Array.
FTECMU	Fault Tolerance and Error Correction Management Unit.
GPIO	General Purpose Input Output.
GRACE	Gravity Recovery And Climate Experiment.
GUI	Graphical User Interface.
HInterNetSet	Homogeneous intranet set.
HIntraNetSet	Homogeneous internet set.
HMPCI	Heterogeneous Multi Processor Communication Interface.
ICAP	Internal Configuration Access Port.
IDDFS	Iterative Deepening Depth-First Search algorithm.
INDRA	INtegrated Design flow for Reconfigurable Architectures.
INDRA 2.0	INtegrated Design flow for Reconfigurable Architectures 2.0.
INT	Interconnection Matrix.
IOB	Input Output Block.
IoT	Internet of things.
ISE	Integrated Software Environment.
ISS	International Space Station.
JTAG	Joint Test Action Group.

LET	Linear Energy Transfer.
LLOC	Logical System Lines Of Code.
LMBT	Link Macros Between Tiles.
LUT	Look Up Table.
MER	Mars Exploration Rover Mission.
MPI	Message Passing Interface.
MRO	Mars Reconnaissance Orbiter.
N-UT	Net Under Test.
NCD	Netlist Circuit Description.
NGC	Native Generic Circuit.
NGD	Xilinx Native Generic Database.
NMC	Xilinx logical description of the design and macro library.
NUT	Net Under Test.
OLT(RE)^2	On-Line Testing of Permanent Radiation Effects in Reconfigurable System.
ORA	Output Response Analyzer.
OTERA	Online Test Strategies for Reliable Reconfigurable Architectures.
P&R	placed and routed.
PAR	Place and Route.
PHI DPU	Polarimetric and Helioseismic Imager Data Processing Unit.
PIP	Programmable Interconnection Point.
PIP-UT	PIP Under Test.
PLB	Processor Local Bus.
PR	Partial Reconfiguration.
PR Module	Partial Reconfigurable Module.
PR Region	Partial Reconfigurable Region.
PR System	Partial Reconfigurable System.
PR Tile	Reconfigurable Tile.

PSRerouter	Post-Synthesis Rerouter.
PW	physical wire.
R3TOS	A Reliable Reconfigurable Real-Time Operating System.
RA-RCC	Responsive Avionics Reconfigurable Computer.
rad	Radiation Absorbed Dose.
rad-hard	radiation-hardened.
RHBD	rad-hard by design.
RHBS	Radiation Hardened By Software.
RPC	Remote Procedure Call.
RRA	Routing Resources Analyzer.
RRMU	Resource and Reconfiguration Management Unit.
RTEMS	Real-Time Operating System for Multiprocessor Systems.
SCARS	Scalable Self-Configurable Architecture for Reusable Space Systems.
SEE	Single Event Effect.
SEFI	Single Event Functional Interrupt.
SEL	Single-Event Latch-up.
SET	Single Event Transition.
SEU	Single Event Upset.
SHRC	Self Hosting Reconfiguration Controller.
SM	switch matrix.
SPI	Serial Peripheral Interface.
SPWRTC	SpaceWire Remote Terminal Controller.
SRAM	Static Random Access Memory.
sub-AUT	sub Area Under Test.
TCD	Testing Circuit Dependent.
TCI	Testing Circuit Independent.
Tcl	Tool Command Language.
TID	Total Ionizing Dose.
TMR	Triple Modular Redundancy.

TORC	Tools for Open Reconfigurable Computing.
TPG	Test Pattern Generator.
UCF	Xilinx User Constraints File.
VHDL	Very High Speed Integrated Circuit Hardware Description Language.
X-CMG	XDL-based Communication Macro Generator.
XDL	Xilinx Desing Language.
XRTC	Xilinx Radiation Test Consortium.

Bibliography

[1] M. Abramovici, J. M. Emmert, and C. E. Stroud. "Roving STARs: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems". In: *Proceedings - NASA/DoD Conference on Evolvable Hardware, EH* (2001), pp. 73–92. DOI: 10.1109/EH.2001.937949.

[2] P. Abusaidi, M. Fernandez, and P. Abusaidi. *Virtex-6 FPGA Routing Optimization Design Techniques*. Tech. rep. 2010, pp. 1–9. URL: http://www.xilinx.com/support/documentation/white_papers/wp381_V6_Routing_Optimization.pdf.

[3] AEROFLEX GAISLER AB. *SPWRTC Development Unit*. Tech. rep. 2008, pp. 1–47. URL: http://www.gaisler.com/doc/GR-SPWRTC-DEV_User_Manual.pdf.

[4] G. Allen, G. Swift, and C. Carmichael. *Virtex-4 VQ static SEU characterization summary*. Tech. rep. 2008. URL: http://trs-new.jpl.nasa.gov/dspace/handle/2014/40768.

[5] N. Battezzati, L. Sterpone, and M. Violante. *Reconfigurable field programmable gate arrays for mission-critical applications*. Springer, 2011, p. 220. DOI: 10.1007/978-1-4419-7595-9.

[6] L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich. "Test Strategies for Reliable Runtime Reconfigurable Architectures". In: *IEEE Transactions on Computers* 62.8 (Aug. 2013), pp. 1494–1507. DOI: 10.1109/TC.2013.53.

[7] R. C. Baumann. "Radiation-induced soft errors in advanced semiconductor technologies". In: *IEEE Transactions on Device and Materials Reliability* 5.3 (Sept. 2005), pp. 305–315. DOI: 10.1109/TDMR.2005.853449.

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

[9] C. Beckhoff, D. Koch, and J. Torresen. "Go Ahead: A Partial Reconfiguration Framework". In: *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. 2012, pp. 37–44. DOI: 10.1109/FCCM.2012.17.

[10] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, M. Violante, A. Paccagnella, M. Rebaudengo, M. S. Reorda, and P. Zambolin. "Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA". In: *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*. Vol. 1. 2004, 584–589 Vol.1. DOI: 10.1109/DATE.2004.1268908.

[11] C. Bernardeschi, L. Cassano, M. G. C. a. Cimino, and A. Domenici. "GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs". In: *Journal of Systems Architecture* 59.10 (2013), pp. 1243–1254. DOI: 10.1016/j.sysarc.2013.10.006.

[12] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999, p. 23. DOI: 10.1007/978-1-4615-5145-4.

[13] V. Betz and J. Rose. "Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size". In: *Custom Integrated Circuits Conference, 1997., Proceedings of the IEEE 1997*. 1997, pp. 551–554. DOI: 10.1109/CICC.1997.606687.

[14] V. Betz and J. Rose. "VPR: A New Packing, Placement and Routing Tool for FPGA Research". In: *Field-Programmable Logic and Applications*. Vol. 1304. 1997, pp. 1–10. DOI: 10.1007/3-540-63465-7_226.

[15] D. Bhatia. *Field-Programmable Gate Arrays*. 1996. DOI: 10.1155/1996/87608.

[16] Bielefeld University. *RAPTOR Family, Modular Rapid Prototyping*. URL: http://www.ks.cit-ec.uni-bielefeld.de/projects/raptor-family.html.

[17] B. Blodget, C. Bobda, M. Huebner, and A. Niyonkuru. "Partial and Dynamically Reconfiguration of Xilinx Virtex-II FPGAs". In: *Field Programmable Logic and Application*. Springer, 2004, pp. 801–810. DOI: 10.1007/978-3-540-30117-2_81.

[18] F. Brosser, E. Milh, V. Geijer, and P. Larsson-Edefors. "Assessing Scrubbing Techniques for Xilinx SRAM-based FPGAs in Space Applications". In: *2014 International Conference on Field-Programmable Technology (FPT)* (Dec. 2014), pp. 2–5. DOI: 10.1109/FPT.2014.7082803.

[19] S. Brown, J. Rose, and Z. Vranesic. "A detailed router for field-programmable gate arrays". In: *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. IEEE Comput. Soc. Press, pp. 382–385. DOI: 10.1109/ICCAD.1990.129931.

[20] F. Bubenhagen, B. Fiethe, T. Lange, H. Michalik, and H. Michel. "Reconfigurable platforms for Data Processing on scientific space instruments". In: *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*. 2013, pp. 63–70. DOI: 10.1109/AHS.2013.6604227.

[21] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. De Wit. "A dynamic reconfiguration run-time system". In: *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No.97TB100186)*. IEEE Comput. Soc, 1997, pp. 66–75. DOI: 10.1109/FPGA.1997.624606.

[22] D. Chen, J. Cong, and P. Pan. "FPGA Design Automation: A Survey". In: *Electronic Design Automation* 1.3 (2006), pp. 195–330. DOI: 10.1561/1000000003.

[23] X. Cheng and Xu. "Heterogeneous Multi-processor SoC: An Emerging Paradigm of Embedded System Design and Its Challenges". In: *Proceedings of the Second international conference on Embedded Software and Systems*. Springer-Verlag, 2005, pp. 3–3. DOI: 10.1007/11599555_3.

[24] A. Cilardo. "New Techniques and Tools for Application-Dependent Testing of FPGA-Based Components". In: *IEEE Transactions on Industrial Informatics* 11.1 (Feb. 2015), pp. 94–103. DOI: 10.1109/TII.2014.2370532.

[25] C. Claus, Z. Bin, M. Hubner, C. Schmutzler, W. Stechele, and J. Becker. "An XDL-based busmacro generator for customizable communication interfaces for dynamically and partially reconfigurable systems". In: *Workshop on Reconfigurable Computing Education at ISVLSI*. 2007. URL: http://xputers.informatik.uni-kl.de/RCeducation07/busmacro_RC_education_claus.pdf.

[26] J.-P. Colinge. *Silicon-on-Insulator Technology: Materials to VLSI*. Springer US, 1997, p. 272. ISBN: 1475726112.

[27] K. Compton and S. Hauck. "Reconfigurable computing: a survey of systems and software". In: *ACM Comput. Surv.* 34.2 (2002), pp. 171–210. DOI: http://doi.acm.org/10.1145/508352.508353.

[28] E. Cooperation and F. O. R. S. Standardization. "Space engineering: Communications guidelines". In: *System* April (2004), pp. 1–174.

[29] J. D. Couch. "Applications of TORC : An Open Toolkit for Reconfigurable Computing by". PhD thesis. 2011.

[30] D. Cozzi, J. Hagemeyer, S. Korf, D. Jungewelter, and M. Porrmann. "DRPM User Manual v6.1". In: (2014), p. 283.

[31] S. Craw. *Manhattan Distance*. U.S. National Institute of Standards and Technology. 2010. DOI: 10.1007/978-0-387-30164-8_506.

[32] M. Darvishi, S. Member, Y. Audet, Y. Blaquière, C. Thibeault, and S. Member. "Circuit Level Modeling of Extra Combinational Delays in SRAM FPGAs Due to Transient Ionizing Radiation". In: *IEEE Nuclear and Space Radiation Effects Conference* (2014).

[33] S. Dhingra, D. Milton, and C. E. Stroud. "BIST for Logic and Memory Resources in Virtex-4 FPGAs". In: *Proc. IEEE North Atlantic Test Workship*. 2006.

[34] D. Dye. "Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite". 2012. URL: http://www.xilinx.com/support/documentation/white_papers/wp374_Partial_Reconfig_Xilinx_FPGAs.pdf.

[35] ESA. *Solar Orbiter*. URL: http://sci.esa.int/solar-orbiter/.

[36] ESA. *Space product assurance - Techniques for radiation effects mitigation in ASICs and FPGAs handbook*. Ed. by R. ESA-ESTEC and S. Division. ECSS Secretariat ESA-ESTEC, 2016. URL: http://microelectronics.esa.int/asic/ECSS-Q-HB-60-02A1September2016.pdf.

[37] European Space Agency. *DRPM: Dynamically Reconfigurable Processing Module*. URL: http://www.esa.int/Our_Activities/Space_Engineering_Technology/Onboard_Data_Processing/DRPM_-_Dynamically_Reconfigurable_Processing_Module.

[38] R. Ferguson and R. Tate. "Use of field programmable gate array technology in future space avionics". In: *Proceedings of the 24th Digital Avionics Systems Conference (DASC 2005)*. Vol. 2. 2005. DOI: 10.1109/DASC.2005.1563418.

[39] T. FLATLEY. "Advanced Hybrid On-Board Science Data Processor - SpaceCube 2.0". In: *Earth Science Technology Forum* (2010). URL: https://esto.nasa.gov/conferences/estf2011/papers/Flatley_ESTF2011.pdf.

[40] L. Fossati and J. Ilstad. "The future of embedded systems at ESA: Towards adaptability and reconfigurability". In: *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2011, pp. 113–120. DOI: 10.1109/AHS.2011.5963924.

[41] V.-i. Fpgas, N. J. Steiner, P. Athanas, M. Jones, and C. Patterson. *A Standalone Wire Database for Routing and Tracing in Xilinx A Standalone Wire Database for Routing and Tracing in Xilinx Virtex , Virtex-E , and Virtex-II FPGAs*. 2002. URL: %7Bhttp://scholar.lib.vt.edu/theses/available/etd-09112002-143335/unrestricted/thesis.pdf%7D.

[42] F. Fu, S. Sun, X. Hu, J. Song, J. Wang, and M. Yu. "MMPI: A flexible and efficient multiprocessor message passing interface for NoC-based MPSoC". In: *23rd IEEE International SOC Conference*. 2010, pp. 359–362. DOI: 10.1109/SOCC.2010.5784695.

[43] J. Gebelein. *An approach to system-wide fault tolerance for FPGAs*. 2009. URL: https://twiki.cern.ch/twiki/pub/Sandbox/RadHardFPGA/WFIFT_P11_Gebelein.pdf.

[44] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann. "Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs". In: *Proc. of the Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. CSREA Press, 2007.

[45] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann. "INDRA - Integrated Design Flow for Reconfigurable Architectures". In: *International Journal of Embedded Systems*. 2007, pp. 1–2.

[46] J. Holt, A. Agarwal, S. Brehmer, M. Domeika, P. Griffin, and F. Schirrmeister. "Software Standards for the Multicore Era". In: *IEEE Micro* 29.3 (2009), pp. 40–51. DOI: 10.1109/MM.2009.48.

[47] K.-Y. Hsieh, Y.-C. Liu, P.-W. Wu, S.-W. Chang, and J. K. Lee. "Enabling Streaming Remoting on Embedded Dual-Core Processors". In: *2008 37th International Conference on Parallel Processing*. 2008, pp. 35–42. DOI: 10.1109/ICPP.2008.32.

[48] W. Huang, F. Meyer, N. Park, and F. Lombardi. "Testing memory modules in SRAM-based configurable FPGAs". In: *Proceedings. International Workshop on Memory Technology, Design and Testing (Cat. NO.97TB100159)*. 1997, pp. 79–86. DOI: 10.1109/MTDT.1997.619399.

[49] M. Hubner, K. Paulsson, and J. Becker. "Parallel and Flexible Multiprocessor System-On-Chip for Adaptive Automotive Applications based on Xilinx MicroBlaze Soft-Cores". In: *19th IEEE International Parallel and Distributed Processing Symposium*. Vol. 00. c. IEEE. 2005, 149a–149a. DOI: 10.1109/IPDPS.2005.325.

[50] K. Huey. *Xilinx Virtex-5QV Update and Space Roadmap*. Tech. rep. 2016. URL: https://indico.esa.int/indico/event/130/session/11/contribution/54/material/slides/0.pdf.

[51] E. Hung, F. Eslami, and S. J. E. Wilton. "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices". In: *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2013, pp. 45–52. DOI: 10.1109/FCCM.2013.40.

[52] S. H. Hung, W. L. Yang, and C. H. Tu. “Designing and implementing a portable, efficient inter-core communication scheme for embedded multicore platforms”. In: *Proceedings - 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2010*. 2010, pp. 303–308. DOI: 10.1109/RTCSA.2010.17.

[53] B. J. Hussein and G. Swift. *Single-Event Upsets*. Tech. rep. Xilinx, 2015, pp. 1–11. URL: http://www.xilinx.com/support/documentation/white_papers/wp395-Mitigating-SEUs.pdf.

[54] *INDRA*. URL: https://sourceforge.net/projects/indrafpga/

[55] H. N. Institute. *RAPTOR Modular Rapid Prototyping*. URL: https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Entwurf_paralleler_Systeme/Lehre/RAPTOR_en.pdf.

[56] X. Iturbe, K. Benkrid, A. T. Erdogan, T. Arslan, M. Azkarate, I. Martinez, and A. Perez. “R3TOS: A reliable reconfigurable real-time operating system”. In: *2010 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2010, pp. 99–104. DOI: 10.1109/AHS.2010.5546274.

[57] A. M. Jallad and L. B. Mohammad. “Comparative analysis of middleware for multi-processor system-on-chip (MPSoC)”. In: *2013 9th International Conference on Innovations in Information Technology (IIT)*. 2013, pp. 113–117. DOI: 10.1109/Innovations.2013.6544403.

[58] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert. “REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems”. In: *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. 2005, 151b–151b. DOI: 10.1109/IPDPS.2005.380.

[59] H. Kalte, M. Porrmann, and U. Rückert. “A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs”. In: *Proceedings of the IEEE Workshop Heterogeneous Reconfigurable Systems on Chip*. 2002.

[60] D. Koch, C. Beckhoff, and J. Teich. “RECOBUS-BUILDER - A NOVEL TOOL AND TECHNIQUE TO BUILD STATICALLY AND DYNAMICALLY RECONFIGURABLE SYSTEMS FOR FPGAS Dirk Koch , Christian Beckhoff , and J Â¨ Hardware / Software Co-Design , Department of Computer Science”. In: *Fpl*. 2008, pp. 119–124. ISBN: 9781424419616.

[61] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Rückert. “Design optimizations for tiled partially reconfigurable systems”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19.6 (June 2011), pp. 1048–1061. DOI: 10.1109/TVLSI.2010.2044902.

[62] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. "RAMP Blue: A Message-Passing Manycore System in FPGAs". In: *Field Programmable Logic and Applications, 2007. FPL '07. International Conference on*. 2007, pp. 54–61. DOI: 10.1109/FPL.2007.4380625.

[63] R. V. Kshirsagar and S. Sharma. "Difference-based partial reconfiguration". In: *International Journal of Advances in Engineering & Technology*. Vol. 1. 2. 2011, pp. 194–197. URL: http://topics-dat175-anand-tauseef.googlecode.com/svn-history/r78/trunk/papers/rtr/differencebasedreconfiguration.pdf.

[64] I. Kuon, R. Tessier, and J. Rose. "FPGA Architecture: Survey and Challenges". In: *Foundations and Trends® in Electronic Design Automation* 2.2 (2007), pp. 135–253. DOI: 10.1561/1000000005.

[65] C. Lavin, M. Padilla, S. Ghosh, B. Nelson, B. Hutchings, and M. Wirthlin. "Using Hard Macros to Reduce FPGA Compilation Time". In: *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. 2010, pp. 438–441. DOI: 10.1109/FPL.2010.90.

[66] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings. "RapidSmith : Do-It-Yourself CAD Tools for Xilinx FPGAs". In: *Language*. Xdl. 2011, pp. 349–355. DOI: 10.1109/FPL.2011.69.

[67] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, B. Hutchings, and M. Wirthlin. *RAPID SMITH: Technical Report and Documentation*. Tech. rep. 2014. URL: http://rapidsmith.sourceforge.net/doc/TechReportAndDocumentation.pdf?format=raw.

[68] T. Lead, G. Allen, R. E. Group, D. Sheldon, T. Staff, and J. Propulsion. "Europa Clipper Comprehensive FPGA Risk Reduction". In: (2014). URL: https://solarsystem.nasa.gov/europa/docs/EuropaClipperComprehensiveFPGARiskReduction_smg_gra_(c).pdf.

[69] G. G. Lemieux, G. G. Lemieux, and S. D. Brown. "A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays". In: *IN PROC. ACM/SIGDA PHYSICAL DESIGN WORKSHOP, LAKE ARROWHEAD, CA* (1993), pp. 215–226. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.1538.

[70] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke. "The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs". In: *IEEE Transactions on Device and Materials Reliability* 5.3 (Sept. 2005), pp. 317–328. DOI: 10.1109/TDMR.2005.854207.

[71] A. Lesea. *Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits*. Tech. rep. Xilinx, 2011, pp. 1–8. DOI: XilinxWP286(v1.1).

[72] J.-j. Li, S.-c. Wang, P.-c. Hsu, P.-y. Chen, and J. K. Lee. "A Multi-core Software API for Embedded MPSoC Environments". In: *Methods and Tools of Parallel Programming Multicomputers*. Ed. by C.-H. Hsu and V. Malyshkin. Vol. 6083. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 40–50. ISBN: 978-3-642-14821-7.

[73] P. Mahr, C. Lörchner, H. Ishebabi, and C. Bobda. "SoC-MPI: A Flexible Message Passing Library for Multiprocessor Systems-on-Chips". In: *2008 International Conference on Reconfigurable Computing and FPGAs*. 2008, pp. 187–192. DOI: 10.1109/ReConFig.2008.27.

[74] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. "Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs". In: *Proc. 12th Conf. on Field-Programmable Logic and Applications (FPL)*. FPL '02. Springer-Verlag, 2002, pp. 795–805. DOI: 10.1007/3-540-46117-5_82.

[75] L. Matilainen, E. Salminen, T. D. Hämäläinen, and M. Hämäläinen. "Multicore Communications API (MCAPI) implementation on an FPGA multiprocessor". In: *Proceedings - 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2011*. 2011, pp. 286–293. DOI: 10.1109/SAMOS.2011.6045473.

[76] A. S. McEwen, E. M. Eliason, J. W. Bergstrom, N. T. Bridges, C. J. Hansen, W. A. Delamere, J. A. Grant, V. C. Gulick, K. E. Herkenhoff, L. Keszthelyi, R. L. Kirk, M. T. Mellon, et al. "Mars reconnaissance orbiter's high resolution imaging science experiment (HiRISE)". In: *Journal of Geophysical Research E: Planets* 112.5 (2007), pp. 1–40. DOI: 10.1029/2005JE002605.

[77] L. McMurchie and C. Ebeling. "PathFinder". In: *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays - FPGA '95*. ACM Press, 1995, pp. 111–117. DOI: 10.1145/201310.201328.

[78] H. Michel, F. Bubenhagen, B. Fiethe, H. Michalik, B. Osterloh, W. Sullivan, A. Wishart, J. Ilstad, and S. A. Habinc. "AMBA to SoCWire network on Chip bridge as a backbone for a Dynamic Reconfigurable Processing unit". In: *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2011, pp. 227–233. DOI: 10.1109/AHS.2011.5963941.

[79] S. Miura, T. Hanawa, T. Boku, and M. Sato. "XMCAPI: Inter-core communication interface on multi-chip embedded systems". In: *Proceedings - 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing, EUC 2011*. 2011, pp. 397–402. DOI: 10.1109/EUC.2011.78.

[80] N. Montealegre, D. Merodio, A. Fernandez, and P. Armbruster. "In-flight reconfigurable FPGA-based space systems". In: *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2015, pp. 1–8. DOI: 10.1109/AHS.2015.7231177.

[81] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. "GridRPC: A remote procedure call API for Grid computing". In: *Lecture notes in computer science*. Vol. 2536. GRID '02. Springer-Verlag, 2002, pp. 274–278.

[82] NASA. *MER*. URL: http://mars.nasa.gov/mer/overview/.

[83] 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.

[84] J. Ou and V. K. Prasanna. "Energy Efficient Hardware-Software Co-Synthesis Using Reconfigurable Hardware". PhD thesis. 2009, p. 202. ISBN: 978-0-542-91214-6.

[85] M. Renovell, J. Portal, J. Figuras, and Y. Zorian. "Minimizing the number of test configurations for different FPGA families". In: *Proceedings Eighth Asian Test Symposium (ATS'99)*. 1999, pp. 363–368. DOI: 10.1109/ATS.1999.810776.

[86] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. "Testing the interconnect of RAM-based FPGAs". In: *IEEE Design and Test of Computers* 15.March (Jan. 1998), pp. 45–50. DOI: 10.1109/54.655182.

[87] F. Rittner, R. Glein, T. Kolb, and B. Bernard. "Broadband FPGA payload processing in a harsh radiation environment". In: *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2014, pp. 151–158. DOI: 10.1109/AHS.2014.6880171.

[88] P. Roche, J.-L. Autran, G. Gasiot, and D. Munteanu. "Technology Downscaling Worsening Radiation Effects in Bulk: SOI to the Rescue". In: *International Electron Devices Meeting (IEDM)*. 2013, pp. 31.1.1–31.1.4. DOI: 10.5772/2600.

[89] J. Romoth, D. Jungewelter, J. Hagemeyer, M. Porrmann, and U. Ruckert. "Optimizing inter-FPGA communication by automatic channel adaptation". In: *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*. 2012, pp. 1–7. DOI: 10.1109/ReConFig.2012.6416767.

[90] J. Rose. "Parallel global routing for standard cells". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9.10 (1990), pp. 1085–1095. DOI: 10.1109/43.62733.

[91] J. Rose, R. J. Francis, D. Lewis, and P. Chow. "Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency". In: *IEEE Journal of Solid-State Circuits* 25.5 (1990), pp. 1217–1225. DOI: 10.1109/4.62145.

[92] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. "The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing". In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA '12*. ACM Press, 2012, p. 77. DOI: 10.1145/2145694.2145708.

[93] RTEMS. "Getting Started with RTEMS". In: October (2003). URL: https://docs.rtems.org/releases/rtemsdocs-4.6.5/share/rtems/html/index.html.

[94] S. Russell, M. Vesely, C. Graham, and M. Petkovic. *Progress Towards FedSat 2001 A'stralian Space Odyssey*. 1999. URL: http://digitalcommons.usu.edu/smallsat/1999/all1999/69.

[95] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 2009. DOI: 10.1017/S0269888900007724.

[96] J. S and V. K. Agrawal. "Detection and Diagnosis of Faults in the Routing Resources of a SRAM based FPGAs". In: *International Journal of Computer Applications* 53.13 (Sept. 2012), pp. 18–22. DOI: 10.5120/8481-2421.

[97] D. Sabena, L. Sterpone, M. Scholzel, 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.

[98] K. Sebastian. "Erhöhung der Zuverlässigkeit FPGA-basierter Systeme durch dynamisch partielle Rekonfiguration". PhD thesis. 2016.

[99] B. Senouci, A. Kouadri.M, F. Rousseau, and F. Petrot. "Multi-CPU/FPGA Platform Based Heterogeneous Multiprocessor Prototyping: New Challenges for Embedded Software Designers". In: *2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*. 2008, pp. 41–47. DOI: 10.1109/RSP.2008.27.

[100] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam. "Mitigation of Radiation Effects in SRAM-Based FPGAs". In: *ACM Computing Surveys* 47.2 (2015), pp. 1–34. DOI: 10.1145/2671181.

[101] J. Silcock, J. Silcock, and A. Goscinski. *Message Passing, Remote Procedure Calls and Distributed Shared Memory as Communication Paradigms for Distributed Systems*. Vol. 28. Technical reports: Computing series. Deakin University, School of Computing and Mathematics, 1995. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.2490.

[102] J. Smith, T. Xia, and C. Stroud. "An automated BIST architecture for testing and diagnosing FPGA interconnect faults". In: *Journal of Electronic Testing: Theory and Applications (JETTA)* 22.3 (2006), pp. 239–253. DOI: 10.1007/s10836-006-9319-7.

[103] Softing Automotive Electronics GmbH. *CAN-AC2-PCI*. Tech. rep. URL: http://automotive.softing.com/fileadmin/sof-files/pdf/en/ae/data_sheets/vci/Softing-DB_CANAC2pci_E.pdf.

[104] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood. "OpenPR: An open-source partial-reconfiguration toolkit for xilinx FPGAs". In: *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. Xdl. 2011, pp. 228–235. DOI: 10.1109/IPDPS.2011.146.

[105] A. Sreeramareddy, J. G. Josiah, A. Akoglu, and A. Stoica. "SCARS: Scalable Self-Configurable Architecture for Reusable Space Systems". In: *2008 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2008, pp. 204–210. DOI: 10.1109/AHS.2008.77.

[106] STAR-Dundee. *SpaceWire Brick*. URL: https://www.star-dundee.com/sites/default/files/SpaceWire%20Brick.pdf.

[107] STAR-Dundee. *STAR Fire*. URL: https://www.star-dundee.com/products/star-fire.

[108] N. Steiner and P. Athanas. "An Alternate Wire Database for Xilinx FPGAs". In: *Proceedings - 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004*. IEEE Computer Society, 2004, pp. 336–337. DOI: 10.1109/FCCM.2004.13.

[109] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French. "Torc: Towards an Open-Source Tool Flow Neil". In: *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '11*. FPGA '11. ACM, 2011, p. 41. DOI: 10.1145/1950413.1950425.

[110] J. O. Stiegler and L. K. Mansur. "Radiation Effects in Structural Materials". In: *Proceedings of the 9th Workshop on Electronics for LHC Experiments*. Vol. V. 14. 1979, pp. 191–194. DOI: 10.1146/annurev.ms.09.080179.002201.

[111] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici. "BIST-based diagnosis of FPGA interconnect". In: *Proceedings. International Test Conference*. 2002, pp. 618–627. DOI: 10.1109/TEST.2002.1041813.

[112] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. "Built-in self-test of FPGA interconnect". In: *IEEE International Test Conference (TC)*. 1998, pp. 404–411. DOI: 10.1109/TEST.1998.743180.

[113] X. S. X. Sun, J. X. J. Xu, B. C. B. Chan, and P. Trouborst. "Novel technique for built-in self-test of FPGA interconnects". In: *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159)*. International Test Conference. 2000, pp. 795–803. DOI: 10.1109/TEST.2000.894276.

[114] M. Tahoori. "Application-dependent testing of FPGAs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.9 (2006), pp. 1024–1033. DOI: 10.1109/TVLSI.2006.884053.

[115] J. B. Ã. Team. *AutoIt Online Documentation*. URL: https://www.autoitscript.com/autoit3/docs/.

[116] The Qt Company. *Qt Creator*. URL: https://www.qt.io/ide/.

[117] The University of Texas. *Gravity Recovery And Climate Experiment (GRACE)*. URL: http://www.csr.utexas.edu/grace/.

[118] I. A. Troxel, M. Fehringer, and M. T. Chenoweth. "Achieving Multipurpose Space Imaging with the ARTEMIS Reconfigurable Payload Processor". In: *2008 IEEE Aerospace Conference*. IEEE, 2008, pp. 1–8. DOI: 10.1109/AERO.2008.4526469.

[119] Vector Informatik. *CANalyzer*. URL: http://vector.com/vi_canalyzer_en.html.

[120] B. White and B. Nelson. "Tincr - A custom CAD tool framework for Vivado". In: *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. 2014, pp. 1–6. DOI: 10.1109/ReConFig.2014.7032560.

[121] D. White. *Considerations surrounding single event effects in FPGAs, ASICs, and processors*. Tech. rep. 2011. URL: http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf.

[122] Xilinx Inc. "7 Series FPGAs Clocking Resources". In: (2015), pp. 1–112. URL: http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[123] Xilinx Inc. *7 Series FPGAs Configuration: User Guide*. 2015. URL: http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf.

[124] Xilinx Inc. *7 Series FPGAs Overview*. 2015, pp. 1–112. URL: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[125] Xilinx Inc. "Command Line Tools". In: 628 (2012), pp. 1–419. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/devref.pdf.

[126] Xilinx Inc. *Device Reliability Report*. Second Hal. Sept. 2015, pp. 1–110. URL: http://www.xilinx.com/support/documentation/user_guides/ug116.pdf.

[127] Xilinx Inc. *Early access partial reconfiguration user guide*. Tech. rep. 2008.

[128] Xilinx Inc. *FPGA Editor Overview*. URL: http://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/fpga_editor.htm.

[129] Xilinx Inc. *ISE In-Depth Tutorial (UG695)*. v14.1. 2012. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf.

[130] Xilinx Inc. *OS and Libraries Document Collection*. 2009. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/oslib_rm.pdf.

[131] Xilinx Inc. *Partial Reconfiguration User Guide*. Apr. 2011. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug702.pdf.

[132] Xilinx Inc. *Power Consumption at 45nm (WP298)*. Tech. rep. 2016. URL: http://www.xilinx.com/support/documentation/white_papers/wp298.pdf.

[133] Xilinx Inc. *QPro Virtex 2 .5V Radiation-Hardened FPGAs*. Tech. rep. 2010, pp. 1–17. URL: http://www.xilinx.com/support/documentation/data_sheets/ds028.pdf.

[134] Xilinx Inc. *QPRO XQR4000XL Radiation Hardened FPGAs*. Tech. rep. 2000, pp. 1–20. URL: http://www.xilinx.com/support/documentation/data_sheets/ds071.pdf.

[135] Xilinx Inc. *Radiation-Hardened, Space-Grade Virtex-5QV Family Overview*. Xilinx. Nov. 2014. URL: http://www.xilinx.com/support/documentation/data_sheets/ds192_V5QV_Device_Overview.pdf.

[136] Xilinx Inc. *Space-Grade Virtex-4QV Family Overview*. Nov. 2014. URL: http://www.xilinx.com/support/documentation/data_sheets/ds653.pdf.

[137] Xilinx Inc. *Spartan-6 Family Overview*. 2011, p. 11. URL: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf.

[138] Xilinx Inc. "Spartan-6 FPGA Clocking Resources User Guide". In: 382 (2012), pp. 1–118. URL: http://www.xilinx.com/support/documentation/user_guides/ug382.pdf.

[139] Xilinx Inc. *Spartan-6 FPGA Configuration User Guide (UG380)*. 2014. URL: http://www.xilinx.com/support/documentation/user_guides/ug380.pdf.

[140] Xilinx Inc. "UltraScale Architecture and Product Overview". 2016. URL: http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[141] Xilinx Inc. *UltraScale Architecture Configuration User Guide (UG570)*. v1.6. 2015. URL: http://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf.

[142] Xilinx Inc. *Virtex 4 FPGA Configuration User Guide*. 2009. URL: http://www.xilinx.com/support/documentation/user_guides/ug071.pdf.

[143] Xilinx Inc. "Virtex Series Configuration Architecture User Guide". In: (2004). URL: http://www.xilinx.com/support/documentation/application_notes/xapp151.pdf.

[144] Xilinx Inc. *Virtex-4 Family Overview*. Tech. rep. 2010, pp. 1–9. URL: http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf.

[145] Xilinx Inc. "Virtex-4 FPGA User Guide". In: (2008). URL: http://www.xilinx.com/support/documentation/user_guides/ug070.pdf.

[146] Xilinx Inc. "Virtex-5 Family Overview". In: *Xilinx Inc.* Vol. 5.0. DS100. Xilinx. 2009, pp. 1–13. URL: http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf.

[147] Xilinx Inc. *Virtex-5 FPGA Configuration User Guide*. 2012. URL: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

[148] Xilinx Inc. "Virtex-5 FPGA User Guide". In: (2012), pp. 1–385. URL: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[149] Xilinx Inc. *Virtex-6 Family Overview*. Xilinx. 2010.

[150] Xilinx Inc. "Virtex-6 FPGA Clocking Resources User Guide". In: (2012). URL: http://www.xilinx.com/support/documentation/user_guides/ug362.pdf.

[151] Xilinx Inc. *Virtex-6 FPGA Configuration, User Guide*. 2015. URL: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[152] Xilinx Inc. *Virtex-E 1.8V Field Programmable Gate Arrays: Complete Data Sheet (All Four Modules)*. 2014. URL: http://www.xilinx.com/support/documentation/data_sheets/ds022.pdf.

[153] Xilinx Inc. *Vivado Design Suite Properties Reference Guide (v2014.2)*. Nov. 2014, pp. 1–419. URL: http://china.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/ug912-vivado-properties.pdf.

[154] Xilinx Inc. "Vivado Design Suite Tcl Command Reference Guide". In: 835 (2013), pp. 1–977.

[155] Xilinx Inc. "Xcell Journal". In: 50 (2004), p. 116. URL: http://www.ncbi.nlm.nih.gov/pubmed/24371147.

[156] Xilinx Inc. "Xcell Journal". In: (2011), pp. 1–68.

[157] Xilinx Inc. "Xcell Journal". In: 84 (2013), p. 68. URL: http://www.xilinx.com/publications/archives/xcell/Xcell84.pdf.

[158] Xilinx Inc. "Xcell Journal". 2014. URL: http://www.xilinx.com/publications/archives/xcell/Xcell86.pdf.

[159] Xilinx Inc. "Xcell Journal". In: 90 (2015). URL: http://www.xilinx.com/publications/archives/xcell/Xcell90.pdf.

[160] Xilinx Inc. *Xilinx, Power Consumption in 65nm FPGAs*. Tech. rep. 2. 2007. URL: http://www.xilinx.com/support/documentation/white_papers/wp246.pdf.

[161] Xilinx Inc. *Zynq UltraScale+ MPSoC*. URL: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

[162] Xilinx Inc. *Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide*. 2016. URL: http://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf.

[163] Xilinx Inc. "Zynq-7000 All Programmable SoC Overview". In: 190 (2013), pp. 1–21. URL: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[164] Xilinx Inc. *Zynq-7000 AP SoC*. Tech. rep. 2015. URL: https://www.xilinx.com/support/documentation/product-briefs/zynq-7000-product-brief.pdf.

[165] J. Yao, B. Dixon, C. Stroud, and V. Nelson. "System-level built-in self-test of global routing resources in virtex-4 FPGAs". In: *2009 IEEE International Symposium on Sustainable Systems and Technology, ISSST 2009*. 2009, pp. 29–33. DOI: 10.1109/SSST.2009.4806782.

[166] Z. Zhang, Z. Wen, L. Chen, F. Zhang, and T. Zhou. "A Novel BIST Approach for Testing Logic Resources Using Hard Macro". In: (2008), pp. 379–381. DOI: 10.1109/ICNNSP.2008.4590376.

[167] J. F. Ziegler. "Terrestrial cosmic ray intensities". English. In: *IBM Journal of Research and Development* 42.1 (Jan. 1998), pp. 117–140. DOI: 10.1147/rd.421.0117.

Advised Thesis

[168] D. Kleibrink. “Integration von SpaceFibre-Schnittstellen in FPGA-basierte Satellitensysteme”. Diplomarbeit. Universität Paderborn, 2014.

[169] F. Mascolo. “Design and implementation of a routing algorithm to maximize test coverage of permanent faults in FPGAs”. Master Thesis. Università di Pisa, 2015. URL: https://etd.adm.unipi.it/t/etd-09062015-235709/.

[170] L. Santangelo. “Viv2XDL: a bridge between Vivado and XDL based software”. Master Thesis. 2014. URL: https://etd.adm.unipi.it/t/etd-09012014-224107/.

[171] T. Schlüssler. “Realiserung einer Konfigurationsinfrastruktur für fehlertolerante FPGA-basierte Satellitensysteme”. Bachelor Thesis. Bielefeld University, 2013.

[172] D. Sorrenti. “Exploiting Partial Dynamic Reconfiguration for On-Line On-Demand Detection of Permanent Faults in SRAM-based FPGAs”. Master Thesis. Università di Pisa, 2014. URL: https://etd.adm.unipi.it/t/etd-03282014-113705/.

[173] M. Vorfeld. “Kommunikationslösungen für Datenübertragung in FPGA-basierten Satellitensystemen”. Master Thesis. Bielefeld University, 2014.

Author's Publications

[174] L. Cassano, D. Cozzi, D. Jungewelter, S. Korf, J. Hagemeyer, M. Porrmann, and C. Bernardeschi. "An inter-processor communication interface for data-flow centric heterogeneous embedded multiprocessor systems". In: *Design Technology of Integrated Systems In Nanoscale Era (DTIS), 2014 9th IEEE International Conference On*. 2014, pp. 1–6. DOI: 10.1109/DTIS.2014.6850669.

[175] L. Cassano, D. Cozzi, S. Korf, J. Hagemeyer, M. Porrmann, L. Sterpone, and P. Torino. "On-Line Testing of Permanent Radiation Effects in Reconfigurable Systems". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 2013, pp. 717–720. DOI: 10.7873/DATE.2013.154.

[176] D. Cozzi. "Homogeneous Communication Router for Xilinx FPGAs". PhD thesis. Politecnico di Milano, 2011. URL: http://hdl.handle.net/10589/12964.

[177] D. Cozzi, D. Jungewelter, S. Korf, J. Hagemeyer, and M. Porrmann. *Dynamically Reconfigurable Hardware for Resource Efficiency and Fault Tolerance in Space Applications*. 2014. URL: https://indico.esa.int/indico/event/59/session/5/contribution/8.

[178] D. Cozzi, S. Korf, L. Cassano, J. Hagemeyer, A. Domenici, C. Bernardeschi, M. Porrmann, and L. Sterpone. "OLT(RE)2 : an On-Line on-demand Testing approach for permanent Radiation Effects in REconfigurable systems". In: *IEEE Transactions on Emerging Topics in Computing* (2016), pp. 1–13. DOI: 10.1109/TETC.2016.2586195.

[179] D. Cozzi and K. Sebastian. *OLT(RE)2: A Tool Flow for Mitigation of Permanent Faults in Reconfigurable Systems*. Tech. rep. March. European Space Research and Technology Centre (ESTEC), 2016. URL: https://indico.esa.int/indico/event/130/session/7/contribution/43.

[180] J. Hagemeyer, A. Hilgenstein, D. Jungewelter, D. Cozzi, C. Felicetti, U. Rueckert, S. Korf, M. Koester, F. Margaglia, M. Porrmann, F. Dittmann, M. Ditze, et al. "A scalable platform for run-time reconfigurable satellite payload processing". In: *Proceedings of the 2012 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2012*. 2012, pp. 9–16. DOI: 10.1109/AHS.2012.6268642.

[181] J. Hagemeyer, D. Jungewelter, D. Cozzi, S. Korf, and M. Porrmann. *DRPM architecture overview*. 2012. URL: https://amstel.estec.esa.int/tecedm/website/conferences/sefuw/d2_p6_SEFUW_2012_drpm_hagemeyer_final.pdf.

[182] D. Jungewelter, D. Cozzi, D. Kleibrink, S. Korf, J. Hagemeyer, M. Porrmann, and J. Ilstad. "AXI-based SpaceFibre IP core implementation". In: *2014 International SpaceWire Conference (SpaceWire)*. 2014, pp. 1–6. DOI: 10.1109/SpaceWire.2014.6936258.

[183] S. Korf, D. Cozzi, D. Jungewelter, J. Hagemeyer, M. Porrmann, and J. Ilstad. "Leveraging dynamic reconfiguration to increase fault-tolerance in FPGA-based satellite systems". In: *2014 Design, Automation and Test in Europe (DATE)*. Vol. 62. 8. 2013, p. 2013. URL: https://www.date-conference.com/date14/files/file/date14/ubooth/2602.pdf.

[184] S. Korf, D. Cozzi, M. Koester, J. Hagemeyer, M. Porrmann, U. Rückert, and M. D. Santambrogio. "Automatic HDL-Based Generation of Homogeneous Hard Macros for FPGAs". In: *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. 2011, pp. 125–132. DOI: 10.1109/FCCM.2011.36.

[185] S. Korf, G. Sievers, J. Ax, D. Cozzi, T. Jungeblut, J. Hagemeyer, M. Porrmann, and U. Rückert. "Dynamisch rekonfigurierbare Hardware als Basistechnologie für intelligente technische Systeme". In: *Wissenschaftsforum 2013 Intelligente Technische Systeme*. Proceedings Wissenschaftsforum 2013 Intelligente Technische Systeme. 2013, pp. 79–90. ISBN: 978-3-942647-29-8.

[186] D. Sorrenti, D. Cozzi, S. Korf, L. Cassano, J. Hagemeyer, M. Porrmann, and C. Bernardeschi. "Exploiting dynamic partial reconfiguration for on-line on-demand testing of permanent faults in reconfigurable systems". In: *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 2014, pp. 203–208. DOI: 10.1109/DFT.2014.6962065.