DISSERTAÇÃO DE MESTRADO

RTRLIB: A HIGH-LEVEL MODELING TOOL FOR DYNAMICALLY PARTIALLY RECONFIGURABLE SYSTEMS

Regina Marcela Ivo

Brasília, Dezembro de 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO

RTRLIB: A HIGH-LEVEL MODELING TOOL FOR DYNAMICALLY PARTIALLY RECONFIGURABLE SYSTEMS

Regina Marcela Ivo

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Mestre em Sistemas Mecatrônicos

FICHA CATALOGRÁFICA

IVO, REGINA MARCELA
RTRLIB: A HIGH-LEVEL MODELING TOOL FOR DYNAMICALLY PARTIALLY RECONFIGURABLE SYSTEMS [Distrito Federal] 2019.
xvi, 76 p., 210 x 297 mm (ENM/FT/UnB, Mestre, Engenharia Mecânica, 2019).
Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Mecânica

1. Partial Reconfiguration	2. High-Level Modeling
3. System-on-Chip	4. FPGA
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

IVO, R. M. (2019). *RTRLIB: A HIGH-LEVEL MODELING TOOL FOR DYNAMICALLY PARTIALLY RECONFIGURABLE SYSTEMS*. Dissertação de Mestrado, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 76 p.

Regina Marcela Ivo
Depto. de Engenharia Mecânica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

RESUMO

Reconfiguração dinâmica parcial é considerada uma interessante técnica a ser aplicada para o aumento da flexibilidade de sistemas implementados em FPGA, em função da implementação dinâmica de módulos de *hardware* enquanto o restante do circuito permanece em operação. Trata-se de uma técnica utilizada em sistemas com requisitos muito restritos, como adaptabilidade, robustez, consumo de potência, custo e tolerância à falhas. Entretanto, a complexidade de desenvolvimento de sistemas com reconfiguração dinâmica parcial é consideravelmente alta quando comparada à de sistemas com lógica totalmente estática. Nesse sentido, novas metodologias e ferramentas de desenvolvimento são necessárias para reduzir a complexidade de implementação desse tipo de sistema.

Nesse contexto, esse trabalho apresenta o RTRLib, uma ferramenta de modelagem em alto nível para o desenvolvimento de sistemas com reconfiguração dinâmica parcial em dispositivos Xilinx Zynq a partir da especificação e parametrização de alguns blocos. Sob condições específicas, o RTRLib automaticamante produz os *scripts* de *hardware* e *software* para implementação da solução utilizando o Vivado Design Suite e o SDK. Tais *scripts* são compostos pelos comandos necessários para a implementação do sistema desde a criação do projeto de *hardware* até a criação do arquivo de *boot*. Uma vez que o RTRLib é composto por *IP-Cores* previamente caracterizados, a ferramenta também pode ser utilizada para a análise, em fase de modelagem, do sistema a ser implementado, por meio da estimação de características importantes do sistema, como o consumo de recursos e latência.

O presente trabalho também inclui novas funcionalidades implementadas no RTRLib no contexto do *design* de *hardware* e de *software*, como: generalização do *script* de *hardware*, mapeamento de IO, floorplanning por meio de uma GUI, criação de um gerador de *script* de *software*, gerador de template de aplicação *standalone* que faz uso do *partial reconfiguration controller* (PRC) e implementação de uma biblioteca para aplicações FreeRTOS.

Por fim, quatro estudos de casos foram implementados para demonstrar as funcionalidades da ferramenta: um sistema de classificação de terrenos baseado em redes neurais, um sistema com regressores lineares utilizado para controle de uma prótese miocinética de mão e, por último, uma aplicação hipotética de um sistema com requisitos de tempo real.

Palavras-chave: Reconfiguração Parcial, Modelagem de Alto Nível, Sistemas em Chip, FPGA.

ABSTRACT

Partial dynamic reconfiguration is considered an interesting technique to increase flexibility in FPGA designs due to the dynamic replacement of hardware modules while the remainder of the circuit remains in operation. It is used in systems with hard requirements such as adaptability, robustness, power consumption, cost, and fault-tolerance. However, the complexity to develop dynamically partially reconfigurable systems in considerably higher comparing with static designs. Therefore, new design methodologies and tools have been required to reduce the design complexity of such systems.

In this context, this work presents the RTRLib, a high-level modeling tool for the development of dynamically reconfigurable systems on Xilinx Zynq devices by a simple system specification and parametrization of some blocks. Under specific conditions, RTRLib automatically generates the hardware and software scripts to implement the solution using Vivado and SDK. These scripts are composed by the sequential design steps from hardware project creation to the boot image elaboration. Since RTRLib is composed of pre-characterized IP-Cores, the tool also can be used to analyze the system behavior during the design process by the early estimation of essential characteristics of the system such as resource consumption and latency.

The present work also includes the new functionalities implemented on RTRLib in the context of the hardware and the software design, such as: hardware script generalization, IO mapping, floorplanning by a GUI, software script creation, generator of a standalone template application that uses PRC, and implementation of a FreeRTOS library application.

Finally, four case studies were implemented to demonstrate the tool capability: a system for terrain classification based on neuron networks, a linear regressor system used to control a myokinetic-based prosthetic hand, and a hypothetical real-time application.

Key-words: Partial Reconfiguration, High-Level Modeling, System-on-Chip, FPGA.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ALU	Arithmetic Logic Unit
API	Application Programming Interface
APU	Application processing unit
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BSP	Board Support Package
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
DCP	Design Check-Point
DPR	Dynamic Partially Reconfigurable
FF	Flip-Flop
FP	Floating-Point
FPGA	Field Programmable Gate Arrays
FPU	Float Point Unit
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Description Language
IC	Integrated Circuit
IDF	Isolation Design Flow
IP	Intellectual Property
ISR	Interruption Service Routine
LUT	Look-up Table
MIO	Multiplexed Input/Output
MODE	Multi-Objective Differential Evolution
MPSoC	Multi-processor System-on-Chip
NMOS	N-type Metal-Oxide Semiconductor
NMR	N-Modular Redundancy
OCM	On Chip Memory
OOC	Out of Context
PCAP	Processor Configuration Access Port
PCB	Printed Circuit Board
PIP	Programmable Interconnect Point
PL	Programmable Logic
PLD	Programmable Logic Devices
PR	Partial Reconfiguration
PRC	Partial Reconfiguration Controller
PS	Processing System
PSO	Particle Swarm Optimization
RAM	Random Access Memory
ROM	Read Only Memory
RTOS	Real-Time Operating System
SDK	Software Development Kit
SEU	Single Event Upset
SoCs	Systems-on-Chip
TCL	Tool Command Language
XML	eXtensible Markup Language

1 INTRODUCTION

SRAM-based Field-Programmable Gate Arrays (FPGAs) are semiconductor devices based on a parallel matrix architecture, which are widely used to develop systems with high-performance, including embedded systems, that correspond to a branch in high growth in the engineering for diverse applications. Systematically, embedded systems can be understood as systems designed with specific hardware and software to perform a specific task; that is, they are specialized systems, which differentiate them from general-purpose computational systems (22). However, for many embedded applications, the systems need to be developed to allow not only high performance but also other vital requirements associated with robustness, low power consumption, low cost, physical size restrictions, and also fault-tolerance (23).

In this context, FPGAs are an interesting solution for systems with hard requirements in terms of processing, as they allow the development of embedded systems based in parallel architectures directly in hardware. FPGAs are also known as reconfigurable architectures since they support the definition of new functions that can be mapped in the logic fabric. In this context, it is possible to observe, for this type of system, the flexibility potential, usually present in software projects, combined with the efficiency and speed of a hardware system (24).

To meet the restrict requirements of embedded applications in terms of performance, power consumption, and physical dimensions, one of the techniques employed is the implementation of dynamically partially reconfigurable (DPR) systems. FPGAs are hardware devices that have to be reconfigured when powered up, and this is called static reconfiguration. However, some FPGAs allow reconfiguration at run-time; it is referred to as dynamic reconfiguration. The last-mentioned type of reconfiguration can be divided into two categories: full and partial. On the one hand, in the case of full dynamic reconfiguration, all the hardware features are erased and reconfigured. On the other hand, partial dynamic reconfiguration is a technique that allows the reconfiguration of part of a circuit to be changed while the remainder of the circuit remains in operation (25).

Partial dynamic reconfiguration was initially employed in the context of communication and embedded applications that required an approach that would allow a reduction in the number of components and power consumption (26). It is possible to highlight several applications for dynamic reconfiguration, such as remote sensors, aeronautical applications, biomedical devices, and several others.

In terms of advantages of using DPR systems stand out the increase of the solution flexibility, the decrease of the number of physical resources and consequently the FPGA's size and costs reduction, and the decrease of dynamic power consumption. These assumptions are possible in detriment of the functions loading by demand through the functionality of time multiplexing of the configurations (27).

One exciting possibility is the usage of DPR solutions based on Systems-on-Chips (SoCs) architectures, due to the combination of flexibility and performance of an HW/SW co-design. Besides the precisely co-design benefits, the use of SoCs also allows the partial reconfiguration (PR) through software triggers directly. On the other hand, in terms of disadvantages of DPR systems, one must consider: (a) the reconfiguration time, which can introduce an increase in the total latency of the circuit, reducing its performance;

(b) the area required for the implementation of hardware responsible for controlling the reconfiguration process depending on the reconfiguration strategy adopted by the designer (28).

1.1 PROBLEM DESCRIPTION

The increasing of the project restrictions of embedded systems has evidenced the consequent increase in the availability of solutions based on reconfigurable systems, even those with dynamic partial reconfiguration. However, it turns out that the complexity of developing a DPR is considerably higher when compared to that of a system with totally static logic.

From this perspective, it is possible to see the difficulties in designing DPR systems, such as the hardness to explore the effects of some characteristics of reconfigurable sub-systems, for instance, the reconfiguration time, on the performance and behavior of the system as a whole.

New design methodologies and tools have been required to reduce the complexity of the design of DPR SoCs. However, a bibliographical review of state of the art shows that few studies have been carried out aiming at the construction of high-level tools for automating the design flow of systems with partial dynamic reconfiguration. When comparing the works found with the vast amount of studies aimed at the applications of dynamic reconfiguration in several branches of engineering, there is a gap in terms of design tools and methodologies for systems of this nature.

In this context, research was carried out in cooperation between three institutes, referring to the University of Brasilia (Brasilia, Brazil), KTH Royal Institute of Technology (Stockholm, Sweden), and Saab ab (Linköping, Sweden). The initial idea was centered on the development of solutions based on fault-tolerant methodologies for applications in autonomous avionics systems. However, one of the results of the mentioned study was the creation of a high-level modeling tool for DPR SoCs, so-called RTRLib.

The present work aims to continue the development of the RTRLib, in which it is proposed the implementation of some functionalities, not yet available in the tool and the improvement of other functionalities already available but which presents some limitations. RTRLib is a high-level modeling tool that automatically provides TCL scripts for developing DPR systems on Xilinx Zynq devices, allowing, at design time, the estimation of the resources utilization, latency and reconfiguration time of each reconfigurable partition (RP).

The above-mentioned features are possible because the RTRLib is a platform-based design tool that uses the refinement-by-replacement methodology, allowing non-experts users to make use of functional blocks, interconnecting them as a process network to express a specific behavior of the system. Then, each functional block is refined and replaced by previously characterized IP-cores, enabling an introspection process to be performed, obtaining the structural HDL code of each reconfigurable module (RM).

Since specific details of the IPs are known, the tool can provide an early estimation of the resources consumption and latency of each RM, guiding the designer regarding technical issues such as the size of each RP, the communication interface between RPs, and the reconfiguration process. In the case of hardware redundancy schemes for fault-tolerant systems, the tool also estimates the failure rate and reliability of each RM.

1.2 OBJECTIVES

1.2.1 General Objective

The general objective of this work is the improvement of the functionalities of the RTRLib, aiming to obtain a high-level modeling tool to be used to automate, under specific conditions, the development design flow of dynamically partially reconfigurable systems.

1.2.2 Specific Objectives

The following specific objectives are aimed:

- Improvement of the hardware script generation focusing on the strategy with the Partial Reconfigurable Controller (PRC), automating the steps for RPs IP packing and automatic constraints file generation.
- Development of an approach for floorplanning based on a GUI and its integration into the RTRLib.
- Implementation of the software script, including a generator of standalone template application with PRC and a library to implement applications with FreeRTOS.
- Demonstration of the feasibility of the proposed approach implementing case studies and analysis of potential architectures to be developed with RTRLib.

1.3 METHODOLOGY

The proposed methodology for the development of this work is described below:

- The first phase consisted of a bibliographical review of the state of the art of systems modeling tools with partial dynamic reconfiguration, to place the present research in the gaps of the scientific knowledge in the area and to delimit the possible contributions of the work, taking into account the scope constraints previously defined in the initial RTRLib modeling.
- In the second phase, some experiments based on Xilinx DPR tutorials (29) were realized. The primary goal of this step was to familiarize with the practical development of DPR systems using the PRC IP-Core on Zynq-7000 platform.
- The third phase consisted of familiarizing with the RTRLib, by verifying and analyzing all the functionalities as well as the current limitations of the tool to verify possible improvement proposals. Each limitation point was identified during tests with the platform and through the results of the

analysis of the codes. At this stage, the feasibility and the priority of implementing each functionality were evaluated, and the scope of this work was restricted. The selection of the topics to be developed in this work was held by the generalization of the SoC design flow, enabling an end to end development from the system specification to the software application generation.

- In the fourth phase, the implementation of the functionalities specified in the previous stage was done. Explicitly: hardware script generalization, AXI-Lite interface creation for the RPs and IP packing, automatic synthesis of the logic of the RMs with the HDLs provided by the user, IO mapping by the automatic .xdc creation by the provided constraints, floorplanning by a GUI, software script creation, generator of a standalone template application that uses PRC, and generation of a FreeRTOS library application.
- The final step consisted of the implementation of four case studies using RTRLib to demonstrate its feasibility. It is presented: A straightforward example of 2RPs with 2RMs each that implements arithmetic operations using the floating-points IP-Cores, a terrain classification system based-on multi-layer perceptron neuron network implemented in one RP and 3 RMs each one with different topology, one system with 1RP and 2RMs that implements linear regressors with a different number of operators, and one example of usage of the FreeRTOS blocks and its generated software application.

1.4 CONTRIBUTIONS OF THE WORK

Besides the technological contributions listed in the objectives of the work, it is also possible to mention some academic contributions, such as:

- A paper, available in the appendix, entitled: *RTRLib: A High-Level Modeling Tool for the Implementation of Dynamically Partial Reconfigurable System-on-Chips*, was accepted for publication in the 2019 IEEE International Conference on Reconfigurable Computing and FPGAs that will happen December 9-11,2019.
- Continuation of the collaboration between the University of Brasilia (UnB), Royal Institute of Technology (KTH), and Saab ab.

1.5 DOCUMENT ORGANIZATION

The next parts of this work are divided as follows: Chapter 2 comprises the theoretical basis that starts from the definition of aspects of reconfigurable systems and presents relevant concepts of partial reconfiguration and FreeRTOS. The Chapter then concludes with a literature review to substantiate the later choices and definitions. In Chapter 3, the RTRlib is formally presented, describing it through its proposed methodology, limitations, and functionalities. Then, before concluding, Chapter 4 describes the selected case studies.

2 THEORETICAL FOUNDATION

This Chapter presents a review of elementary concepts related to reconfigurable hardware as well as partial reconfiguration and some characteristics of the Zynq family. Finally, the last Section shows a review of the state of the art of high-level modeling tools for RTR systems will be presented.

2.1 RECONFIGURABLE ARCHITECTURES

The metal oxide semiconductor (MOS)-based integrated circuits has been the technology of choice of the IC industry since its introduction in the 1970s years, first the n-type metal-oxide-semiconductor (nMOS) and then complementary metal-oxide-semiconductor (CMOS).

Gordon Moore predicted the growth rate of the transistors integration degree (30). This prediction is known as Moore's Law, which defends the doubling in transistor density every two years. Figure 2.1 shows a comparison between Moore's prediction and the microprocessor capability from over some decades. It is possible to realize that, however, Moore's law held for some decades, around the 2000 years, start to appear a significant gap, and the current expectations are that it will continue to increase.

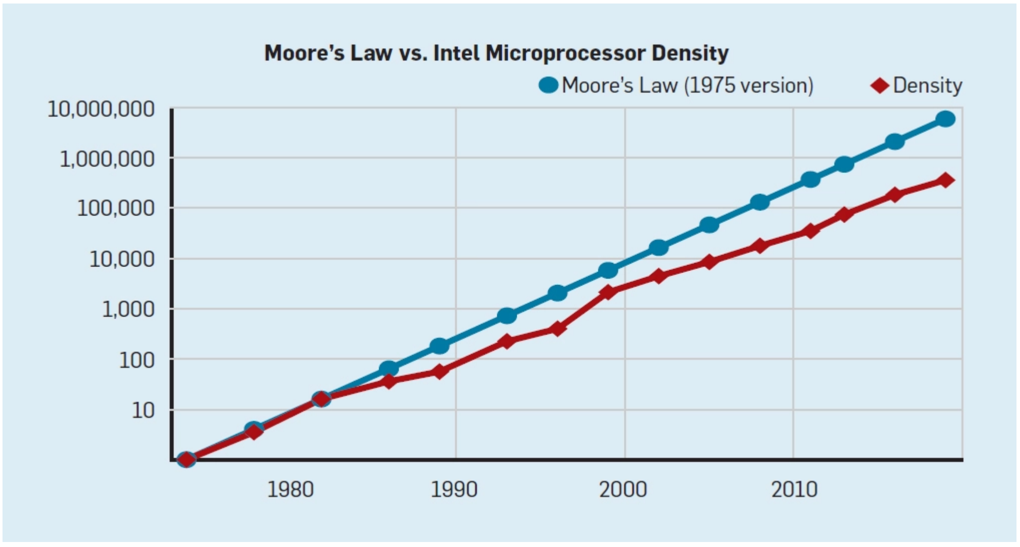


Figure 2.1: FPGA Transistors per chip of Intel microprocessors vs. Moore's Law (1).

Although the computer architecture has undergone several changes in the last decades, from the number of circuits that can be mapped and integrated onto silicon wafers to the sophistication degree with algorithms can be mapped directly on hardware. It is possible to highlight one feature that remained constant, and it is the Von Neumann concept of computer design (31).

Before Von Neumann, each computer machine was designed to perform a single predetermined function. The basic difference behind the Von Neumann architecture is the ability to store program instructions in memory along with the data on which those instructions operate.

Figure 2.2 presents a Von Neumann architecture diagram, where it is possible to see that three distinct sub-systems compose the Von Neumann architecture: a memory, the input/output interface and a central processing unit(CPU), with the last one formed by an arithmetic/logic unit (ALU) and a control unit (32).

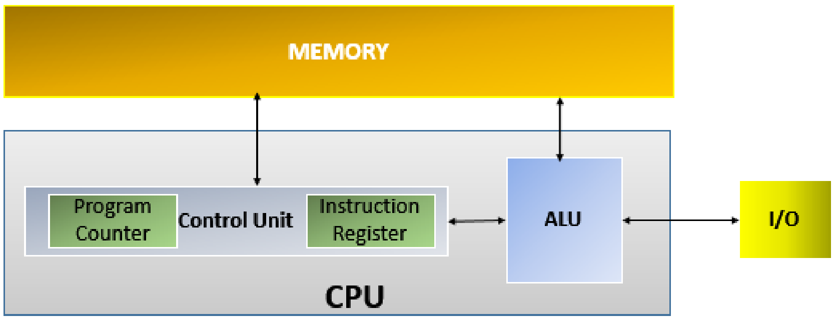


Figure 2.2: Von Neumann architecture diagram.

Although the growth of IC scalability is expressive, as CMOS technology approaches its fundamental limits, the improvement in performance has been affected, and the rate of performance improvement has been reduced, as it is shown in Figure 2.3. In this context, achieving a more significant improvement in the rates of performance, it is essential architectural project that uses the IC capability more efficiently.

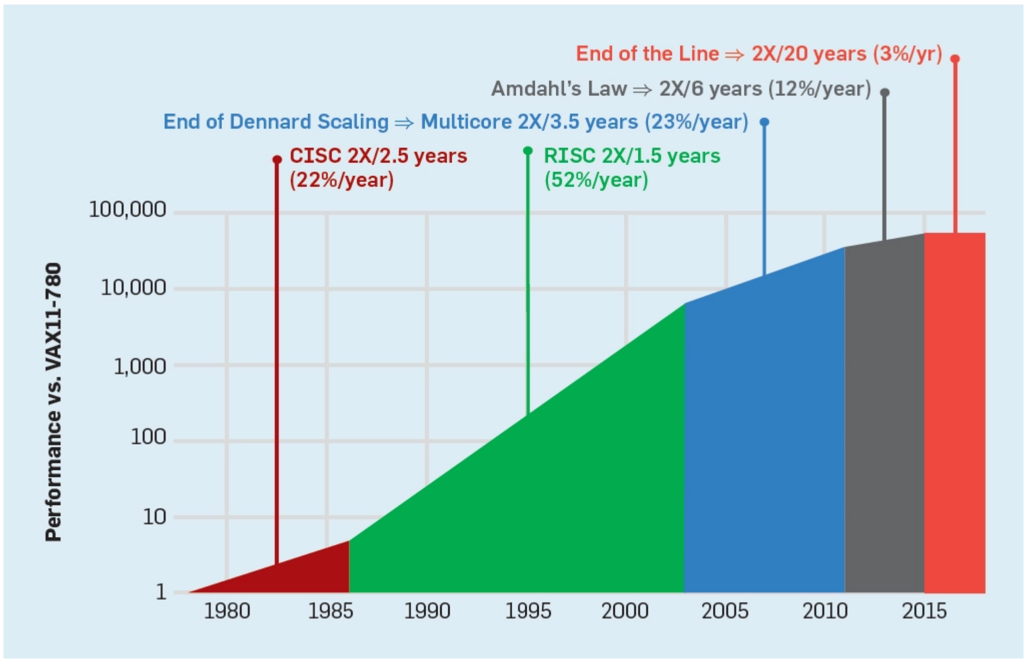


Figure 2.3: Growth of computer performance using integer programs (1).

Even though the systems based-on von Neumann architectures has been achieved high-computational performances, nowadays, several problems need to integrate an HW/SW solution aiming to keep the flexibility and the computational performance. These type of solutions requires customizable architectures which allow the functionality modification even after the manufacturing. Figure 2.4 presents a classification for ICs based on criteria such as programmability and customization capability.

Logic devices can be divided into two categories: customizable or non-customizable. On the one

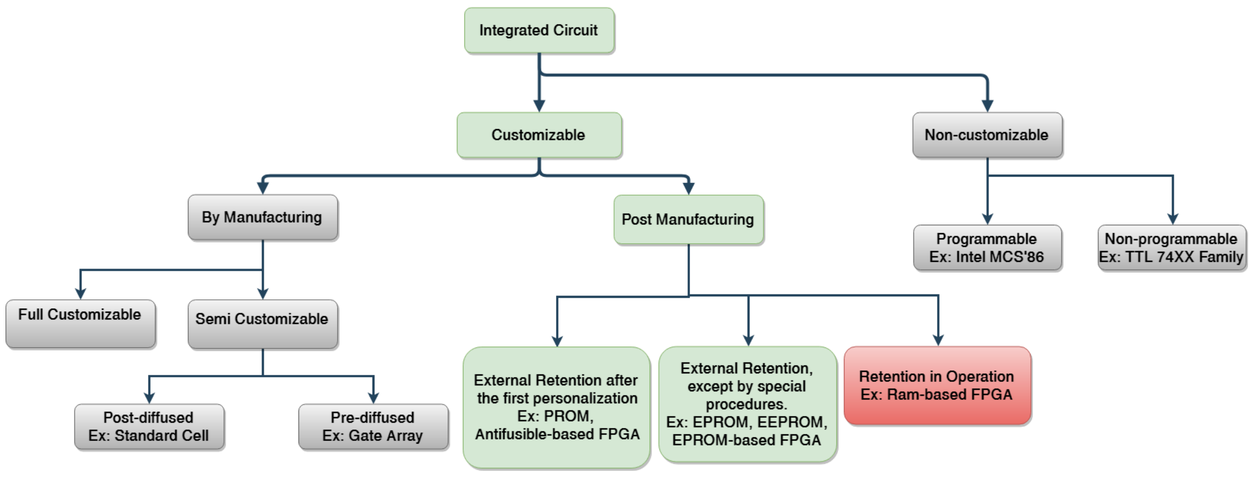


Figure 2.4: Taxonomy for ICs (modified from (2)).

hand, the non-customizable devices, such as the Application-Specific Integrated Circuits (ASICs), do not enable to make changes to its architecture after manufacturing. On the other hand, customizable devices, such as the Programmable Logic Devices (PLDs) and Field Programmable Gate Arrays (FPGAs), can be reconfigured and modified as need even after prototyping.

Historically it is possible to see that the FPGAs are an advancement of PLDs. Generally, the FPGAs distinguishes from the PLDs by two essential features:

- The PLDs were designed to implement combinational logic circuits with an array of gates and wires, while in FPGAs, the logic is implemented with function generators and discrete memories.
- Most FPGAs use static RAM cells to hold the configuration information, rather of PLDs that are non-volatiles (3).

2.1.1 FPGAs

Field Programmable Gate Arrays (FPGAs), are Integrated Circuits (ICs) that can be reprogrammed after fabrication, constructed by the replication of the same configurable logic block (CLB) organized in a matrix. Each CLB contains a small number of inputs and outputs. Inside each CLB can be found small cells, commonly based on Look-up Tables (LUTs), multiplexers, and Flip-Flops (FFs).

The basic structure of an FPGA is based on the composition of the following resources: I/O blocks, used to make the interface with the I/O pins of the device, logic blocks, to implement the logic functions and the network interconnection based on wires and switches. Figure 2.5 shows a simplified version of the FPGA architecture. It is described some essential concepts related to FPGAs as follows.

- *Look-Up Table (LUT):* The fundamental building block of an FPGA, that can implement any logic function of N variables. Basically, a LUT is a truth table that can be used to implement logic functions, small Read Only Memory (ROM), small Random Access Memory (RAM) or shift register. It is possible to combine LUTs to perform more complex logic functions, larger memories or larger

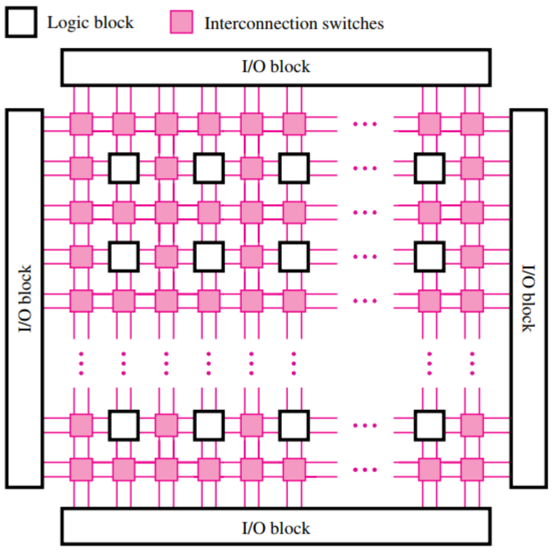


Figure 2.5: Basic FPGA architecture(3).

shift registers is also possible. The hardware implementation of a LUT can be described as a set of memory cells connected to a collection of multiplexers. The array of inputs acts as the selector of multiplexers that selects the result of the correspondent inputs.

- *Switch Matrix:* It provides a facility routing between elements within a CLB and from one CLB to other elements on the programmable logic.
- *Flip-Flop:* The primary storage unit within the FPGA fabric, this register is paired with a LUT to store its result, and also to assist in logic pipelining.
- *Input/Output(I/O) pads:* Registers used to store the result of the LUT.
- *Configurable Logic Blocks (CLBs):* Small building blocks of an FPGA, composed of regular groups of logic elements that are laid out in matrix array on the programmable logic. The composition of a CLB is presented in Figure 2.6. The most common type of Configurable Logic Blocks (CLBs) is built using Look-Up Tables (LUTs), a small set of inputs, flip-flops, and multiplexers, allowing logic functions to be implemented.
- *Slice:* Sub-unit of the CLB, capable of implement combinatorial and sequential logic circuits. In the case of Zynq family (used in this work and detailed later), one slice is composed mainly of 4 Look-Up Tables and 8 Flip-Flop.

Although these elements are enough to implement any algorithm, contemporary FPGA architectures, showed in Figure 2.7, also integrate other additional computational and data storage elements, listed below:

- Embedded memories for distributed data storage.
- Phase-locked loops (PLLs) for driving the FPGA fabric at different clock rates.
- High-speed serial transceivers.

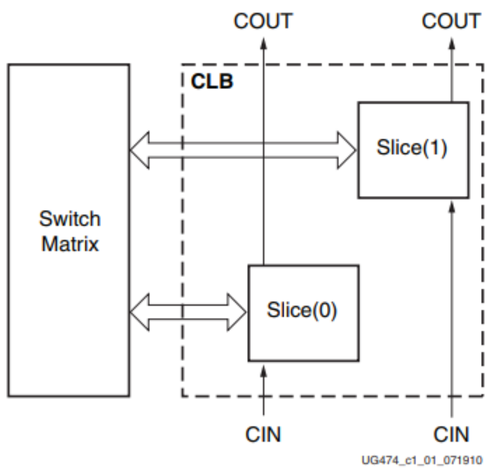


Figure 2.6: Composition of a generic Configurable Logic Block (CLB) (4).

- Off-chip memory controllers.
- Multiply-accumulate blocks.

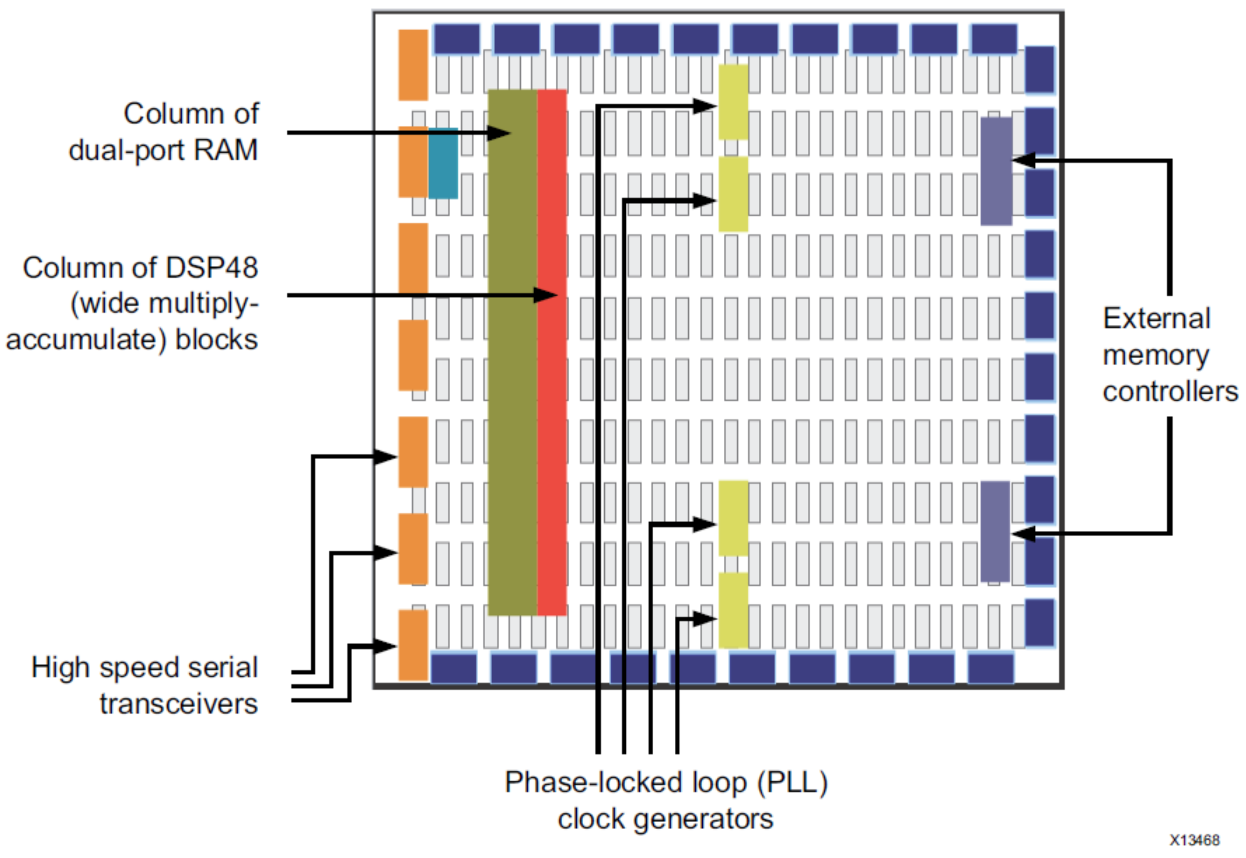

Column of dual-port RAM
Column of DSP48 (wide multiply-accumulate) blocks
High speed serial transceivers
External memory controllers
Phase-locked loop (PLL) clock generators
X13468

Figure 2.7: Contemporary FPGA architecture(5).

The incorporation of these blocks increases the computational density and efficiency of the device as it will be explained below.

DSP Block: The DSP Block is an arithmetic logic unit implemented into the FPGA fabric, which is shown in Figure 2.8. It is composed of three different chain blocks, which allows performing function such as:

$$p = a \times (b + d) + c \tag{2.1}$$

or

$$p+ = a \times (b + d) \tag{2.2}$$

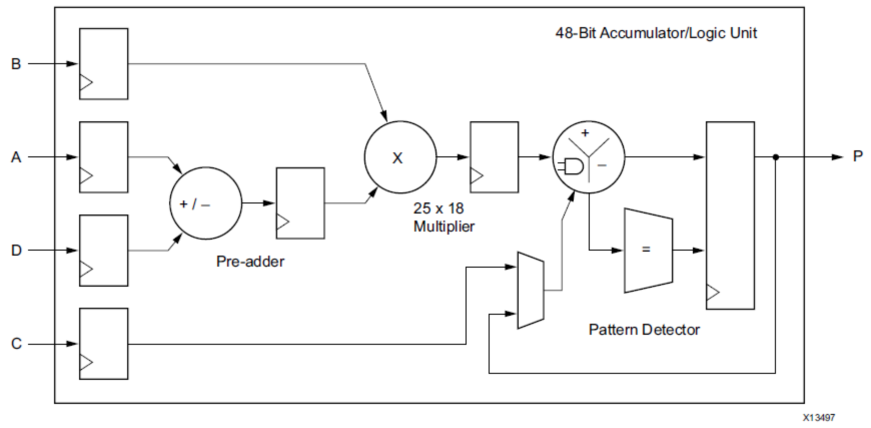


Figure 2.8: Structure of a DSP block (5).

Storage Elements

BRAMs are dual-port RAM modules embedded into the FPGA fabric. These type of memory allows for parallel, same-clock-cycle access to different locations, due to the dual-port nature. In the device, there are two types of BRAM, that can hold 18 or 32 kbits.

BRAMs can implement either a ROM or a RAM. In the case of the ROM configuration, the data is written as part of the FPGA configuration and, therefore, can not be modified, which implies that the data can only be read during the runtime of the circuit. In the case of the RAM configuration, data can be read and written at any time during the runtime of the circuit.

2.1.2 Design steps with FPGA

Figure 2.9 presents the design flow with FPGAs. It is important to note that this design flow is very similar to ASICs design flow. Below each development step is briefly described.

- *System planning and HDL design:* This step comprises the functional description of the system considering the area and performance restrictions. The system functionality is expressed using hardware description language (VHDL or Verilog).
- *Synthesis:* This step produces a circuit netlist, which contains a list of logic elements and its connections, is created from the HDL or the schematic.

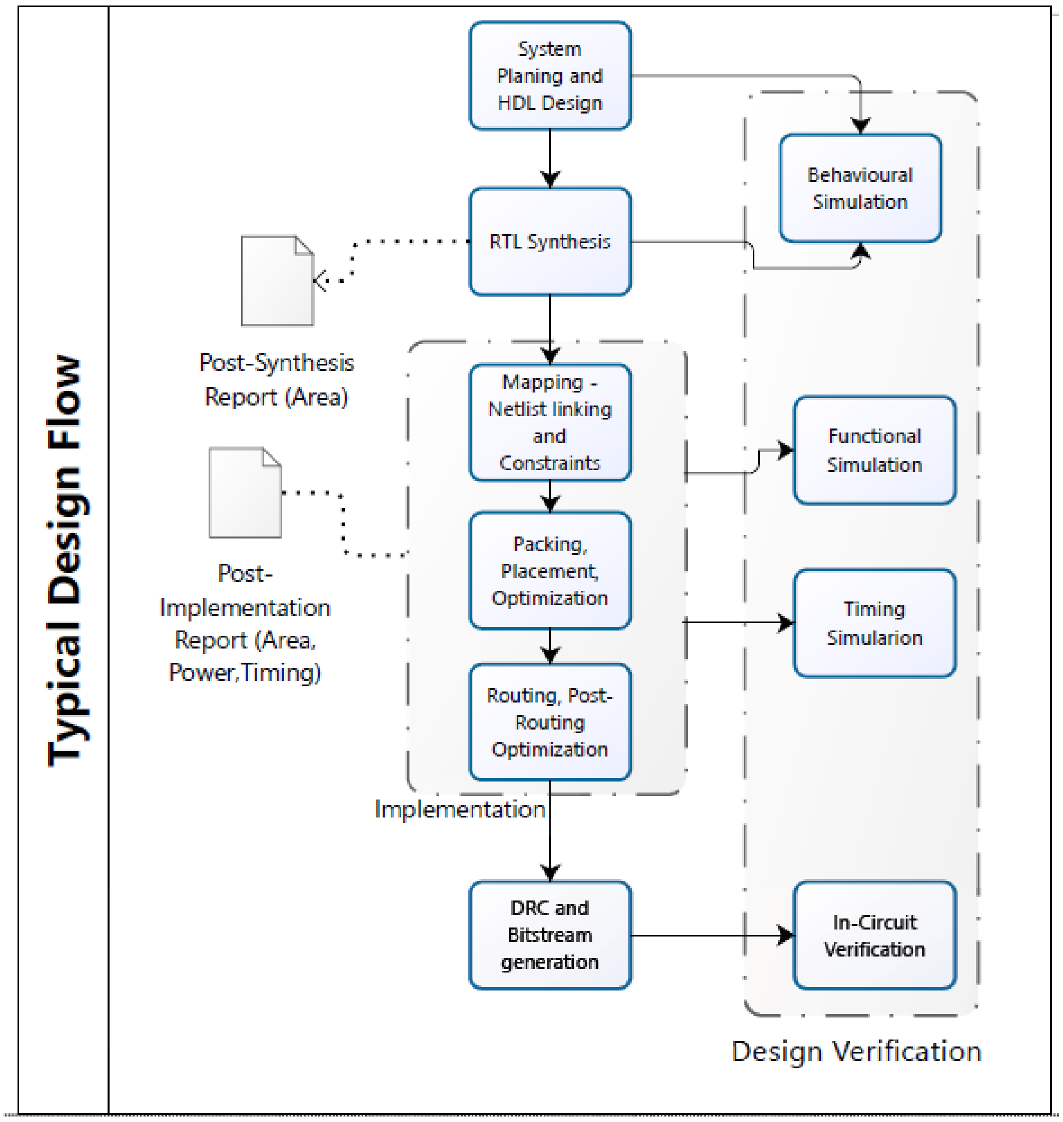

Typical Design Flow
System Planing and HDL Design
RTL Synthesis
Behavioural Simulation
Post-Synthesis Report (Area)
Mapping - Netlist linking and Constraints
Functional Simulation
Post-Implementation Report (Area, Power,Timing)
Packing, Placement, Optimization
Timing Simularion
Routing, Post-Routing Optimization
Implementation
DRC and Bitstream generation
In-Circuit Verification
Design Verification

Figure 2.9: Design Flow with FPGA (adapted from (6)).

- *Mapping:* In this process the logic functions are mapped into the configured elements of the device.
- *Placement:* The logic functions are placed in specific positions on FPGA.
- *Routing:* The connections between the logic blocks are made.
- *Bitstream generation:* The bit-stream file is generated with the device components data, and then the program is loaded on the board.

In terms of design verification, it is possible to describe each step as follows.

- *Behavioral simulation:* From the netlist, the circuit behavioral is analyzed to verify if it matches with the expected results. The verification is done using a testbench file which received the test vectors.
- *Timing simulation:*This simulation is essential to do the timing analysis and to verify if the timing restrictions of the circuit are being matched.

- *Functional simulation:* It includes the behavioral and the timing simulation, and it tests the operation of the circuit considering the physic information of the circuit components to guarantee the circuit operation.
- *In-Circuit Verification:* After the device programming, the final verification is realized on the board.

Modern FPGAs consists of up to two million logic cells that can be used to implement a variety of applications with the advantages of the performance of a hardware solution and the flexibility of a programmable solution. Figure 2.10 presents different objectives for adopting FPGA prototyping.

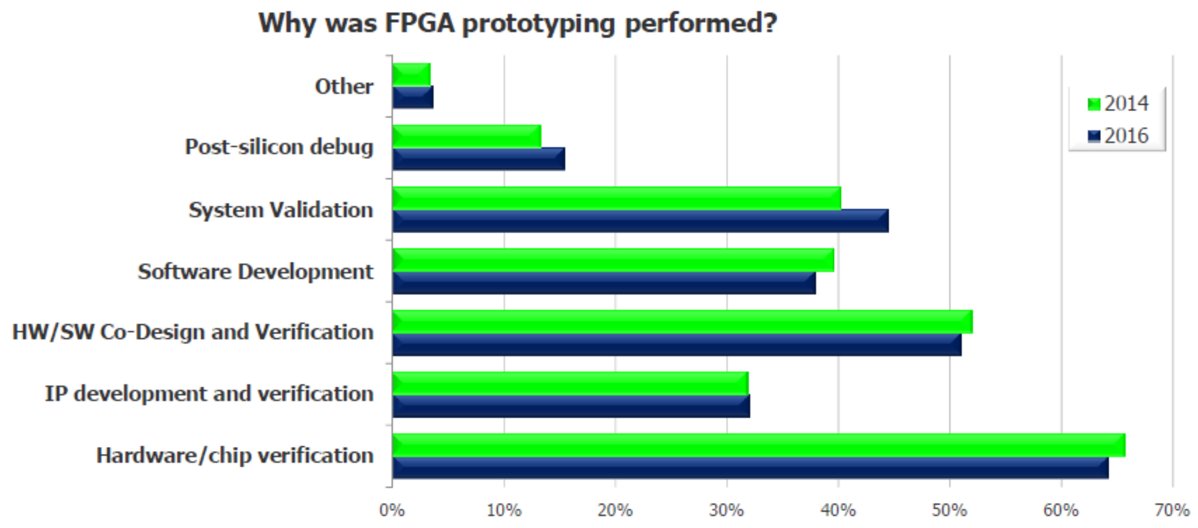


Figure 2.10: Results of a study in trends in functional verification performed by Mentor Graphics in 2016, describing design projects doing FPGA prototyping (7).

2.2 HARDWARE/SOFTWARE CO-DESIGN

Several embedded applications deal with different types of constraints such as performance, flexibility, size, weight, power consumption, among others. When high-performance and flexibility are required, it is important to use a hybrid hardware-software solution (33).

The combination of software elements with hardware elements in the project is called HW/SW co-design that consists of partitioning the system's functionalities in such a way that software is used for flexibility. In contrast, the hardware is used for performance (34). The idea is to speed up system execution time by performing some operations in parallel. In other words, the partitioning performance depends on the distribution of the more critical tasks as well as the parallelism level between the tasks (35).

An HW/SW co-design is based on a platform formed by one GPP based on Von Neumann architecture, some custom components, bus communication, and one main memory, as is shown in Figure 2.11 (36).

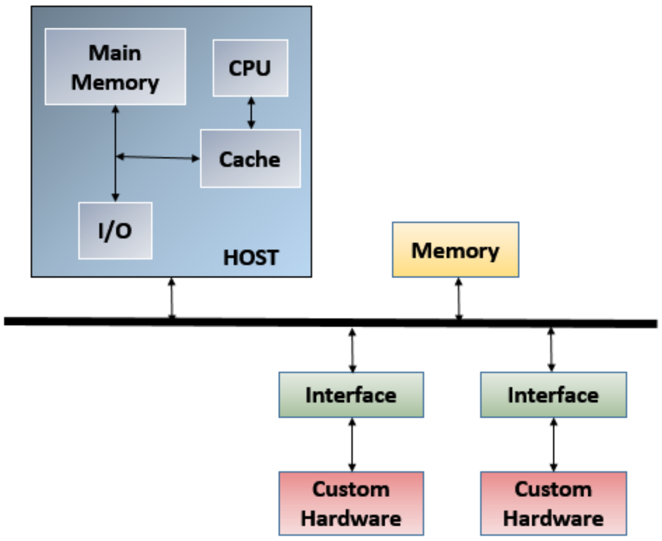


Figure 2.11: HW/SW CO-Design Diagram (adapted from (8)).

2.3 CHARACTERISTICS OF THE ZYNQ-7000 FAMILY ARCHITECTURE

The Zynq-7000 AP SoC architecture consists of a combination of a dual-core ARM Cortex-A9 processor with an FPGA logic fabric, based on Xilinx 7-series FPGA architecture, forming, respectively, the Processing System (PS) and the Programmable Logic (PL). To realize the communication between the two parts of the device, the architecture contains industry-standard AXI interfaces, which provide low latency and high bandwidth (10).

The PL part is recommended to implementations of high-speed logic, arithmetic, and data flow subsystems. On the other hand, the PS supports software routines that can be running in an operating system or not. Therefore, the designer must separate the overall functionality of the design system between hardware and software properly to obtain the maximum performance of the system. Some features of the PS and the PL are presented as follows.

2.3.1 Processing System - PS

It is important to emphasize that the Zynq PS involves not only the ARM processor, but a set of associated processing resources forming an Application Processing Unit (APU), peripheral interfaces, cache memory, and clock generation circuitry and other modules. This structure can be viewed in Figure 2.12.

The ARM Cortex-A9 can operate at up to 1GHz, depending on the particular Zynq device. The APU encompasses two ARM processing cores, with its Media Processing Engine (MPE) and Float Point Unit (FPU), Memory Management Unit (MMU), and a level 1 cache memory of 32KB for instructions and data. There is also a level 2 cache memory of 512 KB to be shared by the two cores, an On-Chip Memory (OCM) of 256KB and a Snoop Control Unit (SCU) that is responsible for forming a bridge between the ARM cores and the level 2 cache memory and for making the interface with the PL.

Multiplexed Input/Output (MIO) achieves the communication between the PS and external interfaces, through 54 pins connectivity. Another way to make a connection is through Extended MIO (EMIO), which can be used when extension beyond 54 pins is necessary or as a procedure of interfacing the PS with an IP block implemented in the PL.

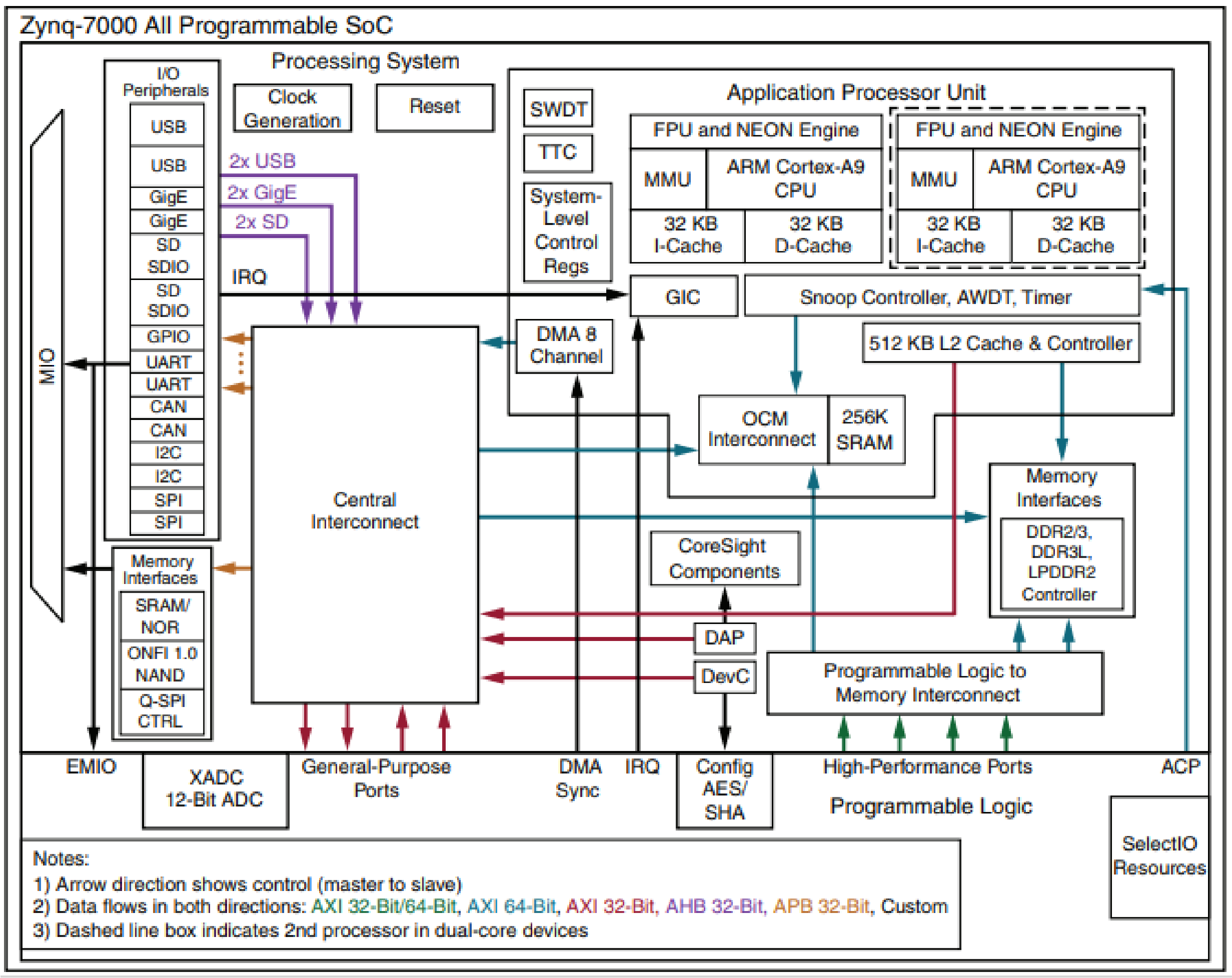


Figure 2.12: Functional blocks of the Zynq-7000 architecture (9).

2.3.2 Programmable Logic - PL

The PL is predominantly composed of general-purpose FPGA logic fabric, which is composed of Configurable Logic Blocks (CLBs), DSP Blocks, BRAMs, and Input/Output for interfacing. Specifically, this part of the Zynq architecture is based on Artix®-7 and Kintex®-7 FPGA fabric.

There are two special-purpose components that deserve special attention, the DSP48E1 slices for high-speed arithmetic and the Block RAMs for dense memory requirements. Because of the high proximity between storage data operations and intensive computation operations, these components are integrated close into logic array.

The block RAMs in the Zynq-7000 are equivalent to those on Xilinx 7 series FPGAs, each of them can store up to 36Kb of information, being possible to combine two or more Block RAMs to form larger capacity memories.

As presented before, the DSP48E1s are specialist slices designed for implementing high-speed arithmetic on signals with medium to long arithmetic wordlengths. They are primarily composed of a pre-adder/subtractor, multiplier, and post-adder/subtractor and as well as a logic unit.

The I/O banks are categorized as High-Performance (HP) or High-Range (HR). The HP interfaces are used for high-speed interfaces to memory, and other chips with a limitation of a voltage of 1.8V. The HR interfaces cater to a wide variety of I/O standards and permit voltages up to 3.3V.

Other I/O external interfaces to the PL that can be mentioned, as follows:

- A set of JTAG ports are available to facilitate the configuration and the debugging of the PL.
- Beyond the facility to generate and distribute its clock signals, the PL receives four separate clock inputs from the PS.
- The XADC is another important IP, which is a set of two separate 12-bit Analogue to Digital Converter (ADC) mixed-signal hardware that has the capability of sampling external analog signals at 1MSPS. The PS-XADC interface located on PS controls this IP.

2.4 PS-PL INTERFACES

As mentioned before, the challenge lies in the ability of the designer to decide which part of the system executes on PS and which executes on PL appropriately to optimize the platform's performance. Therefore it is essential to highlight the mechanisms of connections between the PS and the PL.

2.4.1 Advanced eXtensible Interface - AXI

AXI stands for Advanced eXtensible Interface, and the current version is AXI4, which is part of the ARM AMBA 3.0 open standard. Many devices and IP blocks produced by third-party manufacturers and developers are based on this standard. The AMBA®standard was initially developed by ARM for use in micro-controllers, with the first version being released in 1996. Since then, the standard has been revised and extended, and it is now described by ARM as " the facto standard for on-chip communication " (37). Xilinx contributed actively to defining AXI4 as an optimal interconnect technology for use within the FPGA fabric. Table 2.1 presents some features of each type of AXI4 interface.

2.4.2 Extended Multiplexed Input/Output - EMIO

As was mentioned before, EMIO is a method that enables to make several connections from PS to external interfaces through the PL. Figure 2.13 shows two use models of EMIO. One option is to use the EMIO to interface the PS with a peripheral block in the PL. Another option is to connect the PS directly to the desired external pins of the PL, providing additional 64 inputs and 64 outputs with corresponding output enable. Therefore, the EMIO deals with signal transfer between two domains, and the connections are arranged in two 32-bit banks (10).

Table 2.1: Types of AXI interfaces (20).

Interface	**Features**	**Burst**	**Data Com.**	**Applications**
AXI4	For memory-mapped links, and providing the highest performance.	up to 256	32 to 2014	Embedded, Memory
AXI4-Lite	A simplified link supporting only one data transfer per connection (no bursts). Memory mapped.	1 - not	32 or 64 bits	Small Logic Control, FSM
AXI4-Stream	For high speed streaming data. No address mechanism. Only Data, Burst.	unlimited	Any	Communication, Video, DSP

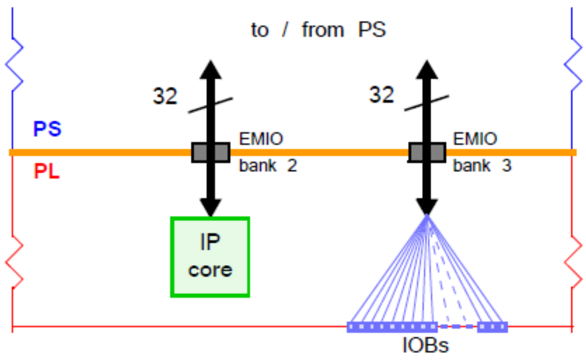


Figure 2.13: EMIO as an interface between PS and PL (10).

2.5 PARTIAL RECONFIGURATION

Partial Reconfiguration (PR) can be defined as the ability of the system to dynamically modify blocks of the logic by downloading partial bitstreams at the same time the remaining logic stay operating without interruption. In other words, it eliminates the need to fully reconfigure and re-establish links aloowing designers to change functionalities on the fly, which dramatically enhances the flexibility thtat the FPGAs offers. In this context, the utilization of the PR technique optimizes the usage of silicon by loading the functionality that is needed at any time, that permits designs to be smaller and/or with fewer components, possible improving the system upgradability (28).

Concerning the main benefits of partial reconfiguration, it is possible to highlight the potential reduction in terms of area and, consequently, power consumption, strategy to be used for fault recovering, performance and a faster system start (17).

In the function of the time multiplexing of the logic, the same portion of the FPGA can be used to implement different logics decreasing the total area of the circuit. Although, due to the area reduction, there are more potential benefits than power and cost savings carried out by PR described previously. The possibility to implement the system into smaller FPGAs reduces the size of the embedded system's package and, in the case of complex systems implemented as a network in multiple FPGAs, PR might also be used for achieving a higher level of integration even reducing the number of FPGAs used in the solution. In this context, reducing the number of FPGAs, the PCB design complexity is decreased, and the off-chip

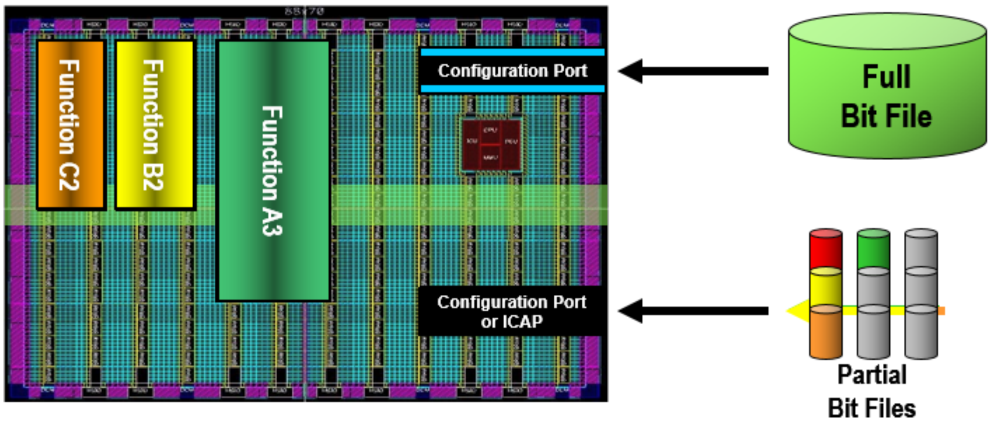


Figure 2.14: Partial Reconfiguration Concept Description (11).

communication reduces, which also saves power. Furthermore, lower consumption leads also to lower cooling effort.

Beyond area and power savings, PR also improves the system by speeding up the start-up phase. For some applications, the initial configuration time by fully programming the FPGA can be too long compared to the application timing restrictions. In this case, PR can be used to allow only the time-critical parts of the system to be configured at the start and the remaining logic implemented at a second configuration step.

Another feature that PR can help is in system performance. Considering more areas available, it is possible to exploit higher levels of parallelism. Then, non-critical modules can share the same area been placed in reconfigurable areas while the designer can provide more resources to the accelerators.

Some important terminologies regarding PR are described below (11).

- *Reconfigurable Partition (RP):*Portion on the design hierarchy intended for the reconfiguration.
- *Reconfigurable Module (RM):* Part of the logical design that occupies the Reconfigurable Partition space.
- *Static Logic:* All logic in the design that is not marked as reconfigurable.
- *Partition Pins:* Refers to the interface between the reconfigurable and the static logic.
- *Configuration:* Corresponds to a complete FPGA design that is composed of the static logic and one variant for each reconfigurable instance, that is, one RM for each RP. In other words, it consists of *static logic* and one variant for each reconfigurable instance.
- *Trusted route:* A route that allows on-chip communication between two isolated functions.
- *Feed-through Routes:* Connections between static logic that crosses an RP.
- *Footprint:* Arrangement of resources that are used inside an RP.
- *Frame:* One column in a clock region, it is also the smallest reconfigurable unit of a Zynq FPGA.

It is also necessary to pay attention to some requirements when implementing a PR design that was established to guarantee that the bitstreams can be precisely created and safely downloaded into an active device. These requirements can be synthesized into the following premises:

1. The logical and physical interface of an RP remains consistent for all the RM implementation.
2. The logic and routing of an RM are fully contained within a physical region, which is then translated into a partial bitstream.
3. If the dedicated initialization feature is used, the static logic must be kept out of the reconfigurable region.

2.5.1 PR Design Flow

From the PR concepts presented previously, this Section presents the PR design flow commonly used by software development tools. An overview of this process is shown in Figure 2.15.

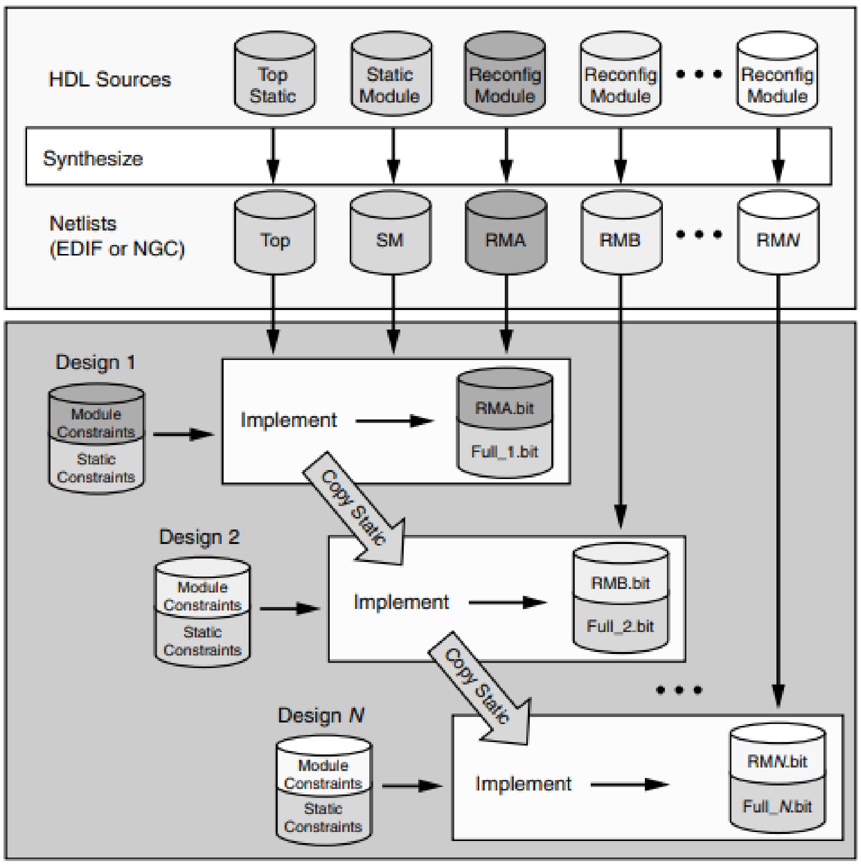


Figure 2.15: Overview of the Partial Reconfiguration Software Flow (12).

1. *Structure design:* The static logic, RPs, and RMs must be defined.
2. *Bottom-up synthesis:* It refers to the synthesis of the design by modules. In Vivado, this is named out-of-context(OOC) synthesis. In this case, the OOC synthesis generates a separate design checkpoint

per OOC module and ensures that no optimizations occur across the RP boundaries. The static logic of the top level is synthesized with black-box property for each RP.

3. *Define resources to be reconfigured:* In this step, the regions of the Pblocks that will host the RPs must be defined into the device, considering the utilization requirements of the related RP.
4. *Implementation:*This refers to the floorplanning of the RPs and the place and route of all design configurations. Therefore, full design constraints in-context must be applied. The regions of the Pblocks that will host the RPs must be defined into the device, considering the utilization requirements of the related RP, the pblock must be set as a reconfigurable module and all the configurations implemented.
5. *Final Verification:* It uses the *PR verify tool*. This step has the intention to validate the consistency of place and routed results of all the system to ensure that the static implementation, including interfaces to reconfigurable regions, is consistent across all configurations.
6. *Generate Bitstreams:* This is the final step that consists of generating all full and partial bit files automatically.

In a few words, to implement a PR design is similar to implementing multiple non-PR designs that share some common logic. In this context, this common logic, between all the designs, must be identical.

2.5.2 Routing Isolation

Controlling fault propagation in design is required for developing a fault-tolerant design. In these cases, it is important to design a system with independent functions to ensure that a failure in one function does not propagate to another one. One strategy to allow the usage of a single-chip instead of multichip is the function isolation. One widely used technique to create an isolated design is the Xilinx Isolation Design Flow (IDF), which was developed to allow independent functions to operate on a single-chip focusing on providing safety-critical functions to applications such as avionics, automotive and industrial.

To separate isolated functions within a single chip, it is used as a fence. That is, a set of unused tiles within no routing or logic is present. Furthermore, each function to be isolated must be at its own level of the hierarchy. Enabling the HD.ISOLATED attribute, all available routes in a design are automatically analyzed according to some rules, and the routes which meet these restrictions are selected as *trusted routes*, those restrictions are described below. Trusted routes:

- Do not have programmable interconnect points (PIPs) in the fence between isolated regions.
- Have one source and one destination isolated regions.
- Are entire stays contained in the fence between the source and the destination regions.
- Do not come within one fence tile from another unintended isolation region.

Figure 2.16 shows an example of routes that would be filtered by them, while Figure 2.17 shows an example of a safety-critical design and its implemented view.

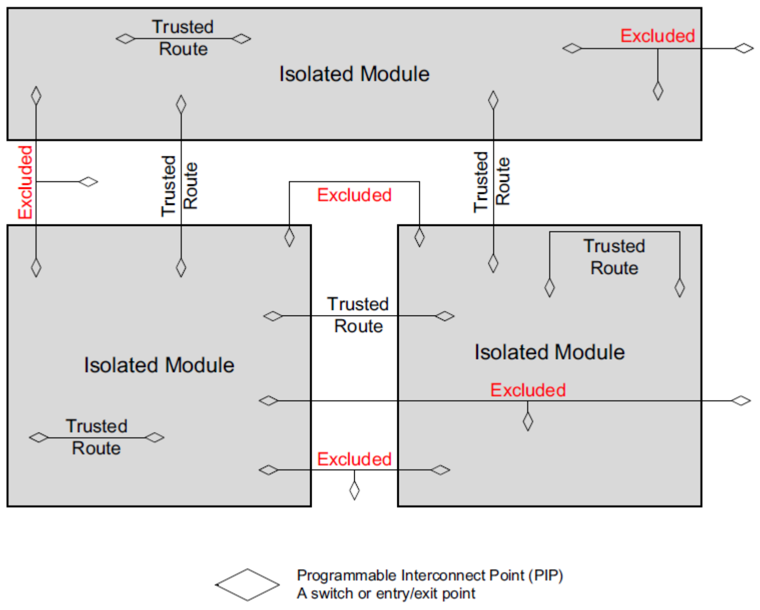


Figure 2.16: Available routes after applying trusted rounting rules (13).

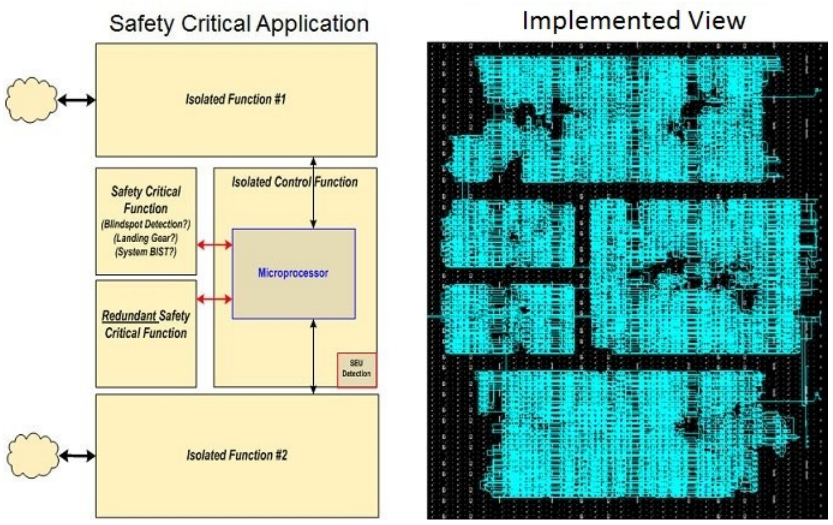

Safety Critical Application
Implemented View
Isolated Function #1
Safety Critical Function (Blindspot Detection?) (Landing Gear?) (System BIST?)
Isolated Control Function
Microprocessor
Redundant Safety Critical Function
SEU Detection
Isolated Function #2

Figure 2.17: Example of a safety critical application implemented in FPGA with the Xilinx IDF (14).

2.5.3 Static/Reconfigurable Decoupling

During the PR process, the static logic must ignore the outputs from an RM since they do not output valid data until the PR is complete, and the reconfigured logic is reset. To solve this issue, common design practice is to decouple the RP. A common practice is based on the usage of one or more PR decoupler cores. When active, some selecting signals that cross the RP boundary from the RP to the static logic are driven to user-configurable value. Otherwise, when inactive, these signals are passed unaltered (15).

A simple decoupler that decouples a single signal is based on a MUX 2:1, that pass the unaltered signal during normal operation and pass a user-defined signal during the partial reconfiguration. Nevertheless, it is possible to decouple signals of RPs with interfaces more complex such as AXI4-Lite and AXI4-Stream. Xilinx recommends which signals must be decoupled, but they are user-defined, as follows (38).

- All the control signal generated from the RPs.
- In the case of the RM being loaded cannot be fully reset before an operation, all control signals driven into the RP must be decoupled.
- The input clock must be decoupled in case that an RM has logic that can start to execute without being qualified by a decoupled control signal.

2.5.4 Reconfigurable Partition Interfaces

Along different Xilinx PR design flows over the years, different reconfigurable partition interfaces has been proposed, from *bus-macro*, and then to *proxy logic*, and lastly to *partition pins*.

In the case of bus-macro, a pair of LUTs are created and connected through fixed wires. One LUT is positioned in the static region and the other one in the reconfigurable region (39). The proxy logic technique, otherwise, inserts a single LUT1 element for each partition pin that has to be a fixed interface (12).

Otherwise, the partition pins are the physical and logical connection between the RP and the static logic, automatically created for all reconfigurable partition ports. From the perspective of the partition pins placement in the RP boundary, some details must receive attention to achieve the consistency requirement of design, as described in Table 2.2.

Table 2.2: Description of types of partition interface usage in PR designs.

Partition Interface Usage	**Description**
Both Static and RM sides have active logic	This is the optimal case.
Static logic has an active driver, but the RM does not have an active load	It comprehends the situation that not every RM has the same I/O requirements, the unused inputs ports are left unconnected.
Static logic has active loads, but the RM does not have an active driver	The outputs ports unused by the RM are driven by ground (logic 0)
Static logic does not have an active driver, but the RM has an active load	This results in an ERROR.
Static logic does not have active loads, but the RM has an active driver	This does not result in an error, but it is recommended to remove these partition outputs.
Neither static nor the RM has driver or loads for a partition port.	It is recommended to remove this RM since it is unnecessary to instantiate this port list.

2.5.5 Xilinx PR IPs

To assist users in specific aspects of a reconfigurable design and turn the implementation easier, Xilinx has created four intellectual properties (IP) cores available in the Vivado IP Catalog. Table 2.3 presents a simplified description of each of these IPs.

Table 2.3: Xilinx PR IPs for users assistance.

Xilinx IP	**Description**
PR Controller	The PRC IP allows the management of the configuration for self-controlling partially reconfigurable designs. It is also equipped with an AXI4-Lite interface that allows the core to be reconfigured at run time.
PR Decoupler	The PRD IP is used to decoupling, ensuring a safe and managed boundary between the RP and the static logic during the partial reconfiguration.
PR AXI Manager	PR AXI Shutdown Managers are used to guaranteeing the safety of the AXI interfaces between the RP and the static logic during the reconfiguration process to avoid a system deadlock in the function of failures to complete AXI transactions.
PR Bitstream Monitor	The PR Bitstream Monitor is used for debugging purposes since it identifies the partial bitstreams at key places in the datapath, extracts, and reports this information.

2.5.6 Configuration Modes

Partial reconfiguration is supported using different configuration modes according to the device target. For the case of the Zynq family, three configuration modes are available: ICAP, PCAP, and JTAG (15).

- *ICAP:* The internal configuration access port enables PL to be reconfigured from within PL itself. Since it is a primitive inside the PL, it requires some resource utilization, and this point must be considered by the designer.
- *PCAP:* So-called processor configuration access port, it is an interface that enables full and partial reconfiguration of the PL from the PS. The primary advantage of this configuration mode is that it does not require any PL resources. Nevertheless, its main drawback is that it blocks the processor during the reconfiguration process (40).
- *JTAG:* This mechanism is generally used for quick testing and debug.

One relevant detail to be pointed out is that the ICAP and PCAP cannot be used simultaneously and are mutually exclusive. However, it is possible to switch from one configuration mode to another just in case that no commands or data are being transmitted or received during the switching time.

A partial bitstream can be downloaded in the same manner as a full bitstream. In this context, in the case of configuring through a microprocessor, an application running in the external microprocessor determines which partial bitstream should be downloaded into the device, the external memory then redirects the corresponding bit file to the FPGA configuration port. This process is depicted in Figure 2.18.

2.5.7 Bitstream Type Definitions

In the case of a partial reconfiguration design, four different types of bitstreams are created, *Full Configuration Bitstreams*, *Partial Bitstreams*, *Blanking Bitstreams*, and *Clearing Bitstreams*.

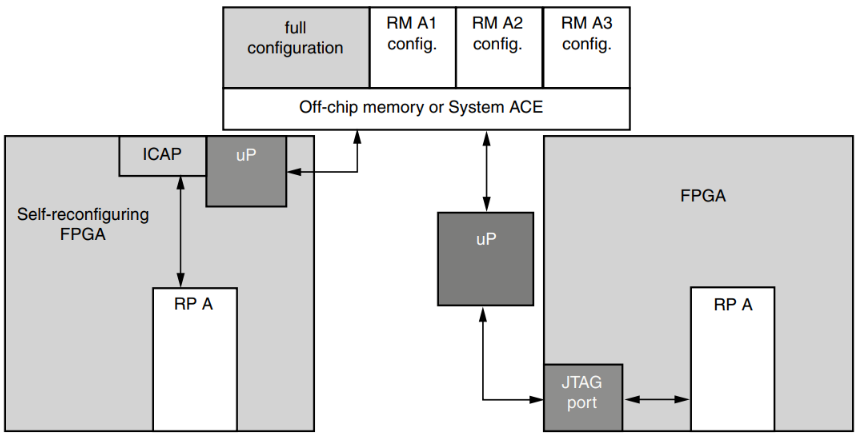


Figure 2.18: Configuring through a microprocessor (15).

Full configuration bitstreams contain all the need information to reset the FPGA, configure it with a complete design, and also verify if the BIT file is not corrupted. Every PR design starts with a standard configuration of the full device using this type of bitstream. After the download of a full BIT file is completed and verified, the FPGA enters in the user mode. However, in the case of the BIT file is corrupted, the FPGA does not enter in the user mode.

In contrast, partial bitstreams have limited specific address sets to program a specific region of the device, and the size of a partial bitstream is directly proportional to the size of the RP it is reconfiguring.

When the partial BIT file is loaded, the FPGA is already in the user mode, then to verify the BIT file integrity, it is necessary to apply a technique of per-frame CRC checks, that will be described later in this work. Just in case that the property of *reset after reconfiguration* is set, the DONE pin pulls LOW until the partial reconfiguration successfully completes. Figure 2.19 presents a comparison between configuring using full and partial bitstreams.

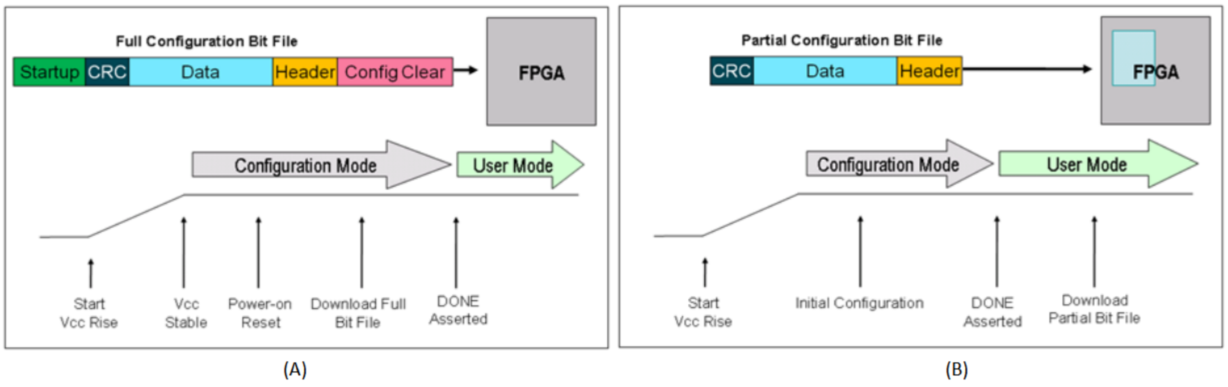


Figure 2.19: (A)Configuring with a Full Bistream (B) Configuring with a Partial Bistream (15).

A special category of the partial bitstream, so-called *Blanking Bitstreams*, is the one that represents a logical black box. Since it is not completely empty, because of the tie-off LUTs and any static routing that happens to pass through this region of the FPGA, Vivado refers to it as a grey box. The grey box variation is saved as another configuration checkpoint.

Clearing Bitstreams are created just in designs for UltraScale devices and are used to clear an existing module before loading a new module. Since the present project does not intend to deal with this type of device, more details will be let out of this work.

2.5.8 Relocation

The bitstream relocation technique states that a partial bitstream that is already implemented in an RP can be modified in such a way that it can be reconfigured in another RP. Therefore, the design time and also the memory need to store the partial bitstreams decrease, so on each reconfigurable hardware module requires only one partial bitstream. The relocation technique has three requirements: compatibility of partition pins, no presence of feed-through routes, and footprint compatibility. Below, these three features are explained in detail.

As described before, partition pins define an interface to connect the static logic with the RP. For a relocatable PR system, the partition pins must be at the same position within each RP to ensure compatibility in all relocatable partial bitstreams.

Figure 2.20 (a) and (b) shows an example of different placement of partition pins that leads to incompatible partial bitstreams. Comparing 2.20 (a) and (c), it is possible to see that the placement of partition pins are equal. The problem, in this case, is the presence of a feed-through route. The partial bitstream of (a) does not have the same feed-through route as the partial bitstream of (c). Then, a relocation of (a) into (c) disconnects the feed-through route, possibly generating an error in the static logic.

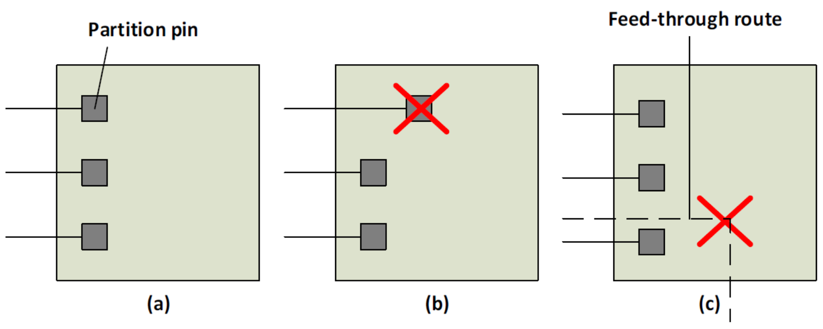


Figure 2.20: Partitions with feed-through route and different placement of partition pins (16).

There are two types of relocation, 1D and 2D. The former refers to move partial bitstreams only in the vertical direction; the latter also moves bitstreams in the horizontal direction. In Figure 2.21 it is shown 4 different footprints. The footprints of the RP 1 and 3 are identical, and then they are automatically compatible. However, the partitions in 2 and 4 are not compatible with the partitions 1 and 3, because these relocations cause a static logic overwriting. Finally, it is possible to note that only the partition 2 is relocated into all partitions because its footprint is contained in all the partitions.

2.5.9 Reconfiguration Styles

Figure 2.22 illustrates some possible arrangement configurations for the reconfigurable region.

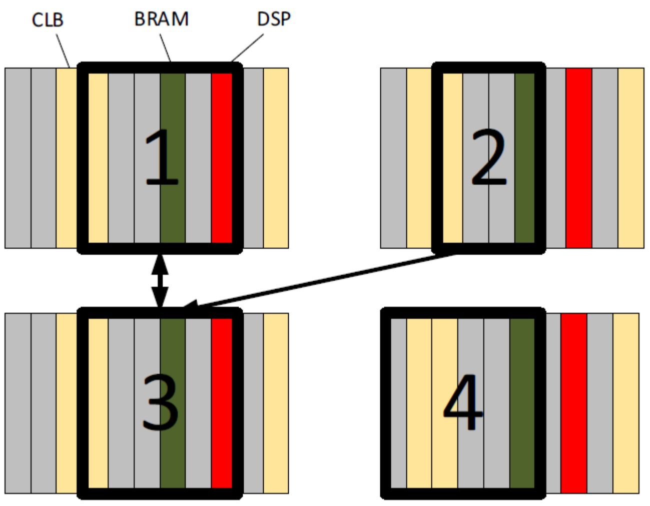


Figure 2.21: Different footprints of partitions implemented in an FPGA (16).

The simplest case is called *island style*. The island style allows hosting one RM per island, and a system might provide multiple islands. This type of reconfiguration style can be divided into two classes, *single island style* and *multi island style*.

The single island style states that a set of modules can run just on one specific island, while the multi island style allows the modules to be relocated to different islands. Therefore it is possible to have a system with multiple RPs but with the single island style if the modules are not relocatable. Currently, RTRLib only allows the single island style reconfiguration.

Figures 2.22B) and 2.22C) represent the internal fragmentation by tiling the reconfigurable region. This is motivated by the need to optimize the utilization of the FPGAs resources based on the idea that a module can occupy several tiles, and multiple RM can be hosted simultaneously in a reconfigurable region. If the reconfigurable region is tilled by the one-dimensional approach, the technique is called *slot-style* and the two-dimensional approach is called *grid-style* (17).

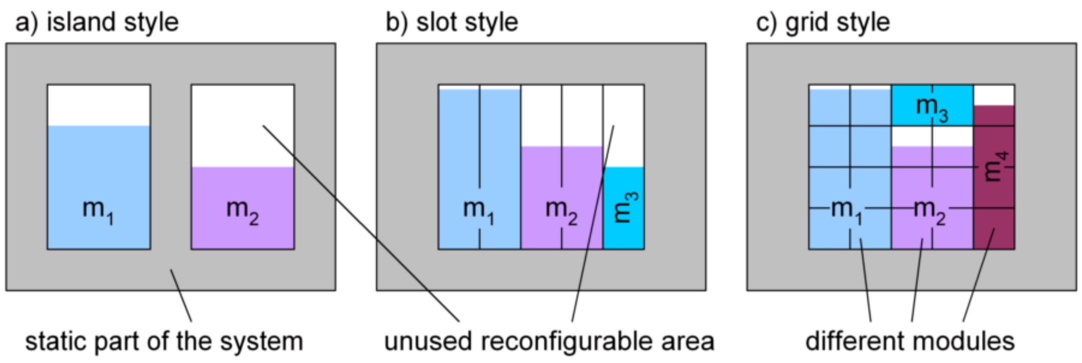


Figure 2.22: Types of reconfiguration styles (17).

2.6 FREERTOS

FreeRTOS is a portable real-time kernel, with which it is possible to construct applications that achieve critical requirements of real-time. Ideally suited to embedded applications that use microcontrollers or small microprocessors and demand execution in real-time. Some features can be highlighted, such as the capacity multi-task based on priorities, the simple scheduler, and semaphores to control the resource sharing between diverse tasks. FreeRTOS is distributed as C files that contain the configuration, and the APIs source files (41).

For the case of Xilinx applications, SDK generates a Board Support Package (BSP) for the hardware project. It contains an initialization code to exhibit the Zynq ARM processors and the drivers for the Zynq peripherals. Figure 2.23 presents the FreeRTOS kernel composition and services.

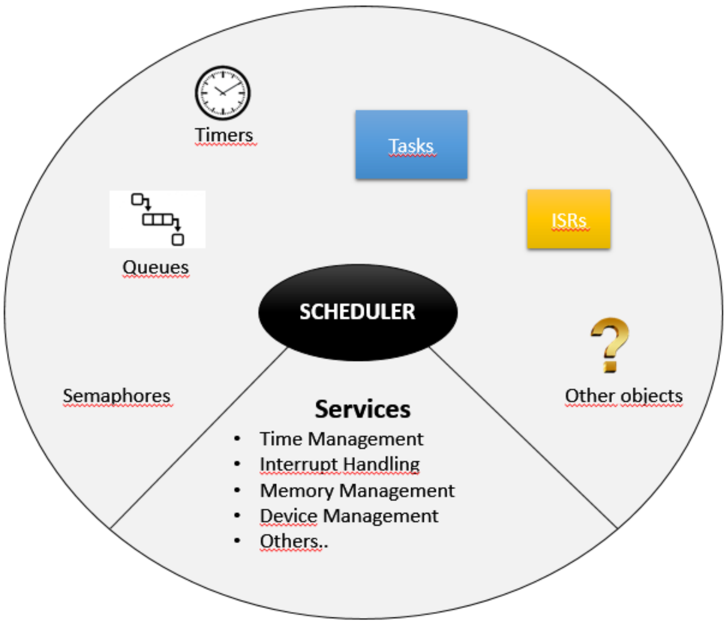


Figure 2.23: FreeRTOS kernel composition and services.

As can be viewed in Figure 2.23, the kernel is essentially composed of the scheduler and some objects used to implement the functions of the system. The main objects of the kernel are tasks, queues (conventional, mailboxes and queue-sets), semaphores (binary and counting), timers (one-shot timers, and auto-reload timers), and ISRs. These objects will be depicted in detail in the next Chapter, along with the FreeRTOS implementation proposed with RTRLib.

The scheduling algorithm is a software routine that decides which task in the ready state will transit to the running state. It can be configured, using the configuration source file, in three different modes of operation: *Prioritized Pre-emptive Scheduling with Time Slicing*, *Prioritized Pre-emptive Scheduling without Time Slicing*, and *Co-operative Scheduling*.

This is done by the configuration of the parameters of the pre-emption and time slicing usage. Pre-emption means that the scheduling algorithm pre-empts the task that is running when another task with higher priority enters the ready state. On the other side, in the case of tasks with the same priority, time-slicing is used to share processing time, even when none of the tasks enters the blocked state or yields.

Therefore, according to the desired system operation, the designer must define the scheduling algorithm, taking into account that the first one enables the two parameters, the second one uses just the pre-emptive, and for the cooperative mode, neither the pre-emptive nor the time-slicing are used.

2.7 RELATED WORK

A few academic tools have been developed for automating the entire design flow and overcome the difficulties in designing DPR systems. When designing dynamically partially reconfigurable(DPR) systems, several decisions must be adopted to guarantee the correctness of the implementations. The interface between the static logic and the reconfigurable regions, the routing isolation and decoupling mechanisms (42), (39), the reconfiguration strategy, and the high-level analysis are some of those decisions. This Section reviews some relevant propositions in this line found by a systematic review.

The systematic review is a method that allows the maximization of the search potential, which aims to find the maximum number of results in an organized way. There are several different methodologies to develop a systematic review, and the present work adopts the method presented by Akobeng (43) that can be summarized by the eight steps described below.

1. Delimitation of the research question.
2. Choose of the databases.
3. Definition of key-words for the search.
4. Search and storage of the papers.
5. Selection of the papers by the abstract, according to the inclusion and exclusion criteria.
6. Data extraction of the selected papers.
7. Analysis of the papers.
8. Synthesis and interpretation of the data.

2.7.1 Definition of the key-words

The key-words must be able to synthesize the essential ideas and concepts that are investigated in a study. In this context, it is also necessary to be careful in the definition in the key-word to ensure that they are sensible enough to provide consistent results and not so general, what would so many results that would not be possible to analyze.

The IEEE Thesaurus (44) and the IEEE Taxonomy (45) comprises the descriptors, or the most preferred terms, of the interest topics. The IEEE defines the IEEE Thesaurus as the terminology used in a controlled and well-defined fashion that permits to obtain better results in the searching process. Taking in mind the research question, they were used as the vocabulary reference for the definition of the key-words of the search.

The defined key-words were: *Partial Reconfiguration, Field Programmable Gate Array, System on Chip, Hardware Design*, and *Design Tool*. Where, only the *Partial Reconfiguration* key-word was not found in the IEEE Thesaurus (44), but it was included because it expresses the central research concept of the project. It was decided for this project, the research of the available partial reconfiguration design tools.

2.7.2 Search and selection

The key-words initially selected were compared to the author's key-word of the papers that were found in the first non-structured search to verify if they were able to track the references. From this analysis, the key-words were refined. However, the association of *"partial reconfiguration"* in conjugation with the other key-words previously defined would restrict to a set with less than 200 papers. To have a more representative set of papers to start the construction of the papers dataset to be analyzed, initially, a union set of six ways of the search was defined, as shown in the Table 2.4.

Table 2.4: Definition of the research key-words.

P1: Topic=("Partial Reconfiguration") AND ("FPGA")
P2: Topic=("Partial Reconfiguration Framework")
P3: Topic=("Partial Reconfiguration") AND ("Design Tool")
P4: Topic=("Dynamic and Partial Reconfiguration") AND Topic=("Design Flow")
P5: Topic=("Dynamically Reconfigurable Systems") AND ("FPGAs")
P6: Topic=("Reconfiguration") AND ("FPGAs") AND ("Automation")

NOTE: The term "Topic" above indicates a search in the title, key-words and abstract of the papers

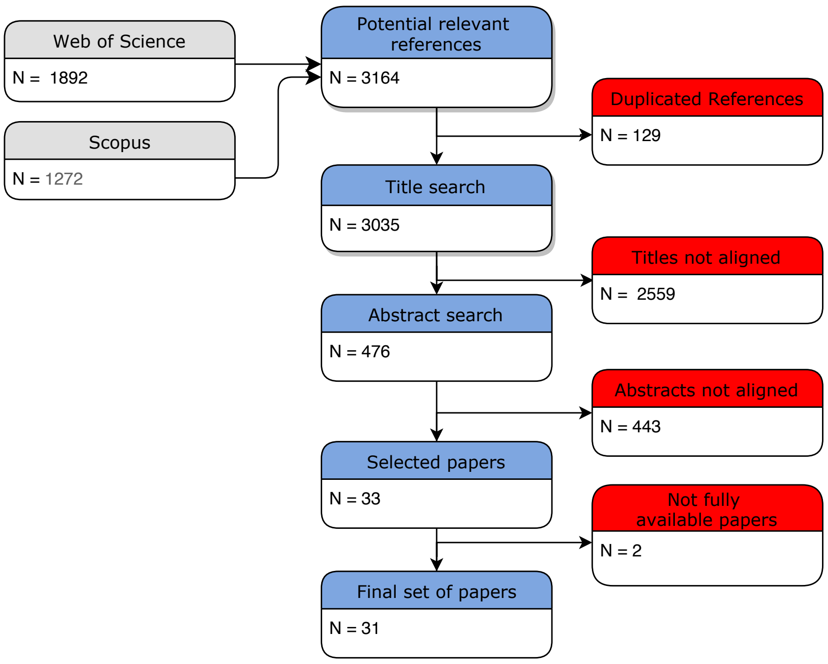


Figure 2.24: Search paper results.

Figure 2.24 shows a diagram with the obtained results in any step of the systematic review realized using the Web of science e Scopus databases. For the reference management, the ENDNOTE (46) and the Mendeley Desktop (47) were used.

2.7.3 Analisys, Synthesis, and Interpretation

As evidenced by the previous discussion, on the technical literature are available reports of numerous studies of the issues related to the use and effective exploitation of PR designs. However, the analysis presented here is concentrated in the 11 frameworks selected that are more closely related to this work. Table 2.5 summarizes the features provided by the works in the state-of-the-art compared with RTRLib.

Table 2.5: Comparision of the main PR design tools.

Feature	**Recobus Builder (48)**	**GoAhead (42)**	**RePaBit (16)**	**IMPRESS (39)**	**FASTER (49)**	**CAOS (50)**	**RTRLib**
Interface	Various macros	Direct wire binding	Bus macro	Virtual Interface	Not reported	Not reported	Partition Pins
PR-PR Communication	No	Yes	No	Yes	No	No	No
GUI	Yes	Yes	No	No	Not reported	Yes	Yes
HDL generation for RMs	No	No	No	No	No	using HLS	Only structural design
High-level analysis	Yes	No	No	No	Yes	Yes	Yes
Fault-Tolerance	No	No	No	No	No	No	Only hardware redundancy
Application generation	No	No	No	No	Yes	Yes	Standalone,Linux,FreeRTOS
IDE	ISE (XDL)	ISE (XDL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)
Supported devices	Virtex II/Pro and Spartan 3	Virtex and Spartan 6	Zynq Soc	Zynq Soc	Virtex 5 and Zynq	Xilinx VU9P	Zynq Soc

Koch et al. (48) present their contribution to the design of reconfigurable FPGA-based systems with a technique to promote point-to-point communication links between some RMs and the static part of the system and with their tool *RecoBus-builder* projected to automate the design steps required to construct RTR systems. Their communication methodology was constructed motivated by the desire of to permit a flexible integration of multiple modules with different sizes, through a sophisticated communication structure, that uses I/O bars, ideal for data streaming, and a RecoBus what allows the integration of multiple RMs into one shared area, by the idea of to partitioning the complete reconfigurable area with a fine grid. The RecoBus-builder considers all logic, placement, and route information to build an optimized hard macro, which means that, it utilizes the minimum amount of slices.

The GoAhead tool (42) has a complex communication architecture synthesis and is prepared to support other Xilinx FPGAs. Distinctly from the other tools presented here, the GoAhead Tool also provides static/partial decoupling, hierarchical reconfiguration, and reconfigurable region crossing. The GoAhead was used for implementing complex reconfigurable systems such as the ones presented in (52) and (53).

The RepaBit tool (16) makes use of bus macro interfaces for the RPs. It produces relocatable, in 1D and 2D, partial bitstreams for Xilinx Zynq devices. Compared to other works concerned with the relocation, the innovation present in RepaBit is the usage of the Xilinx Isolation Design Flow (IDF) to avoid feed-through routes.

The IMPRESS tool (39) develop a custom *virtual interface* based on fixed nodes shared between the static logic and the RPs, supports relocatable bitstreams by avoiding feed-through routes, permits communication between RPs through the use of the AXI interface, decouples the logic during the reconfiguration, and allows reconfiguration through the Processor Configuration Access Port (PCAP).

The FASTER tool (49) is an integrated semi-automatic framework that allows the development of reconfigurable heterogeneous MPSoC systems. The user must provide a C-based application, an XML file that contains information about the target architecture, and the HDL implementations for the hardware cores. The FASTER methodology is divided into four steps: *High-Level analysis*, *Optimizations for region and micro-reconfiguration*, *Compile-time scheduling*, and *mapping into reconfigurable regions*.

The CAOS platform (50) is designed to encourage the community to adopt, develop, and improve HPRC (High-Performance Reconfigurable Computers) systems. It comprehends the full process of application optimization, from the identification and optimization of the kernel functions to the generation of the runtime management for the target system.

Among the references presented in Table 2.5 other relevant approaches are presented in (54), (55), (56), (57), and (58) as described as follows.

Like ReCoBus-Builder, OpenPR (54) uses hard macros for the interface routing and special blocker macros to constrain the routing. OpenPR is an open-source PR toolkit very similar to the Xilinx PR toolkit. The OpenPR toolkit was written in C/C++, the choice of the language was justified by the needed performance of the system, motivated by the suggestions presented in (59) of use C++ instead of Java to improve performance and memory overhead. It is essential to highlight the limitation that the signals have to just connect to the reconfigurable area along the top and bottom border reported by the authors in (54), which does not favor routing interface signals into vertically aligned carry chains since OpenPR permits the use of slot base as the reconfiguration style.

Another research focused on the reduction of PR design time effort was developed by Yousuf (55) and named DAPR (Design Automation for Partially Reconfigurable). To fulfill its goal, the DAPR was modeled based on the idea of isolating the designers from the PR design low-level specific details, but it supports only Virtex-4 FPGAs. The simulated annealing algorithm (60) is used to optimize the floorplanning, to find the optimum PR design, since this process requires a full place and route of the architecture, the fitness evaluation demands high computational efforts and can take several hours.

Oomen et al. presented the first tool that aims to create DPR systems using the Xilinx Vivado. It uses bus macro interfaces based on LUTs for the communication between the static logic and the RPs and targets Virtex-7 FPGAs. *Oomen et al.* also presented a case study that does the reconfiguration through the Internal Configuration Access Port (ICAP).

This work introduces RTRLib, a high-level modeling tool that automatically provides TCL scripts for developing DPR systems on Xilinx Zynq devices, allowing the early estimation of the resource utilization, latency, and reconfiguration time of each RM. The features mentioned above are possible because the RTRLib is a platform-based design tool that uses the semi-formal refinement-by-replacement methodology,(61) allowing non-experts users to make use of functional blocks, interconnecting them as a process network to express a specific behavior of the system. Then, each functional block is refined and replaced by previously characterized IP-cores, enabling an introspection process to be performed, obtaining the structural HDL code of each RM. Since specific details of the IPs are known, the tool is capable of providing an early estimation of the resources consumption, latency and throughput (assuming clock conditions)of each RM, guiding the designer regarding technical issues such as the size of each RP, the communication interface between RPs and the reconfiguration process. In the case of hardware redundancy schemes for fault-tolerant systems, the tool also estimates the failure rate and reliability of each RM.

High-level modeling tools for DPR systems can be used to guide the design during the development process. For instance, a common solution to overcome the data dependency between RPs during the reconfiguration process is the adoption of FIFOs. However, determining the size of the FIFO depends directly on the latency of the RMs and on the reconfiguration time. Therefore, the early estimation of the size of the RPs, the latency and throughput of each RM, and the reconfiguration time are relevant to guide the decision making as well as the development process. In the case of fault-tolerant systems, the estimated hardware resources used for each RM can be used to estimate the size in MB of the configuration memory for each RP. This information is useful to estimate, under specific conditions, the soft error rate caused by SEU (single event upsets) affecting configuration memory cells and BRAMs. Thus estimating the failure in time of the systems (18).

3 RTRLIB METHODOLOGY AND IMPLEMENTATION

This Chapter describes step by step the various phases composing the RTRLib proposed methodology. It assists the designers through every implementation stage, starting from the project specification. In the case of the entire hardware design, it ends with the final bitstreams and, in the case of some SW/HW co-design, ending with the generation of a template of the software application. Also, this Chapter dedicates to the RTRLib diagnosis. It has the intention of describing in detail the usage, requirements, limitations, and functionalities to understand which are the functional capabilities of the tool and how it works.

3.1 INTRODUCTORY REMARKS

RTRLib is a platform-based and model-based design tool for DPR systems, which uses Matlab and Simulink as a high-level development framework. The idea behind the tool is to promote the construction experience to become more flexible. Since Simulink is a block diagram environment for Model-Based Design, it supports simulation, automatic code generation, and continuous testing of embedded applications. In particular, RTRLib explores the Simulink's *Variant Systems* functionality and uses these features to provide a simple interface of utilization, which allows non-expert users to develop RTR solutions.

RTRLib is based on the semi-formal refinement-by-replacement methodology (61). It has been developed to simulate and analyze in design time the system behavior during the reconfiguration process. RTRLib also automatically provides the scripts that allow the partial reconfiguration solution to be mapped on FPGA-based System-on-Chips (SoCs).

The tool is modeled based on several pre-characterized IP-Cores available on FPLib, (62) (63). Therefore, to estimate the size of each RP, the designer must implement the high-level description of the RMs using the available IP-Cores. RTRLib has been prepared to translate the description of the RM in the RTRLib environment into its HDL description. However, this process is also based on these IP-Cores, which enforces the need to use them when the system is being modeled.

The tool has been tested with Matlab (2016a) and Vivado (2016.4/2017.4). Figure 3.1 shows a block diagram of the main flow phases; each one of them will be depicted in detail within this Chapter.

3.2 PROPOSED METHODOLOGY

The proposed RTRLib design flow can be divided into four different approaches, mapped in the following lanes: *highlevel framework, hardware redundancy design methodology, PR hardware design flow*, and *PR software design flow*. Figure 3.2 illustrates the overall RTRLib design flow.

The first step of the design flow is the system modeling in a high-level description. Here the designer must provide all the details concerned with the desired system. It is possible to model only hardware

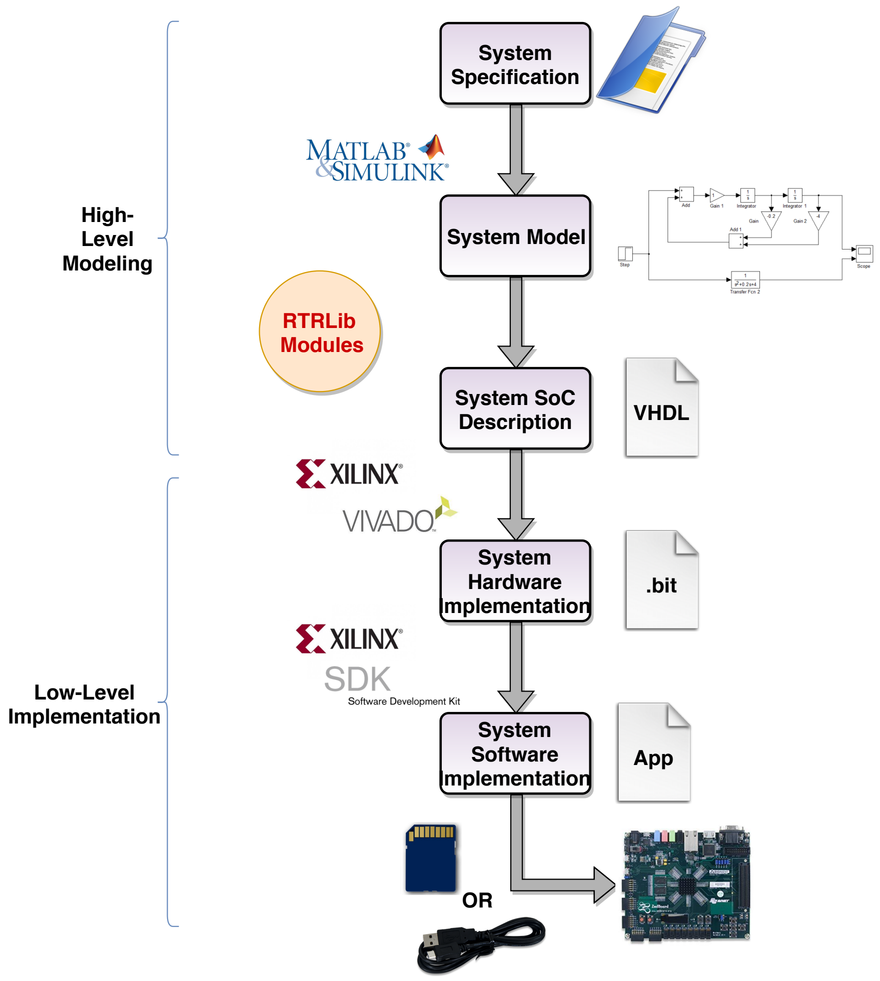


Figure 3.1: RTRLib proposed methodological flow block diagram

systems and also SoC solutions based on the ARM Cortex A9 processor embedded in the Zynq devices.

In the context of the hardware redundancy design, the RTRLib aims to implement a fault-tolerant design methodology based on the structural design of an NMR solution (64). The designer provides the mission details, such as the FPGA family and mission time of each module. With this data, the resources estimation and size in MB of each RP, the system reliability is estimated and provided. Two bio-inspired methodologies were applied (mono-objective and multi-objective) where the primary goal is to find the optimum redundancy level for each stage, that is, to discover the optimal number of replicas of each module in each step.

Taking the model constructed from the description, for both cases redundant and non-redundant, the RTRLib initiates the PR hardware design flow. Its main result is a script with the TCL commands. This script has to be used later in the Vivado TCL console environment to generate the system automatically according to the specifications provided by the designer.

From the point of view of the PR software design flow, the Zynq platform enables the utilization of two different embedded OS, the C/C++ standalone and the FreeRTOS823 Xilinx. At the configuration of the RTRLib ARM block, the user can select from any of them. In the specific case of the C/C++ standalone solution, the RTRLib creates a template that allows the reconfiguration through software triggers

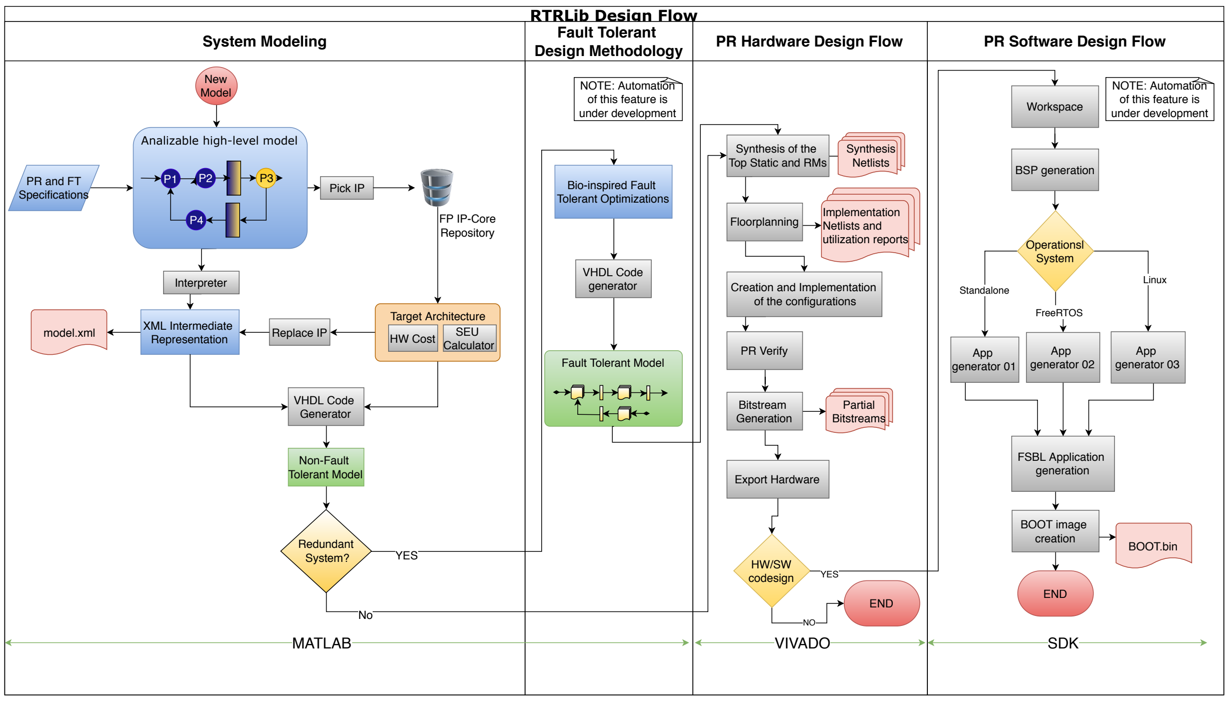


Figure 3.2: Overview of the RTRlib proposed design flow. The blue rectangles represent high-level models. The green squares represent the version of the fault-tolerant models. The tasks related to the design transformation and characterization, such as the automatic VHDL code generation, are identified by the gray boxes. The generated files are represented in magenta, while the decision points are shown in yellow.

sent through the UART. In the case of the FreeRTOS solutions, the RTRLib provides a visual and simple interface to construct personalized applications.

3.3 HIGH-LEVEL MODELING FRAMEWORK

As previously mentioned, RTRLib uses the Simulink framework to obtain the system description, allowing for fast prototyping and easy error detection. This Section describes the available functional blocks in the RTRLib Library and the modeling process of a DPR system using the tool.

3.3.1 Functional Blocks

In RTRLib, the DPR system must be modeled as a process network, where each process is represented as a functional block. Figure 3.3 presents the functional blocks used by RTRLib during the high-level description of a system. Each block must be inserted and parametrized according to the specification, Table 3.1 presents the configuration parameters of the main blocks.

The *Top_module block* contains the initial specification of the system. Since RTRLib is a platform-based design tool, the user must select one of the two supported platforms (Zedboard and Zybo development kits). Concerning to the PR strategy, it supports three options:

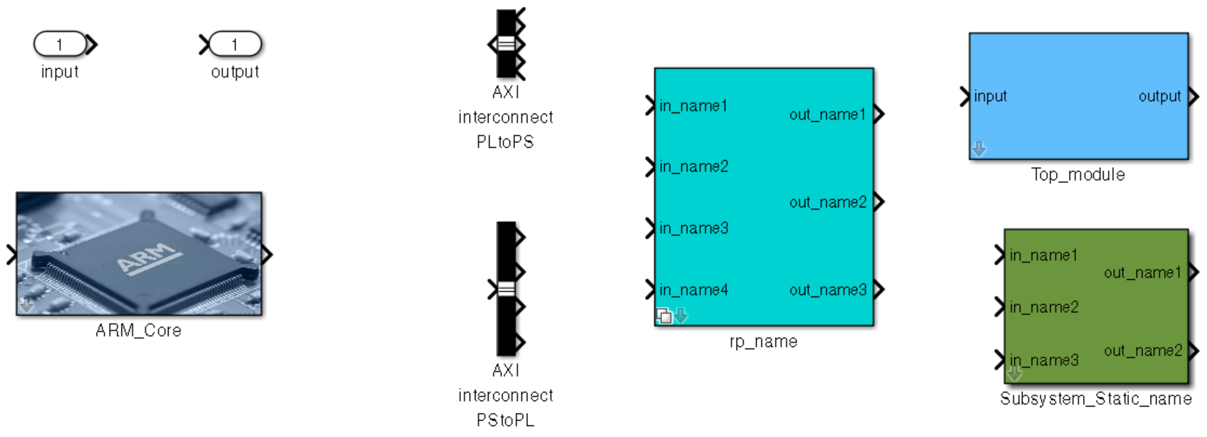


Figure 3.3: RTRLib - Functional blocks.

1. *PRC (Partial Reconfigurable Controller) + ICAP:* This strategy uses the PRC IP. When hardware trigger events occur, the PRC pulls partial bitstreams from memory and delivers them to the ICAP. The PRC is configured according to the number and names of the RPs, number, and names of the RMs, and also the address and size of each RM.

2. *PCAP + ARM:* This strategy allows only software triggers events. In this case, the ARM controls the reconfiguration process through the PCAP.

3. *Manual:* By this strategy, using the Vivado Hardware Manager Tool through the JTAG connection, the user can manually configure each RP by downloading each partial bitstream.

The *Arm_core* contains some PS resources to be used in the design (SD-CARD, UART, XADC, DDR3 Controller, among others). Also, the type of application must be selected between standalone or FreeRTOS.

The RPs blocks are used for each RP in the system. They are defined as *variant systems*. On the other hand, static logic IPs are also defined in the model. The only parameter available is related to the connection that indicates to the RTRLib back-end if the block is connected to the ARM or not. Finally, as it can be viewed in Figure 3.3, it is also available the connection blocks, to connect the PS to the PL through the AXI-Lite interface.

Table 3.1: Configuration parameters of the main RTRLib blocks.

Top_module	
Strategy	PRC/ARM/Manual
Board	Zedboard/Zybo
RPs and Static IPs	
Connection	ARM/TOP
ARM_Core	
SD Card	enable/disable
UART	enable/disable
XADC	enable/disable
DDR3 Controller	enable/disable
Application Type	Standalone/FreeRTOS

3.3.2 Functional Blocks for FreeRTOS applications

Beyond the blocks previously described, for the case of an RTOS application, RTRLib also has a parametrized block to construct the structure of a FreeRTOS-based application. The idea behind this block follows the RTRLib general idea, that is, the designing simplification. The system creates the structure of the application but does not generate the logic to be implemented.

The library is composed of one block dedicated to the general parameters that configure the system and, inside that, some blocks that configure the functions to be used in the software application. Figure 3.4 (A) presents the FreeRTOS block, and Figure 3.4 (B) presents the blocks used to represent the APIs that composes the RTOS application to be developed. Table 3.2 presents the configuration parameters of the FreeRTOS block. Similarly, in Table 3.3 is shown the settings for the APIs blocks.

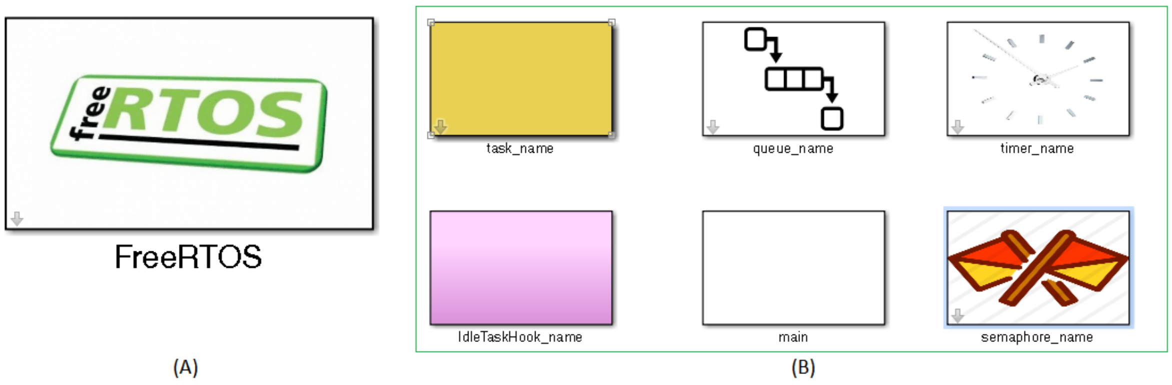


Figure 3.4: (A) FreeRTOS block (B) RTOS functional blocks.

Table 3.2: Configuration parameters of the FreeRTOS block.

FreeRTOS		
General tasks parameters	Maximum priority	For generic method, keep as low as possible. For architecture optimized the value must be lower than 32.
	Method of tasks priorities	Generic/Architecture optimized
	Idle hook task	enable/disable
General scheduler parameters	Scheduler algorithm	Prioritized pre-emptive scheduling with time slicing
		Prioritized pre-emptive scheduling without time slicing
		Co-operative scheduling
General timers parameters	Utilization of software timers	enable/disable
	RTOS daemon stack depth	Configuration of the daemon (timer service) task, created when the scheduler is initialized.
	RTOS daemon priority	
	Length of the timer command queue	Configuration of the queue that receives commands from the calling task to the daemon task.
	vTask suspend	enable/disable
General interrupt parameters	Min. numerical priority	Related to the interrupts priorities itself
	Max. logic priority	Indicates the interrupts precedence over other interrupts

Table 3.3: Configuration parameters of the RTOS functional blocks.

Task	
Priority	Can be assigned from 0, which is the lowest priority
Stack depth	The number of words not the number of bytes
Self deleting	enable/disable
Delay	None/Absolute/Relative
Delay Period	When using delay, this value is converted into number of ticks that the task remain in the blocked state
Semaphore	
Type	Binary/Counting
Maximun Value	Range of a counting semaphore
Minimun Value	
Timer	
Type	One-Shot/Auto-Reload
Timer's period	Specified in ticks
Queue	
Type	Queue/Mailbox/Queue Set
Number of items	Not available for mailbox type
Size of each data	Size in bytes
Max number of items the queue set has to hold at one time	Parameter available only for the queue set type

3.3.3 Design steps for modeling DPR systems using RTRLib.

All the system, including the static and the reconfigurable part, must be modeled as a process network. Currently, the tool supports only synchronous models. In this context, the user has to attempt to some specific instructions and steps to model a DPR system using RTRLib:

1. The Top Module Block must be instantiated and configured.

2. Inside the Top Module, the RPs must be instantiated.

3. RTR systems are modeled as Variant Systems, then the blocks that represent the RPs have to be set as a variant object. This is done going to File ⇒ Model Properties and configuring the PreLoadFcn* for each RM in all RPs.

4. Into each RP, the RMs have to be instantiated. At this level, the blocks must not be connected. At simulation, connectivity is automatically determined, based on the active variant and port name matching. It is important to remember that to estimate the size of the RPs is necessary to model the RMs using IP-Cores.

5. All the Input/Output bit width must be set. The user provides the desired board mapping through a .xlsx file, where the first column refers to the name of the IO and the second to the pin.

6. Then, in the case of an HW/SW co-design, the Arm_core has to be instantiated, and its parameters configured, defining which resources (SD-CARD, UART, XADC, DDR3 Controller) will be used and the type of application.

7. For the case of the FreeRTOS application, the user can also use the related blocks to generate the system configuration file. In this case, the FreeRTOS block has to be configured and inserted into the ARM block. To do the structure application generation, all the APIs that will be used must be inserted inside the FreeRTOS block.

8. For each RP, the pblock has to be created and positioned using the GUI. All the PR requirements must be considered to ensure the designed system attend to the DRC rules. As described previously, at this step, RTRLib prints some warning and error messages to assist the user is this process.

9. Finally, all the connections between the blocks and through the AXI interconnect from PS to PL and from PL to PS must be made.

10. The platform is not prepared yet for the entire generation of the HDLs for the RMs logic. Currently, it just creates the structure (port map of the components). Therefore, for the actual version of the RTRLib, the user must provide, in a dedicated folder, the HDL files that implement the logic of each RM. The files must be inserted according to the name of the related RP and RM.

11. For the software design, there is also a folder to receive the user's files to be imported into the application. The software files automatically generated by RTRLib are also created in the same folder.

Figure 3.5 presents a Neuron Network as an example of high-level modeling using RTRLib. In this case, the system has one RP that is connected to the ARM. Inside it, there are 3 RMs. Finally, the structure of the neurons is described based on the available IP-Cores of addition and multiplication.

3.4 LOW-LEVEL IMPLEMENTATION

This Section is dedicated to the explanation of the RTRLib's back-end implementation. From the obtained high-level modeling, RTRLib extracts the configuration parameters and creates the scripts needed for the SoC implementation to be used in Xilinx Vivado and SDK. Additionally, some files in HDL, XDC, C, and .h extensions to automate some steps of the flow are generated. Therefore, it is presented the minutiae of the HW/SW .tcl scripts generation, and also which files are automatically created by the tool.

3.4.1 Proposed Fault Tolerant Design Methodology

When the fault-tolerant feature is selected, RTRLib produces a fault-tolerant solution based on the hardware redundancy scheme (64). In this case, the designer must provide useful parameters such as the mission details provided by the designer, such as the Mean Time to Failure (MTTF) of each module and the mission time and also the hardware cost estimation.

Currently, two bio-inspired techniques are applied to solve the optimization problem, the mono-objective Particle Swarm Optimization (PSO) (65) and the Multi-Objective Differential Evolution (MODE) (66).

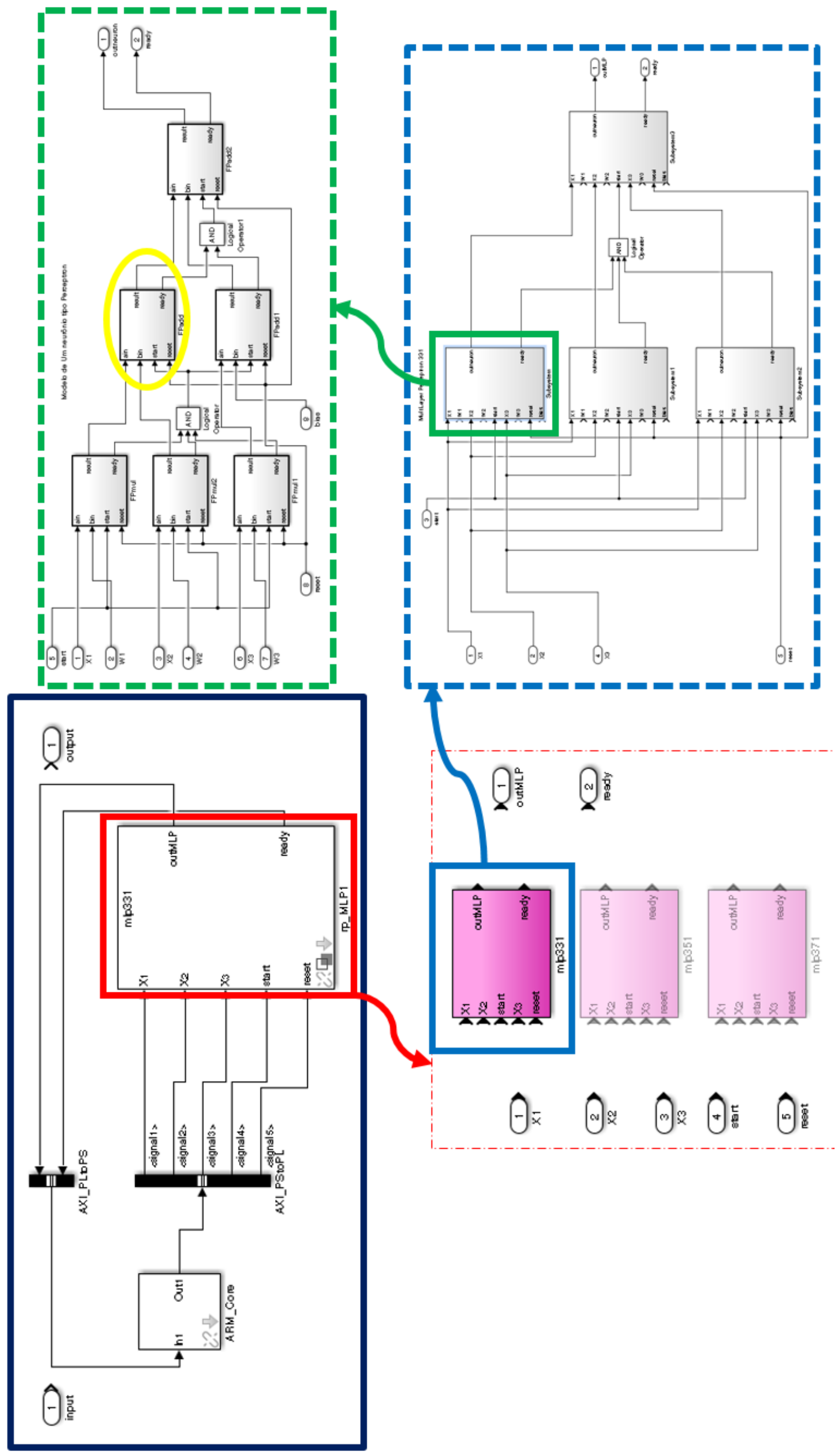


Figure 3.5: Example of using RTRLib

Currently, the FT Methodology is based on the optimization of two main criteria, the reliability and the hardware cost, R(x), and C(x), respectively. The main goal is to find the optimum redundancy level for each stage, that is, to discover in each stage the optimal number of replicas of the modules. Equations 3.1 and 3.2 present the definition of each one.

$$R(x) = \prod_{i=1}^{N} \sum_{i=n+1}^{2n+1} (2n+1)R^i(1-R)^{2n+1-i} \tag{3.1}$$

$$C(x) = p_1 LUTs + p_2 FFs + p_3 DSPs + p_4 BRAMs \tag{3.2}$$

In Equation 3.2 the number of LUTs, FFs, DSPs and BRAMs are expressed in percentage, and the parameters p_1, p_2, p_3 and p_4 are defined by the designer according to the need of the solution. Therefore, in the case of the mono-objective optimization, the fitness function is a result of the weighted sum between these criteria, like it is expressed by Equation 3.3.

$$f(x) = w_1 R(x) + w_2(1 - C(x)) + P(x) \tag{3.3}$$

Where w_1 and w_2 are the weights that must be adjusted by the designer to determine the impact of each criterion in the optimization process, and *P(x)* is a penalty function. The fault-tolerant model is constructed using a VHDL generator. It creates a *Port Map* structure to automatically instantiate and connects the IP-Cores. The final estimation of the final reliability and hardware cost is then provided

3.4.2 IP-core repository

Currently, the RTRLib is essentially modeled based on the IP-Core repository reported in (62) and (63), which is composed of floating-point units that implement arithmetic and trigonometric operations. The arithmetic based on floating-point was chosen by the need to implement solutions that require a satisfactory relationship between precision, power consumption, and performance. The implementation was realized using the IEEE-754 standard. This standard allows the designer to work with single precision (32 bits), double precision (64 bits), and even with a custom representation according to the precision requirements of the application.

Currently, the repository is composed of the floating-point units in VHDL that implement the following operations: addition and subtraction, multiplication, division, square root, sine and cosine, arctangent, and exponential. It was created a script that receives the word size and the selected Zynq device and re-synthesizes the IP-Cores in an out-of-context mode. As the system is modeled based on these pre-characterized IP-Cores, it is possible to have an early estimation of some essential features in the DPR design. For instance, the utilization resources in terms of LUTs, FFs, DSPs, and BRAMs for the target FPGA, and the latency in clock cycles. In the case of hardware redundancy designs, the failure rate and reliability of each RM can be estimated for the target device under specific conditions (18).

Figure 3.6 shows an example of the block that implements the sine and cosine operation, depicting the latency of 53 clock cycles. All the other IPs were analogously constructed at high-level.

This description allows the users to do preliminary tests and simulations with their architecture, even in the high-level phase of the design. It reduces the time of extensive behavioral simulation, taking in mind that many errors can be detected in this design step.

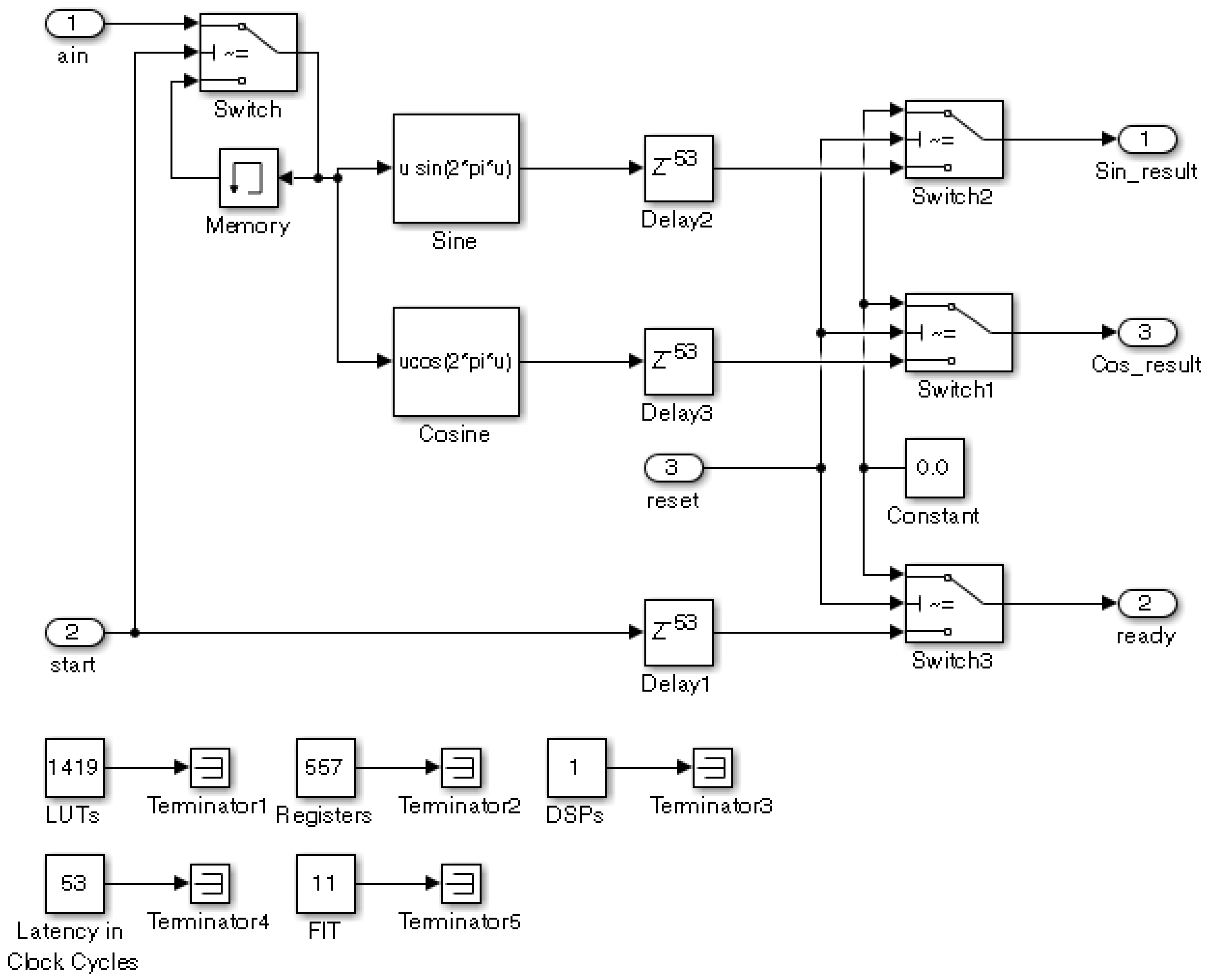


Figure 3.6: Example of the IP-Cores High-Level Description. The utilization report for the case of words with 27 bits and the FIT value estimated from Xilinx Reliability Report (18).

3.4.3 Floorplanning GUI

A GUI-based floorplanning tool was developed to manually define the placement and size of the pblocks, where each RM will be implemented according to its resource requirements. The implementation of the floorplanning tool can be divided into four steps: (a) device mapping; (b) resources estimation of the pblock; (c) range of slices, DSPs and BRAMs; and (d) warning and error messages.

It was created a function that reads the coordinates of the desired points in the image of the device. At this stage, the boundaries of the chip, the boundaries of each clock region, and the static region of the chip are mapped. This information is used to guide the user in the process of pblock positioning.

According to the information about available resources in each column of the device, the positions of each BRAM, DSP, and CLB were also mapped. Figure 3.7 shows the coordinates of the DSPs and BRAMs corresponding to the Zedboard (Zynq 7020 device). The process of mapping the CLBs and its corresponding slices was similar.

To estimate the resources of the pblock, the module compares the area selected by the user with the

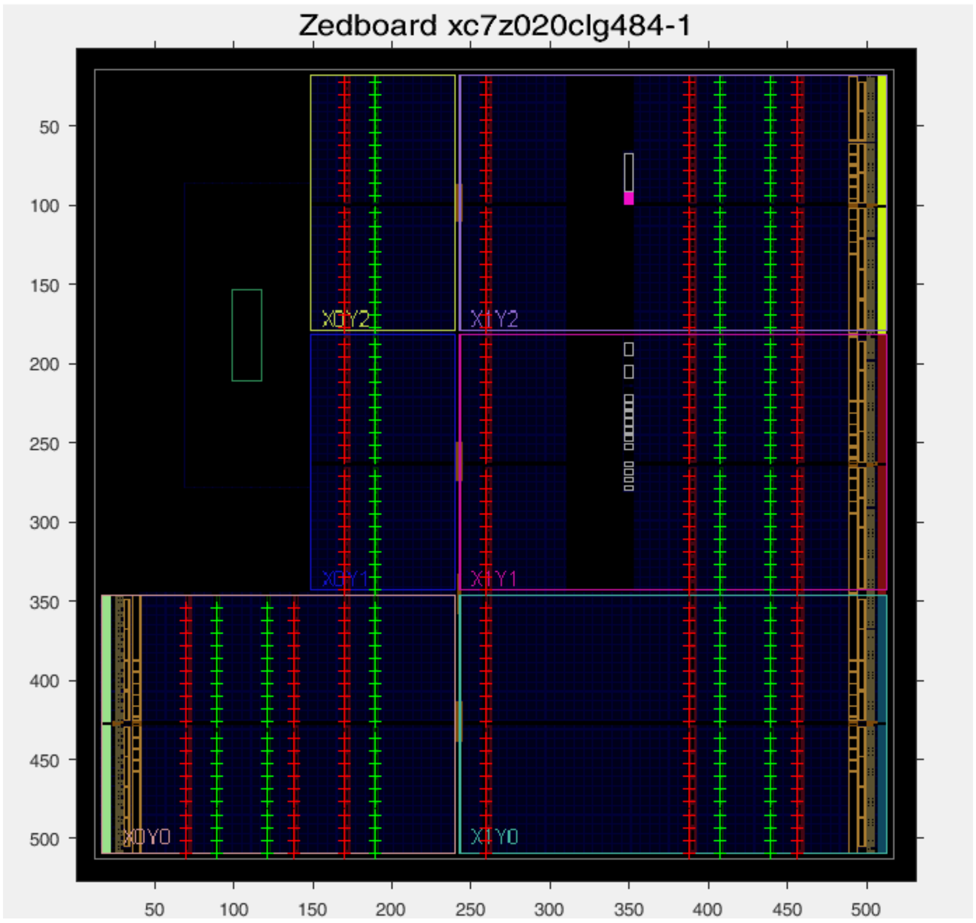
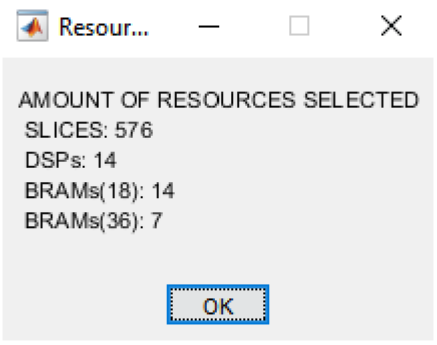


Figure 3.7: Mapping of the coordinates of the DSPs (green points) and the BRAMs (red points) for the Zedboard (Zynq 7020 device).

area in the figure occupied by each logic element (previously mapped), and it computes the presence of this element in the selected area. The final amount of resources is printed to the user, as previously mentioned.

After positioning the pblock, the system prints the resource estimation in terms of slices, DSPs, BRAMs(18), and BRAMs(36) of the selected area, as shown in Figure 3.8. If the selected resources are enough, the user can finish the floorplanning process; otherwise, the user must recreate the rectangle of this pblock.



```
AMOUNT OF RESOURCES SELECTED
 SLICES: 576
 DSPs: 14
 BRAMs(18): 14
 BRAMs(36): 7
```

Figure 3.8: Screen of resources estimation of the selected area during high-level pblock floorplanning.

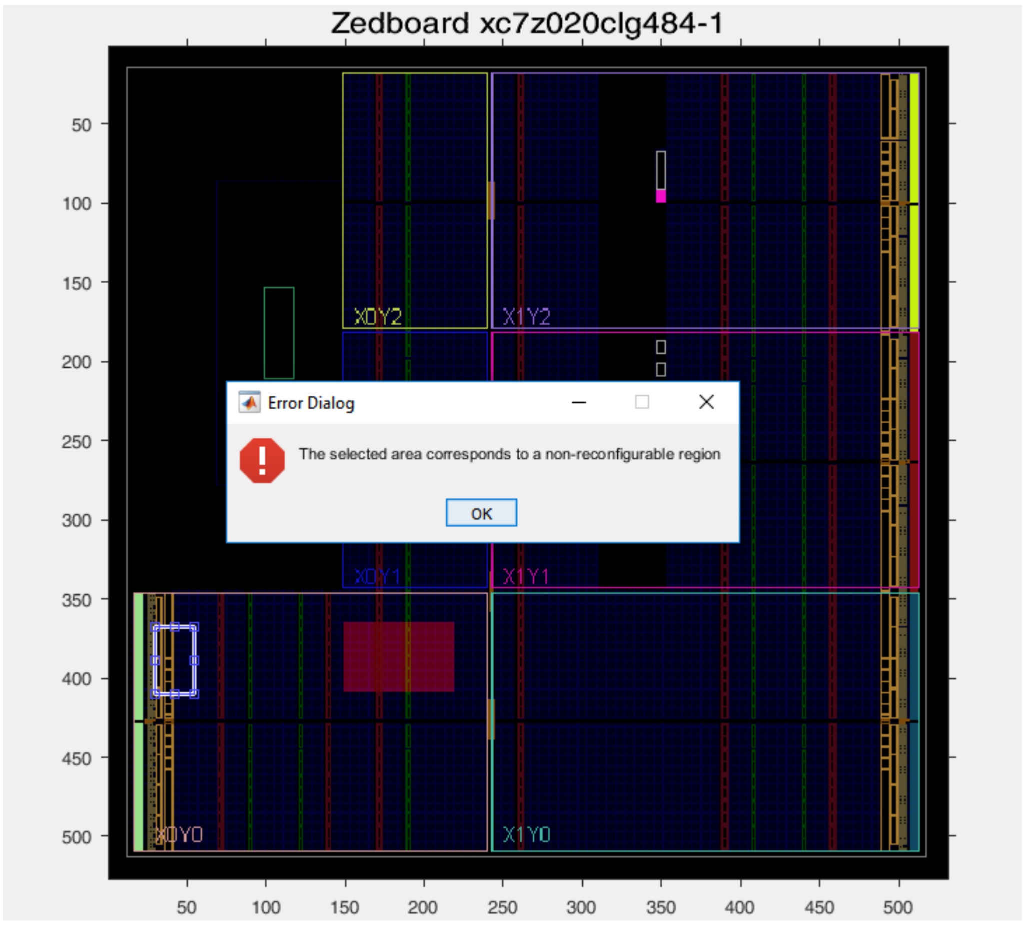
Finally, the module stores the corresponding range of slices, DSPs, and BRAMs and sends it as parameters to the main RTRLib script generation module.

In relation to the warning and error messages, several DRC rules must be accomplished to ensure the effectiveness of the placement and routing. RTRLib is equipped with some error and warning messages that prevent the user from making the most common mistakes, as listed below. Figure 3.9 shows an example of floorplanning through the GUI with an error message.

In the case of multiple RP positioned in the same column, the RTRLib identifies it as an error to attend the granularity condition. It is important to point out that this condition is only possible on Zynq Ultrascale devices.

Another error occurs in the condition of overlapping of two different reconfigurable areas. Concerning the selected region of the FPGA, the user must attempt to some details, such as avoiding to place pblocks out of the borders of the chip and selecting areas containing. Two examples are the pblock placed out of the borders of the chip and the selected area containing non-reconfigurable resources.

There are some cases that RTRLib allows the positioning of the pblock, but prints a warning message. For instance, when the RP crosses two or more clock regions and when the selected amount resource is not enough in relation to the estimated amount of resources. There are two other issues related to the size of the RP, the minimum width, and the availability of resources to place the partition pins.



The selected area corresponds to a non-reconfigurable region

Figure 3.9: RTRLib Floorplanning GUI showing a Zedboard FPGA.

3.4.4 Hardware Script Generation

From the high-level description, RTRLib generates a TCL script that must be executed in the Vivado Design Suite. Taking into account specific conditions, the RTRLib aims to automate the entire design flow from the synthesis out-of-context of the specifications of the modules to the partial bitstream generation and hardware exportation.

Initially, the RTRLib creates the folder structure for dynamic reconfiguration design flow and the repository with the available IP-Cores.

Then the script verifies which RP is connected to the ARM through the AXI Interconnect. A wrapper, according to the AXI-lite interface specifications, is created for each RP connected to the ARM processor.

Next, it creates the project, with the specification of the target language and the device. Then it creates a new block design and instantiates into it all the IP-Cores related to those specified in the high-level description, including the ARM-Core, all the RPs, and GPIOs. For the PRC strategy, the PRC IP-Core is included and configured according to the requirements of the system's reconfigurable part. This is done in the function of some parameters: number and names of RPs, number, and names of RMs, triggers' type, triggers' type, address, and size of each RM.

Once the block design is already configured, the top level is synthesized. All the HDL descriptions of the modules are synthesized in out-of-context mode. When the synthesis process is done, the report utilization is generated in text files to provide more precise information about the resources consumption in terms of LUTs, slices, DSPs, BRAMs, FFs, and multiplexers.

The floorplanning is executed according to the user's specifications through the GUI, setting the cells related to the RPs as hardware reconfigurable, and creating the pblocks related to each RP. Taking into account the granularity of the Zynq family, the snapping mode property of the pblocks is enabled. The RPs not connected to the ARM are configured to reset after the reconfiguration process.

Since the user has already provided the constraints related to the IO pins through a .xslx file, the pins used to produce the hardware triggers have to be described in this file too. The system reads the constraints and creates the .xdc template file for each board. This module sets the features related to the bank and pins for each IO.

For the rest of the hardware design flow, the steps executed by the script are based on the partial reconfiguration design flow, specifically, the generation of all the configurations, the verification of the configurations, and the partial bitstreams creation. For each RP is necessary to create the bitstreams to restore it to a specific configuration and a blanking bitstream to configure it as an empty module. Therefore, it is not necessary to compute and to create all the possible reconfiguration combinations between all the reconfigurable modules. With all the bitstreams related to the static and the partial design generated, the hardware is exported in the .hdf format.

Finally, the script verifies if the system is an HW/SW codesign. In the case that the project is a hardware-only design, it closes the project and finishes the flow. If the project is an HW/SW codesign, the script launches the SDK and initiates the DPR software design flow. Thus, if the solution makes use of the PCAP, then the user can take advantage of the software template produced by RTRLib containing the necessary API functions to perform the reconfiguration process.

With the generated bitstreams, the reconfiguration time can be calculated. The total configuration time is directly related to the size of the partial bitstream and the bandwidth of the configuration port that is being used. Table 3.4 shows the maximum bandwidth for the 7 series devices.

Table 3.4: Maximum Bandwidths for Configuration Ports in 7 Series Architectures (15).

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

3.4.5 Software Script Generation

From the high-level description, the RTRLib generates a TCL script that must be executed in the Xilinx SDK. Concerning the software specifications, the platform is prepared for different approaches. Taking into account specific conditions, the script guides the designer from the workspace creation to the BOOT file generation.

1. Within the created workspace, the first step is the Board Support Package (BSP) generation. The BSP is generated according to the hardware specification contained in the .hdf file exported previously during the hardware design flow. By default, the first core of the ARM Cortex-A9 embedded in the PS is selected, and the libraries *xilffs* and *xilrsa* are inserted due the FAT file system allows partial bit files to be read. The BSP is also created in the function of the selected type of application that will run into it.
2. With the changes done to the BSP, the mss file is updated, and the BSP sources are regenerated with the previous changes done in the BSP settings.
3. Then, the application is generated targeting the previous BSP. All the files in the application folder are imported, including the files automatically created and the user's ones.
4. Since the system has a PR approach, the script also creates the first stage bootloader application, *Zynq FSBL.*
5. For standalone and RTOS application types, the platform has different approaches that will be described in detail in the following Sections.
6. Lastly, the project is built, and the BOOT file is created.
7. If the user wants to modify the generated applications, this must be done manually before building the project.

3.4.6 Generation of the FreeRTOS application

FreeRTOSconfig.h is the FreeRTOS header file. It contains the general settings of the software system. According to the target device, some constants are previously defined, so RTRLib creates the FreeRTOSconfig.h based on the constants to applications that target Xilinx devices available in (67). Besides these constants, the header also contains the general parameters of the desired application. The block FreeRTOS in RTRLib is prepared to receive the main parameters of a general application. Then, the system parses this data and modifies the Xilinx FreeRTOSconfig.h template according to the user's specifications. Regarding the C application, it is important to mention that the RTRLib generates the structure of the application. Algorithm 1 presents the pseudocode of the FreeRTOS template generated by RTRLib.

Algorithm 1: Pseudocode of the template generated for FreeRTOS applications.

```
input  : FreeRTOS_param: Struct with the data of the blocks under the FreeRTOS block,
         name: Name of the design
begin
    Insert the default libraries;
    Include the timer constant;
    for k=1 to number of tasks do
        Create the tasks prototypes;
        Insert the tasks handles;
    end
    for j=1 to number of queues do
        Create the timers prototypes;
    end
    for k=1 to number of timers do
        Insert the queues handles;
        Insert the timers handles;
    end
    /* The users variable goes here */
    Initialize and configure the GPIOs;
    for k=1 to number of tasks do
        Create tasks;
    end
    for j=1 to number of queues do
        if Type = Queue Set then
            Create queue Set;
        end
        else if Type = Mailbox then
            Create Mailbox;
        end
        else
            Create queue;
        end
        Check if the queues were created;
    end
end
```

```
    for k=1 to number of timers do
        if Type = One Shot Timer then
            pdFalse;
        end
        else
            pdTrue;
        end
        Create timers;
        Check if the timers were created;
    end
    Start the scheduler;
    for k=1 to number of tasks do
        Describe the tasks body;
        /* The user must insert the logic implemented by the tasks here      */
        if Task has absolute delay then
            Insert the absolute delay in number o ticks;
        end
        else if Task has relative delay then
            Insert the absolute delay in number o ticks;
            Insert the TickType xLasWakeTime and get the TickCount;
        end
        if The task deletes itself then
            Add the vTaskDelete API
        end
        for k=1 to number of timers do
            Describe the timers body;
            /* The user must insert the logic implemented by the timers here   */
        end
    end
end
```

3.4.7 Application with PRC generation

This module creates a C-based standalone application template for the case of a system with multiple RPs reconfigured by the PRC. Algorithm 2 presents a simplified pseudo-code of the template generation. This module also parametrizes and generates the functions: SD_INIT (that initializes the SD_CARD), SD_TranfersPartial (that transfers the partial bitstreams from the SD-CARD to the DDR3), get_number (that reads the user inserted value), and get_operands (that receives the value of the operands for the case of math calculations e writes in the AXI registers). Specific detail of these function can be obtained by the code description and are hidden here because it does not contain vital information about the primary module purpose.

Algorithm 2: Pseudocode of the template generated for application with PRC

```
input : RPsystem: Struct with the data of the system
1  begin
2      Insert the default libraries;
3      Define the parameters such as addresses and bit file length for the RMs and the blanking
         configurations;
4      Define the PRC addresses for registers of status of each RP and control, triggers and base
         address for each RM;
5      Flushes and disable Data Cache;
6      Initializes SD controller;
7      for i = 1 to number of RPs do
8          if RPsystem(i).Variant = 'on' then
9              for k = 1 to number of RMs do
10                 Transfers partial bitstream related to the RM to DDR;
11             end
12             Transfers blanking bitstream related to the RP to DDR;
13         end
14     end
15     Invalidate and enable Data Cache;
16     Initialize Device Configuration Interface;
17     De-select PCAP as the configuration device as we are going to use the ICAP;
18     Display PRC status for all RPs;
19     for i = 1 to number of RPs do
20         if RPsystem(i).Variant = 'on' then
21             for k = 1 to number of RMs do
22                 Initializes the bitstream address and size registers for the RMs;
23             end
24             Initializes the blaking bitstreams address and size registers;
25         end
26     end
27     Initializes RM trigger ID registers for all RMs of each RP;
28     Initializes RM address and control registers for all RMs of each RP;
29     Reads RM bitstreams address and size registers RMs;
30     Reads RM Trigger and address registers for RPs;
31     for i = 1 to number of RPs do
32         if RPsystem(i).Variant = 'on' then
33             Restart the RPs;
34         end
35     end
       /* Display Menu                                                    */
36     while $Exit != 1$ do
37         do
38             Print the reconfiguration option related to the RMs;
39             Print the blanking reconfiguration option related to the RPs;
40         while Any option was selected;
41     end
42     For each option print the reconfiguration status register;
43 end
```

3.5 KNOWN LIMITATIONS OF THE RTRLIB

RTRLib is currently under development, which means that it works under specific conditions. For some applications, some conditions are not considered, and some functionalities are not implemented yet. Table 3.5 presents a set of some functionalities that were not yet implemented on the RTRLib. In this context, Table 3.5 also shows the topics covered for this work; each of them is depicted in detail within the next Sections.

Table 3.5: Identified limitations of the RTRLib.

Hardware Features	Hardware script generalization	✓
	RPs AXI-Lite interface generator and IP packing	✓
	RPs AXI-Stream interface generator and IP packing	
	RMs IPs automatic synthesis	✓
	Floorplanning	✓
	IO mapping constraints	✓
	Power estimation	
	HDL generation for RMs	
	Resources Estimation	✓
Software Features	Software script generator	✓
	Application Type Selection	✓
	Standalone application generator	✓
	FreeRTOS application generator	✓
	Linux application generator	
PR Features	Decoupling	
	Isolation	
	Relocation	
	Bitstream Compressing	
	Computation of the reconfiguration time	
	Data dependency between RPs	
FT Methodology	Integration with the RTRLib Design Flow	
	SEU Estimation	

4 CASE STUDIES

This Chapter is dedicated to the presentation of some case studies implemented using the RTRLib. The first one is a PR design that implements arithmetic operations using the FPU cores, and the second one is a terrain classification based on Multi-Layer Perceptron. The third case is a system that explores a DPR solution for switching different linear regressors used to control a myokinetic-based prosthetic hand. Finally, it is presented a hypothetical real-time application implemented using the FreeRTOS capabilities of the RTRLib.

4.1 CASE STUDY 1: SIMPLE DPR SYSTEM USING FPUS

To evaluate the benefits of the proposed tool, a dynamically reconfigurable architecture with 2 RPs was developed as an example of the utilization of the RTRLib. The first RP, called *rp_odd*, contains three RMs that implement the arithmetic calculation of addition, division, and multiplication by constant. The second RP, called *rp_even*, contains two RMs that implements the sine and the computation of the exponential function of a given value. All the five operations make use of the floating-point IP-Cores.

For this example, it was selected a non-fault-tolerant system, the strategy of reconfiguration using the PRC, and both RPs connected to the ARM through the AXI-Lite Interconnect. The input data of the RPs are sent by the ARM processor through the AXI-Lite, and, in the opposite direction, the ARM reads the result of the operations through the AXI. The reconfiguration is done through the ICAP by hardware triggers mapped in the switches of the board and controlled by the PRC. Figure 4.1(A) shows the high-level modeling of the system, with the connections between the RPs and the ARM processor.

As previously described, inside each RP, the respectively RMs are inserted. Figure 4.1(B) presents the case of the first RP, which contains 3 RMs. Inside each RM the logic to be implemented must be constructed using the IP-Cores available in the repository. Table 4.1 shows the utilization report of all the reconfigurable modules.

Table 4.1: RMs utilization report for the Zedboard (Zynq 7020) for the case study using FPUs.

Name RM	Function	LUTs		FFs		DSPs	
		Available:	53200	Available:	106400	Available:	220
RM_t1	ADD	334	0,63%	275	0,26%	0	0%
RM_t2	EXP	1785	3,36%	596	0,56%	1	0.45%
RM_t3	DIV	278	0,52%	320	0,30%	1	0.45%
RM_t4	SINE	1419	2,67%	557	0,52%	1	0.45%
RM_t5	MUL	135	0,25%	255	0,24%	1	0.45%

Figure 4.2 shows the block design created through the script generated using RTRLib, which has the IPs related to the RPs, the ARM-Core processor IP, PRC, and the AXI Interconnect.

Finally, all the rest of the design flow is executed, and all the configurations and its respective partial

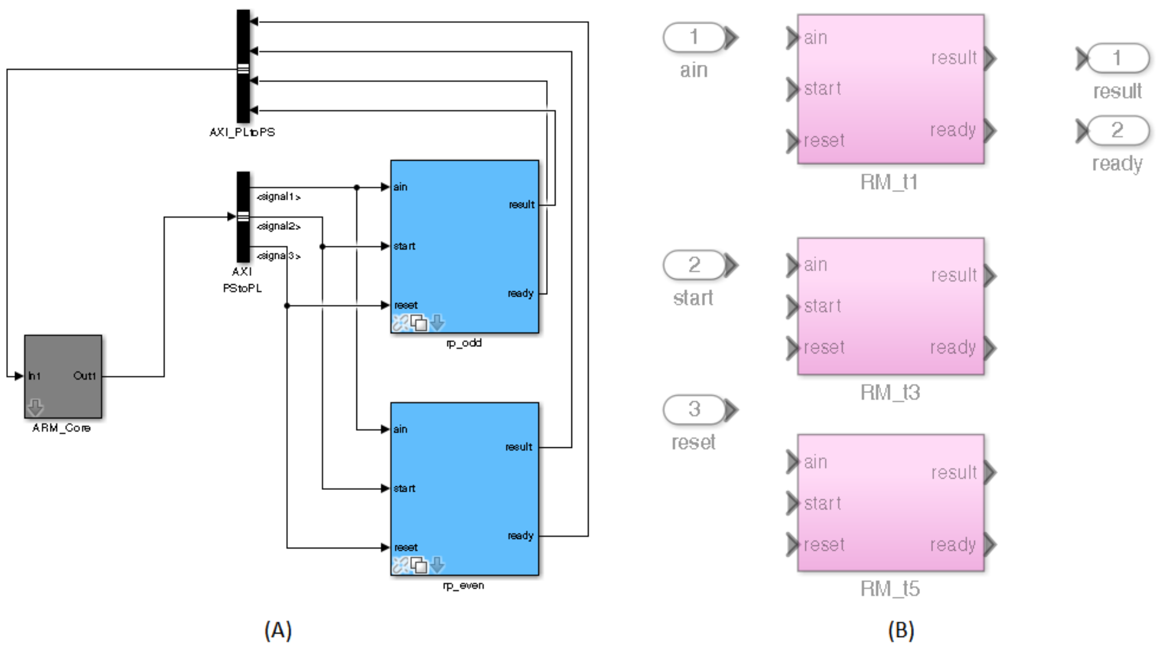


Figure 4.1: (A) Detail of the system-level model using the RTRLib, showing the RPs, AXI, and the connections. The data input of the RPs is sent to the ARM processor through the UART port (not depicted in the figure). (B) Reconfigurable Modules of the example using IPs.

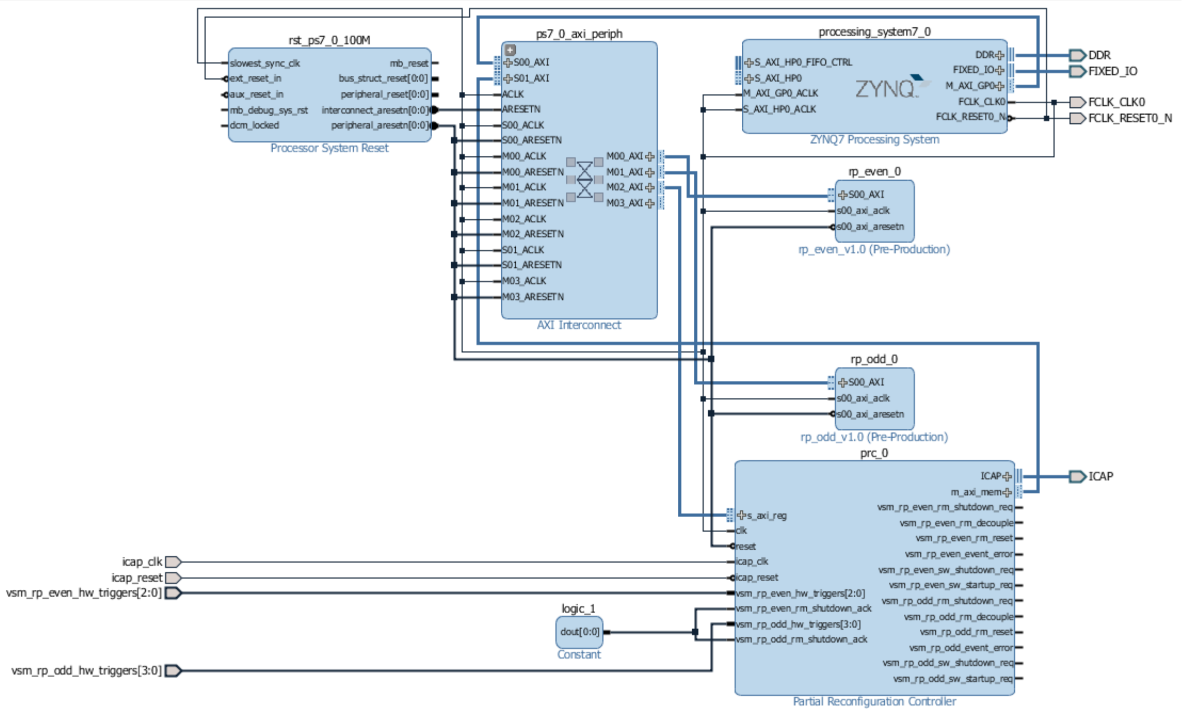


Figure 4.2: Block Based Architecture with two RPs, PRC, AXI switch and Zynq7 Processing System.

bitstreams were effectively generated. The size in kB of each partial bitstream was obtained and using the model reported by (68), which also uses a Zedboard and states that the relationship between the reconfiguration time and the size of the partial bitstream is essentially linear, the reconfiguration time was estimated. Those results are presented in Table 4.2.

Table 4.2: Comparison between the size of the partial bitstreams and the reconfiguration time.

Partial bitstream	Size (KB)	Estimated reconfiguration time (ms)
Top level with blanking configuration	3951	30,422
RP odd	223	1,717
RP even	484	3,727

Figure 4.3 shows the circuit layout containing the floating-point adder in the first RP and the floating-point sine estimator in the second RP.

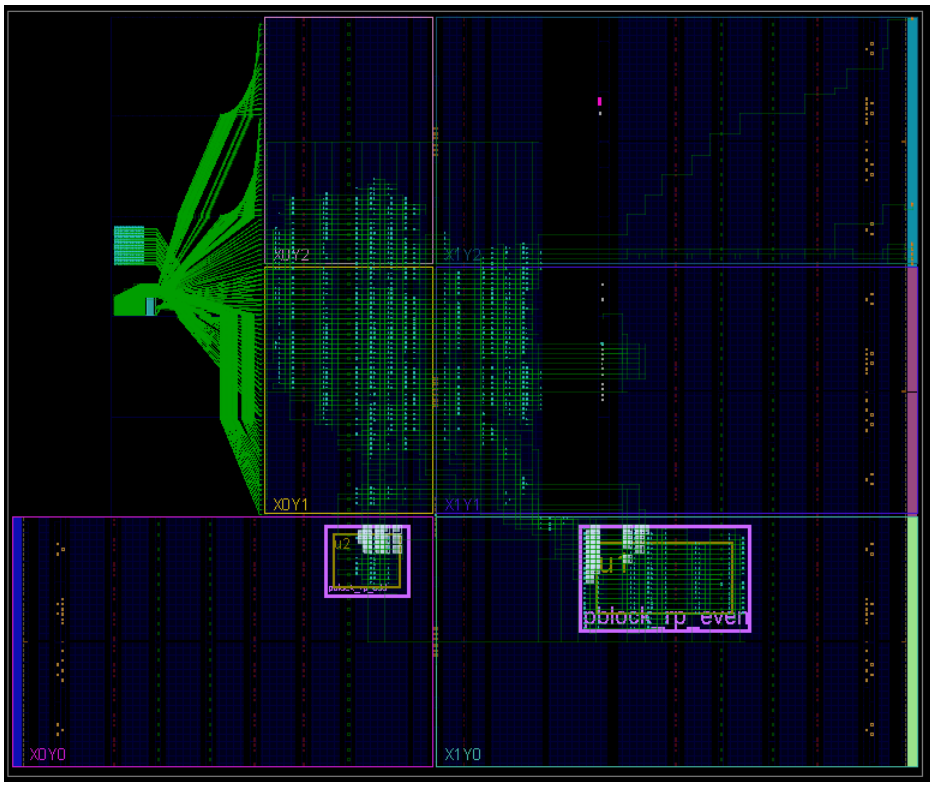


Figure 4.3: Final implementation of the reconfigurable architecture including the static system and the reconfigurable partitions, with the reconfigurable modules that computes the adder and the sine function, respectively, implemented on a Zynq-7020 device.

4.2 CASE STUDY 2: TERRAIN CLASSIFICATION USING MLP ANN

In mobile robotics, sometimes, the robot must adapt to the environmental conditions dynamically. One of the environmental aspects that can impact in robot performance is the type of terrain the robot is moving. In this context, the problem consists of the classification of terrains between four types: synthetic grass, grass, asphalt, and smooth floor. The proposed solution for the classification problem is the usage of a multi-layer perceptron (MLP) artificial neural network (ANN) in which the robot has common embedded system requirements such as power consumption, size, and performance.

Besides this, robots are also classified as embedded systems with hard requirements such as power consumption, size, and performance. To overcome this question, it is proposed a PR approach, depicted as follows:

- The system has one RP and three RMs. Each RM consists of one MLP; that differs by the number of neurons in the hidden layer, respectively, 3, 5, and 7. Therefore, the system has four different possible configurations, including the blanking configuration.
- The input layer consists of the three-axis accelerometer measurements (ax, ay, az), which are sent through the UART interface.
- The input and output layers are fixed since they depend on the features and the number of the terrain classes, respectively, 3 and 4.
- In relation to the dataset, 70% were used for training, 15% for tests, and 15% for validation.
- The output classification is sent by UART.

4.2.1 Experimental Conditions

For acquiring the training data, a simple mobile robot without damping and velocity control was used. It was controlled by infra-red equipped with two Arduinos, one MPU5060 sensor (that is composed by a three-axis accelerometer and a three-axis gyroscope), and one SD-CARD module.

It was realized 32 experiments with each one of the four types of terrains. In all the experiments, after the calibration of the sensor, the mobile robot was put to move in a straight line forward for 3 seconds. However, to improve the generalization capability of the network, the experiments were not realized in the same part of the terrains, but at different points. Figure 4.4 shows one image of the mobile robot in each of the four terrains.

The dataset was refined using Matlab, in which the collected data when the robot was stopped were excluded. Then, 100 samples of each experiment were selected, resulting in 3200 samples for each terrain. As the data represent temporal series, the absolute value of the sample does not represent significant information by itself. In this context, the dataset was constructed, guaranteeing the 100 samples of each experiment stayed together. Also, using Matlab, the 3 types of configurations were trained, as it is shown in Figure 4.5. As classification results, it was obtained 54.2%, 54.7% and 56.8% for the networks MLP_334, MLP_354 and MLP_374 respectively.

Figure 4.6(A) shows the high-level modeling of the system, with the connections between the RPs and the ARM processor, while Figure 4.6(B) presents 3 RMs inside the RP. As mentioned before, the RMs differ from each other by the number of neurons in the hidden layer. Figure 4.7 shows the implementation of the RM containing an ANN with 3 neurons in the hidden layer. The ANNs with 5 and 7 neurons were similarly implemented.

The description of the neuron was performed based on the IP-Cores available in the repository. Figure 4.8 presents the neuron with 3 inputs as an example.

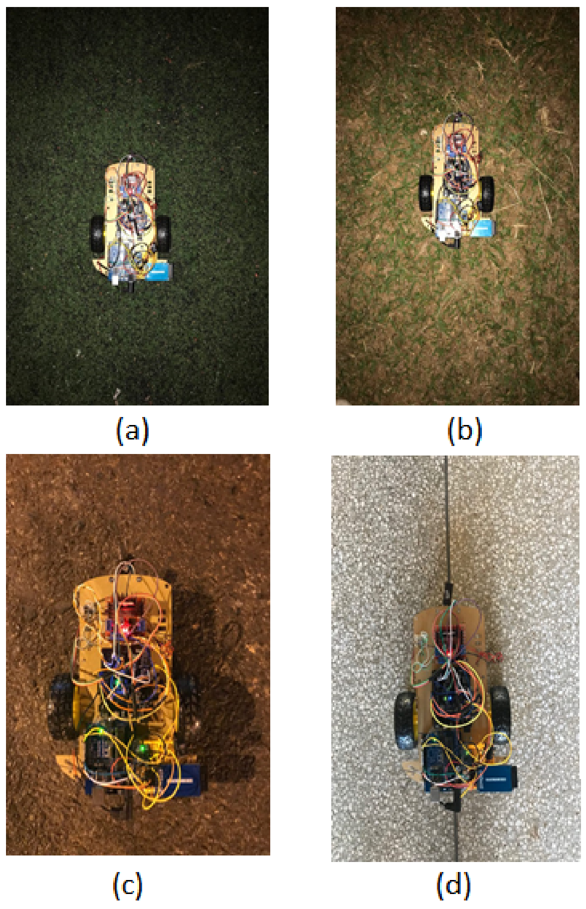


Figure 4.4: Terrain Types: (a) Synthetic Grass;(b) Grass;(c) Asphalt;(d) Smooth floor.

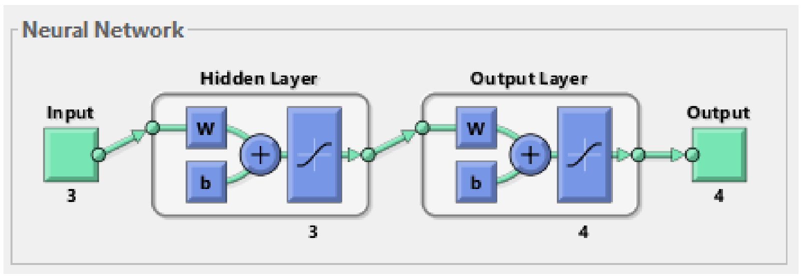


Figure 4.5: MLP334 configuration that contains 3 neurons in the hidden layer.

Since RTRLib does not generate the logic of the RMs automatically yet, the 3 networks were implemented manually in VHDL. Table 4.3 shows the utilization report of all the reconfigurable modules.

Table 4.3: RMs utilization report for the Zedboard (Zynq 7020) for the case study of terrain classification.

Name RM	LUTs		FFs		DSPs	
	Available:	53200	Available:	106400	Available:	220
RM_MLP_334	1113	2,09%	296	0,28%	3	1%
RM_MLP_354	7838	14,73%	1923	1,81%	22	10%
RM_MLP_374	10535	19,80%	2507	2,36%	29	13%

Figure 4.9 shows the block design created through the script generated using RTRLib, which has the

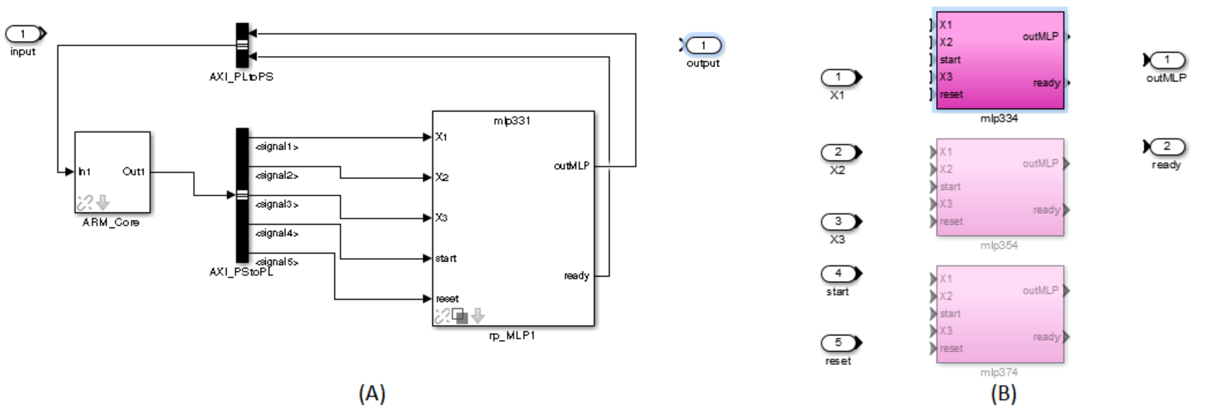


Figure 4.6: (A)Detail of the system level model of the MLP-based ANN using RTRLib for terrain classification. The data input of the RP is sent to the ARM processor through the UART port (not depicted in the figure). (B) Reconfigurable Modules of each network architecture.

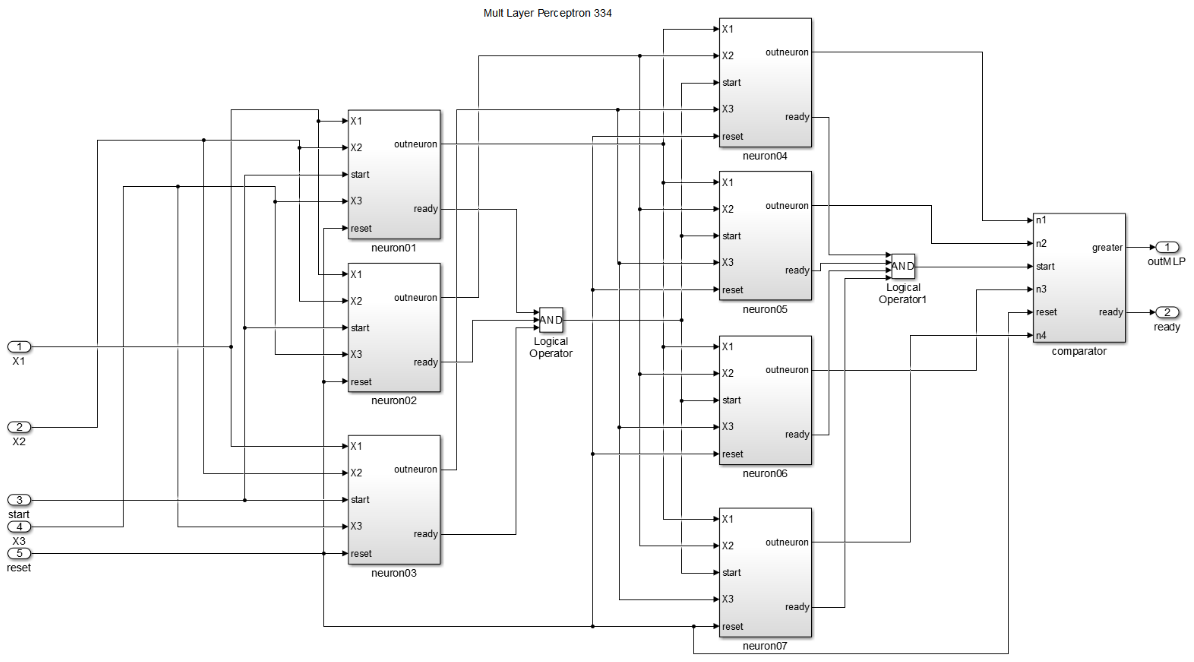


Figure 4.7: RM MLP334 that contains 3 neurons in the hidden layer.

IP related to the *rp_MLP*, the ARM-Core processor IP, PRC, and the AXI Interconnect.

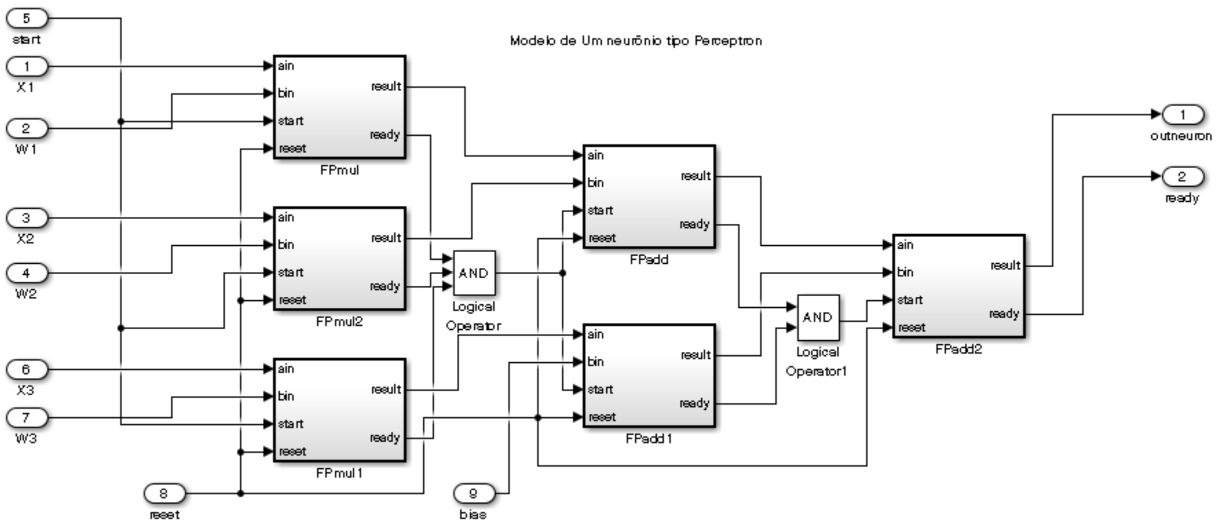


Figure 4.8: Implementation of a three-input neuron in RTRLib using the floating-point IP-Cores.

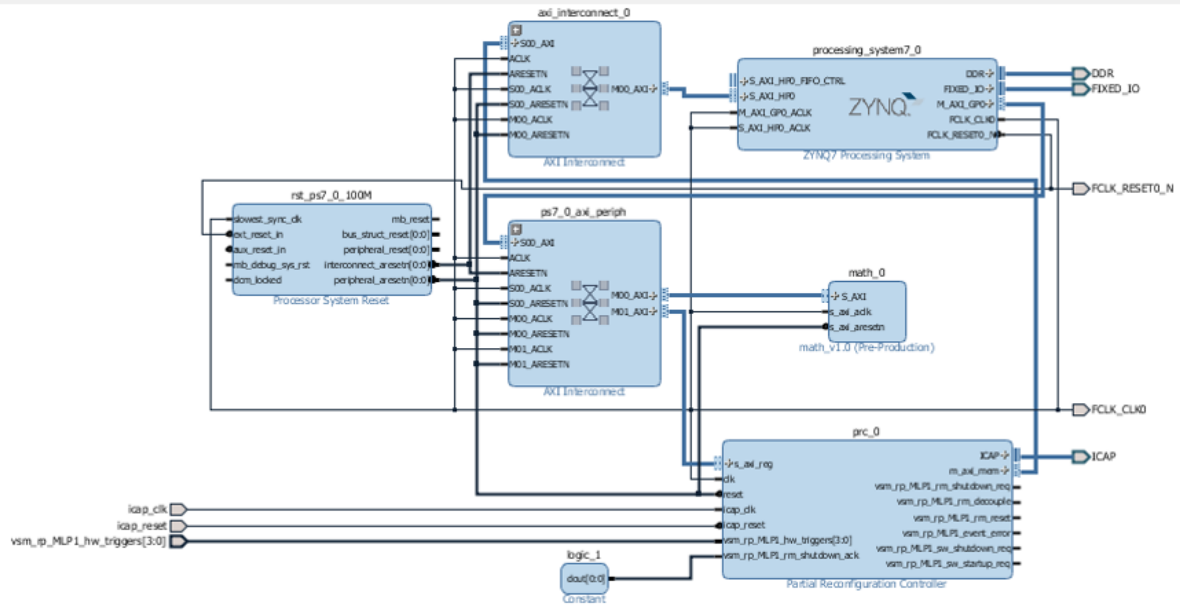


Figure 4.9: Block Based Architecture with the MLP RP, PRC, AXI switch and Zynq7 Processing System.

4.3 CASE STUDY 3: LINEAR REGRESSORS FOR MYOKINETIC PROSTHETIC HAND CONTROL

This application was developed in collaboration with Sergio Pertuz (PPMEC/UnB Ph.D. student), Prof. Helon Ayala (PUCRJ), and the BioRobotics Institute of the Scuola Superiore Sant'Anna (Pisa, Italy). The main goal of the application is to track the muscle contractions with implanted permanent magnets by means of magnetic field sensors (myokinetic interface). For that, the Italian group has developed a prototype which exploits six 3-axis sensors to localize four magnets implanted in a forearm mockup, for the control of a dexterous hand prosthesis (69). The Brazilian group (Prof. Helon Ayala and LEIA group) has developed linear regressors and Radial Basis Functions (RBF) ANNs to predict the position of the magnets and then classify the intended movement.

In this context, a DPR solution can be used for different purposes: (a) to adapt the myokinetic control interface according to the timing constraints of the acquisition board and the control unit which must attend the time response of the prosthetic hand; (b) to reduce the cost and power consumption of the FPGA-based control system by switching between different control strategies for each degree of freedom of the prosthetic hand; and (c) to adapt the myokinetic control interface by switching the appropriate hardware architecture which tracks the appropriate muscles, depending of the desired degree of freedom to be controlled.

Figure 4.10 depicts the myokinetic control interface. It is composed of an acquisition unit and a computation unit. The acquisition unit is a grid of 32 three-axis magnetic field sensors (MAG3110, NXP Semiconductors NV), placed in a 4x8 matrix. The grid of sensors is connected to a 16-bit PIC microcontroller, which sequentially samples the sensor readings every 21.0 ms and transmits them to the control unit at a fixed period of 29.1 ms over a UART RS-485 communication bus. The control unit processes the 96 magnetic field data (3-axes x 32 sensors) and continuously retrieves the poses of the magnets (see Figure 4.11). Taking into account the timing restrictions of the acquisition and communication processes as well as the timing specifications of the prosthetic hand, the control algorithm executed by the control unit must be selected to optimize the overall execution time and power consumption of the system.

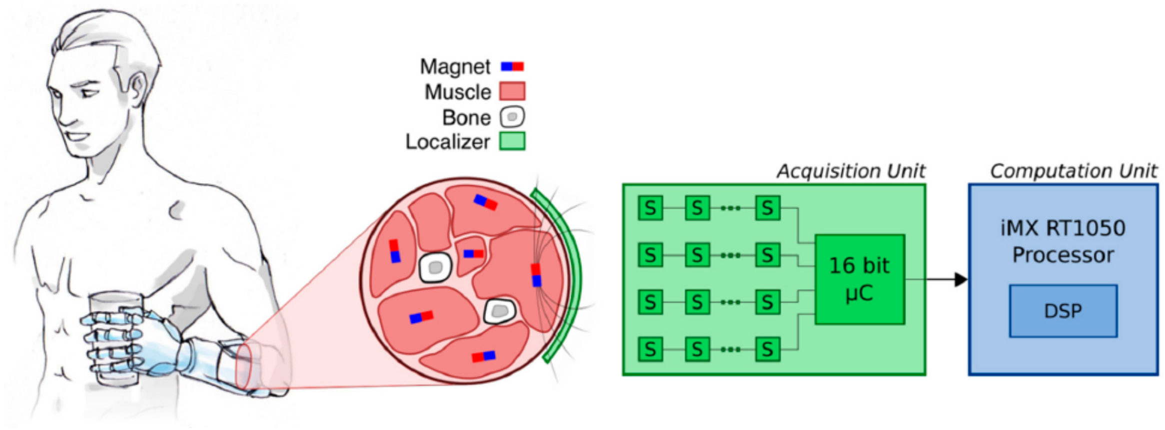


Figure 4.10: Overview of the embedded myokinetic control interface for a prosthetic hand (19).

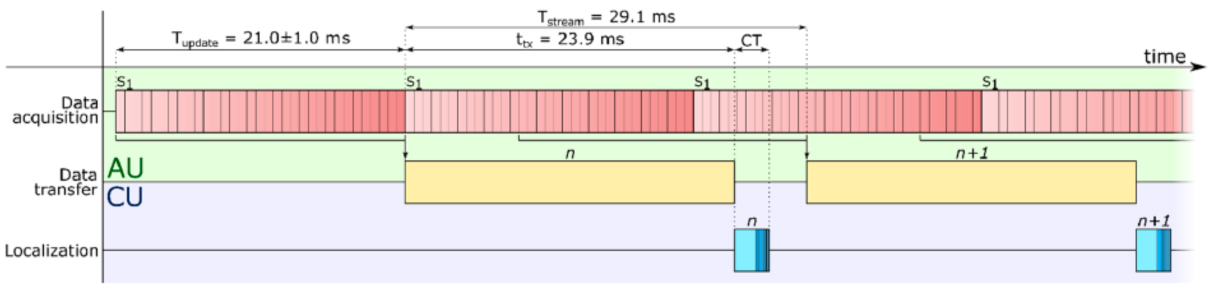


Figure 4.11: Temporal diagram of the tasks involved in the embedded localizer. (19).

In this case study, different linear regressors will be loaded into one RP, allowing the execution time of the position estimation process to be adjusted. Two different linear regressors were manually implemented in VHDL using the available IP-Cores, with 4 and 8 operators, respectively. For this example, it was selected the strategy of reconfiguration using the PRC, and the RP is connected to the ARM through the AXI-Lite Interconnect. The input data of the RP is sent by the ARM processor through the AXI-Lite, and, in the opposite direction, the ARM reads the result of the operations through the AXI. The system is prepared to realize reconfiguration by hardware and software triggers, and PRC controls the ICAP. Figure 4.12(A) shows the high-level modeling of the system, with the connections between the RP and the ARM processor.

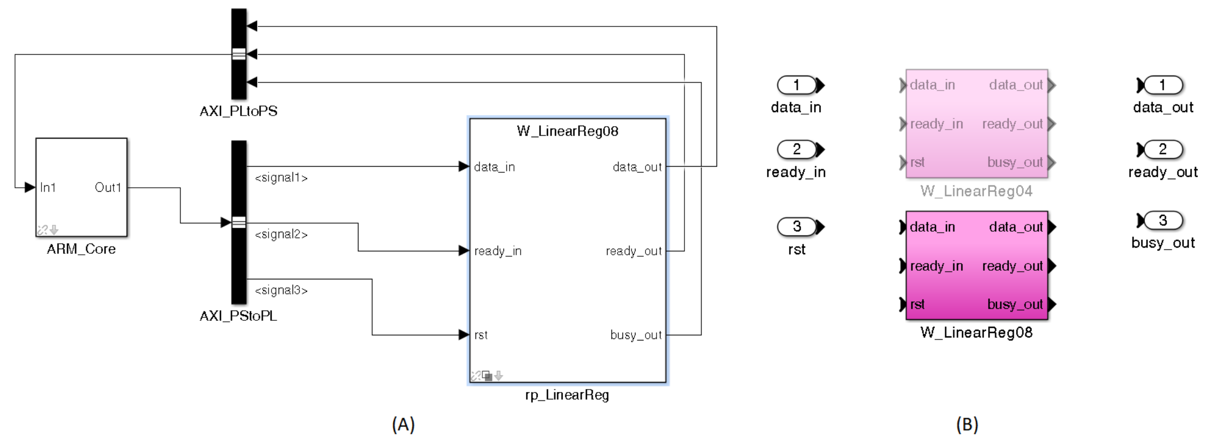


Figure 4.12: (A) Detail of the system-level model using the RTRLib, showing the RP, AXI, and the connections. The data input of the RP is sent to the ARM processor through the UART port (not depicted in the figure). (B) Reconfigurable Modules of 4 and 8 operators.

Figure 4.13 shows the block design created through the script generated using RTRLib, which has the IPs related to the RPs, the ARM-Core processor IP, PRC, and the AXI Interconnect. Figure 4.14 shows the circuit layout containing the module with 4 operators in the RP.

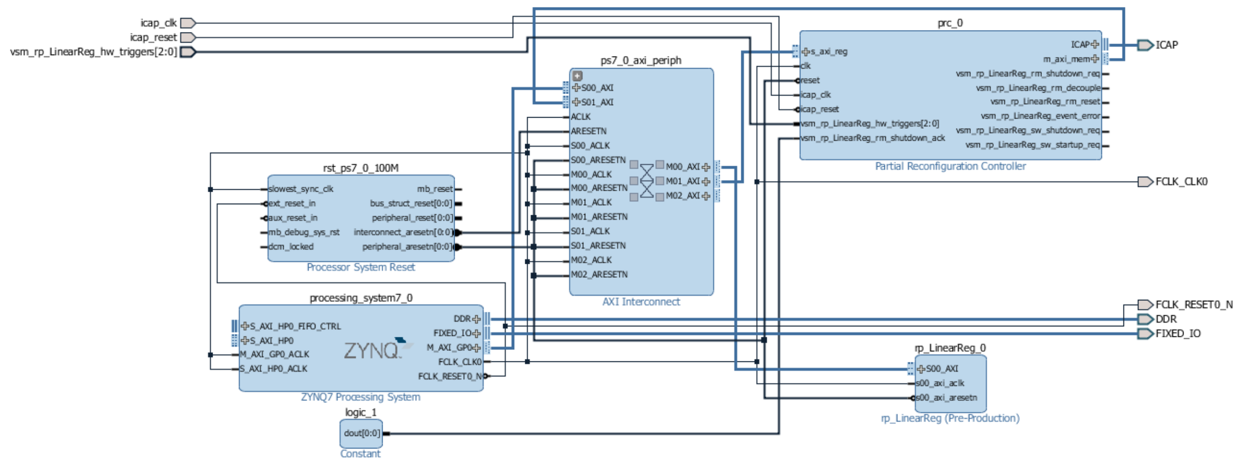


Figure 4.13: Block Based Architecture with the RP, PRC, AXI switch and Zynq7 Processing System.

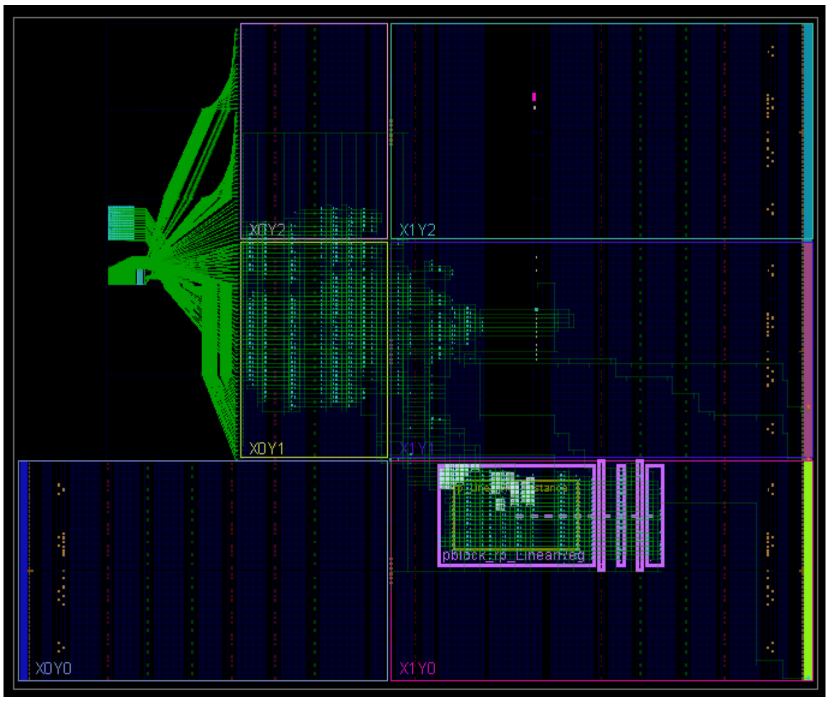


Figure 4.14: Final implementation of the reconfigurable architecture including the static system and the reconfigurable partitions, with the reconfigurable module with a linear regressor with 4 operators. Implemented on a Zynq-7020 device.

4.4 CASE STUDY 4: REAL TIME APPLICATION

This case study refers to a hypothetical application based on a tutorial (21) of the official FreeRTOS webpage. It is essential to highlight that, although it is a hypothetical case study, the same approach applied here can be used to solve several problems using real-time requirements.

The application consists of controlling a plant connected on a fieldbus network (it could be a motor, heater, and others). The control terminal also has two LEDs indicators, one LCD display, and a matrix keypad that is scanned using general-purpose IOs. Besides this, the system also reads the data from two sensors connected to the same fieldbus network, and maintain both local and remote user interfaces. The former interface is a configuration utility that runs on a PDA connected by RS232. The latter consists of one embedded webserver to which a remote monitoring computer can attach. Figure 4.15 shows a diagram of the system.

In this case study, the sequencing and timing constraints are the focus of interest. Therefore, specifically, functional requirements are hidden. Table 4.4 shows the top-level requirements of each function of the system.

In this context, each function of the system was implemented according to its timing restrictions. Table 4.5 presents the timing categories of each function of the system.

To demonstrate possible ways in which the FreeRTOS real-time kernel can be used and how the RTR-Lib approach can simplify the development of the real-time applications, it is presented here a traditional solution that uses a *fully preemptive approach*.

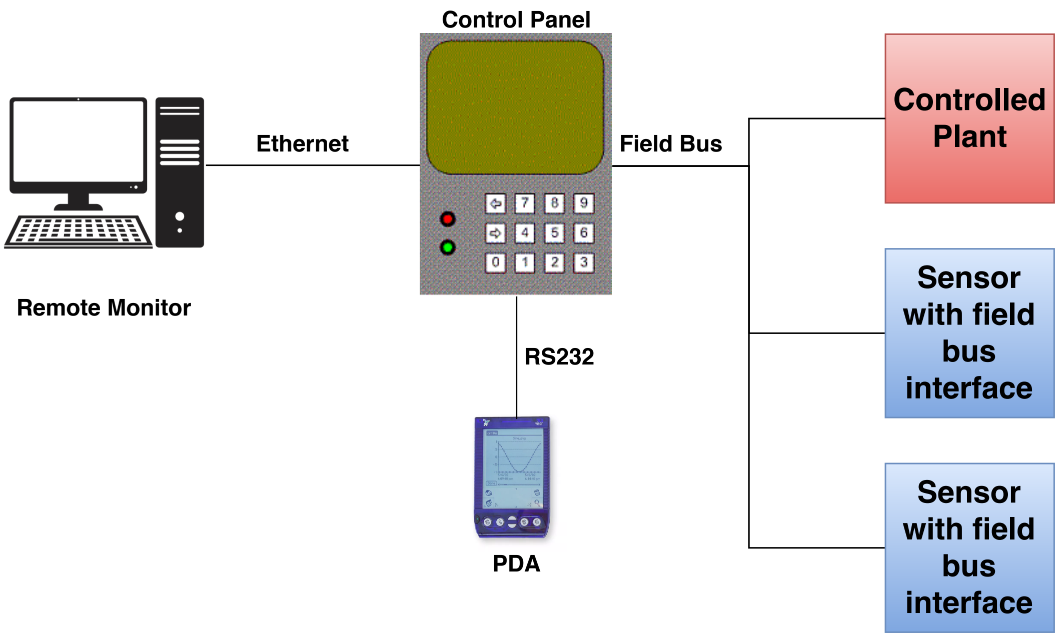


Figure 4.15: System diagram showing the components and the type of connections.

Table 4.4: Top-level software requirements (21).

Function	Description	Requirements description
Plant control	Each control cycle shall perform the following sequence: (1) Transmit a frame on the fieldbus to request data from the networked sensors. (2) Wait to receive data from both sensors. (3) Execute the control algorithm. (4) Transmit a command to the plant.	The control function must strictly transmit a request every 10ms, and the resultant command shall be transmitted within 5ms.
Local operator interface (keypad and LCD)	This interfaces can be used by the operator to select, view, and modify system data.	The keypad must be scanned at least every 15ms, while the LDC shall update within 50ms of a key being pressed.
LEDs output	The LED is used to indicate the system status. The green LED indicates the system is running as expected, while the red one indicates a fault condition.	The flash rate must be maintaned within 50ms.
RS232 PDA interface	The PDA RS232 interface shall be capable of viewing and accessing the same data as the local operator.	The same timing constraints of the local operator interface apply here.
TCP/IP Ethernet interface	The remote interface is connected through a TCP/IP interface	The web server shall service HTTP requests within one second.

For each functionality of the system, a separate task was created, and the traditional real-time preemptive multitasking solution is used. Therefore, as it can be viewed in Table 4.6, this solution is based on the utilization of 5 tasks besides the Idle task. For each type of communication protocol, it is used one queue on which events can be received, resulting in 3 queues.

The high-level model was created instantiating the *FreeRTOS* block inside the *ARM_block*, and inside

Table 4.5: Timing requirements categories (21).

Category	Component	Description
Strict timing	The plant control	The control function has very restricted timing as it has execute every 10ms
Flexible timing	LED output	There is a large timing band within which the LED can function
Deadline only timing	Human interfaces: keypad, RS232, and TCP/IP Ethernet communications	These functions have different type of timing requirement as only a maximum limit is specified.

it, the blocks related to the functions described below were inserted, as it is possible to see in Figure 4.16.

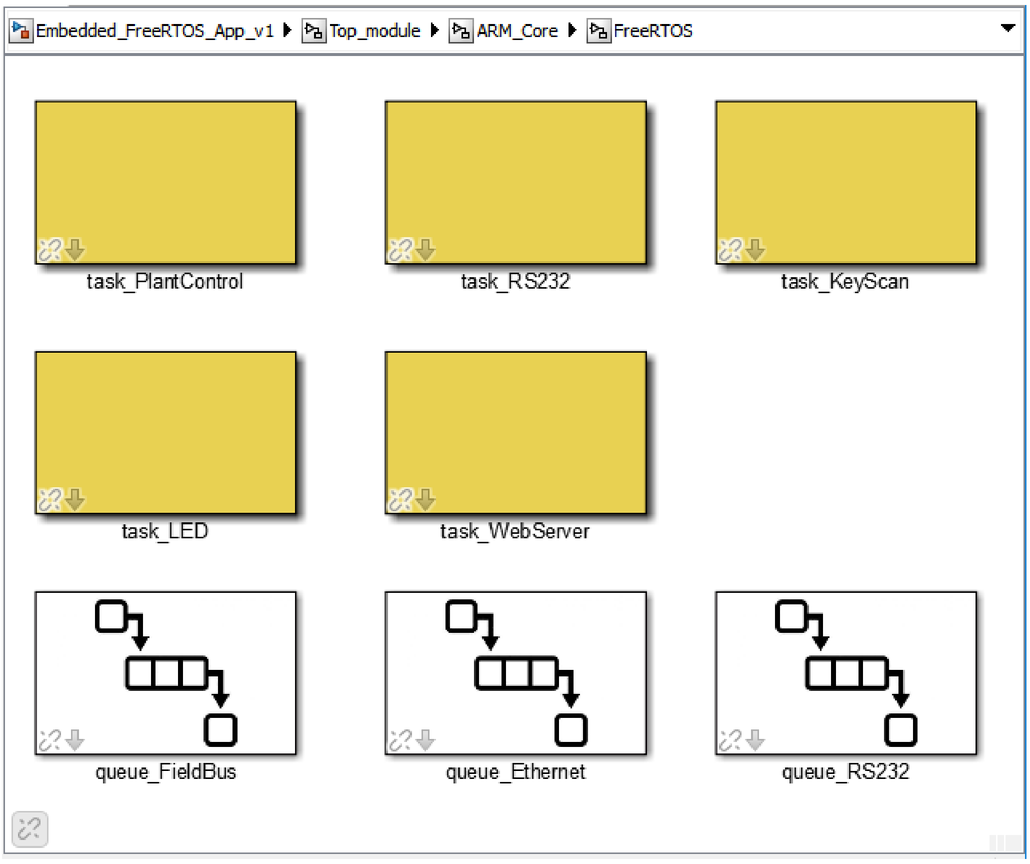
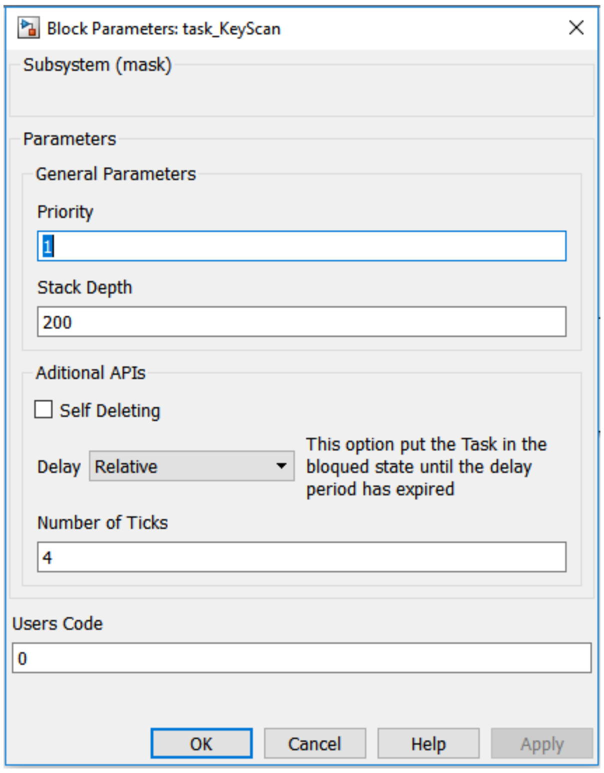


Figure 4.16: Function blocks of the fully preemptive approach inserted into the FreeRTOS block.

Figure 4.17 (A) shows the configuration parameters of the *task_KeyScan*. The *priority* was set based on the timing requirements of the system, as can be seen in Table 4.6. The *stack depth* was initially defined as the minimum stack depth of the system. Since the task has a fixed execution period, the delay was set as the absolute type. The keypad must be scanned at least every 15ms, so the delay parameter receives this value. The parameters configuration of all other tasks was set in a similar way.

The only configuration parameters of the queue blocks are the number of items of the queue and the size of each data. Figure 4.17 (B) shows the configuration parameters of the *queue_Ethernet*, considering a hypothetical case of a packet of 1500 bytes and the size of the queue equal to the size of the packet. The parameter configuration of all other queues was set similarly.



Block Parameters: task_KeyScan

Subsystem (mask)

Parameters

General Parameters

Priority

1

Stack Depth

200

Aditional APIs

☐ Self Deleting

Delay: Relative

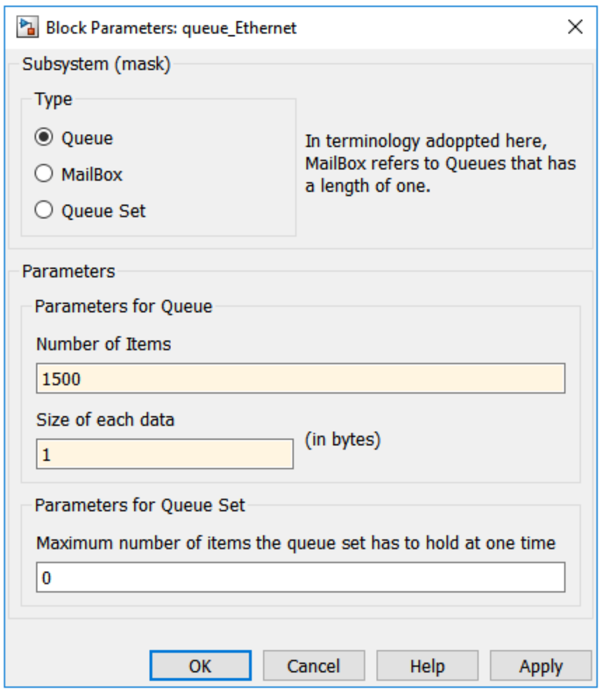
This option put the Task in the bloqued state until the delay period has expired

Number of Ticks

4

Users Code

0



Block Parameters: queue_Ethernet

Subsystem (mask)

Type

● Queue

○ MailBox

○ Queue Set

In terminology adoppted here, MailBox refers to Queues that has a length of one.

Parameters

Parameters for Queue

Number of Items

1500

Size of each data

1

(in bytes)

Parameters for Queue Set

Maximum number of items the queue set has to hold at one time

0

Figure 4.17: (A) Screen of configuration of the tasks parameters. In the example is shown the *task_KeyScan* configuration (B) Screen of configuration of the queues parameters. In the example is shown the *task_Ethernet* configuration.

Table 4.6: Functions tasks and priorities of the fully preemptive solution.

Priority	Tasks
2	task_PlantControl
1	task_RS232, task_KeyScan
0	task_Idle, task_LED, task_WebServer

After the parametrization of the blocks, the RTRLib must be executed to generate the FreeRTOSConfig.h and the template for the application. In the sake of simplicity, it is shown in 4.1 just the resulted lines in the function main related to the creation of the *task_KeyScan*, the *queue_Ethernet*, and the initialization of the scheduler, but the original template generated contains all the functions defined in 4.16.

```
int main (void){


        BaseType_t xTaskCreate(  task_KeyScan, //POINTER
                                 ( const char * ) "task_KeyScan",//NAME for Debbuging
                                 200,//STACK DEPTH
                                 NULL,
                                 1,// Priority
                                 TaskHandle_task_KeyScan );//HANDLE TASK

        queue_Ethernet = xQueueCreate(1500,1);
        //Checking if the Queue was created
        configASSERT( queue_Ethernet );

        vTaskScheduler();
         for(;;)
}
```

Listing 4.1: Simplified version of the main function generated by RTRLib

A comparison between the generated and the original function of the *task_KeyScan* is presented as follows. As previously described, RTRLib does not concern the logic generation, it focuses on the template application generation. Therefore, it is possible to see that the system successfully generates the function structure according to the parameters and are left some spaces that allow the user to insert the desired logic.

```
void KeyScanTask( void *pvParmeters )
{

char Key;
TickType_t xLastWakeTime;

    xLastWakeTime = xTaskGetTickCount();
    for( ;; ){
        vTaskDelayUntil( &xLastWakeTime, DELAY_PERIOD );

        if( KeyPressed( &Key ) ){
            UpdateDisplay( Key );
        }
    }
}
```

Listing 4.2: Original function of the *task_KeyScan*

```
static void task_KeyScan( *pvParameters )
{

        //The User Variables goes here

        const TickType_t xDelay_4ms = pdMS_TO_TICKS(4);
        TickType_t xLastWakeTime;
        xLastWakeTime = xTaskGetTickCount();
        for(;;){
                vTaskDelayUntil(&xLastWakeTime,xDelay_4ms);

                //The User Logic goes here

        }
}
```

Listing 4.3: Function of the *task_KeyScan* automatically generated by RTRLib

5 CONCLUSIONS AND FUTURE WORK

5.0.1 Final remarks

This work introduced the RTRLib, a new platform-based high-level tool that was projected to automate the partial reconfiguration design flow steps. The main contributions of the RTRLib are: (a) DPR system design is more accessible to non-expert designers; (b) The RTRLib can allow for early estimation of the hardware resources and size of each reconfigurable module, thus enabling an analysis of the effect of the latency and reconfiguration time of each RP over the system; (c) The proposed tool also provides a design methodology for fault-tolerant systems based on the hardware redundancy approach. All these features allow the prototyping time to reduce and enable fast creation of high-performance systems; (d) Automatic generation of software application using PRC and a library to construct systems with real-time requirements.

RTRLib was presented in detail, from its proposed methodology to its front-end framework based on blocks that allow the construction of the DPR system as a process network in Simulink, and all the modules that compose the RTRLib back-end built to generate the TCL hardware and software scripts.

During this work, some functionalities of the tool were improved, while others were inserted. In relation to the hardware features: the hardware script generalization, the AXI-Lite interface creation for the RPs and IP packing, the automatic synthesis of the logic of the RMs with the HDLs provided by the user, IO mapping by the automatic .xdc creation by the provided constraints, and floorplanning by a GUI. In the case of the software features: a software script creation, a generator of a standalone template application that uses PRC, and generation of a FreeRTOS library application.

As a proof of concept of the RTRLib methodology, it was presented four different case studies: (a) A PR design that implements arithmetic operations using the available FPU IP-Cores; (b) A terrain classification based on Multi-Layer Perceptron;(c) A system that explores a DPR solution for switching different linear regressors used to control a myokinetic-based prosthetic hand; (d) A hypothetical real-time application implemented using the FreeRTOS capabilities of the RTRLib. The case studies were implemented to demonstrate the functionalities of the tool. It was observed the designs, as well as the software applications, were successfully generated.

It was observed that under specific conditions, RTRLib is able, from the high-level modeling, to automatically generate an entire design flow from the hardware project creation to the boot image elaboration. Since the system implementation using VIVADO and SDK is done through scripts, an alteration in the specification can be fast implemented using the framework. However, there are several conditions that are not considering yet in RTRLib that must be inserted.

5.0.2 Suggestions for future works

The first step suggested for future works is the analysis of the remaining limitations of the RTRLib, to determine which function has a higher priority to be implemented. However, it is already possible to see that the implementation of the PR features is really needed to provide a complete PR design flow implementation.

Since RTRLib is ideally projected for critical systems, the implementation of the isolation technique is essential to ensure the construction of reliable systems. An interesting reference for the implementation of isolation is the IDF (13) that provides an entire design flow elaborated to guide the construction of isolated systems based on the concept of trusted routes, as previously described in this work.

Another PR feature that deserves special attention is the decoupling, to ensure the development of systems that the static logic ignores the outputs from an RM to avoid unexpected system behavior in function of the presence of not valid output data of the RM during reconfiguration, and also to allow a security implementation of systems with data dependency between RPs.

In the context of the floorplanning, it is crucial to investigate strategies to enhance the RTRLib tool floorplanning algorithm to enable slot and grid styles, since RTRLib allows only the island style currently. Also, another possibility is to implement automatic floorplanning that does not use the GUI. When the entire flooplanning strategy is ready, the relocation technique can be implemented, taking into account all the requirements of a relocatable system.

In the context of the software design flow, a suggested approach is to implement a script that allows the development of designs using PetaLinux, by automating the process of the PetaLinux image creation in the function of the co-processors connected through the AXI-Stream and also automating the device tree elaboration. Besides that, it would be interesting to construct a software template for the case of the PCAP reconfiguration strategy.

Also relating to software applications, it would be interesting to implement a case study with the FreeRTOS library to construct an application that allows reconfiguration through software triggers. A co-design that attends timing and area hard restrictions, demonstrating the PR benefits in terms reduction of area and also the capability in terms of performance of such design. In this case, a reconfiguration directly through the PCAP to avoid additional resource utilization.

In relation to the fault-tolerant methodology, it is necessary to integrate the PSO and MODE algorithms into the RTRLib and study strategies to obtain an early SEU estimation. Since partial reconfiguration is a technique used to recover the system from a fault, it would be good to analyze the integration of the redundancy from two different views. The former is to use the bio-inspired algorithm to find the optimal redundancy level for each process and maximize its reliability; the latter is to replicate the RPs and reconfigure it in the case of fault detection.

Bibliography

1 HENNESSY, J. L.; PATTERSON, D. A. A new golden age for computer architecture. *Commun. ACM*, v. 62, n. 2, p. 48–60, 2019.

2 CALAZANS, N. L. V. *Projeto Lógico Automatizado de Sistemas Digitais Seqüenciais*. [S.l.]: DCC/IME, 1998.

3 BROWN, S. D. *Fundamentals of digital logic with Verilog design*. [S.l.]: Tata McGraw-Hill Education, 2007.

4 GUIDE, U. Series FPGAs configurable logic block. *Xilinx, San Jose, CA*, v. 1, 7.

5 XILINX. *Introduction to FPGA Design with Vivado High-Level Synthesis - UG998*. 2019. <https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf>. Accessed: 2019-03-18.

6 XILINX. *Vivado Design Suite User Guide Design Flows Overview - UG892*. 2018. <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug892-vivado-design-flows-overview.pdf>. Accessed: 2018-10-08.

7 FOSTER, H. D. Technical report, *TRENDS IN FUNCTIONAL VERIFICATION: A 2016 INDUSTRY STUDY*. 2016.

8 HAUCK, S.; DEHON, A. *Reconfigurable computing: the theory and practice of FPGA-based computation*. [S.l.]: Elsevier, 2010. v. 1.

9 GUIDE, U. Zynq 7000 technical reference manual ug585. *Xilinx, San Jose, CA*, v. 1.12.1, 2018.

10 CROCKETT, L. H.; ELLIOT, R. A.; ENDERWITZ, M. A.; STEWART, R. W. The zynq book. *Strathclyde Academic Media*, 2014.

11 GUIDE, U. *Introduction to Partial Reconfiguration Methodology*. 2015.

12 XILINX. *Partial Reconfiguration User Guide - UG702*. 2013. <https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug702.pdf>. Accessed: 2018-02-21.

13 XILINX. *Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs - XAPP1222*. 2016. <https://www.xilinx.com/support/documentation/application_notes/xapp1222-idf-for-7s-or-zynq-vivado.pdf>. Accessed: 2019-06-01.

14 XILINX. *Isolation Design Flow*. 2019. <https://www.xilinx.com/applications/isolation-design-flow.html>. Accessed: 2019-06-01.

15 XILINX. *Vivado Design Suite User Guide - Partial Reconfiguration - UG909*. 2017. <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug909-vivado-partial-reconfiguration.pdf>. Accessed: 2019-01-21.

16 RETTKOWSKI, J.; FRIESEN, K.; GÖHRINGER, D. Repabit: Automated generation of relocatable partial bitstreams for Xilinx Zynq fpgas. In: IEEE. *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. [S.l.], 2016. p. 1–8.

17 KOCH, D.; TORRESEN, J.; BECKHOFF, C.; ZIENER, D.; DENNL, C.; BREUER, V.; TEICH, J.; FEILEN, M.; STECHELE, W. Partial reconfiguration on FPGAs in practice—tools and applications. In: IEEE. *ARCS Workshops (ARCS), 2012*. [S.l.], 2012. p. 1–12.

18 XILINX. *User guide UG116, Device Reliability Report, Second Half 2018*. [S.l.], 2015.

19 CLEMENTE, F.; IANNICIELLO, V.; GHERARDINI, M.; CIPRIANI, C. Development of an embedded myokinetic prosthetic hand controller. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 14, p. 3137, 2019.

20 XILINX. *AXI Reference Guide - UG761*. 2011. <https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf>. Accessed: 2018-11-06.

21 REAL Time Application Design Tutorial. <https://www.freertos.org/tutorial/index.html>. Accessed: 2019-10-10.

22 SASS, R.; SCHMIDT, A. G. *Embedded systems design with platform FPGAs: principles and practices*. [S.l.]: Morgan Kaufmann, 2010.

23 WOLF, M. *High-performance embedded computing: applications in cyber-physical systems and mobile computing*. [S.l.]: Newnes, 2014.

24 SANTAMBROGIO, M. D.; PILATO, C.; PNEVMATIKATOS, D.; PAPADIMITRIOU, K.; STROOBANDT, D.; SCIUTO, D. The faster vision for designing dynamically reconfigurable systems. In: IEEE. *IC Design & Technology (ICICDT), 2013 International Conference on*. [S.l.], 2013. p. 5–8.

25 ROUSSEAU, B.; MANET, P.; GALERIN, D.; MERKENBREACK, D.; LEGAT, J.-D.; DEDEKEN, F.; GABRIEL, Y. Enabling certification for dynamic partial reconfiguration using a minimal flow. In: EDA CONSORTIUM. *Proceedings of the conference on Design, automation and test in Europe*. [S.l.], 2007. p. 983–988.

26 XILINX. *Two competitive FPGA methodologies for run-time reconfiguration*. 2008. <http://signal-processing.mil-embedded.com/articles/two-fpga-methodologies-run-time-reconfiguration/>. Accessed: 2017-11-16.

27 ZHANG, X.; NG, K. W. A review of high-level synthesis for dynamically reconfigurable FPGAs. *Microprocessors and Microsystems*, Elsevier, v. 24, n. 4, p. 199–211, 2000.

28 CARVALHO, E.; MÖLLER, F.; MORAES, F.; CALAZANS, N. Design frameworks and configuration controllers for dynamic and partial reconfiguration. *PPGCC-PUCRS Technical Report Series, TR042*, 2004.

29 PARTIAL Reconfiguration Flow on Zynq using Vivado. 2016. <https://www.xilinx.com/support/university/vivado/vivado-workshops/Vivado-partial-reconfiguration-flow-zynq.html>. Accessed: 2018-03-04.

30 MOORE, G. E. The microprocessor: engine of the technology revolution. *Communications of the ACM*, Association for Computing Machinery, Inc., v. 40, n. 2, p. 112–115, 1997.

31 RILEY, H. N. The von neumann architecture of computer systems. *Computer Science Department, California State Polytechnic University*, 1987.

32 BACKUS, J. *Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*. [S.l.]: ACM, 2007.

33 HARTENSTEIN, R. W.; BECKER, J.; KRESS, R. Costum computing machines vs. hardware/software co-design: From a globalized point of view. In: HARTENSTEIN, R. W.; GLESNER, M. (Ed.). *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. p. 65–76. ISBN 978-3-540-70670-0.

34 CASTELLANO, J. P.; SANCHEZ, D.; CAZORLA, O.; SUAREZ, A. Pipelining-based tradeoffs for hardware/software codesign of multimedia systems. In: *Proceedings 8th Euromicro Workshop on Parallel and Distributed Processing*. [S.l.: s.n.], 2000. p. 383–390.

35 CARCHIOLO, V.; MALGERI, M.; MANGIONI, G. An approach to the synthesis of hw and sw in codesign. In: *Proceedings of 5th International Workshop on Hardware/Software Co Design. Codes/CASHE '97*. [S.l.: s.n.], 1997. p. 173–177. ISSN 1092-6100.

36 DESCHAMPS, J.-P.; BIOUL, G. J.; SUTTER, G. D. *Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems*. [S.l.]: John Wiley & Sons, 2006.

37 ARMAMBA Open Specifications. <http://www.arm.com/products/system-ip/amba/amba-open-specifications.php>. Accessed: 2018-08-16.

38 XILINX. *Partial Reconfiguration Decoupler - PG227*. 2016. <https://www.xilinx.com/support/documentation/ip_documentation/pr_decoupler/v1_0/pg227-pr-decoupler.pdf>. Accessed: 2019-13-05.

39 ZAMACOLA, R.; MARTÍNEZ, A. G.; MORA, J.; OTERO, A.; TORRE, E. de L. Impress: Automated tool for the implementation of highly flexible partial reconfigurable systems with Xilinx Vivado. In: IEEE. *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. [S.l.], 2018. p. 1–8.

40 VIPIN, K.; FAHMY, S. A. Zycap: Efficient partial reconfiguration management on the xilinx zynq. *IEEE Embedded Systems Letters*, IEEE, v. 6, n. 3, p. 41–44, 2014.

41 BARRY, R. Mastering the FreeRTOS real time kernel. *Real Time Engineers Ltd*, 2016.

42 BECKHOFF, C.; KOCH, D.; TORRESEN, J. Go ahead: A partial reconfiguration framework. In: IEEE. *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. [S.l.], 2012. p. 37–44.

43 AKOBENG, A. K. Understanding systematic reviews and meta-analysis. *Archives of disease in childhood*, BMJ Publishing Group Ltd, v. 90, n. 8, p. 845–848, 2005.

44 ELECTRICAL, I. of; ENGINEERS, E. *IEEE Thesaurus*. 2019. <https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/ieee-thesaurus.pdf>. Accessed: 2019-03-03.

45 ELECTRICAL, I. of; ENGINEERS, E. *IEEE Taxonomy*. 2019. <https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/ieee-taxonomy.pdf>. Accessed: 2019-03-03.

46 EAPEN, B. R. et al. Endnote 7.0. *Indian Journal of Dermatology, Venereology, and Leprology*, Medknow Publications, v. 72, n. 2, p. 165, 2006.

47 ZAUGG, H.; WEST, R. E.; TATEISHI, I.; RANDALL, D. L. Mendeley: Creating communities of scholarly inquiry through research collaboration. *TechTrends*, Springer, v. 55, n. 1, p. 32–36, 2011.

48 KOCH, D.; BECKHOFF, C.; TEICH, J. Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs. In: IEEE. *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. [S.l.], 2008. p. 119–124.

49 PNEVMATIKATOS, D.; PAPADIMITRIOU, K.; BECKER, T.; BÖHM, P.; BROKALAKIS, A.; BRUNEEL, K.; CIOBANU, C.; DAVIDSON, T.; GAYDADJIEV, G.; HEYSE, K. et al. Faster: facilitating analysis and synthesis technologies for effective reconfiguration. *Microprocessors and Microsystems*, Elsevier, v. 39, n. 4-5, p. 321–338, 2015.

50 RABOZZI, M.; BRONDOLIN, R.; NATALE, G.; SOZZO, E. D.; HUEBNER, M.; BROKALAKIS, A.; CIOBANU, C.; STROOBANDT, D.; SANTAMBROGIO, M. D. A cad open platform for high performance reconfigurable systems in the extra project. In: IEEE. *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. [S.l.], 2017. p. 368–373.

51 RABOZZI, M. Caos: Cad as an adaptive open-platform service for high performance reconfigurable systems. In: *Special Topics in Information Technology*. [S.l.]: Springer, 2020. p. 103–115.

52 BECKHOFF, C.; KOCH, D.; TORRESON, J. Automatic floorplanning and interface synthesis of island style reconfigurable systems with goahead. In: SPRINGER. *International Conference on Architecture of Computing Systems*. [S.l.], 2013. p. 303–316.

53 BECKHOFF, C.; WOLD, A.; FRITZELL, A.; KOCH, D.; TORRESEN, J. Building partial systems with goahead. In: IEEE. *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. [S.l.], 2013. p. 1–1.

54 SOHANGHPURWALA, A. A.; ATHANAS, P.; FRANGIEH, T.; WOOD, A. Openpr: An open-source partial-reconfiguration toolkit for xilinx FPGAs. In: IEEE. *2011 IEEE International Parallel & Distributed Processing Symposium Workshops and PhD Forum*. [S.l.], 2011. p. 228–235.

55 YOUSUF, S.; GORDON-ROSS, A. Dapr: Design automation for partially reconfigurable FPGAs. In: *ERSA*. [S.l.: s.n.], 2010. p. 97–103.

56 OOMEN, R.; NGUYEN, T.; KUMAR, A.; CORPORAAL, H. An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs. In: IEEE. *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. [S.l.], 2015. p. 1–4.

57 LAVIN, C.; PADILLA, M.; LAMPRECHT, J.; LUNDRIGAN, P.; NELSON, B.; HUTCHINGS, B. Rapidsmith: Do-it-yourself cad tools for xilinx FPGAs. In: IEEE. *2011 21st International Conference on Field Programmable Logic and Applications*. [S.l.], 2011. p. 349–355.

58 CANCARE, F.; SANTAMBROGIO, M. D.; SCIUTO, D. A design flow tailored for self dynamic reconfigurable architecture. In: IEEE. *2008 IEEE International Symposium on Parallel and Distributed Processing*. [S.l.], 2008. p. 1–8.

59 STEINER, N. J. *A standalone wire database for routing and tracing in Xilinx Virtex, Virtex-E, and Virtex-II FPGAs*. Tese (Doutorado) — Virginia Tech, 2002.

60 LAARHOVEN, P. J. V.; AARTS, E. H. Simulated annealing. In: *Simulated annealing: Theory and applications*. [S.l.]: Springer, 1987. p. 7–15.

61 NIAKI, S. H. A.; SANDER, I. Co-simulation of embedded systems in a heterogeneous moc-based modeling framework. In: IEEE. *2011 6th IEEE International Symposium on Industrial and Embedded Systems*. [S.l.], 2011. p. 238–247.

62 MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCÓN, M. FPGA based floating-point library for cordic algorithms. In: IEEE. *2010 VI Southern Programmable Logic Conference (SPL)*. [S.l.], 2010. p. 55–60.

63 MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCÓN, M. Tradeoff of FPGA design of a floating-point library for arithmetic operators. *Journal of Integrated Circuits and Systems*, v. 5, n. 1, p. 42–52, 2010.

64 DUBROVA, E. *Fault-tolerant design*. [S.l.]: Springer, 2013.

65 KENNEDY, J. Particle swarm optimization. In: *Encyclopedia of machine learning*. [S.l.]: Springer, 2011. p. 760–766.

66 STORN, R.; PRICE, K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, Springer, v. 11, n. 4, p. 341–359, 1997.

67 FREERTOS - the small footprint professional grade free RTOS ports and demo application list sorted by microncontroller vendor and microcontroller family. <https://www.freertos.org/a00090.html#XILINX>. Accessed: 2018-06-06.

68 FAZZOLETTO, E. Technical report, *Characterization of partial and run-time reconfigurable FPGAs*. 2016.

69 TARANTINO, S.; CLEMENTE, F.; BARONE, D.; CONTROZZI, M.; CIPRIANI, C. The myokinetic control interface: tracking implanted magnets as a means for prosthetic control. *Scientific reports*, Nature Publishing Group, v. 7, n. 1, p. 17149, 2017.

APPENDIX

RTRLib: A High-Level Modeling Tool for the Implementation of Dynamically Partial Reconfigurable System-on-Chips

Regina Marcela Ivo
Department of Mechanical Engineering
University of Brasilia
Brasília, Brazil
Email: regina.ivo29@gmail.com

Daniel M. Muñoz
Department of Mechanical Engineering and Electronics Engineering, Faculty of Gama
University of Brasilia
Brasília, Brazil
Email: damuz@unb.br

***Abstract*—Partial Reconfiguration allows the time-multiplexing resources to be explored, accomplishing requirements such as adaptability, robustness, power consumption, cost, and also fault tolerance. This paper presents a new partial reconfiguration toolkit called RTRLib that aims to simplify the development of dynamically reconfigurable systems, consequently reducing the development time. The RTRLib is a platform-based high-level modeling tool for run-time reconfigurable systems based on the semi-formal refinement-by-replacement methodology used to simulate and analyze the system behavior during the reconfiguration process. The RTRLib also automatically provides the Vivado and SDK scripts, under some restrictions. Since RTRLib is based on previous characterized IP-cores, it is possible to have an earlier estimation of the reconfiguration time and the reconfigurable partitions resource utilization during the design process. The proposed solution allows the designer to construct the partial reconfiguration solution to be mapped on FPGA-based System-on-Chips by a simple system specification and parametrization.**

***Index Terms*—Partial Reconfiguration; High-level Modeling; System-on-Chip; FPGA**

I. Introduction

Dynamic partial reconfiguration is a powerful technique applied to high-performance embedded systems, allowing requirements such as adaptability, robustness, fault tolerance, low power consumption, and low cost to be achieved, taking into account physical size restrictions [1].

The using of dynamically reconfigurable systems has some advantages, of which it is possible to stands out the increasing of the solution flexibility through the functionality of the time-multiplexing of the configurations. Also, the hardware resources and the consumption of dynamic power are decreased, reflecting on the reduction of the size and costs of the FPGA (Field Programmable Gate Array) [2]. When designing dynamically partial reconfigurable (DPR) systems, it is essential to analyze in advance the following aspects: (a) the impact of the reconfiguration time, since it can increase the total latency of the system, reducing its performance; (b) the area required for each configuration and; (c) the data dependency between reconfigurable regions and; (d) the reconfiguration strategies depends on the adoption of hardware or software triggers and involves the use of specific hardware responsible for controlling the reconfiguration process [3]. These considerations demonstrate the need of a clear vision about the system behavior during the development process.

A DPR system is composed of a static part and one or multiple *Reconfigurable Partitions* (RP), each RP can be loaded with different *Reconfigurable Modules* (RM), where each RM implements a specific functionality of the system. Then, the FPGA resources can be time-multiplexed by the dynamic reconfiguration of the FPGA configuration memory with various hardware functionalities, defined by different design specifications. The Vivado IDE allows the development of DPR systems by generating the bitstream of the static logic as well as partial bitstreams (PBS) related to each RM.

The complexity of developing a DPR system is considerably higher when compared to the one of a static system. From this perspective, it is possible to see the difficulties in designing DPR systems, such as the hardness of exploring the effects of some RMs characteristics, for instance, the impact of the reconfiguration time and latency on the performance and behavior of the overall system.

In this paper, the authors introduce the RTRLib, a new platform-based high-level modeling tool that intends an easy generation of DPR systems, specifically focusing on all the details concerned with the RTRLib *high-level modeling framework*. An example of proof of concept consisting of two RPs (the first one with 3 RMs and the second one with 2 RMs) connected to the ARM processor of a Zynq device was used for validating the proposed tool. The results demonstrate the correctness of the extraction of the design specification as well as the effectiveness of the final implementation, including the partial bitstreams obtained from the created TCL scripts.

The rest of the paper is organized as follows: Next Section presents a general description of the RTRLib. Section III compares the RTRLib with previous works in the literature, Section IV presents the *high-level modeling framework* and, before concluding, Section V presents an example and results.

CAPES. CNPq. CISB. Saab AB.

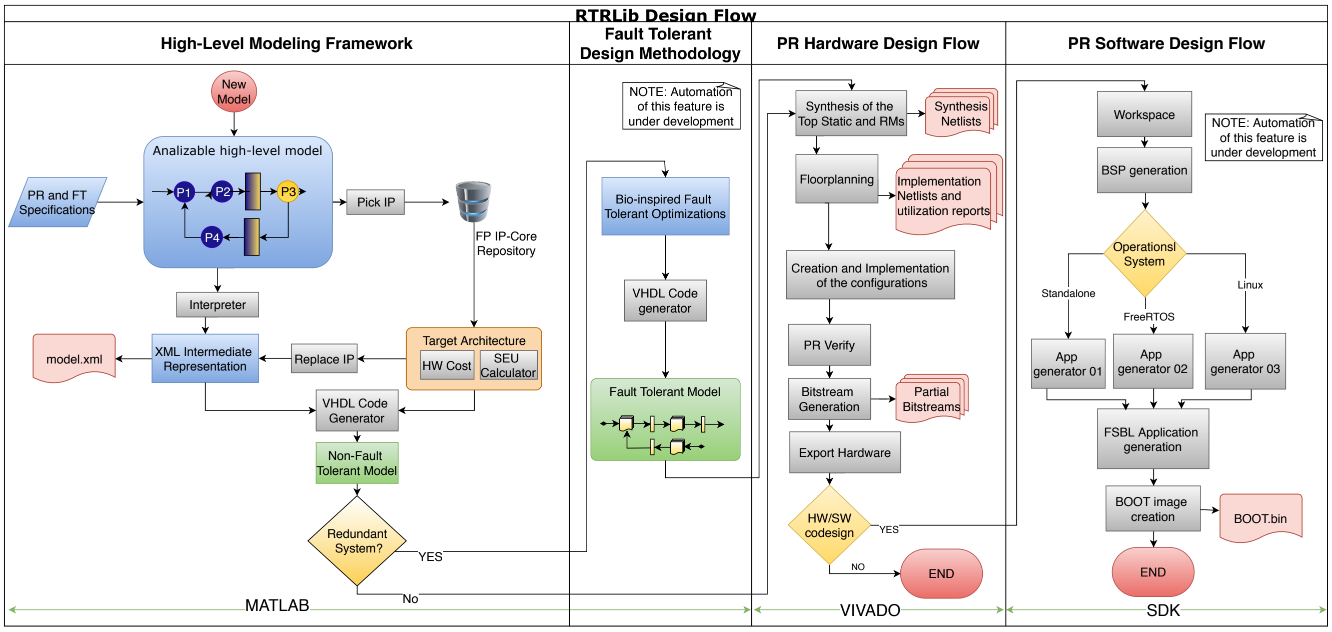


Fig. 1. Overview of the RTRlib proposed design flow. The blue rectangles represent high-level models. The green squares represent the version of the fault-tolerant models. The tasks related to the design transformation and characterization, such as the automatic VHDL code generation, are identified by the gray boxes. The generated files are represented in magenta, while the decision points are shown in yellow.

II. RTRLIB - GENERAL DESCRIPTION

RTRLib is a high-level modeling tool that automatically provides the Vivado and SDK TCL scripts for developing DPR systems on Xilinx Zynq-7000 devices, allowing the estimation of the resource utilization, latency, and reconfiguration time of each RM.

These features are possible because the RTRLib is a platform-based design tool that uses the refinement-by-replacement methodology [4], allowing non-experts users to make use of functional blocks, interconnecting them as a process network to express a specific behavior of the system.

The proposed RTRLib design flow can be divided into four different approaches, mapped in the following lanes: *high-level modeling framework, hardware redundancy design methodology, PR hardware design flow*, and *PR software design flow*. Figure 1 illustrates the overall RTRLib design flow.

The first step of the design flow using the RTRLib is the *high-level modeling framework* description, Section IV describes in detail this part of the system.

In the context of the hardware redundancy design, the RTRLib aims to implement a fault-tolerant design methodology based on the structural design of an NMR solution [5]. The designer provides the mission details, such as the FPGA family and mission time of each module. From this data, the resources estimation, the size in MB of each RP, and the system reliability are estimated and provided. Two bio-inspired methodologies were applied (mono-objective and multi-objective) where the primary goal is to find the optimum redundancy level for each stage, that is, to discover the optimal number of replicas of each module in each step.

Taking the model constructed from the description, for both cases redundant and non-redundant, the RTRLib executes the PR hardware design flow of which the main result is a script with the TCL commands to generate the system desired.

From the point of view of the *PR software design flow*, the Zynq platform enables the utilization of three different embedded OS, the C/C++ standalone, the FreeRTOS823 Xilinx, and C/C++ Linux. At the configuration of the RTRLib ARM block, the user can select from any of them. In the specific case of the C/C++ standalone solution, the RTRLib creates a template that allows the reconfiguration through software triggers sent through the UART. In the case of the FreeRTOS solutions, the RTRLib provides a visual and simple interface to construct personalized applications.

III. State of the Art

A few academic tools have been developed for automating the entire design flow and overcome the difficulties in designing DPR systems. When designing DPR systems, several decisions must be adopted to guarantee the correctness of the implementations. The interface between the static logic and the reconfigurable regions, the routing isolation and decoupling mechanisms [6], [7], the reconfiguration strategy, and the communication between RPs are some of those decisions.

Table I summarizes the features provided by the works in the state-of-the-art compared with RTRLib. Other relevant approaches are presented in [8], [9] [10], [11], and [17].

Koch et al. [12] introduce its I/O bar communication technique and the RecoBus-builder tool to automatically generate

TABLE I
COMPARISION OF THE MAIN PR DESIGN TOOLS

Feature	**Recobus Builder [12]**	**GoAhead [6]**	**RePaBit [13]**	**IMPRESS [7]**	**FASTER [14]**	**CAOS [15]**	**RTRLib**
Interface	Various macros	Direct wire binding	Bus macro	Virtual Interface	Not reported	Not reported	Partition Pins
Decoupling	Yes	Yes	No	Yes	No	No	Yes
PR-PR Communication	No	Yes	No	Yes	No	No	No
GUI	Yes	Yes	No	No	Not reported	Yes	Yes
HDL generation for RMs	No	No	No	No	No	using HLS	Only structural design
High-level analysis	Yes	No	No	No	Yes	Yes	Yes
Fault-Tolerance	No	No	No	No	No	No	Only hardware redundancy
Application generation	No	No	No	No	Yes	Yes	Standalone,Linux,FreeRTOS
IDE	ISE (XDL)	ISE (XDL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)
Supported devices	Virtex II/Pro and Spartan 3	Virtex and Spartan 6	Zynq Soc	Zynq Soc	Virtex 5 and Zynq	Xilinx VU9P	Zynq Soc

the communication infrastructure in PR designs. The GoAhead tool [6] implements reconfigurable interfaces without introducing resource overheads and also provides static/partial decoupling, hierarchical reconfiguration, and reconfigurable crossing region.

The RepaBit tool [13] makes use of bus macro interfaces and adopts the isolation design flow (IDF) for producing relocatable partial bitstreams for Xilinx Zynq devices. The IMPRESS tool [7] develop a custom *virtual interface* based on fixed nodes shared between the static logic and the RPs, supports relocatable bitstreams by avoiding *feed-through* nets, and permits communication between RPs through the use of the AXI interface.

The FASTER tool [14] is an integrated semi-automatic framework that allows the development, from a C-based description, of reconfigurable heterogeneous MPSoC systems. The CAOS platform [15] is designed to encourage the community to adopt, develop, and improve HPRC (High-Performance Reconfigurable Computers) systems. It comprehends the full process of application optimization, from the identification and optimization of the kernel functions to the generation of the runtime management for the target system. For both tools, the authors do not mention in detail all the target devices, however it is mentioned study cases using Virtex 5 and Zynq devices, in [16], and Xilinx VU9P devices, in [14] respectively.

This paper introduces RTRLib focusing on the description of the RTRLib *high-level modeling framework*, as described in the next Section.

IV. HIGH-LEVEL MODELING FRAMEWORK

The RTRLib has been developed in MATLAB and Simulink. Some essential aspects of the RTRLib modeling process are discussed below.

Initially, the user must instantiate and configure the *Top Module Block*, selecting the target board (Zedboard or Zybo) and the reconfiguration strategy between the three options:

1) *PRC (Partial Reconfigurable Controller) + ICAP:* This strategy uses the PRC IP. When hardware trigger events occur, the PRC pulls partial bitstreams from memory and delivers them to the ICAP. The PRC is configured according to the number and names of the RPs, number and names of the RMs, and also the address and size of each RM.
2) *PCAP + ARM:* This strategy allows only software triggers events. In this case, the ARM controls the reconfiguration process through the PCAP.
3) *Manual:* By this strategy, using the Vivado Hardware Manager Tool through the JTAG connection, the user can manually configure each RP by downloading each partial bitstream.

Then, the *Reconfigurable Partitions Blocks* must be inserted inside the *Top Module Block*. Each RP is modeled as a variant system, allowing the high-level model to use global variables for simulating a reconfiguration process by selecting one of the available subsystems. At this step, the designer must define if each RP will be connected through the AXI protocol or not. In the case of the AXI protocol, the tool automatically generates the HDL code of the AXI-Lite communication interface connected to the RP. On the other hand, the RPs that are not connected to the ARM are configured to reset after the reconfiguration process.

In a deeper level of abstraction, the designer inserts the *Reconfigurable Module Blocks* into each RP block, as many RM there are in each RP. At this level, the blocks do not have to be connected. During the simulation step, the connectivity is automatically determined based on the name of the active variant system.

The RTRLib repository is composed of arithmetic operators, FIR filters, and neuron networks, among others. As the system is modeled based on these pre-characterized IP-Cores, it is possible to have an early estimation about some essential features in the DPR design, such as the utilization resources in terms of LUTs, FFs, DSPs, and BRAMs for the target FPGA, the latency in clock cycles, and to estimate the size of each RP and its reconfiguration time. In the case of hardware redundancy designs, failure rates, and reliability of each RM estimations according to measurements, according to the target device, are also provided. This description allows the users to do preliminary tests and simulations with their architecture even in the high-level phase of the design, reducing the time of extensive behavioral simulation, taking in mind that a lot of errors can be detected in this design step.

In the case of SoC designs, the *ARM_Core block* and the *AXI-Interconnect Blocks* must be included. The parameters configurations for the *ARM_Core block* are based on the usage of the SD-Card, the UART, the XADC, and the DDR3 Controller as well as on the definition of the application type that will be embedded in the ARM Core (standalone, FreeRTOS or Linux Application).

The interpreter creates an intermediate representation in the XML format that contains the description of the tree diagram of the system, which includes information on the quantity of RPs, number of RMs in each RP, configuration parameters of each block, and connections.

The floorplanning is done through the GUI. The pblocks, where each RP will be placed, must be manually defined, taking into account the resource requirements of each RM. Several DRC rules must be accomplished to ensure the effectiveness of the placement and routing. RTRLib is equipped with some error and warning messages that early prevent the user from making the most common mistakes.

After high-level modeling, RTRLib makes use of the Vivado PR design flow. It executes from the creation of the folder structure and the repository with the available IP-Cores to the hardware exportation. At the end, the script verifies if the system is an HW/SW codesign. In the case that the project is a hardware-only design, it finishes the flow. However, if the project is an HW/SW codesign, the script launches the SDK and initiates the *PR software design flow*.

V. PROOF OF CONCEPT USING RTRLIB

As a proof of concept of the proposed tool, a dynamically reconfigurable architecture with 2 RPs is provided as an example of the utilization of the RTRLib. For the first RP, there are 3 RMs that implement the arithmetic calculation of addition, division, and multiplication by constant. For the second RP, there are 2 RMs that implement the sine and the computation of the exponential function of a given value, where all the five operations are done using the floating-point IP-Cores. For this example, it was selected a non-fault-tolerant system, the strategy of reconfiguration using the PRC, and both RPs connected to the ARM through the AXI-Lite Interconnect.

The ARM processor sends the RPs input data through the AXI-Lite. In the opposite direction, the ARM reads the result of the operations through the AXI. The reconfiguration is done through the ICAP by hardware triggers mapped in the switches of the board and controlled by the PRC. Figure 2 shows the high-level modeling of the system, with the connections between the RPs and the ARM processor. Table II shows the utilization report of all the reconfigurable modules.

TABLE II
RMs UTILIZATION REPORT FOR THE ZEDBOARD (ZYNQ 7020). THE RP_ODD CONTAINS 188 SLICES AND 4 DSPs, WHILE THE RP_EVEN CONTAINS 438 SLICES, 3 BRAMs AND 6 DSPs.

Name RM	Function	LUTs		FFs		DSPs	
		Available:	53200	Available:	106400	Available:	220
RM_t1	ADD	334	0,63%	275	0,26%	0	0%
RM_t2	EXP	1785	3,36%	596	0,56%	1	0.45%
RM_t3	DIV	278	0,52%	320	0,30%	1	0.45%
RM_t4	SINE	1419	2,67%	557	0,52%	1	0.45%
RM_t5	MUL	135	0,25%	255	0,24%	1	0.45%

Figure 3 (A) shows the block design created through the script generated using RTRLib, which has the IPs related to the RPs, the ARM-Core processor IP, PRC, and the AXI Interconnect.

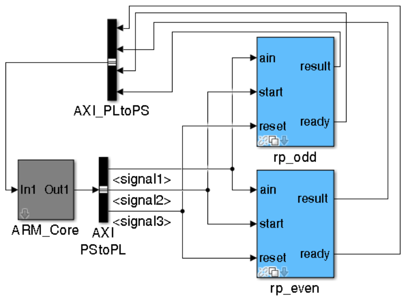


Fig. 2. Detail of the system-level model using the RTRLib, showing the RPs, AXI, and the connections. The data input of the RPs is sent to the ARM processor through the UART port (not depicted in the figure).

The size in kB of each partial bitstream was obtained and using the model reported by [18], which also uses Zedboard and states that the relationship between the reconfiguration time and the size of the partial bitstream is essentially linear, the reconfiguration time was estimated. Those results are presented in Table III. Figure 3(B) shows the circuit layout containing the floating-point adder in the first RP and the floating-point sine estimator in the second RP.

TABLE III
COMPARISION BETWEEN THE SIZE OF THE PARTIAL BITSTREAMS AND THE RECONFIGURATION TIME

Partial bitstream	Size (KB)	Estimated Reconfiguration Time (ms)
Blanking	3951	30,422
RP odd	223	1,717
RP even	484	3,727

VI. CONCLUSIONS AND FUTURE WORK

This paper introduced the RTRLib, a new platform-based high-level tool that was projected to automate the partial reconfiguration design flow steps. The main contributions of the RTRLib are: (a) DPR system design is more accessible to non-expert designers; (b) the RTRLib allows for an early estimation of the hardware resources and size of each reconfigurable module, thus enabling an analysis of the effect of the latency and reconfiguration time of each RP over the system; (c) The proposed tool also provides a design methodology for fault-tolerant systems based on the hardware redundancy approach. All these features allow the prototyping time reducing and enable fast creation of high-performance systems.

A simple proof of concept was modeled in RTRLib, in which two reconfigurable partitions implement floating-point arithmetic and trigonometric operations. The obtained scripts were effectively executed in Vivado, and the circuits were characterized in terms of resource consumption and size (in KB) of the RPs and reconfiguration time.

As future works, it is crucial to investigate strategies to enhance the RTRLib tool floorplanning algorithm to enable

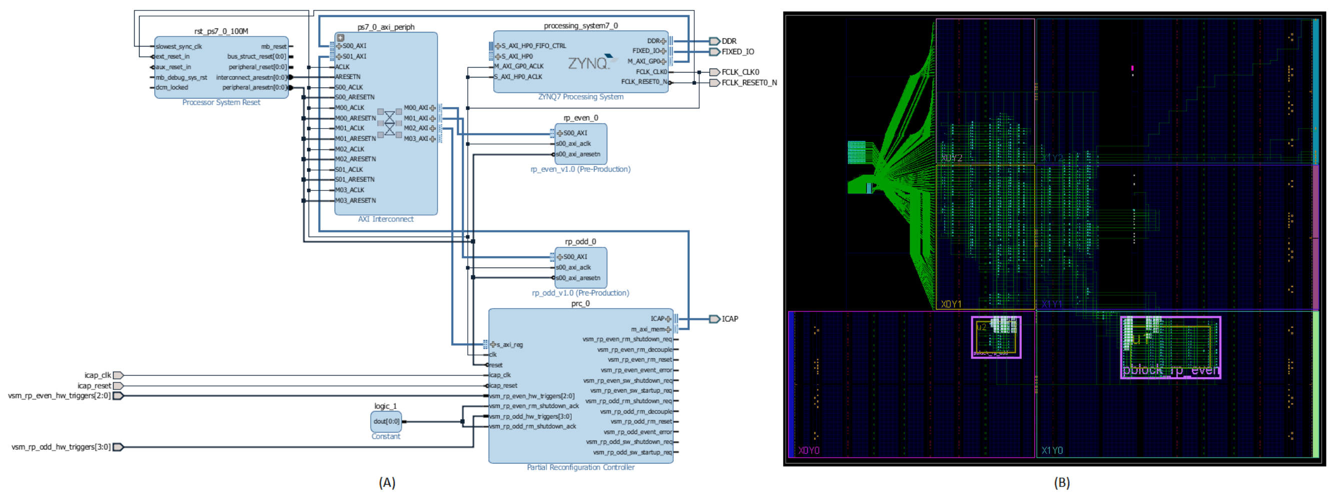


Fig. 3. (A) Block Based Architecture with two RPs, PRC, AXI switch and Zynq7 Processing System (B) Final implementation of the reconfigurable architecture, including the static system and the reconfigurable partitions, with the reconfigurable modules that compute the adder and the calculus of the sine function, respectively. Implemented on a Zynq-7020 device.

slot and grid styles, analyze data dependency between RPs, integrating models for reconfiguration time and power consumption estimation, and implementing optimization algorithms to minimize the reconfigurable resources into the RPs.

Acknowledgment

Authors thank to Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 and the financial support of CNPq, National Council of Scientific and Technological Development of Brazil, CISB, Swedish-Brazilian Research and Innovation Centre, and of Saab AB.

References

[1] M. Wolf, *High-performance embedded computing: applications in cyber-physical systems and mobile computing*. Newnes, 2014.

[2] X. Zhang and K. W. Ng, "A review of high-level synthesis for dynamically reconfigurable FPGAs," *Microprocessors and Microsystems*, vol. 24, no. 4, pp. 199–211, 2000.

[3] J. Zhu, I. Sander, and A. Jantsch, "Performance analysis of reconfigurations in adaptive real-time streaming applications," *ACM Transactions on Embedded Computing Systems*, vol. 11S, no. 1, pp. 12:1–12:20, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2180887.2180888

[4] S. H. A. Niaki and I. Sander, "Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework," in *IEEE Int. Symposium on Industrial and Embedded Systems*, June 2011, pp. 238–247.

[5] E. Dubrova, *Fault-tolerant design*. Springer, 2013.

[6] C. Beckhoff, D. Koch, and J. Torresen, "Go ahead: A partial reconfiguration framework," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 37–44.

[7] R. Zamacola, A. G. Martínez, J. Mora, A. Otero, and E. de La Torre, "Impress: Automated tool for the implementation of highly flexible partial reconfigurable systems with Xilinx Vivado," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2018, pp. 1–8.

[8] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, "Openpr: An open-source partial-reconfiguration toolkit for xilinx fpgas," in *2011 IEEE International Parallel & Distributed Processing Symposium Workshops and PhD Forum*. IEEE, 2011, pp. 228–235.

[9] R. Oomen, T. Nguyen, A. Kumar, and H. Corporaal, "An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–4.

[10] K. Bruneel and D. Stroobandt, "Automatic generation of run-time parameterizable configurations," in *2008 International Conference on Field Programmable Logic and Applications*. IEEE, 2008, pp. 361–366.

[11] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapidsmith: Do-it-yourself cad tools for xilinx FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 349–355.

[12] D. Koch, C. Beckhoff, and J. Teich, "Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. IEEE, 2008, pp. 119–124.

[13] J. Rettkowski, K. Friesen, and D. Göhringer, "Repabit: Automated generation of relocatable partial bitstreams for Xilinx Zynq fpgas," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2016, pp. 1–8.

[14] D. Pnevmatikatos, K. Papadimitriou, T. Becker, P. Böhm, A. Brokalakis, K. Bruneel, C. Ciobanu, T. Davidson, G. Gaydadjiev, K. Heyse *et al.*, "Faster: facilitating analysis and synthesis technologies for effective reconfiguration," *Microprocessors and Microsystems*, vol. 39, no. 4-5, pp. 321–338, 2015.

[15] M. Rabozzi, R. Brondolin, G. Natale, E. Del Sozzo, M. Huebner, A. Brokalakis, C. Ciobanu, D. Stroobandt, and M. D. Santambrogio, "A cad open platform for high performance reconfigurable systems in the extra project," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 368–373.

[16] M. Rabozzi, "Caos: Cad as an adaptive open-platform service for high performance reconfigurable systems," in *Special Topics in Information Technology*. Springer, 2020, pp. 103–115.

[17] F. Cancare, M. D. Santambrogio, and D. Sciuto, "A design flow tailored for self dynamic reconfigurable architecture," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[18] E. Fazzoletto, "Characterization of partial and run-time reconfigurable FPGAs." KTH Royal Institute of Technology, 2016, Technical report.